

Article

Utilizing A Game Engine for Interactive 3D Topographic Data Visualization

Dany Laksono and Trias Aditya *

Department of Geodetic Engineering, Faculty of Engineering, Universitas Gadjah Mada (UGM),
Yogyakarta 55284, Indonesia

* Correspondence: triasaditya@ugm.ac.id

Received: 30 May 2019; Accepted: 25 July 2019; Published: 15 August 2019

Abstract: Developers have long used game engines for visualizing virtual worlds for players to explore. However, using real-world data in a game engine is always a challenging task, since most game engines have very little support for geospatial data. This paper presents our findings from exploring the Unity3D game engine for visualizing large-scale topographic data from mixed sources of terrestrial laser scanner models and topographic map data. Level of detail (LOD) 3 3D models of two buildings of the Universitas Gadjah Mada campus were obtained using a terrestrial laser scanner converted into the FBX format. Mapbox for Unity was used to provide georeferencing support for the 3D model. Unity3D also used road and place name layers via Mapbox for Unity based on OpenStreetMap (OSM) data. LOD1 buildings were modeled from topographic map data using Mapbox, and 3D models from the terrestrial laser scanner replaced two of these buildings. Building information and attributes, as well as visual appearances, were added to 3D features. The Unity3D game engine provides a rich set of libraries and assets for user interactions, and custom C# scripts were used to provide a bird's-eye-view mode of 3D zoom, pan, and orbital display. In addition to basic 3D navigation tools, a first-person view of the scene was utilized to enable users to gain a walk-through experience while virtually inspecting the objects on the ground. For a fly-through experience, a drone view was offered to help users inspect objects from the air. The result was a multiplatform 3D visualization capable of displaying 3D models in LOD3, as well as providing user interfaces for exploring the scene using “on the ground” and “from the air” types of first person view interactions. Using the Unity3D game engine to visualize mixed sources of topographic data creates many opportunities to optimize large-scale topographic data use.

Keywords: LOD3; first-person view; drone view; Unity3D; game engine; Mapbox

1. Introduction

Three-dimensional representations of the real-world on computers and mobile phones have been widely used in both virtual globes and games. Virtual globes, which include a wide range of graphical computer applications presenting the earth's surface as a globe (e.g., Google Earth [1,2]) and open-source application programming interfaces (APIs) known as World Wind [3] and Cesium [4] were launched initially to enable 3D geospatial data dissemination and visualization [5]. Since their introduction, virtual globes have opened up many possibilities for experts and users to exchange and disseminate 3D world data for many application domains (e.g., environmental protection and disaster response [6]) and as a platform for collaboration [7,8]. In contrast, 3D representations of earth in games were initially used to provide realistic scenes, but advanced developments in computer game engines have allowed developers to use them as a multiplatform for creating interactive visualizations [9]. Recent literature has demonstrated the use of game engines for visualizing geospatial data for many applications, such as building information modeling and archaeology [10–12].

Compared with virtual globes, game engines provide more functions to enrich user interactions and experiences. An important built-in feature of game engines is the first-person view (FPV), which is the ability of the user of a video game or a drone to see from a particular visual perspective other than one's actual location. FPV enables game engine users to control their movement and even move skillfully or perform a set of extreme actions such as rolling and acrobatic movements while interacting with real-world models. Game engines' support of various 3D data formats also affords the use of diverse topographic data. This potential should be realized as high user interaction with large-scale topographic databases derived from different data sources, which can be useful for supporting human analytical reasoning and communication as shown, for example, in Ref. [6] for crisis management and Ref. [13,14] for interactive geodesign. The challenge that was addressed in this work was the utilization of mixed sources of large-scale topographic data both from regular (e.g., OpenStreetMap) and professional (e.g., terrestrial laser scanner data) users into a game engine for more meaningful 3D visualizations.

Game engines have long been known to provide a user experience similar to real-world situations. Serious games, that is, games used for purposes other than pure entertainment, have been used in different cases for education, simulation, and game-based learning [15], examples of which include disaster education [16], military simulations, city visualizations, and various other applications. Gamification strategies have been employed for developing game applications with serious purposes, such as data collection and information dissemination for environmental protection [17].

Games are an effective way to provide simulated environments to players and, thus, are an advantageous media for visualization. A game engine, as a platform for developing games, could be used to provide user experiences for exploring virtual or ancient cities. Game developers could also utilize the ability of game engines to design user interactions strategy for the purpose of visualizing data. With these advantages, game engines can be used to create real-world models. Previous works have utilized game engines to visualize real-world terrain [18], create a 3D city model [12], or simulate an environment [19]. However, most geovisualizations using game engines lack the ability to provide georeferenced models and, thus, do not serve as an ideal platform for visualizing topographic data.

The objective of this paper is to present our findings from exploring the Unity3D game engine for visualizing large-scale topographic data of the Universitas Gadjah Mada (UGM) campus from mixed sources of terrestrial laser scanner models and topographic map data. All campus building footprints, campus roads, and terrains were taken from a topographic map at a scale of 1:2500 and were represented as level of detail (LOD) 1 models. A digital topographic dataset was created from terrestrial survey activities done by the UGM office. Two LOD3 models of campus buildings were generated from laser scanner data. Building information and attributes, as well as visual appearance, were added to 3D features. User interactions and environment simulations were built utilizing the Unity3D game engine's scripting.

Section 2 presents related work on the use of game engines for visualizing geographic data and providing user interactions. Section 3 provides the methods used to develop 3D visualizations of topographic data. Section 4 presents the results from the application development, and Section 5 discusses the lessons learned regarding the 3D map visualizations and user interactions.

2. Related Works

2.1. 3D Geospatial Data Visualization

Meaningful 3D geospatial visualization of large volumes of data has challenged map producers and cartographers to create usable visualization methods. Three-dimensional interactivity is an essential user experience that has become a standard feature for most 3D visualization platforms (e.g., Google Earth and Cesium), such as 3D zoom, pan, and orbital display, that were commonly applied utilizing the WebGL framework. Animation and motion effects were also added to 3D visualization platforms [20]. Three-dimensional geospatial data visualization can be delivered by map producers to a large audience in more captivating and intuitive ways using an interactive storytelling approach

[21]. User interactions, animation, and motion effects, first person view capability, and possibilities of for use as a platform for interactive storytelling 3D visualization can be realized using game engines.

2.2. Unity Game Engine

Research into the development of geospatial-related applications based on a game engine has seen growth across multiple disciplines in recent years [12,22–25]. Use of game engines as a platform for 3D city visualization as alternatives to 3D Geographic Information Systems/Computer-Aided Design (GIS/CAD) systems and virtual globes in the six largest cities in Finland have been assessed by Virtanen et al. [26]. Powerful 3D city visualization and interactions that offer “immersive experiences” (e.g., Virtual Reality/Augmented Reality (VR/AR) applications) or user engagement (e.g., applications for public participation) on multiple computing platforms (e.g., mobile, desktop, VR/AR etc.) have benefitted greatly from game engine technology. The Unity3D (www.unity3d.com) is one of the well-known multiplatform game engines for building 2D and 3D games and simulations [27]. Unity could also be used to provide VR and AR games on platforms such as consoles or handheld mobile devices using a single codebase. Unity supports C# as its default language for scripting game logic, including user interactions, Non-Playable Characters (NPCs), artificial intelligence, and game environments. Game design in Unity could also utilize “Assets”, i.e., game components developed by other game developers.

As a game engine, Unity3D could be used to develop a simulation of a real-world environment. Geospatial data obtained from real-world measurement could be converted to Unity-ready 3D formats for visualization in the Unity3D game engine. Previous works such as Ref. [28] have provided use cases on using geospatial data (i.e., terrain data from LiDAR and 3D buildings) for visualization in Unity. However, this approach is lacking in terms of real-world coordinates, making it more difficult to integrate different geo-related data from multiple sources.

Mapbox Unity SDK (www.mapbox.com/unity) is an extension of the Unity3D game engine capable of handling geospatial data in a Unity3D environment. Mapbox enables Unity to provide “slippy map” tiles from a Mapbox server for underlying terrain in setting the environment in Unity3D. Thus, any georeferenced data could be overlain on Unity based on its real-world coordinates and integrated into a single game environment together with other game elements such as players, NPCs, and 3D virtual environments. Based on this rationale, it is possible to utilize the Mapbox Unity SDK for developing 3D city simulations based on geospatial data obtained from multiple sources. This paper presents our findings in developing 3D visualization with the Unity Game Engine based on topographic map data of the UGM campus in Yogyakarta. Mapbox for Unity SDK was used to provide a platform for integrating the data, and user interactions were developed using Unity C# scripting capability.

3. Methods

3.1. Data Integration

The dataset utilized in this research was a combination of a campus topographic map at a 1:2500 scale, created from terrestrial survey activities, and two LOD3 3D models representing two UGM buildings obtained from terrestrial laser scanner (TLS) measurements. Unity3D only accepts several 3D formats to be consumed as a game object, e.g., OBJ and FBX. Thus, in order to import 3D models and topographic maps from the GIS format, it was necessary to conduct data editing and conversion prior to development in the Unity Game Engine. The data preparation workflow for Unity3D development is shown in Figure 1.

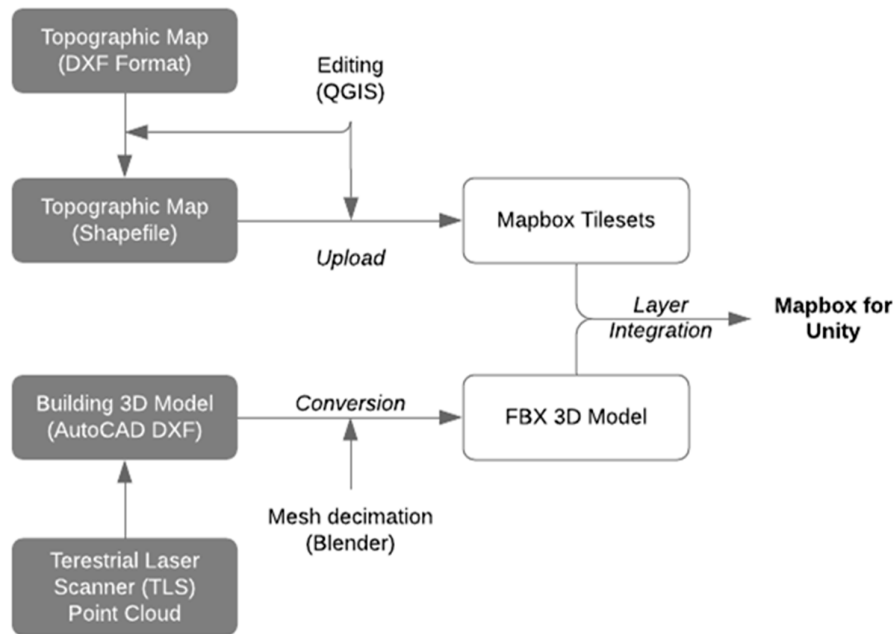


Figure 1. Data preparation workflow for Mapbox with Unity.

Three-dimensional models of two buildings on the campus were obtained from two different TLS measurements. The first building modeled, The Department of Geodetic Engineering Building created from TLS measurements using the Topcon GLS-2000 instrument, resulted in 170,951,903 points with a derived registration accuracy of 0.003 meters. The second building model, the University's Main Building, surveyed using a Faro Focus X33 instrument, resulted in 73,594,371 points with a registration accuracy of 0.015 meters. The first building had more points as the survey measured all corridors and interior rooms. However, the corresponding model did not consider all corridors and rooms.

Topographic data was obtained as AutoCAD DXF data, which was converted to shapefile for editing in QGIS open-source software. Mapbox for Unity accepts only the Tileset format stored in the Mapbox Cloud; thus, each layer was edited separately, and additional attributes such as height and name were aggregated into each layer. QGIS was used for editing topographic data from the DXF format to shapefile and also for editing its attributes (Figure 2). Resulting layers from 2D topographic data were (1) buildings, (2) roads, (3) parks, (4) city forests, (5) fields, and (6) contours. These layers were further converted to the Mapbox Tileset format, resulting in each layer's Map ID ingestion into Mapbox for Unity. Tileset was the data format used in Mapbox Studio to serve raster or vector layers that were divided into several tiles for each zoom level. Mapbox for Unity consumed these layers and translated each layer into a Unity Game Component.

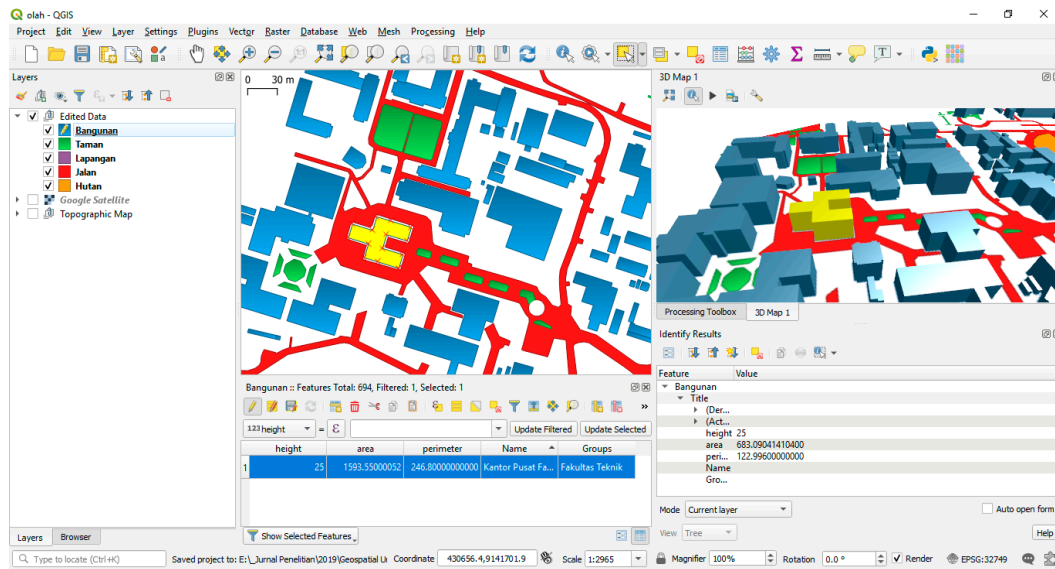


Figure 2. Editing topographic data in QGIS.

Three-dimensional models were converted to AutoCAD 3D DXF format. For this 3D file to be available in Unity, conversion to a Unity-ready format was performed. Blender (Figure 3) was utilized to convert the 3D model into FBX format. Further processing in Blender was required to simplify the model. The Blender Decimation operation was used to reduce the number of faces in the 3D model for visualizing in Unity. Decimation in Blender works by collapsing the model, i.e., by reducing the number of vertices. For this conversion, a decimation ratio of 0.5 was used to reduce the number of vertices from 497.654 to 284.802 and faces from 848.392 to 424.196. The resulting 3D model was then imported into Unity as a game component, which was processed using Mapbox for Unity and overlain with topographic data from Mapbox Tilesets.

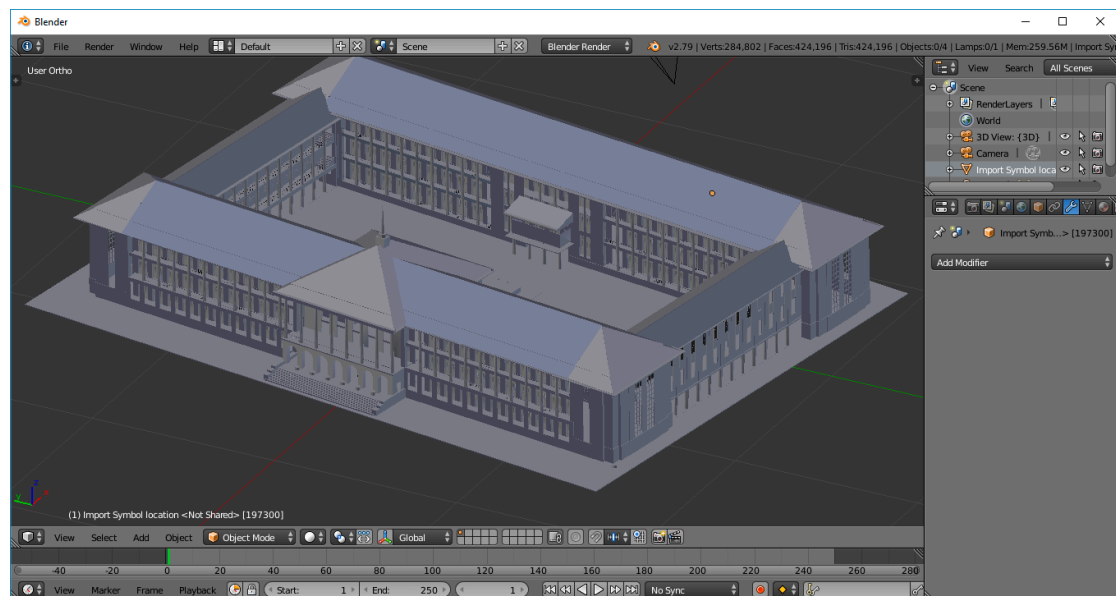


Figure 3. Blender for 3D model conversion.

3.2. Visual Appearances and Attributes Information

With topographic data and the 3D model ready, Mapbox for Unity was used to overlay both datasets into Unity3D. Mapbox for Unity used three different layers as additional game components

in Unity3D. The three layers composed the basic structure of Mapbox Unity3D world with additional data and custom scripts (i.e., “modifiers”) overlain on top of them. The three layers were Imagery Layer, Terrain layer, and Vector layer. Imagery and Terrain layers were provided by Mapbox slippy maps (i.e., Mapbox Dark) and Mapbox Terrain. OpenStreetMap tile was used in the Mapbox slippy maps. Vector layers were composed of Mapbox Tilesets that were obtained from the 2D topographic map. Map layer abstraction setup as configured in Mapbox for Unity is shown in Figure 4.

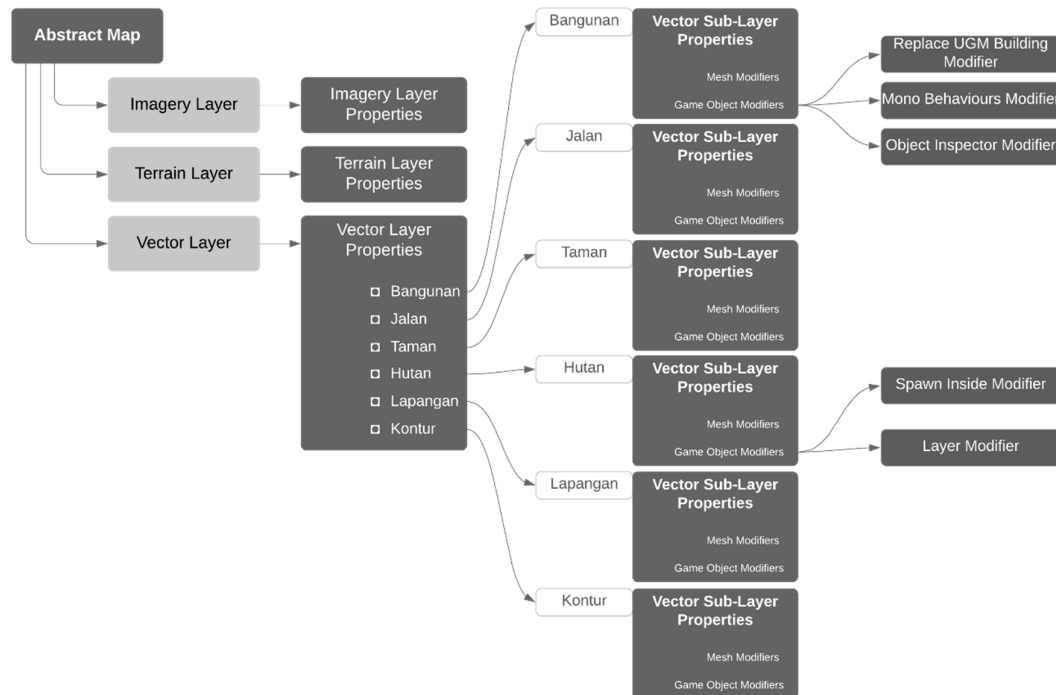


Figure 4. Map abstraction for Mapbox Unity3D.

Each vector layer from Mapbox Tileset was configured with different symbols and properties. The building layer (called “Bangunan” in Figure 4), for example, was configured based on its height attribute, while top and side textures of the buildings were customized from Mapbox’s provided textures. Three-dimensional models from this layer were obtained as an extrusion from the terrain elevation. Other layers were given symbols based on textures available on Unity. Table 1 provides the different configurations and symbols applied for each layer in Unity3D. Roads, parks, city forests, fields, and contours in Table 1 were called “Jalan”, “Taman”, “Hutanj”, “Lapangan”, and “Kontur” in Figure 4, respectively.

Table 1. Configurations for each Mapbox Tileset layer.

Layer	Texture	Custom Modifiers
Buildings ¹	Mapbox top and side material	Highlight building on mouse-over Display building’s attributes Replace building with 3D model Set colliders
Roads	Mapbox light color with transparency	Remove colliders
Parks	Custom color (light green)	Remove colliders
City Forests	Custom color (green)	Spawn 3D tree model Set collider for trees
Fields	Custom color (brown)	Remove colliders
Contours	Custom color (light brown)	Remove colliders

¹ 3D models from the TLS replaced the extruded model in building layer.

Custom C# scripts (i.e., “modifiers”) were applied for each layer in the Mapbox map to provide custom interactions and experiences of the game components in the layer. For instance, modifiers were used to setup colliders, i.e., a condition wherein game players should collide instead of walking through an object. Colliders were set for building and city forest layers, while for roads, parks, fields, and contours, colliders were removed. Modifiers could also be used to set up custom scripts for certain conditions in the layer, such as spreading the 3D tree model in the forest area based on the city forest layer. Environments were also set utilizing Unity’s functionality, e.g., Skybox [29] for providing a sky atmosphere.

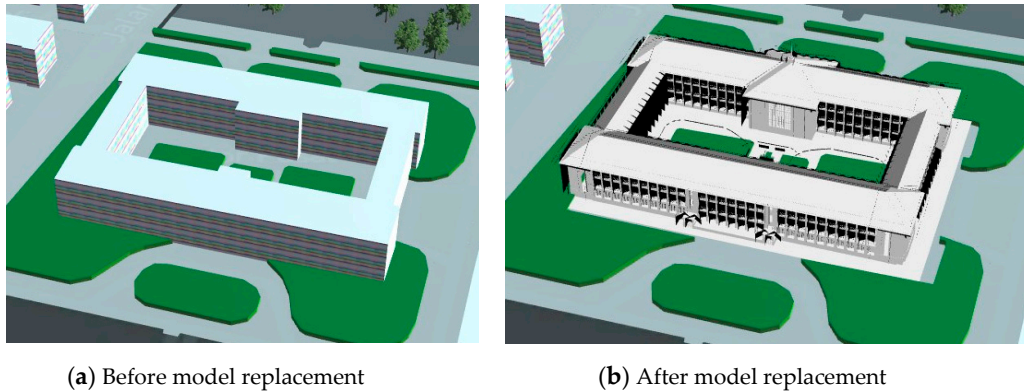


Figure 5. Replace model modifiers: before replacement (a) and after replacement (b).

One of the most important uses of Modifier is to place the LOD3 3D model obtained from TLS into its real-world position. The FBX model was rescaled and rotated to match its real-world condition, while Mapbox’s Replace Feature Modifier was used to place the model into the correct position based on its corresponding extruded 3D model from the topographic map. Figure 5 shows the 3D model in the building layer before and after model replacement using Replace Feature Modifier. The LOD3 model from TLS could be positioned correctly and overlaid with other topographic data from Mapbox Tileset.

3.3. Map Interactions

With all the data in place, user interface and interaction logic were built into the Unity Game Engine. Custom scripts using C# were utilized for camera navigation, player movements, and user interface. Unity provides a set of basic navigations for setting camera position dynamically. However, custom scripts were developed to better suit navigation in the 3D map environment. Camera navigations were divided into the following:

- **Bird Eye View.** Users control the camera movements on the map, i.e., panning, rotating/tilting, look-around, and zoom in/out.
- **FPV.** Camera is attached to the player, where users control their movement freely.
- **Drone View.** The user controls a quadcopter drone to explore the scene.

Bird Eye View navigation was obtained using the raycasting method, where camera position was deduced based on intersecting rays and the ground surface [30]. Different functions were assigned to each button in a mouse for providing user interactions. Figure 6 shows functions assigned to each mouse button on the custom C# script for Bird Eye View.

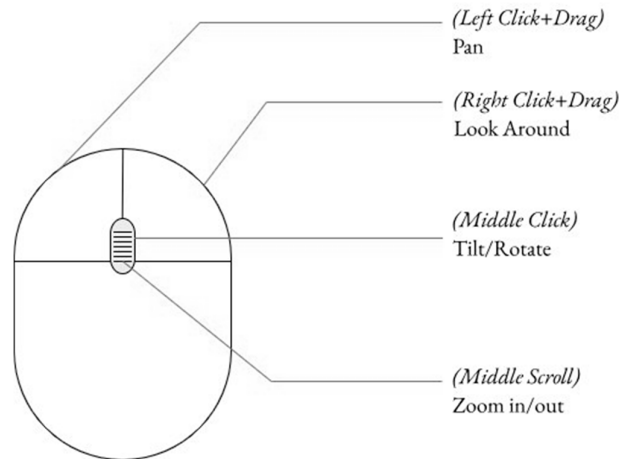


Figure 6. Mouse interactions built using a custom C# script for Bird Eye View navigation.

FPV utilized the Unity Standard Asset for the first-person shooter (FPS) controller [31]. This asset comprises game components to deliver user navigation strategies, including a FPV. The first-person controller used keyboard (W-A-S-D, shift, and arrow buttons) for controlling the movements of the player, while the mouse pointer was used for the viewing direction of the camera. A similar configuration was used for the drone controller.

User interface was provided using buttons to access each function. The menu consisted of buttons to toggle Minimap, switch to FPV, Explore by Drone, Reset Camera Position, and show the “help” menu. The Minimap was built using C# scripting to provide visual cues as a top-down-looking map to assist in navigating the scene during Bird Eye, First-Person, or Drone View. The menu was built using Unity UI components and was controlled using C# scripts for their interactions. Figure 7 shows the logical flow designed for the application interface.

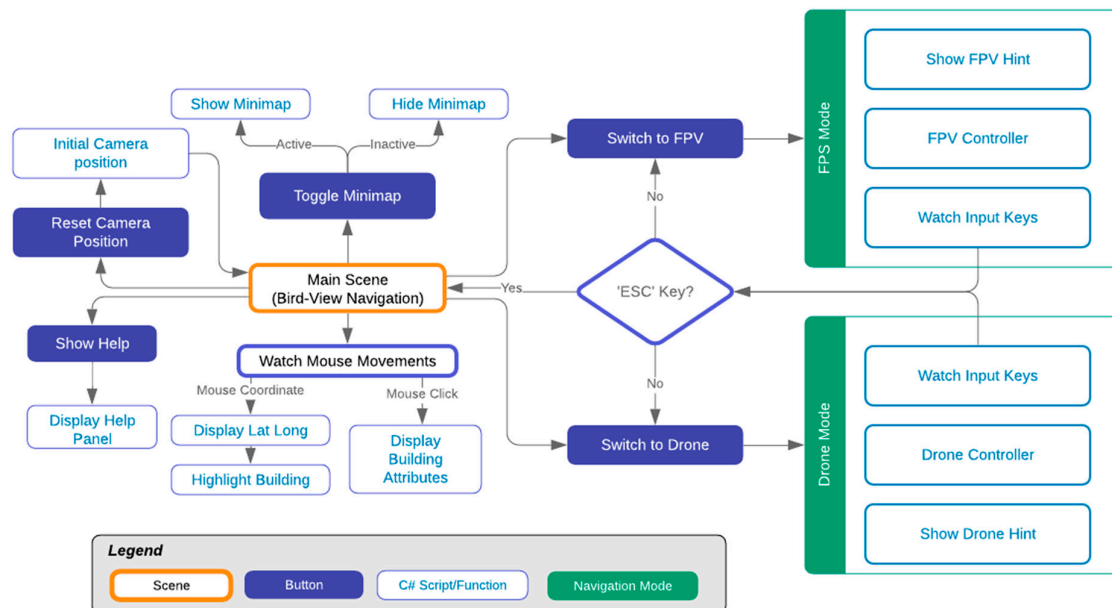


Figure 7. Logical flow of the application interface.

Several scripts combined with game components and assets were used to control the user interface. The C# scripts captured user interaction and provided functions related to user input. These scripts watched mouse movements, displayed information, converted the user screen to coordinates,

showed/disabled interface components, and loaded attribute information based on user interaction. Interactions between the scripts and game components allowed the interactions to be designed for adaptation to various visualizations required for 3D exploration of spatial data. Figure 8 shows an example of a script used for handling mouse interactions for panning, tilting, zooming, and rotating the camera in the Bird Eye View navigation (See Appendix B).

```

26 //tilting variables
27 private Vector3 pivot;
28 private Vector3 rotationAxis;
29 private float y_rotate;
30 private float x_rotate;
31 public float rotationSpeed = 10f;
32
33 // Update is called once per frame
34 void Update () {
35     // --GETTING INPUT--
36     // Left mouse button: Panning
37     if (Input.GetMouseButtonDown (0)) {
38         touchStart = mouseWorldPosition (groundZ);
39         isPanning = true;
40     };
41     if (Input.GetMouseButton (0)) {
42         Vector3 direction = touchStart - mouseWorldPosition (groundZ);
43         cam.transform.position += direction;
44     }
45
46     // Right mouse button: Rotating
47     if (Input.GetMouseButtonDown (1)) {
48         Vector3 angles = cam.transform.eulerAngles;
49         yaw = angles.y;
50         pitch = angles.x;
51         isRotating = true;
52     }
53
54     // Middle mouse button: Tilting

```

Figure 8. Sample C# script for handling mouse interactions.

4. Results

The online version of the interactive 3D Viewer could be accessed by users from the resulting website (please see supplementary materials). The web version was obtained from Unity as built into the WebGL platform, which was further deployed to an Apache webserver. The multiplatform nature of Unity3D was used to build different versions on a Windows desktop computer and Android using the same codebase. The WebGL version was the easiest to disseminate to users since no requirements other than web browsers were needed to open the 3D visualization. The WebGL version worked inside the web browser, providing user interactions built into Unity3D. Users navigate the scene using a mouse and keyboard on the ground with FPV and from the air with Bird Eye View (i.e., defined as FPS or drone mode in the interface). Mapbox for Unity provided slippy maps and layers to integrate topographic data and a 3D model from a TLS into a game environment. Bird Eye View and FPV could be used to navigate the scene, using a mouse and keyboard for interactions.

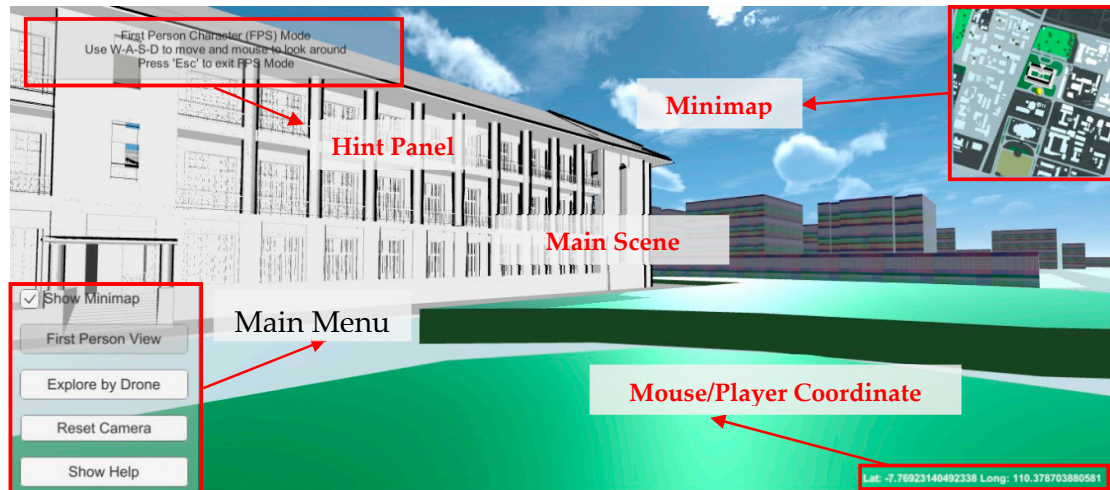


Figure 9. Main interface of the 3D geovisualization application.

Figure 9 shows the main interface for interacting with the 3D scene. The main menu is shown in the bottom-left part of the screen, showing buttons for interacting and controlling the camera to explore the scene. The main scene adapts to the user's selection of navigation mode, whether Bird Eye View, First Person View, or Drone Explorer. Latitudinal and longitudinal coordinates shown on the bottom-right part of the screen also adapt to each navigation mode. In Bird View mode, the coordinates represent the world coordinates of the mouse; hence, moving the mouse should give the coordinates of the mouse pointer projected to the ground plane. In First-Person and Drone Explorer view, coordinates represent those of the center of the screen; hence, moving the player or drone should give different coordinates. The Minimap in the top-right corner helps in navigating the scene by providing a top view of the current camera position, which could be especially helpful during First-Person or Drone View navigation.

First-Person character mode can be activated using the FPV button on the Unity screen. Users can use keyboard buttons W-A-S-D to move forward and backward as well as to move right and left following the space between objects. Users can gain a walk-through experience since the 3D model is representing the building at a high level of detail (LOD3). As the other campus buildings represented in this study were built from topographic LOD1 models, the FPV can only help users navigate along streets and outer parts of the buildings. Simple attributes are offered to let users know the name of the structures. As discussed in the works related to the 3D map display in comparison to the 2D map display for navigational purposes [32–34], 3D map display provides superiority in helping users' navigational processes. Navigational processes here refer to activities of self-orientation, spatial knowledge development, and navigational decision-making. The FPV mode in the Unity platform arguably helps users to better develop spatial knowledge and self-orientation. FPV mode in Unity supports a responsive, interactive 3D cartography that offers possibilities for users to navigate their movements freely (e.g., colliding or walking through objects) while inspecting the built environment models. Figure 10 shows the FPV mode used for inspecting the model of the University Main Building obtained from the TLS (see also Figures C1 and C2).

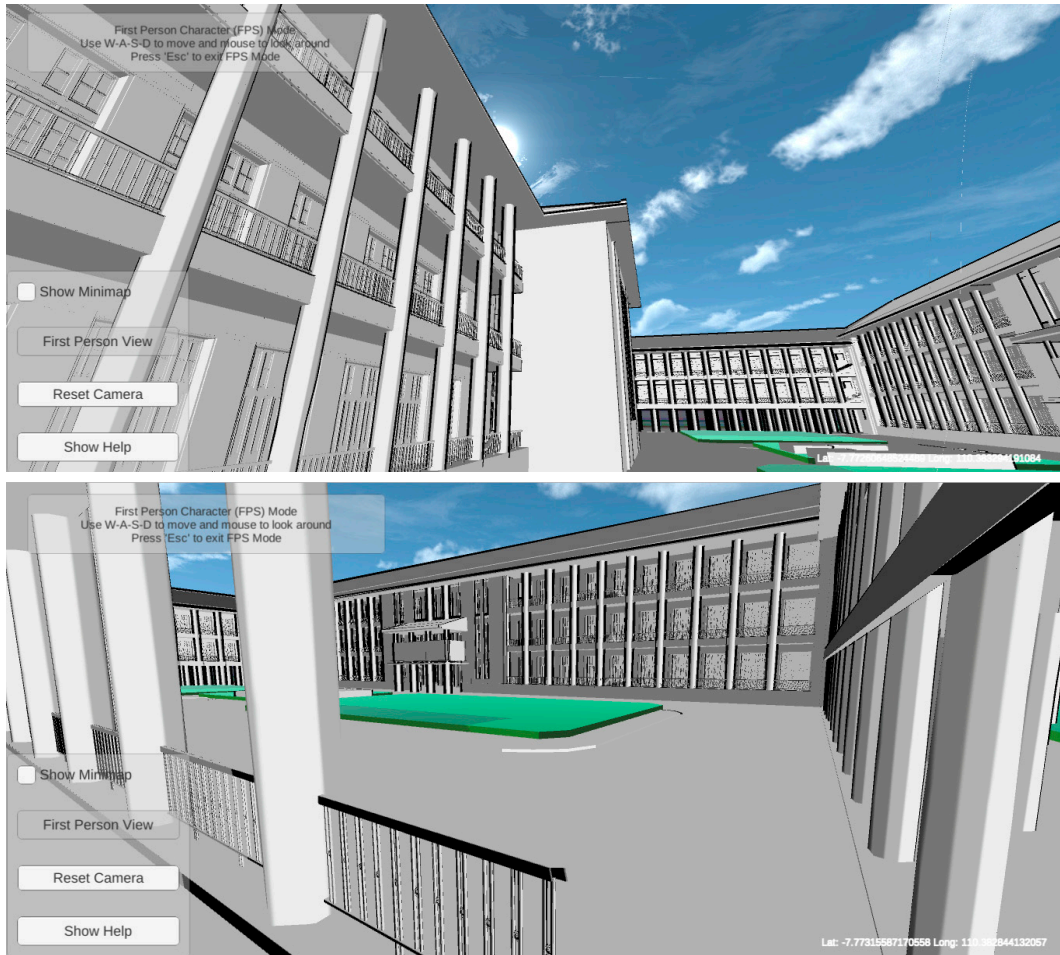


Figure 10. FPV mode for inspecting the LOD3 building.

Drone View is similar to FPV in that both navigation modes are focused on the player. This mode also shows the strength of the game engine for 3D visualization of geospatial data. The drone could be controlled to explore the scene by hovering above the buildings or performing quick inspection from the air. Utilizing the three navigational modes (i.e., Bird View, First-Person, and Drone View), users are provided with a free exploration, first-person (“walking view”) or flying view of the scene. Combined with highly detailed 3D buildings obtained from ground surveys, an interactive visualization could be built to obtain better user perception of the modeled scene either on the ground or from the air. Figure 11 shows the utilization of the drone for inspecting building attributes by flying above the ground (see also Figures C3 and C4).



Figure 11. Drone view for air-based inspection.

5. Discussion

Mapbox for Unity was used to develop an interactive 3D visualization of topographic data integrated with a 3D building from TLS. The LOD3 3D models were imported into Unity and placed according to locations of the building models in the topographic map. Previous attempts at incorporating geospatial data into Unity [19,28,35] have been restricted due to the unavailability of projection and coordinate systems in the game engine. Mapbox for Unity tackled this problem by providing a basic platform for geospatial data as tiled layers, thus enabling multiple sources of geospatial data to be overlaid based on real-world coordinates. However, for the case of presenting a 3D model from other sources, the 3D model only replaces extruded buildings from the topographic map with its rotation and scale manually set to match orientation and size of the building. This approach, while correctly placing the model location, still has the limitation of presenting the buildings to their true scale and location.

Name	Status	Size	Time	Waterfall
<input type="checkbox"/> events	200	45 B	237 ms	
<input type="checkbox"/> worldexplorer_web.json	200	844 B	297 ms	
<input type="checkbox"/> worldexplorer_web.wasm.code.unityweb	200	6.5 MB	8.86 s	
<input type="checkbox"/> worldexplorer_web.data.unityweb	200	42.7 MB	39.86 s	
<input type="checkbox"/> worldexplorer_web.wasm.framework.unityweb	200	93.4 KB	512 ms	
<input type="checkbox"/> events	200	325 B	434 ms	
<input type="checkbox"/> 34185.vector.pbf?events=false&access_token=...	200	25.0 KB	592 ms	
<input type="checkbox"/> 34183?events=false&access_token=pk.eyJ1ljo...	200	38.9 KB	589 ms	
<input type="checkbox"/> 34185?events=false&access_token=pk.eyJ1ljo...	200	49.7 KB	598 ms	
<input type="checkbox"/> 34184?events=false&access_token=pk.evJ1ljo...	200	36.1 KB	565 ms	

60 / 68 requests | 51.1 MB / 51.2 MB transferred | 51.6 MB / 52.3 MB resources | Finish: 16.2 min

Figure 12. Chrome Dev Tools analysis of the WebGL performances.

Another potential issue was the performance of 3D visualization. Though most browsers nowadays already support WebGL [20], the performance of presenting a 3D environment from Unity depended on the users' computer hardware and internet capabilities [36]. On a poor internet connection, loading time for the 3D visualization is quite significant, since most of the data, including Mapbox Tileset, was loaded from the Apache webserver or Mapbox Cloud. This was evident by looking at the loading time for each component of the web visualization. Figure 12 shows the loading time for each component generated from the Unity WebGL converter, which was obtained from Chrome Developers Tools of <https://geoinsight.ugm.ac.id/ugm3d>. From the figure, it can be seen that loading time for the Unity data component is highest compared to Unity Framework and Unity Code. The compiled Unity data contained the 3D model, which caused the initial loading time of the page to be very slow (39.86 seconds for downloading a 42.7 MB file in this case). Furthermore, it could be seen that using Mapbox Tiles (vector.pbf from the above picture) could reduce the initial loading time of the page. Currently, mesh decimation is done to the 3D model using Blender in order to increase the performance of the 3D visualization, which results in a smaller file size and faster loading time. In the future, tilings of the 3D model should also be used inside a game engine for presenting larger and more complex buildings. More investigations are needed to discover the trade-offs between detailed 3D geospatial data and visualization performance, especially on a game engine such as Unity3D.

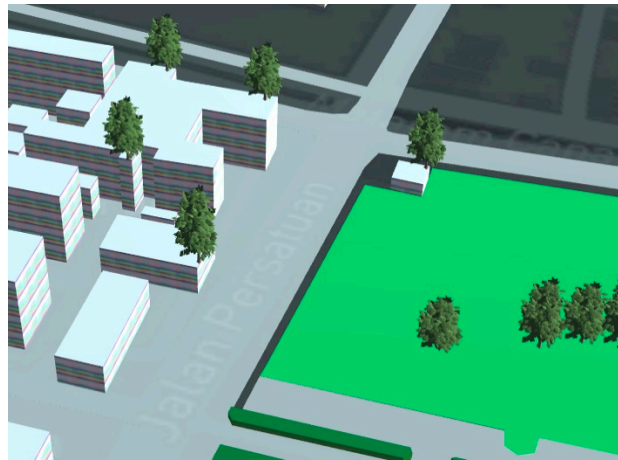


Figure 13. Misplaced 3D trees outside of forest area.

Another issue occurred when using Mapbox for Unity for spreading 3D models inside a polygonal area. 3D models of trees were spawned on top of the City Forest layer using Mapbox Spawn Inside Modifier. However, the 3D trees were placed not only inside the forest polygon but also outside of the layer (see Figure 13). Thus, in order to properly visualize 3D geospatial models, further efforts should be made by developing custom scripts on top of the currently available Mapbox for Unity functions. This also highlights the problem of integrating multiple-sourced spatial data in a game engine. This paper presents a pipeline for integrating a 3D model acquired from a TLS with 2D topographic maps where the datasets have different levels of accuracy. Different levels of accuracy apparently cause problems where spatial relationships between the datasets are considered, as pointed out by Ref. [37]. This is especially true in the case of replacing an extruded building from a topographic map with a 3D model from a TLS using Mapbox for Unity.

Further usability assessments of the implementation of FPV of 3D topographic data visualization is required to investigate the potential use of FPV and Drone View in support of more effective and engaging 3D visualizations. Potential uses of 3D topographic visualization beyond its basic ability to support navigation and orientation applications, as suggested in Ref. [21] with a game engine platform, could be further exploited to support storytelling and narrative cartography in order to “ground stories in real places” [38] or to map “real-life” stories [39].

6. Conclusions

This paper presents our findings in developing a 3D visualization system for an integrated topographic map and 3D model using a game engine. Unity3D was utilized to develop a user interface for exploring the 3D environment built from a topographic map. Mapbox for Unity was employed to enable Unity to utilize geospatial data. Layers from topographic maps were converted into Mapbox Tilesets, which were then converted to a game object inside Unity3D. A LOD3 3D model from TLS was overlaid onto the map by replacing the extruded model from the topographic map with a converted 3D model from TLS. Custom scripts based on Unity C# scripting were developed to provide a user interface for exploring the scene. The resulting 3D visualization was built into WebGL and deployed to an Apache webserver and could be accessed from <https://geoinsight.ugm.ac.id/ugm3d>.

While most game engines cannot utilize geospatial data in its entirety, the Mapbox extension for Unity could be employed to enable the presentation of geospatial data as a game object in its real-world coordinates. This approach could further benefit the incorporation of geospatial data into a game engine to simulate real-world situations. However, the current scheme for presenting 3D buildings from high density and high accuracy topographic models using Mapbox in Unity still possesses a limitation. This work presents two 3D buildings by replacing the existing 3D models that have lower resolution (such as LOD1 models of extruded building footprints) with 3D models acquired from TLS measurements. With this method, the 3D model acts as a 3D point symbol that requires manual scaling and rotation to match the real characteristics of the building. Future investigations are needed to utilize 3D geospatial data as a game object in Unity, with attention given to its performance and capabilities for handling geospatial data formats.

Supplementary Materials: The present work is available online at <http://geoinsight.ugm.ac.id/ugm3d>

Author Contributions: Conceptualization, Trias Aditya, and Dany Laksono; methodology, Dany Laksono and Trias Aditya; software development, Dany Laksono; validation, Dany Laksono and Trias Aditya; writing—original draft preparation, Dany Laksono and Trias Aditya; writing—review and editing, Dany Laksono and Trias Aditya; visualization, Dany Laksono; supervision, Trias Aditya; funding acquisition, Trias Aditya.

Funding: This research received a funding grant from The Directorate of Higher Education's Excellence Grant for Research Development No. 2761/2019

Acknowledgments: The authors would like to thank Dr. Istarno and Ruli Andaru, M.Eng., for the TLS data acquired in this research.

Conflicts of Interest: The authors declare no conflict of interest

Appendix A

Table A1. List of Abbreviations.

Abbreviations	Full text	Descriptions
API	Application Programming Interface	A set of routines which specifies how specific function of a software is being used
DXF	AutoCAD Drawing Exchange Format	A binary or an ASCII representation of a drawing file. It is often used to share drawing data between other CAD programs
FBX	Autodesk FilmBoX	A 3D asset exchange format that facilitates higher-fidelity 3D data exchange
FPV	First-Person View	An interactive visualization where the player experience virtual world through the eye of a character
LiDAR	Light Detection and Ranging	A remote sensing method that uses light in the form of a pulsed laser to measure ranges (variable distances) to the Earth
LOD	Level of Detail	A number which specifies level of 3D model representation detail [40]
SDK	Software Development Kit	A set of software creation tools to build application package based on certain framework
TLS	Terrestrial Laser Scanner	A terrestrial survey method that uses laser scanning technology to scan the object and record the 3D point-clouds, also known as topographic LiDAR
UGM	Universitas Gadjah Mada	University Campus in Yogyakarta, Indonesia

Appendix B

Bird Eye View navigation script was developed using C# in Unity. This script serves Bird View navigation using mouse and keyboard for panning, tilting, rotating, and zooming using mouse buttons.

```
// Bird View Navigation in Unity

// for Mapbox Unity 2.0

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class WorldNavigation: MonoBehaviour {

    public Camera cam;

    private bool isPanning;

    private bool isRotating;

    private bool isTilting;

    private bool isZooming;

    private Vector3 touchStart;

    public float groundZ = 0;
```

```
//rotation params

public float speedH = 2.0f;

public float speedV = 2.0f;

private float yaw;

private float pitch;

//tilting variables

private Vector3 pivot;

private Vector3 rotationAxis;

private float y_rotate;

private float x_rotate;

public float rotationSpeed = 10f;

// Update is called once per frame

void Update () {

    // --GETTING INPUT--

    // Left mouse button: Panning

    if (Input.GetMouseButtonDown (0)) {

        touchStart = mouseWorldPosition (groundZ);

        isPanning = true;

    };

    if (Input.GetMouseButton (0)) {

        Vector3 direction = touchStart - mouseWorldPosition (groundZ);

        cam.transform.position += direction;

    }

    // Right mouse button: Rotating

    if (Input.GetMouseButtonDown (1)) {

        Vector3 angles = cam.transform.eulerAngles;

        yaw = angles.y;

        pitch = angles.x;

        isRotating = true;
```



```

    }

    // Middle mouse button: Tilting

    float currentCamX = cam.transform.position.x;

    float currentCamZ = cam.transform.position.z;

    if (cam.transform.position.y < 0.5f) {

        cam.transform.position = new Vector3 (currentCamX, 0.5f, currentCamZ);

    }

    if (Input.GetMouseButtonDown (2)) {

        pivot = centerScreenPosition (groundZ);

        isTilting = true;

    }

    // Middle Mouse Scroll

    if (Input.GetAxis ("Mouse ScrollWheel") != 0) {

        isZooming = true;

    }

    // canceling all functions

    if (!Input.GetMouseButton (0)) isPanning = false;

    if (!Input.GetMouseButton (1)) isRotating = false;

    if (!Input.GetMouseButton (2)) isTilting = false;

    if (Input.GetAxis ("Mouse ScrollWheel") == 0) isZooming = false;

}

// LATE UPDATE

void LateUpdate () {

    //Right Button: Camera Rotation

    if (isRotating) {

        yaw += speedH * Input.GetAxis ("Mouse X");

```

```

        pitch -= speedV * Input.GetAxis ("Mouse Y");

        // Clamp pitch:

        pitch = Mathf.Clamp (pitch, -90f, 90f);

        Quaternion rotation = Quaternion.Euler (pitch, yaw, 0);

        cam.transform.rotation = rotation;

    }

    //middle mouse: tilting

    if (isTilting) {

        y_rotate = Input.GetAxis ("Mouse X") * rotationSpeed;

        x_rotate = Input.GetAxis ("Mouse Y") * rotationSpeed;

        //cam.transform.RotateAround (pivot, Vector3.left, rotationAngleOnX);

        OrbitCamera (cam, pivot, y_rotate, x_rotate);

    }

    // mouse scrollwheel: zooming

    if (isZooming) {

        Vector3 desiredPosition;

        desiredPosition = mouseWorldPosition (groundZ);

        float distance = Vector3.Distance (desiredPosition, transform.position);

        Vector3 direction = Vector3.Normalize (desiredPosition - transform.position) * (distance *
Input.GetAxis ("Mouse ScrollWheel"));

        transform.position += direction;

    }

}

// Raycasting Mouse Position for Panning

private Vector3 mouseWorldPosition (float z) {

    Ray mousePos = cam.ScreenPointToRay (Input.mousePosition);

    Plane ground = new Plane (Vector3.forward, new Vector3 (0, 0, z));

```

```

        float distance;

        ground.Raycast (mousePos, out distance);

        return mousePos.GetPoint (distance);
    }

    // Raycasting center of screen for Tilting
    private Vector3 centerScreenPosition (float z) {

        //Ray mousePos = cam.ScreenPointToRay(Input.mousePosition);

        Ray centerScreen = cam.ViewportPointToRay (new Vector3 (0.5f, 0.5f, 0f));

        Plane ground = new Plane (Vector3.forward, new Vector3 (0, 0, z));

        float distance;

        //ground.Raycast(mousePos, out distance);

        ground.Raycast (centerScreen, out distance);

        //return mousePos.GetPoint(distance);

        return centerScreen.GetPoint (distance);
    }

    // calculating orbit camera
    public void OrbitCamera (Camera cam, Vector3 target, float y_rotate, float x_rotate) {

        Vector3 angles = transform.eulerAngles;

        angles.z = 0;

        cam.transform.eulerAngles = angles;

        cam.transform.RotateAround (target, Vector3.up, y_rotate);

        cam.transform.RotateAround (target, Vector3.left, x_rotate);

        cam.transform.LookAt (target);
    }
}

```

Appendix C



Figure C1. Screenshot of First-Person View (FPV) navigation.

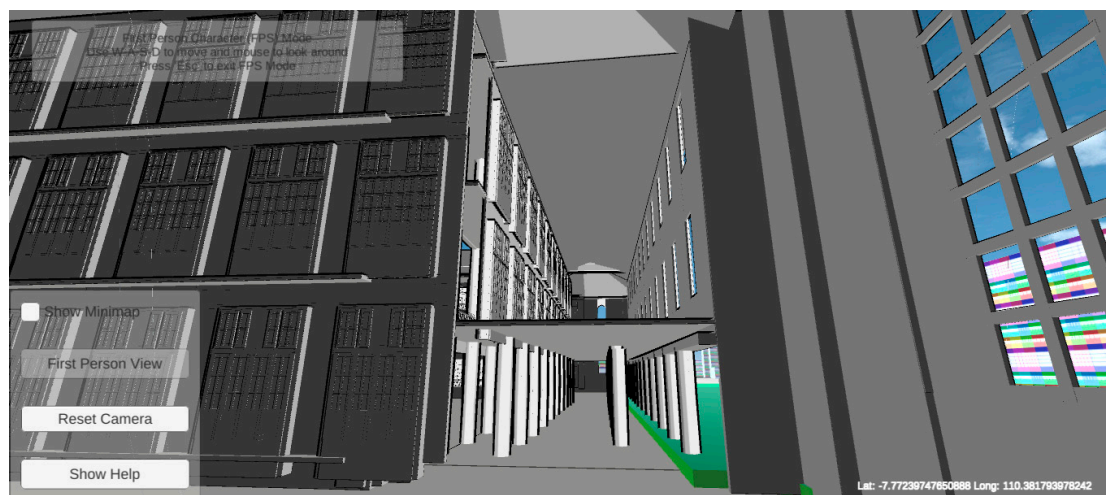


Figure C2. Screenshot of FPV navigation.

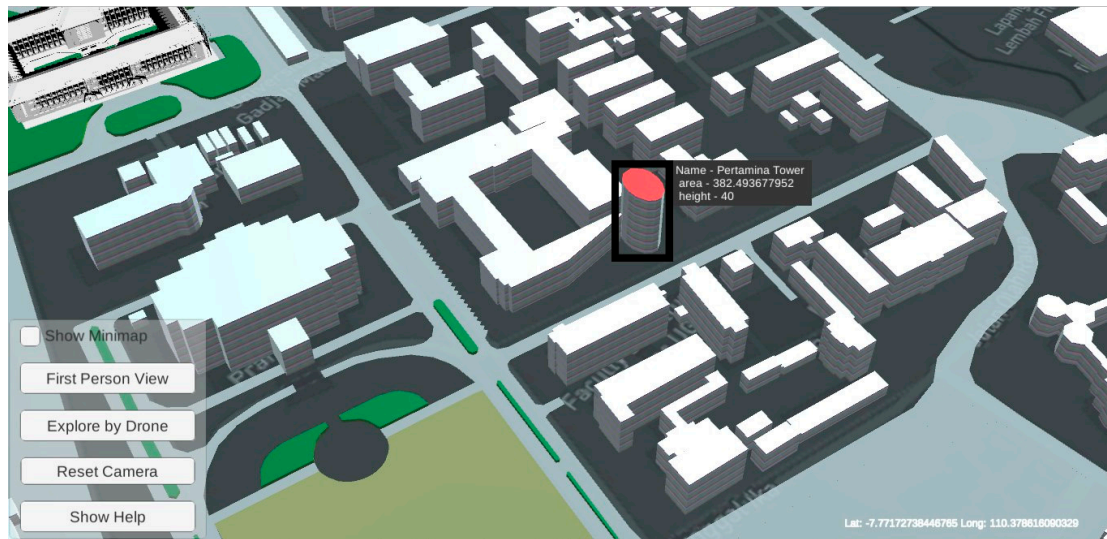


Figure C3. Screenshot of Bird Eye View navigation.

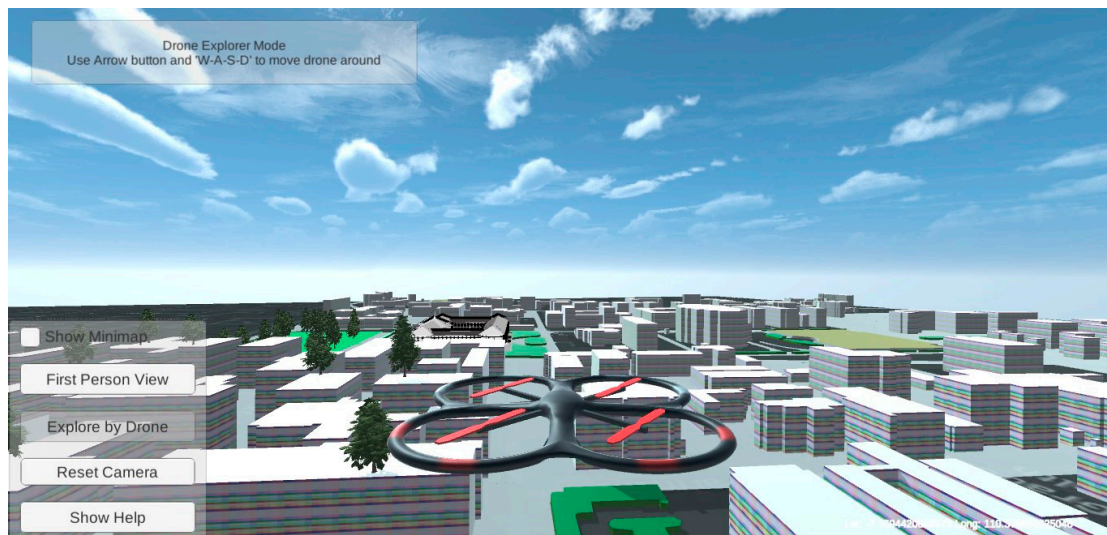


Figure C4. Screenshot of Drone View navigation.

References

1. Farman, J. Mapping the digital empire: Google Earth and the process of postmodern cartography. *New Media Soc.* **2010**, *12*, 869–888.
2. Yu, L.; Gong, P. Google Earth as a virtual globe tool for Earth science applications at the global scale: Progress and perspectives. *Int. J. Remote Sens.* **2012**, *33*, 3966–3986.
3. Boschetti, L.; Roy, D.P.; Justice, C.O. Using NASA's World Wind virtual globe for interactive internet visualization of the global MODIS burned area product. *Int. J. Remote Sens.* **2008**, *29*, 3067–3072.
4. Chaturvedi, K.; Kolbe, T.H. Dynamizers: Modeling and implementing dynamic properties for semantic 3d city models. *Proc. Eurographics Work. Urban Data Model. Vis.* **2015**, 43–48. doi:10.2312/udmv.20151348.
5. Brovelli, M.A.; Minghini, M.; Moreno-sanchez, R. Free and open source software for geospatial applications (FOSS4G) to support Future Earth. *Int. J. Digit. Earth* **2017**, *10*, 386–404.
6. Tully, D.; El Rhalibi, A.; Carter, C.; Sudirman, S.; Rhalibi, A. Hybrid 3D Rendering of Large Map Data for Crisis Management. *ISPRS Int. J. Geo Inf.* **2015**, *4*, 1033–1054.
7. Virtanen, J.; Hyypä, H.; Kämäräinen, A.; Hollström, T.; Vastaranta, M.; Hyypä, J. Intelligent Open Data 3D Maps in a Collaborative Virtual World. *ISPRS Int. J. Geo Inf.* **2015**, *4*, 837–857.

8. Lin, H.; Chen, M.; Lu, G. Virtual Geographic Environment: A Workspace for Computer-Aided Geographic Experiments. *Ann. Assoc. Am. Geogr.* **2013**, *103*, 465–482.
9. Trenholme, D.; Smith, S.P. Computer game engines for developing first-person virtual environments. *Virtual Real.* **2008**, *12*, 181–187.
10. Natephra, W.; Motamedi, A.; Fukuda, T.; Yabuki, N. Integrating building information modeling and virtual reality development engines for building indoor lighting design. *Vis. Eng.* **2017**, *5*, 21.
11. Rua, H.; Alvito, P. Living the past: 3D models, virtual reality and game engines as tools for supporting archaeology and the reconstruction of cultural heritage e the case-study of the Roman villa of Casal de Freiria. *J. Archaeol. Sci.* **2011**, *38*, 3296–3308.
12. Indraprastha, A.; Shinozaki, M. The Investigation on Using Unity3D Game Engine in Urban Design Study. *ITB J. Inf. Commun. Technol.* **2009**, *3*, 1–18.
13. Rafiee, A.; Van Der Male, P.; Dias, E.; Scholten, H. Interactive 3D geodesign tool for multidisciplinary wind turbine planning. *J. Environ. Manag.* **2018**, *205*, 107–124.
14. Trubka, R.; Glackin, S.; Lade, O.; Pettit, C. A web-based 3D visualisation and assessment system for urban precinct scenario modelling. *ISPRS J. Photogramm. Remote Sens.* **2016**, *117*, 175–186.
15. Susi, T.; Johannesson, M.; Backlund, P. *Serious Games: An Overview*; Institutionen för kommunikation och information, Skövde, Sweden, 2007.
16. Wahyudin, D.; Hasegawa, S. *The Role of Serious Games in Disaster and Safety Education: An Integrative Review*; In Proceedings of the 25th International Conference on Computers in Education. Asia-Pacific Society for Computers in Education: Christchurch, New Zealand, 2017; pp. 180–190.
17. Aditya, T.; Laksono, D. Geogame on the peat: Designing effective gameplay in geogames app for haze mitigation. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2018**, *42*, 5–10.
18. Mat, R.C.; Shariff, A.R.M.; Zulkifli, A.N.; Rahim, M.S.M.; Mahayudin, M.H. Using game engine for 3D terrain visualisation of GIS data: A review. *IOP Conf. Ser. Earth Environ. Sci.* **2014**, *20*, 012037.
19. Reyes, M.E.P.; Chen, S.C. A 3D Virtual Environment for Storm Surge Flooding Animation. In Proceedings of the 2017 IEEE Third International Conference on Multimedia Big Data (BigMM), Laguna Hills, CA, USA, 19–21 April 2017; pp. 244–245.
20. Evangelidis, K.; Papadopoulos, L.; Papatheodorou, K.; Mastorokostas, P.; Hilar, C. Geospatial Visualizations: Animation and Motion Effects on Spatial Objects. *Comput. Geosci.* **2018**, *111*, 200–212.
21. Thöny, M.; Schnürer, R.; Sieber, R.; Hurni, L.; Pajarola, R. Storytelling in Interactive 3D Geographic Visualization Systems. *ISPRS Int. J. Geo Inf.* **2018**, *7*, 123.
22. Tsai, Y.T.; Jhu, W.Y.; Chen, C.C.; Kao, C.H.; Chen, C.Y. Unity game engine: Interactive software design using digital glove for virtual reality baseball pitch training. *Microsyst. Technol.* **2019**, *9*, 1–17.
23. Alatalo, T.; Pouke, M.; Koskela, T.; Hurskainen, T.; Florea, C.; Ojala, T. Two real-world case studies on 3D web applications for participatory urban planning. In Proceedings of the 22nd International Conference on 3D Web Technology, Brisbane, Australia, 5–7 June 2017.
24. Lee, W.L.; Tsai, M.H.; Yang, C.H.; Juang, J.R.; Su, J.Y. V3DM+: BIM interactive collaboration system for facility management. *Vis. Eng.* **2016**, *4*, 5.
25. Dutton, C. Correctly and accurately combining normal maps in 3D engines. *Comput. Games J.* **2013**, *2*, 41–54.
26. Julin, A.; Jaalama, K.; Virtanen, J.P.; Pouke, M.; Ylipulli, J.; Vaaja, M.; Hyypä, J.; Hyypä, H. Characterizing 3D City Modeling Projects: Towards a Harmonized Interoperable System. *ISPRS Int. J. Geo Inf.* **2018**, *7*, 55.
27. Petridis, P.; Dunwell, I.; Panzoli, D.; Arnab, S.; Protopsaltis, A.; Hendrix, M.; Freitas, S.; De Freitas, S. Game Engines Selection Framework for High-Fidelity Serious Applications. *Int. J. Interact. Worlds* **2012**, *2012*, 1–19.
28. Bayburt, S.; Baskaraca, A.P.; Karim, H.; Rahman, A.A.; Buyuksalih, I. 3D Modelling And Visualization Based On The Unity Game Engine—Advantages And Challenges. *ISPRS Ann. Photogramm. Remote. Sens. Spat. Inf. Sci.* **2017**, *W4*, 161–166.
29. Unity3D. Skybox Series. 2019. Available online: <https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633> (accessed on 28 March 2019).
30. Unity3D. Unity Scripting API: Physics Raycast. Available online: <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html> (accessed on 12 May 2019).

31. Unity3D. Unity3D Standard Assets. 2019. Available online: <https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351> (accessed on 19 March 2019).
32. Aditya, T.; Laksono, D.; Sutanta, H.; Izzahudin, N.; Susanta, F. A usability evaluation of a 3d map display for pedestrian navigation. *ISPRS Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2018**, *42*, 3–10.
33. Liao, H.; Dong, W. An Exploratory Study Investigating Gender Effects on Using 3D Maps for Spatial Orientation in Wayfinding. *ISPRS Int. J. Geo Inf.* **2017**, *6*, 60.
34. Liao, H.; Dong, W.; Peng, C.; Liu, H. Exploring differences of visual attention in pedestrian navigation when using 2D maps and 3D. *Cartogr. Geogr. Inf. Sci.* **2017**, *44*, 474–490.
35. Halik, Ł. Challenges in Converting the Polish Topographic Database of Built-Up Areas into 3D Virtual Reality Geovisualization. *Cartogr. J.* **2018**, *55*, 391–399.
36. Bakri, H.; Allison, C. Measuring QoS in web-based virtual worlds. In Proceedings of the 8th International Workshop on Massively Multiuser Virtual Environments, Klagenfurt, Austria, 10–13 May 2016; pp. 1–6.
37. Belussi, A.; Migliorini, S. A Framework for Integrating Multi-accuracy Spatial Data in Geographical Applications. *Geoinformatica* **2012**, *16*, 523–561.
38. Caquard, S.; Cartwright, W. Narrative Cartography: From Mapping Stories to the Narrative of Maps and Mapping. *Cartogr. J.* **2014**, *51*, 101–106.
39. Cartwright, W.; Field, K. Exploring Cartographic Storytelling. Reflections on Mapping Real-life and Fictional Stories. In Proceedings of the International Cartographic Conference, Rio de Janeiro, Brazil, 23–28 August 2015.
40. Biljecki, F.; LeDoux, H.; Stoter, J. An improved LOD specification for 3D building models. *Comput. Environ. Urban Syst.* **2016**, *59*, 25–37.



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).