Generating a Novel Scene-Graph Structure for a Modern GIS Rendering Framework

David Tully, Abdennour El Rhalibi, Christopher Carter, Sud Sudirman
School of Computing and Mathematical Science
Liverpool John Moores University
Liverpool, UK
{ D.Tully@2008, A.Elrhalibi@, C.J.Carter@, S.Sudirman@ }.ljmu.ac.uk

***Abstract*—Within this paper we discuss and present a novel modern 3D Geographical Information System (GIS) framework Project-Vision-Support (PVS). The framework is capable of processing large amounts of geo-spatial data to procedurally extract, extrapolate, and infer properties to create realistic real-world 3D virtual urban environments. The paper focuses on the generation of a novel scene-graph structure used in a number of algorithms and novel procedures for the increased rendering speeds of large virtual scenes and the increased processing capabilities as well as ease of use to manipulate a worlds worth of data. The scene-graph structure, made of two sections, depicts the spatial boundaries of the UKs Ordnance Survey (OS) scheme down to 1km^2. Each 1km^2 node contains the second section of the scene-graph structure, generated from the OpenStreetMap (OSM) classifications; involving buildings, highways, amenities, boundaries, and terrain. Leaf nodes contain the model mesh data. Generation of the spatial scene-graph for the UK takes 7.99 seconds for 6,313,150 nodes. The scene-graph structure allows for fast dispersal of render states, as well as scene manipulation by pre-categorising the data into branches of the scene-graph structure. Searching a node by name is evaluated using depth-first-search and breadth-first-search giving 0.000186 and 0.036914 seconds respectively within a scene-graph of 3257 nodes.**

Keywords— GIS, Scene-Graph, OpenStreetMap, XNA, Project-Vision-Support

I. INTRODUCTION

Modern GISs are restricted through a number of factors such as licenses which state the data or software cannot be used for certain applications or domains. Open data does not include license restrictions but is often inaccurate or error prone. Accurate data is expensive to capture or create. Current applications such as Google Maps[1] restrict user's interaction within a scene. They do not generate realistic 3D scenes in which a user can navigate and interact with in any manner they wish. These problems can be overcome with the generation of a novel and open framework dedicated to the processing of multiple types of geographical/geospatial data to analyze and infer further accurate or near accurate data to be visualized using novel data structures by a rendering engine. A framework such as this must allow users to view and manipulate scenes in a flexible manner and view a scene depicting data for which they are interested in. A project which utilizes a similar technique is the SAVE project [1], [2]. Combining GIS data and a modern computer game engine is legitimately sensible. Game engines have considerably fallen in price and reduced the need of domain experts through ease to use Application Programming Interface (APIs). A framework must be able to generate 3D model assets for viewing, as well as additional data for categorization and analysis. For 3D generation of assets, models can be generated by an artist within an external modelling application. More commonly, Procedural Content Generation (PCG) is being employed to generate assets by data-driven processes. A framework capable of depicting a worlds worth of data, realistically needs advanced procedural generation techniques helped by supporting algorithms. This framework we introduce is Project-Vision-Support (PVS).

The scene-graph structure is the focus of the paper. We discuss the generation of a novel scene-graph structure used within the pre-processor and runtime system of PVS. The need for a new scene-graph data structure is through the large amounts of varying data obtained from OSM, OS, and LiDAR. Structuring this data is a concise way, replicating real-world categorizations of assets (buildings, highways, amenities etc.). This allows lookup references to particular types of nodes, improving search time and processing by pruning unneeded branches. We also discuss the ability to combine the scene-graph structure with a rendering technique called 'array indexed shader function', or as the game industry has coined 'UberShader'. We also introduce the data sets we will utilize within the PVS framework. The data we have chosen for the project has depicted the generation and reasoning behind the scene-graph structure; the base of the scene-graph depicts the spatial organization obtained from the UKs Ordnance Survey, and the top of the scene-graph is generated from input of OpenStreetMap categorizations; splitting the scene-graph into a spatial partition as the base, and separating city assets for the top of the scene-graph.

II. A FRAMEWORK FOR A MODERN GIS

PVS is generated of offline and online processes, parsing real-world open data for the procedural generation of realistic virtual urban-environments allowing users to select and visualize assets within a scene under their own interests. PVS is built of integral separate libraries, each for a specific purpose, and interacting with the other libraries in a specific configuration. The libraries we have created are;

[1] https://www.google.co.uk/

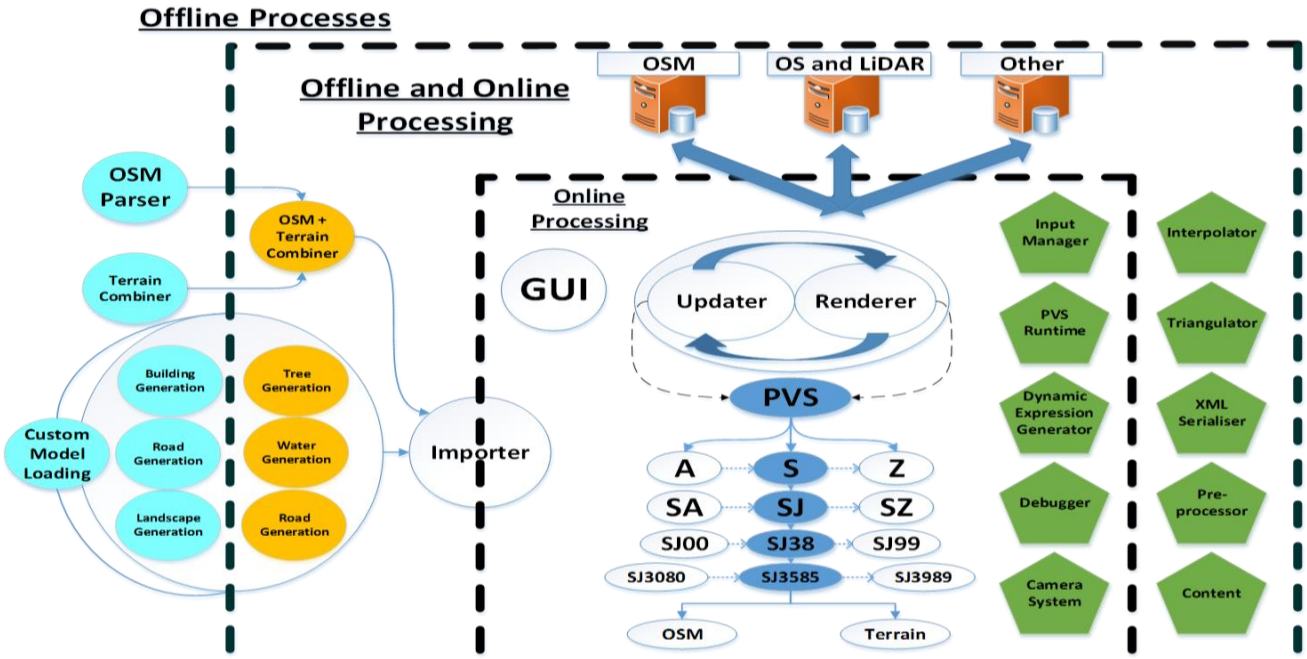

Figure 1 High level overview of the PVS framework. Blue are offline processes. Yellow is both offline and online processes. Green represents libraries within the framework.

- **Triangulator** – generates polygonal triangles from an arbratory list of 2D points. Used for creating boundaries and rooftops of buildings.
- **XMLSerialiser** – serialises C# objects to XML, and vice-versa.
- **Content** – raw textures, models, and geospatial data.
- **Tweener** – an interpolation library commonly used with animation techniques but we also use it for the generation of highway model meshes.
- **Dynamic Expression Library** – Allows user generated procedural query expressions to be applied to all objects within a scene.
- **Input Manager** – keyboard, mouse, and touch input.
- **2D/3D Camera System** – allow the viewing of a scene in a multiple of perspectives by 3D and 2D camera systems.
- **Pre-processor** – generation and compilation of assets by means of PCG, and classification of the geo-data.
- **Runtime** – Contains the 3D model object class, scene-graph structure, and screen manager system.

We also use specific class objects shared between the pre-processor and runtime system, which need to be duplicated, or vary slightly. This is a troublesome issue for maintenance, but removes the issue of the dreaded diamond configuration[2].

- **Scene-Graph structure** – discussed later.
- **OSM Tag Data** – data primarily extracted from the OSM database but can contain inferred data.
- **Ordnance Survey Reference lookup table** – due to the size of the UK OS grid, we produce a procedurally generated lookup table for fast retrieval of results.
- **Material Structure** – this is the material object used in conjunction with rendering the 3D assets. A 3D model is generated in the pre-processor with a material object which is generated by inferring data from the OSM Tag Data as stated above, or from the model object created by a 3D modelling program.

Figure 1 shows the high level overview and connections between the created libraries and the pre-processor and runtime system, and where the scene-graph sits within the runtime application.

A. Data

Within this section we state the data we use within PVS, and its relevance to the generation of the scene-graph structure.

1) UK Ordnance Survey

The OS reference scheme used for splitting the UK into manageable 1km^2 areas or divisions of 10. We copy this spatial partition for the base of our scene-graph structure. OS data is height data available from the UK Government. The files contain height points separated 50meters apart. It is critical that the OS data is accurate for the algorithms used within PVS. Using OS data for realistic terrain visualizations is undesirable due to its low resolution. Terrain characteristics are lost at this resolution. Higher resolution terrain data is needed. OS data is used as a base for map generation procedures combined with error prone LiDAR data files discussed in a future section.

The spatial partition references are generated as shown in the code below. The code is written within the C# language but can easily be ported to many other languages due to the simply nature of code and single object class. This improves scalability and flexibility. Firstly, a class object to store the references is needed. The object will store a name reference,

[2] https://en.wikipedia.org/wiki/Dependency_hell

and a List or an Array of children. A List in this context and language simplifies traversal techniques, but increases traversal processing. Figure 2 depicts the algorithm used.

```
List<string> tsingleprefixes = new List<string>();
tsingleprefixes.Add("a");
// ...
tsingleprefixes.Add("z");

_osworldnode = new OSNode("World");

foreach (string tprefix1 in tsingleprefixes) {
OSNode t500knode = new OSNode(tprefix1);
_osworldnode._children.Add(t500knode);
    foreach (string tprefix2 in tsingleprefixes) {
    OSNode t100knode = new OSNode(tprefix1 + tprefix2);
    t500knode._children.Add(t100knode);

        for (int tnorthing10k = 0; tnorthing10k < 10; tnorthing10k++) {
        for (int teasting10k = 0; teasting10k < 10; teasting10k++) {
        OSNode t10knode = new OSNode(tprefix1 + tprefix2 +
                                     tnorthing10k.ToString() +
                                     teasting10k.ToString());
        t100knode._children.Add(t10knode);

            for (int tnorthing1k = 0; tnorthing1k < 10; tnorthing1k++) {
            for (int teasting1k = 0; teasting1k < 10; teasting1k++) {
            OSNode t1knode = new OSNode(tprefix1 + tprefix2 +
                                        tnorthing10k.ToString() +
                                        tnorthing1k.ToString() +
                                        teasting10k.ToString() +
                                        teasting1k.ToString());
            t10knode._children.Add(t1knode);
```

Figure 2 Code to generate the OSNode spatial partition scene-graph

The reason we do not contain bounds in the form of bounding box structure is due to the amount of memory needed, and the generation will negate the speed of the generation of the scene-graph. Parsing the OSNode name and procedurally generating bounding box structures are used to partition and store the PVG assets within 1km^2 nodes of the scene-graph; i.e. if an asset is within a 1km^2 node bounds, then attach it to the appropriate scene-graph branch.

To generate a complete scene-graph, we utilise a pre-defined reference list which contains the single letter prefixes. These prefixes represent the highest level of the OS reference scheme. A node of this level represents a 500km^2 area. To generate the scene-graph structure, a series of for-loops are needed. Each loop will be responsible to generate the children of the parent node; starting with a single prefix (S), to a double prefix which represent the 100k^2 areas (SJ), to the double prefix with 10k northing and easting which cover 10km^2 area (SJ39), to the double prefix with the 1km^2 references (SJ3090).

The time taken to generate the complete coverage of the OS reference schemes, which is a 2,500km^2 area containing; 25 500km *OSNode* references, 625 100km *OSNode* references, 62,500 10km *OSNode* references, and 6,250,000 1km *OSNode* references, takes 7.9888502 seconds. This does not include the bounding boxes.

This section of the scene graph is simply a conversion of the of the OS schema and will be used to organize the geo-spatial data of OSM, OS, and LiDAR. Before we discuss the addition of the OSM classification branches, we state that the spatial scene graph structure explained is the first half of the scene graph structure and used to spatially organize the OSM and the terrain model data. We will explain the next step for creating a full scene graph structure for a modern GIS within the pre-processing section.

The top of the scene-graph will be a predefined node reference which we state is the *'World'* and all other nodes will be attached as a child, or a grandchild.

2) Light Detection and Ranging

Light Detection and Ranging (LiDAR) is the technique of pulsing light beams, normally from a low-flying aircraft, and recording the time taken to bounce back to a sensor. Leberl et.al [3], claims that LiDAR is rapidly replacing the photogrammetric approach due to its reliability and ease of capture compared to that of the commonly used photogrammetry. We add to this claim with the research of Montoya et.al. [4] whom use LiDAR equipment attached to drones. This removes the huge cost of capturing aerial imagery needed by photogrammetry. LiDAR provides data accurate to 0 ~ 15cm of error. Data points can be achieved to a high resolution up to 25cm. A large problem with the LiDAR data is the large amounts of data, the multiple resolutions which we have (2m, 1m, 50cm, and 25cm), and the lack of accurate data, or the total lack of any data for unmapped areas. Some areas of the UK are only mapped at 1 resolution, or a multiple of the 4 resolutions. As stated, to counter act this issue and create map files which contain a complete, but may not be highly accurate of the real-world terrain, set of data points, OS data will be used as a base for all processes for the generation of additional data. OS data is processed and interpolated to 2m resolution to then be added to the LiDAR maps. Repeating this processes through the resolutions, complete terrain data sets can be generated. When we state complete we justify this as a terrain map which has no missing data values, even if the values are not accurate to the real-world terrain, and simply interpolated with neighboring data points using Catmull-Rom [5], [6]. Terrain maps will contain within their own branch of the scene-graph. Multiple resolutions of the terrain model mesh can be attached. For now, we develop the prototype with only 1m resolution terrain maps. Referencing the terrain model mesh within its own branch of the scene-graph, easy node lookups can generated. This also allows custom search techniques to be implemented for searching of singular or groups of terrain model meshes being analyzed.

3) Open Street Map

OSM is a volunteered mapping project which maps all areas of the globe from an army of volunteers and Government sponsored data. OSM data is exported from the web portal. It contains *Nodes*, *Ways*, *Relations*, and *Tag* data. For more information of OSM data see [7]–[9]. For this work we discuss the OSM partitions needed to generate a novel scene-graph structure for partitioning OS, LiDAR, and OSM data. The OSM data categories we have, splits the data into manageable branches of the scene-graph structure. Each

branch is pertinent to our visualisations, and are selected with commonality in mind. Within a city, overarching asset types emerge through classification; buildings, highways, amenities, waterways, and a few others. Each of these can be sub-classed:

- Buildings (**Emergency** (*Hospital, Police, Health, Military*), **Other**)
- Highways **(Roads, Link, Paths, Special, Waterways, Railways**)
- Boundaries (**Water, Emergency, Other**)
- Amenities

We have not stated all variants of the Highway types due to the amount of categories. The 'Link' highways are highways which connect the different types of highways together. For example, to connect to a motorway from a lower graded highway, then a link is needed.

Open data and OSM have been used to generate many variations of common computer games. Games set within real-world parameters: Friberger et.al [10][11] use open GIS data to create interesting variations of well-known games; Monopoly with real-world place names, Top Trumps with real-world countries with accurate states, and using real-world maps to create 2D tile maps for a Civilisation style game. This proves well for our work.

B. Scene-graphs

Scene-graph technology has been used in A closely related Doctoral Thesis of Bo Mao, '*Visualisation and Generalisation of 3D City Models*', has many similarities to our research, and the he explains the need for a modern GIS which integrates 3D objects for improved realism [12]. The use of CityGML[3] is used to generate building model mesh data and rendered using X3D[4]. The work and research is built for mainstream web browsers. Due to the lack of infrastructure and rendering capabilities of modern web browsers, he employs a LoD technique which renders a scene in different scales; block, building, and building with façade to improve rendering speeds and scene realism. He introduces a novel scene graph structure call CityTree, to represent groups of buildings. This proves the needs for custom scene-graph structures to organise geospatial data, and apply novel algorithms to view real-world scenes in a number of options.

C. Preprocessor system

The pre-processor pre-compiles the runtime asset data for the added benefit of; increased load time, typification and classification of GIS data, error checking and data augmentation, among others. To pre-process the OSM data, data is passed through a custom pipeline. The pipeline organizes, categories, checks for errors within data, combines data with additional data-sets for the inference, and augmentation of the raw data at hand.

A raw OSM XML map file is parsed into a C# class object. Multiple processes are applied to the class object to generate additional data. This data is generated for future processes. With each *Node* and *Way* of OSM having custom reference IDs we place the *Nodes* and *Ways* into individual Key/Value data structure (C# Dictionary). The Key will be the Node or Way ID, and the value will be said *Node* or *Way*. This allows speedy looping and lookups of the *Nodes*. This is especially needed when checking the *Nodes* within a *Way* or the *Ways* within a *Relationship*. Each of the data types are checked for duplications and other errors which are a major problem with large OSM maps. While processing the *Nodes* and *Ways*, additional data is produced. We convert the Longitude and Latitude to X, Y coordinates utilising DotNetCoords[5]. Converting form Longitude and Latitude, to X, Y coordinates is currently not accurate due to conversion error due to the curvature of the Earth not being uniformly round. This error increases the further away from the (0,0) coordinate which is located near the bottom left of the UK. We plan to utilise this error by sampling error at distinct distances and recording the error. Using this as a benchmark, we can negate this error to improve the conversion rate. This is needed due to the multiple projections used between OSM, and OS/LiDAR.

We also generate the centroid of a *Way*, and store this within the *Way* class object. Because a *Way* is a boundary, it can be thought of as a none-overlapping polygon.

We utilise the spatial scene-graph to organise the data while processing. Looping over the *Nodes*, *Ways*, and *Relations*, the generation of the 3D assets are created. We utilise the spatial scene-graph already stated, and create a *PVSNode* structure which is used to create a separate scene-graph. Each *PVSNode* contains everything needed to be saved to XML and Binary to be used within the runtime pipeline. These additions consist of 3D Model data, shader material, translation, orientation, scale, OSM tag information, a string name, a string pathname, and its parent name and pathname.

The model data, translation, orientation, and scale are self-explanatory. The 3D model object can be nullable which turns the *PVSNode* into a place holder style node, or as we wish to use it, as a classification node to organise specific objects by spatial locations, or classifications (objects in node SJ3080, building object, specific points in space, and many others).

The name and pathname are used for the generation of the runtime-scenegraph. It also stores its parents name and pathname for this reason. This will be explained in a later section.

The OSM tag object is a nullable type. Each object we classify from OSM will have a corresponding OSM tag object. These will be for Buildings, Highways, Amenities, and Boundaries. Each tag will have data pertinent to each of the objects. Each tag will have a default parameter settings list.

[3] http://www.citygml.org/

[4] http://www.web3d.org/x3d/what-x3d

[5] https://www.doogal.co.uk/dotnetcoords.php

We state that domain experts should be consulted to improve these default parameters. Additional processes infer data of OSM to improve these OSM tag parameters. Inferred data can be the height of a building by estimating the height of the average floor to a building and multiplying this by the number of floors of the building.

The 3D model object can either be a converted FBX object or a procedurally generated asset derived from the data-sets already stated. Parsed from an FBX model object, the processor will extract the shader parameters needed; diffuse, emissive, specular, texture and others. If data is not available, references to default textures and parameter values are set. The most important parameters which is set is the Shader Index value. This value is used for selecting the HLSL shader vertex function, and pixel function used to render the object. Generating this data within the pre-processor allows for interesting classifications and assigning of a particular shader index value to each of the models, or groups of models. This is the essence of the use of the UberShader. This value is stored within the model object, and to modify the model, changing this shader index value can easily change the visualisation of a scene by passing the value through the scene-graph structure, setting each model of each node.

After processing, serialisation takes place. The serialisation converts class object into an XML file which is saved to disk. At this stage the object is not saved as binary. To convert the XML file to binary, additional processing is needed. The use of XNA was chosen for this reason. The conversion of XML to Binary is done through a custom processor which is triggered when the solution is compiled. This means no objects are compiled when the application is run, thusly increased loading speeds are achieved even for large city visualisations.

One thing we have yet to discuss is the categorisation of the OSM branches. As stated within the previous section, the OSM categories partition the generated assets within individual branches of the scene-graph. The scene-graph is depicted within Figure 1.

Within the next section we discuss the runtime scenegraph structure which is very much similar to that of the pre-processor but additional data parameters are included.

D. Runtime system

The runtime scene-graph is very similar to that used within the pre-processor. It is built up of 2 class objects; PVSSpatial, and PVSNode. We stated earlier within the paper the decision to store the names of the node, and a reference to its parent's name, and not the parent object itself. The recursive nature of scene-graphs create an infinite loop when saving an object to disk or loading from disk. Saving a node with a reference to its parents node, will call for that parent node to also save. It is easier to store a name to other nodes, and have the file name the same as the node being saved. This has multiple advantages. When selecting an area to load within the runtime simulation, the node selected can be 5 layers deep within the scene-graph; node SJ3585 from Figure 1 for example. The loading process can load its parents, and load its children simultaneously; loading to the base of the scene-graph, and loading to the leaf of the scene-graph respectively. This loads a scene-graph in a middle out procedure. Additional checks are needed to make sure the nodes have not been loaded already.

The PVSSpatial object contains the spatial parameters needed to render a possible model object. We state possible because the node may not contain a model mesh object, thusly turning the node into a placeholder node. Placeholder nodes are used for pruning particular branches, and search optimisations, as well as spatial partitions, and scene manipulations. Scene manipulation can be translation to a parent node of a section, i.e. the buildings node, to move all buildings within that branch. This is achieved because all PVSSpatial node types are updated and rendered in association with their parent's world translation, orientation, and scale. A world matrix structure contains the spatial parameters in an optimised structure. If a parent's world matrix is updated, it will update and move the child world matrix. The PVSNode is inherited from the PVSSpatial node, but it extends the functionality of the PVSSpatial with the addition of a list of PSVSpatials which are the children of node, as well as a rendering function and an update function.

The PVSSpatial object contains many derivable and over ridable functions, and properties used for optimisation techniques. These functions are helper functions which are applied to single nodes or can be applied to the scene-graph as a whole. These functions are used for passing data through the scene-graph as well as applying search techniques to the scene-graph. We stipulate that the passing of information should be applied from a root node. With each node object containing these functions, any node of a scene-graph can act as a root node, because the algorithms and search techniques are applied to the node in question, and its children in a bottom to top recursive nature.

Within the render function, responsible for rendering the 3D model, systematic checks are carried out. Utilising the properties of the object, we check if its '*IsWorldTranslationCorrect'* is true or false. This property is set during the update function. If its parent or any of its recursive parents have been updated in terms of translation, orientation, or scale, then the *World* matrix of the node in question is updated. When this is complete, the lighting parameter held within the effect file are updated. To complete the rendering the graphics state of the node is updated. This is due to each node being able to set the graphics state of the graphic card. Different rendering effects applied to objects need the graphics card to be in a particular rendering state; opaque or translucent etc. This option greatly improves the flexibility and scalability of the scenegraph structure.

A user can generate their own rendering technique in the effect file code with custom vertex and pixel functions, and simply update the shader index to a node of the scene-graph or to the whole scene-graph. The only thing that needs to be updated is the effect file which can be done by domain experts. The framework does not need to be recompiled, only the effect file needs to be converted to binary and added at pre-processing stage or runtime. The combination of the custom scene-graph structure designed for the organisation and categorisation of big-geospatial-data-sets with the pre-processing of data for the procedural generation of assets, and advanced rendering capabilities proves a valuable structure to have for this framework and also for other domains; computer games, serious games, and visualisations.

III. Conclusion

We have discussed the design of a novel scene-graph structure developed to contain the assets and categorizations needed for a modern GIS with scalability and flexibility. The pre-processor utilized two similar scene-graph structures. Firstly the OS spatial partition scene-graph used for easy lookup and the procedural generation of bounding boxes generated from the *OSNode* names and used to cauterize the spatial locations of assets. The second utilizes the same scene-graph as the base but is augmented to contain model mesh data and OSM tag data. This scene-graph is saved as XML and Binary to be used within the runtime scene-graph structure. The runtime scene-graph would load the saved PVSNode structures to be used for real-time search techniques and real-time visualizations. The scene-graph can scale to contain small amounts of data, to the visualization and origination of a countries worth of data. The flexibility of the scene-graph structure allows dispersal of the UberShader index value to dynamically change the visual appearance of the simulation within real-time. The scene-graph structure structures the OSM data, and can be extended for future work.

To evaluate the scene-graph, iterative depth-first-search (DFS) and iterative breadth-first-search (BFS) [13] is used to search for a node with the name of ‘Saint George’s Hall’. We use iterative instead of recursive due to its increased processing speed through reduction of memory overhead. Results are shown in

Node Count in Scene-Graph	DFS Seconds	BFS Seconds
815	0.001399	0.00645
1629	0.0002394	0.014573
2449	0.0004203	0.016436
3257	0.000186	0.000186

Table 1 Time to search for a node using DFS and BFS.

Categorising nodes by types and then searching proves slower than searching a tree with either DFS or BFS; taking 0.12 seconds to return the same node.

IV. Future Work

Our future work consists of multiple aims. Analyse the speed of the scene-graph structure by implementing multiple search techniques. Optimizations can be achieved by changing and processing the internal data structures; optimizations such as using Array datatypes instead of List data types. As stated within the paper, domain experts should be contacted to query the default for the type of OSM tag defaults for buildings, highways and the like.

V. References

[1] J. Isaacs, “Immersive and non immersive 3D virtual city: decision support tool for urban sustainability,” *... Constr. Vol ...*, vol. 16, no. January, pp. 149–159, 2011.

[2] R. a. Falconer, J. Isaacs, D. J. Blackwood, and D. Gilmour, “Enhancing urban sustainability using 3D visualisation,” *Proc. ICE - Urban Des. Plan.*, vol. 164, no. 2002, Jun. 2011.

[3] F. Leberl, A. Irschara, T. Pock, P. Meixner, M. Gruber, S. Scholz, and A. Wiechert, “Point Clouds Lidar versus 3D Vision,” *Photogramm. Eng. Remote Sens.*, vol. 76, no. 10, pp. 1123–1134, 2010.

[4] A. Montoya, B. Vandeportaele, S. Lacroix, and G. Hattenberger, “Flight autonomy of micro-drone in indoor environments using lidar flash camera,” *IMAV 2010, Int. Micro Air Veh. Conf. Flight Compet.*, 2010.

[5] G. Kelly and H. McCabe, “Citygen: An interactive system for procedural city generation,” *Fifth Int. Conf. ...*, 2007.

[6] C. Twigg, “Catmull-Rom splines,” *Computer (Long. Beach. Calif).*, pp. 4–6, 2003.

[7] D. Tully, A. El Rhalibi, M. Merabti, Y. Shen, and C. Carter, “Game Based Decision Support System and Visualisation for Crisis Management and Response,” in *The 15th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*, 2014.

[8] D. Tully, A. El Rhalibi, C. Carter, and S. Sudirman, “Hybrid 3D Rendering of Large Map Data for Crisis Management,” *ISPRS Int. J. Geo-Information*, vol. 4, pp. 1033–1054, 2015.

[9] Z. P. William Hurst, Graham Davis, Abdennour El Rhalibi, David Tully, “Predicting and Visualising City Noise Levels to Support Tinnitus Sufferers,” in *8th International Conference on Image and Graphics (organised by Microsoft Research)*, 2015.

[10] A. Cardona, A. Hansen, J. Togelius, and M. Gustafsson, “Open Trumps, a Data Game,” *fdg2014.org*.

[11] J. Togelius and M. G. Friberger, “Bar Chart Ball, a Data Game,” *Proc. 8th Int. Conf. Found. Digit. Games (FDG 2013) fdg2013.org*, pp. 451–452, 2013.

[12] B. Mao, “Visualisation and generalisation of 3D City Models,” 2010.

[13] N. P. Russel Stuart, *Artificial Intelligence: A Modern Approach, 3rd Edition*. Prentice Hill, 2009.