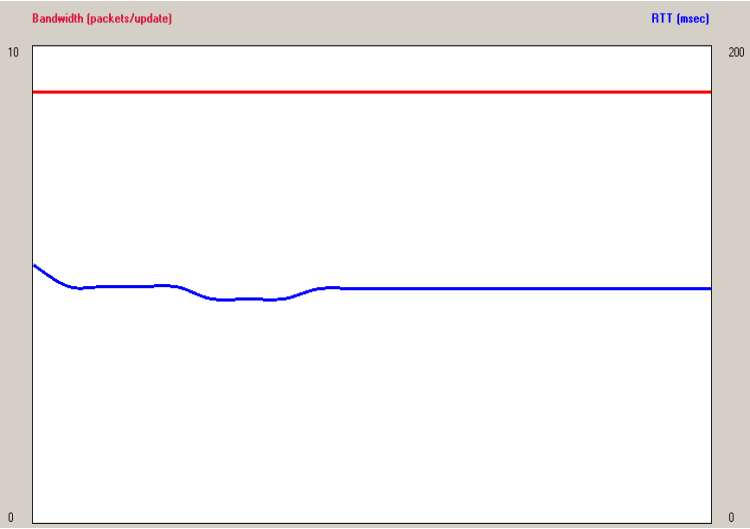
MECAC WEEKLY REPORT (April 29 – May 5)

This week we have first worked on bandwidth visualization tool on server network component of the Virtual Turkey. We have then investigated the causes of bugs that we have found by observing the visualization software that we have developed. We have finally investigated occlusion culling possible libraries to use, and how to integrate occlusion culling to area of interest calculations on the server side.

First of all, we have implemented a bandwidth visualization class that is located on server network component. The class works upon a click on the management tool on the server interface. The interface of the bandwidth visualizer graphs the maximum allowed bandwidth and RTT collectively on single canvas. The tool also updates the graph upon receiving new information from the server network component reflecting a change on bandwidth and or average RTT computed. A screen shot for the visualization tool has been given below to depict what we have done during the first couple of days in this week.



The Bandwidth variable drawn with red pen shows maximum number of allowed packets per network update. The RTT denotes the average round trip time summed over each client connected to the server and normalized by the number of clients. As we have given the algorithm previous week, we have TCP like congestion control mechanism that balances the network traffic according to a statistical information about the congestion on routers.

The network simulation for the congestion control mechanism implemented in our server network component turned out to be much harder than visualizing it. We have first tried to randomly trigger sleeps in test clients. When test clients sleep, they imitate congestion on the routers between server and the client. However this imitation like all others is fake, and cannot be resolved by the congestion control mechanism implemented in our algorithm. Our algorithm tries to keep the RTT time of some playable level by reducing the bandwidth. Bandwidth reduction however does not cause the simulated clients to reduce their RTT in response as a result we needed much more complex simulation algorithm to alleviate this problem.

Bandwidth visualizer tool has also enabled us to spot couple of bugs in congestion control mechanism. Most important of which was the maximum number of packets that can be pushed through the link that connects server to the internet. When tested locally, with servers and test clients running both on a local machine, roundtrip time is unchanged with respect to bandwidth since all packets are already sent out requiring no extra bandwidth usage. When a server is heavily loaded this kind of bug is even harder to spot since the bandwidth wont sky rocket since it will be constantly adjusted according to average RTT by our congestion control algorithm. However for this special case we have tested, with server and clients running on the local computer of us, bandwidth unnecessarily increases. Even worse, the bandwidth overflows to negative numbers and blocks whole communication of the server for such a simple test environment. Like this and couple of others, we have used bandwidth visualizer tool to spot bugs and correct them.

Perhaps one of the most important deliverables of our project was to integrate occlusion into our area of interest calculations. In 3D computer graphics, hidden surface determination (also known as hidden surface removal (HSR), occlusion culling (OC) or visible surface determination (VSD)) is the process used to determine which surfaces and parts of surfaces are not visible from a certain viewpoint. A hidden surface determination algorithm is a solution to the visibility problem, which was one of the first major problems in the field of 3D computer graphics (1). When we integrate hidden surface removal to our area of interest calculations we can reduce the network overhead significantly by simply not sending position updates of invisible players in Virtual Turkey. Umit has worked on occlusion culling techniques and possible libraries that we could use to integrate the occlusion culling mechanism to our area of interest calculation. There is couple of approaches that we could take to achieve this.

As a result of literature search, two common types of occlusion culling are *occlusion query* and *early-z rejection* (1), (2):

Early-z Rejection: In the rasterizer stage of the rendering process, early-z compares the depth value of a fragment to be rendered against the value currently stored in the z-buffer. If the depth test is failed (the current pixel is behind the pixel in the Z-buffer), the fragment is not visible and rejection occurs, otherwise the fragment is shaded and its depth value replaces the one in the Z-buffer. This approach requires sorting the objects from front to back.

Occlusion query: It is fundamentally simple. It determines how many pixels should have ended up visible on-screen. These pixels successfully passed various tests at the end of the pipeline, such as the frustum visibility test, the scissor test, the alpha test, the stencil test, and the depth test. Occlusion query uses simple bounding boxes instead geometrically complex objects. If the object's bounding box ends up occluded by other on-screen polygons, we can avoid the object entirely. There's an important difference between occlusion query and early-z rejection. Early-z occurs at the rasterization level, whereas Occlusion query rejects polygons at the geometry level.

For example server could determine the hidden players at run time and dynamically determine which position updates are unnecessary. However this would result in huge computation demand on the server side and would require us to consider the tradeoff between computational power and the network overhead. There is one more feasible approach that we have chosen to go for. We can precompute each viewport from each point offline and use this information when the MMOG is up and running. This has one obvious disadvantage if implemented carelessly. When the map changes the precomputation information becomes stale, and as result inconsistent occlusions can result in invisible players on some spots that were otherwise perfectly visible and not occluded. Therefore, we need to recompute this occlusion map on each change in the virtual environment. Next week we will focus on approach described in this document and visualization of it in the server management component.

References

(1) http://en.wikipedia.org/wiki/Hidden_surface_determination
(2) Effective Occlusion Culling for the Interactive Display of Arbitrary Models, Hansong Zhang