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 28, 2012

Clients:
David Auber
Romain Bourqui

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

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 amounts of data generally comes visual clutter, in our case due to edge crossing. This solution is based on an edge bundling technique coupled with a grid built from the original graph. The authors also used a GPU-based rendering method to highlight bundle densities without losing edge color. Our solution uses the Tutte algorithm in order to quickly obtain a graph without crossing, based on a triangular-face grid, which helps to read relationships 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 leads to 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 edge 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.

Our objective all along this project is to read what has been done before relating to this problem, to provide an objective point of view on those previous works, and propose our contribution. We have implemented a method, then optimize its performances thanks to current technologies and sets our boundaries.

The first part of this document presents review-related work on reducing edge clutters and enhancing edge bundle 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 graphs do not belong to these classes, and algorithms 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, the force-directed method produces pleasant and structurally significant results but does not help user comprehension due to data complexity. The authors of the paper specify two techniques for that reduction: compound visualization and edge bundling. But their interest goes to Edge Bundling, which suggests 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 are formed.

Up to now, several techniques have been 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.The Edge bundling technique routes edges into bundles. This uncovers high level edge patterns and emphasizes relationships. Yet an important constraint is the impossibility for some nodes to move while avoiding edges crossing because node positions provide information: consequently, compound visualization is not suitable.

Some existing representations take into account this duty: i.e. some reducing edge clutters (Edge routing, Interactive techniques, Confluent Drawing, Node clustering, Edge clustering) and other enhancing edge bundles visualizations (Smoothing curves, Coloring edges). Edge routing uses shortest-path edge routing to bound the number of edge crossings and use non-point-size to avoid node-edge overlaps. This does 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 routes edge along the hierarchy tree branches. Both methods cannot be applied to any graph. Edge Clustering routes edges either on the outer face of the circle or in its inner faces and bundles them to optimize area utilization.

The publication we are currently working on is based on a new edge bundling algorithm for efficient graph drawing. By using specific dicretization methods such as quad-tree and Voronoï diagrams which are cheap in computing time, the authors obtained a new separation of the region where they can draw a graph. As a result, their final discretization algorithm deals with both *quad tree* and *Voronoï* diagrams for the grid precision(*quad tree*) and the computing time (*Voronoï*).

By using their own specific "shortest path" Dijskstra algorithm, which suggests that in order to obtain a decent number of bundles (rather than the small amount of bundles created by the classic Dijskstra algorithm), they can add new concepts such as *roads* and *Highways*. This means to reduce the weight of an edge (of the grid obtained before) if it is highly used in "classical" Dijskstra, but only after computing several shortest paths between linked nodes of the original graph. The returned value of this specific algorithm is the shortest path computed on the original graph.

Several optimizations of the specific shorstest path algorithm such as function calls reductions, multithreading reduces tremendously computing time of graphs and then leads to different levels of reduction.

the rendering of all previous specific algorithms using efficiently the architecture of the computer permits them to obtain graphs well cluttered but less appealing due to bends. Polylines and Bezier curves are quite interesting because they reduce the “zigzag” effect which appears in edge weighting. Else, Edge-splatting computes through GPU (OpenGL) is also a good technique to render smooth curves to the clutter reduction. However, the graphs produced by these Bezier curves are very poor in information and very expensive in computing: due to edge superposition over nodes, information is lost when edge weights are adjusted. Moreover, the graph is not as “aesthetic” as it should be with edge splatting.

That is why we propose in our work the Tutte algorithm based on a triangular-face convex graph, which implementation is quite easier, and renders more informative and aesthetic graphs

CHAPTER 2

TUTTE

2.1 Tutte's algorithm

The basic graph theory terminology defined in the article [3, 4] will be used. Let $G = (V, E)$ be a planar graph. A mapping Γ of G into the plane is a function $\Gamma : V \cup E \to P(\mathbb{R}^2)$. This function 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.

A way to build embeddings of any planar, 3-connected graph $G = (V, E)$ have been produced by Tutte in 1963 [5]. Let C be a cycle of vertices. Those 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.

Theroem 1 *(Tutte's Theorem) Let Ve a set 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. If for 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), so Γ is an embedding of G into the plane, with strictly convex interior faces.*

2.2 Tutte Sequential

To obtain the graph resulting from the Tutte theorem, an algorithm is needed. In this section, a sequential algorithm is described. This algorithm is an iterative solution. To compute a solution, a set of nodes making up a convex polygon must be decided. Let P be this set of nodes. Let G_k be the graph generated at the step k.

To obtain the graph of the next step, all the nodes of the interior of the convex polygon P will be visited. For each node visited, the barycentric coordinates of its neighbourhood are computed. Then these coordinates are used to update the position of the current node. Once each node has been visited, the computation of the graph G_{k+1} is completed.
In this project, the stop condition used for this algorithm is an epsilon between the relative positions of the node of the graph G_k and G_{k+1}. For all nodes of the graph, if the length of each movement is inferior to a given epsilon, the graph G_{k+1} is the solution.

The figure 2.1 gives an exemple of the mouvement of one node during one step.

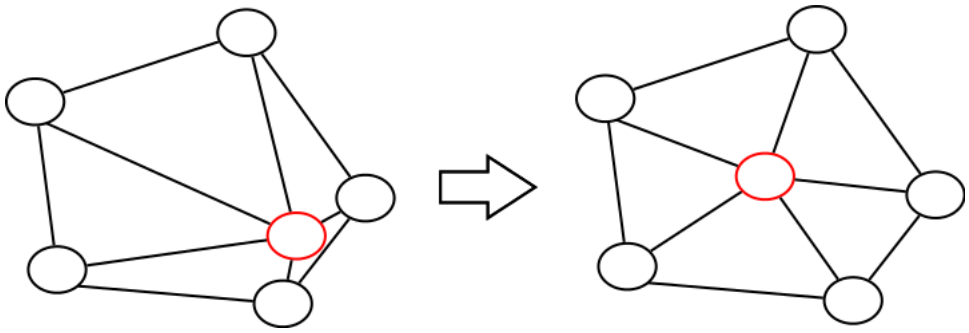

Figure 2.1: barycenter computation of the red node

This gives a synthetic view of this algorithm :

```
G = {V,E}
P = set of nodes constitute a convex polygon
procedure tutte(G, P, epsilon)
  epsilon_current = 0
  for each node of (V \ convex polygon)
    barycenter = barycenter of node
    epsilon_current = max(epsilon_current, distance(node, barycenter)
    node = barycenter
  if (epsilon_current < epsilon)
    exit()
  tutte(G, P, epsilon)
```

From the complexity point of view, during one iteration all the nodes of the set V - *convex polygon* are visited. For each node, all its neighbourhood is visited. Let $aver\_d$ be the average degre of this planar graph. So the compexity of an iteration is $O(n \times aver\_d)$ with $n = |V|$.

2.3 Tutte Parallel Algorithms

In sequential version, the algorithm presented is an asynchronous Tutte. In fact, there is a second approach, it is the synchronous Tutte. The difference between the two approach is :

- in asynchronous version: each movement is applied directy after being computed
- in synchronous version: all movement are computed before being applied

In sequential version, the synchronous approach don't present benefits. It is more convenient to consider neighbors movement. In parallel version, the synchronous approach may be interesting in order to reduce critical sections.

2.3.1 Asynchronous parallel version

Distribution of nodes: Graph coloring

In order to implement a parallel asynchronous version of the Tutte algorithm, it is necessary to separate graph nodes into different sets. The objective is to extract an independence between nodes. In fact, each node has to move while the neighbours maintain their positions. Thus, the independence must be between the moving node and its neighbours. This problem is similar to the famous problem of graph coloring.

The objective of the modified Tutte algorithm is to handle graphs of thousands of nodes. To separate such a number of nodes, it is more effective to use a heuristic of the algorithm of graph coloring.
The greedy algorithm is a simple and good solution to separate nodes into sets fast and effectively.

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
```

This algorithm is known to use at most $d(G) + 1$ colors where $d(G)$ represents the largest value of the degree in the graph G. However, its shortcommings is that it produce sets of different size. This can be inconvenient for task distribution.

Applying Tutte algorithm to sets

Once a set of nodes is obtained, 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 set per thread : This distribution is far from being optimal. In fact, each thread has to lock the

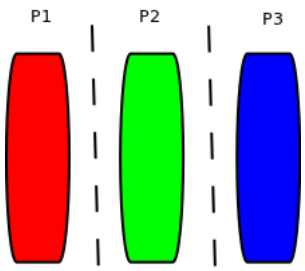

Figure 2.2: One set per thread

neighbours nodes before moving the concerned node, which introduces an important critical section. In addition to being unfair, this distribution is limited by the number of sets produced.

The best distribution for sets obtained by graph coloring is to execute n threads on one set. Each thread moves a number of nodes of the set without any critical section, since each node of the set is not the neighbour of all the other nodes of the same set. Once the thread has moved all its nodes of the set, it must wait for other threads to have completed the same process (implemented by a barrier). Then, the overall process is applied to the next set.

2.3.2 Synchronous parallel version

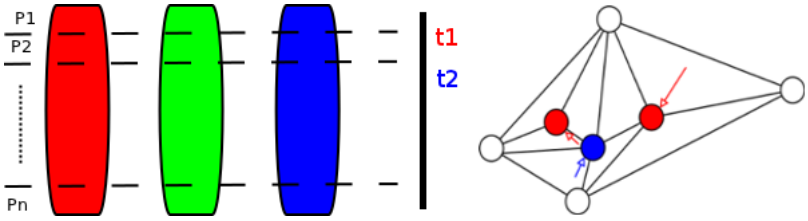

Figure 2.3: n threads per set

CHAPTER 3

IMPLEMENTATIONS

3.1 Data Structure

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

3.1.1 Issues

As a Tulip Data Structure contains a lot of information, it is expensive to manipulate them. Furthermore, we do not need all the information from a given Tulip graph. Especially, we do not need all the properties about nodes to conduct the Tutte algorithm. For instance, for a given node we just want to know if it is fixed, for a fixed node, position never changes during the Tutte algorithm. In addition to that, as we are looking for performance, we need a light structure matching the principe of Tutte algorithm. The following points are the main reasons which lead us to set up a new data structure.

1. The fact that a given node is fixed or not is indicated firstly by a mobility property. However, there is another property indicating nodes which are part of graph contouring, and these nodes need to be fixed too. Therefore, 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 data structure there is a hierarchy of graphs. However, 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 Implementations

We tested three implementations in order to find out the right one. Because we care of memory leak, we merely store only the information needed to run the algorithm in our structure.

First implementation

In this implementation our structure is such that a given node contains its neighbourhood. So one can easily access the neighbourhood of a given node for it is very crucial in a Tutte algorithm implementation. To do this we define a class that contains the various data needed on a given node (the attibutes) 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;
};
```

Vertex attributes needed

n : this attribute is of `node` type of Tulip library and contains the `ID` of the node.

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

coord : this attribut is of `Coord` type of tulip library and is used to store the node coordinates.

voisin : this attribute is of `vector` type of `C++` library and contains the neighbourhood.

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 corresponding setter and getter 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()`.

Second implementation

Third implementation

In this third implementation we do not use a class to store the different informations about node to run Tutte algorithm. As we are looking for a more light data structure in order to ameliorate memory access, we use a struct to group datas needed about a given node under one name (Data).

```
  struct Data {
    node n;
    Coord coord;
    bool mobile;
  } Data;
```

In addition of the structure above, we use two tables. A `data store table` table to store datas about nodes and `neighbourhoods table` to link nodes with their neighbourhoods. The picture below illustrate the principle.

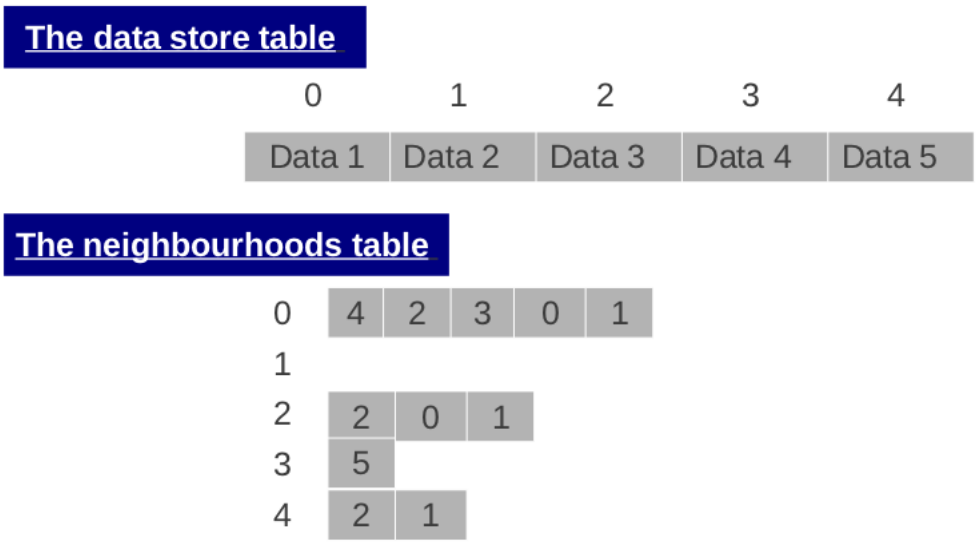

Figure 3.1: Third implementation graph representation

One can read in the `neighbourhoods table` that neighbours of the node 0 are nodes `4, 2, 3, 0, 1` and node 1 have not got neighbours. One can access all the informations about node 0 located at the index 0 of the `data store table`.

3.2 Results

The authors provided us with three graphs haven been given in order to test our different implementations of the Tutte method.

These graphs have the following characteristics :

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 work on the graph,we obtain these results for performance aspect :

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

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 existing methods.

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

Another valuable improvement would be to turn our standalone program into a Tulip plugin in order to integrate it in this software.

A theoretical point remains unsolved : despite the fact that the contour on which we apply the Tutte algorithm is not necessarily convex, we obtain correct results. It would be useful to understand why Tutte works on such kind of graphs: It could be caused by how the grid is built or by the following constraint: two fixed nodes can not be linked by an edge.

BIBLIOGRAPHY

[1] 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.

[2] 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.

[3] 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.

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

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

[6] Gormen, T.H. and Leiserson, C.E. and Rivest, R.L. and Stein, C. Introduction to algorithms. *In MIT press Cambridge, MA*, 16:"Greedy Algorithms", 1990.

(refaire biblio en se centrant plus sur tutte)