A NETWORKED RENDERING PARADIGM FOR REMOTE RENDERING

Nguyen Thanh Dong, Le Dieu Linh

Received: 15 March 2017 / Accepted: 7 June 2017 / Published: July 2017

Abstract: *Advances in 3D graphics make it possible to generate highly realistic 3D models which usually contain a huge number of polygons. This large number of polygons gives rise to many challenges with respect to real-time rendering performance, storage requirements, and the transmission of graphics dataset over the network. In this paper, a networked rendering paradigm based on our pipeline-splitting method is introduced to facilitate the remote rendering system. Experimental results show that our method can reduce memory cost and computational workload for the client compared to that of client-side method.*

Keywords: *Remote rendering, cloud computing, networked graphics.*

1. Overview

As 3D models are becoming more realistic, it is challenging to render such models in real time due to the limited resources on mobile devices. As a result, there is a need to make use of client/server architecture in order to offload graphics rendering workload to the remote server leaving only minor tasks on the client-side. In general, methods related to the rendering of 3D graphics in the network environment can be roughly divided into three major categories: *client-side method, server-side method, and hybrid method*.

Client-side method:

In this method, the client is fully responsible for rendering the entire 3D models and the server simply sends graphics data to the client. A conventional way of client-side rendering is to transmit graphics commands to the client to be processed at the client [1, 2]. This method can reduce workload at the server, but it increases the processing demand on the client. This is suited for small applications but is insufficient for applications that require high rendering power. Moreover, graphics data to be transmitted to the client may be large leading to a long downloading time. To make it possible for the transfer of large models, the server performs the simplification and conversion to calculate a progressive representation composed of a

Nguyen Thanh Dong
Faculty of Information and Communication Technologies, Hong Duc University
Email: Nguyenthanhdong@hdu.edu.vn (✉)

Le Dieu Linh
Department of Information System, Hong Duc University
Email: Ledieulinh1504@gmail.com (✉)

simplified model and a series of refinements that the client will progressively download and display [3-5].

Server-side method:

In contrast to the client-side method, this method involves the server as completely responsible for graphics processing. The server renders the 3D scenes and transmits the rendered images to the client to be displayed [6-8]. This is highly beneficial to thin clients which often lack specialized hardware and are memory-limited [9-11]. However, the limitation of this method is that the server may become congested when serving a large number of clients and an appropriate network connection needs to exist. This maybe fine for fixed type networks but not appropriate for wireless networks. In addition, the latency due to the constant transmission of rendered images from the server to client may cause a reduction of interactivity. This is also costly in terms of network bandwidth. Image based rendering (IBR) techniques can be implemented in the client to improve frame rates and to deal with the transmission delay [12, 13]. However, there are some tradeoffs between the image quality and transmission latency [8].

Hybrid method:

In this method, both the client and server get involved in the rendering process. Rendering tasks are partially accomplished at the server and the remainder is performed at the client. The rendering workload can be shared between the server and client [14, 15]. However, deciding which parts to be performed at the client and which parts to be performed at the server is not an easy task. Noguera, et al., [14] proposed a technique to split the rendering workload between the server and the client based on the view volume. The client is responsible for rendering the terrain which is close to the viewer and the server renders the terrain far away from the viewer. Diepstraten, et al., [16], in a different manner, split up the image generation in order to balance workload between the client and the server. The server partially renders the 3D scene and sends 2D primitives to be processed on the client. However, this may lead to the downgrading of image quality since the client has to rely on feature lines abstracted from 3D models to draw the image.

In this paper, we introduce a new networked paradigm for remote rendering. A novel method to split the rendering pipeline is proposed aiming to break the rendering workload from the point that geometry processing is performed at the server, leaving the remaining parts to be done at the client. Different from conventional pipeline-splitting approaches, our approach relies on transform feedback mode[1] to obtain data from the buffer object in the graphics card, hence achieve hardware acceleration for geometry processing. Various 3D models have been experimented with our framework, the experimental results shown that our method can minimize memory cost and computational workload at the client and the processing time at the server.

[1] *http://www.opengl.org/registry/specs/NV/transform_feedback.txt*

2. Rendering pipeline analysis

In general, a rendering pipeline typically consists of a number of stages including vertex processing, geometry processing, rasterization, and fragment processing. For the sake of simplicity, we consider the pipeline with only two separated stages. The first stage named geometry processing is responsible for vertex transformations, lighting calculations, and triangle assembly. The second stage named rasterization is a combination of clipping/culling, rasterization, and fragment processing.

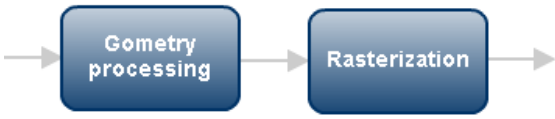
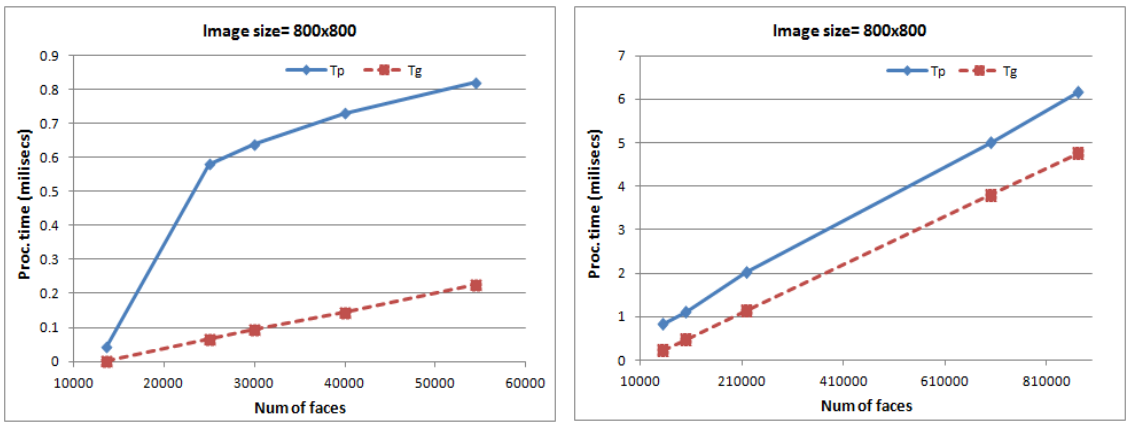


Figure 1. *The analytical rendering pipeline*

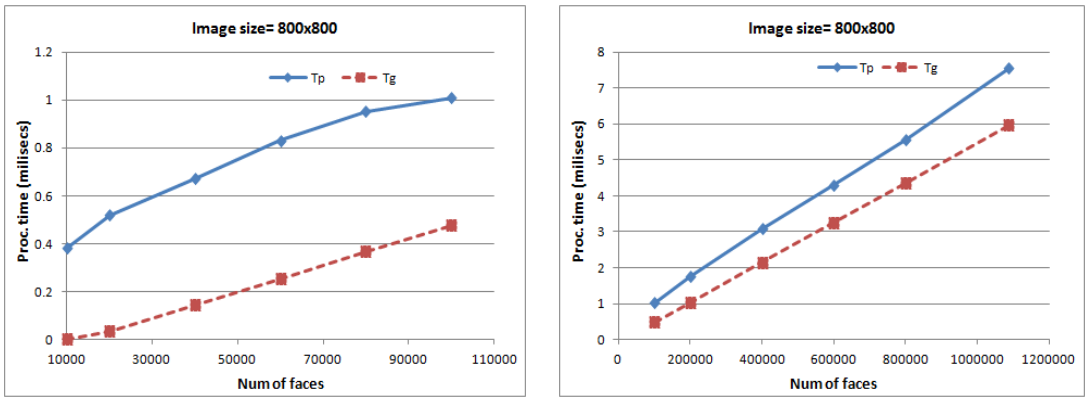
From this perspective, we present an analysis of the rendering pipeline in terms of processing time. It is worth noting that the determination of the most time-consuming stage in the graphics rendering pipeline is challenging as each stage depends on various factors. For example, the processing time at the geometry processing stage depends on the number of primitives while the processing time at rasterization stage depends on the number of input primitives, the viewing angle, and the image resolution.

Let T_p be the processing time of the entire pipeline, and T_g be the processing time of geometry processing stage. The total execution time T_p is equal to the sum of the execution times for the two stages: geometry processing and rasterization. T_g can be roughly estimated by disabling rasterization stage to prevent primitives from being rasterized. Note that we do not take into account the time taken to clear and swap the buffer during the rendering for the sake of simplicity.



(a) (b)

Figure 2. *Processing time at geometry processing stage compared to the rendering time in case of dragon model- graphics card: NVIDIA GeForce 9500GT (a) the number of faces is less than 100k (b) the number of faces ranging from 100k to 1M*



(a) (b)

Figure 3. *Processing time at geometry processing stage compared to the rendering time in case of happy model- graphics card: NVIDIA GeForce 9500GT (a) the number of faces is less than 100k (b) the number of faces ranging from 100k to 1M*

(a) (b) (c) (d)

Figure 4. *3D models are used in the test*

We consider the impact of the image resolution and the number of primitives to the processing time at geometry processing stage and the rendering time of the entire pipeline. Figure 3, 4, 5 demonstrate some experimental results obtained from the test. This shows that for complex 3D models and small image size, tremendous amount of time is spent at geometry processing stage. Therefore, it is desirable to offload geometry processing stage to a dedicated server, and the rasterization is handled at the client. This can balance the rendering workload between the client and the server to some extent.

3. Networked rendering framework

In this section, we describe a scheme for remote rendering based on our pipeline-splitting method. At first, we present a paradigm for a networked rendering pipeline that extends the traditional rendering pipeline to include network transmission of geometry data. The rendering pipeline is divided in a way that some stages of it are offloaded to the remote server and the remainders remain at the client.

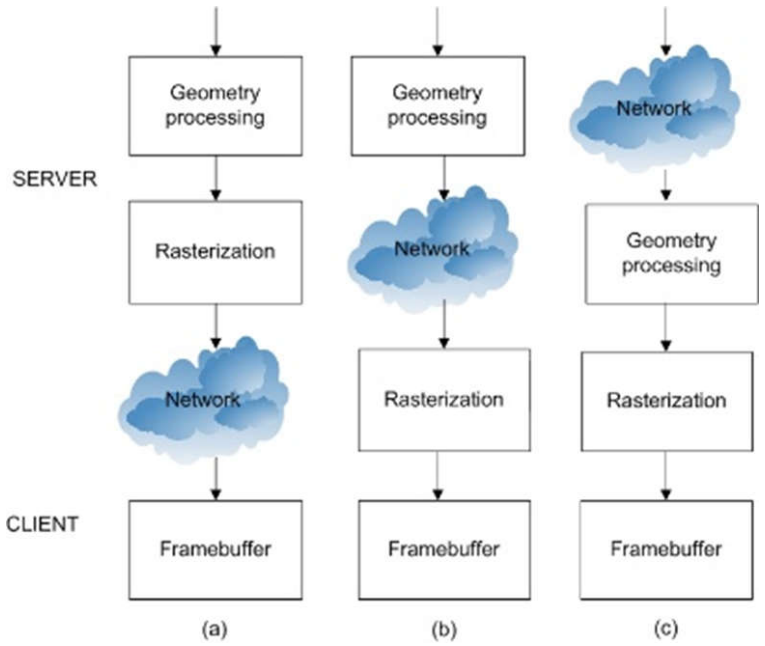


Figure 5. *Different architectures of networked rendering pipeline, (a) the entire pipeline is placed on server, (b) geometry is placed on server, rasterization is on client, (c) the entire pipeline is placed on client*

Pipeline splitting

Typically, the rendering pipeline resides on a single machine. It is difficult to achieve pipeline splitting due to the tight coupling of geometry and rasterization stage. Williams et al. [17, 18] proposed a method to separate the geometry stage and rasterization stage by adding two extensions to OpenGL library: triangle-feedback and triangle-rasterize. The triangle-feedback function passes all primitives through the geometric portion without rasterizing them and the triangle-rasterize function takes the data from geometric portion then put it into rasterization stage. To achieve hardware acceleration for rasterization, a vertex program is implemented to pass primitives into the hardware rasterizer on the graphics card. Graphics hardware acceleration, however, remains uncompleted for geometry processing. Banerjee, et al., [19, 20] combined Mesa3D[2] and socket networking code together to build RMesa which can split up the rendering pipeline into sub stages. The client can offload some stages in the pipeline to the remote server to be processed and then get the result back. Unfortunately, the approach offers no graphics hardware-acceleration for both geometry processing and rasterization. In our research, we split the rendering pipeline based on transform feedback mode. The use of transform feedback makes it possible to capture vertex attributes of the primitives processed by geometry processing stage. Vertex attributes are selected to store in a buffer, or several buffers separately which can be retrieved sometime later. The rest of pipeline can be discarded by disabling rasterization stage to prevent primitives from being rasterized. This way uncouples geometry processing stage from rasterization stage. The

[2] *http://www.mesa3d.org/*

transformed primitives copied from transform feedback buffer then can be rasterized in a different machine by simply put it back to the buffer without passing any transformation parameters. It is worth noting that the entire process happens inside the pipeline, therefore an advantage of our method is that it supports hardware-acceleration to both geometry processing and rasterization stage.

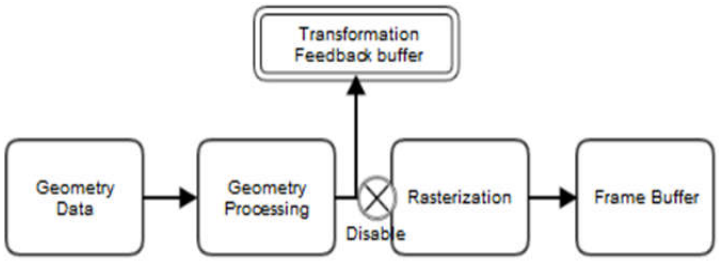


Figure 6. *Transform feedback operation-vertices are transformed and stored in the transform feedback buffer object which can be obtained in the middle*

Remote rendering based on pipeline-splitting method

In this section, we introduce a remote rendering framework making use of pipeline-splitting method that we have presented earlier. The basic concept is similar to image-based rendering, the major difference is that the sever sends back transformed primitives instead of rendered images to the client.

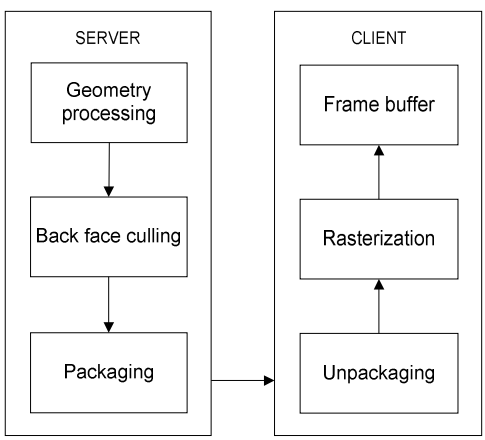


Figure 7. *Client-server architecture for the proposed framework*

Table 1. *Notation 1*

Symbols	Quantity
F	List of faces constructed the mesh
F_c	The remaining faces after culling
M, N	The number of faces stored in F and F_c respectively
CHUNK	Number of faces stored in a packet
p	Number of packets to be sent to the client

In our proposed framework, the server performs geometry processing on demand according to the viewing parameters received from the client. The back-face culling method then is employed to cull invisible primitives from transformed ones. The remaining primitives then are packaged to be sent to the client for rasterization.

To deal with restrictions in network performance and bandwidth, we take into account the network protocol for the data transmission. For the sake of transmission efficiency, it is important that UDP is employed for data transmission and TCP is used for exchanging messages and commands. To further reduce the latency, graphics content is packetized or can be compressed prior to the transmission. A chunk of primitives is grouped in a packet to be sent to the client for further processing. The number of packets to be sent for the rendering of a frame can be calculated as follows:

$p = \lceil M/CHUNK \rceil = \lceil \alpha N/CHUNK \rceil$ where $\alpha = M/N$ is culling ratio $(0 < \alpha \leq 1)$.

Transmission latency

Supposed that the time taken to transmit a packet to the client is t_p. t_p depends on network capacity (bw) and the size of packet (s_p): $t_p = s_p/bw$.

Let T be the transmission time of all primitives after performing back-face culling. This is equivalent to the transmission of p packets:

$$T = p \times t_p = \lceil \alpha N/CHUNK \rceil \times (s_p/bw)$$

It can be seen that the transmission latency is linearly proportional to the number of faces (N).

Table 2. *Time to transmit a packet*

CHUNK	**t_p (secs)**	
	10 Mbps	100 Mbps
600	0.03456	0.003456
300	0.01728	0.0017728
200	0.01152	0.001152
100	0.00576	0.000576

Table 3. *A theoretical estimation of the time it takes to transmit 3D models with different level of details (CHUNK=600)*

N	**P**	**T (secs)**	
		10 Mbps	100 Mbps
10000	17	0.58752	0.058752
20000	34	1.17504	0.117504
40000	67	2.31552	0.231552

60000	100	3.456	0.3456
80000	134	4.63104	0.463104
100000	167	5.77152	0.577152

4. Experimentation

We have implemented a remote rendering system on Windows in C++ using OpenGL making use of the proposed pipeline-splitting method to split the rendering workload between the server and client. The server we used in the test is Intel ® Core ™ i7 CPU, 3.24 GB of RAM, with NVIDIA GeForce 9500. A DELL T6600, Intel® Core™ 2 Duo CPU 2.2 GHz, 2G RAM is used as a client.

Processing time in the pipeline

We make a comparison between local rendering and our method in terms of processing time in the rendering pipeline at the client. As the number of faces being processed at the client has been reduced and geometry processing has been carried out at the remote server, our method can reduce the processing time at the client.

Table 4. *A comparison between our proposed method and local rendering in terms of processing time*

Model	Num of verts	Num of faces	Local rendering (secs)	Our method (secs)
Beethoven	2521	5030	0.0042	0.0027
Car	5247	10474	0.0072	0.0048
Ateneam	7546	15014	0.0100	0.0060
Dragon	10006	20000	0.0170	0.0080
Venus	19847	43357	0.0320	0.0180
Bunny	34834	69451	0.0486	0.0276

We compare our method with server-side rendering in terms of processing time at the server. In case of server-side rendering, we measure the processing time of the entire pipeline plus the time taken to copy data from the frame buffer to CPU. For our method, we measure the processing time at geometry processing stage and the time to copy data from the transform feedback buffer. When the number of primitives to be processed is small and the image size is large, the processing time at the server is significantly reduced in our method compared to that of server-side rendering. Note that when the fragment processing is relatively cheap, the transform feedback could end up being a major bottleneck leading to more processing time at the server in our method compared to that of server-side rendering.

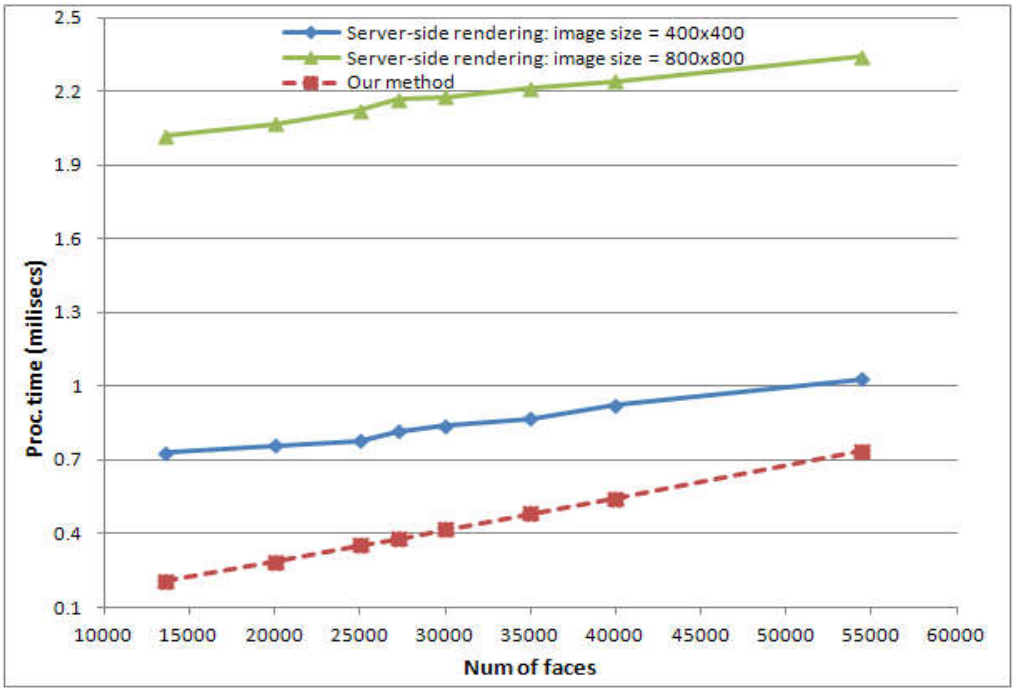


Figure 8. *A comparison between server-side rendering and our method in terms of processing time tested with dragon model*

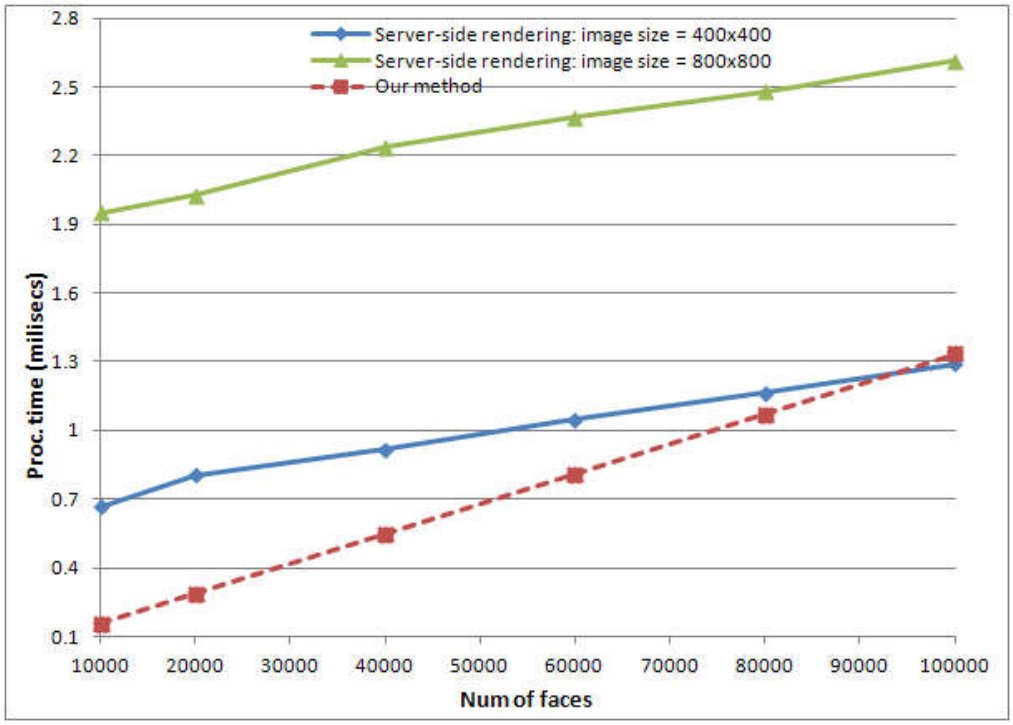


Figure 9. *A comparison between server-side rendering and our method in terms of processing time tested with happy model*

Storage requirements

After back-face culling[3] is performed at the server, only visible faces are sent to clients for further processing. Therefore, the amount of faces to be handled at the client is significantly reduced. As can be seen in the Figure below, about 40-50% of the faces are actually processed at the client. As such, our method would be of great benefits to thin clients since they are limited in their storage capacity.

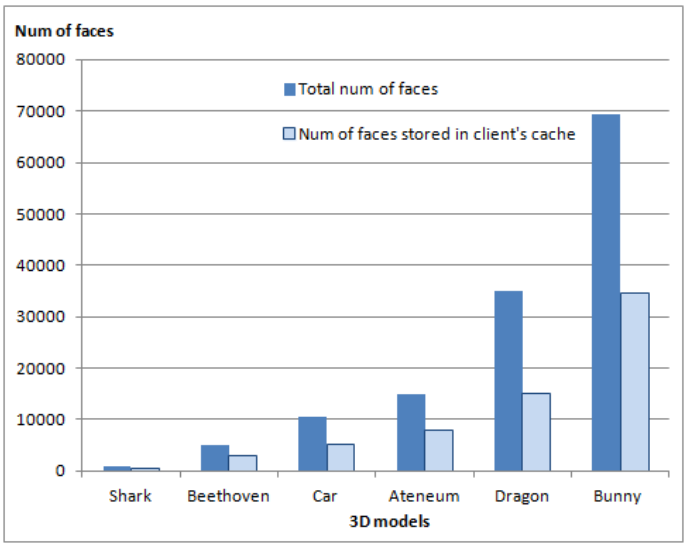


Figure 10. *Average number of faces processed at the client*

Network communication

The data transfer capability is considered a major bottleneck in the remote rendering. Network communication for the proposed framework is built on TCP/IP sockets. We employ UDP for the transmission of graphics datasets and TCP for sending commands from client to server and vice versa. We have previously presented a theoretical analysis of transmission latency in previous section. Therefore, this experiment is also able to verify the theoretical analysis of our proposed framework. Our test is conducted in both a 10Mbps and 100Mbps Ethernet connections. To further reduce the transmission latency, we can make use of a compression/decompression technique. However, it is worth noting that the process of compression/decompression may introduce some delays to the system.

Table 5. *Transmission latency measured in different network connections*

Model	**Num of faces**	**Latency (seconds)**	
		10 Mbps	100 Mbps
Shark	734	0.0380	0.0043
Apple	1704	0.0750	0.0084

[3] *https://en.wikipedia.org/wiki/Back-face_culling*

Ant	912	0.0380	0.0044
Beethoven	5030	0.1778	0.0199
Car	10474	0.3432	0.0337
Ateneam	15014	0.3840	0.0469
Big dodge	16646	0.5261	0.0543
Dragon 1	20000	0.6247	0.0641
Dragon 2	35000	1.0741	0.1117
Venus	43357	1.2881	0.1359
Bunny	69451	2.1737	0.2124

5. Conclusion

In this paper, we have investigated the graphics rendering pipeline in terms of processing time. We have proposed a networked rendering paradigm based on our pipeline-splitting method to facilitate remote rendering. It is shown that our method can reduce memory cost and computational workload at the client compared to that of client-side rendering and processing time at the server compared to that of server-side rendering. The work also can be applied to distributed-rendering as we can distribute geometry processing and rasterization to be handled on different machines in the cloud. It is worth noting that our framework can work with pretty large 3D models, however, there must be a limit since the residual list is linearly proportional to the number of faces of the 3D model.

References

[1] G. Jung and S. Jung (2006), *A Streaming Engine for PC-Based 3D Network Games onto Heterogeneous Mobile Platforms*, in Technologies for E-Learning and Digital Entertainment. vol. 3942, Z. Pan, R. Aylett, H. Diener, X. Jin, S. Göbel, and L. Li, Eds., ed: Springer Berlin / Heidelberg, pp. 797-800.

[2] A. Mohr and M. Gleicher (2002), *HijackGL: reconstructing from streams for stylized rendering*, presented at the Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering, Annecy, France, pp.13-ff.

[3] G. Hesina and D. Schmalstieg (1998), *A Network Architecture for Remote Rendering*, presented at the Proceedings of the Second International Workshop on Distributed Interactive Simulation and Real-Time Applications, pp.88.

[4] M. Isenburg and P. Lindstrom (2005), *Streaming meshes*, in Visualization, 2005. VIS 05. IEEE, 2005, pp. 231-238.

[5] H. T. Vo, S. P. Callahan, P. Lindstrom, V. Pascucci, and C. T. Silva (2007), *Streaming Simplification of Tetrahedral Meshes*, Visualization and Computer Graphics, IEEE Transactions on, vol. 13, pp. 145-155.

[6] X. Liu, H. Sun, and E. Wu (2000), *A hybrid method of image synthesis in IBR for novel viewpoints*, presented at the Proceedings of the ACM symposium on Virtual reality software and technology, Seoul, Korea.

[7] Z. J. Y. Lei, D. Chen, and H. Bao (2004), *Image-Based Walkthrough over Internet on Mobile Devices*, in Proc. GCC Workshops, pp. 728-735.

[8] Y. a. C.-O. Mann, D. (1997), *Selective Pixel Transmission for Navigating in Remote Virtual Environments,* Eurographics '97, Volume 16, Number 3.

[9] A. Boukerche, T. Huang, and R. W. N. Pazzi (2005), *A real-time transport protocol for image-based rendering over heterogeneous wireless networks*, presented at the Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems, Montral, Quebec, Canada.

[10] A. Boukerche, F. Jing, and R. B. de Araujo (2006), *A 3D image-based rendering technique for mobile handheld devices*, in World of Wireless, Mobile and Multimedia Networks, International Symposium, pp. 7 pp.-331.

[11] F. Lamberti and A. Sanna (2007), *A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices*, Visualization and Computer Graphics, IEEE Transactions on, vol. 13, pp. 247-260.

[12] P. Bao and D. Gourlay (2003), *Low bandwidth remote rendering using 3D image warping*, in Visual Information Engineering, 2003. VIE 2003. International Conference on, 2003, pp. 61-64.

[13] P. Bao and D. Gourlay (2006), *A framework for remote rendering of 3-D scenes on limited mobile devices*, Multimedia, IEEE Transactions on, vol. 8, pp. 382-389.

[14] J. M. Noguera, R. J. Segura, C. J. Ogáyar, and R. Joan-Arinyo (2011), *Navigating large terrains using commodity mobile devices*, Computers & Geosciences, vol. 37, pp. 1218-1233, 2011.

[15] M. Levoy (1995), *Polygon-assisted JPEG and MPEG compression of synthetic images*, presented at the Proceedings of the 22nd annual conference on Computer graphics and interactive techniques.

[16] J. Diepstraten, M. Gorke, and T. Ertl (2004), *Remote Line Rendering for Mobile Devices*, presented at the Proceedings of the Computer Graphics International.

[17] J. L. Williams and R. E. Hiromoto (2005), *Sort-middle multi-projector immediate-mode rendering in Chromium*, in Visualization. VIS 05. IEEE, 2005, pp. 103-110.

[18] J. L. Williams and R. E. Hiromoto (2003), *A proposal for a sort-middle cluster rendering system*, in Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2003. Proceedings of the Second IEEE International Workshop on, 2003, pp. 36-38.

[19] K. S. Banerjee and E. Agu (2005), *Remote execution for 3D graphics on mobile devices*, in Wireless Networks, Communications and Mobile Computing, International Conference on, 2005, pp. 1154-1159 vol.2.

[20] E. Agu, B. Kutty, N. Shirish, O. Rekutin, and D. Kramer (2005), *A middleware architecture for mobile 3D graphics*, in Distributed Computing Systems Workshops. 25th IEEE International Conference on, 2005, pp. 617-623.