CSE 190 Discussion 2

PA1: Where's Waldo 3D

Agenda

- Starter Code
- OpenGL Review
- Using Assimp
- Technical tips
- Extra Credit Hints

Starter Code

Starter Code

- Original Starter code used OGLplus
- Second version here
  - Uses OpenGL
  - Added:
    - Shader.h/cpp
    - Shader.vert/frag
    - cube.h/cpp

Suggested re-factoring:

- Recommend breaking up the original main.cpp file into individual files
  - Core.h
  - main.cpp
  - GlfwApp.h/cpp
  - ovr.h/cpp
  - RiftManagerApp.h/cpp
  - RiftApp.h/cpp
  - Scene.h/cpp
  - ExampleApp.h/cpp
- From learnopengl tutorials:
  - Model.h/cpp
  - Mesh.h/cpp

Core.h + main.cpp

- Core.h
  - Includes at top of original main.cpp
  - What lives here:
    - general library includes
    - global variables
- Main.cpp
  - 3 utility functions at the top of the current main.cpp
  - The main function at the bottom of the current main.cpp

```cpp
#include "core.h"
#include "Proj1.h"

bool checkFramebufferStatus(GLenum target = GL_FRAMEBUFFER) { ... }

bool checkGlError() { ... }

void glDebugCallbackHandler(GLenum source, GLenum type, GLuint id, GLen

//////////////////////////////////////////////////////////////////////
//
// Execute Proj1
//
//////////////////////////////////////////////////////////////////////

int main(int argc, const char* argv[]) {

    int result = -1;
    try {
        if (!OVR_SUCCESS(ovr_Initialize(nullptr))) {
            FAIL("Failed to initialize the Oculus SDK");
        }
        result = Proj1().run();
    }

    catch (std::exception & error) {
        OutputDebugStringA(error.what());
        std::cerr << error.what() << std::endl;
    }

    ovr_Shutdown();
    return result;
}
```

GlfwApp.h/cpp + ovr.h/cpp

- GlfwApp class
  - Don’t have to worry about or change this code at all
  - Deals with creating the windows
- Ovr namespace
  - Again no changes necessary to this code
  - Helper class to translate the glm matrices/vectors into ovr matrices/vector

RiftManagerApp.h/cpp + RiftApp.h/cpp

- RiftManagerApp class
  - Again no changes necessary to this code
  - Handles the initial set up of the app
- RiftApp class
  - Again no changes necessary to this code
  - Handles more specific details to the app
  - If want to add callbacks here can do so

Scene.h/cpp

- Scene.h/cpp
  - Uses OpenGL now
  - The main point of this is to create your sphere scene
  - Can keep most of it
    - Already has loops creating 5x5x5
    - Only need minimal changes to change to spheres of the appropriate spacing/scale

ExampleApp.h/cpp

- ExampleApp class
  - Here is where the bulk of your changes will happen
  - Right now all it does is:
    - set the background color
    - Initialize and render the cube scene
  - Main logic/functionality for the project lives here

OpenGL Review

OpenGL - Initialize Buffers

- VAO, VBO(s), EBO
  - VAO = container for buffers
  - VBO = hold model information
  - EBO = hold index information
- Generate the VAO and VBO(s)
- see
  - Cube() in cube.cpp
  - setupMesh() in Mesh.cpp

```
void setupMesh()
{
    // create buffers/arrays
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);
```

OpenGL - Sending/Filling Buffers

1. Bind VAO
2. For each VBO
   a. Bind buffer to VAO
   b. Send data
   c. Create channel for each type in VBO
   d. Tell how to read VBO
3. For EBO
   a. Bind buffer
   b. Send the data

See Cube() and setupMesh()

```
    glBindVertexArray(VAO);
    // load data into vertex buffers
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // A great thing about structs is that their memory layout is sequential for all its items.
    // The effect is that we can simply pass a pointer to the struct and it translates perfectly to a glm::vec3/2
    // again translates to 3/2 floats which translates to a byte array.
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW);

    // set the vertex attribute pointers
    // vertex Positions
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    // vertex normals
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
    // vertex texture coords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, TexCoords));
    // vertex tangent
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Tangent));
    // vertex bitangent
    glEnableVertexAttribArray(4);
    glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Bitangent));

    glBindVertexArray(0);
}
```

OpenGL - Sending/Filling Buffers

1. Bind VAO
2. For each VBO
   a. Bind buffer to VAO
   b. Send data
   c. Create channel for each type in VBO
   d. Tell how to read VBO
3. For EBO
   a. Bind buffer
   b. Send the data

See Cube() and setupMesh()

```
    glBindVertexArray(VAO);
    // load data into vertex buffers
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // A great thing about structs is that their memory layout is sequential for all its items.
    // The effect is that we can simply pass a pointer to the struct and it translates perfectly to a glm::vec3/2
    // again translates to 3/2 floats which translates to a byte array.
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW);

    // set the vertex attribute pointers
    // vertex Positions
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    // vertex normals
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
    // vertex texture coords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, TexCoords));
    // vertex tangent
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Tangent));
    // vertex bitangent
    glEnableVertexAttribArray(4);
    glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Bitangent));

    glBindVertexArray(0);
}
```

OpenGL - Sending/Filling Buffers

1. Bind VAO
2. For each VBO
   a. Bind buffer to VAO
   b. Send data
   c. Create channel for each type in VBO
   d. Tell how to read VBO
3. For EBO
   a. Bind buffer
   b. Send the data

See Cube() and setupMesh()

```
    glBindVertexArray(VAO);
    // load data into vertex buffers
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // A great thing about structs is that their memory layout is sequential for all its items.
    // The effect is that we can simply pass a pointer to the struct and it translates perfectly to a glm::vec3/
    // again translates to 3/2 floats which translates to a byte array.
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW);

    // set the vertex attribute pointers
    // vertex Positions
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    // vertex normals
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
    // vertex texture coords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, TexCoords));
    // vertex tangent
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Tangent));
    // vertex bitangent
    glEnableVertexAttribArray(4);
    glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Bitangent));

    glBindVertexArray(0);
}
```

OpenGL - Sending/Filling Buffers

1. Bind VAO
2. For each VBO
   a. Bind buffer to VAO
   b. Send data
   c. Create channel for each type in VBO
   d. Tell how to read VBO
3. For EBO
   a. Bind buffer
   b. Send the data

See Cube() and setupMesh()

```
    glBindVertexArray(VAO);
    // load data into vertex buffers
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // A great thing about structs is that their memory layout is sequential for all its items.
    // The effect is that we can simply pass a pointer to the struct and it translates perfectly to a glm::vec3/
    // again translates to 3/2 floats which translates to a byte array.
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW);

    // set the vertex attribute pointers
    // vertex Positions
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    // vertex normals
    glEnableVertexAttribArray(1);
    // vertex texture coords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, TexCoords));
    // vertex tangent
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Tangent));
    // vertex bitangent
    glEnableVertexAttribArray(4);
    glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Bitangent));

    glBindVertexArray(0);
}
```

OpenGL - Sending/Filling Buffers

1. Bind VAO
2. For each VBO
   a. Bind buffer to VAO
   b. Send data
   c. Create channel for each type in VBO
   d. Tell how to read VBO
3. For EBO
   a. Bind buffer
   b. Send the data

See Cube() and setupMesh()

```
    glBindVertexArray(VAO);
    // load data into vertex buffers
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // A great thing about structs is that their memory layout is sequential for all its items.
    // The effect is that we can simply pass a pointer to the struct and it translates perfectly to a glm::vec3/
    // again translates to 3/2 floats which translates to a byte array.
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW);

    // set the vertex attribute pointers
    // vertex Positions
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, TexCoords));
    // vertex tangent
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Tangent));
    // vertex bitangent
    glEnableVertexAttribArray(4);
    glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Bitangent));

    glBindVertexArray(0);
}
```

OpenGL - Sending/Filling Buffers

1. Bind VAO
2. For each VBO
   a. Bind buffer to VAO
   b. Send data
   c. Create channel for each type in VBO
   d. Tell how to read VBO
3. For EBO
   a. Bind buffer
   b. Send the data

See Cube() and setupMesh()

```
    glBindVertexArray(VAO);
    // load data into vertex buffers
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // A great thing about structs is that their memory layout is sequential for all its items.
    // The effect is that we can simply pass a pointer to the struct and it translates perfectly to a glm::vec3/
    // again translates to 3/2 floats which translates to a byte array.
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW)

    // set the vertex attribute pointers
    // vertex Positions
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    // vertex normals
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
    // vertex texture coords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, TexCoords));
    // vertex tangent
    glEnableVertexAttribArray(3);
    glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Tangent));
    // vertex bitangent
    glEnableVertexAttribArray(4);
    glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Bitangent));

    glBindVertexArray(0);
}
```

Shader Quick Review

- Vertex Shader
  - Part of the early steps in the graphic pipeline.
  - Nothing is yet rendered at this stage
  - gl_position is the mandatory output variable
- Fragment Shader
  - Same as the pixel shader
  - Part of the rasterization step
  - pixels between vertices are colored and lights are applied

Shader Quick Review

- Need the reference to the shader program
  - Returned from provided shader.h

```
#define DEFAULT_VERTEXT_SHADER_PROGRAM "assets/shaders/shader.vert"
#define DEFAULT_FRAGMENT_SHADER_PROGRAM "assets/shaders/shader.frag"
GLuint defaultShaderProgram = LoadShaders(DEFAULT_VERTEXT_SHADER_PROGRAM, DEFAULT_FRAGMENT_SHADER_PROGRAM);
```

- Pass variables to shader program
  - Get the reference to shaderProgram and your variable in shader
    - GLint location = glGetUniformLocation(shaderProgram, "#yourVariableName")
  - Parse MVP matrices:
    - void **glUniformMatrix4fv**(GLint location, GLsizei count, GLboolean transpose, const GLfloat *value);
  - Parse numerical value:
    - void **glUniform1f** (GLint location, GLfloat v0);
    - void **glUniform1i** (GLint location, GLint v0);
  - More data types: See the link to glUniform

OpenGL - Draw

- Indicate which shader to use
  - Need to do this before sending uniforms
- Send variables/uniforms to shader
  - MV, P matrices …
- Draw the object
  - If using mesh.h from learnopengl Call Draw on the mesh object:
    - mesh.draw(shader)

```
void Cube::draw(GLuint shaderProgram, const glm::mat4& projection, const glm::mat4& view) {

    glUseProgram(shaderProgram);

    // Calculate the combination of the model and view (camera inverse) matrices
    glm::mat4 modelview = view * toWorld;
    // We need to calcullate this because modern OpenGL does not keep track of any matrix oth
    // Consequently, we need to forward the projection, view, and model matrices to the shade
    // Get the location of the uniform variables "projection" and "modelview"
    uProjection = glGetUniformLocation(shaderProgram, "projection");
    uModelview = glGetUniformLocation(shaderProgram, "modelview");

    // Now send these values to the shader program
    glUniformMatrix4fv(uProjection, 1, GL_FALSE, &projection[0][0]);
    glUniformMatrix4fv(uModelview, 1, GL_FALSE, &modelview[0][0]);

    // Now draw the cube. We simply need to bind the VAO associated with it.
    glBindVertexArray(VAO);
    // Tell OpenGL to draw with triangles
    glDrawArrays(GL_TRIANGLES, 0, 3 * 2 * 6); // 3 vertices per triangle, 2 triangles per fac
                                              // Unbind the VAO when we're done so we don't a
    glBindVertexArray(0);
}
```

OpenGL - Draw

- Indicate which shader to use
  - Need to do this before sending uniforms
- Send variables/uniforms to shader
  - MV, P matrices …
- Draw the object
  - If using mesh.h from learnopengl Call Draw on the mesh object:
    - mesh.draw(shader)

```
void Cube::draw(GLuint shaderProgram, const glm::mat4& projection, const glm::mat4& view) {

    glUseProgram(shaderProgram);

    // Calculate the combination of the model and view (camera inverse) matrices
    glm::mat4 modelview = view * toWorld;
    // We need to calcullate this because modern OpenGL does not keep track of any matrix oth
    // Consequently, we need to forward the projection, view, and model matrices to the shade
    // Get the location of the uniform variables "projection" and "modelview"
    uProjection = glGetUniformLocation(shaderProgram, "projection");
    uModelview = glGetUniformLocation(shaderProgram, "modelview");

    // Now send these values to the shader program
    glUniformMatrix4fv(uProjection, 1, GL_FALSE, &projection[0][0]);
    glUniformMatrix4fv(uModelview, 1, GL_FALSE, &modelview[0][0]);

    // Now draw the cube. We simply need to bind the VAO associated with it.
    glBindVertexArray(VAO);
    // Tell OpenGL to draw with triangles
    glDrawArrays(GL_TRIANGLES, 0, 3 * 2 * 6); // 3 vertices per triangle, 2 triangles per fac
                                              // Unbind the VAO when we're done so we don't a
    glBindVertexArray(0);
}
```

OpenGL - Draw

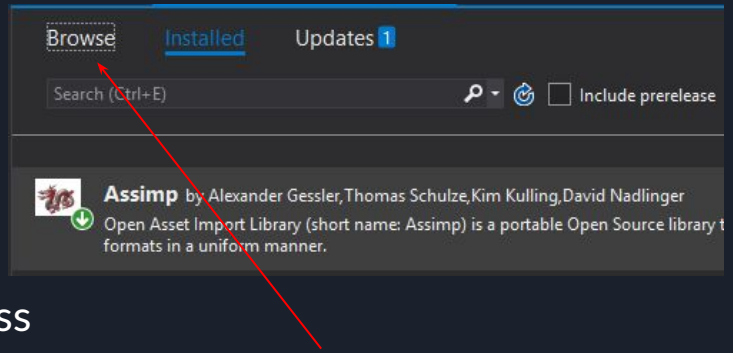
- Indicate which shader to use
  - Need to do this before sending uniforms
- Send variables/uniforms to shader
  - MV, P matrices …
- Draw the object
  - If using mesh.h from learnopengl Call Draw on the mesh object:
    - mesh.draw(shader)

```
void Cube::draw(GLuint shaderProgram, const glm::mat4& projection, const glm::mat4& view) {

    glUseProgram(shaderProgram);

    // Calculate the combination of the model and view (camera inverse) matrices
    glm::mat4 modelview = view * toWorld;
    // We need to calcullate this because modern OpenGL does not keep track of any matrix oth
    // Consequently, we need to forward the projection, view, and model matrices to the shade
    // Get the location of the uniform variables "projection" and "modelview"
    uProjection = glGetUniformLocation(shaderProgram, "projection");
    uModelview = glGetUniformLocation(shaderProgram, "modelview");

    // Now send these values to the shader program
    glUniformMatrix4fv(uProjection, 1, GL_FALSE, &projection[0][0]);
    glUniformMatrix4fv(uModelview, 1, GL_FALSE, &modelview[0][0]);

    // Now draw the cube. We simply need to bind the VAO associated with it.
    glBindVertexArray(VAO);
    // Tell OpenGL to draw with triangles
    glDrawArrays(GL_TRIANGLES, 0, 3 * 2 * 6); // 3 vertices per triangle, 2 triangles per fac
                                              // Unbind the VAO when we're done so we don't a

    glBindVertexArray(0);
}
```

Using Assimp

Loading Model Using Assimp

- Add assimp packages to your project:
  - See last discussion slides for details
  - Simply go to Nuget and search for assimp. Then install and it solves everything for you.
- Create your own Model and Mesh files
- Include assimp in Model class:
  - #include <assimp/Importer.hpp>
  - #include <assimp/scene.h>
  - #include <assimp/postprocess.h>
- Mesh class should be similar to Cube class



Loading Model Using learnopengl files

- Include:
  - Model.h
  - Mesh.h
- Model.h
  - Model constructor loades the model
    - Call constructor inside CubeScene
  - processMesh()
    - It extracts the vertices and normals from the obj file
    - You might need to modify this method so that you know where to store these information and where to render them

Drawing objs Using learnopengl files

- To draw:
  - Model’s draw function takes Shader variable then uses its member variable ID in the draw function
  - The given shader.h file returns this ID when create the shader, so you will need to make some changes to Mesh.h and Model.h match up

Technical Tips

- Access the controller information
- Summon console
- Count time and random number
- Run code without headset

Access Controller position/rotation

- controller information: hmdState.handPoses[ovrHand_Right]
- Remember namespace ove has helper functions to convert between ovr and glm

```
// get the general state hmdState
double ftiming = ovr_GetPredictedDisplayTime(_session, 0);
ovrTrackingState hmdState = ovr_GetTrackingState( _session,
                                                  ftiming, ovrTrue);

// Get the state of hand poses
ovrPoseStatef handPoseState = hmdState.HandPoses[ovrHand_Right];

//Get the hand pose position.
glm::vec3 controllerPosition = ovr::toGlm(
                               handPoseState.ThePose.Position);

// Get hand rotation
glm::quat controllerRotation = ovr::toGlm( ovrQuatf(
                               handPoseState.ThePose.Orientation) );
```

Access Controller buttons

```
typedef enum ovrButton_
{
    ovrButton_A          = 0x00000001, /// A button
    ovrButton_B          = 0x00000002, /// B button
```

- Check ovr_Buttons in OVR_CAPI.h
- Understand ovr_inputState
- Example:

```
// Get Current Input State
ovrInputState inputState;
ovr_GetInputState(session, ovrControllerType_Touch, &inputState);
if (inputState.Buttons & ovrButton_A){
        Printf("A is pressed")
}
```

- Fun fact:
  - Note the bitwise operator &, it works here because ovrButton uses one-hot encoding so that & will be evaluated to 0 as long as values on the two sides are different

Summon Consoles

- Consoles can be initialized with this codes:

```
AllocConsole();
freopen("conin$", "r", stdin);        // give access to reading
freopen("conout$", "w", stdout);           // give access to writing to
stdout
freopen("conout$", "w", stderr);      // give access to writing to stderr
printf("Debugging Window:\n");        // print a sample message
```

- Try figuring out where to put those codes

Count time using OculusSDK

- Having access to the time is important
  - You need to count a time of one minute in your game
  - It is also needed if you want to implement ovr_avatar that displays player's hand models
- Get the current time
  - You can use the time offered by time.h
  - However, oculusSDK has a handy way for it
    - double currentTime = ovr_GetTimeInSeconds();
  - Simulate a deltaTime using deltaTime = newTime - oldTime
  - Use delta time to compute the time passed.
- A note about generating random number:
  - int RandomNum = rand()%[range of random number] + [smallest number in range]

Run code without the headset

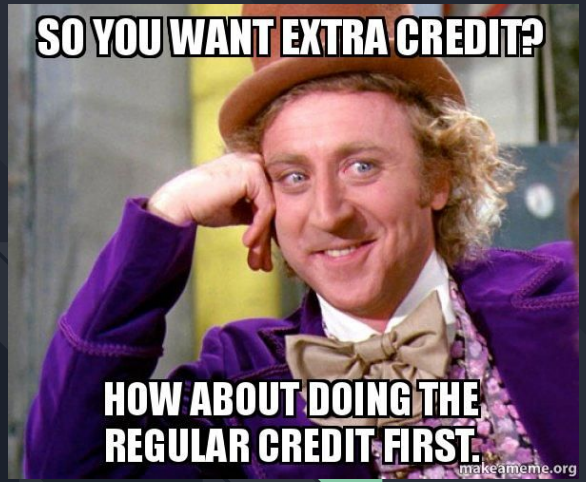
- Note: It may not work after Oculus App 1.14.x (still confirming with Oculus support)
- Since Oculus does not allow downloading/running with the lower-version runtime other than up-to-date version, running code without the headset may not be a feasible way to everyone.

> 3.2 Oculus Runtime Software. Subject to these Terms, including the license provided in Section 3.1, you may access, install, and use the Oculus Runtime Software ("Runtime"). In order to maximize your enjoyment, safety, and overall experience through our Services, the Runtime may only be used with Oculus approved hardware devices and with software developed using the Oculus Rift Software Development Kit, as specified in the Oculus Rift Software Development Kit license agreement. We also require that you use only the then-current version of the Runtime. You acknowledge that the Runtime incorporates proprietary information, and that you will not disclose it to any other person or entity.

- There is an unofficial way to downgrade to Oculus 1.3 (in reddit), but try it with your own risk in your own pc if that's really you want, and it takes time. (Note: VR Lab PC is not capable to do that since school restrict the Administrator authority on it)
- After you installed Oculus 1.3, refer to this thread to initialize the OclulusDebugTool.exe

Extra Credit

- It is for bonus.
- Do the regular part first.
- We tend not to provide detailed help for it

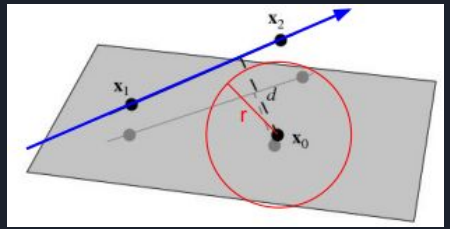


Extra Credit Tips
Oculus Avatar

- Download Avatar SDK:
  - Avatar SDK
- Official documentation on how to use Avatar SDK:
  - Getting Started with Avatar SDK
- Simple online samples on how to use the Avatar SDK in the link above
- How do I import?
  - Refer to discussion 1 slides about how to add packages and resolve linker dependencies.

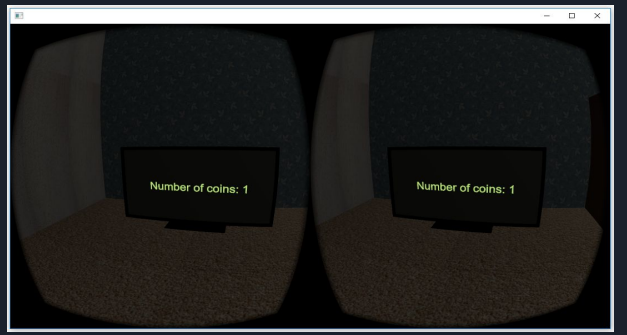
Extra Credit Tips
New Game Play

- Be creative!
  - New ways of selection/manipulation?
  - Make the environment more interesting?
  - Make the set of spheres dynamic?
    - A dodgeball game ?
  - How would you implement a laser pointer?
    - Point-line distance reference



Extra Credit Tips
Display Text in the VR headset

- Text Rendering
- Text rendering with OpenGL and C++
- Drawing Text in a Double-Buffered OpenGL Window
- Rendering things to the Oculus Rift



Extra Credit Tips
Wrap Waldo as texture for the sphere

Waldo's Picture = Sphere's texture

- Some helpful resources:
  - Texturing a Sphere
  - How to wrap an image around a sphere in opengl?
  - GLSL Programming/GLUT/Textured Spheres

QUESTIONS?