‘TrakWorks’ Brief

The ‘TrakWorks’ system is a networking system that allows for reading/writing game object data across a local area network (LAN), via a server-client relationship.

Objective

The objective of the ‘TrakWorks’ system is to:

- Send game object data across a LAN network
- Allow for custom and unknown game object data
- Undemanding integration into other projects

3rd Party Libraries

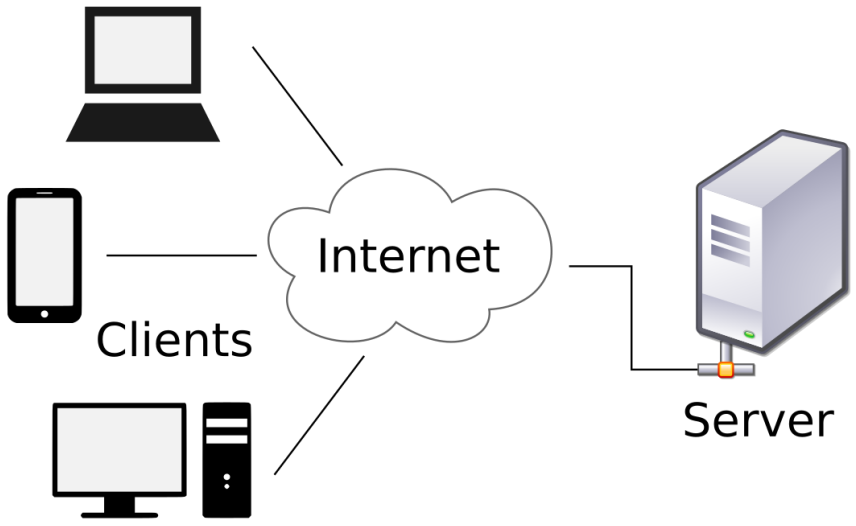
Third party libraries to be utilised:

- RakNet
  - A networking middleware developed by Oculus VR
- Bootstrap
  - Graphics middleware with basic application functionality
- OpenGL
  - Programming interface for rendering vector graphics

Mathematical Operations

The main mathematical operation involved in developing a networking system is network theory. This defines networks as a series of graphs made up of nodes and edges, each node being a client/server and each edge being the connection between them.

Network theory helps us to understand the relationship between the client/s and/or server/s, by analysing this relationship we can identify any flaws or improvements to the system. The image below demonstrates a simple client/server relationship:

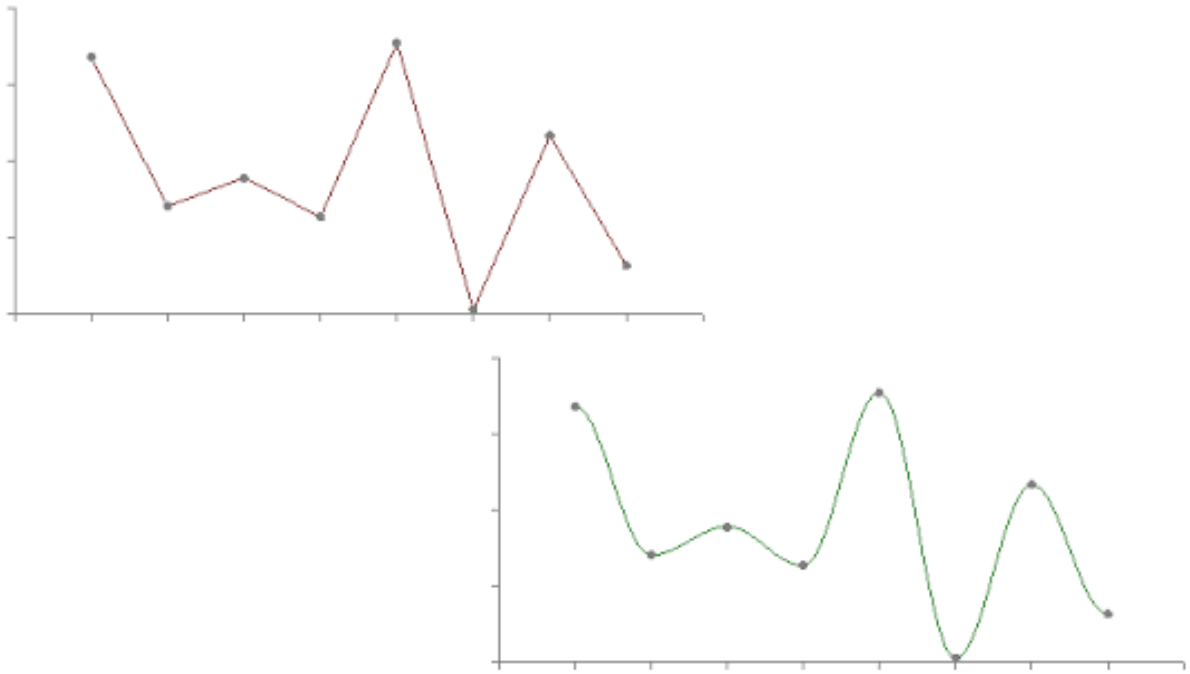


‘TrakWorks’ Brief

Mathematical Operations Cont.

Another mathematical operation used in networking is number theory. This is simply the study of relationships between numbers. Number theory helps us understand how different sub-systems of the network interact with each other, allowing for us to take complete control over the functionality and optimization of the system.

Another mathematical operation that will be implemented into the ‘TrakWorks’ system is cosine interpolation. Cosine interpolation is the simplest interpolation function that offers smooth transitions between each adjacent segment, or in our case between each frame update. It works by orienting a piece of a cosine wave in such a way that it provides smooth transitions between each point. Below is the difference between linear and cosine interpolation:



Code snippet for a cosine interpolation function:

```
double CosineInterpolate(
   double y1,double y2,
   double mu)
{
   double mu2;

   mu2 = (1-cos(mu*PI))/2;
   return(y1*(1-mu2)+y2*mu2);
}
```

‘TrakWorks’ Brief

Advanced Algorithms

There are many algorithms involved in networking but the algorithm I will be implementing in the ‘TrakWorks’ networking system is universal data packaging.

Universal data packaging in this scenario is an algorithm that will take any form of data and package it to be used within the ‘TrakWorks’ server-client system. This will be done by using a base ‘Data’ class that the user can override to contain any data they want to be transmitted across the server. For example, if the user wants to create their own player class they can then create a ‘PlayerData’ class that contains all the player data.

This can then be extended to contain all data types, both built in types (float, int, etc.) and custom types (player data, entity data, etc.). So the user can control exactly what data will be transmitted between clients.

Integration

For the ‘TrakWorks’ system to be integrated into other applications the client project will be compiled into a dynamic linked library (.dll), while the server project will remain a separate executable (.exe). In this way any other developer can install the .dll and .exe files and link them to their C++ project via Linker.

The user will then store a reference to a client within their class, using this reference the user can connect their class to the server. Custom client data can then be passed through this client reference to be sent to the server. The server will be setup so that it can receive any derived ‘Data’ class.

Modularity

The ‘TrakWorks’ system will be modular in the sense that the use of its functions and input parameters will be overridden by the user. I.e. for the client input data a generic data type will be created that the user can override to replace with their own custom data. This will allow for the system to be integrated into almost any application.

Another way in which this system will be modular is how it handles client ‘game objects’, as they too will be overridable. This is slightly different from how the data works as instead of the member variables being overridable the individual read/write functions will be virtual to allow overriding. For instance, the user can overwrite the write function to send an additional message to all clients on the server, i.e. printing a custom message.

Game objects will have a protected ‘Data’ variable that is setup to read and write any size data. In this way the system is inherently modular as the end user can customize exactly what data is sent across the LAN server. The end user will also extend the game object read/write functions by overriding them to do exactly what they want.

‘TrakWorks’ Brief

References

- *RakNet documentation* (no date) *RakNet: Main Page*. Available at: http://www.jenkinssoftware.com/raknet/manual/Doxygen/main.html (Accessed: May 2, 2023).
- *Introduction to network theory - cl.cam.ac.uk* (no date). Available at: https://www.cl.cam.ac.uk/teaching//1011/PrincComm/slides/graph_theory_1-11.pdf (Accessed: May 2, 2023).
- *Number theory* (2023) *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/science/number-theory (Accessed: May 2, 2023).
- (no date) *Interpolation methods*. Available at: http://paulbourke.net/miscellaneous/interpolation/ (Accessed: May 2, 2023).
- *Client-server model* (2022) *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/client-server-model/ (Accessed: May 4, 2023).