CS637 - Pellegrino

CS637: Interactive Computer Graphics
Drexel University
January 27, 2009
Don Pellegrino
Graduate Student Research Paper Presentation


All quotations are from the article unless otherwise specified.
"This paper presents a new algorithm for force directed graph layout on the GPU. The algorithm, whose goal is to compute layouts accurately and quickly, has two contributions. The first contribution is proposing a general multi-level scheme, which is based on spectral partitioning. The second contribution is computing the layout on the GPU. Since the GPU requires a data parallel programming model, the challenge is devising a mapping of a naturally unstructured graph into a well-partitioned structured one. This is done by computing a balanced partitioning of a general graph. This algorithm provides a general multi-level scheme, which has the potential to be used not only for computation on the GPU, but also on emerging multi-core architectures. The algorithm manages to compute high quality layouts of large graphs in a fraction of the time required by existing algorithms of similar quality. An application for visualizations of the topologies of ISP (Internet Service Provider) networks is presented."
Relevance

Visual navigation of large graphs contextualizes a knowledge space to help provide insight to users.

APIs provide object-oriented interface to Graphs:
* Prefuse (http://www.prefuse.org) - Java, 2D
* VTK (http://www.vtk.org) - OpenGL, C++, 2/3D

OpenGL provides the most direct access and highest performance access to stream processing of GPU. Custom APIs from ATI and NVIDIA in C.
Problem: Graph Layout

Battista is the standard citation:

Common force-directed layout algorithms:
Fruchterman-Reingold
Kamada-Kawai

Problem: “… the force directed algorithm, is computational expensive – $O(V^2 + E)$ - (p1310).”
Solution: GPU Processing

“GPUs are geared towards repetitively performing the same computation on large streams of data. Therefore, the GPU suits uniformly structured data, such as images or matrices. Graphs do not possess a uniform structure, hence they do not admit any intuitive and natural representation that suits computation on the GPU (p1310).”

Recall Lecture 1 – shift from separate dedicated graphics unit to CPU and now to local GPU.

GPU = Stream Processing
Fruchterman Reingold with Kamada-Kawai

“Another algorithmic contribution of the paper is devising a layout algorithm that combines the strengths of two different well-known layout algorithms (p1310).”

Well... sort of.

“We advise starting with Fruchterman Reingold and using several optimal distances until a stable result appears. The drawing can then be improved using Kamada-Kawai with the actual location of vertices as starting positions. Finally, improve the drawing by manual editing, which we discuss under Section 1.3.4.2 [de Nooy, Mrvar and Batagelj, “Exploratory Social Network Analysis with Pajek,” Cambridge University Press, New York, NY, 2005, p17].”
1. Initial coarsening: compute $G_1, G_2, \ldots, G_{\text{coarsest}}$ where $G_{k+1} = \text{edge collapse}(G_k)$.

2. Partitioning initialization: set $P_{\text{level}}=0$ part num=0 to $G_{\text{coarsest}}$. Set $l =0$.

3. Partitioning: try to partition each graph $P_{\text{ln}}$. This creates a new set of graphs $P_{l+10}, P_{l+11}, \ldots$. If no graph $P_{\text{ln}}$ could be partitioned, goto step 7.

4. Multi-level construction: construct $L_l$ out of $G_{\text{coarsest}}$, where each node in $L_l$ corresponds to a graph $P_{\text{ln}}$.

5. Layout initialization: compute an initial layout for $L_l$, using interpolated initial positions from the coarser $L_{l-1}$.

6. Layout: compute the layout for $L_l$. This is the core step of the algorithm, which uses our variant of the force-directed approach. Set $l = l+1$, goto step 3.

7. Compute a layout for $G_{\text{coarsest}}$ using interpolated initial positions from $L_{\text{f inest}}$, the finest graph layout computed in stage 6.

8. Final un-coarsening: Compute layouts for $G_{\text{coarsest}−1}, G_{\text{coarsest}−2}, \ldots, G_0$ by repetitively interpolating from $G_i$ to $G_{i−1}$ and laying out $G_{i−1}$. 

1. Initial Coarsening

Pre-processing stage.

Data reduction technique.

“At each level $k$, given a fine graph $G^k$, a coarser representation $G^{k+1}$ is constructed using a series of edge collapse operations (p1312).”

“... three initial coarsening steps are performed (p1312).”
“This step initializes the variables using in the recursive partitioning of graph $G^{coarsest}$ in the next step. The graph $P_0$, which is set to $G^{coarsest}$, is created (p1312).”
6. Layout

“This is the core step of the algorithm, which uses our variant of the force-directed approach (p1312).”
“On the GPU, input and output are represented as two-dimensional arrays of data, called textures. The challenge is to map the graph and its elements onto textures, even though graphs do not admit any intuitive and natural representation as balanced arrays (p1313).”
Data Storage (Textures) – Graph Layout

- **Graph Layout**
  - **Location Texture**: (x,y) positions of all the nodes in the graph. - Note 2D. Also holds the partition number of each node.
  - **Neighbors Texture**: For each node a pointer into the adjacency texture to the coordinates of the first neighbor of the node. Also holds degree of each node.
  - **Adjacency Texture**: List of pointers into the location texture.
Textures – Graph Layout

Fig. 5. Representing graph edges on the GPU. Node X has three neighbors: Y, Z and W.
Data Storage (Textures) - Partitions

- Geometric partitions
  - Partition Information Texture
    - Constant information.
    - Coordinates in the location texture of the upper left corner of the partition.
    - Width and height of the partition rectangle.
    - Number of nodes in the last row of the partition.
    - Number of nodes in the partition.
  - Partition Center of Gravity Texture
    - Modified during the layout computation.
    - Current (x,y) coordinates of the center of gravity of each partition.
Data Storage (Textures) - Forces

- Forces
  - Attractive Force Texture
    - For each node the sum of the attractive forces exerted on it by its neighbors.
  - Repulsive Force Texture
    - The sum of repulsive forces.
Stream Processing

Rectangle = Stream
Oval = Kernel

Diagram:
- **partition info**
- **location**
- **neighbors**

- **partition C.G.**
- **repulse**
  - **repulsive forces**
  - **anneal**
  - **updated location**

- **attract**
  - **attractive forces**
  - **anneal**
  - **updated location**

- **adjacency**

Feedback loop:
- **partition info** → **location** → **neighbors** → **partition C.G.** → **repulse** → **anneal** → **updated location** → **feedback**

- **location** → **neighbors** → **adjacency** → **partition C.G.** → **repulse** → **anneal** → **updated location** → **feedback**
Kernels

- **Partition**
  - Calculate center of gravity for each partition.

- **Forces**
  - Repulse
  - Attract

- **Anneal**
  - “It reads the attractive force, repulsive force, and location textures and updates a second copy of the location texture (p 1314).”
  - Uses estimates to iterate toward a global optimization function – close but non-optimum solution.
The Graph Partitioning Archive

http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/
GreenMax from PNNL


“We present an information visualization tool, known as GreenMax, to visually explore large small-world graphs with up to a million graph nodes on a desktop computer. A major motivation for scanning a small-world graph in such a dynamic fashion is the demanding goal of identifying not just the well-known features but also the unknown-known and unknown-unknown features of the graph. GreenMax uses a highly effective multilevel graph drawing approach to pre-process a large graph by generating a hierarchy of increasingly coarse layouts that later support the dynamic zooming of the graph. This paper describes the graph visualization challenges, elaborates our solution, and evaluates the contributions of GreenMax in the larger context of visual analytics on large small-world graphs. We report the results of two case studies using GreenMax and the results support our claim that we can use GreenMax to locate unexpected features or structures behind a graph.”