Dynamic Terrain Data Exchange in a Collaborative Terrain Editor

Jos Timanta Tarigan[1*], Opim Salim Sitompul[1], Muhammad Zarlis[2], and Erna Budhiarti Nababan[1]
E-mail: jostarigan@usu.ac.id, opim@usu.ac.id, muhammad.zarlis@binus.edu, ernabrn@usu.ac.id
[1] Faculty of Computer Science and Information Technology, Universitas Sumatera Utara Medan, Indonesia
[2] Information Systems Management Department, BINUS Graduate Program - Master of Information Systems Management, Bina Nusantara University, Jakarta, Indonesia

Keywords: Dynamic terrain streaming; multimedia communication, communication protocol; computer supported cooperative work

Received: September 21, 2022

In a computer supported cooperative work (CSCW), data consistency between collaborating users is a crucial issue. Based on the type of the application, ensuring data consistency can be a lengthy process that takes time and affects the system's performance. In most 3D application, terrain data are massive due to its size. Exchanging this data may be expensive and may cause significant delay. In a real-time collaborative terrain editor, this issue becomes more significant due to terrain data exchange is consistently occurred between collaborating users. We present a solution to perform a conflict-free dynamic terrain data exchange in a real-time collaborative terrain editor. Our objective is to develop a method that able to ensure data consistency amongst collaborating peers in real-time manner. The main idea of our method is to split the terrain into smaller patches and synchronize the changes efficiently by only exchanging the modified patches. We applied our solution to a collaborative terrain editor application to test its performance in a real-time collaborative editing session. The tests were done in multiple scenarios, using different patch model, brush size (in the terrain editor), and connection setup between server and collaborating clients. The result shows that our protocol is capable to maintain data consistency between collaborating clients in a real-time terrain edition session. The delay is varied and highly depends on the data size and client-server environment setup. The overall test shows that it is possible to perform a collaborative terrain editing with an acceptable response time delay. In this paper, we present our proposed method, the implementation, and the result data from the test.

Povzetek: V prispevku je opisana metoda za sprotno izmenjavo podatkov pri opisu dinamičnega terena.

1 Introduction

In modern industry, the use of information technology to support collaborative works has become a vital component to increase productivity [1], [2]. The concept of Collaborative Virtual Environment as a computer-based system where users are allowed to collaborate within computer-based context has been used extensively since the early 90s with the introductory of internet to the public [3]. However, the use of Computer Supported Collaborative Work (CSCW) may face several issues such as data consistency amongst collaborators [4].

In a Cloud-Based Collaborative Design [5], [6] data exchange can be a significant issue due to complexity of the data. Based on the application, there are various aspects that needs to be considered when performing data exchange amongst collaborators. In real-time collaboration scenario, data exchange requires additional time and may significantly affect the interactivity of the system. In 3-dimension (3D) design application, interactivity is a major issue since it may affect user's performance. Delay between user's input and system's response must be minimized to avoid noticeable delay. Hence, it is necessary to minimize this issue by using an optimal protocol optimally designed for this task.

In our previous research, we developed an application that allows multiple users to perform 3D terrain editing in real-time called Collaborative Terrain Editor [7], [8]. The application architecture requires terrain data transfer amongst collaborating users. We noticed an issue during the development that performing massive data exchange cause delay in response time and might raise an interactivity issue. Moreover, ensuring data validity amongst clients might also raise additional issue. In this paper, we propose a model to this issue by developing a method to exchange terrain data in a collaborative terrain editing application. Our model is specifically designed for a real-time collaborative application and is optimized for a specific type of 3D content, dynamic terrain. We implemented our solution in CTE to test its validity and performance.

We proposed a network protocol that can efficiently transfer dynamic terrain data while ensuring data synchronization amongst collaborating users. Our solution is implemented as a communication protocol. To test the performance of the proposed solution, we implemented our protocol in a collaborative 3D terrain editor. The paper is structured as follows: we first describe the outline of the Collaborative Terrain Editor, the data representation and communication, and the synchronization mechanism between users. The second part of this paper will describe the problem that occur during the synchronization process and propose a solution to tackle the problem. Finally, we will describe the method proposed in this paper and show how our method can decrease the problem.

2 Related works

2.1 Collaborative 3D modeling

There are numerous works that has been conducted to study the concept collaborative 3D modeling. Ha et al. introduced Lets3D, a 3D editing tool that allows multiple users to collaborate in real-time [9]. Imae and Hayashibara developed ChainVoxel, a collaborative editing of voxel-based 3D models [10]. Other works also provides a solution to perform a collaborative 3D modeling in a specific case and/or environment such as interior design [11], avatar (gesture and emotion) [12], virtual reality/spaces [13, p.], [14], [15], co-located collaborators using a tabletop system [16], and to support multidisciplinary 3D product CAD modeling [17].

In manufacturing industry, Cloud-Based Collaborative Design has been explored and commonly implemented in modern industry. This paradigm allows users to collaborate on a cloud-based system. One of the most common media to exchange the design data is to use Feature-Based Data Exchange (FBDE). The idea of FBDE is to share information regarding the modeling procedure such as history, constraints, parameters, and features [18] instead of the model. In a Cloud-Based Design and Manufacturing (CBDM) environment, the use of FBDE is common to allow multiple peers sharing Computer Aided Design (CAD) data [19], [20]. There are also various researches focus on extending the capability of FBDE such as security [21], collaboration [22], undo mechanism [23]. AR/VR/MR [24], and common 3D-information such as Buidling Information Modeling (BIM) technology [25], [26].

2.2 Terrain representation and streaming

Most 3D applications contain massive and detailed 3D terrain. Hence, storing terrain data as a common 3D object with vertices in 3-dimensional space could be expensive. There are numerous methods invented to store 3D terrain efficiently. One of the most common method to represent terrain is using uniform grid called heightfield or heightmap. This method assumes terrain as a 2-dimensional image with the position of each pixel represents the location and its color represents its height.

While heightfield is simple and robust, it can be extremely redundant in a flat area due to the data contains multiple repetitive value. There are several methods to solve this issue, either by simplification or compression. Simplification methods focus on reducing the terrain data while preserving it shapes. One most notable method is to manage the Triangulated Irregular Network (TINs). Unlike regular grid which contains points sampled at equal distance, TINs allow the amount of data sampled in an area to adapt based on the complexity of the terrain. One interesting feature to consider in developing a terrain representation model is to apply a deformable terrain. This feature introduces a new challenge since deforming a terrain requires data manipulation which may be expensive in a real time system. There are various works that proposed a solution for real-time terrain deformation/modification [27], [28]. Additionally, there are also various works on terrain representation that focus on decreasing terrain data size [29] and increasing data streaming performance [30].

A more related subject to our work is the concept of streaming a dynamic terrain. As opposed to static terrain, dynamic terrain allows its data to be modified based on a certain event. Streaming a dynamic terrain may introduce a new issue, data synchronization. When multiple users are capable to modify the terrain data, there should be a protocol to ensure that each user holds the same terrain data. Elis et al. developed a multi-user 3D battle simulation with a deformable terrain [31]. In the simulation, users are capable to deform the terrain by performing a certain action. In their architecture, multiple computers are acted as servers. Clients will then connect to a specific server based on the configuration. Each action made by the client will be processed by the corresponding server. The server will then collaborate with other servers to synchronize the data. Another similar work to our research is proposed by Mendoza et al. [32] which proposed an architecture for collaborative terrain sketching with mobile device. However, the solution proposed by their work for data sharing is similar to the one proposed by Ellis et al.; instead of distributing the modified mesh data, the system distribute the state change or editing operation messages.

2.3 Collaborative terrain editor

Our system is built based on Collaborative Terrain Editor (CTE) [7], [8], a 3D terrain editor application that allows multiple users to perform real-time collaboration. The application is intended to allow multiple users to collaborate a terrain in real-time manner. Fig. 1 shows the basic interface of CTE.

The client side of the system is for the user/editor. It lets users to perform basic terrain editing using a brush-like tool that changes the elevation of the map in a certain area based the size and shape of the brush. Additionally, user also able to add noise feature that will add random details on the terrain. The server side of the system is a console-based application. Its role is to accept users' input from connected clients, perform the changes to the terrain, and send the modified terrain data back to the client. Collaborating users must be connected to the server. All terrain data is kept on the server.

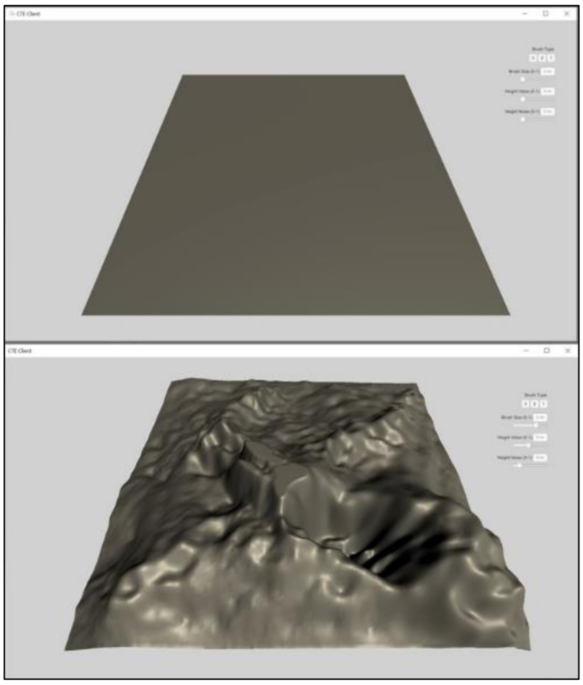


Figure 1: The Interface of Collaborative Terrain Editor

The system is built with thin-client-server architecture; the terrain deformation calculation is performed by the server. This design is intended so the computational cost of modifying the terrain can be done by the server. However, this design requires server to distribute dynamic terrain data. We have compared similar research that propose the same idea. Ellis et al. [31] shares a similar solution for multi-user dynamic terrain distribution system. While the requirement is similar, the network architecture design is different. The system by Ellis et al. relies on clients to compute the terrain data changes. Thus, the server only requires to distribute user's action instead of terrain data. Mendoza et al. [32] also develop a multi-user terrain editing system that relies on AR. Multiple users can interact by using mobile phones and tablet to edit and observe the same 3D terrain. Their protocol, however, is similar to Ellis et al. and relies on broadcasting user's action to collaborating users. There is not terrain data transfer during the editing process. Several previous works on 3D terrain streaming are also not compatible with our system as they are dealing with static terrain data [30], [33], [34].

3 Proposed method

We develop our solution based on the architecture of CTE described in the previous section. The problem that we try to solve can be summarized in this description: how to perform data exchange that ensure the synchronization of terrain data while maintaining the interactivity of the system in a client-server based collaborative terrain editing session. Our proposed solution consists of two main parts: the representation of the terrain data that consist of terrain segmentation and compression, and the communication protocol.

3.1 Terrain data representation

Our terrain representation is using tiling system that is commonly used in large terrain representation to either optimize data in memory/storage or increase data transfer performance in a networked system. In our case, the latter is an important factor since data communication is crucial in collaborative system [35].

The tiling system divides the terrain into smaller uniform tiles (we will be using the term patch(es) instead). Each of these patches contains an identification value that defines the position of the patch. Additionally, we also perform data compression to decrease the terrain data in order to minimize the transfer delay. While we use 16-bit heightfield to render the terrain, we truncate the data into 8-bit value during the transfer process. To minimize the error caused by the compression, the 8-bit data is quantized relative to the minimum and maximum value of each patch. We argue that values in each patch tends to have a similar or slightly varied, thus reducing it to 8-bit will not cause a significant error.

Another issue that needs to be addressed is the data consistency amongst clients and server. In a real-time collaborative system, each peer must be capable to validate data consistency and perform data synchronization if required. These actions must be performed with minimum time frame to maintain user interactivity. To do this, we developed a method using a sequence number (`seqNumber`) which will be discussed in the next section.

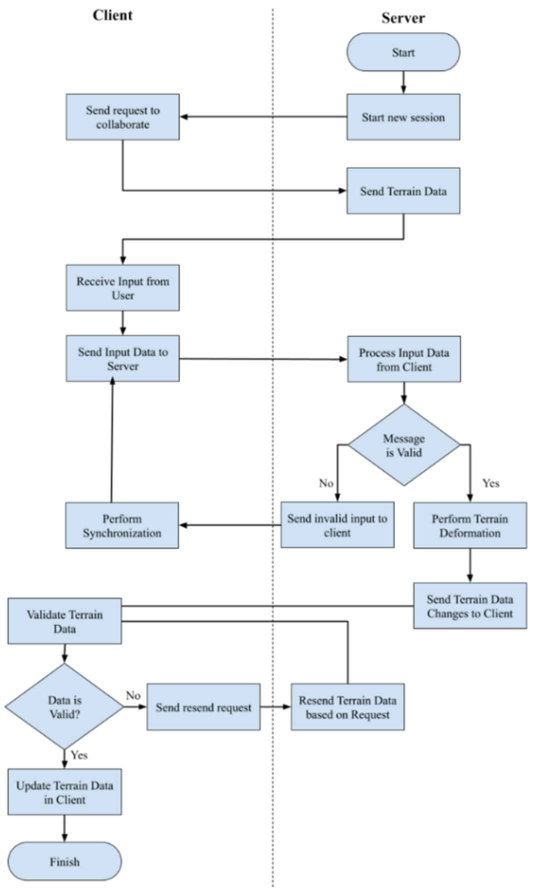
Based on the previous description, a patch in our model contains `patchId` (4 bytes integer), (4 bytes integer), `minValue` and `maxValue` (2 bytes integer/short each), followed by the compressed terrain data (16 bytes, 64 bytes, and 256 bytes char for 4×4, 8×8, and 16×16 respectively). Therefore, the total size of each patch, including the header and terrain data, in model 4×4, 8×8, and 16×16 are 28 bytes, 76 bytes, and 268 bytes respectively.

3.2 Protocol overview

When multiple collaborating clients involved in a session, unsynchronized data can be an issue. Changes from one client may overlap with changes from others. Hence, we developed Patch Sequence Number Method to tackle this issue. Each patch in the terrain is embedded with a single unique integer value called sequence number (`seqNumber`). When a patch is modified by the server, it gets the maximum sequence number of the terrain increased by one. Hence, every patch has a unique value, and the last updated patch has the highest value. Server keeps the highest value to track with the latest update. Simultaneously, the client also keeps the highest value it has received during the data transfer. Therefore, it is guaranteed that if the values owned by client and server is different, data synchronization is required.

Additionally, the client could use this sequence number to detect missing data/patches. When the client received the data from the server, it sorts the sequence number of the incoming patches. If the data is complete, there should be no missing values between the smallest and the highest value. However, if there is a missing data, the client can simply find these missing values and request the corresponding patches from the server.

The communication protocol is intended to distribute the terrain changes between server and multiple clients. It is built specifically for our terrain representation, relying on the sequence number on each patch to distribute the terrain data and, if necessary, perform data synchronization. The collaboration session started initialized the session. Clients then send a request to join the session. Upon entering the session, server send the current terrain data to the client in patches (including the terrain metadata). If the terrain data is valid (with no missing or invalid patches detected), the client will generate the terrain and render it on the screen for the user to interact. If otherwise, the client sends a resend request to the server. When the user performs an input to alter the terrain, the client sends the input data to the server. The server validates the input data (by making sure that the content received by the client is up to date), and if it is valid, the server will perform the changes according to the user's input. These changes are then distributed to connected clients. The overview of CTE communication protocol is shown in Figure 2.



Client
Server
Start
Start new session
Send request to collaborate
Send Terrain Data
Receive Input from User
Send Input Data to Server
Process Input Data from Client
Message is Valid
No
Yes
Perform Synchronization
Send invalid input to client
Perform Terrain Deformation
Send Terrain Data Changes to Client
Validate Terrain Data
Data is Valid?
No
Send resend request
Resend Terrain Data based on Request
Yes
Update Terrain Data in Client
Finish

Figure 2: The flowchart of the proposed method

Based on the protocol overview and the sequence number described earlier, we developed a communication protocol sequences diagram as shown in Figure 3 for unsynchronized (left) and synchronized client (right). In this protocol, each request and response are started by a two-digit character as keyword that defines the type of the data received. Both sequences started with a collaborating client sent editing data to the server. The client wraps the editing data and add the sequence number it currently holds. This value is the highest sequence number it holds and defines the last update that the client has received from the server. This data will then be transmitted with a keyword UE (User Edit). When the server received the packet, it will evaluate the validity of the request by examining the sequence number it received from the client. If the sequence sent by the client is different (smaller) than the value owned by the server, then the client is not synchronized. The server will then send a message EI (Edit Invalid) followed by the correct sequence number. Upon receiving this message, the client waits for the synchronization process. The server will then find all the patches with sequence number larger than the client's number and initiate a synchronization process by sending a TS (Terrain Synchronization) message followed by the list of numbers in the patches that are going to be sent during this process. These patches are then sent to the client by using the keyword SD (Synchronizing Data). The last patch is sent using the keyword SF (Synchronization Finished). During this process, the client updates the sequence number using the highest value from the received patch. When this synchronization process is performed, the server applies the editing data received earlier and modify the terrain data accordingly.

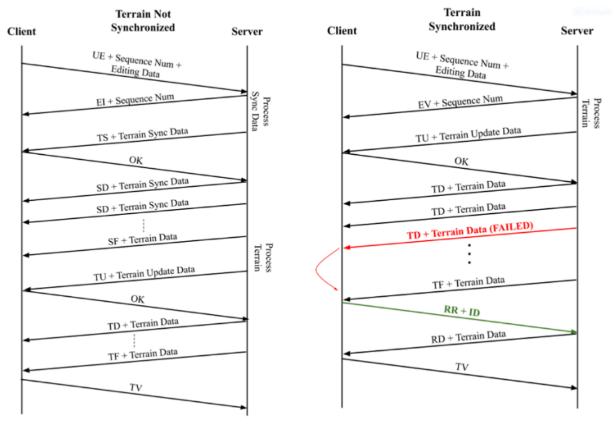


Figure 3: Network protocol diagram of the proposed method

When both client and server are synchronized, the server will proceed to process the update sent earlier by the client. When the update has been implemented, the server will send the updated patches (with the updated sequence numbers) to the client. Prior to sending the update data, the server will send a notification to the client with a keyword TU (terrain update) followed by update metadata (total patches, author's client ID, and update time). The client will then response with an OK notification and the server may proceed to send the updated patches with the keyword TD.

During transmission, there is a possibility that the update data was not delivered successfully during the transmission (as shown as the red line in the sequence diagram). The client acknowledges this issue when there are missing patches sequence number. Updated patches contain new sequence numbers, and these values are sequential. Hence, when the client receives the update, it can detect the missing patches by sorting them based on their sequence number. If the client detects a missing value, it can request a resend (with the keyword RR) followed by the missing number. The server will respond by sending the patch based on the request using the keyword RD (Resend Data). When all the updated data is delivered, sorted, and successfully implemented on the client side, the client will send a keyword TV (terrain valid) notifying the server that the data transmission has been successfully delivered.

4 Result and discussion

To test our proposed method, we attached it as part of the protocol in Collaborative Terrain Editor (CTE). We have successfully implemented our method in the application and ensure that the protocol able to support real-time collaborative terrain editing from multiple devices (in our tests, we use 3 clients connected to 1 server). Based on the test result, we noticed that our method is capable to ensure synchronized data amongst user. However, our objective is to measure the performance of the method. Hence, we performed various tests using our protocol. We also use a few different settings combinations to find the optimal settings. The first setting is the size of the terrain that needed to be transmitted. We simulate this by assigning inputs with various sizes, assuming that the server will responds by sending terrain update with the same size. The second setting is the size of each patch. In the test, we use three different sizes of patch: 4×4, 8×8, and 16×16. The third setting is the client-server environment. We use different server settings to measure how the system perform in various networking environment and how the server configuration may affect the system's performance. To measure the performance of our proposed method, we use two parameters: the system's response time (in milliseconds) and the size of transmitted data (in bytes). Additionally, since the size of patch may affect the error caused by the compression, we will also gather the error rate of each model.

4.1 Data compression performance

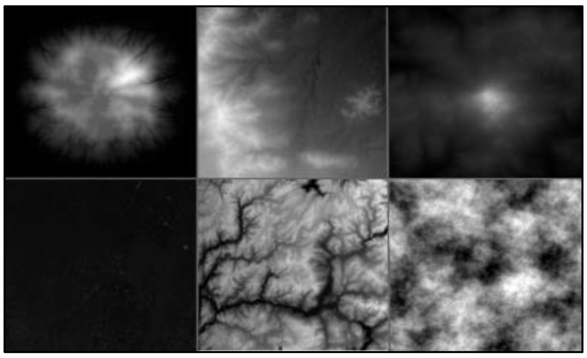


Figure 4: The heightfield images used in the Test

The first test is to observe the error rate of our terrain representation caused by the compression. We perform the test by comparing the original 16-bit terrain (with value ranged from 0 to 65,536) with the compressed 8-bit terrain (with values ranged from 0 to 256). The comparison is performed on 6 different heightmap with different characteristics and features which can be seen in Figure 4 (top: 1. island, 2. mountain range, 3. Hill; bottom: 4. Urban area, 5. riverbank, and 6. noise-generated terrain).

We collected 3 variables to measure the error rate of each heightmap. We assume a heightmap with n points where p_i is the value of point with index i in the original 16-bit heightmap and p_i' is the value of the same point in the compressed 8-bit patch. The first variable is the average difference (AVGDIF). This variable represents the average difference of all the points in the map which can give us a thorough view on the overall error. The average difference can be calculated as follows.

$$AVGDIF = \frac{\sum_{i=0}^{n} |p_i' - p_i|}{n}$$

The second variable is the average maximum (AVGMAX) which represents the average maximum difference of all patches. This variable gives a thorough observation regarding the maximum error among all patches caused by the compression. Given the maximum difference between original and compressed value in patch j is $max(|p' - p|)_j$, hence, the average maximum of an heightfield that contains m patches can be calculated as follows.

$$AVGMAX = \frac{\sum_{j=0}^{m} max(|p' - p|)_j}{m}$$

The third variable is the maximum difference (MAXDIF) which represents the maximum difference between the original and the compressed point in the heightfield. The maximum difference can be calculated using this formula.

$$MAXDIF = max(|p' - p|)_i$$

Table 1 shows the error rate collected during the compression test. The error-rate test result shows that error caused by the compression is minimum. In the first 4 heightmaps, average difference is 1 to 2 units (from a range of 0 to 65.535) when using 4×4 and 8×8 model. The average maximum values are also relatively small compared to the value range. In terrain 5 and 6, however, the difference increased significantly due to the high frequency of the map. This pattern occurred throughout the test where the 4×4 model gives the least error values, followed by 8×8 and 16×16, and terrain with high frequency gives a worse result. While the value difference is minimal, it is important to notice that in most of the test, most of the points were changed (shown by a high percentage difference). Nevertheless, based on direct observation on the terrain, the pattern of the terrain persists after the compression.

Table 1: Data compression performance result

HF	Model	AVGDIFF	AVGMAX	MAXDIF
	4×4	1.69	1,362	16
1	8×8	1.93	3,857	31
	16×16	4.19	7,916	45
	4×4	1.06	2,417	14
2	8×8	2.53	5,234	21
	16×16	4.77	9,605	30

3	4×4	1.02	1,084	11
	8×8	1.18	2,476	25
	16×16	2.47	5,007	42
4	4×4	1.13	1,717	21
	8×8	1.46	3,062	23
	16×16	2.22	4,504	24
5	4×4	4.02	8,974	66
	8×8	9.38	19,354	96
	16×16	16.54	33,509	107
6	4×4	4.09	9,191	29
	8×8	9.04	18,655	44
	16×16	16.11	32,505	66

4.2 Response time

In the second test, we collected the response time data of the system after applying our protocol. The response time is measured from the time the first data is sent from the client to the server until the last data is received and validated by the client. Since the data must be valid, the response time also includes the synchronization process during the transmission.

The test was performed in 3 different cases based on the connection and distance between the client and server: local area network-based environment (LAN) and two internet-based networks with different server location, Singapore (SG) and United States (US). We also use a different server specification to observe whether the hardware affect the overall response time. Both SG2 and US2 has twice the CPU and memory specification compared to SG1 and US1. We also perform by using three kind of different brush sizes: small, medium, and large for brush with diameter of 5, 10, and 15 respectively. Additionally, we also test 3 different patch models to find the patch size with the best performance. We perform the test 10 times for each scenario and collected 2 response time data: average and maximum.

In the first test, we connect 3 collaborating users to the server and one of the users performing terrain editing while the other two simply receiving the data. Table 2 shows the result of our response time test of the first test. All data is presented in millisecond.

Table 2: Response time from the first test result

	Small		**Medium**		**Large**	
	Avg.	**Max**	**Avg.**	**Max**	**Avg.**	**Max**
LAN						
4×4	6	8	6	9	10	12
8×8	4	8	5	10	8	15
16×16	5	8	5	9	8	14
SG1						
4×4	94	167	108	152	139	140
8×8	73	177	103	144	102	142
16×16	71	125	95	154	90	108
SG2						
4×4	65	78	70	83	99	129
8×8	51	79	53	82	67	104
16×16	50	80	58	98	65	111
US1						
4×4	362	391	372	501	426	372
8×8	320	380	334	622	380	426
16×16	293	319	356	541	382	495
US2						
4×4	322	385	314	336	401	311
8×8	289	345	288	362	363	532
16×16	255	319	267	284	326	505

The result shows that server's round-trip time is the main contribution to the delay. Internet-based test significantly higher than LAN-based test and the US-based server gave the highest response time compared to the other test. The overall result from LAN-based test produced less than 10 milliseconds response time. In result, the users did not notice any delay during the editing and responded positively. The first internet-based test using server located in Singapore gave a significant delay increase up to 130 milliseconds. While the delay is increased significantly, the application itself is still usable and the user were able to perform editing normally. The US-based test however, affected the user's capability due to the high response time. Most of the users argue that this delay makes the editor feels unresponsive.

In the second test, we asked 2 connected users to perform terrain editing concurrently and continuously. This test is aimed to observe how concurrent data input might affect the performance. Table 3 shows the results of the second test.

Table 3: Response time from the second test result

	Small		**Medium**		**Large**	
	Avg.	**Max**	**Avg.**	**Max**	**Avg.**	**Max**
LAN						
4×4	8	11	8	11	11	13
8×8	8	10	9	11	10	15
16×16	9	11	9	11	10	15
SG1						
4×4	102	191	120	167	164	201
8×8	89	190	142	171	175	193
16×16	100	195	123	177	145	190
SG2						
4×4	85	102	82	112	132	153
8×8	78	101	79	128	126	147
16×16	91	112	81	125	132	149
US1						
4×4	521	555	601	821	701	951
8×8	495	581	590	794	658	857
16×16	455	572	611	801	700	1016
US2						
4×4	501	591	511	599	561	912
8×8	477	568	498	581	551	786
16×16	481	601	407	600	583	1112

As expected, there was a significant increase in response time especially on the internet-based test when multiple users concurrently perform terrain editing. The increase is varied based on the behaviour of the editing process. While the LAN-based setup still has a relatively low delay time, the internet-based setup becomes significantly noticeable and affected the application interactivity. We also noticed that in some cases when the users editing the same area continuously, the delay reached 1 seconds and the users responds negatively to this delay. However, the data also shows that hardware boost were able to reduce the response time better than the previous test. The SG2 and US2 on the second test able to reduce the delay time up to 50% compared to SG1 and US1.

5 Conclusion and future works

In this paper, we proposed a solution to perform dynamic data exchange in a client-server environment. The protocol guarantees that the data is synchronized amongst peers. Moreover, the protocol is optimized so the data transfer and synchronization process can be performed efficiently to reduce the data and time required to transfer the terrain data.

We tested the validity and performance of our protocol by attaching it to a real-time collaborative terrain editing system, CTE. Based on our test, the protocol is capable in maintaining data synchronization between connected peers. The performance test also shows that the proposed method able to perform terrain data distribution efficiently based on the response time tests in multiple scenarios depending on the amount of data and the connection between client and server.

While our current solution works as expected, it still opens for further optimization and expansion. Our current focus is to increase the compression performance considering there are numerous previous research focused on heightfield compression. Our main issues to implement a better compression are the complexity of dynamic terrain data and the real time requirement of the system. Additionally, we would also like to expand the possibility in using the proposed method in other application that require dynamic terrain data synchronization. We are confident that our method, with slight modification, is applicable to different cases that face similar issues. We are interested in testing our protocol in other application such as game engine, battle simulation, or GIS.

Reference

[1] Y. O. de Lima and J. M. de Souza, "The future of work: Insights for CSCW," 2017, pp. 42–47. https://doi.org/10.1109/CSCWD.2017.8066668

[2] W. Reinhard, J. Schweitzer, G. Volksen, and M. Weber, "CSCW tools: concepts and architectures," *Computer*, vol. 27, no. 5, pp. 28–36, 1994, https://doi.org/10.1109/2.291293

[3] C. Greenhalgh, "Large Scale Collaborative Virtual Environments," University of Nottingham, Nottingham, 1997. Accessed: Apr. 07, 2018. [Online]. Available: https://pdfs.semanticscholar.org/e505/12849626f01537b6e1542ee6867b60db6595.pdf

[4] J. Grudin, "Why CSCW applications fail: problems in the design and evaluationof organizational interfaces," in *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, United States, 1988, pp. 85–93. https://doi.org/10.1145/62266.62273

[5] G. Andreadis, G. Fourtounis, and K.-D. Bouzakis, "Collaborative design in the era of cloud computing," *Advances in Engineering Software*, vol. 81, pp. 66–72, Mar. 2015, https://doi.org/10.1016/j.advengsoft.2014.11.002

[6] S. Sharma, F. Segonds, N. Maranzana, D. Chasset, and V. Frerebeau, "Towards Cloud Based Collaborative Design – Analysis in Digital PLM Environment," in *IFIP International Conference on Product Lifecycle Management*, 2018, pp. 261–270.

[7] M. Nasution, J. Tarigan, I. Jaya, S. Hardi, and S. Sitorus, "Collaborative 3D terrain editing application," *International Journal of Engineering and Technology(UAE)*, vol. 7, pp. 57–60, Jan. 2018, https://doi.org/10.14419/ijet.v7i4.40.24075

[8] J. T. Tarigan, R. W. Sembiring, M. S. Lydia, O. S. Sitompul, M. K. M. Nasution, and M. Zarlis, "Application Architecture for Collaborative Terrain Editing," in *Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)*, China, 2017. https://doi.org/10.18178/wcse.2017.06.092

[9] Y.-U. Ha, J.-H. Jin, and M.-J. Lee, "A Robust Collaborative 3D Editing Tool Utilizing Distributed Consensus Protocol," in *Advanced Science and Technology Letters*, 2015, vol. 117, pp. 57–60. https://doi.org/10.14257/astl.2015.117.13

[10] K. Imae and N. Hayashibara, "ChainVoxel: A Data Structure for Scalable Distributed Collaborative Editing for 3D Models," in *2016 IEEE 14th Intl Conf on DASC/PiCom/DataCom/CyberSciTech*, Aug. 2016, pp. 344–351. https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016.75

[11] M. Steiakaki, K. Kontakis, and A. Malamos, "Real-Time Collaborative Environment for Interior Design based on Semantics, Web3D and WebRTC," in *Proceedings of the 15th International Symposium on Ambient Intelligence and Embedded Systems*, Greece, 2016

[12] R. Klauck, S. Lorenz, and C. Hentschel, "Collaborative work in VR Systems: A software-independent exchange of avatar data," in *2016 IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2016, pp. 133–136. https://doi.org/10.1109/ICCE-Berlin.2016.7684738

[13] C. Gadea, D. Hong, D. Ionescu, and B. Ionescu, "An architecture for web-based collaborative 3D virtual spaces using DOM synchronization," in *2016 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, Jun. 2016, pp. 1–6. https://doi.org/10.1109/CIVEMSA.2016.7524313

[14] B. Lee, X. Hu, M. Cordeil, A. Prouzeau, B. Jenny, and T. Dwyer, "Shared Surfaces and Spaces: Collaborative Data Visualisation in a Co-located Immersive Environment," *IEEE Trans. Visual. Comput. Graphics*, vol. 27, no. 2, pp. 1171–1181, Feb. 2021, https://doi.org/10.1109/TVCG.2020.3030450

[15] D. Mechta, S. Harous, and M. Djoudi, "Tele-Collaboration System in CVLab," *IJCAI*, vol. 46, no. 2, Jun. 2022, https://doi.org/10.31449/inf.v46i2.3205.

[16] B. Ens *et al.*, “Uplift: A Tangible and Immersive Tabletop System for Casual Collaborative Visual Analytics,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1193–1203, Feb. 2021, https://doi.org/10.1109/TVCG.2020.3030334.

[17] Y. Bathla and S. Szenasi, “A Web Server to Store the Modeled Behavior Data and Zone Information of the Multidisciplinary Product Model in the CAD Systems,” *IJCAI*, vol. 44, no. 2, Jun. 2020, https://doi.org/10.31449/inf.v44i2.2660.

[18] Y. Wu, F. He, D. Zhang, and X. Li, “Feature-based data exchange as Service for Cloud Based Design and Manufacturing,” in *Proceedings of 2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, May 2015, pp. 594–599. https://doi.org/10.1109/CSCWD.2015.7231025.

[19] F. Tao, L. Zhang, V. Venkatesh, Y. Luo, and Y. Cheng, “Cloud manufacturing: A computing and service-oriented manufacturing model,” *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 225, Nov. 2011, https://doi.org/10.1177/0954405411405575.

[20] X. Wu, F. Qiao, and K. Poon, “Cloud manufacturing application in semiconductor industry,” in *Proceedings of the 2014 Winter Simulation Conference*, Savannah, Georgia, Dec. 2014, pp. 2376–2383.

[21] Yiqi Wu, Fazhi He, and Yueting Yang, “A Grid-Based Secure Product Data Exchange for Cloud-Based Collaborative Design,” *IJCIS*, vol. 29, 2020, https://doi.org/10.1142/S0218843020400067.

[22] D. French, E. Red, A. Hepworth, C. Jensen, and B. Stone, “Multi-User Computer-Aided Design and Engineering Software Applications,” in *Cloud-Based Design and Manufacturing (CBDM): A Service-Oriented Product Development Paradigm for the 21st Century*, 2014, pp. 25–62. https://doi.org/10.1007/978-3-319-07398-9_2.

[23] Y. Cheng, F. He, B. Xu, S. Han, X. Cai, and Y. Chen, “A multi-user selective undo/redo approach for collaborative CAD systems,” *Journal of Computational Design and Engineering*, vol. 1, no. 2, pp. 103–115, Apr. 2014, doi: https://doi.org/10.7315/jcde.2014.011.

[24] P. Wang *et al.*, “A comprehensive survey of AR/MR-based co-design in manufacturing,” *Engineering with Computers*, vol. 36, Oct. 2020, https://doi.org/10.1007/s00366-019-00792-3.

[25] H. Fan, B. Goyal, and K. Z. Ghafoor, “Computer-aided architectural design optimization based on BIM Technology,” *IJCAI*, vol. 46, no. 3, Sep. 2022, https://doi.org/10.31449/inf.v46i3.3935.

[26] J. Feng, Z. Zhang, Y. Xu, and A. Zhang, “Intelligent engineering management of prefabricated building based on BIM Technology,” *IJCAI*, vol. 46, no. 3, Sep. 2022, https://doi.org/10.31449/inf.v46i3.4047.

[27] Y. Xia, Y. Chen, and D. Wang, “Real-Time LOD Rendering of Tire Tracks in Dynamic Terrain,” in *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, Oct. 2019, pp. 206–209. https://doi.org/10.1109/EITCE47263.2019.9095098.

[28] J. Svensson, *REAL-TIME RENDERING OF DEFORMABLE SNOW COVERS*. 2019. Accessed: Jun. 16, 2022. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-165167

[29] Z. Ge and W. Li, “Geometry compression method for terrain rendering with GPU-based error metric,” in *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry - VRCAI ’11*, Hong Kong, China, 2011, p. 387. https://doi.org/10.1145/2087756.2087823.

[30] 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. https://doi.org/10.1145/2338714.2338726.

[31] C. Ellis, P. Babenko, B. Goldiez, J. Daly, and G. A. Martin, “Dynamic Terrain for Multiuser Real-Time Environments,” *IEEE Comput. Grap. Appl.*, vol. 30, no. 1, pp. 80–84, Jan. 2010, https://doi.org/10.1109/MCG.2010.5.

[32] S. Mendoza, A. Cortés-Dávalos, L. M. Sánchez-Adame, and D. Decouchant, “An Architecture for Collaborative Terrain Sketching with Mobile Devices,” *Sensors (Basel)*, vol. 21, no. 23, p. 7881, Nov. 2021, https://doi.org/10.3390/s21237881.

[33] P.-C. Wang, A. I. Ellis, J. C. Hart, and C.-H. Hsu, “Optimizing next-generation cloud gaming platforms with planar map streaming and distributed rendering,” Jun. 2017, pp. 1–6. https://doi.org/10.1109/NetGames.2017.7991544.

[34] S. Petrangeli, G. Simon, H. Wang, and V. Swaminathan, “Dynamic Adaptive Streaming for Augmented Reality Applications,” in *2019 IEEE International Symposium on Multimedia (ISM)*, Dec. 2019, pp. 56–567. https://doi.org/10.1109/ISM46123.2019.00017.

[35] J. T. Tarigan, O. S. Sitompul, M. Zarlis, and E. B. Nababan, “Multi Patch 3D Terrain Representation for Collaborative Terrain Editor,” *J. Phys.: Conf. Ser.*, vol. 1566, p. 012116, Jun. 2020, https://doi.org/10.1088/1742-6596/1566/1/012116.