MECAC WEEKLY REPORT (April 8 – April 14)

This week we have mainly worked on decreasing network traffic by utilizing well known compression algorithms. We have dwelled on lossless compression algorithms more as we need precise position updates and messages in between clients of Virtual Turkey. However lossy compression algorithms could be future research for MECAC as they tend to decrease network traffic considerably. One major issue when using lossy compression algorithms for MMOGs is that the algorithm should have variable precision for each client. Near-by clients position updates should be more precise than those of far away. A fourier transform based compression algorithms can be used for such application. Cinar has researched current approeaches to lossy compression which we can utilize for Virtual Turkey. However, as a culmination of the research for both lossy and lossless compression algorithms we have decided to use a lossless compression algorithm for our purposes.

There are two well known lossless compression algorithms for which we have implemented in C#. These are LZ77 and LZ78 (1), which are mainly known for gzip encoding. Mert and Umit have investigated feasibility of those algorithms. LZ77 algorithms achieve compression by replacing portions of the data with references to matching data that have already passed through both encoder and decoder. A match is encoded by a pair of numbers called a *length-distance pair*. While the LZ77 algorithm works on past data, the LZ78 algorithm attempts to work on future data. It does this by forward scanning the input buffer and matching it against a dictionary it maintains.

Despite the fact that LZ78 algorithm can provide better compression rates, we have decided to use LZ77 since it requires single pass over the data without forward scanning. We have used Umits report for previous week to estimate average size of the messages that are transmitted between server and client. For a message size of approximately 300 bytes, LZ78 algorithm unnecessarily scans the message couple of times wheras LZ77 algorithm immediately deflates the message with a single pass on it in a linear time. Following example gives a picture of how we compressed the messages from server to client.

32 bit unique client identifier	64 bit Client position, two 32bit floats.	32 bit unique client identifier	64 bit Client position, two 32bit floats.	….

A typical update message has the form as above. When a server distributes such messages with our EventMatrix class, it compresses a number of client updates to a single one. We have LZ77 and LZ78 interfaces implemented on both server and client side. We use following function to compress a message on the server side.

```
public void CompressLZ77(out String outgoingMessage)
```

Compress message scans the message from left to right to match the previously found bit patterns. The encoder and decoder must both keep track of some amount of the most recent data, such as the last 2 kB, 4 kB, or 32 kB. The structure in which this data is held is called a sliding window, which is why LZ77 is sometimes called sliding window compression. The encoder needs to keep this data to look for matches, and the decoder needs to keep this data to interpret the matches the encoder refers to. In our case, we scan over client positions in Virtual Turkey and check if bit patterns are repeated. If so, we replace the sliding window with special marker to indicate a reference to previously unencoded data. Since the client positions are stored as floats which obey to IEEE single-precision floating point standards (3). The compression rate is over 50% on average.

When a message is received on client side following function is utilized on client side to decompress the message and push the updates to the graphics component.

```
public void DecompressLZ77(String incomingMessage)
```

Note that these compression algorithms are only utilized during bulk status updates from server to client. When a client uploads his own position to the server, no compression is used since such a compression would increase latency since a linear pass over the message is required for both client and server side.

For the implementation of the Zip library we have used GZipStream class (2) which provides a nice interface to handle compression and decompression on strings which are on memory. Using System.IO.Compression.GzipStream package we were able to integrate LZ77 to Virtual Turkey.

We have also uploaded Server package to both svn server in MODSIMMER and those that are in CENG. Since the Client package is over a gigabyte, we have decided to keep local copies of the client, while maintaining the server source code through svn interface.

svn+ssh://e1560051@external.ceng.metu.edu.tr/depo/svn/490.2010/mecac/ working_server

Server package is under working_server directory given as above.

References :

1) http://en.wikipedia.org/wiki/LZ77_and_LZ78
2) http://msdn.microsoft.com/en-us/library/system.io.compression.gzipstream.aspx
3) http://en.wikipedia.org/wiki/Single_precision