INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 2, PP. 353–360
Manuscript received Fenruary 4, 2025; revised May, 2025. doi: 10.24425/ijet.2025.153580

Detailed maps conversion technique for usage in realtime 3D mobile games

Maciej Kopczynski

***Abstract*—Research presents findings and outlines techniques for converting detailed GIS maps into simplified versions suitable for real-time 3D mobile games, specifically optimized for devices with medium to low processing power. This approach is particularly valuable for developers of mobile 3D applications that incorporate real-world maps, such as tycoon, strategy, or geolocation-based games. The results demonstrate balance between retaining essential map details from the original data and achieving high-performance 3D graphics across a range of mobile devices.**

***Keywords*—map; 3D mobile app; conversion; mobile device**

I. Introduction

LOCATION-**B**ased **M**obile **G**ames (**LBMG**) are a distinctive category of games where real-world map data and geographic location play a central role in game design. Unlike traditional games, LBMGs leverage mobile devices such as smartphones and tablets not only for gameplay but also for tracking the player's position in real time. An example of such a game is [14]. The integration of location services with gameplay introduces complexities that make the development of LBMGs more challenging than that of conventional games.

The primary challenge in developing LBMGs lies in ensuring a seamless and continuous flow of information between real-world and virtual maps. Achieving this while maintaining a high frame-per-second (FPS) rate is crucial, as it directly affects gameplay quality. However, this process requires analyzing, selecting, and converting vast amounts of geospatial data, which can overwhelm mobile devices, particularly those with lower processing power. This makes the development process demanding even for experienced developers, and the end-users may also face issues if their devices lack the necessary computing performance.

This research proposes an optimization solution to streamline the preprocessing, analysis, and visualization of large GIS datasets for mobile devices. The hypothesis suggests that reducing the level of detail in these datasets can significantly cut CPU and GPU processing times, particularly on lower-end devices. This approach also accelerates the production of new LBMGs and enhances game stability, performance, and user experience. Key elements requiring optimization include:

- data acquisition and selection: only relevant GIS data should be selected to ensure value in gameplay,
- efficient data storage: proper structuring allows quick data retrieval at various detail levels, maintaining game smoothness,
- data visualization: Clear and engaging visuals are crucial for attracting players and enhancing gameplay immersion,
- optimized indexing: organizing data to support quick search and integration of real-world and virtual elements enhances player experience.

Currently, there are few comprehensive solutions for optimizing data for mobile game engines, forcing developers to rely on multiple tools, which increases complexity and demands extensive knowledge of their integration capabilities. Game developers often rely on a combination of different tools to achieve the desired outcomes.

The existing literature largely focuses on describing concepts or offering partial optimizations for converting this type of data into a more efficient, ready-to-use format. The integration of open geospatial data into 3D game engines for urban digital twin applications is discussed in [1], while industrial applications are highlighted in [2]. A cost-effective virtual tour experience within a GIS-based educational game is detailed in [3]. Frameworks for location-based mobile games are exemplified in [4], [7], and [5]. A global geospatial analysis platform built on Google Earth Engine is introduced in [6]. Foundational concepts related to gamification and spatial elements in video games are outlined in [8] and [9]. Existing methods for optimizing maps and analyzing performance in games are presented in [10] and [11]. The process of designing maps specifically for games is described in [12] and [13].

Market products like ArcGIS [16], OpenMapTiles [17], Mapbox [18] and Bing Maps [19] offer services and game SDKs that provide map data with varying levels of detail, depending on the zoom level. However, products that offer raster tiles cannot be converted into a 3D game environment because the maps are pre-rendered in 2D, making it impossible to extract detailed information from flat image data. Even for 2D games or cleverly designed 2.5D games, which are simulating some sort of depth, preparing raster data with an adequate level of detail demands a significant amount of disk space, which is often a problem on mobile devices.

The work was supported by the grant WZ/WI-IIT/3/2023 from Bialystok University of Technology. Research results are based on the project "Geo Game Service – a scalable service that provides useful data about geolocation information in asynchronous multiplayer games" financed by National Center of Research and Development.

M. Kopczynski is with Faculty of Computer Science, Bialystok University of Technology, Bialystok, Poland (e-mail: m.kopczynski@pb.edu.pl).

The paper is structured as follows. In Section II, a comprehensive overview of the fundamental definitions relevant to the study is presented, laying the groundwork for the concepts discussed in subsequent sections. Section III goes into the detailed description of the implemented solutions, outlining the various techniques and strategies employed, as well as the optimizations made within the test applications to enhance performance and efficiency. This section also provides insights into the rationale behind the chosen methods and highlights the challenges addressed during implementation. Finally, Section IV is dedicated to a thorough presentation and analysis of the experimental results, including a discussion of the outcomes and their implications, along with any observed trends and key findings derived from the experimentation.

II. Basic definitions

To comprehensively measure the results of map transformations and evaluate the performance of applications on mobile devices, it is important to establish a set of clear and precise definitions. For each geographical region under consideration, the following parameters will be assessed to determine efficiency and accuracy:

- R_{points} - the total number of data points used to define the shape of the road network. This metric is crucial for understanding the complexity and detail of the road representation.
- E_{dist} - the aggregate distance of all roads within the region. This value helps in assessing the overall connectivity and extent of the road infrastructure.
- T_{gen} - the time required to generate the entire area, which provides an indication of the computational efficiency of the generation process.
- T_{load} - the duration needed to load and process the region in the client environment. This parameter is vital for evaluating the responsiveness and usability of the application on mobile devices.
- T_{frame} - the time taken to render the resulting geometry within the client environment, impacting the frame rate and smoothness of the user experience.
- R_{verts} - the total number of vertices forming the road network mesh. This value indicates the geometric complexity of the road network, which can influence rendering performance.
- R_{tris} - the total number of triangles used to construct the final road network mesh. This metric is important for understanding the rendering workload and graphical performance impact.

To ensure that each playable region in the game environment is both realistic and engaging, the algorithm for generating city and road graphs must adhere to the following essential requirements:

1) **Connected Weighted Graph (CWG)** - each region must be modeled as a connected weighted graph, with cities represented as vertices (V) and roads as edges (E). This structure ensures that all cities within the region are interconnected, facilitating navigation and travel within the game.
2) **Minimum Number of Cities (MNoC)** - the graph CWG must contain a minimum of V_{count} vertices, each representing a unique city or town. This requirement guarantees that the region has a sufficient number of urban areas to make the gameplay dynamic and diverse.
3) **City Separation** - to maintain realistic spatial distribution, each city must be placed no closer than V_{sep} kilometers (in terms of geodetic distance) from any other city. This spacing helps create a more authentic representation of a geographical area, preventing cities from clustering unrealistically.
4) **Road Network (RN)** - the road network must have a minimum total weight of E_{dist} kilometers. This requirement ensures that the generated road system is extensive enough to support meaningful exploration and gameplay. The total weight of the edges E must meet or exceed this threshold to be considered successful.

III. Solution description

The application was developed using .NET technology, with C# as the primary programming language. The PC-side development and coding tasks were carried out using Microsoft Visual Studio 2022. For the mobile component, the application was designed to run on devices using the Android operating system. This part of the development was implemented using Unity 2021 Long Term Support (LTS) version. Unity 2021 LTS was chosen for its stability and extensive support for 3D rendering, real-time simulation and cross-platform deployment.

Fig. 1 shows solution layout.

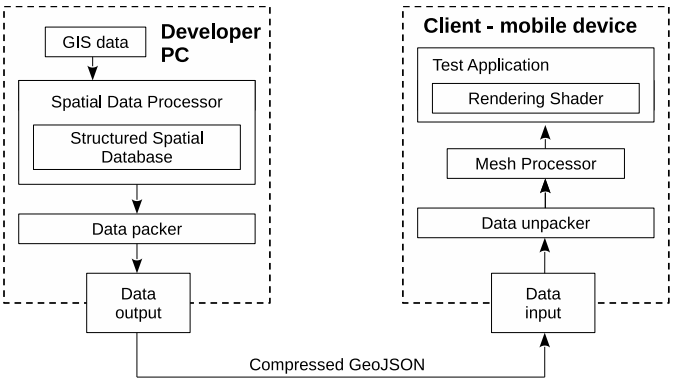


Fig. 1. General layout of solution.

The system is composed of two main components: a test application running on a mobile device (client) and a real map processing part operating on a PC (developer). The transformed geospatial data is packaged together with the game files and integrated into the client application using a compressed GeoJSON format [15].

The key functional components of the developer-side application on the PC are as follows:

- *GIS Data Source* - a source of real-world map data obtained from OpenStreetMap, which provides comprehensive and up-to-date geospatial information.

- *Structured Spatial Database* - a PostgreSQL database enhanced with the PostGIS extension, used to store and manage geospatial data. This database structure ensures quick access and spatial queries for data processing.
- *Spatial Data Processor* - the core unit responsible for processing geographical data, including tasks like data filtering, transformation and optimization for further use in the mobile environment.
- *Data Packer* - a module that prepares the processed geospatial data by converting and compressing it into game-compatible data packages. This component ensures that the data is efficiently formatted for integration with the client application.

On the mobile device side, the main functional components include:

- *Data Unpacker* - a module responsible for decompressing and unpacking the embedded data, preparing it for further use. It ensures that the transformed data is ready for processing by subsequent components.
- *Mesh Processor* - this unit processes the unpacked geospatial data, converting it into mesh structures suitable for visualization within the game environment.
- *Rendering Shader* - a component used by the test application to render and visualize the processed data, ensuring high-quality 3D representation on the mobile device.

GIS Data Source is created from the *Planet.osm* export from OpenStreetMap. The essential elements required for this project are the exporting of route geometry tagged with the highway category using one of the following tags: motorway, trunk, primary, secondary and tertiary. Those are joined by city data contained in node geometries, which are tagged with place type matching one of the following values: city, town and village. The suitability of *osm2pgsql* and *imposm3* tools for importing the data was evaluated. Both tools were found to be capable of transforming the binary package into a structured database. *Imposm3* was used in this paper, however, benchmarking these tools was beyond the scope of this paper. Additionally, data deemed unnecessary for the game's functionality, such as street names, was removed to keep the database size manageable and optimized for performance. The dataset is organized within a PostgreSQL database using the PostGIS extension, which facilitates spatial indexing and enhances GIS operations through specialized acceleration structures.

The *Spatial Data Processor* is a vital module on the developer side, designed specifically to filter and extract only the data essential for the client environment. This module determines the structure and layout of the game world by selectively processing geospatial data based on gameplay requirements. *Spatial Data Processor* is implemented as a .NET application and communicates with the *Structured Spatial Database*. Using this connection, the processor refines the raw data and assembles it into the final form of the game world. This process includes defining spatial relationships, applying necessary transformations and converting data into a format compatible with the mobile application.

The algorithm within the *Spatial Data Processor* generates a set of routes and cities, representing E (edges) and V (vertices), respectively. These routes are then evaluated to calculate R_{points}. The entire process halts at this stage, recording T_{gen} as the total runtime of the generation algorithm.

After collecting the resultant set of roads, an additional geometry aggregation pass is executed. During this step, roads in close proximity are merged into single entities. Furthermore, as placeholders for 3D models, each urban city center is transformed into a roundabout, with roads inside it and extended to meet the circle's boundary.

This method creates a functional road network, incorporating all key roads connecting major cities while filtering out redundant links and excessive geometry from variable-quality OSM data. Consequently, the data volume is reduced and normalized to a standard complexity level, suitable for presenting balanced gameplay maps that maintain performance expectations for mobile devices.

The *Data Packer* and *Data Unpacker* modules play a crucial role in transforming spatial data into an intermediate format and back, facilitating seamless integration into the client application. To minimize the size of the spatial data payload, the GeoJSON format is used in combination with LZMA compression, ensuring efficient data transmission and storage.

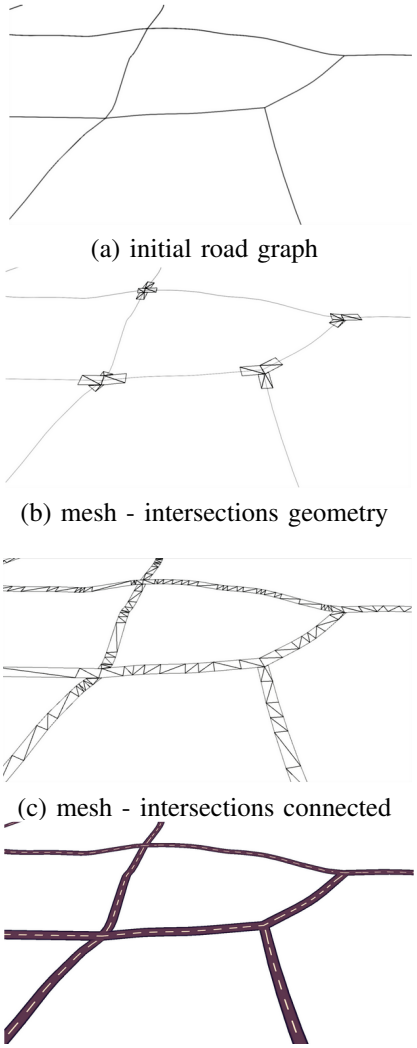
Data Unpacker decompresses these data packets and reconstructs the map data, converting it into a navigation graph. This graph serves as the foundation for pathfinding algorithms, enabling efficient navigation between points within the defined region. Once the navigation graph is established, it is used to generate a detailed road network mesh. The time taken to load and process this data is recorded as T_{load}. The final road mesh is represented by a set of triangles, R_{tris}, and a set of vertices, R_{verts}. To optimize performance on mobile devices, the algorithm is designed to reuse vertices across adjacent triangles, significantly reducing GPU memory consumption and enhancing rendering efficiency.

The generated mesh is subsequently loaded into the rendering part, where the rendering performance is measured as T_{frame} during specific test scenarios.

The entire mesh generation process is illustrated in Fig. 2. Part a) depicts a sample section of the initial road graph. Part b) illustrates the first stage of mesh generation, focusing on the geometry of intersections. Part c) shows the second stage, where connections between intersections are created, and part d) displays the final mesh appearance with textures applied, providing a complete visual representation of the road network.

A. Spatial Data Processor algorithm

The main goal of the algorithm running on the developer's PC is to construct a road network that effectively connects a series of cities within a specified region. This network must meet predefined criteria, including constraints on the total length of the roads and the spatial distribution of the cities. The following pseudocode provides an outline of the steps involved in the processing algorithm.



(a) initial road graph

(b) mesh - intersections geometry

(c) mesh - intersections connected

(d) final appearance with textures

Fig. 2. Visualisation of mesh network road generation

INPUT: Region-clipped map data including way (linear geometry, eg. roads) and $point$ (points geometry data)
OUTPUT: List of $routes$ and $cities$

```
allCityClasses ← [city, town, village]
roadClassesConsidered ← [motorway, trunk]
additionalRoadClasses ← [primary, secondary, tertiary]
repeat
    roadClassesConsidered.push(allRoadClasses.pop())
    roads ← CollectRoads(way, roadClassesConsidered)
    mergedRoads ← Union(roads)
    simplifiedRoads ← Simplify(roads)
    allCities ← CollectCities(point, allCityClasses)
    citiesConnected ← FilterDisconnectedCities(allCities, simplifiedRoads)
    citiesSorted ← SortCitiesByPopulation(cities)
    citiesReduced ← ReduceDensity(citiesSorted, V_sep)
    citiesTrimmed ← TakeFirst(citiesReduced,V_count)
    cities ← SnapCitiesToRoads(citiesTrimmed, simplifiedRoads)
    for each city cityStart in cities do
        for each city cityEnd in cities do
            routes.push(CalculateRoute(cityStart, cityEnd))
        end for
        totalDistance ← SumDistances(routes)
    end for
until totalDist ¡ E_dist or additionalRoadClasses = ∅
```

The input for the algorithm is regional world map data clipped and stored in a PostgreSQL database, which includes roads and linear geometries represented as way and point geometries represented as $point$. The output consists of two lists: $routes$, which signify connections between selected $cities$. Initially, the algorithm categorizes cities and roads based on the standardized OpenStreetMap attribute guidelines. It begins with the highest road classes (motorway and trunk) and progressively includes lower classes (primary, secondary, and tertiary) until the specified conditions are satisfied. The algorithm extracts road segments from the structured spatial database, treating each segment as an individual instance. These segments are merged with a minor tolerance to join roads that might have been added across different OpenStreetMap changesets. The merging of the roads is a subroutine that transforms each of the lines into a polygonal area encompassing a distance (5 km) surrounding each road geometry. All of the polygons are then merged together and simplified using the Douglas-Peucker algorithm with a tolerance of 500 m to make computation easier in the next part without losing too much of the quality. Subsequently, an approximate medial axis is calculated for this kind of polygonal geometry, thereby creating the resulting road mesh which is then used for subsequent steps. It should be noted that created roads may be slightly off compared to real-world values. However, by merging adjacent roads, a result is produced that is both computationally less demanding and visually more appealing than the raw data. In a subsequent step, cities are snapped to the mesh in order to correct any remaining inaccuracies. The algorithm then simplifies road lines by removing redundant points using the Visvalingam–Whyatt algorithm. The algorithm collects all cities in the region as point data and filters out those not connected to the main graph of the calculated road network. The cities are sorted by population to select the largest within the region, and lower-ranked cities within the specified V_{sep} area are removed to reduce clutter. Only the top V_{count} cities are retained, and they are snapped to points on the road network, creating new nodes if needed. Finally, Dijkstra's algorithm is employed to find the shortest path between each pair of cities, which is then added to the $routes$ list. The total distance of the roads in the $routes$ list is calculated and compared against the defined stop-condition threshold, E_{dist}. The process repeats until the total distance meets or exceeds E_{dist} or until there are no remaining roads in the dataset.

B. Challenges and limitations

Crucial challenge in solution development was the creation of a stable experience, regardless of the player's gameplay decision. The player's experience must be consistent across all possibilities without the introduction of artificial limitations on

their choices. This requirement called for the development of an algorithm that can process input data of varying complexity and deliver a similar experience in terms of gameplay, focusing on technical performance. Algorithms were designed to process both dense urban areas and sparsely populated regions. It was imperative to target fixed ranges of desired city counts and total length of roads in order to achieve such consistency. Both gameplay design and performance target scale nearly linearly with the length of roads and amount of cities. This relationship is then reflected in the mesh vertex count, which can be a bottleneck in a limited-resource environment of mobile GPU deployment targets.

Another challenge is transforming everything into small packages that can be easily expanded and handled in isolated fashion. The conversion of all data into GeoJSON as an intermediate format facilitates the distribution, inspection, and further expansion of game assets. The fact that it is a well-established RFC standard also provides a degree of safety in maintaining the use cases for years, with the added benefit of being utilized by future third-party tooling compatible with the standard to further improve the process.

IV. Experimental results

The results presented in this study were obtained using a PC configured with 64 GB of DDR4 RAM operating at 3200 MHz, paired with a 6-core AMD Ryzen 5600X processor. The database engine utilized was PostgreSQL 14, with the tablespace hosted on an SSD connected via NVMe at PCIe Gen3 speeds, ensuring high data transfer rates and efficient performance. The mobile devices tested included a Samsung Galaxy S20, a Razer Phone, and a Samsung Galaxy S6, each updated to the latest Android operating system versions released by their respective manufacturers. The Samsung Galaxy S20 was classified as a high-performance device, offering cutting-edge features and superior processing power. The Razer Phone represented a mid-range option, balancing performance and efficiency. Lastly, the Samsung Galaxy S6 was categorized as a lower-performance device, providing insight into how the application performs on older hardware.

Three distinct areas were selected for the experiments, representing different geographic characteristics around the world—from the sparsely populated wilderness of Canada to the densely packed urban regions on the U.S. east coast. The specific areas studied are:

- *Sample 1: Northeastern US.* This sample includes few combined states. It is characterized by dense urban regions with numerous roads but covers a relatively small land area compared to the other samples.
- *Sample 2: Texas.* This is the largest state in the contiguous United States, featuring numerous major cities like Houston, Dallas, Austin, and San Antonio, which are separated by extensive stretches of open land.
- *Sample 3: Saskatchewan.* A sparsely populated province in Canada with a large area but a low number of cities and roads connecting them.

For the purposes of this analysis, roads were defined according to OpenStreetMap (OSM) classifications, specifically including highways tagged as ”motorway,” ”trunk,” ”primary,” ”secondary,” and ”tertiary.” Settlements were identified as locations tagged as ”city,” ”town,” or ”village” in the OSM data. To enhance the realism of the game environment, cities located just outside the primary boundary of the region were included within a buffer zone. This allowed for the creation of external roads that seamlessly fade into the game’s fog effect, reducing the sense of isolation and enhancing immersion.

The parameters for the algorithm on the developer’s PC were configured to ensure that each processed map region fell within a specified range of city complexity, targeting between $10 \leq V_{count} \leq 20$ settlements. Additionally, a minimum road complexity threshold was set, requiring $E_{dist} > 8000$ to ensure adequate connectivity and detail in the road network.

To ensure reliability and reduce variability caused by background processes on both the PC and mobile devices, all time measurements were averaged over 100 runs. This averaging approach minimizes the influence of extraneous system activities, providing a more accurate representation of the algorithm’s performance. For consistency, the mobile device tests maintained the same projection length and camera movement scheme across all analyzed region types.

The results from the PC processing of selected regions are presented in Tables I and II. These tables detail the steps involved in preparing the map data for subsequent testing on mobile devices. The *Input* columns provide information about the raw data complexity for each region: *Area* specifies the size of the region, *Road network* lists the total length of relevant road types, and *Settlements* indicates the number of identified settlements. The T_{gen} column shows the total time taken from the initial data reading to the completion of the final packaged data for mobile use.

TABLE I
RESULTS OBTAINED FROM PROCESSING REAL MAP DATA - INPUT

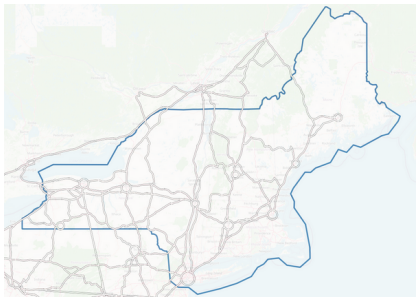
Region	Input		
	Area	Road network	Settlements
	$[km^2]$	$[km]$	$[-]$
Northeastern US	348 076	89 020	1 325
Texas	708 011	156 635	1 141
Saskatchewan	504 367	32 319	179

TABLE II
RESULTS OBTAINED FROM PROCESSING REAL MAP DATA - OUTPUT DATA

Region	Output		T_{gen}
	E_{dist}	R_{points}	
	$[-]$	$[-]$	$[s]$
Northeastern US	19 003	5 143	207
Texas	28 818	6 749	115
Saskatchewan	20 017	4 365	14

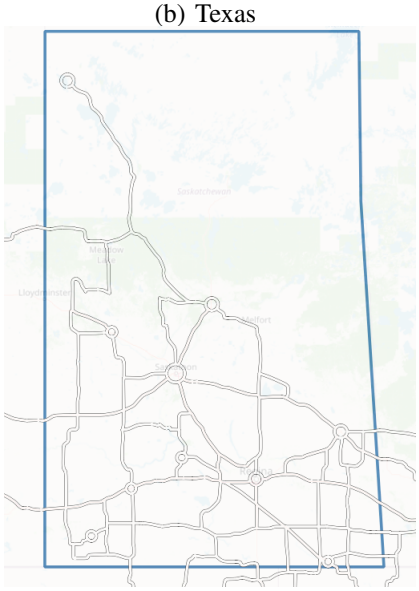
Fig. 3 shows three regions after input map processing.

As observed, the generation time varies significantly based on the complexity of the road network and the number of cities within a region, but it does not strongly correlate with the physical size of the region. This is because the algorithms operate on geometric data and graph structures. For example,



(a) Northeastern US

(b) Texas



(c) Saskatchewan

Fig. 3. Visualisation of three regions after map processing

despite Saskatchewan's large land area, it has relatively few roads, resulting in shorter processing times.

An additional insight from Fig.3 and the data presented in TablesI and II is that, despite significant variations in input characteristics across different regions, the resulting output sets remain relatively consistent in terms of appearance and properties. This uniformity is particularly advantageous for gameplay applications, where maintaining regional balance is crucial to ensure fair and equitable experiences for all players. By minimizing geographic variability, the algorithm ensures that players are neither penalized nor given an advantage based on their starting location. Moreover, with suitable parameter adjustments, the algorithm can be adapted to generate balanced and consistent game environments in any region worldwide, as long as the area includes properly classified roads and settlements.

Map data generated on the PC was transferred to the mobile device for performance testing using a dedicated test application. This testing measured rendering performance and processing times, which are critical factors affecting the overall efficiency of the mobile application. Generated geometry was also examined: R_{verts} represents the total number of vertices in the road network mesh, and R_{tris} indicates the number of triangles forming the final mesh. Importantly, these values depend solely on the map region's characteristics and remain unchanged regardless of the mobile device used. Fig. 4 provides an example screenshot of the Texas region as displayed within the mobile application.

Fig. 4. Exemplary screenshot of Texas region from test application.

Table III shows the results obtained on various mobile devices across different regions, using both raw and processed map data. The T_{load} column indicates the time taken to load the map data from the initial data access point, while T_{frame} represents the frame rendering time. The *Raw map* column group includes measurements for unprocessed map data, filtered only by the specified road and city tags. In contrast, the *Processed map* column group contains data processed by the PC before being tested on the mobile devices.

TABLE III
PERFORMANCE RESULTS ON MOBILE DEVICES FOR SELECTED REGIONS

Mobile device	Filtered raw map		Processed map	
	T_{load} $[ms]$	T_{frame} $[ms]$	T_{load} $[ms]$	T_{frame} $[ms]$
Northeastern US				
Samsung Galaxy S20	30 350	8.53	106.32	0.03
Razer Phone	49 920	18.44	153.31	0.05
Samsung Galaxy S6	79 072	37.48	288.58	0.11
Texas				
Samsung Galaxy S20	28 222	7.01	88.72	0.02
Razer Phone	44 842	15.08	122.67	0.04
Samsung Galaxy S6	75 786	29.25	235.45	0.12
Saskatchewan				
Samsung Galaxy S20	1 526	0.36	68.67	0.02
Razer Phone	2 120	0.79	110.92	0.05
Samsung Galaxy S6	3 903	1.66	174.14	0.10

The geometric results for the filtered raw and processed map data are as follows:

- filtered raw map:
  - $R_{verts} = 4\ 402\ 754$ for Northeastern US; 3 690 430 for Texas; 193 471 for Saskatchewan,
  - $R_{tris} = 4\ 213\ 483$ for Northeastern US; 3 355 568 for Texas; 177 887 for Saskatchewan.
- processed map:
  - $R_{verts} = 12\ 142$ for Northeastern US; 15 924 for Texas; 9 866 for Saskatchewan,
  - $R_{tris} = 10\ 592$ for Northeastern US; 14 022 for Texas; 8 926 for Saskatchewan.

The timing and geometric analysis reveal that each region was successfully converted into a renderable package that delivers consistent performance, regardless of variations in the complexity of the map data. This uniformity ensures that the game meets similar performance standards across a wide range of devices, accommodating differences in computing power and graphical capabilities. Achieving this level of consistency is vital in modern game development, where players demand a stable and smooth experience, regardless of the device they are using or the specific gameplay scenarios they choose.

For lower-tier mobile devices, however, certain challenges arise. Specifically, regions with highly detailed and extensive raw map data, such as the Northeastern US, could not be loaded in their entirety due to hardware limitations. To address this, the map had to be divided into smaller, manageable segments, with the overall timing results computed as the cumulative sum of all processing and rendering operations. Additionally, it's important to consider the constraints imposed by rendering engines. Most engines default to using 16-bit integers for mesh index buffer data, which restricts the number of indices per mesh to a maximum of 65536. While this limitation is suitable for many scenarios, higher-end mobile devices have the capability to support 32-bit indices, allowing for more complex and detailed mesh structures.

V. Conclusions

The research conducted demonstrates that it is feasible to develop efficient and adaptable methods for processing real-world data tailored to the game development industry, particularly for mobile devices. Optimizing the transformation of map data for use in geolocation-based games is a critical aspect of development. This ensures consistency between in-game maps and the ever-evolving real-world maps, which are frequently updated with new details due to urban expansion, construction of new roads, and improved cartographic accuracy.

The proposed solution accelerates the creation of new geolocation games and contributes to a smoother gameplay experience. One significant advantage is that it reduces the need for deep expertise in GIS data acquisition, storage, and processing, as well as the real-time visualization of extensive GIS datasets within game engines on mobile devices. However, developers must carefully choose the appropriate parameters for the presented algorithms to meet the desired level of detail for the transformed map data. Comparative analysis of different outputs is recommended to ensure that the final product aligns with project requirements.

It is crucial to maintain the vertex count as low as possible to achieve optimal performance. For older mobile devices, the practical limit is approximately 100 000 vertices per scene, whereas modern devices can handle up to 1 million vertices. Utilizing unprocessed data can quickly exceed these limits, leading to excessive rendering times that monopolize most of the processing resources. This would prevent the real-time rendering of other essential 3D objects, severely impacting the framerate and overall user experience. Moreover, the storage demands of unprocessed map data are impractical for mobile games on medium to lower-grade devices, particularly since road networks are just one of many objects that need to be rendered in contemporary games.

Future research will focus on enhancing and refining these methods. Key areas for development include advanced strategies for selecting settlements, creating simplified road meshes for larger cities, and introducing mechanisms for generating parallel roads while adhering to different road classification requirements. Additionally, further testing on a broader range of mobile devices and larger geographic regions will be undertaken to assess scalability and robustness.

References

[1] Rantanen T, Julin A, Virtanen J-P, Hyyppä H, Vaaja MT. Open Geospatial Data Integration in Game Engine for Urban Digital Twin Applications. ISPRS International Journal of Geo-Information. 12(8):310, 2023.

[2] Schleich B, Anwer N, Mathieu L, Wartzack S. Shaping the digital twin for design and production engineering. CIRP Ann. 2017, vol. 66, pp. 141—144, 2017.

[3] Varinlioglu G, Sepehr VA, Eshaghi S, Balaban O, Nagakura T. GIS-Based Educational Game Through Low-Cost Virtual Tour Experience – Khan Game. Proceedings of the 27th CAADRIA Conference, Sydney, 9-15 April 2022, pp. 69–78, 2022.

[4] Ionescu G, Valmaseda JMD, Deriaz M. GeoGuild: Location-Based Framework for Mobile Games. International Conference on Cloud and Green Computing, Karlsruhe, Germany, 2013, pp. 261–265, 2013.

[5] Predescu A, Mocanu M, Chiru C. A case study of mobile games design with a real-world component based on Google Maps and Unity. 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Pitesti, Romania, pp. 1-6 2021.

[6] Gorelick N, Hancher M, Dixon M, Ilyushchenko S, Thau D, Moore R. Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sens. Environ. vol. 202, pp. 18–27, 2017.

[7] Lee A, Chang YS, Jang I. Planetary-Scale Geospatial Open Platform Based on the Unity3D Environment. Sensors, vol. 20, no. 20, 2020.

[8] Deterding S, Dixon D, Khaled R, Nacke L. From Game Design Elements to Gamefulness: Defining Gamification, Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, pp. 9–15, 2011.

[9] Wolf M. 3 Space in the Video Game. The Medium of the Video Game; Wolf, M., Ed.; University of Texas Press: New York, NY, USA, 2021.

[10] Abubakar A, Zeki AM, Chiroma H. Optimizing Three-Dimensional (3D) Map View on Mobile Devices as Navigation Aids Using Artificial Neural Network. 2013 International Conference on Advanced Computer Science Applications and Technologies, Kuching, Malaysia, pp. 232–237, 2013.

[11] Koh E, Park G, Lee B, Kim D, Sung S. Performance Validation and Comparison of range/INS integrated system in urban navigation environment using Unity3D and PILS. Proceedings of the 2020 IEEE/ION Position, Location and Navigation Symposium (PLANS), Portland, OR, USA, 20–23 April 2020; pp. 788–792, 2020.

[12] Zagata K, Medynska-Gulij B. Mini-Map Design Features as a Navigation Aid in the Virtual Geographical Space Based on Video Games. ISPRS Int. J. Geo-Inf, 12(2), 58, 2023.

[13] Toups ZO, Lalone N, Alharthi SA, Sharma HN, Webb AM. Making Maps Available for Play: Analyzing the Design of Game Cartography Interfaces. ACM Trans. Comput. Hum. Interact., vol. 26, pp. 1–43, 2019

[14] Transportico game homepage, https://play.google.com/store/apps/details?id=com.riftcat.transportico Last accessed 16 September 2024.
[15] RFC 7946 standard homepage, https://datatracker.ietf.org/doc/html/rfc7946 Last accessed 16 September 2024.
[16] ArcGIS homepage, https://www.arcgis.com/index.html Last accessed 24 September 2024.
[17] OpenMapTiles homepage, https://openmaptiles.org/ Last accessed 24 September 2024.
[18] Mapbox homepage, https://www.mapbox.com/ Last accessed 24 September 2024.
[19] Bing Maps homepage, https://www.bing.com/maps Last accessed 24 September 2024.