MECAC WEEKLY REPORT (May 5 – May 11)

This week we have worked on integrating occlusion culling on area of interest computation. The area of interest calculation was performed in EventMatrix class of server network component. Mert has worked on integration of the occlusion computer that Umit and Cinar has implemented.

```
private double scoreUrgency(Client from, Client to)
{
    Vector3 posFrom = from.Character.Position;
    Vector3 posTo = to.Character.Position;

    return Math.Sqrt(Math.Pow(posFrom.X - posTo.X, 2) + Math.Pow(posFrom.Z - posTo.Z, 2));
}
```

Script 1: Current area of interest computation

Currently, the urgency of each message is computed according to distance between sender and receiver. This approach is definetely flawed considering the occlusions in between. For example sender and receiver may be close but seperated by a large mountaion. In this case the urgency algorithm we have implemented above will determine that the update messages from sender to receiver is really urgent and should be transmitted even in low bandwidth conditions. It is obvious that this message is not that important and even has no effect in graphics calculations of the receiver since the player is occluded and not seen. Such position updates also creates liabilities to cheating as some players can trace the network comunication and see players that are invisible otherwise. This situation is illustrated in the above case where our current area of interest computation fails to find an effective solution.

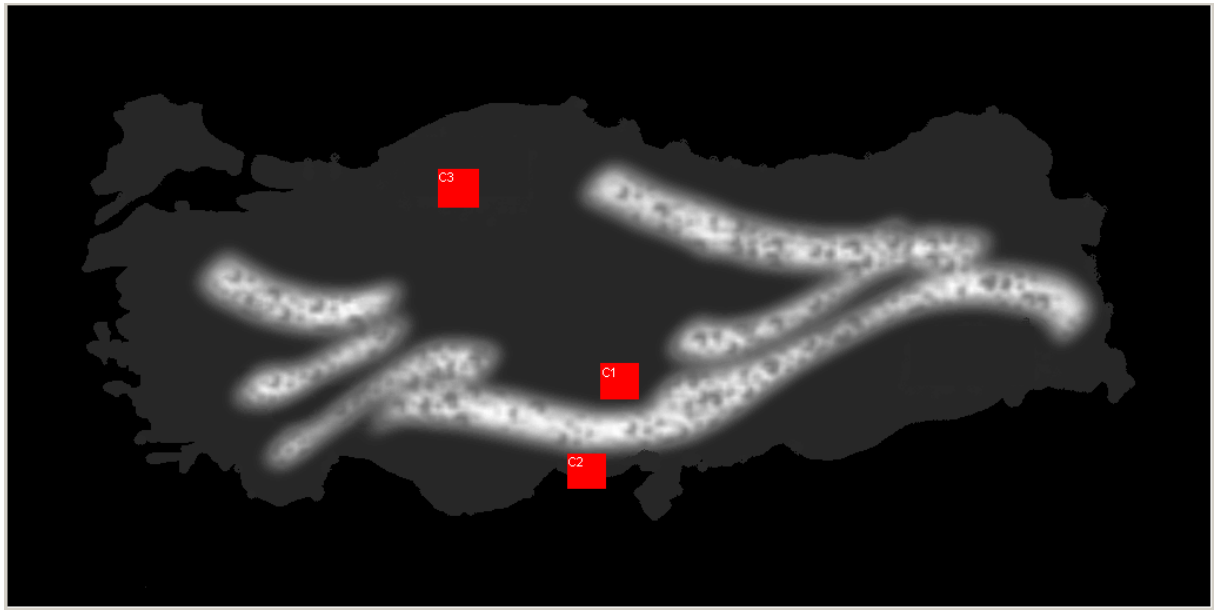

Figure 1. The case where current area of interest computation is not effective.

To solve this problem Umit has worked on precomputation of occlusions for each position in the virtual world of MMOG. We have taken two step approach to compute the area of interest. First we pass over a predetermined circle to find possible occluding objects. To determine which points are occluding in the map we have integrated the heightmap data that is serialzed from client to the server. We have used binary serializer in c# to write the height map data of the field to a file named “MyFile.bin”

```
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Create, FileAccess.Write, FileShare.None);
formatter.Serialize(stream, field.Array);
stream.Close();
```

Script 2: Serialization of the height data in the client component.

Above code script is taken from client source code where we serialize “field” data to a file. We have then used this file to deserialze the float array and use it in occlusion computation in the server component. Figure 1 is drawn from the field data in the server network component. In figure 1, white locations are the highest points on the map, and black locations are the lowest points of height.

We have used this heightmap data to determine the occlusions that stem from mountains in Virtual Turkey. In the first pass we have computed the height difference between each positions in the map. We have marked the positions that have greater height difference than a threshold as “occluding positions”. The threshold is the height of the player that wanders around the virtual map.

In the second pass we determined the visibility of map positions according to following algorithm:

1. For each position P in the map
   For each position Q in the map such that distance(P,Q) < circle_threshold
   Let S be the set of “occluding posisionts” for P
   For each point X in set S
   1. İf euclid_distance(line(P,Q),point X)) < threshold
      Return “ occluded “
   2. Else
      Return “ visible ”

Cinar has worked on storage mechanism for the computation results from Umit. We have designed simple MySQL table to store visible area points for each position. Determination of a positions visibility relative to another position takes a single query to this table. Table schema is as follows:

```
VisiblePoints {
        Point source
        Point destination
}
```

To check the visibility of a point A relative to B, we look if that row exists in the database by following query :

```
SELECT * FROM VisiblePoints WHERE source='A' AND destination='B'
```

This query should take less than milisecond to finish and easy to use in our area of interest computation. We have then integrated this result to sort the messages according to the urgency. The 'scoreUrgency' function given in script 1 has been changed to also consider if the locations are visible to each other.

Currently we have not integrated eye angle to the determination of the visiblity. But the approach we have used could be improved to include that information as well. This is better done dynamically since we just need to implement a component that determines if a point falls into eye range.

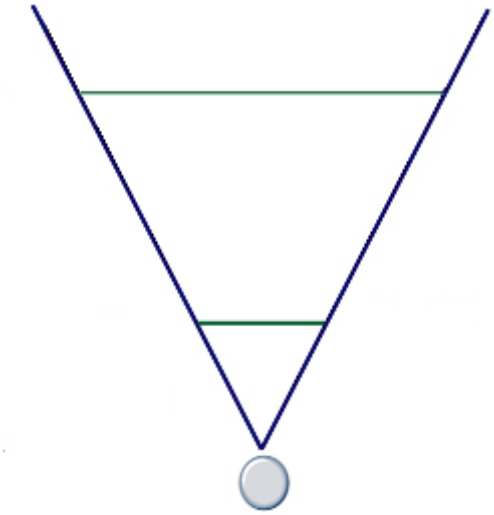

Figure 2. Integration of the view angle to the urgency computation

Score urgency function would simply check if a point falls into the view angle illustrated as above. This small modification would significantly reduce the area of interest each client and therefore the network communication. This remains as a future work for MECAC.