ScienFIST.org

INTERNATIONAL JOURNAL OF INFORMATION TECHNOLOGIES, ENGINEERING AND MANAGEMENT SCIENCE

3D Graph Visualization Performance using Physical Engine

Alexander Brezáni [*], Michal Zábovský [**]
[*] Faculty of Management Science and Informatics, University of Zilina, Slovak Republic
[**] University Science Park, University of Zilina, Slovak Republic
alexander.brezani@uniza.sk, michal.zabovsky@uniza.sk

Abstract

The visualization of complex data is one of the most challenging tasks of exploratory data analysis. Linked data are usually visualized as complex graphs or networks. Due to the complexity of data, 2D projection based only on x and y coordinates is complicated and does not allow natural interaction mechanism for additional data examination. 3D visualization uses additional z dimension but requires some level of interaction and camera movement to reveal occluded data. In this paper we are presenting 3D visualization approach based on physical engine provided by Unity game engine to respect requirements for interaction, camera movement, and use of physical forces for data spatialization. We will show advantages of this approach together with limitations based on game engine adaptation for 3D graph visualization.

Keywords: graph visualization, networks, 3D visualization, data structures, Unity game engine

Introduction

Graph is an abstract representation of relations among a given set of data entities. In mathematics, graph theory is the study of graphs, which are mathematical structures used to pairwise relations between objects. [1] Formally, relations are called edges and data entities vertices.

Since graphs are usually represented visually by drawing, vertices and edges have their specific visual representation defined by points/circles and lines. A graph drawing is the special representation of graph itself which can significantly improve overall understanding of the problem represented by underlying graph structure.

Graph visualization is very useful for better data structure understanding and supports knowledge discovery. There are tools for even general graph visualization or graph visualization tools working with relations specific to knowledge domain e.g. molecular biology. One of the most generally used tools is Gephi [2] which implements variety of graph visualization algorithms, features for real-time visualization, graph metrics, and other data oriented features.

Most of the methods for graph visualization are based on 2D algorithms allowing to place vertices and edges into the two-dimensional plane (node-link layout) with respect to overall graph structure readability. Usually, the orthogonal projection is used to render the visualization. Due to the size and complexity of data represented by graph (usually referred as network), additional dimension representing depth is added in 2.5D approach. Then nodes are at different depths in that they have an individual depth value for position. [3]

3D graph visualizations are rendered with a three-dimensional projection and usually require some level of camera movement to reveal occluded data. [4] Moreover, some advanced algorithms for 3D visualizations use force-directed approach to generate layout for overall graph spatialization. [5]

Requirements for camera movement, interaction with user and physically based spatialization defines basic framework for 3D graph visualization. All the mentioned functionalities are provided by game engines, together with advanced physical and interaction model, collision detection and real-time user interaction with advanced camera-based operations.

In presented paper we will show implementation of 3D graph visualization in the Unity game engine [6] with the respect to defined requirements. We will discuss main limitations of this approach and show experimental evaluation of basic performance metrics.

The paper is structured as follows, *Methodology and tools* explains Unity game engine environment and physical engine features used for 3D graph visualization, *Evaluation criteria* define performance criteria for successful graph visualization, *Graph definition in 3D* and *Force-directed data spatialization* sections describe graph creation inside the game engine, *Experimental evaluation* and *Results* present our findings and experimental results, and finally *Conclusion* concludes the article with additional work summary.

Methodology and tools

Game engines provide physical framework with implemented physical model, collision detection, physical interactions, and scene user interaction by using cameras, lights, special effects, and audio. We choose Unity engine for 3D graph visualization with respect to future implantation in existing immersive CAVE system which is based on Unity.

Unity implements physical engine optimized for 3D environments. The use of physical engine for force-directed graphs provides time efficient workflow for graph definition together with 3D layout definition. Physical engine embedded in Unity and other game engines is designed to handle massive numbers of everchanging objects affected by environment's physical forces. However, special importance must be given to the scene design to not overpopulate it with unnecessary objects. Overpopulated scene, even with optimized physical engine, can cause unstable application's behavior and increase the time required to compute stable state of graph.

To define performance boundaries for visualization in Unity, we defined and performed series of tests on various scenarios to find optimal number of elements in virtual scene. In each scenario, graph of network was visualized in 3D space with various size, number of nodes, number of edges and spatial limitations (space used for graph visualization).

Evaluation criteria

Overall rendering performance is one of the main criteria in interactive 3D visualization. Underlaying performance of selected framework or game engine such as Unity or Unreal Engine will directly affect the time required for completing visualization, which can result in positive or negative user experience. Visualization in 3D space requires certain level of user interaction, otherwise it is not much different from common 2D projection. When interaction with model happens, framework or engine should update visualized environment which comes with certain cost in form of CPU, GPU, and RAM resources. Update speed after interaction is influenced by these resources, as well as engine's ability to utilize them, therefore choosing the right engine or framework is crucial for ensuring fast and responsive application.

Graph definition in 3D

Visualization of graphs in 3D space can be difficult for people with decent amount of knowledge about graph layout and 3D space and even harder for beginners, which do not know basics of 3D layout. To minimize user's requirements for skills to create readable layout we use force-directed algorithms which allows user to create 2D or 3D layouts without prior knowledge and experience. Force-directed graphs apply physical properties, like gravity, weight, speed, and many others, to nodes and edges. Resulted layout is determined by strength of applied forces and static properties of nodes and edges.

However, nodes and edges in force-directed graphs are constantly moving until physical forces reach balanced state. Time to reach balanced state is different in various graphs and depends on multiple metrics. Rendering time is directly influenced by size of graph, number of nodes and edges, strength of edges and many others which interfere with physical forces.

Utilization of CPU, GPU and RAM is another important factor to consider. Game engines work with physical engine natively and do most of the calculations without direct input from programmer or user. However, calculations could stress resources to the point where application become slow and unusable. Based on the method used for initial placement of nodes and edges, the time required for stable state could differ. Without any preprocessing we can position nodes in space on coordinates x=0, y=0, z=0 which is also called space origin. This approach ensures that nodes should not be biased by their initial position. Nodes starting at zero coordinates are more likely to travel further distance until they reach their final position and stabilize. Each unnecessary move of nodes or edges using resources which could be used to process other nodes. Those, the decision about the need for future node processing is quite important from the point of overall visualization efficiency.

Another approach uses random initial position for each node. This approach has the potential to eliminate initial delay (referred in game engines as the lag), because nodes does not have to be rendered at once. Additionally, the sum of forces that are applied to each node is in most cases smaller than the sum forces in first approach because distance between nodes is greater.

Both methods for initial nodes placement must be considered carefully mostly because of expected visualization performance. Node placement to the space origin is in principle better for the application of forces to each node and the final quality of rendered graph visualization. It is easier for the implementation, since all of the work is performed by the physical engine itself. Major drawback comes with the initial delay or lag caused by the fact, that initial application of all forces for nodes from one point needs significant computing resources. Another problem is caused by the physical principle of the graph reorganization. Since the application of forces is physically based, nodes tend to accelerate quickly and thus some additional time and resources are needed for nodes deceleration and reorganization.

Other methods for initial node placement include graph metrics defined for each general graph defined by the graph theory. One of important metrics of graph is the centrality which is represented by number or ranking for each node within the graph corresponding to its network position. Applications include identifying the most influential person(s) in a social network, key infrastructure nodes in the Internet or urban networks, super-spreaders of disease, and brain networks. [7] For legible graph is recommended to calculate centrality beforehand and then use calculated information to create cleaner layout [8, 9].

Force-directed data spatialization

The force in force-directed graphs can be represented in different ways. Unity engine provides multiple options to use such as 1) physical engine by using force functions, 2) spring joints implemented in Unity, or 3) other libraries.

Physical engine contains functions which allows to apply force to game objects. However, they are not suitable for simple visualization, because force is applied either once or for each rendered frame. If the force is applied for each frame, final layout of nodes will most likely end up as one cluster. If graph contains edges with repulsive forces, layout can result in one cluster or layout where distance between nodes has tendency to increase into maximal available distance or infinity.

In our application we used spring joints to represent attractive and repulsive forces. Spring joints can represent repulsive and attractive force with correctly set parameters. Furthermore, edges can be set to maintain desired distance between nodes, which results in cleaner visualization and can serve as boundary which helps to visualize clusters.

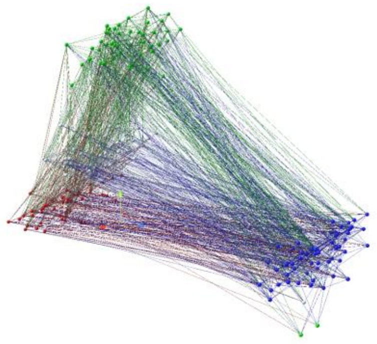

Figure 1: Rendered force-directed graph

The Figure 1 shows final rendered graph created in the Unity engine. Applied forces caused nodes to split naturally into clusters displayed in different colors. All the distances are result of the force application and thus the structure of the resulted graph can be interpreted separately according to appropriate source of data and knowledge domain.

Experimental evaluation

For the experimental evaluation, we developed application for evaluation of optimal number of nodes and edges according to overall visualization and rendering performance. The goal is to find optimal ration of nodes and edges which Unity can calculate without loss of time. Number of nodes and edges rendered and computed in one scene defines boundaries for future visualizations. To know how sparse and connected graphs can be in one scene, set of experimental scenarios were created in environment that focus on game engine performance evaluation.

Nodes and edges are generated with random properties such as color and position, to remove possible bias caused by any specific dataset. We use random positions to remove initial bias caused by accumulated force which is applied when nodes are positioned at the scene origin. Based on the type of starting and end node, edges are created and the adjective force on spring joint is set when both nodes have same type. Otherwise, the repulsive force is set when nodes have different types.

Results

To reach Unity game engine limits we define the set of experiments to be performed. Stable and smooth visualization are important factors for resulting interactive experience but in our experiment, we focus on time required to calculate stable layout. For each scenario, graph made of x nodes is created with different graph density. Each scenario was executed 100 times and final average time was computed.

Table 1 Rendering time for different networks

Scenario	Nodes [count]	Edges [count]	Graph density [%]	Time [s]
1	100	1237	25	27.74
2	100	2475	50	21.76
3	100	3712	75	32.08
4	250	7781	25	31.09
5	250	15562	50	41.08
6	500	31187	25	7.08
7	500	62375	50	8.50

Table 1 shows results from performed experiments. Scenarios 1 to 4 show similar time for graph layout calculation, but they also show that the time does not depend on graph density. This is confirmed by measured times for scenario 6 and 7 where the overall rendering time decreased significantly. This is caused by the density inside defined rendering boundaries where dense graphs converge faster into the balanced state and causing less unwanted oscillations that must be processed.

Graph density is important factor when it comes to practical graph visualization. For dense graphs with density above 50% loss of information is observed, because edges are covering most of the space and hide clusters and nodes which may be important. This can be improved by additional alpha parameter for edges and nodes based on their physical distance from user or by some other techniques for multilayer visualizations.

Conclusion and future work

Game engines limitations define basic bottlenecks for 3D graph visualization. As shown in our article, some limitations are not intuitive enough due to the specific, complex physical environment implemented by game engines. Performance metrics should be thus defined according to experimental results obtained from specific game engines and physical interaction systems.

In future work, experiments will be expanded with the use of different representations of edges. Furthermore, we will be observing graph behavior in segmented space where only part of the graph is visualized at one time. This approach may show us more options for future optimization of layout computing and speed up the process necessary for force directed algorithms.

Acknowledgements

"This publication was realized with support of Operational Program Integrated Infrastructure 2014 - 2020 of the project: Innovative Solutions for Propulsion, Power and Safety Components of Transport Vehicles, code ITMS 313011V334, co-financed by the European Regional Development Fund".

References

[1] "Graph Theory," Wikipedia - The Free Encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Graph_theory. [Accessed 08 2022].

[2] "Gephi - The Open Graph Viz Platform," Gephi.org, [Online]. Available: https://gephi.org/. [Accessed 08 2022].

[3] F. Mcgee, M. Ghoniem, G. Melancon and B. Pinaud, "The State of the Art in Multi-Layer Network Visualization," in *Computer Graphics Forum*, 2019.

[4] "Graph Drawing," Wikipedia - The Free Encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Graph_drawing. [Accessed 08 2022].

[5] J. Mathieu, V. Tommaso, H. Sebastien and B. Mathieu, "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software," *PLOS ONE,* vol. 9, no. 6, 2014.

[6] "Unity," Unity Technologies, [Online]. Available: https://unity.com/. [Accessed 08 2022].

[7] "Centrality," Wikipedia - The Free Encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Centrality. [Accessed 08 2022].

[8] J. M. Hernández and P. Van Mieghem, "Classification of graph metrics," 2011.

[9] M. Zábovský, K. Matiaško and K. Zábovská, "Database Exploration Using Metrics and Visualization," in *Zastosowania Internetu : praca zbiorowa. - Dąbrowa Górnicza: Wyższa Szkoła Biznesu*, Dąbrowa Górnicza, 2012.