CSE 190 Discussion 7

Final Project:
Collaborative VR

Final Project: Collaborative VR

- The assignment is up on the webpage: http://ivl.calit2.net/wiki/index.php/Project4S18
- Due on Tuesday of the Finals week (June 12th) at 3:00pm
- You will create a networked VR application
- Some features you need to implement:
  - Make use of 6 dof head tracking
  - Make use of touch controllers
  - Two users should be able to see each other’s head and hand position.
  - Come up with an application that requires collaborative work between two users.
  - Make use of audio
  - Create at least one 3D object on your own.
  - Details in the assignment page.

Networking

- You have a few options for which networking libraries you want to use.
- Sockets (Recommended)
  - More low level, sending bytes over the network
  - Sample code here: https://www.codeproject.com/Articles/412511/Simple-client-server-network-using-Cplusplus-and-W
- RPC
  - Call functions over the network
  - Documentation here: http://rpclib.net/
- You can use other, more extensive libraries for networking online, but it will be up to you to figure out how they work.

Sockets

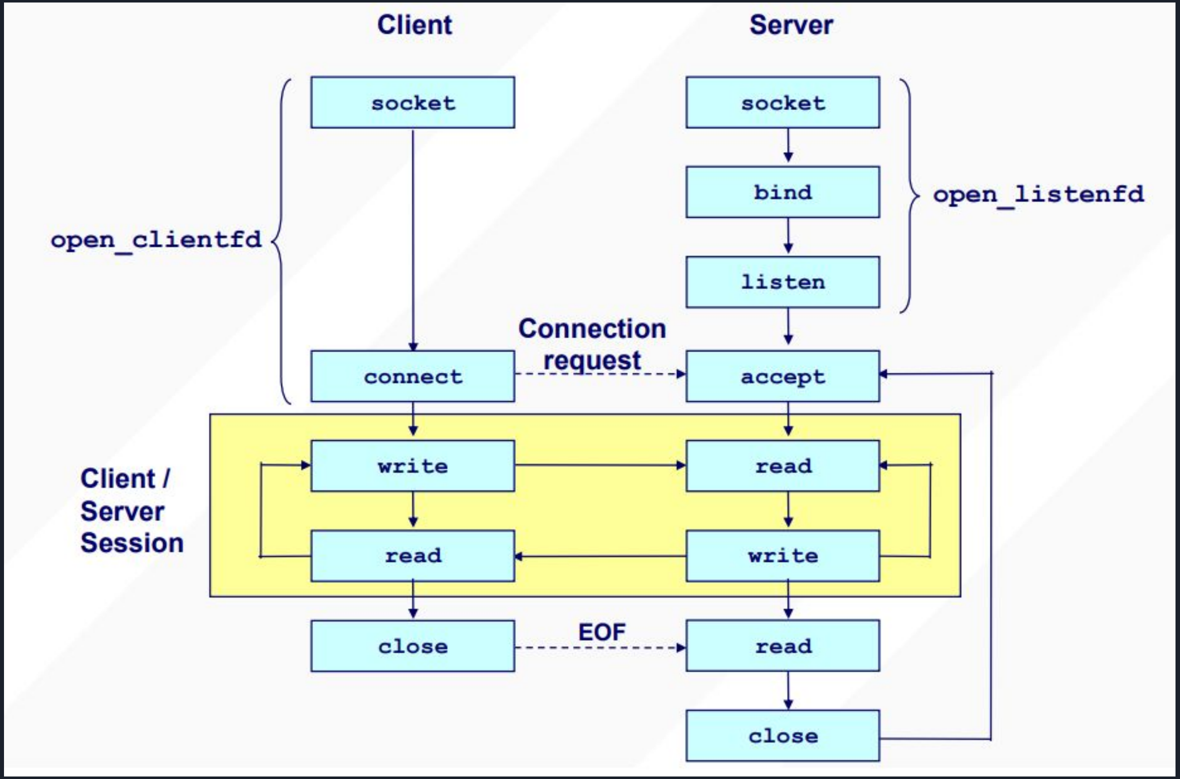


Image from CSE 124

Sockets Sample

- All the code for the loop above is within the example listed previously
- How the sample basically works:
  - You serialize data into one large array and send it over the network
  - The receiver parses it based off the packet headers you create
- Quick Networking Crash Course:
  - 127.0.0.1 is the localhost, use it to test on the same computer
  - To find your local IP, open up command prompt, and run `ipconfig`, and look for the interface that looks correct. Then look for your IPv4 address.
  - Make sure you use different ports, or the second person who tries to use it will get an error
  - Ports numbers <1024 are reserved

Network Architecture

- There is a lot of flexibility in how you can create your network architecture
- Important things to keep in mind:
  - The clients should run at 90 fps, but the server can run at 30 ticks per second
    - Keep your camera calculations on the client, only send over the scene graph
  - Having the server run at a constant rate might make some calculations easier.
  - Don’t try to do everything at once. First make sure you can get communication over the network, then integrate it with your code.
- I will go over a simple Server-Client architecture

Server-Client

- The server is a simulation of the virtual world
- The client is a window into the virtual world

Server

- Have the server run at a constant rate to simplify things. 30 tick might be a good starting point
- The server will maintain data structures for all the objects in the simulation
  - A scene graph might be good here
- The server will run a loop consisting of something similar to the following
  - Receive input from clients
  - Update the state of the simulation (collisions, physics, etc)
  - Send state to clients
  - Wait until next tick
- Having a constant tick rate simplifies how the server works and limits the work the server puts on the client

Client

- Client maintains local copy of state
- Client loop runs as fast as possible
  - Receive messages from server
  - Render world
  - Collect input events to send to server
- The client does not act on input directly, the server will do the update, and send the world state back to the client
- Having an authoritative server helps avoid conflicts

Events

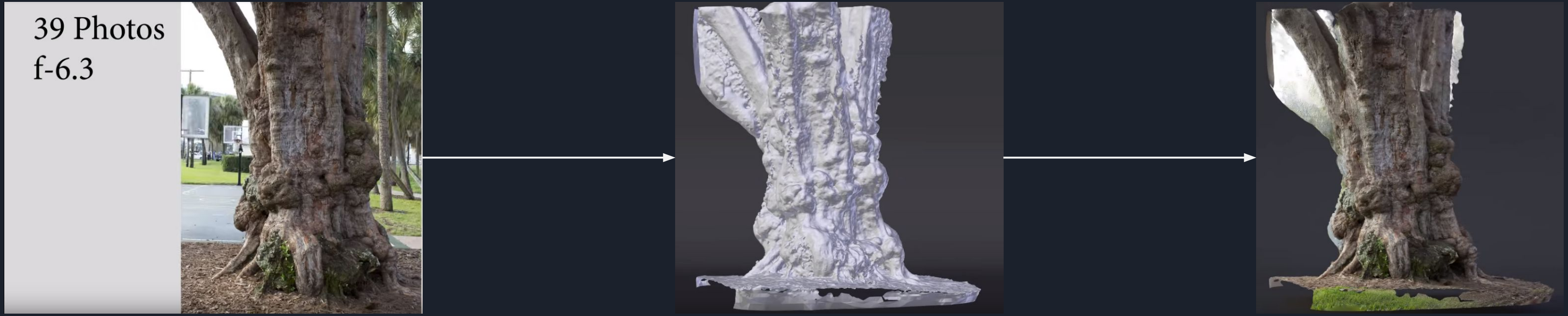
- The server needs to send object data to the clients
- Send events between the Server and Client.
  - Sending “fire” events from client to server
  - Sending “hit” events from server to client
- Adds in indirection between button presses and actions

Architecture Adjustments

- Since our case is very simple, with only two users, you can simplify a lot.
- You can have one of the computers run a combined server/client, and run the server on another thread, or only do the server update every so often.
- And if your game is simple enough, you could run the server along with the client in the same program.
- In the end, do whatever makes sense to you.

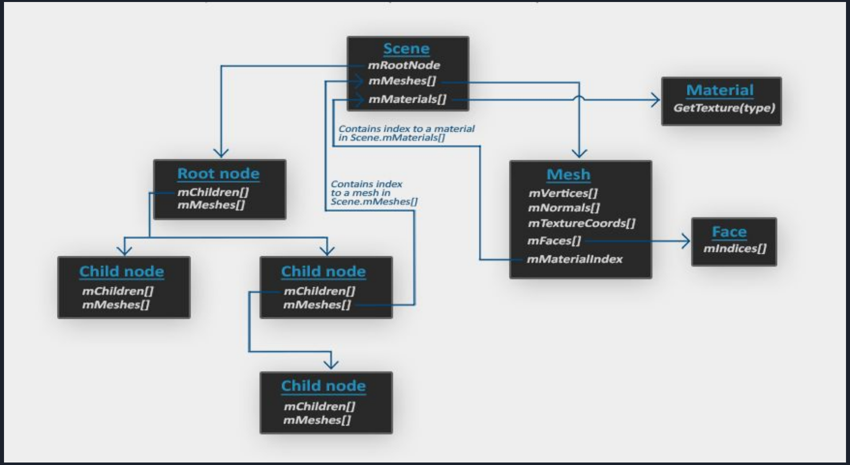
3D Model Customization

- In this assignment, you need to custom make at least one 3D object by yourself.
- You can scan your model from just pictures using tools like Agisoft Photoscan
  - There is also a 3D scanner in the VR lab that you can try out
  - You can read more about Agisoft Photoscan here
- You can create or pre-process your 3D meshes using tools like Blender or MeshLab
- Make sure your model is optimized enough
  - Because there is network communication involved, models that are too big will results to a framerate drop.
  - You can use MeshLab to reduce the polygon count of your model.



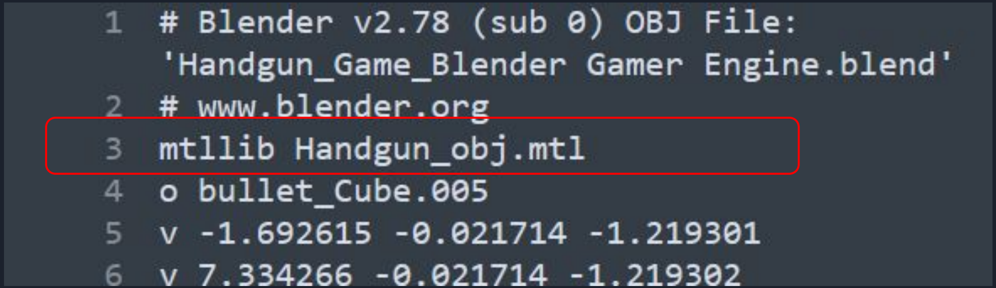
Model Loading

- When you are creating your model, make sure the model is exported to Wavefront .obj file or fbx file or files that you are familiar with.
- If you continue using Open Assimp Import Library, check out its supported file formats here.
    - Common file formats like fbx and obj are supported.
    - Regardless of the file format we imported, the data structure of Assimp stays the same.



MTL files

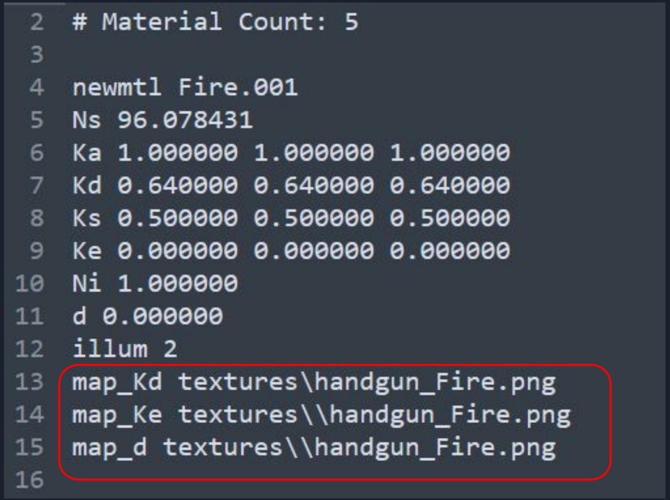
- There are cases that some generated model files come with other supplementary files in order to store more color, material, and texture information.
- .mtl (Material Library File) is one of them, and it is used to contain material definitions
- You can check if your obj file contains a mtl file definition by
    - Open the obj file as text file and check the headers to find the name of the mtl file, which often has the same name as the obj file.
    - And you need to place same in the same directory to load them.
- To access loaded materials in assimp:
    - `aiMaterial* material = scene->mMaterials[mesh->mMaterialIndex];`



```
# Blender v2.78 (sub 0) OBJ File: 'Handgun_Game_Blender Gamer Engine.blend'
# www.blender.org
mtllib Handgun_obj.mtl
o bullet_Cube.005
v -1.692615 -0.021714 -1.219301
v 7.334266 -0.021714 -1.219302
```

Texture files

- Some materials might specify some texture files that needs to be used for this material
- For example, in OBJ MTL files, some material can specify a path to the image files, and you need to make sure those images files are placed in the correct position as specified in the .mtl file.



```
# Material Count: 5

newmtl Fire.001
Ns 96.078431
Ka 1.000000 1.000000 1.000000
Kd 0.640000 0.640000 0.640000
Ks 0.500000 0.500000 0.500000
Ke 0.000000 0.000000 0.000000
Ni 1.000000
d 0.000000
illum 2
map_Kd textures\handgun_Fire.png
map_Ke textures\\handgun_Fire.png
map_d textures\\handgun_Fire.png

```

- To load these texture images, you can refer to the Model.cpp example in the tutorial.
- `vector<Texture> loadMaterialTextures()`
- `unsigned int TextureFromFile()`
- These two functions demo how to load the texture images.
- To find texture coordinates of the mesh:
    - `mesh->mTextureCoords[0]`

OpenAL: Open Audio Library

Slides link:

https://docs.google.com/presentation/d/1nha9ENV41FhftHJiDlru5kM8vvh3Wr-5MzHVSk6yfu4/edit?usp=sharing

Reference

- Client-Server network using C++ and Winsock:
  - https://www.codeproject.com/Articles/412511/Simple-client-server-network-using-Cplusplus-and-W
- MTL file format explained:

  http://paulbourke.net/dataformats/mtl/
- Agisoft Photoscan tutorial:
  - http://www.agisoft.com/pdf/PS_1.1%20-Tutorial%20(BL)%20-%203D-model.pdf
- Agisoft Photoscan Review article:
  - https://3dscanexpert.com/agisoft-photoscan-photogrammetry-3d-scanning-review/
- Assimp usage:
  - Library: https://github.com/assimp/assimp
  - Tutorial and sample codes: https://learnopengl.com/Model-Loading/Assimp

QUESTIONS?