Multi Patch 3D Terrain Representation for Collaborative Terrain Editor

J T Tarigan, O S Sitompul, M Zarlis, E B Nababan
Faculty of Computer Science and Information Technology, Universitas Sumatera Utara
opim@usu.ac.id

Abstract. In this paper, we propose a method to represent a new method to represent large-scale 3D terrain efficiently to be used in a client-server collaborative terrain editor. The objective is to design a terrain representation that allows changes to be distributed between connected clients without throttling the bandwidth. The method divides the terrain into smaller patches so it can be streamed partially to connected clients. Moreover, we also reduce the data representation by using 8-bit values and modify the terrain representation to decrease the inaccuracy caused by data reduction. We present a thorough description of the proposed method and a demonstration of the technique in a practical collaborative 3D terrain editor application. We performed several tests with different terrain specifications and measured the interactivity level of the application. Based on these tests, we can conclude that the proposed method is capable of reducing the time-to-interactive of the application and consequently increasing the interactivity level of the system.

1. Introduction
Modern computer allows developers to produce applications with extensive 3D graphics. Some 3D applications such as open-world games and flight simulators where exploration is the main component of the gameplay may require a large-scale terrain. Generating this content can be complicated and time-consuming. One proposed solution to diminish this problem is to allow content developers to work collaboratively. However, allowing multiple peers to work collaboratively from multiple device requires a well-designed protocol dan data representation to ensure data synchronization between connected clients is performed efficiently. This research is intended to develop a method to represent 3D terrain so it can be transmitted efficiently in a client server-based collaborative terrain editor.

The objective of this research is to develop a terrain representation that may reduce the data required to transmit changes in the part of the terrain made by one user to others. The idea of our method is to split the terrain into smaller, uniformly sized patches of terrain that allows a more efficient data representation. This concept allows the data to be sent as blocks rather than points, which, as we will prove later, may increase the efficiency of the data communication. Additionally, we also present a new method that allows the terrain data used to be presented as 8-bit values without losing the details from a 16-bit or 32-bit values data. Our method uses zigzag data sequence and stores derivative values instead of the height values that is similar to the idea of macroblock in image and video compression. By relying on the idea neighboring points tend to have similar value, we are confident that our method is capable to store

In this paper, we will discuss the design of our proposed method and present the result of our research. To validate our proposed solution, we built a simple client-server-based terrain editor application that
uses our method. We then perform a series of tests and observe the result. During the test, we collect two essential values: the size of the data transmitted during the online editing session and the interactivity of the application. The paper is structured as follows: the next section discusses similar works that related to this research. In the third section, we will go through the design of the solution that we proposed in this paper. The fourth section discusses the implementation of the solution that we proposed and discuss the result of the test. Finally, we present the conclusion of our work in the final section.

2. Similar and Related Works
Research on terrain rendering and representation has been available in the early 80's. The most popular representation of 3D terrain, heightfield, was proposed by Fishman et al. [1] and formally redefined as a tool for terrain maps representation by Miller [2]. However, depending on the size and the details required by the application, heightfield data can be both space and power-consuming. Since then, there are various research focused on creating a more effective and efficient terrain representation, especially in 3D world. Some of this research focused on developing an efficient terrain representation based on the purpose of the system. In this section, we will go through several previous study that intended to develop a new method to represent terrain effectively and efficiently.

One of the most significant projects that related to our study was developed by Ellis et al. This project was intended to develop a system that allows multiple connected users to perform changes on a terrain [3]. The project focused on the architecture and network protocol in distributing the state of the terrain to connected users. The system split the terrain into multiple patches and distribute the patches to users that needed the most (users may access nearby terrain more frequently than others). While the intention is similar to our project, their solution does not fit perfectly in our system.

Oswald et al. developed an efficient data representation for 3D rendering by using BFOS algorithm that can optimally reduce the number of triangles for a terrain rendering [4]. The main idea proposed by this work is to reduce the amount of triangles in a less dynamic area (vast flat terrain) by developing a tree from a heightfield data and prune the nodes with similar values. Losasso et al. [5] proposed Geometry clipmaps that aimed to control the level-of-detail in terrain rendering by splitting the terrain into several categories based on the viewer's location and movement. Frasson et al. proposed a screen-space approach and a scalable and flexible data structure using a bounding volume hierarchy [6].

There are also various researches that focused on optimizing 3D object data streaming through networked devices. Poudreux et al. [7] proposed a technique that split the heightfield into regular tiles which are streamed and managed adaptively. The proposed method allows the user's application to request tiles based on its location. The size of the tiles is set to be adaptive so the system may choose to favor either interactivity or quality. Cellier et al. [8] develop a system that allows a large-scale terrain to be streamed to a web clients by compressing the content with a reversible mesh simplification. Herzig et al. developed X3D-EarthBrowser; a web-based terrain browser that uses progressive image-based transmission as a method to transmit the terrain content [9]. Combined with the progressive queries and dynamic level-of-details, the system is capable to give an acceptable interactivity to view a large-scale terrain. Deb et al. developed a protocol to transmit large scale terrain data based on the user's point of view [10]. The idea in this research was to split the terrain into smaller parts and send only the part that is going to be rendered by the client.

3. Design
In this section, we will discuss the design of the proposed solution. As previously stated, this research is part of our broader research project to develop a client-server based terrain editor [11], [12]. Based on this design, we defined that our proposed system must be able to represent a dynamic terrain in a smaller size to be transmitted efficiently amongst connected clients. Additionally, the design of the system should allow lightweight clients (thin clients) to be connected to the server. Hence, any computational task required by the proposed method should be assigned to the server.
Our first objective is to represent a vast terrain into smaller pieces of terrain and develop a cheaper data transmission to transmit the terrain [1]. For each point at position \((x, z)\), there will be a value \(y\) to represent the height of the terrain at that point (assuming that the \(y\) value defines the height of an object). This height values are kept as a 2D image where the \(x\) and \(z\) are the coordinate and \(y\) is the value at that coordinate. Based on the details desired from the terrain, the height that can be represented \(y\) values is vary.

The idea of our proposed method is instead of dealing with one large terrain, we split it to a set of uniform patches called meshGroup. Each meshGroup has the same size and the same amount of meshes and vertices. Given a terrain with \(m \times m\) dimension, we use a \(s\) amount of smaller terrain with \(n \times n\) where \(s \times n = m\). Each patch contains \(n \times n\) meshes and \((n + 1) \times (n + 1)\) vertices. Each of these meshGroup will be assigned with an ID number derived from its position.

Another important point of our proposed method is how each data is stored in a meshGroup. Instead of a horizontal line scanning, we store the data as a zigzag sequence. This method allows adjacent points to be placed sequentially in the array and, in result, increase the chance of having similar (or slightly different) neighboring values. By storing the differences instead of the value, we can decrease the bit-size of the data. In our design, we use an 8-bit data to represent a 32-bit heightmap value.

**Figure 1.** An example of a terrain with 64 8×8 meshGroups (left) and zigzag scanning of a \(n \times n\) meshGroup (right)

In storing the value, we also use a relative delta for each meshGroup. That is, for each meshGroup, we find the maximum and minimum values, calculate the difference, and divide the difference with 256 to get the delta. This design may require additional calculation upon changing the values which will happens frequently due to the objective of this design is to be implemented in a map editor. Upon changing, meshGroup has to check whether the new value is smaller or larger than the minimum or maximum value, respectively, and have to recalculate the difference and rewrite the data all over again if one of these two values is changed. Additionally, instead of storing the height of each point, each value stored in each point is referred to the difference from its previous neighbor. This method will help to reduce the error when dealing with high height intervals that subsequently causes a high delta.

4. Implementation
To validate our thesis, we built a client-server collaborative 3D terrain editor. We use Unity Game Engine as our graphic engine and use its terrain object to simplify building the test application. Users can interact with the application by simply pointing to an area to change the elevation at that certain area. We also add a brush-like feature and allows users to change size of the brush. This will affect the amount of neighboring points affected per input. This feature is common in various terrain editor applications. Figure 2 shows the screenshot of our application running on a Windows based client. In the test, we use a terrain with 1024×1024 resolution consists of 8×8 meshGroup. Hence, the terrain consists of 128×128 (16.384) meshGroups.
The editor application was connected to a dedicated server. The server is responsible to accept the changes from the editor acting as its client application. The server will then apply those changes and modify the affected meshGroups and recalculate those meshGroups. These meshGroups data will be sent to all connected clients. The clients will then apply the changes to the terrain by converting the meshGroups data into terrain data.

As previously described, our objective is to minimize the data required between client and server. Since the client is only sending the user input (position and brush size), the data in this transmission is small. Hence, our observation is focused on the meshGroups data transmitted from the server to the client. To have a basic understanding of the required data size, we will first observe the necessary data to send one meshGroup. Each meshGroup consists of 64 points each represented of an 8-bit (1 byte). Hence, the meshGroup points will require 64 bytes of data.

Additionally, the client needs to send the index of the meshGroup (4 bytes of integer value), the maximum and minimum values (each is 4 byte of floating point, 8 bytes in total). Hence to send a full meshGroup, the client needs to send 72 bytes of data. In comparison, sending a single point (without meshGroup) which will require additional data to define the index of the point is 8 bytes (4 bytes for the integer index and 4 bytes for the floating-point height value). While the previous case looks redundant, keep in mind that terrain editors tend to use a sized brush to modify an area instead of a single point. The figure below shows a series of cases that we use in this test: brush with a radius of 3 in one meshGroup, two meshGroups, and four meshGroups (most left to third from the left) followed by test cases with brush size 12 and 28.

Table 1 below compares the data transmission in each of these cases and 1 more case.

| Case Number | Total Points Affected | Total Data using FP32 (bytes) | Total MeshGroup Affected | Total Data with MeshGroup (bytes) |
|-------------|-----------------------|-------------------------------|--------------------------|----------------------------------|
| 1           | 32                    | 256                           | 1                        | 72                               |
| 2           | 32                    | 256                           | 2                        | 144                              |
| 3           | 32                    | 256                           | 4                        | 288                              |
| 4           | 512                   | 4.096                         | 16                       | 1.152                            |
| 5           | 2.304                 | 18.432                        | 36                       | 2.592                            |
Another important aspect to discuss is the accuracy of the terrain generated by our method. One way to measure this accuracy is to compare the original and the compressed terrain value. However, it is hard to summarize the error since its behavior highly depends on the pattern of the terrain (or meshGroup). Hence, we only measure the accuracy based on direct observation of the pattern of the terrain. To do this, we imported a set of high-quality terrain data and perform a direct comparison between the original and the compressed. Based on these observations, we can conclude that our method is capable to generate an almost identical terrain (pattern). Figure 4 below shows an example of a high-quality terrain taken from terrain.party [13] rendered using the original value (Left) and the compressed value (right).

![Figure 4. Comparison of original terrain (left) and the compressed one (right)](image)

While the image above shows that our proposed method is capable of producing similar terrain, there is a continuity issue that we are yet to solve. Since each patch has a different data dynamic, there is a possibility that the data on the edge of a patch is different from its neighbors. This issue, however, is visually imperceptible since, in the end, our graphic engine treats these patches as a single terrain and smoothen the difference by interpolating both points.

5. Conclusion & Future Works
In this paper, we developed a method to represent terrain in a smaller form of data intended to increase the efficiency of a client-server terrain editor. The purpose of the method is to reduce the transmitted terrain data between clients and server. Based on our test, the proposed method has successfully given a significant data reduction while maintaining the overall pattern of the terrain. Even with a high rate of error compared to the original one, our method has successfully maintained the overall pattern of the terrain and makes the error unnoticeable. This result is acceptable due to the use of the random generator is common in most terrain editor application.

The next phase of this research is to implement the proposed terrain representation in our Collaborative Terrain Editing System. There are several adjustments that we need to implement in order to be able to use this method effectively and efficiently such as communication protocol for distributing changes between clients and synchronizing data between client and server. We are confident that this method may contribute significantly to increase the performance of our system by reducing the communication throughput during the collaboration process. However, due to the additional calculation required to process the patch, it is interesting to observe how this implementation correlate the performance of the server and, in effect, reduce the usability of the system.

References
[1] B. Fishman and B. Schachter, "Computer display of height fields," Comput. Graph., vol. 5, no. 2–4, pp. 53–60, Jan. 1980, doi: 10.1016/0097-8493(80)90007-2.
[2] G. S. P. Miller, "The definition and rendering of terrain maps," ACM SIGGRAPH Comput. Graph., vol. 20, no. 4, pp. 39–48, Aug. 1986, doi: 10.1145/15886.15890.
[3] C. Ellis, P. Babenko, B. Goldiez, J. Daly, and G. A. Martin, "Dynamic Terrain for Multiuser Real-Time Environments," *IEEE Comput. Graph. Appl.*, vol. 30, no. 1, pp. 80–84, Jan. 2010, doi: 10.1109/MCG.2010.5.

[4] S. P. Oswald, K. Ramchandran, and T. S. Huang, "Efficient terrain data representation for 3D rendering using the generalized BFOS algorithm," 1997, vol. 1, pp. 448–451, doi: 10.1109/ICIP.1997.647803.

[5] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids," in *ACM SIGGRAPH 2004 Papers on - SIGGRAPH '04*, Los Angeles, California, 2004, p. 769, doi: 10.1145/1186562.1015799.

[6] A. Frasson, T. A. Engel, and C. T. Pozzer, "Efficient Screen-space Rendering of Vector Features on Virtual Terrains," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, New York, NY, USA, 2018, pp. 7:1–7:10, doi: 10.1145/3190834.3190851.

[7] J. Pouderoux and J.-E. Marvie, "Adaptive streaming and rendering of large terrains using strip masks," in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia - GRAPHITE '05*, Dunedin, New Zealand, 2005, p. 299, doi: 10.1145/1101389.1101452.

[8] F. Cellier, P.-M. Gandoin, R. Chaine, A. Barbier-Accary, and S. Akkouche, "Simplification and streaming of GIS terrain for web clients," in *Proceedings of the 17th International Conference on 3D Web Technology - Web3D '12*, Los Angeles, California, 2012, p. 73, doi: 10.1145/2338714.2338726.

[9] P. Herzig, M. Englert, S. Wagner, Y. Jung, and U. Bockholt, "X3D-EarthBrowser: visualize our earth in your web browser," in *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13*, San Sebastian, Spain, 2013, p. 139, doi: 10.1145/2466533.2466556.

[10] S. Deb, P. J. Narayanan, and S. Bhattacharjee, "Streaming terrain rendering," 2006, p. 181, doi: 10.1145/1179849.1180075.

[11] Jos Timanta Tarigan, Rahmat Widia Sembiring, Maya Silvi Lydia, Opim Salim Sitompul, Mahyuddin K.M. Nasution, and Muhammad Zarlis, "Application Architecture for Collaborative Terrain Editing," in *Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)*, Malaysia, Apr. 2017.

[12] M. Nasution, J. Tarigan, I. Jaya, S. Hardi, and S. Sitorus, "Collaborative 3D terrain editing application," *Int. J. Eng. Technol.*, vol. 7, pp. 57–60, Jan. 2018, doi: 10.14419/ijet.v7i4.40.24075.

[13] "terrain.party." https://terrain.party/ (accessed Mar. 10, 2020).