Progressive Compression for Lossless Transmission of Triangle Meshes in Network Applications

Timotej Globačnik*
Institute of Computer Graphics Laboratory for Geometric Modelling and Multimedia Algorithms
Faculty of Electrical Engineering and Computer Science, University of Maribor
Maribor / Slovenia

Abstract

This paper presents a usage of algorithms for progressive compression of triangle meshes in a network applications with a single server and is composed from two major parts. First one describes algorithm proposed by Alliez and Desbrun, which is used for progressive compression and reconstruction of triangle meshes. The second part considers our experience with distribution the data between clients using a single server. Different distribution rules were implemented and tested. Because server can send data only to one client at the time, this approach is suitable only for small server applications.
Keywords: progressive compression, triangle meshes, network applications, sockets

1 Introduction

In recent years, the performance of compute hardware and broad-band internet connection improved dramatically. It is possible to realistic render the complex 3D graphics scene even on low-cost computers practically in real time. The boundary of complex free-form geometric objects is usually represented by a huge number of triangles – by triangle meshes. More realistic is the model greater number of triangles is required for its representation. However, due the large number of triangles the transfer of such geometric models through the internet is still limited. As with other multimedia types (digital video, digital music), the compression of triangular meshes is needed. Geometric data such triangular meshes differ importantly from other type of data. Two types of algorithms for compression triangular meshes can be distinguished [6].

- The geometric model is compressed in one step. This approach is known as a *single-rate coding*. The biggest drawback of this approach is inability of showing the rough picture of the model before it is fully reconstructed. This makes this approach inappropriate for internet applications.
- The model is compressed in several steps, where each step removes a set of vertices that satisfy special criteria. We are talking about *progressive coding*. In this case, geometric object is compressed in different levels of details – LOD [8, 9]. To keep the model recognizable also at the lowest level of detail, the number of steps is limited. The mesh created after last step is considered as a *base mesh*. The base model is usually further compressed with the single-rate coder. The reconstruction algorithm works in the opposite direction. First, the base mesh is reconstructed. After that, each pass adds certain amount of details to the mesh reconstructed in the previous pass, until it is fully reconstructed and identical to the original model.

The progressive triangular meshes compression algorithms can be divided into three main groups [6, 7]:

- **Polyhedral simplification algorithms.** The main idea of these algorithms is to remove the vertices from the mesh by altering its topology and, if necessary, also the position of the remaining vertices what reduces errors produced by the simplification process. These algorithms can be used in applications using lossy compression techniques but are inappropriate for applications, where topology must be preserved.
- **Geometric compression algorithms.** To reduce the storage needed for geometric data both, lossy or lossless techniques can be used. Because general-purpose binary compression algorithms do not achieve optimal results, some other solutions had been purposed. One of them, called *Deering's approach [1]*, is based on normalization of every vertex position into a unit cube and reducing the fixed number of less important decimals of vertex coordinates.
- **Connectivity compression algorithms.** Usually, the mesh is represented as a set of vertices forming triangles and additional information of how these triangles is connected - topology. The main goal of these algorithms is to find a way, how to represent triangle mesh where connectivity data would take as little storage space as possible.

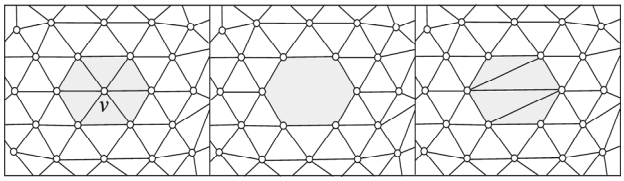
This article presents a progressive triangular mesh compression algorithm as introduced by Alliez and Desbrun [2]. Although the approach bases on a simple idea, it has good performances. The main contribution of our paper is a description of a new approach where server application uses distribution rules for a decision which client will receive data first and what amount of data will be transmitted. The solution bases on TCP/IP protocol using sockets. At the end, the practical results are given.

* timotej.globacnik@uni-mb.si

2 Progressive Compression of Triangle Meshes

In this Section we briefly describe the basic idea of algorithm, presented by Alliez and Desburn [2]. This algorithm enables lossless progressive compression of 3D triangle meshes and works in two steps named *valence-driven decimating conquest* and *cleaning conquest*. Similarly, the reconstruction procedure follows the same procedure but in the reverse order, i.e. *cleaning conquest* followed by *decimating conquest*. To understand the algorithm, used terms are described at first:

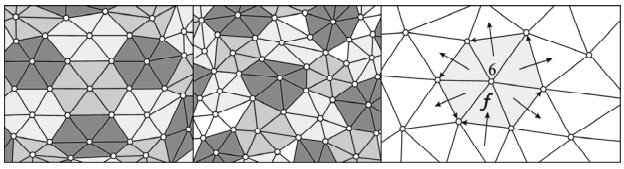
- *Valence of vertex v* corresponds to the number of vertices, connected to vertex *v*. In Figure 1a valence of vertex *v* is 6.
- *Patch* consists from central vertex and its incident triangles. Each triangle is defined with central vertex and two adjacent vertices. Edges, connecting all adjacent vertices form *patch border*. Figure 1a represents a patch with central vertex *v* surrounded by 6 adjacent vertices forming border of the patch.
- *Degree of patch* is defined by the number of triangles incident to the central vertex of the patch.
- *Vertex removal* is an operation, which removes the central vertex from the patch and retriangulates the hole (see Figure 1b, c).



a) b) c)

Figure 1: Operations on triangular mesh

- *Independent set* is a set of patches in the triangle mesh, where each triangle lies only in one patch. Independent set is called *optimal independent set* in the case when every triangle of the mesh is located in one patch (Figure 2a).
- *Null patch* refers to triangle, which does not belong to any patch of triangle mesh (white triangles in Figure 2b). In this case, triangular mesh is not optimal independent.
- *Gate* is an oriented edge, which enables moving from one patch to another (Figure 2c). The gate can be the input or the output gates. Each input gate defines its front face f.
- During compression or decompression, each vertex or triangle can have its *status flag* set to one of the three states: *free*, *conquered*, or *to_be_removed*.
- *Retriangulation tag* is associated to each vertex of the mesh. It can have two possible values: + or -. When vertex valence should lower during retriangulation process, its tag is set to -, on the contrary vertex tag remains +.



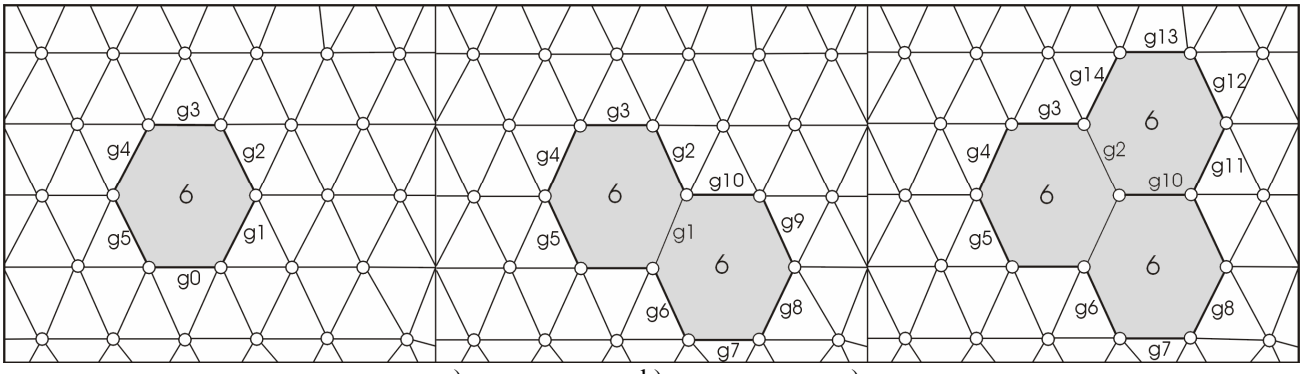
a) b) c)

Figure 2: An optimal independent set (a), null patches(b), gates(c).

The vertex can be removed from the mesh when:

- surface manifold property is not violated,
- normals of adjacent vertices are not flipped; this condition can also be ignored, because it has no influence on the compression algorithm, but it is important for rendering,
- user defined requirements are satisfied (for example, to preserve the shape of the mesh also at the low level of details).

The main idea of the compression algorithm can be explained using Figure 3. Vertices with valence $d \leq 6$ are removed during mesh traversal, while leaving their patch borders intact. At the beginning of compression procedure, first gate g_0 is randomly selected from all edges of the mesh. In Figure 3a, the shaded patch is entered through the gate g_0. The central vertex and its incident faces are removed, status flag of border vertices is set as *conquered* and 5 output gates are pushed into FIFO, which now contains the following gates: g_1, g_2, g_3, g_4, g_5 (Figure 3a). In the second step (Figure 3b), gate g_1 is taken from the queue, the central vertex and its incident faces are removed, again status flags of border vertices are set and another 5 gates are pushed into the queue, which now contains g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}. The same procedure is applied with the third patch



a) b) c)

Figure 3: Removing central vertices

(Figure 3c). Process continues until there are any gates in the queue.

As we can see, the way compression algorithm traverses the mesh depends on the order in which gates are pushed into FIFO. Therefore, to allow the proper reconstruction of the mesh both, compression and reconstruction algorithm must generate output gates in the same order. Furthermore, to avoid patch overlapping, next solution should be considered. If status flag of the central vertex of the patch is set as *conquered* this patch has already been visited therefore current gate is discarded and next one is taken from the queue.

To keep the patches nicely triangulated, adaptive triangulation method is used according to the valence of its central vertex and retriangulation tags of patch border vertices. The retriangulation tags of patch border vertices are set during mesh traversal using rules proposed by both authors. The proper rule is chosen according to the valence of central vertex of the patch and retriangulation tags of both vertices defining input gate of the patch (see Figure 4 for an example). These rules make retriangulation deterministic, thus proper reconstruction of the mesh is guaranteed. Sometimes it can happen that some vertices of the patch have their tags already set – usually when adjacent patch has already been conquered. In that case, tags of those vertices are left unchanged, but patch is still retriangulated according to the rules. Since the reconstruction algorithm performs these operations in same way deterministic triangulation of the patch is preserved.

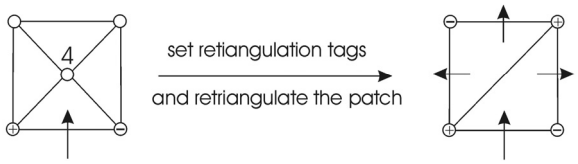


Figure 4: Set retriangulation tags of degree 4 patch.

After *decimating conquest*, the majority of vertices in the mesh would have their valence 3 or 9 (see Figure 5a). Therefore, to improve mesh regularity *cleaning conquest* is performed. Cleaning conquest removes only valence 3 of vertices. As can be seen on Figure 5b, patches of degree 3 are separated by a triangle. After removing the vertex with valence 3 gates, which are put into queue, are the two edges of each triangle, adjacent to the patch border. Consider an example in Figure 5b. When patch marked by the arrow is entered through the input gate, the central vertex is removed and 4 new gates (marked by arrows) are sent into the queue. Finally, status flag of each triangle adjacent to the patch is set as *conquered*. When cleaning conquest is terminated, the valence of remaining vertices is the same as before decimation conquest (see Figure 5c). At the end, the data are compressed by the arithmetic coder [4].

Mesh reconstruction is performed by the inverse procedure.

3 Network application

Even after a triangle mesh is compressed, it can still take considerable amount of memory space. Nowadays this does not represent a problem, if the compressed data are already on the computer, but can drastically slow down the reconstruction (and visualization), when data are transferred to the remote computer. To solve these problems, the users are hiring faster internet connections, or distribute the load to multiple servers connected into clusters. Since all these solutions increase the costs of system maintenance, we provide single-server solution, where server has the possibility to decide – using *distribution rules* – which client will receive data first and what amount of data will be transmitted.

3.1 Communication between server and clients

The application is divided into two parts: a server and a client application. The *server application* maintains three main tasks:

- receive requests from clients,
- select an active client (the client which will receive data first) and
- transmit data to the selected client.

The main tasks of the client application are:

- send requests to server,
- receive data from server,
- reconstruct the received mesh and render it on the screen.

To make communication between both applications possible also over the internet, *TCP/IP protocol* was selected. If we look at Figure 6, we can distinguish two

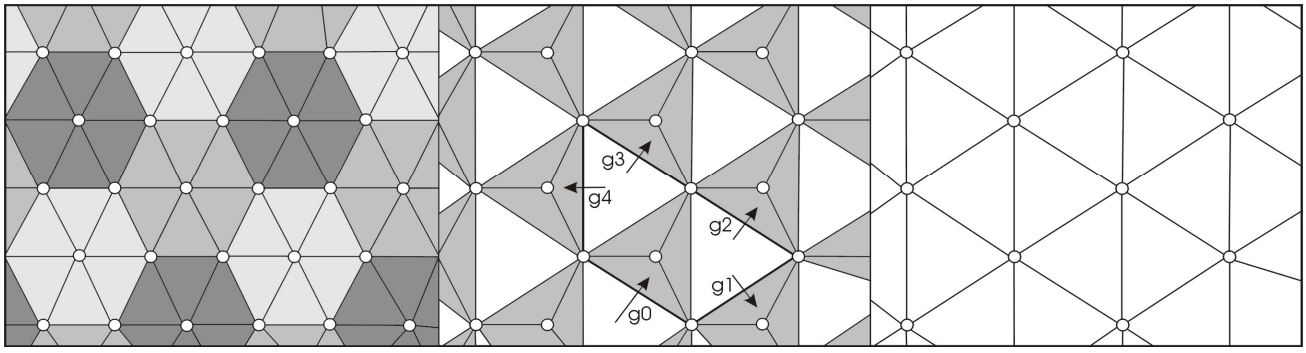


Figure 5: Triangle mesh before decimating conquest (a), triangle mesh after decimating conquest (b) triangle mesh after cleaning conquest (c).

types of data being sent between server and client: *client requests* and *mesh data*. As the time needed for client requests is much shorter than for transferring the mesh data, we decided to send them through two channels *sockets*.

As we described in Section 2, triangle mesh is compressed by alternatively performing decimatoin and cleaning conquest. Each process creates a sequence of vertices at its output. These sequences are then compressed using arithmetic coder and saved into separated file (also called *data packet*).

After the client connects to the server, it sends *get_mesh_list* request through the request channel. The server responds by sending list of compressed meshes stored in its mesh database over the data channel (see Figure 6). From that list, the user selects the desired mesh by sending *get_base_mesh* request to the server over the request channel. When, this request is selected by the distribution rule, the base mesh is loaded from the mesh database and sent to the client. The client starts immediately with the reconstruction and rendering of the mesh. After the base mesh is rendered, the client requests following data packet by sending *get_next_packet* request. Because server already knows which packet was transmitted before, no additional data is required from the client. The next packet is loaded from the mesh database and sent to the client. Procedure is finished, after the last packet is received, thus mesh is fully reconstructed. However, the time needed for the reconstruction of the mesh, depends on selected distribution rule.

3.2 Distribution rules

Four distribution rules were implemented and each of them evaluates the client request in a different way. The main aim is to achieve the fastest response of the server to client's requests as well as maintaining server utilization at the lowest possible level. All these rules are uninterruptible, i.e., the next client is selected only after previously selected one received the requested data.

First In First Out (FIFO)
Server sends the data to the first client in the *client requests* queue. After that, second request is served, then third and so on. This rule is commonly used in the most client – server applications, where server can send the data to only one client at a time. If the first client in the queue has very slow connection speed, sending data to that client would make response times to all other clients significantly longer.

Least Completed First Served (LCFS)
The rule selects the client, which received the smallest amount of data up to now. This means that the client, which sends the first request, would have higher priority than others what allow it to start reconstruction of the mesh immediately. Using this rule, the first passes of the mesh are reconstructed faster than the others, if there are many client requests, waiting in a queue. However, if many clients send their first request at the same time, some clients can wait very long, before their requests are served.

Smallest Packet First (SPF)
Server checks the sizes of all packets requested by the clients and selects the smallest one. Because packets needed at the beginning of the mesh reconstruction are usually smaller than the latter ones, the client can wait longer to receive the final packets and to finish the mesh reconstruction.

Fastest Connection Speed First (FCSF)
Client with the higher speed connection get the data first. However, when client requires first data packet, its connection speed is automatically set to the fastest. After that, server calculates the time needed to transfer each packet to the client and sets its priority accordingly. This rule also offers another option where the server could send to client with the high speed connection two packets in a row, while slower clients get only one packet at the time.

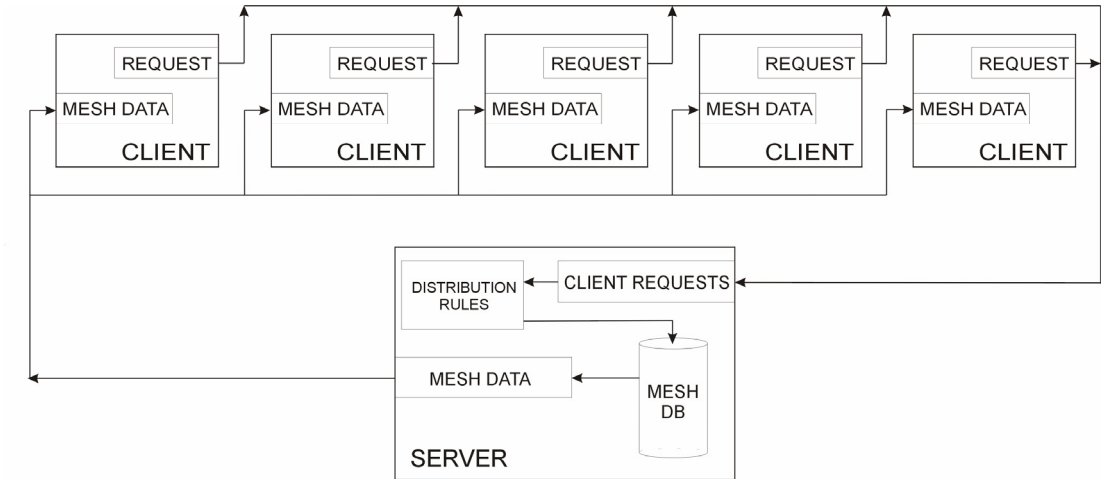


Figure 6: Architecture of our system.

4 Results

Four distribution rules, described in previous section were tested with real application. We have chosen four progressively compressed trianglular meshes (Figure 7), where each mesh have been saved as a sequence of data packets – files. Thus first mesh, Ball Joint, consists from 10 packets where packet 1 represents the base mesh, packet 2 first pass, packet 3 second pass, etc. Second mesh, Club, is composed from the base mesh and 4 passes. Dinosaur is represented by 7 packets while Venus needs 10 packets to be fully reconstructed. Sizes of packets of these meshes are presented in Table 1.

Mesh name	Packets [kB]									
	1	2	3	4	5	6	7	8	9	10
Ball Joint	71	4	2	5	3	8	4	11	6	14
Club	476	7	21	13	25	/	/	/	/	/
Dinosaur	84	2	4	2	6	4	8	/	/	/
Venus	70	4	2	5	3	7	4	11	6	14

Table 1: Sizes of every data packets of each mesh, used for testing purposes.

These meshes were stored on the server - an AMD Athlon 1400+ with 384MB of RAM, running Windows XP operating system. Its connection speed was ADSL 1024/256. For testing purposes, we have used five clients with different connection speeds. The first one, (*Client1*) was connected to the internet through an analogue modem 33.6 kb/s, the connection speed of the second client (*Client 2*) was ADSL 1024/256 kb/s, while the last three clients (*Client 3, Client 4 and Client 5*) communicated with server through ADSL 2048/384 kb/s communication line.

We tried to evaluate the distribution rules with three different approaches. In the first one, the same mesh (Ball Joint) was simultaneously requested by all three clients. For each of the client, we estimated the time needed to receive all data packets of the mesh (see Figure 8). The test was performed using all four distribution rules. The results can be seen in Table 2.

Rule / Client	Estimated time in seconds.			
	FIFO	LCFS	SPF	FCSF
1 – Ball Joint	48.154	48.713	48.658	45.164
2 – Ball Joint	44.210	43.952	9.951	28.113
3 – Ball Joint	44.642	44.591	7.691	30.244

Table 2: All clients requested the same mesh.

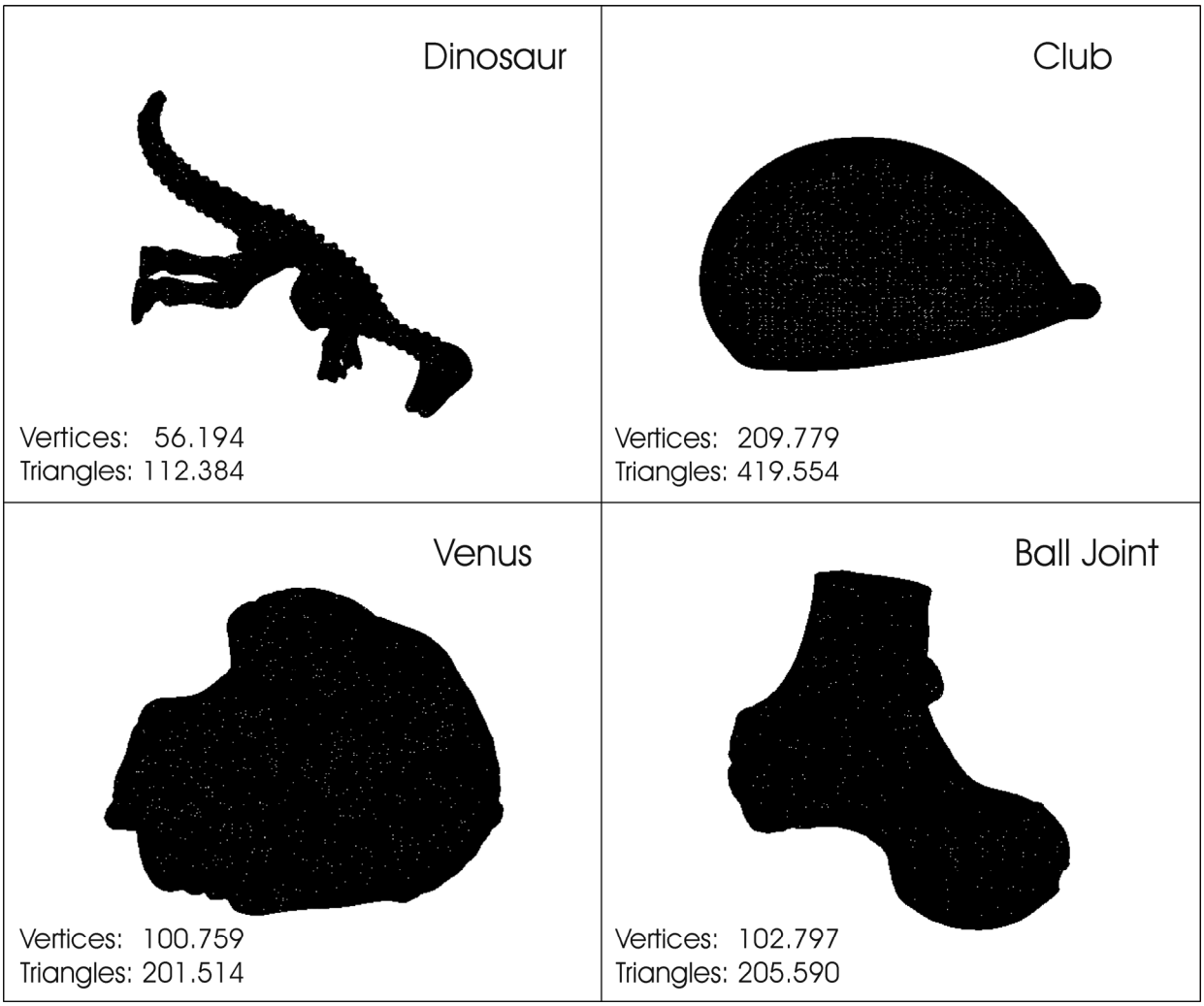


Figure 7: Four testing examples.

If we look at the results, estimated times are almost identical, when FIFO rule was used, because the two clients with faster connections must wait for the slower one to finish transferring its packet.
Since LCFS rule in this case gives the same priority to all clients (because all of them requested the same mesh), the results are very similar to those, estimated with FIFO rule.
SPF gives the best result in this test because first packet of the mesh is the biggest in size, therefore the client which first received the packet 1 has higher priority than other clients. The same rule could also be the worse one, if the slowest client would requested the packet 1 first.
When FCSF rule is used, all three clients must receive packet 1, before their speed is calculated. After that, faster clients receive the data before slower ones. We can see that the times of both fast clients are almost identical, because server upload speed is limited to 32kB, so none of them can receive data with more than 32kB/s.

To see how efficient are the rules, when different meshes are requested, we repeated the above procedure, but in this case, each client requested different mesh.

Client \ Rule	Estimated time in seconds.			
	FIFO	LCFS	SPF	FCSF
1 – Venus	58.413	59.217	56.022	58.413
2 – Dinosaur	48.282	57.162	30.131	42.010
3 – Club	44.687	57.920	57.772	41.800

Table 3: Different meshes were requested.

Again, in the case of FIFO rule, even when meshes required by the faster clients are shorter than the mesh, requested by the slower one, the times of all three are very similar.
LCFS rule results are even worse, because the fastest two clients must wait for the slower one before their request can be complied.
Again, SPF rule gives higher priority to client which requested the smallest mesh. For that reason, mesh Dinosaur is transferred much faster than other two meshes.
When FCSF rule is used, clients with faster connections have the advantage and thus, shorter transfer times. This time could be even shorter if Client 1 would requested mesh with smaller first packet – because all clients have the highest priority before they receive first packet.

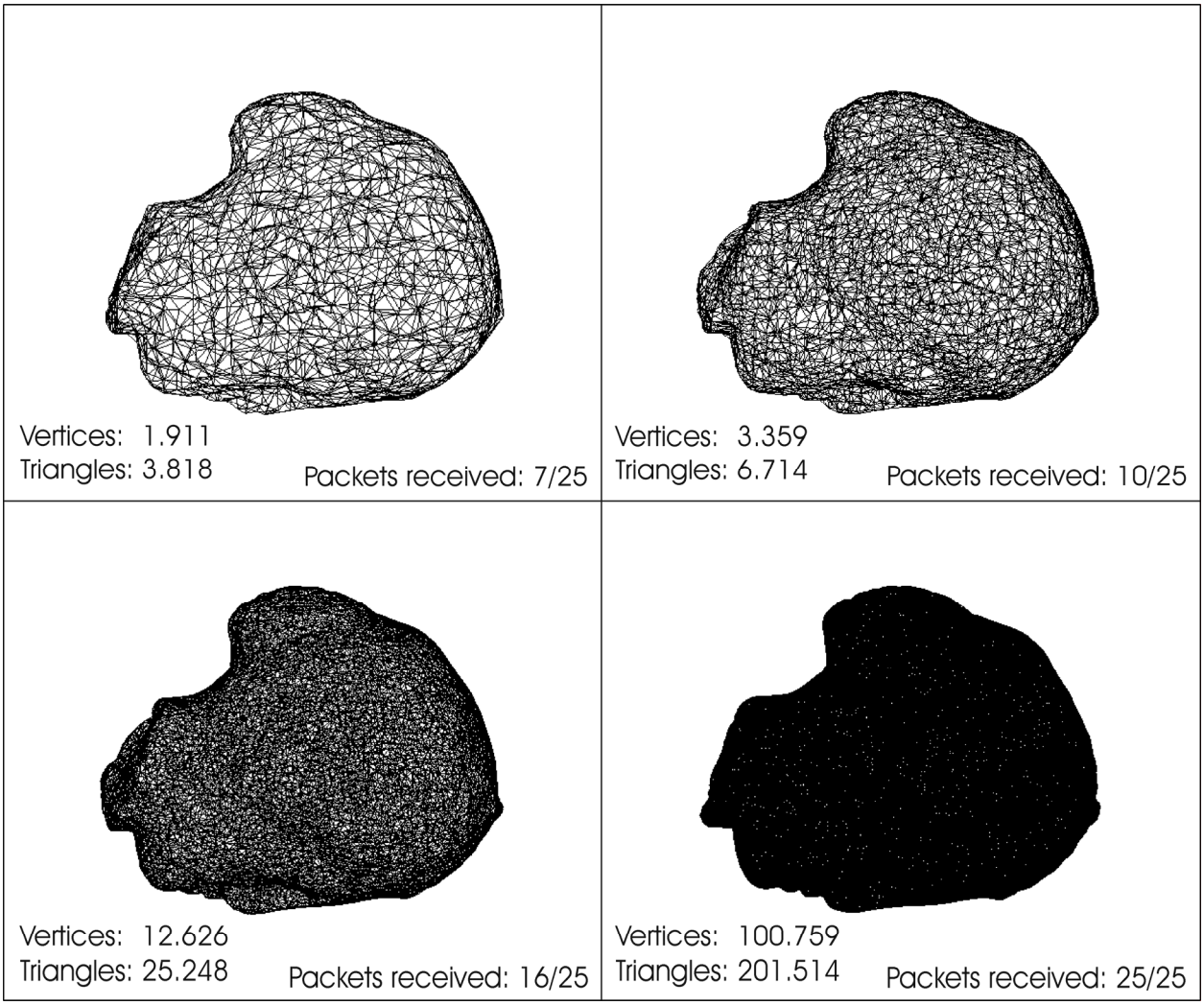


Figure 8: Reconstruction of triangle mesh Venus.

In third approach, we wanted to see, how the number of packets influence the efficiency of the individual rules. This approach is similar to the previous one, however this time all clients have the same connection speed.

Client \ Rule	Estimated time in seconds.			
	FIFO	LCFS	SPF	FCSF
3 – Venus	26.396	23.513	24.149	25.947
4 – Dinosaur	25.312	26.447	7.083	24.904
5 – Club	24.335	21.430	26.227	24.037

Table 4: Different meshes were requested by clients with equal connection speeds.

The only rule, which is affected by the size of data packets is rule SPF. As we can see, the smallest mesh (Dinosaur), having half number of vertices of mesh Venus and a quarter number of vertices of mesh Club is transferred much faster than the other two meshes. However, the times of other two meshes are very similar because server chooses the client faster that it can send request for new data packet thus in that time, there is only one request in the queue.

5 Conclusion

It is obvious that the selection of efficient rule is essential to achieve the good response times and keep the number of requests at the lowest possible level. In our case, the only appropriate and useful rule, which satisfies both conditions mentioned above, is rule FCSF. FIFO and LCFS rules are to much affected by the connection speed of the clients, while SPF enables fast reconstruction of smaller meshes. However, the rules could be more efficient, if the server would interrupt the transfer after expiration of certain time period giving opportunity to other clients.

References

[1] M. Deering. *Geometric Compression.* Computer Graphics (Proc. SIGGRAPH), pages 13 – 20, August 1995.

[2] P. Alliez and M. Desbrun. *Valence – driven Connectivity Encoding for 3D Meshes.* EUROGRAPHICS, pages 480 – 489, 2001.

[3] M. Schindler. *A Fast Renormalization of Arithetic Coding.* In Proceedings of IEEE Data Compression Conference, Snowbird, UT, page 572, 1998.

[4] I.H. Witten, R.M. Neal, and J.G. Cleary. *Arithmetic Coding for Data Compression.* Communications of the ACM, 30(6), 1987.

[5] A. Khodakovsky, P. Schroder, and W. Sweldens. *Progressive Geometry Compression.* In ACM SIGGRAPH 00 Conference Proceedings, pages 271–278, 2000.

[6] Jingliang Peng, Chang-Su Kim, and C.-C. Jay Kuo. *Technologies for 3D Triangular Mesh Compression: A Survey.* Submitted to Journal of Visual Communication and Image Representation, 2003.

[7] G. Taubin and J. Rossignac. *Geometric Compression through Topological Surgery.* ACM Transactions on Graphics, Vol. 17, No. 2, 1998; and IBM Technical Report RC-20340, January 1996.

[8] *W*. J. Schroeder, J. A. Zarge, W. E. Lorensen, *„Decimation of Triangle Meshes"*, Computer Graphics, Vol. 26, No. 2, 1992, pp. 65-70.

[9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, *„Mesh Optimization"*, Proceedings of the 1993 ACM SIGGRAPH conference, 1993, pp. 19-26.