Article

Force Directed 3D Graph Visualization Algorithm

Alexander Brezani [*] , Jozef Kostolny , Michal Zabovsky

Posted Date: 23 April 2024

doi: 10.20944/preprints202404.1504.v1

Keywords: graph visualization; spring embedders; force-directed algorithms; 3D visualization; Unity Engine

Article

Force Directed 3D Graph Visualization Algorithm

Alexander Brezani *, Jozef Kostolny and Michal Zabovsky

Faculty of Management Science and Informatics, University of Zilina, Zilina SK-01026, Slovak Republic
* Correspondence: alexander.brezani@fri.uniza.sk

Featured Application: Graph visualization is very useful for better data structure understanding and supports knowledge discovery. There are tools for general graph visualization or graph visualization tools working with relations specific to knowledge domain e.g. molecular biology. Our tool introduces general visualization technique in 3D for fast and extensive graph and network visualization on different datasets.

Abstract: Graph visualization has evolved significantly over time, offering various methods to effectively represent complex data structures. This study introduces a novel force-directed algorithm for 3D data visualization, specifically tailored for commonly used datasets in linked data visualization. Leveraging the power of graph theory and methods, the algorithm aims to address the challenge of creating visually appealing and easily interpretable visualizations, which often necessitate advanced user interaction. By utilizing 3D space and the Unity Engine, the research successfully visualizes data and develops interactive visualizations, overcoming limitations of basic force-directed implementations. The main contribution of presented research is in the force-directed algorithm with springs and controlled placement as a visualization technique that combines the use of springs and attractive forces to stabilize a graph in a 3D environment. The combination of these two techniques also helps prevent the formation of a single large cluster within the graph, enhancing the overall visualization quality.

Keywords: graph visualization; spring embedders; force-directed algorithms; 3D visualization; Unity Engine

---

1. Introduction

Graph visualization has made substantial progress throughout time, with the creation of multiple methods to assist in accurately displaying intricate data structures. This paper especially examines a class of algorithms called force-directed graph visualization algorithms, with a focus on their use in three-dimensional (3D) space. The utilization of these algorithms in 3D environment has distinct obstacles. Adding an extra dimension (the z dimension) might cause the visualization more complex, making it more challenging to provide a clear depiction.

Force-directed graph drawing algorithms are a type of algorithms that can be used to create visually appealing representations of graphs. Their goal is to arrange the nodes of a graph in a way that minimizes the number of crossing edges and ensures that the edges have similar lengths. This is achieved by assigning forces to the nodes and edges based on their positions, and then using these forces to simulate the movement of the nodes and edges or to reduce their energy [1]. Force-directed algorithms are a class of algorithms that are among the most effective at calculating the distributions of simple undirected graphs. These algorithms, referred to as spring embedders, determine the arrangement of a graph only based on the network's structure, without relying on any specific knowledge provided by the vertices, edges, or attributes. Graphs generated using these techniques possess natural aesthetic appeal, demonstrate symmetry, and tend to yield layouts without any intersecting edges for planar graphs. Research conducted in this area has been documented in publications by Di Battista et al. [2] and Brandes [3].

One of the first force-driven methods for drawing (visualizing) graphs is an algorithm based on barycentric representations [4]. More traditionally, the force representation decomposition method is based on Eades [5], but the most common implementation of the algorithm is Fruchterman and

Reingold [6]. Both implementations rely on spring forces defined analogously to Hooke's law. In these methods, there are repulsive forces between all nodes and attractive forces between nodes that are mutually incident.

The advantages of force-driven algorithms in 2D space lie in their simplicity and comprehensibility. This form of visualization provides fast and intuitive data presentations, which is advantageous for various applications of social network visualizations or data structures. However, despite these advantages, 2D space also carries disadvantages. The limited space can lead to overlapping vertices and edges, which reduces the readability of the graph. Also, the lack of realism can limit the ability to capture details of relationships and distances between vertices.

The forces between nodes can be calculated based on their theoretical distances in the graph, determined by the lengths of the shortest paths between them. For example, the algorithm of Kamada and Kawai [7] uses spring forces proportional to the theoretical distances of the graph. In general, force-oriented methods define an objective function that maps each graph layout to a number in R^+ that represents the energy of the layout. This function is defined such that low energies correspond to layouts in which neighboring nodes are close to each other based on some pre-specified distance and in which non-adjacent nodes are well spaced [7].

The analyzed areas of visualization research focus on removing the limitations of conventional methods for representing data (often static and with narrow scope) so that the available information contained in the analyzed data can be extracted [8,9]. The optimal way to achieve such advanced enhancements is to exploit the potential of interactivity to support complex data analysis. Such interactive systems enriched with integrated visual interfaces (synergy of simulation, analysis, and visualization modules) are designed to overcome the limited human analytical capacity to synthesize a multitude of heterogeneous data into informative insights [10].

Linking visualization with interactive features is a key aspect in the design of visual analytics systems [11]. The focus of such interfacing has traditionally been on 2D displays containing abstract data. These techniques are quite appropriate when considering the visualization of continuous numerical attributes to reveal potential relationships, patterns, and correlations between these attributes. However, 2D representations are not suitable for all tasks and do not work at all in certain contexts. For example, when analyzing geospatial data, such visualizations experience overlap (occlusion), leading to loss of information [12]. This is mainly due to the large number of data points that overlap each other, making it difficult for a human analyst to see individual data points and make inferences about potential relationships. This can potentially interfere with visual perception and inferences related to discovery (locating a geometric representation), access (retrieving a shape, color, or other property), and spatial relationships (the interrelationship of geometric representations) [13].

Furthermore, relative positioning in 2D space is not equally suitable for performing the task of absolute positioning in 3D space, because the inherent physical limitations of 2D space make it impossible to account for the full range of other multidimensional geospatial attributes [14]. Alternative visualizations using 3D representations enriched with complex navigation features may therefore be more appropriate when exploring the internal structure of such data. 3D visualizations essentially capture the full spectrum of dimensions of the underlying data and supporting information, while interaction and navigation techniques mimic the sense of exploring the real-world environment in a familiar way.

Presented paper is structured as follows: Materials and Methods sections defines basic goals, principles, and algorithms together with our modifications of formally used techniques, Experimental Results section summarize experimental observations and results, Discussion describes all the iterative steps, findings and solutions based on experimental results and Conclusion shortly concludes article with defined steps for future research.

2. Materials and Methods

The goal of 3D visualization is to provide a visual representation of the complex internal structure of networks, to facilitate the understanding and analysis of network structures, and to reveal patterns and relationships within network data. 3D visualization allows users to explore and interact with the network in a three-dimensional space, providing a more comprehensive and intuitive way of working. Using 3D network visualization, users can gain insight into the spatial arrangement of nodes and connections, identify clusters or communities within the network, and identify any patterns or anomalies that may not be readily discernible in a traditional 2D view. In addition, 3D network visualization can support the exploration of dynamic networks where connections and relationships between nodes change over time. Overall, 3D network visualization aims to improve the understanding and analysis of complex networks, allowing users to make informed decisions and gain deeper insight into the data being analyzed.

The basic force-oriented approaches have a fundamental limitation imposed by the number of vertices to be rendered/processed. Their use is limited to small graphs and the results are poor for graphs with more than a few hundred vertices. There are several reasons why traditional power-oriented algorithms do not work well for large graphs.

One of the main obstacles to their scalability is the fact that the physical model usually has many local minima. Even with the help of sophisticated mechanisms for avoiding local minima, basic power-oriented algorithms are not able to consistently produce good distributions for large graphs. Similarly, barycentric methods do not work well for large graphs, mainly due to resolution issues (for large graphs, the minimum vertex separation is very small, leading to unreadable visualizations) [1].

In the late 1990s, several techniques emerged extending the functionality of power-directed methods to graphs with tens of thousands and even hundreds of thousands of vertices. The main idea behind these approaches is the multilevel decomposition technique, where the graph is represented by a series of progressively simplifying structures and arranged in order from the simplest to the most complex. These structures can be coarser graphs (as in the approach of Hadany and Harel, Harel and Koren, and Walshaw) [15–17] or vertex filtering as in the approach of Gajer, Goodrich, and Kobourov [18].

Our approach follows basic principles introduced for 2D algorithm with respect to specifics of 3D space and implementation based on physical forces. We applied incremental development approach to identify problems during the transformation to higher dimensional space and find solutions for effective and useful visualization in three-dimensional space.

2.1. Simple Spring Algorithm

We are following general definition of mathematical graph where graph $G = (V, E)$ is a mathematical structure consisting of a set of vertices V (also called nodes) and a set of edges E (also called links), where the elements of E are disordered pairs $\{u, v\}$ of distinct vertices, where $u, v \in V$.

The basic implementation of the force-driven algorithm uses the principle of attractive and repulsive forces based on the analogy of electro-magnetic force interaction. Simple spring algorithm is also known as the spring embedder algorithm. The algorithm was formulated in 1984 by Eades [5] with three basic principles.

First, to represent the graph, we replace the vertices with steel rings and replace each edge with a spring to create a mechanical system. The vertices are placed in a certain initial arrangement and spaced so that the spring forces acting on the ring-vertices move the system to a state of minimum energy.

The second principle defines logarithmic force springs - *attractive forces*. The force defined by the spring is then expressed in the form:

$$\mathrm{c}_1 \cdot \log(\mathrm{d} / \mathrm{c}_2) \tag{1}$$

where d is the length of the spring and c_1 and c_2 are constants. Experience shows that Hooke's law springs (linear springs) are too strong when their ends (vertices) are far apart. The solution is to use the logarithmic definition of force. In the case when $d = c_2$, the springs do not evolve any force.

Finally, third principle defines *repulsive forces*. Non-adjacent vertices repel each other and thus, the inverse power is applied defined as:

$$c_3 / d^2 \tag{2}$$

where c_3 is constant and d is the distance between the vertices. Original pseudocode of defined algorithm is [5]:

The values $c_1 = 2, c_2 = 1, c_3 = 1, c_4 = 0.1$ use in Algorithm 1 are suitable for most of the visualization. Previously not mentioned constant c_4 is used in multiplication with resulted force when the vertex is moved to its new position. Almost all graphs reach the minimum energy state after running the simulation step M=100 times. However, the above, based on our experimental findings, is only valid for graphs in 2D space.

Algorithm 1: Spring algorithm by Eades

```
1.  algorithm SPRING(G: graph);
2.  place vertices of G in random locations;
3.  repeat M times
4.  calculate the force on each vertex;
5.  move the vertex c4 * (force on vertex)
6.  draw graph on CRT or plotter.
```

This simple description of the working principle describes the essence of spring algorithms and their inherent simplicity, elegance, and conceptual intuitiveness. The goals of an "aesthetically pleasing" layout were originally defined by two criteria: 1) all edge lengths should be equal, and 2) the resulting layout should exhibit as much symmetry as possible [1].

As the simplest implementation in 3D, we decided to use algorithm based on internal spring system implementation used by Unity Engine [19]. The Unity engine is most widely known for its use in the development of video games; yet it also possesses remarkable applicability in the field of scientific computing. Since it excels in specific domains, it can be an extremely useful instrument when employed strategically. Unity specializes in the creation of dynamic real-time 3D visualizations with implemented physical environment and variety of tools for work in three-dimensional space. When it comes to the exploration of scientific data, this ability proves to be extremely beneficial. Built in 3D physics includes concept of joint which, in principle, could be used for definition of force-based connections between two physical object or between physical object and a fixed point in space. Joints can apply forces that move physical objects and joints limits can restrict that movement [20]. Spring joint concept in Unity Engine allows the distance between two physical objects to change as though they were connected by a spring.

Following the principle introduced by Eades, the algorithm represents a graph as a physical system, where nodes are connected by springs. Each node is assigned two types of forces: attractive forces and repulsive forces. The goal of the algorithm is to minimize the total energy of the spring system by moving nodes to new positions. The attractive force (f_a) acts on neighboring nodes connected by a spring, while the repulsive force (f_r) acts on all nodes of the graph. Attractive and repulsive forces in 3D are defined with respect to original 2D algorithm:

$$f_a(d) = c_a \cdot \log(d), \qquad f_r(d) = c_r \cdot \frac{1}{d^2} \tag{3}$$

where original c_1 is defined as $c_a = 2$, $c_2 = 1$, thus does not appear in our formula, and c_3 is defined as $c_r = 1$. Coefficient $c_4 = 0.1$ for the first 3D implementation is with respect to original definition. The distance d is defined as an abstract unit of distance in accordance with the Unity Engine implementation.

Even though the basic identified difference is in the use of three vertex coordinates in the defined algorithm instead of the original two, we have identified several fundamental limitations and specificities, which we discuss in more detail in the experimental results section.

2.2. Force-Directed Algorithm

The force-directed algorithm pushes the boundaries of using the physics of the individual components of the graph through the physical engine (Unity Engine) further in order to obtain a better graph layout in 3D by incorporating other properties of the data. Proposed algorithm creates graphical representations using an emulated physical force system. Gravity assigns nodes mass proportional to their network centrality, allowing more central nodes to be visualized in central locations of physical space. In addition, the principle of granting the acceleration of the node using the resulting calculated force is used, as opposed to the original calculation of the resulting position of the node after the application of the resulting force calculation.

Algorithm first calculates direction vector by using value for repulsive force $f_r = 1$ and attractive force f_a equal to edge value or $f_a = 1$ in the case the graph has no edge values. Edge values are assigned by data features selected or by other analytical needs. Resulting vector then indicates direction in which appropriate node should be moved. The resulting movement in computed direction is defined by the impulse given to the node by the application of the force F. This force is calculated as the sum of attractive forces f_a for incident vertices minus sum of repulsive forces f_b for non-incident vertices. A node that has its mass m defined based on its degree then gains velocity v based on the initial acceleration a applied by the force F. Resulting movement is defined as uniformly slowed motion by using not only mass m but friction force f_d defined for node as seen in Formula 4. The friction force is defined by drag parameter for object in Unity Engine and is used for calculation of final deceleration for evaluated time interval.

$$a = \frac{F}{m}, \qquad v = v_0 - at \tag{4}$$

It is important to mention that the time t is specific for physical engine implementation. In our experiments $t = 0.02$ seconds which corresponds to 60 FPS defined for real-time visualization of graph. Another observation comes with the friction force which is in fact used as the primary parameter for resulting velocity instead of mass which is used mostly for collision with other physical objects. Based on experimental evaluation, velocity v is internally computed as:

$$v = F \cdot (1 - f_d \Delta t) \tag{5}$$

Defined force driven algorithm is presented by pseudocode described by Algorithm 2.

Algorithm 2: Force-directed algorithm

```
1.  function calculateDirectionVector(graph, node):
2.  direction_vector = (0, 0, 0)
3.  for each n in graph.nodes:
4.  direction = n.directionTo(node).normalized
5.  direction_vector += direction if n.isIncidentTo(node) else -1 * direction
6.  return direction_vector
7.
8.  function calculateForce(graph, node):
9.  force = 0
10. for each vertex in graph.vertices:
11. f = vertex(node).weight if vertex(node).hasWeight else 1
12. force += f if vertex.isIncidentTo(node) else -f
13. return force
14.
15. function calculateVelocity(node, force, mass, initial_acceleration):
16. acceleration = force / mass
17. velocity = initial_acceleration * acceleration
18. return velocity
19.
20. // Force-drive algorithm:
21. for each node in G do
```

```
22.     direction_vector = calculateDirectionVector(graph, node)
23.     force = calculateForce(graph, node)
24.             velocity = calculateVelocity(node, force, node.mass, initial_acceleration)
```

Algorithm 2 can be applied to the graph G with a defined number of steps, in our case we used the definition of the stopping condition. It is obvious that the algorithm does not have a final state, when no force is applied to individual nodes, and therefore, after applying a certain number of steps, oscillations occur between the equilibrium states of the system defined by this graph. This situation is often referred to as jittering. To eliminate this situation, we defined a stop condition based on experimental results when we stop the algorithm if the velocity of 90 percent of the nodes in the graph is less than 1.

2.3. Modified Force-Directed Algorithm with Springs

Both defined algorithms in 3D shows some drawbacks and limitations. Simple spring algorithm is extremely sensitive, and it does not converge to the state when is possible to accept final visualization. It is not only jittering but unbalanced state of whole 3D model changing frequently in time. The force-drive algorithm, on the other hand, shows excessive clustering and node repulsion problem. By combining these two defined algorithms, we were able to eliminate both observed problems. We created a combined algorithm for standard Euclidean space and its modification with a spherical border used to ensure higher compactness and readability of the resulting visualization.

Using springs (spring joints) creates a system in which the vertices are pulled together by attractive forces. This approach leads to a faster stabilization of the network, as the springs ensure a stable operation of the force system. In this approach, edges are represented by attractive forces that pull connected vertices together. This model is advantageous because of fast vertex joining and good internal implementation in physics engines.

In the case of a combination of string and force algorithms, we achieve a balanced approach where springs and attractive forces work together to stabilize the graph. However, it is important to set the force parameters and springs appropriately to achieve an optimal balance between quickly settling the network and minimizing model obscurity. The combination of procedures will also prevent the creation of one large cluster, which could impair the quality of the resulting visualization.

2.4. Force-Directed Algorithm with Springs and Controlled Placement

In this approach we defined special abstract point in the space an anchor. An anchor is defined as a point that prevents the unwanted distance of the visualized nodes of the graph. The reason is problem with network stabilization in the field of view of the camera. This problem is partially solved for nodes that are pushed out of view, but it does not solve the problem of creating one large cluster. One cluster is formed if the attractive forces of the anchor are greater than the sum of the other acting forces in the network. To avoid this undesirable condition, we implemented a spherical constraint that keeps the nodes always at the same distance from the anchor, while at the same time preventing them from moving away in the opposite direction.

2.5. Datasets

During our experimental work we used publicly available datasets with minimal modifications, mostly caused only by customizing for import in the form of a graph. To verify the use of the proposed methods and algorithms, we used basic, comparative benchmark datasets publicly available on the pages of some tools such as Gephi [21], on Kaggle [22], and for large, connected data, a dataset from previous research [23] publicly available on the Snap portal of Stanford University [24]. A general overview of the datasets with their basic characteristics is presented in Table 1.

Table 1. Datasets used for algorithms evaluation.

Dataset name	Description	Number of vertices / edges
Game of Thrones	Character relationships	796 / 3909

Witcher	Fantasy novel series	224 / 2600
Lazega	A network of relationships	36 / 115
Network Science	Network of co-authors	1589 / 2742
Pokec 1000*	Social network data	1000 / 6297

* Subset created for 1000 vertices in specific geographical location.

Game of Thrones [25] and Witcher [26] datasets are available at Kaggle. Lazega [27,28] dataset is originally part of the R (The R Project) libraries and was converted for our experimental framework. Network Science [29,30] data are available for Gephi visualization tool and Pokec 1000 social network dataset is our previous research result with specific number of vertices [31]. All the datasets are publicly available.

3. Experimental Results

In the following section, we present the experimental verification of an innovative algorithm based on the principle of force-driven algorithms, which enables visualization in 3D space, but also integrates spherical representations, normalization, clustering, spring connections, and gravitational interactions to create a robust tool for visualizing complex data networks.

The primary goal is to ensure that edge values in the visualized graph are appropriately normalized, thereby maximizing the interpretability and efficiency of the resulting 3D visualization. Data normalization is especially necessary for parameters used to represent forces and springs, because without normalization spatial complications can occur with vertices on which too much force is applied.

3.1. Evaluation Metrics

In 3D visualization empty space is common and fills most of resulting space of visualized data. Resulting visualization with no additional constraints can lead into two main unsatisfactory outcomes for visualization:

1. A single, unreadable cluster forms within the displayed graph.
2. The distance between nodes always increases, without ever reaching a stable state.

When a single cluster emerges, it's due to the sum of attractive forces between nodes vastly outweighing the sum of repulsive forces in the graph. This cluster, comprising most or all graph nodes, becomes illegible due to the obscuration of edges and nodes by other graph elements. Conversely, when node distances continuously increase, or when the sum of repulsive forces exceeds that of attractive forces, the graph fails to reach a stable state in real-time, rendering the visualization unusable.

To evaluate defined algorithms is necessary define metrics for comparison. Since the use of the criterion of aesthetics of the resulting visualization is problematic without conducting research on users, we used quantitative criteria following two basic problems of visualization algorithms in 3D space. The first criterion is the average distance of the nodes of the graph in the resulting visualization, the second is the average density of the created clusters. The last monitored parameter is the average distance of the created clusters.

Average vertex distance defines overall density of graph. Since both too sparse and too dense a graph represents a problem for the resulting interpretation of the visualization, it is necessary to monitor the average distance of the vertices as a criterion for the quality of the resulting visualization. The pseudocode for calculating the average distance of vertices in the resulting graph is defined by Algorithm 3.

Algorithm 3: Average vertex distance metric

```
1.  // Avg. vertex distance
2.  distances = new List()
3.  for i in nodes:
```

```
4.    for j in nodes:
5.    distances.Add(Distance(i, j))
6.    // sum(distances) / distances.count
7.    distances.Average()
```

Average cluster density tells how close nodes with similar properties are in the final visualization. In the case of a small distance between the nodes in the cluster, the resulting information is difficult to read without interaction with the user in the form of zooming. In case of excessive zooming, information about the overall context of the visualized subset of data is subsequently lost. On the contrary, in the case of too low density, the membership of the nodes within the cluster is not obvious. The defined metric is described by the pseudocode of Algorithm 4.

Algorithm 4: Average cluster density metric

```
1.    // Avg. cluster density
2.    distances = new List()
3.    for each c in clusters:
4.    nodes = c.nodes
5.    distancesCluster = new List()
6.    for i in nodes:
7.    for j in nodes:
8.    distancesCluster.Add(Distance(i, j))
9.    distances.Add(distancesCluster.Average())
10.   // sum(distances) / distances.count
11.       distances.Average()
```

Finally, *average clusters distance* works on similar basis as the distance between vertices, except that it takes groups of vertices into account. Again, the resulting visualization should not lead to an extreme distance between clusters or to their excessive proximity. The metric pseudocode is defined by Algorithm 5.

Algorithm 5: Average clusters distance metric

```
1.    // Avg. clusters distance
2.    distances = new List();
3.    for each centroid in centroids:
4.          for each c in centroids:
5.                if c == centroid continue;
6.                distances.Add(Distance(centroid, c))
7.    // sum(distances) / distances.count
8.    distances.Average()
```

3.2. Simple Spring Algorithm

Although this method produces clean, symmetric layouts for medium-sized graphs, it is considered a demanding algorithm. Its time complexity is often high, making the algorithm impractical in many cases. Additionally, it suffers from inadequate predictability, meaning that repeated runs of the algorithm can lead to different results. This can pose challenges in maintaining the user's mental map during interaction with unstable layouts. Despite these drawbacks, force-directed algorithms are widely used in many visualization frameworks. In our work, the algorithm has been redesigned and optimized to eliminate its characteristic drawbacks, primarily for use in 3D space.

Figure 1 shows the result of the simple spring algorithm in 3D on random generated graph. In the initial experiments, spring connections between nodes that represent attractive forces were tested. At the same time, they also ensure the corresponding repulsive forces, because one of the properties of springs is the contraction of their length. Nodes and edges are generated with random properties such as color and position to remove possible bias caused by any specific data set. We used random

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 23 April 2024 doi:10.20944/preprints202404.1504.v1

positions to remove the initial distortion caused by the accumulated force that is applied when the nodes are placed at the beginning of the scene. Additionally, to direct the edges in the graph, nodes are assigned a color that represents their type. Based on the start and end node types, edges are created, and spring joint strength is set. Spring connections allow a quick and sufficiently clear visualization of the graph, while using the correct settings, they control the expansion of the nodes in space. Their disadvantage lies in the limited forces exerted in one direction and the increased load on the computing unit. Each spring joint must be represented by one specific object in the scene, which causes an increased load not only on the physics but also on the rendering engine.

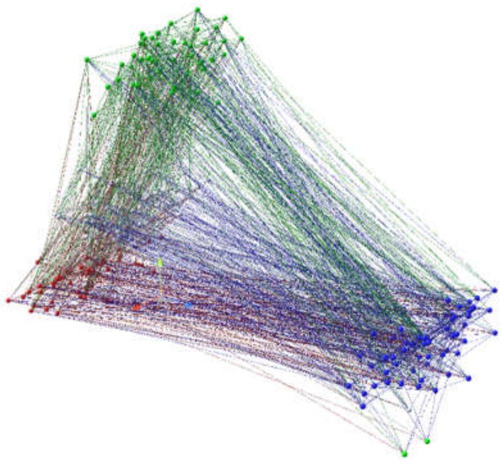

Figure 1. Simple spring algorithm visualization result on randomly generated data.

The resulting visualization shows the fundamental flaw of the algorithm, which is the creation of too dense clusters, while at the same time the clusters are placed at a large distance from each other as seen in Table 2. This is a basic identified shortcoming that led to efforts to improve the used visualization method.

Table 2. Simple spring algorithm experimental results (in abstract units used by Unity Engine).

Dataset name	Avg. vertex distance	Avg. clusters density	Avg. clusters distance
Game of Thrones	348.98	28.65	69.67
Witcher	668.68	27.95	66.71
Lazega	22.01	3.84	26.66
Network Science	166.01	26.89	80.96
Pokec 1000	352.01	26.74	72.03

In practice, a simple spring algorithm can be used to display social networks, relationships in biological systems, or the structure of the web. The advantage is that this approach to graph visualization can reveal patterns and groups of vertices that might be difficult to see in other forms of visualization. This type of produced visualization also has its use in data analysis and complex systems exploration, as their dynamic approach to vertex placement can help reveal structures and relationships that might otherwise remain hidden.

3.3. Force-Directed Algorithm

Proposed force-directed layout algorithm creates graphical representations using an emulated physical force system. The main feature of force-directed algorithms is the physical attraction and repulsion between nodes represented by graph edges. Physical forces enable the creation of clear and understandable node and edge layouts in 3D space without requiring user intervention in the graph assembly process. Since nodes are subject to different forces, situations may arise where nodes are displayed outside the camera's view. In such cases, anchoring or tethering is applied to each node in the network, attracting them towards a reference point. Applying anchoring achieves a state where the entire network is visualized within the camera's field of view.

Results of force-directed algorithm are listed in Table 3. In the comparison with spring algorithm, it is clear, that final visualization needs less space to display result, has lower average density of clusters and even smaller average clusters density. Thus, it better fits the needs for acceptable 3D visualization.

Table 3. Force-directed algorithm experimental results (in abstract units used by Unity Engine).

Dataset name	Avg. vertex distance	Avg. clusters density	Avg. clusters distance
Game of Thrones	100.22	7.14	18.86
Witcher	228.06	11.70	17.23
Lazega	20.17	2.41	6.42
Network Science	28.66	5.47	15.09
Pokec 1000	113.23	15.17	60.68

3.4. Force-Directed Algorithm with Springs and Controlled Placement

By the combination of two previously defined approaches, we were able to achieve a balanced approach where springs and attractive forces collaborate to stabilize the graph, as depicted on Figure 2. The visualization shows, how important is to appropriately adjust the force parameters and springs to achieve an optimal balance between rapid network stabilization and minimizing model clutter. Combining these approaches also prevents the formation of a single large cluster.

Final visualization shows good performance in the identification of clusters, but many vertices are displayed outside the area captured by the camera in 3D. Based on this fact, we proceeded to the application of controlled placement in a combined algorithm, which we consider to be our resulting force-controlled algorithm.

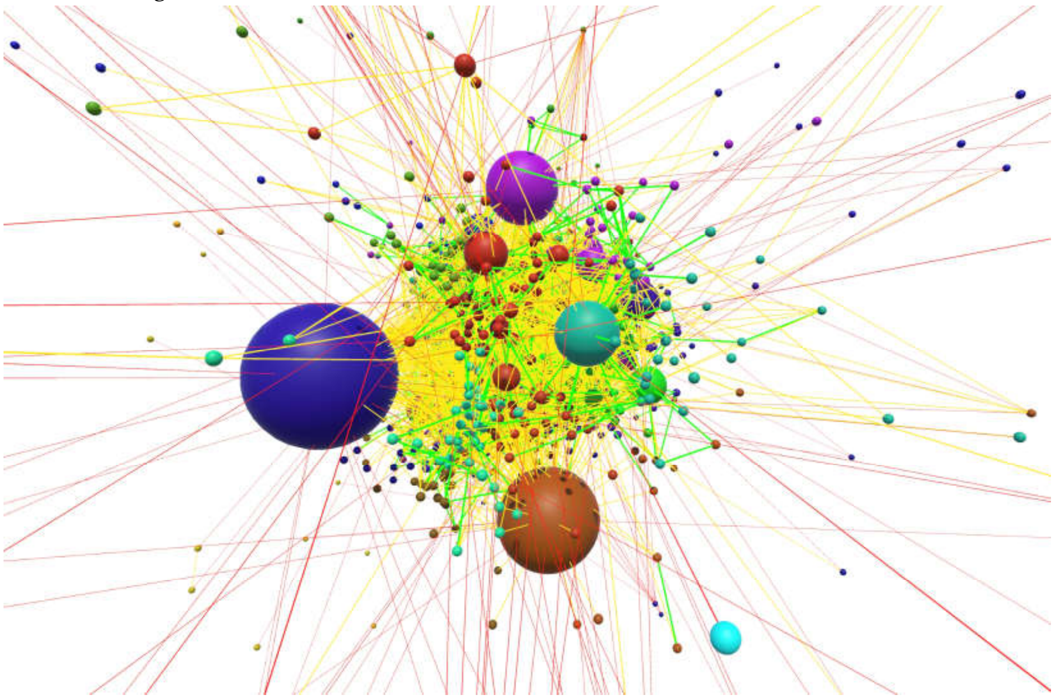

Figure 2. Combination of spring and force-directed algorithms. Final visualization shows good performance in the identification of clusters, but many vertices are displayed outside the area captured by the camera in 3D (dataset Pokec 1000 – 1000 vertices and 6297 edges).

To establish the appropriate methodology for setting constraints in 3D space, a series of measurements focused on detecting limitations and possibilities within the gaming environment were conducted. Recommended limits for the number of gaming objects in one scene were derived from our previous research [32] to create a graph while maintaining temporal and computational complexity below the threshold where visualization becomes unusable. Free 3D space network visualization has several disadvantages that must be considered by the construction of limiting conditions. For instance, the emergence of clusters obstructs network clarity because nodes affected by lesser attractive forces or nodes influenced by excessive repulsive forces obstruct the camera view.

Uncontrolled attractive and repulsive forces in the network can create undesirable clusters unreadable to users. To avoid situations where a node cluster obscures the rest of the visualization, we implemented a constraint on the distance between nodes. This limitation is represented by a spherical barrier serving as a wall for other nodes. The size of the spherical field is directly derived from the size of the node itself larger nodes correspond to larger spherical fields. This approach enables a clearer visualization.

Unlike nodes, edges are allowed to pass through the spherical barrier, creating an interesting effect where connections between selected nodes can be observed without interference from the surroundings. As such, the spherical barrier is mostly transparent but maintains its outline from every angle for consistent representation on the sphere's surface. In 3D graph visualization, effectively displaying distances between vertices is often necessary for better understanding of the graph structure e. g. by utilizing color-coding of edges based on the distance between vertices. In this visualization, edges are color-coded according to the distance between their endpoints. Red color is used for distant vertices, green for close vertices, and yellow for average distance. This color-coding provides a visual indicator of vertex distances in 3D space and enhance the visualization informational value for the user as seen on Figure 3.

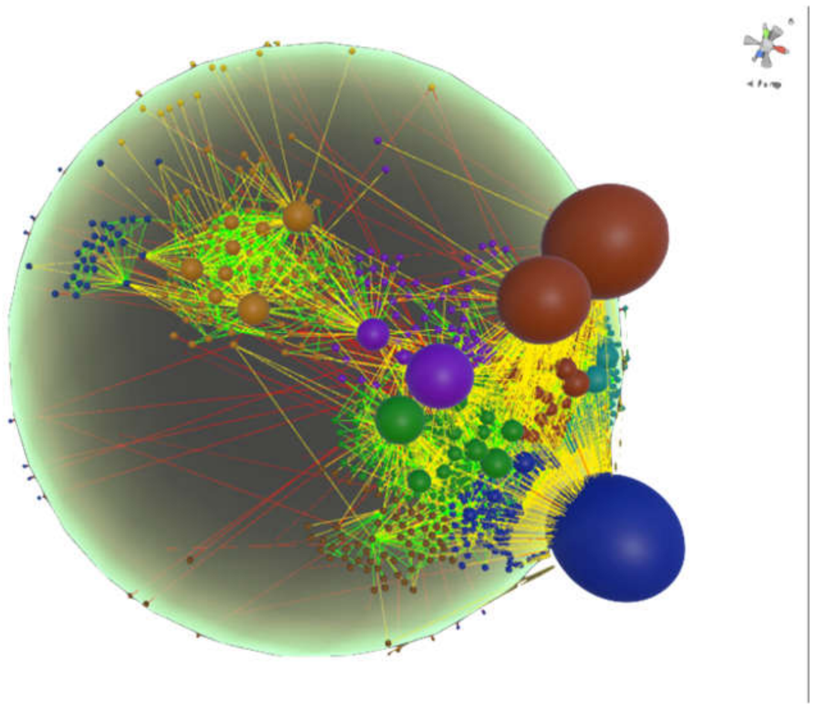

Figure 3. Results of Force-directed algorithm with springs and controlled placement with highlighted effect of spherical border and color-coded edges. Red color is used for distant vertices, green for close vertices, and yellow for average distance (dataset Game of Thrones – 796 vertices and 3909 edges).

Experimental results in Table 4 shows the best performance based on used metrics. Moreover, by using force-directed algorithm with springs and controlled placement we were able to eliminate drawbacks observed in previous implementations based on springs or general force-directed approach. Defined algorithm helps users in 3D quickly identify distant and close parts of the graph, as well as areas requiring further attention or analysis.

Table 4. Force-directed algorithm with springs and controlled placement experimental results (in abstract units used by Unity Engine).

Dataset name	Avg. vertex distance	Avg. clusters density	Avg. clusters distance
Game of Thrones	230.41	11.32	46.25
Witcher	378.91	16.43	42.34
Lazega	73.96	4.86	12.63
Network Science	92.57	6.04	29.33
Pokec 1000	310.08	14.33	55.17

Overall experimental result obtained for defined metrics are in Table 5. The best values are written in bold for easier identification since we are trying to find solution with no extreme placemen. As seen, by iterative development we were able to overcome original drawbacks of force-directed algorithm for 3D environment for most of the datasets or we were very close to the best solution.

Table 5. Overview of performance metrics for Simple spring algorithm, Force-directed algorithm and Force-directed algorithm with springs and controlled placement.

Dataset name	Algorithm	Avg. vertex distance	Avg, clusters density	Avg. clusters distance
Game of Thrones	Spring	348.98	28.65	69.67
	Force-directed	100.22	7.41	18.86
	Force-directed SCP*	**230.41**	**11.32**	**46.25**
Witcher	Spring	668.68	27.95	66.71
	Force-directed	228.06	11.70	17.23
	Force-directed SCP*	**378.91**	**16.43**	**42.34**
Lazega	Spring	**22.01**	**3.84**	26.66
	Force-directed	20.17	2.41	6.42
	Force-directed SCP*	73.96	4.86	**12.63**
Network Science	Spring	166.01	26.89	80.96
	Force-directed	28.66	5.47	15.09
	Force-directed SCP*	**92.57**	**6.04**	**29.33**
Pokec 1000	Spring	352.01	26.74	72.03
	Force-directed	113.23	**15.17**	**60.68**
	Force-directed SCP*	**310.08**	14.33	55.17

* Force-directed 3D algorithm with springs and controlled placement.

Example of visualization by our algorithm for dataset Game of Thrones is in Figure 4 together with the visualization of the same dataset in 2D by using Gephi tool. For thousands of vertices and edges it is much easier to understand internal structure of data and easily distinguish between different types of data by defined clusters. In 2D is possible to improve visualization by additional iterative steps and create more useful model of the network but the picture show comparison of default behavior with no additional interaction with user in 2D and 3D.

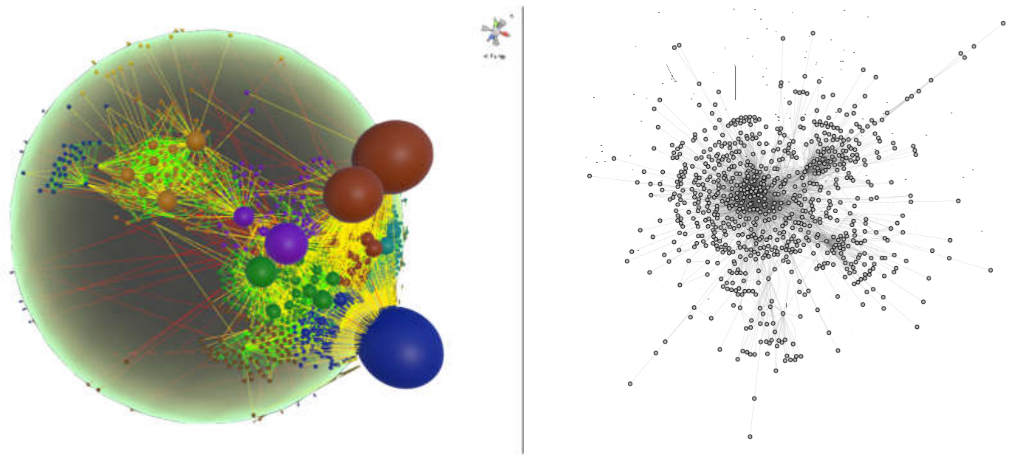
(**a**) (**b**)

Figure 4. Comparison of 3D and 2D visualization for dataset Game of Thrones with 796 vertices and 3909 edges: **(a)** Force-directed 3D algorithm with springs and controlled placement; **(b)** Simple force-driven 2D algorithm implemented in Gephi.

4. Discussion

Implemented algorithms utilize the internal properties of graphs, which facilitates their straightforward application to various types of data. Because the algorithms are built on a 2D base, it is critical to evaluate the obstacles and issues associated with moving algorithms to 3D. In 2D space, the absence of third dimension acts as a constraint, eliminating several undesired viewing states such as vertex overlap in space, among others.

The proposed force-directed algorithm with springs and controlled placement for data visualization in 3D was created iteratively, with new procedures and constraints being gradually introduced, leading to new findings that contributed to the current state of development. In the first iteration, we focused on representing edges in the graph using spring connections. The results of experiments revealed limitations of 3D space and especially the physical gaming environments themselves, determining the maximum usable data volumes that can be visualized using these tools. Another significant finding from the experiments conducted in the first iteration is that vertices in the graph tend to oscillate to levels where the graph can no longer be stabilized when using spring connections. Furthermore, we discovered that spring connections, as represented in game engines, are extremely sensitive to changes in their internal characteristics, which are not well specified for the type of data. Therefore, in our proposed algorithms, we decided to set spring connections as constants, with the only parameter being the normalized force.

The experiment findings demonstrated that while this strategy reduces the model's inclination to oscillation, it also brings additional issues in keeping the model within the camera's field of vision. The forces applied to individual vertices are determined by the nature of the data, and in many cases, vertices subjected to large forces gained so much acceleration that they became disconnected from the rest of the model. This phenomenon can also be seen in 2D environments with algorithms such as Force Atlas 2 [33]. Another undesirable behavior was the formation of a single huge cluster containing most of the data, making the visualization unreadable.

To address these drawbacks, we merged the individual approaches, namely spring connections and constantly applied force, into basic implementation of force-directed algorithm. This technique overcomes oscillation concerns by applying continuous forces, and spring connections keep all vertices within the camera's field of vision. This algorithm produces the best results in terms of complexity when used to simple 3D computations. Nevertheless, there are specific limitations that could potentially obscure or distort the data. The inability of this algorithm to maintain the final model at a particular point in space is one of the major flaws. Despite eliminating the oscillations of individual vertices, the model continues to be subject to a force that compels it to move in a specific direction. The model might accelerate to the point where the camera is unable to track it at a comfortable rate.

To overcome this issue, in the following iteration, we introduced a gravitational force known as an anchor, which maintains the model centered on a single point. The anchor applies the same constant attractive force to each vertex in the model, regardless of the interactions between individual vertices. This method is simple to use, but with a badly set constant force, the entire model may end up in a single cluster, as in the second iteration. To address the issue of a single huge cluster, the entire picture was converted into a spherical projection. In this scenario, the sphere serves as a barrier for vertices, guaranteeing that all vertices are at the same distance from the stated anchor. This iteration of our algorithm ensures clear visualization for most data sets while eliminating most of the undesirable occurrences from the first three rounds.

In successive versions of the algorithm development, our focus was on eliminating obstacles that hinder the clear viewing of 3D objects, such as the overlapping of vertices or the inadequate representation of data on the sphere. In contrast to 2D space, where the lack of one dimension simplifies the issue of vertex overlap, in 3D, we had to use an alternative method including a constant vertex constraint and dynamic repulsive forces.

When using a constant vertex constraint, each vertex defines with its own space, preventing other vertices from approaching within a certain distance to avoid being occluded. However, a problem arises with larger data volumes because vertices may indirectly affect the relationships around them by unnaturally pushing everything away from their center. On the other hand, according to experiments, this approach proved to be effective for smaller volumes, mainly due to computational complexity and implementation simplicity. The second method involves the integration of dynamic repulsive forces, which are active only when vertices approach a defined distance. Each vertex has a distance defined according to its degree, ensuring that vertices with a high degree do not hinder the movement of vertices with a low degree around the sphere. These approaches can be combined, but the condition that dynamic repulsive forces will be active before the vertex approaches the distance defined in the constant constraint must be maintained.

Finally, we concentrated on the graph's edges, including their color, shape, and other features that can be used for edge visualization. Edge coloring can be related to a variety of data-derived features; in our version, it is tied to Euclidean distance to improve spatial awareness. Furthermore, this attribute is consistent with the concept of force-directed algorithms in that spatial distance is an internal property of the graph, therefore no external intervention is required to identify it, and thus the generated algorithm can continue to function in the traditional fashion.

5. Conclusions

We introduced new force-directed algorithm for data visualization in 3D with respect to currently available datasets commonly used for linked data visualization. Graph as the representation for connected data is very useful structure, since it can gain advantage by graph theory and methods but producing nice and readable visualization is always difficult task which requires advanced user's interaction. Using 3D space, we were able to visualize data with the help of Unity Engine and create interactive visualizations with eliminated drawback of basic force-directed implementations.

Future development will be focused on the implementation for larger datasets and development of extensive visual analytics features into application for 3D visual data exploration. As seen, even simple implementations of force-directed principle and spring embedders shows potential for further development and modifications which we are using in 3D visualization preparation for domain specific data.

Author Contributions: Conceptualization, A.B. and M.Z.; methodology, A.B. and J. K.; software, A. B.; validation, A. B., J.K. and M.Z.; formal analysis, J.K.; resources, A.B.; writing—original draft preparation, A.B.; writing—review and editing, J.K.; visualization, A.B.; supervision, M. Z. All authors have read and agreed to the published version of the manuscript.

Data Availability Statement: The original contributions presented in the study are included in the article material, further inquiries can be directed to the corresponding author/s.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kobourov, S.G. Force-Directed Drawing Algorithms. In *Handbook of Graph Drawing and Visualization*; Tamassia, R., Ed.; CRC Press, 2013; pp. 383–408.
2. Tollis, I.G.; Di Battista, G.; Eades, P.; Tamassia, R. *Graph Drawing: Algorithms for the Visualization of Graphs*; Pearson: Englewood Cliffs, 1998; ISBN 978-0-13-301615-4.
3. Brandes, U. Drawing on Physical Analogies. In *Drawing Graphs*; Kaufmann, M., Wagner, D., Eds.; Lecture Notes in Computer Science; Springer: Berlin, Heidelberg, 2001; Vol. 2025, pp. 71–86 ISBN 978-3-540-42062-0.
4. Tutte, W.T. How to Draw a Graph. *Proc. Lond. Math. Soc.* **1963**, *13*, 743–767.
5. Eades, P. *A Heuristic for Graph Drawing*; Congressus Numerantium, 1984; Vol. 42;.
6. Fruchterman, T.M.; Reingold, E.M. Graph Drawing by Force-Directed Placement. *J. Softw. Pract. Exp.* **1991**, *21*, 1129–1164, doi:https://doi.org/10.1002/spe.4380211102.
7. Kawai, S.; Kamada, T. An Algorithm for Drawing General Undirected Graphs. *Inf. Process. Lett.* **1989**, *31*, 7–15, doi:https://doi.org/10.1016/0020-0190(89)90102-6.
8. Anouncia, S.M.; Gohel, H.A.; Subbiah, V. *Data Visualization: Trends and Challenges Toward Multidisciplinary Perception*; 1st ed.; 2020; ISBN 978-981-15-2281-9.
9. Schmidt, J. Visual Data Science. In *Data Science, Data Visualization, and Digital Twins*; Shirowzhan, S., Ed.; IntechOpen: Rijeka, 2022 ISBN 978-1-83962-944-0.
10. Vuckovic, M.; Schmidt, J.; Ortner, T.; Cornel, D. Combining 2D and 3D Visualization with Visual Analytics in the Environmental Domain. *Information* **2022**, *13(1)*, doi:https://doi.org/10.3390/info13010007.
11. Fisher, B.D. Visual Representations and Interactions Technologies. In *Illuminating the Path: A Research and Development Agenda for Visual Analytics*; Thomas, J., Cook, K., Eds.; IEEE Press, 2005; pp. 69–104.
12. Amini, F.; Rufiange, S.; Hossain, Z.; Ventura, Q.; Irani, P.; McGuffin, M.J. The Impact of Interactivity on Comprehending 2D and 3D Visualizations of Movement Data. *IEEE Trans. Vis. Comput. Graph. Zv* **2015**, *21*, 122–135, doi:10.1109/TVCG.2014.2329308.
13. Elmqvist, N.; Tsigas, P. A Taxonomy of 3d Occlusion Management for Visualization. *IEEE Trans. Vis. Comput. Graph.* **2008**, *14*, 1095–1109, doi:10.1109/TVCG.2008.59.
14. Bleisch, S.; Nebiker, S. Connected 2D and 3D Visualizations for the Interactive Exploration of Spatial Information. In Proceedings of the The International Archives of Photogrammetry, Remote Sensing and Spatial Information Science; Beijing, China, 2008; Vol. XXXVII, Part B2.
15. Hadany, R.; Harel, D. A Multi-Scale Algorithm for Drawing Graphs Nicely. *Discrete Appl. Math.* **2001**, *113*, 3–21, doi:https://doi.org/10.1016/S0166-218X(00)00389-9.
16. Harel, D.; Koren, Y. A Fast Multi-Scale Method for Drawing Large Graphs. In Proceedings of the Lecture Notes in Computer Science; Marks, J., Ed.; Springer: Berlin, Heidelberg, 2002; pp. 179–200.
17. Walshaw, C. A Multilevel Algorithm for Force-Directed Graph Drawing. In Proceedings of the Journal of Graph Algorithms and Applications; Marks, J., Ed.; Springer: Berlin, Heidelberg, 2001; Vol. 1984, pp. 171–182.
18. Gajer, P.; Goodrich, M.T.; Kobourov, S.G. A Fast MultiDimensional Algorithm for Drawing Large Graphs? *Proc. 8th Int. Symp. Graph Draw.* **2000**, 211–221.
19. Unity Technologies Unity Engine Available online: https://unity.com/products/unity-engine (accessed on 17 April 2024).
20. Unity Technologies Unity Manual: Joints Available online: https://docs.unity3d.com/2023.2/Documentation/Manual/joints-section.html (accessed on 17 April 2024).
21. Bastian, M.; Heymann, S.; Jacomy, M. Gephi: An Open Source Software for Exploring and Manipulating Networks. In Proceedings of the Third International AAAI Conference on Weblogs and Social Media; 2009; Vol. 3, pp. 361–362.
22. Kaggle Available online: https://www.kaggle.com (accessed on 17 April 2017).
23. Takac, L.; Zabovsky, M. Data Analysis in Public Social Networks. In Proceedings of the International Scientific Conference & International Workshop Present Day Trends of Innovations; Lomza, Poland, 2012.
24. Leskovec, J. Stanford Network Analysis Project Available online: http://snap.stanford.edu (accessed on 17 April 2024).
25. Marchetti, M.M. Game_of_thrones_dataset Available online: https://www.kaggle.com/datasets/mmmarchetti/game-of-thrones-dataset (accessed on 17 April 2024).
26. Sadasivan, A. Witcher Network Available online: https://www.kaggle.com/datasets/avasadasivan/witcher-network (accessed on 17 April 2024).
27. Lazega, E. Lazega Lawyers Network Data 2001, *Cit. 1*.

28. Lazega, E. *The Collegial Phenomenon: The Social Mechanisms of Cooperation Among Peers in a Corporate Law Partnership*; Oxford University Press: Oxford, 2001;
29. Newman, M.E.J. Finding Community Structure in Networks Using the Eigenvectors of Matrices. *Phys. Rev. E* **2006**, *74*, doi:https://doi.org/10.1103/PhysRevE.74.036104.
30. Newman, M.E.J. Coauthorships in Network Science Available online: https://github.com/gephi/gephi/wiki/Datasets (accessed on 17 April 2024).
31. Takac, L.; Zabovsky, M. Pokec Social Network.
32. Brezani, A.; Zabovsky, M. 3D Graph Visualization Performance Using Physical Engine. *Int. J. Inf. Technol. Eng. Manag. Sci.* **2022**, *6*.
33. Jacomy, M.; Venturini, T.; Heymann, S.; Bastian, M. ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software. *PLOS ONE* **2014**, *9*, doi:https://doi.org/10.1371/journal.pone.0098679,.