Adaptive 3D Texture Streaming in M3G-based Mobile Games

Mohammad Hosseini
School of Computing Science
Network Systems Lab (NSL)
Simon Fraser University, Canada
mohammad_hosseini@sfu.ca

Dewan T. Ahmed
Software Engineering Department
College of Computer and
Information Sciences (CCIS)
King Saud University, KSA
dtahmed@ksu.edu.sa

Shervin Shirmohammadi
Distributed and Collaborative
Virtual Environment Research
(DISCOVER) Lab
University of Ottawa, Canada
shervin@discover.uottawa.ca

ABSTRACT

With the growing demand of mobile applications and games, one of the challenges is how to efficiently transmit the bulky 3D information to bandwidth- and computationally-limited mobile devices. In this paper, we propose two methods for improving the transmission delay of M3G-based 3D mobile game content over unreliable and congested networks. We introduce Object Mesh Similarity as a server side approach, in which we try to detect an alternative object with minimum complexities that is similar to the original object, and then transmit this reconstructed object, as well as Texture Stretching as a client-side approach, which leads to the efficient receipt of object textures. Our results show 35% to 70% improvement in the transmission delay of 3D textures.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: General – games; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism.

General Terms

Algorithms, Design.

Keywords

M3G; mobile games; 3D Streaming; Object Mesh Similarity; Texture Stretching.

1. INTRODUCTION

There has been a significant rise in 3D mobile applications such as games and 3D virtual environments. It was estimated that 78.6 million people in the U.S. alone played mobile games in 2009, downloads of mobile games increased tenfold compared to 2003, and mobile games generated more than $1.5 billion annually in revenue [1]. In addition, due to progress in the hardware of mobile devices, these devices can now support 3D graphics, many in fact able to use hardware acceleration and GPU based support of popular 3D formats such as OpenGL. The use of a standard 3D format such as OpenGL has opened the door for a large population of existing developers to make a profit by selling 3D applications and games on mobile devices. One of the APIs supporting OpenGL on mobile devices is the Mobile 3D Graphics API for Java; i.e. M3G, also known as JSR 184. It is an optional API tailored specifically for the generation and presentation of 3D content on mobile platforms. In essence, M3G is a high-level abstraction over OpenGL ES (JSR-239, another optional package) [2, 3, 4]. Due to the limitations in network speed, memory size, and computation resources of mobile devices, a good 3D mobile game must balance view, visual effects, and performance. One of the challenges to achieve this balance is to efficiently handle 3D object textures. While 3D wireframe models can be transmitted relatively quickly due to their small size, 3D textures take a longer time to transmit and render, due to the amount of information they carry. These textures are highly important as they are what the players will see, and so the visual quality of the game depends on these textures.

In this paper, we propose an adaptive texture transmission approach, which consists of server side and client side techniques, to improve the transmission and rendering speed of 3D textures for mobile devices. The trade-off of our approach is a reduction in the visual quality of the textures, which is configurable so that the outcome still produces an acceptable quality. We implement our approach using an M3G-based mobile 3D game and we show that transmission delay with our approach decreases significantly. As we will show in section II, unlike existing approaches which do not consider the mobile environments and the 3D settings at the same time, our approach considers both and leads to a higher efficiency.

The outline of the paper is as follows. In section II, we will explain the related work and their shortcomings, and then in section III, we will propose two techniques, Object Mesh Similarity and Texture Stretching, for faster transmission of 3D objects and their corresponding texture in networked mobile games. In section 4, we present the implementation results, followed by conclusions.

2. RELATED WORK

In this section, we present three categories of state of the art approaches that try to improve the efficiency of transmission and representation of 3D scenes and objects. The categories are progressive meshes, element repository and view dependency.

2.1 Progressive Meshes

In order to reduce the response time, since the complete transmission of the whole objects' meshes could take several seconds, it should be possible to visualize the received data on the client even though the transmission has not finished yet. Progressive Mesh Streaming is a method in which the data is transferred to the clients step by step, by simplification of transmission of large meshes. The approach breaks the information of a scene and the elements inside into several streams, which at first allows a base mesh to be transmitted and rendered on the client-side with low latency [27, 28]. 3D streaming as a general term uses the same core concept [14]. Some work has been done to identify how high the network delay and the data rate can be in a progressive mesh streaming system before it is considered unacceptable by users. However more research is certainly needed to know the bounds on network delay that users can tolerate, although it is still an issue how to partition a progressive mesh into different chunks for transmission [29].

2.2 Element Repository

An object inside a world consists of multiple information such as vertices, colors, or textures. In this approach, the visual appearance of a single object is described by an Element Graph. In order to remove data redundancy for the elements of a 3D scene, the information is stored in several pools which act like repositories and the elements only reference the entries of these pools. There could be several pools like a geometry pool for vertices and texture coordinates, or a topology pool for other topological information.

The systems using this approach usually use an ID for each pool entry which is generated by a checksum algorithm. Each time, the system compares the incoming entry's ID with others and if an Entry's ID is identical to another ID, then the data is supposed to be equal. In in this way, the system checks for data redundancy very fast [27].

2.3 View Dependency

In View-Dependency, the server selects the objects and the elements which are going to be transmitted to the client according to the view-positions and view direction of the clients. In these cases, the client only holds a subset of the world's scene information and this subset changes with the user's view position and visible parts based on area of interest. However, there are some issues like how the visible vertex splits can improve the rendered objects and also deciding which vertex splits to send (and in what order).[26,27,28].

Unlike the above approaches which do not consider the mobile environment and the 3D settings at the same time, our approach considers both, as we shall see in section III. In addition, our proposed methods use texture-based progressive streaming, as opposed to only tree-based or mesh-based progressive streaming used by most other methods. It should be noted however that our method is complementary to all of the above methods, and can be used in conjunction with them for even further efficiency.

3. OUR PROPOSED METHODS

In this section, we propose two methods for reducing the transmission size and thus, reducing response time. Here we are concerned with the objects and their corresponding textures.

In the first method, called Object Mesh Similarity, we construct an alternative object, which is similar to our primary object having minimum difference, but with more simplicity in the whole shape and with reduced complexity of its graph. So in case of detecting a low-speed and congested mobile network, we can initially transfer the alternative object in order to improve the response time and thus the game system's efficiency.

In the second method, we use Texture Stretching as a client-side approach for efficiency improvement in unreliable mobile networks. Using Texture Stretching, we propose a method to improve the response time for dealing with incomplete retrieval of the objects' textures and images which are used in the area of interest.

3.1 Object Mesh Similarity

3D objects are organized by a tree structure and defined by points, polygons and meshes. Transformations such as move, rotate and scale are implemented by a matrix multiplex [2, 3, 12, 13]. In order to speed up objects' transmission in the mobile network, we can consider a method for recognizing objects and find an alternative replacement for each of these objects. In other words, based on the objects' meshes, we try to detect similar objects with fewer complexities and then transmit the alternatively reconstructed objects instead of the original one. In this case, we use the object graph and the corresponding mesh to build a similar object.

In case of having an unreliable and low-speed network for transmission of objects, reducing the size of encoded objects, and consequently the number of transferred UDP packets, is very important. In M3G-based mobile games, vertex and index buffers are combined into Mesh objects; while textures, materials, and other rendering parameters form the appearance of objects. Finally, group nodes hierarchically combine transformations [12]. Considering the graph and the vertices of the object, we can search and construct a similar object with lower numbers of object vectors and thus, lower size of data for transmission. But we should note that the alternative object which is reconstructed using the similarity graph should be in minimum difference with the primary object. So based on the priorities, we first transfer the constructed alternative object. Then we must handle the acting scene, the surrounding objects as well as other necessary parts of the game. As time progresses and we have left behind the congested period, and if no more object transmission is left, we can consider transferring the initial object without omitting surplus details.
The idea of object mesh similarity could be somewhat similar to 3D streaming; it allows viewing 3D meshes with increasing level of details by sending a coarse version of a mesh initially, followed by a sequence of refinements to incrementally improve the quality [14]. But the main difference between object mesh similarity and 3D streaming is that 3D streaming delivers 3D content over networks to let the avatars navigate through the virtual environment without a complete content installation for the playing world, whereas in our approach, we are not concerned about the whole world in which the avatar is acting and playing. Rather, we are concerned about the details of each of the objects involved in the world. The view generated by our approach is more detailed, because our approach works at a lower level. In fact, the two approaches are complementary and object mesh similarity could be applied in 3D streaming for experiencing even better efficiency.
Figure 1 shows a sample, a walking avatar represented in our mobile 3D game demo. Now consider the body of the avatar as an object having eight vertices. Figure 2 marks the corresponding vertices, v1 to v8 for the initial avatar's body graph. Using a polygon recognition algorithm [15], we determine that a quadrilateral could be a good match as a similar object, based on the total shape of the body and the positions of the corresponding graph vertices. By preserving the actual height, the algorithm uses a simple mean for calculating the two sides. Equation (1) simply defines the new X coordinates for the vertices, as follows:

$$X_{v1'} = \frac{X_{v1} + X_{v8}}{2}$$

$$X_{v2'} = \frac{X_{v2} + X_{v3}}{2}$$

$$X_{v3'} = \frac{X_{v4} + X_{v5}}{2} \quad (1)$$

$$X_{v4'} = \frac{X_{v6} + X_{v7}}{2}$$

And thus:

$$W1 = X_{v2'} - X_{v1'} = \frac{X_{v2} + X_{v3} - X_{v1} - X_{v8}}{2}$$

$$W2 = X_{v3'} - X_{v4'} = \frac{X_{v4} + X_{v5} - X_{v6} - X_{v7}}{2} \quad (2)$$

Where W1 and W2 are the widths of the two sides of the reconstructed quadrilateral. Since the actual height, which is defined by h in Figure 2, is preserved, the rough shape of the object remains roughly the same. Thus the reconstructed object resembles the original one and the approach leads to a near similar shape.

We can generalize equations (1) and (2) for any convex n-sided object based on the polygon's set of vertices. Therefore, we can construct an alternate body, having half the numbers of vertices compared with our primary body, also with minimum visual difference. The second object would have lower transmission size up to 50% for the object mesh. But, transferring the texture could still be an issue due to its size.

Figure 2 right shows our simulation results for the constructed alternative shape for the general structure of the body using a Java OpenGL ES based (JSR-239) programming tool. M3G can be efficiently implemented on top of OpenGL ES (although this is not a requirement) [17, 18]. The green points show the graph vertices.

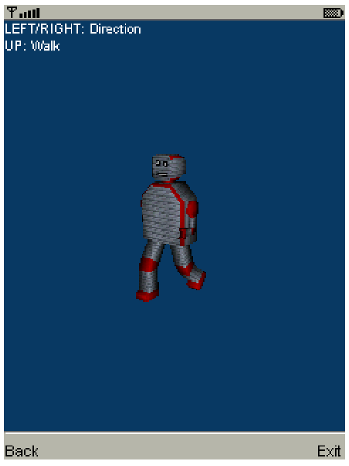


Figure 1. A walking avatar as a sample for Dynamic Meshes

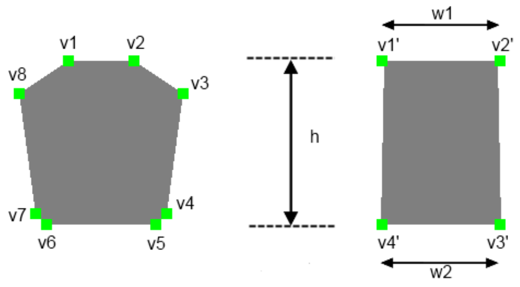


Figure 2. Our initial graph (left) and the reconstructed graph (right) for the avatar's body based on the similarity algorithm.

In this section, we showed one possible method for the implementation of our proposed Object Mesh Similarity technique as a server-side approach. To detect more complex objects, we can use other shape detection methods, some using calculations based on the object's set of vertices and edges, and some others using a database for a best-match, in which the collected data is returned to determine a match with a database of known objects. 3-D Object Recognition using MEGI Model from Range Data [19], using optimal edge-based shape detection algorithms [20], Envelope Detection of Multi-Object Shapes [21] and some state-of-the art methods and means for recognizing two-dimensional shapes, like CSS matching [22] and Shape context [23] could be used in such cases. We must however consider the trade-off between the complexity of the applied algorithm and the response time.

3.2 Texture Stretching

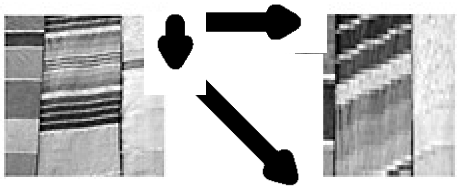
In this section, we propose our second approach for improving the response time in unreliable mobile networks. As was stated in the last section, transferring textures could still be an issue. Due to bandwidth limits and unreliability in mobile networks, it is quite possible that some packets get lost and thus, the state of multiple transmissions would be jeopardized. This action would increase the response time and thus, the total efficiency of the whole system would be decreased [6, 10]. Texture Stretching is our client-side approach for increasing the total responsiveness of the system, i.e. the M3G based mobile game.

In order to speed up the system's texture rendering at the client side, we consider that our mobile network is unreliable. Most texture files have a specific pattern that is repeated throughout the whole picture [24, 25]. We use this repetition characteristic for implementing our proposed method.

In this method, we are concerned with the texture matrix. We set the acquisition rate of the texture to be 20%, i.e. we wait to receive up to 20% of the total texture file before using it for rendering. While transferring the texture from the server to the client and starting to receive the file, it is possible to be faced with packet loss [6, 19]. If we detect packet loss before reaching the 20% limit, we can use a variety of existing retransmission schemes to complete the receipt of 20% of the total texture file. But in case of continuous failures, we can try to reconstruct an alternative texture based on the received parts, by copying the received rows in the texture matrix to replace the next rows of the texture. It should be noted that the 20% limit is chosen as a result of trade-off between the response time and the quality of the received texture; i.e., on one hand we can reduce the response time and consequently save the transmission bandwidth and on the other hand we would gain a better texture quality at the client side based on user acceptability. [29]

Figure 3 shows our approach for the reconstruction of an alternative texture. Figure 3 (I) shows an original n×m sized texture, a unit8 bitmap tile texture as the floor pattern used for our game demo, and the corresponding matrix. Please note that in order to omit the third dimension, we converted the image's color map to a gray scale form. As shown in the figure, considering the dimensions of the texture and the desired part, we would have an (n/5) ×m matrix, since the matrix is responsible for 20% of the total height. (a, …, b), (a', …, b'), etc. define the arrays as the matrix's rows. Figure 3 (II) is the result of running our proposed algorithm to generate an alternative texture, and the reconstructed n×m matrix. So, we see the second texture is in a stretched form compared to the original texture. Figure 4 (I) shows a 3D box in our game which uses the original texture (I) and the reconstructed texture (II). As seen in the results, the reduction of the visual effects is not to a degree that would make the game unusable, and so the tradeoff of visual quality versus response time is justified.

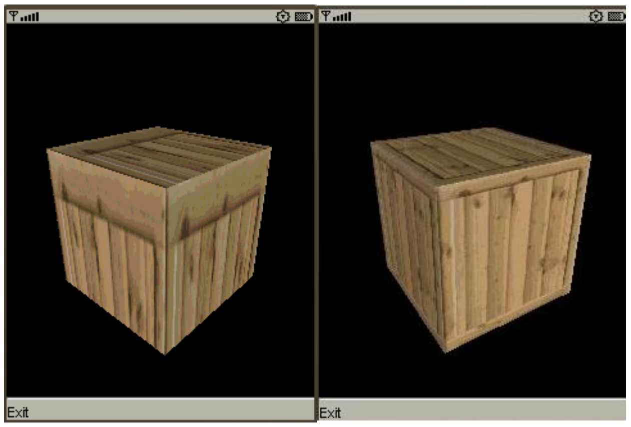
$$\underbrace{\begin{bmatrix} a & \cdots & b \\ a' & \cdots & b' \\ a'' & \cdots & b'' \\ \vdots & \ddots & \vdots \end{bmatrix}}_{n/5 \times m} \longrightarrow \underbrace{\begin{bmatrix} a & \cdots & b \\ \vdots & \ddots & \vdots \\ a & \cdots & b \\ a' & \cdots & b' \\ \vdots & \ddots & \vdots \\ a' & \cdots & b' \\ a'' & \cdots & b'' \\ \vdots & \ddots & \vdots \end{bmatrix}}_{n \times m}$$



(I) (II)

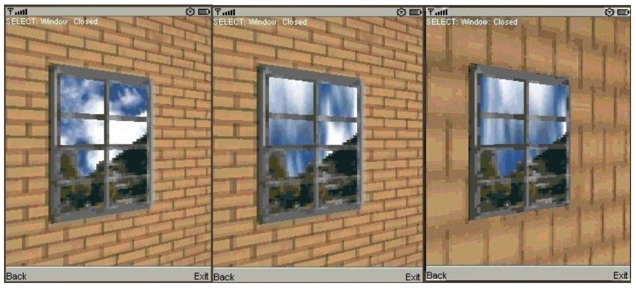
Figure 3. (I) The original tile texture. (II) The reconstructed texture using Texture Stretching algorithm for 20% of the original height.

We have also chosen some other common objects and simulated their visual quality. We have chosen different balls for different uses (tennis, basketball, and soccer), a flower field, and a house and simulated the results of applying texture stretching on the corresponding textures. Figures 5–7 show the simulation results. As can be seen from the results, the visual quality of the objects is such that the player can still easily see what those objects are and the game can be played. The question that arises is how much gain to we obtain in the response rate of the system as a result of this quality reduction. This question is answered in section IV, where we show a delay performance improvement of 35% to 70%.



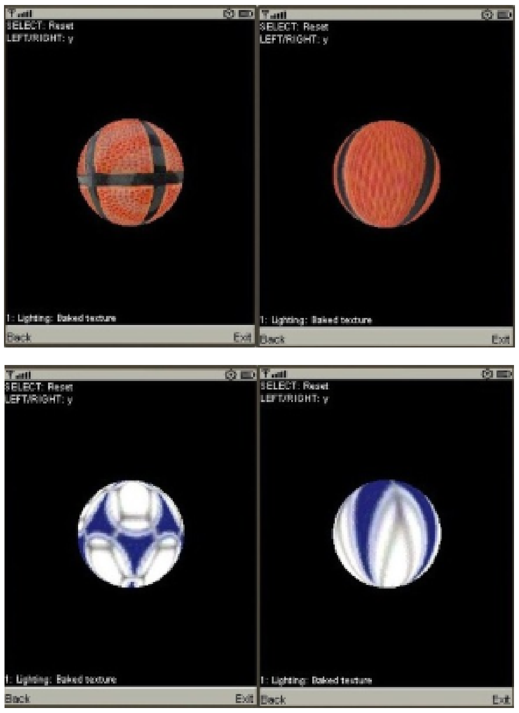
(I) (II)

Figure 4. (I) The original 3D box used in the game demo. (II) The reconstructed 3D box using Texture Stretching.



(I) (II) (III)

Figure 5. Applying a two-level texture stretching for a wall and the outer scenery. (I) The original view. (II) First level texture stretching for the outer animated clouds. (III) Applying the second level texture stretching for the wall texture.



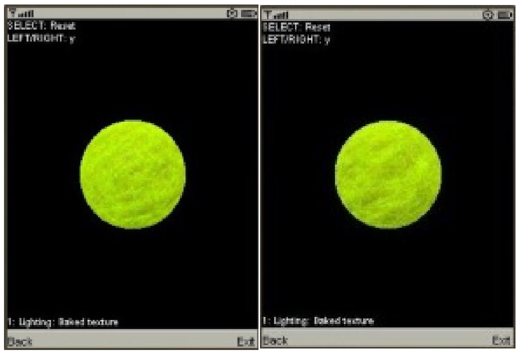


Figure 6. Applying texture stretching on different types of balls. Left side simulations are the original views, and the right side simulations are the results.

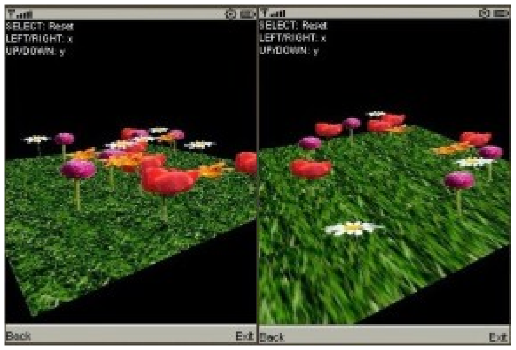


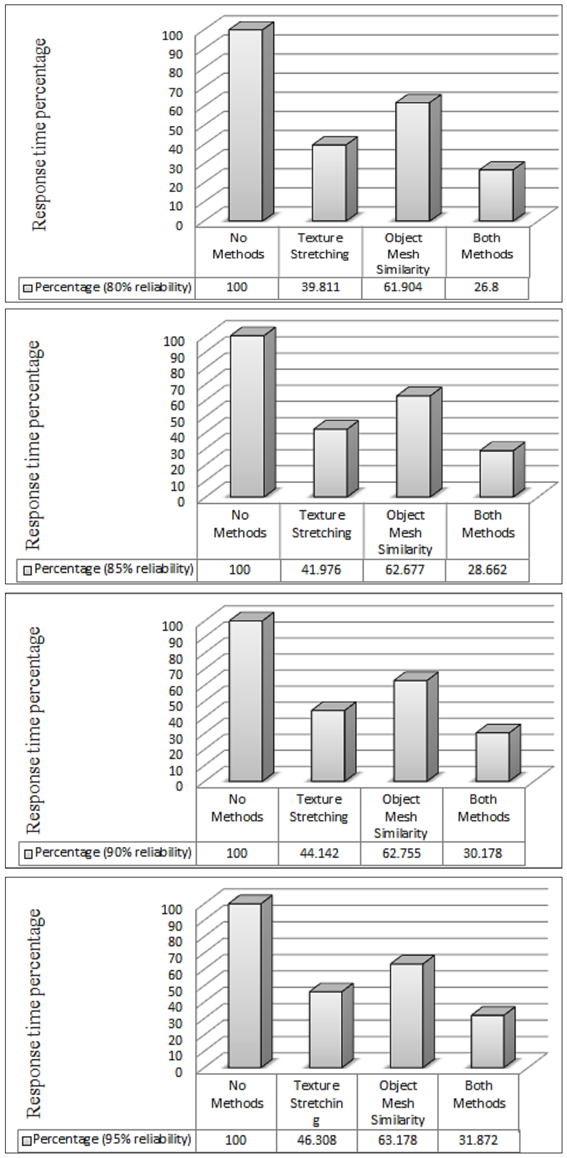
Figure 7. The effect of using texture stretching for the flower garden. As the simulation indicates, the visual effect of the stretched grass texture used in the reconstructed field (right image) is rather similar with the original grass field (left image).

Together with our proposed client-side approach, we can also use some existing and common methods like compression, scaling or media interpolation that can reduce the visual effects and quality. Compression further reduces the size of transmitting textures, while scaling reduces the image resolution by a specific ratio, and then an enlargement would be applied at the client-side to reconstruct the image to its original size. In order to improve the visual effect for achieving a better appearance during game rendering, we can also apply super sampling as an anti-aliasing technique, which is the process of eliminating jagged and pixelated edges, and can be utilized for the smoothing of images rendered in games. However, the computing complexities and response time of these algorithms must be considered in the overall system to ensure they will not introduce unacceptable delays in the game.

Another complementary method is media interpolation, which could be very effective especially for animation transmission during gameplay in congested and unreliable networks. Animations sometimes are based on changing the covering image, in which the total animation sequences are combined to form a single image. Sequences are any group of burst mode images combined into a single animated image file, like GIF files. Using media interpolation, instead of transferring the whole image, we try to transfer the resequenced image, with fewer numbers of sequences. This approach can be used as either a server-side approach or client-side approach.

4. PERFORMANCE EVALUATION

We have run a game implementation of our system over a simulated network to evaluated its performance. The game consisted of several parts, including objects and their corresponding PNG formatted textures, of an old school robot, flower garden, some balls and a house wall with windows which were shown in figures 1 and 4 to 7. We used the latest version of the Sun JAVA Wireless Toolkit for CLDC, which defines a Java runtime environment for low end devices with constrained hardware resources such as mobile phones, and we used Windows XP platform for running the simulations. In the simulations, the network had different levels of reliability at 80%, 85%, 90% and 95% packet transmission success rate.



	No Methods	Texture Stretching	Object Mesh Similarity	Both Methods
□ Percentage (80% reliability)	100	39.811	61.904	26.8

	No Methods	Texture Stretching	Object Mesh Similarity	Both Methods
□ Percentage (85% reliability)	100	41.976	62.677	28.662

	No Methods	Texture Stretching	Object Mesh Similarity	Both Methods
□ Percentage (90% reliability)	100	44.142	62.755	30.178

	No Methods	Texture Stretchin g	Object Mesh Similarity	Both Methods
□ Percentage (95% reliability)	100	46.308	63.178	31.872

Figure 8. The response time percentage for different network reliabilities, i.e. 80%, 85%, 90% and 95%.

In all the simulations, utilization of our proposed methods showed considerable improvements in the response time. Figure 8 shows our simulation results, where each graph is for a different reliability rate. The simulations were done for four different scenarios: without applying any of our proposed approaches, using each of our two approaches separately, and finally, using both approaches at the same time. As the results show, more than 35% to 70% improvement in the response time could be achieved compared with the scenario in which no approaches were applied. For instance considering the mobile network with 80% reliability, using Texture Stretching would result in almost 60.19% less response time, and applying the Object Mesh Similarity method would also lessen the responsiveness up to 38.1%, and in the long run by applying both methods we could improve the response time almost 73.2%. As we can see from the results, combining both methods together always yields the best performance gain. Furthermore, as the reliability of the network becomes less, our methods have a higher impact on performance gain.

5. CONCLUSION

In this paper, a server-side and a client-side approach were introduced to improve the responsiveness of a 3D mobile game. Object Mesh Similarity as a server-side approach and texture stretching as a client-side approach. Performance results indicate that our methods improve the response time in unreliable and congested mobile networks during gameplay. As mentioned in the paper, it is possible to use our method in conjunction with existing complementary methods. one example is 3D object conversion using object mesh splitting. In such cases, if we have some 3D objects that cover an area that is currently going to be transmitted as single mesh objects, the performance would be better if we only transfer the sub-meshes which are currently in the player's view. So only the sub-mesh in the view port needs to be rendered and thus, we could have less resource consumption and faster response time.

6. REFERENCES

[1] Soh, J.O.B. and Tan, B.C.Y. "Mobile Gaming," Communications of the ACM, March 2008, Volume 51, Number 3, pp. 35-39.

[2] Mobile 3D Graphics API for J2METM, http://jcp.org/en/jsr/detail?id=184

[3] Mobile 3D Graphics API 2.0, http://jcp.org/en/jsr/detail?id=297

[4] A Java binding API for OpenGL ES, http://jcp.org/en/jsr/detail?id=239

[5] Kurose, J.F. and Ross, K.W. 2003, Computer Networking: "A Top-Down Approach Featuring the Internet", 2nd Ed., Pearson Education.

[6] P. Lonapalawong, A. Davison, "Improving Response Time in Client/Server 3D Mobile Game", CyberGames, UK, 2007.

[7] Singhal, S.K. 1996, "Effective Remote Modeling in Large Scale Distributed Simulation and Visualization Environments", PhD thesis, Dept. of Computer Science, Stanford University.

[8] L. Pantel and L. C.Wolf, "On the suitability of dead reckoning schemes for games", NETGAMES, pages 79–84. ACM, 2002.

[9] S. Legtchenko, S.Monnet , G. Thomas, "Blue Banana: resilience to avatar mobility in distributed Massively Multiplayer Online Games", Dec. 2009.

[10] J. Pang, F. Uyeda, and J. R. Lorch, "Scaling peer-to-peer games in low-bandwidth environments". In IPTPS '07: Proc. of the 6th International Workshop on Peer-to-Peer Systems, Feb. 2007.

[11] A. R. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: "enabling large-scale, high-speed, peer-to-peer games". In V. Bahl, D. Wetherall, S. Savage, and I. Stoica, editors, SIGCOMM, pages 389–400. ACM, 2008.

[12] K.Pulli, T.Aarnio, V.Miettinen, K.Roimela, and J.Vaarala, Mobile 3D Graphics with OpenGL ES and M3G, Morgan Kaufmann, 2007.

[13] "3D graphics for Java mobile devices", IBM White paper, 2005.

[14] S. Hu, J. Jiang, B. Chen, "Peer-to-Peer 3D Streaming", IEEE internet computing, 2010.

[15] A. Ferreira, M. Fonseca, and J. Jorge. "Polygon Detection from a Set of Lines". In EncontroPortugues de ComputacaoGrafica, 2003.

[16] A. Bharambe, J. Pang, S. Seshan. Colyseus: "a distributed architecture for online multiplayer games". Proceedings of the 3rd conference on Networked Systems Design & Implementation, Berkeley, USA, 2006.

[17] Munshi, A., D.Ginsburg, D.Shreiner. 2008. OpenGL ES 2.0 Programming Guide, Addison-Wesley.

[18] C. Hofele, "Mobile 3D Graphics: Learning 3D Graphics with the Java Micro Edition", Course Technology Press, Boston, USA, 2007

[19] H. Matsuo, A. Iwata, "3-D Object Recognition using MEGI Model from Range Data", IEEE 12th IAPR International Conference, 1994.

[20] Moon, H., Chellappa, R., Rosenfeld, A.: "Optimal edge-based shape detection."IEEE Transactions on Image Processing, 2002.

[21] N. Alajlan, 0. El Badawy, M.S. Kamel, G, Freeman: "Envelope Detection of Multi-object Shapes". ICIAR 2005

[22] S. Abbasi, F. Mokhtarian, and J. Kittler. "Curvature scale space image in shape similarity retrieval". Multimedia Syst., 7(6):467– 476, 1999.

[23] S. Belongie, J. Malik, and J. Puzicha. "Shape matching and object recognitionusing shape contexts", 2001.

[24] Sony Ericsson Developer World white paper, "Mobile 3D Graphics and Java Applications Development for Sony Ericsson Phones", Nov. 2004.

[25] T. Aarnio, "M3G –Java Mobile 3D", Nokia Research Center, 2009.

[26] J. Kim, S. Lee, and L. Kobbelt. "View-dependent mesh streaming with minimal latency", International Journal of Shape Modeling, 11(1):63–90, June 2005.

[27] J. Sahm, I. Soetebier, H. Birthelmer, "efficient representation and streaming of 3D scenes", International Journal of Computers & Graphics, 2004.

[28] W. Cheng, "streaming of high-resolution progressive meshes over the internet", PhD thesis, Department of Computer Science, National University of Singapore

[29] R. N. De Silva, W. Cheng, W.T. Ooi, S. Zhao, "Towards Understanding User Tolerance to Network Latency and Data Rate in Remote Viewing of Progressive Meshes", Proc. International workshop on Network and Operating Systems Support For Digital Audio And Video (NOSSDAV), Amsterdam, The Netherlands, June 2 – 4 2010.