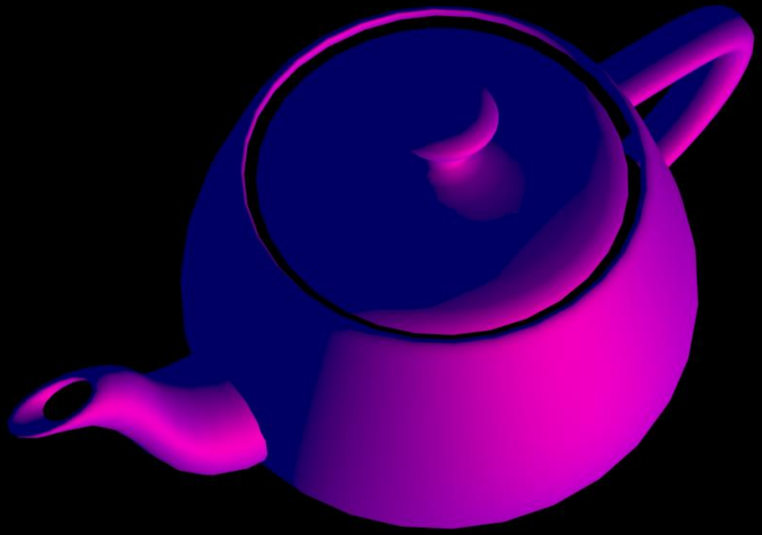


The making of Teapong

- Goals
- Timeline
- External Libraries
- Bird's-eye View
- Modern C++ Features
- Key Insights
- Design Patterns
- Things I'm Not Proud of
- The Subtlest Bug

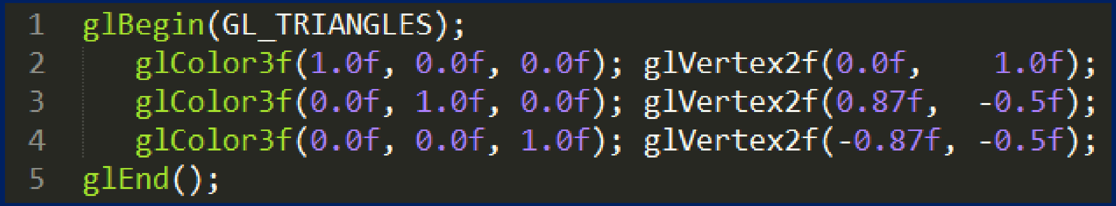
Goals

- Explore fundamental concepts of computer graphics like...
  - Transformations, projections and coordinate systems
  - Texturing and shading
  - The rendering pipeline
- Using...
  - Modern OpenGL
  - Modern C++
- And have the result be...
  - Clean, organized and efficient
  - Fully cross-platform (Windows, macOS and Linux)
  - A funny combination of the "Hello, World!" equivalents of computer graphics and game development!

Goals

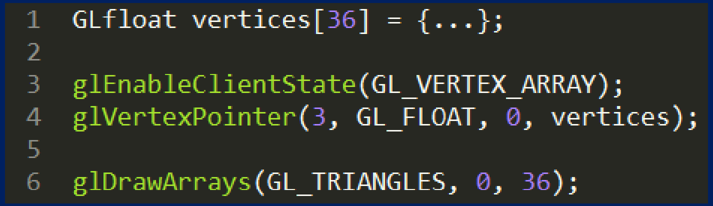
What do I mean with "Modern" OpenGL?

- Immediate mode:



```
glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f); glVertex2f(0.0f,    1.0f);
    glColor3f(0.0f, 1.0f, 0.0f); glVertex2f(0.87f,  -0.5f);
    glColor3f(0.0f, 0.0f, 1.0f); glVertex2f(-0.87f, -0.5f);
glEnd();
```

  - The driver cannot transfer data or tell the GPU to start rendering before glEnd is called.



```
GLfloat vertices[36] = {...};

glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);

glDrawArrays(GL_TRIANGLES, 0, 36);
```

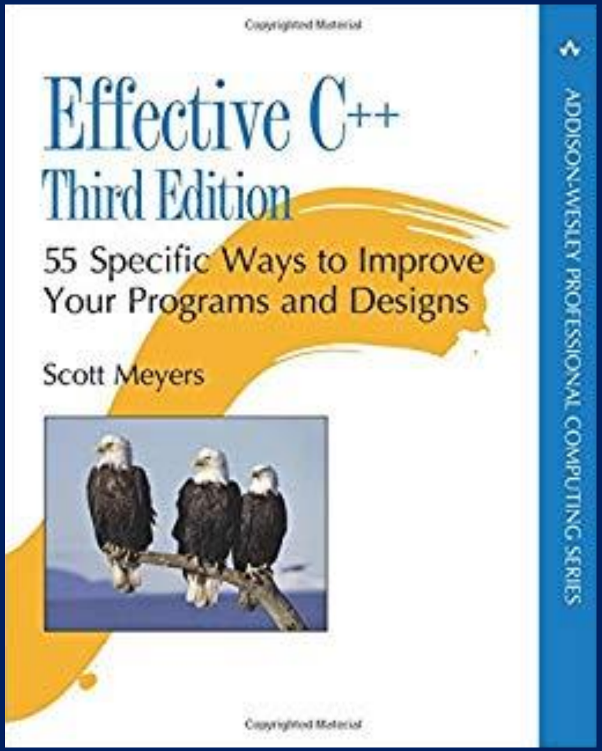
  - The driver can only transfer a copy of your array when glDrawArrays is called, and it must block your application while doing so.

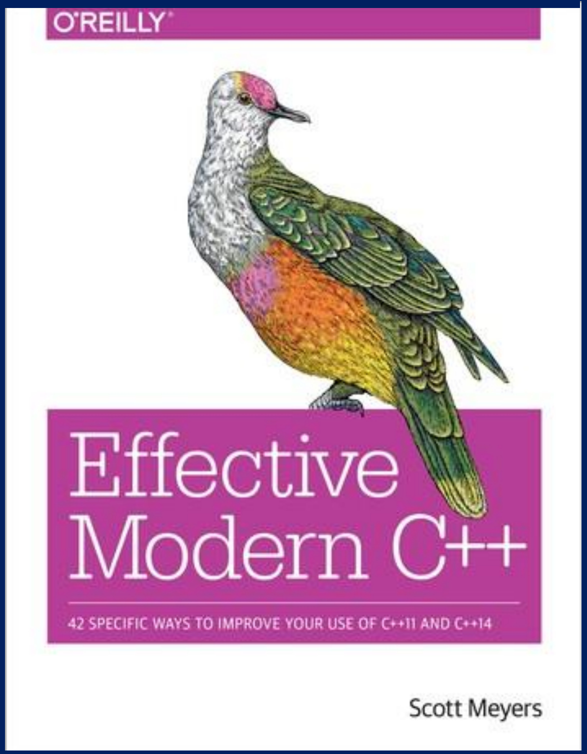
- Retained mode:
  - You fill a buffer with data a give it to OpenGL.
  - Your application no longer owns that buffer, so it cannot modify it.
  - Because of this, the driver can transfer the data in that buffer whenever the bus is free.
  - Any calls to glDrawArrays go into a work queue and return immediately, before actually finishing.
  - Your application and the GPU run asynchronously!

Goals

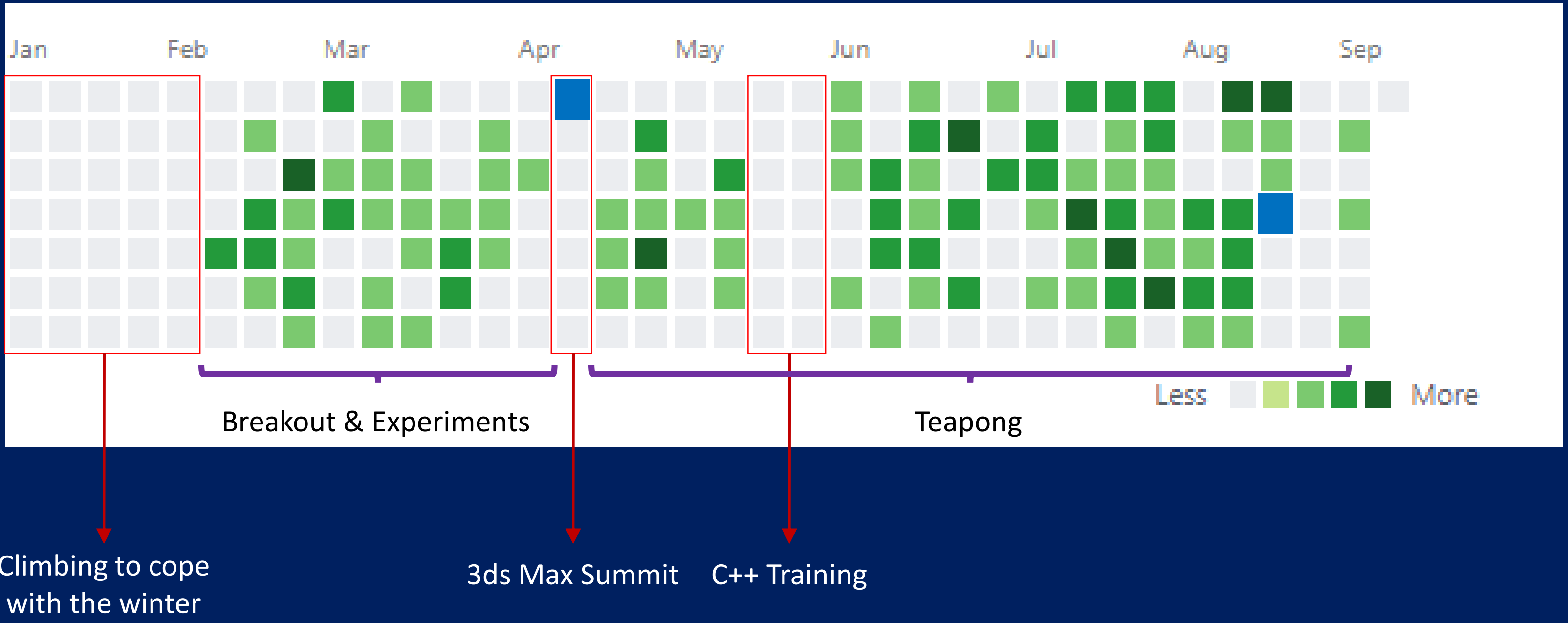
What do I mean with "Modern" C++?

- Incorporate as many ideas as possible from Scott Meyers' "Effective C++" books


Copyrighted Material
Effective C++
Third Edition
55 Specific Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES
Copyrighted Material


O'REILLY
Effective Modern C++
42 SPECIFIC WAYS TO IMPROVE YOUR USE OF C++11 AND C++14
Scott Meyers

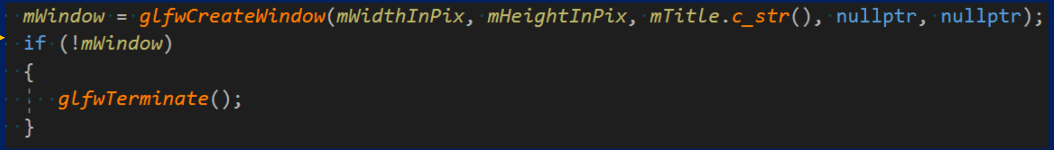
Timeline



- 4 months and 7 days from first push to demo
- 68 days in which code was pushed
- 3200 lines of code and 50 different files

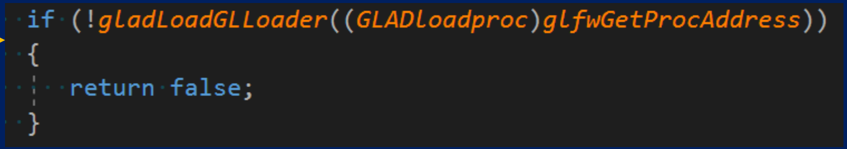
External Libraries

- **GLFW**
  - Create an OpenGL context
  - Create a window
  - Receive input from the keyboard and mouse



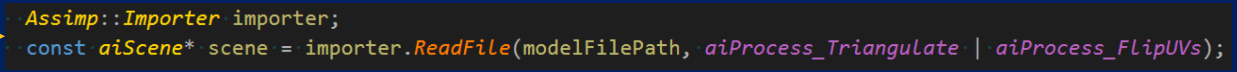
```
mWindow = glfwCreateWindow(mWidthInPix, mHeightInPix, mTitle.c_str(), nullptr, nullptr);
if (!mWindow)
{
    glfwTerminate();
}
```

- **GLAD**
  - Load pointers to OpenGL functions



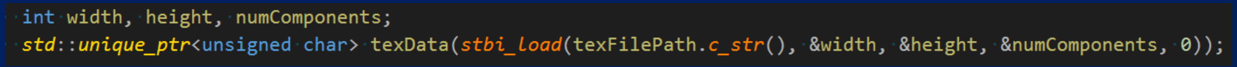
```
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    return false;
}
```

- **OpenGL Mathematics (GLM)**
- **Open Asset Import Library (Assimp)**
  - Import 3D models



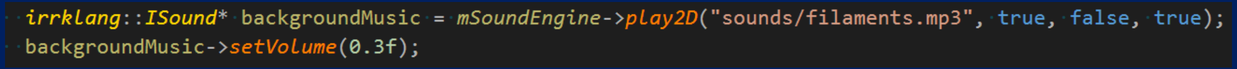
```
Assimp::Importer importer;
const aiScene* scene = importer.ReadFile(modelFilePath, aiProcess_Triangulate | aiProcess_FlipUVs);
```

- **stb_image**
  - Load textures



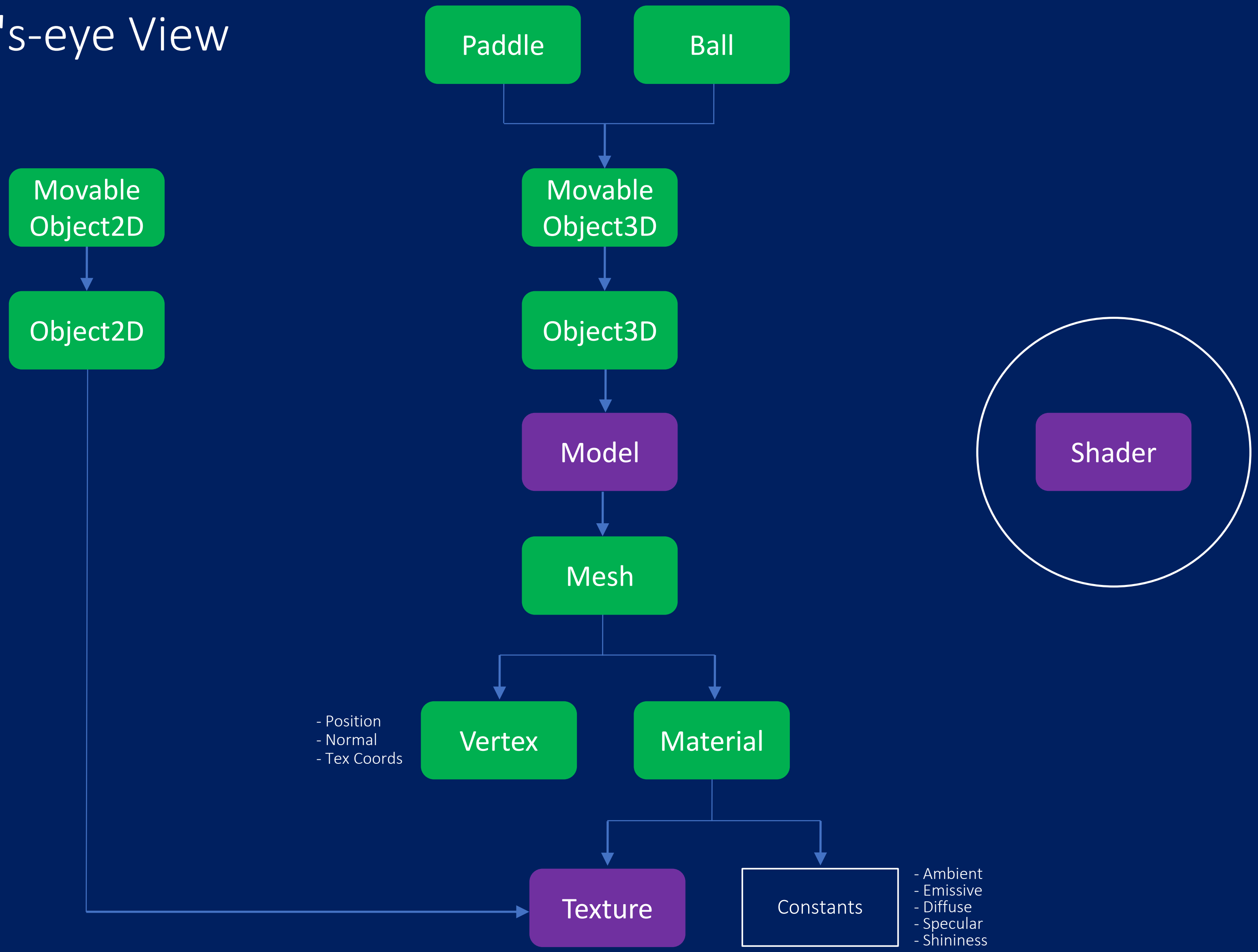
```
int width, height, numComponents;
std::unique_ptr<unsigned char> texData(stbi_load(texFilePath.c_str(), &width, &height, &numComponents, 0));
```

- **irrKlang**
  - Play music and sound effects

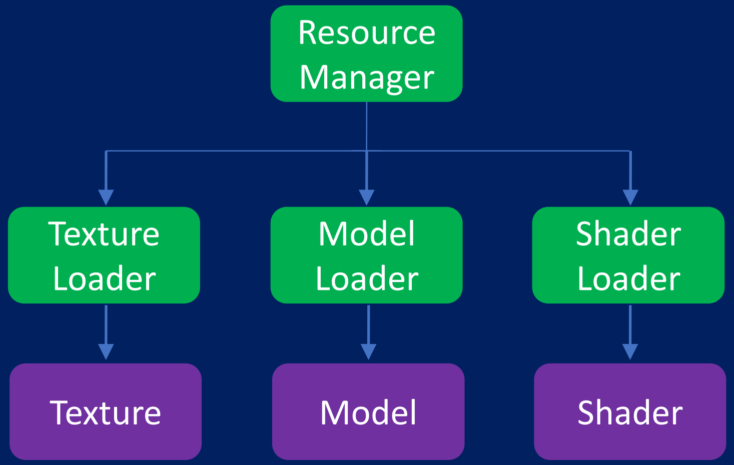


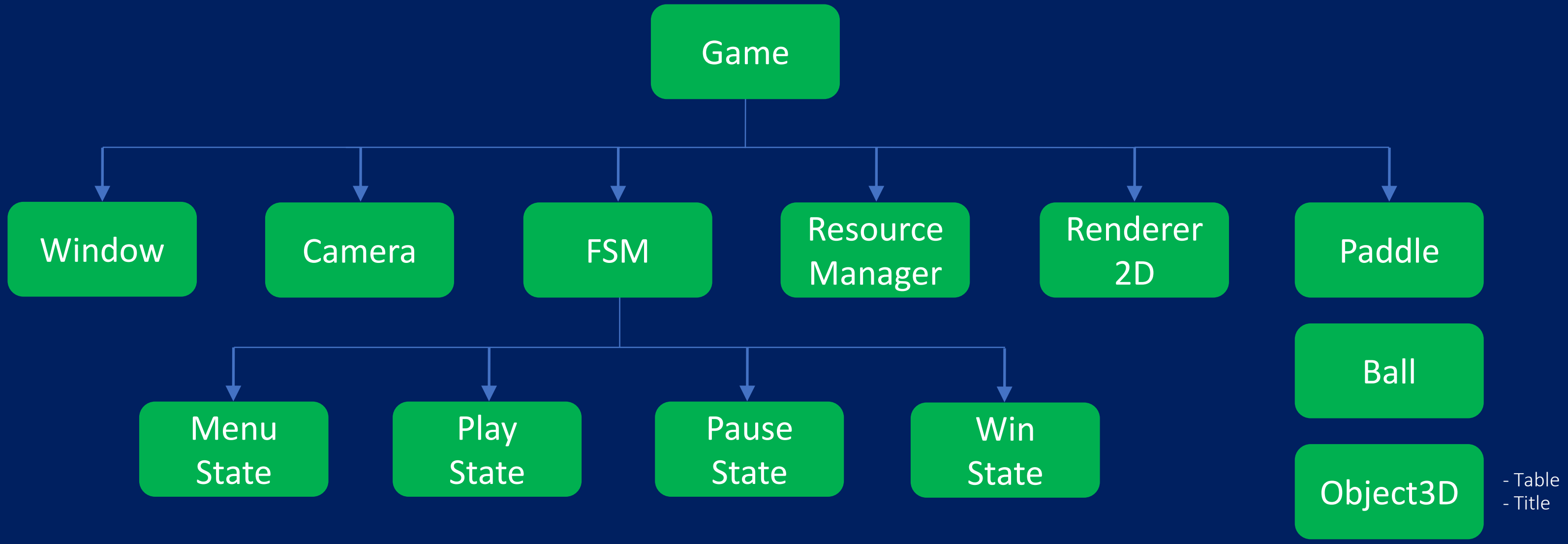
```
irrklang::ISound* backgroundMusic = mSoundEngine->play2D("sounds/filaments.mp3", true, false, true);
backgroundMusic->setVolume(0.3f);
```

Bird's-eye View



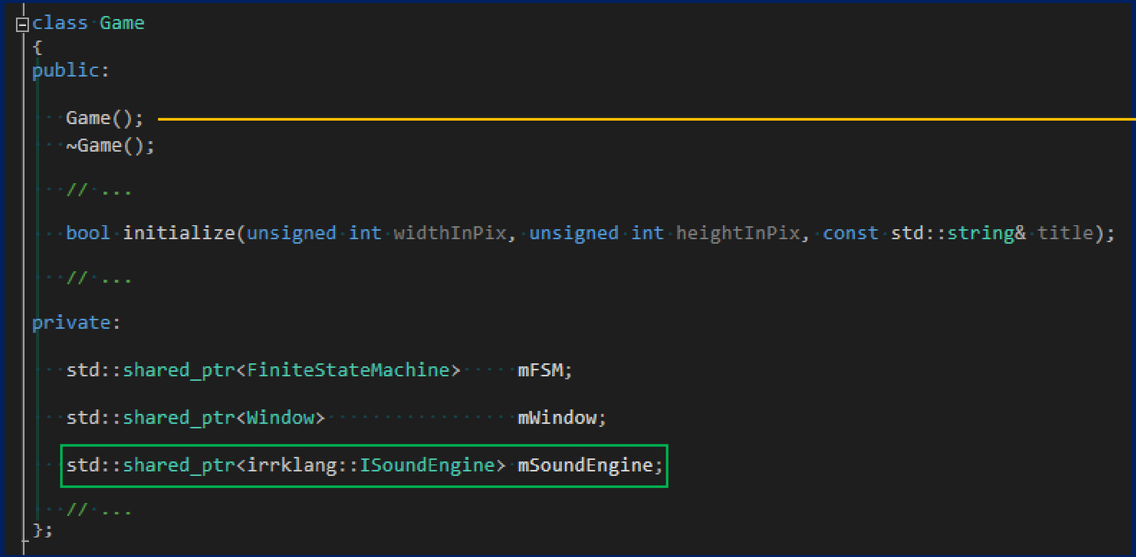
Bird's-eye View



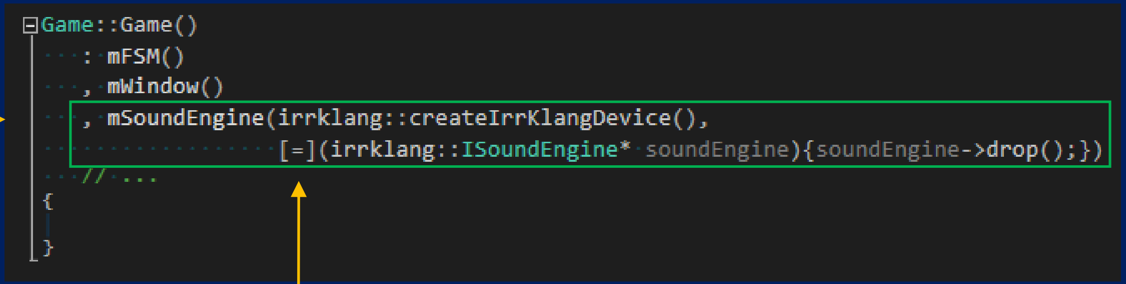


Modern C++ Features

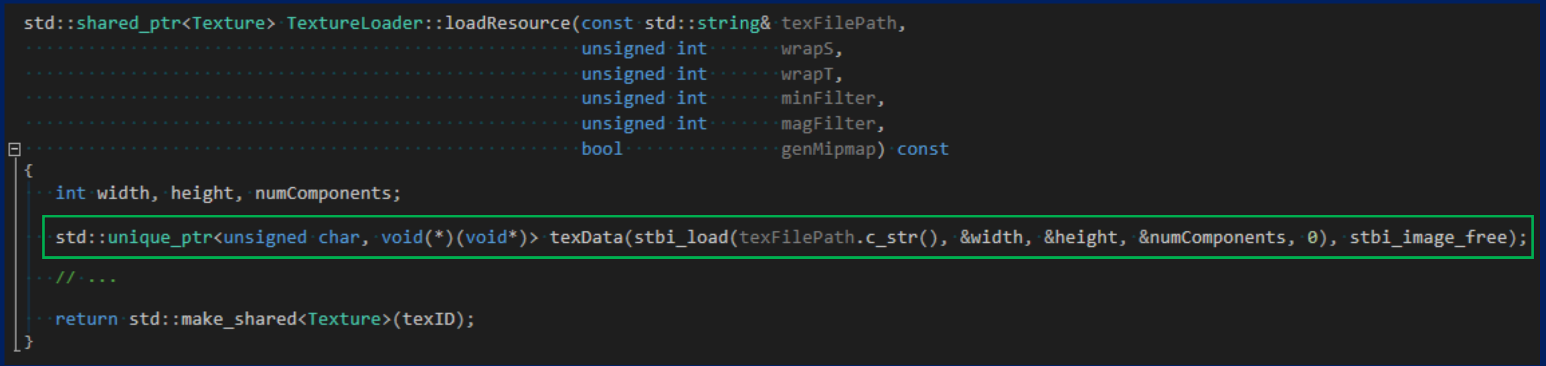
1) Smart pointers (with custom deleters!)



```cpp
class Game
{
public:

    Game();
    ~Game();

    // ...

    bool initialize(unsigned int widthInPix, unsigned int heightInPix, const std::string& title);

    // ...

private:

    std::shared_ptr<FiniteStateMachine>     mFSM;

    std::shared_ptr<Window>                 mWindow;

    std::shared_ptr<irrklang::ISoundEngine> mSoundEngine;

    // ...
};
```



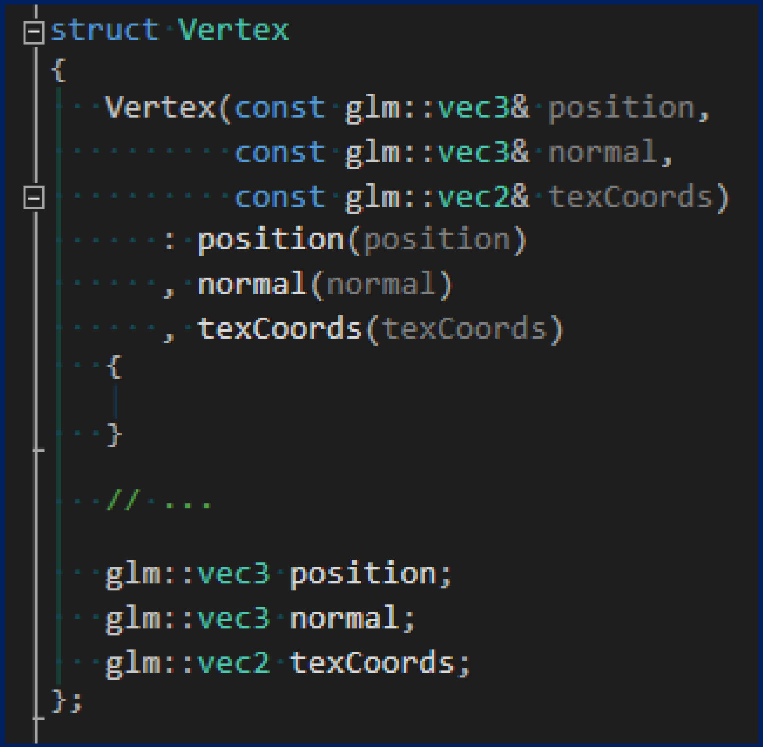
```cpp
Game::Game()
    : mFSM()
    , mWindow()
    , mSoundEngine(irrklang::createIrrKlangDevice(),
                   [=](irrklang::ISoundEngine* soundEngine){soundEngine->drop();})
    // ...
{

}
```



```cpp
std::shared_ptr<Texture> TextureLoader::loadResource(const std::string& texFilePath,
                                                     unsigned int       wrapS,
                                                     unsigned int       wrapT,
                                                     unsigned int       minFilter,
                                                     unsigned int       magFilter,
                                                     bool               genMipmap) const
{
    int width, height, numComponents;

    std::unique_ptr<unsigned char, void(*)(void*)> texData(stbi_load(texFilePath.c_str(), &width, &height, &numComponents, 0), stbi_image_free);

    // ...

    return std::make_shared<Texture>(texID);
}
```

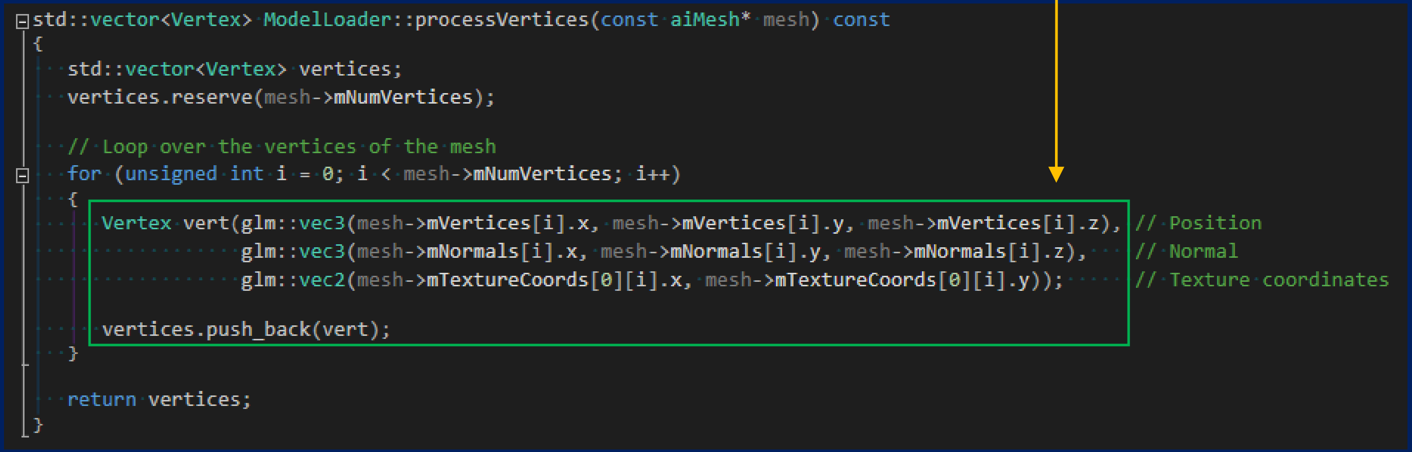
Modern C++ Features

2) Emplacement

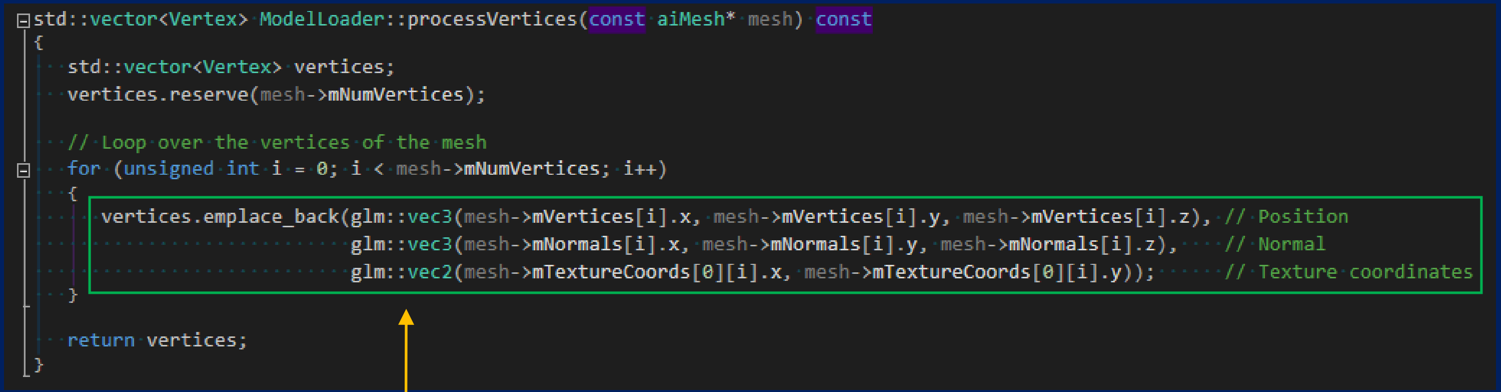


```cpp
struct Vertex
{
    Vertex(const glm::vec3& position,
           const glm::vec3& normal,
           const glm::vec2& texCoords)
        : position(position)
        , normal(normal)
        , texCoords(texCoords)
    {

    }

    // ...

    glm::vec3 position;
    glm::vec3 normal;
    glm::vec2 texCoords;
};
```

Compiler could take advantage of move semantics, but that is not guaranteed



```cpp
std::vector<Vertex> ModelLoader::processVertices(const aiMesh* mesh) const
{
    std::vector<Vertex> vertices;
    vertices.reserve(mesh->mNumVertices);

    // Loop over the vertices of the mesh
    for (unsigned int i = 0; i < mesh->mNumVertices; i++)
    {
        Vertex vert(glm::vec3(mesh->mVertices[i].x, mesh->mVertices[i].y, mesh->mVertices[i].z), // Position
                    glm::vec3(mesh->mNormals[i].x, mesh->mNormals[i].y, mesh->mNormals[i].z),    // Normal
                    glm::vec2(mesh->mTextureCoords[0][i].x, mesh->mTextureCoords[0][i].y));     // Texture coordinates

        vertices.push_back(vert);
    }

    return vertices;
}
```

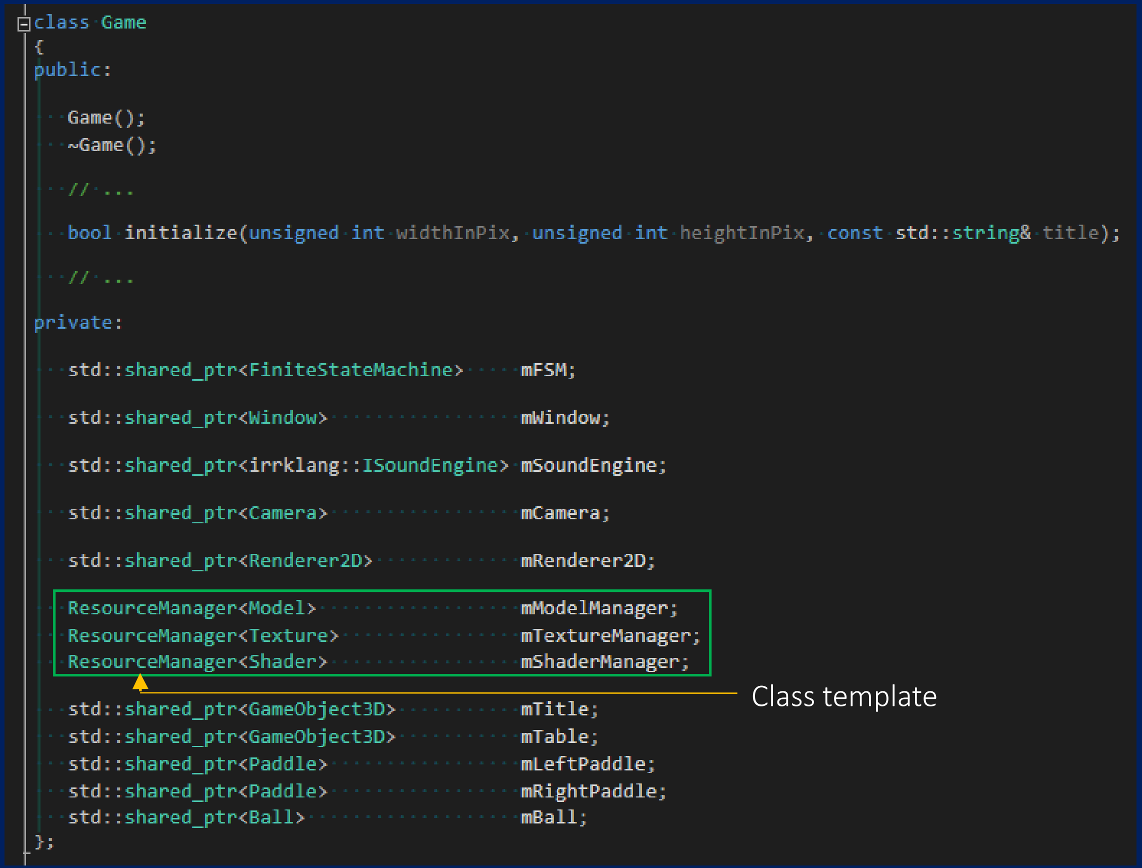


```cpp
std::vector<Vertex> ModelLoader::processVertices(const aiMesh* mesh) const
{
    std::vector<Vertex> vertices;
    vertices.reserve(mesh->mNumVertices);

    // Loop over the vertices of the mesh
    for (unsigned int i = 0; i < mesh->mNumVertices; i++)
    {
        vertices.emplace_back(glm::vec3(mesh->mVertices[i].x, mesh->mVertices[i].y, mesh->mVertices[i].z), // Position
                              glm::vec3(mesh->mNormals[i].x, mesh->mNormals[i].y, mesh->mNormals[i].z),    // Normal
                              glm::vec2(mesh->mTextureCoords[0][i].x, mesh->mTextureCoords[0][i].y));     // Texture coordinates
    }

    return vertices;
}
```

Perfect forward

Modern C++ Features

3) Variadic templates and perfect forwarding



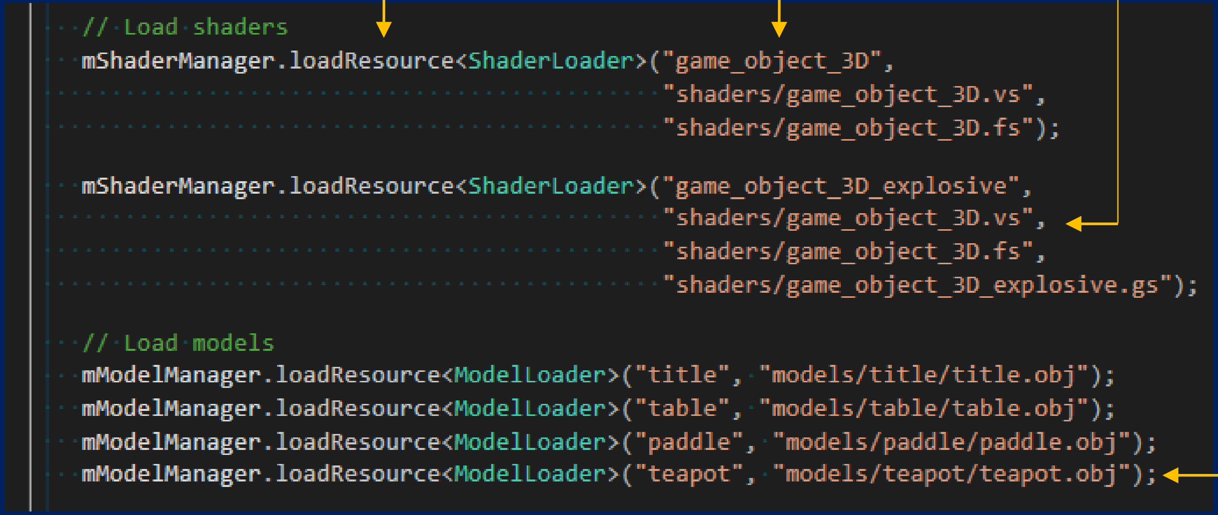
```cpp
class Game
{
public:

    Game();
    ~Game();

    // ...

    bool initialize(unsigned int widthInPix, unsigned int heightInPix, const std::string& title);

    // ...

private:

    std::shared_ptr<FiniteStateMachine>     mFSM;

    std::shared_ptr<Window>                 mWindow;

    std::shared_ptr<irrklang::ISoundEngine> mSoundEngine;

    std::shared_ptr<Camera>                 mCamera;

    std::shared_ptr<Renderer2D>             mRenderer2D;

    ResourceManager<Model>                  mModelManager;
    ResourceManager<Texture>                mTextureManager;
    ResourceManager<Shader>                 mShaderManager;

    std::shared_ptr<GameObject3D>           mTitle;
    std::shared_ptr<GameObject3D>           mTable;
    std::shared_ptr<Paddle>                 mLeftPaddle;
    std::shared_ptr<Paddle>                 mRightPaddle;
    std::shared_ptr<Ball>                   mBall;
};
```

Class template

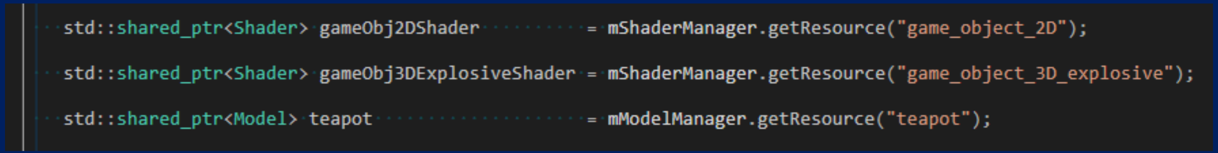
Function template

3 arguments!

4 arguments!



```cpp
// Load shaders
mShaderManager.loadResource<ShaderLoader>("game_object_3D",
                                          "shaders/game_object_3D.vs",
                                          "shaders/game_object_3D.fs");

mShaderManager.loadResource<ShaderLoader>("game_object_3D_explosive",
                                          "shaders/game_object_3D.vs",
                                          "shaders/game_object_3D.fs",
                                          "shaders/game_object_3D_explosive.gs");

// Load models
mModelManager.loadResource<ModelLoader>("title", "models/title/title.obj");
mModelManager.loadResource<ModelLoader>("table", "models/table/table.obj");
mModelManager.loadResource<ModelLoader>("paddle", "models/paddle/paddle.obj");
mModelManager.loadResource<ModelLoader>("teapot", "models/teapot/teapot.obj");
```

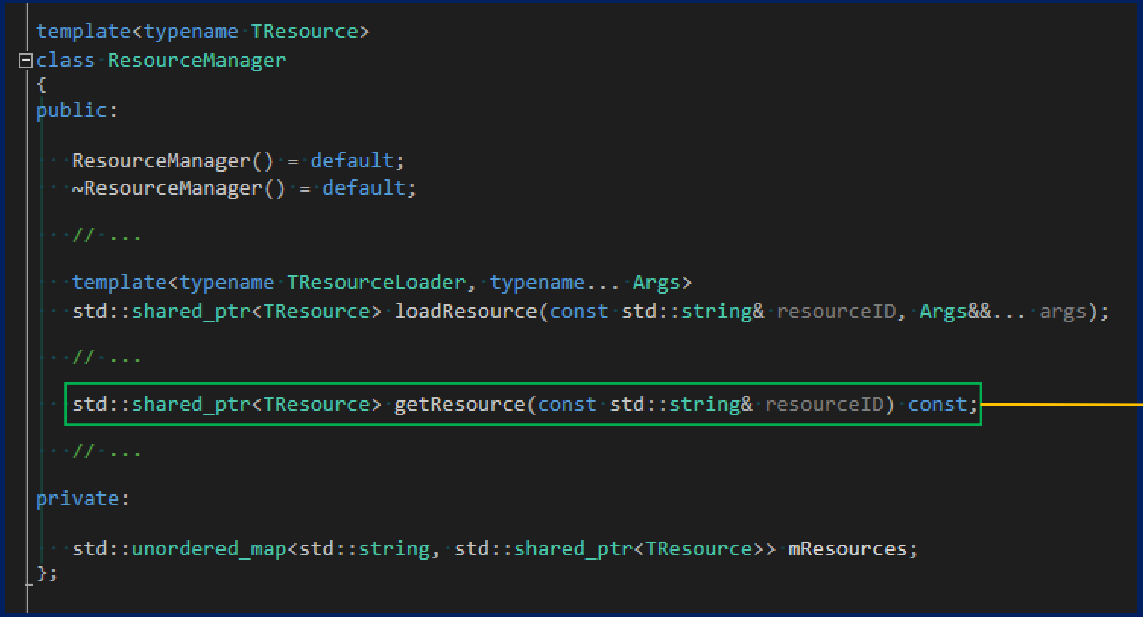


```cpp
std::shared_ptr<Shader> gameObj2DShader          = mShaderManager.getResource("game_object_2D");

std::shared_ptr<Shader> gameObj3DExplosiveShader = mShaderManager.getResource("game_object_3D_explosive");

std::shared_ptr<Model> teapot                    = mModelManager.getResource("teapot");
```

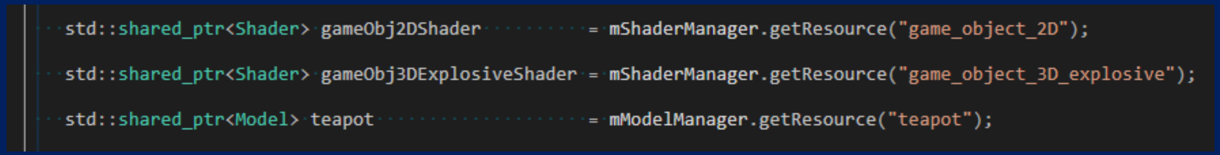
2 arguments!

Modern C++ Features

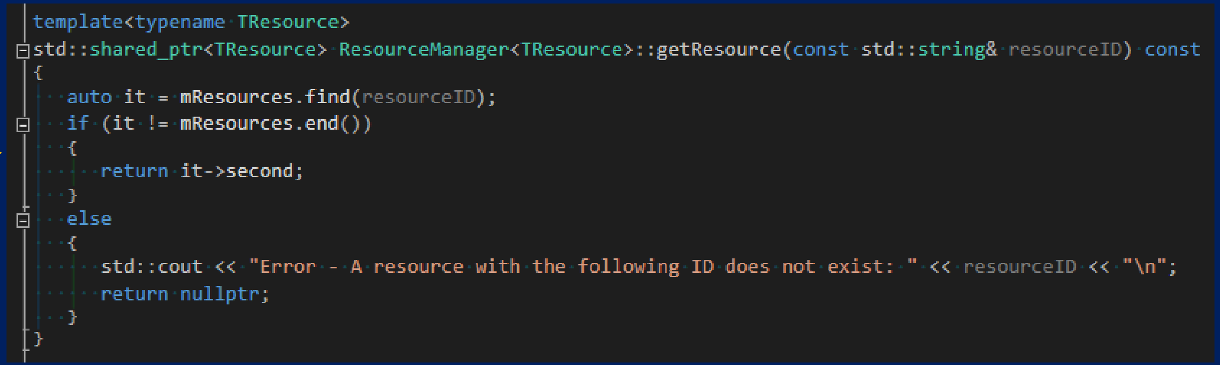
3) Variadic templates and perfect forwarding



```cpp
template<typename TResource>
class ResourceManager
{
public:

    ResourceManager() = default;
    ~ResourceManager() = default;

    // ...

    template<typename TResourceLoader, typename... Args>
    std::shared_ptr<TResource> loadResource(const std::string& resourceID, Args&&... args);

    // ...

    std::shared_ptr<TResource> getResource(const std::string& resourceID) const;

    // ...

private:

    std::unordered_map<std::string, std::shared_ptr<TResource>> mResources;
};
```



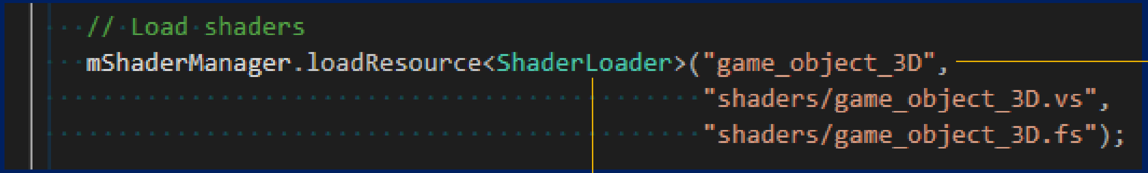
```cpp
    std::shared_ptr<Shader> gameObj2DShader          = mShaderManager.getResource("game_object_2D");

    std::shared_ptr<Shader> gameObj3DExplosiveShader = mShaderManager.getResource("game_object_3D_explosive");

    std::shared_ptr<Model> teapot                    = mModelManager.getResource("teapot");
```



```cpp
template<typename TResource>
std::shared_ptr<TResource> ResourceManager<TResource>::getResource(const std::string& resourceID) const
{
    auto it = mResources.find(resourceID);
    if (it != mResources.end())
    {
        return it->second;
    }
    else
    {
        std::cout << "Error - A resource with the following ID does not exist: " << resourceID << "\n";
        return nullptr;
    }
}
```

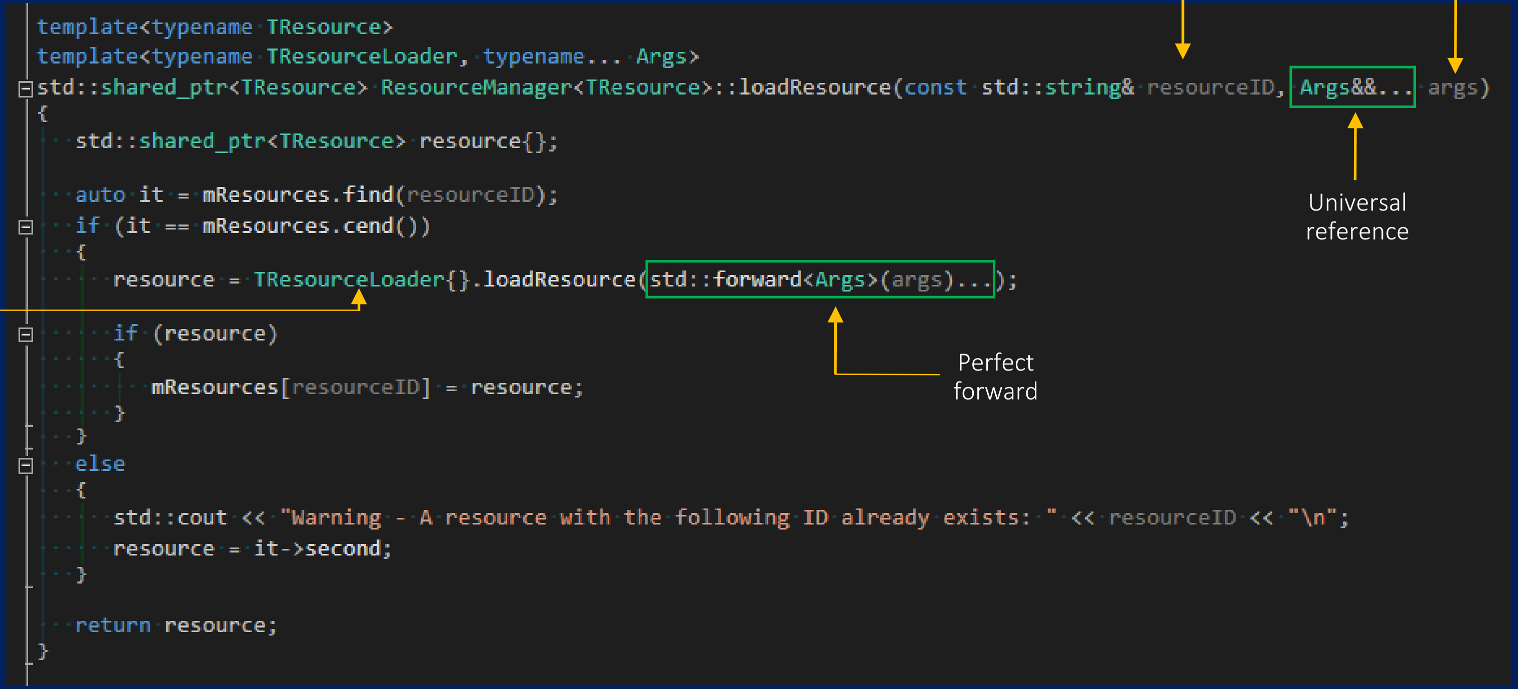
Modern C++ Features

3) Variadic templates and perfect forwarding



```cpp
// Load shaders
mShaderManager.loadResource<ShaderLoader>("game_object_3D",
                                          "shaders/game_object_3D.vs",
                                          "shaders/game_object_3D.fs");
```

Peel off the first argument from the parameter pack and forward the rest

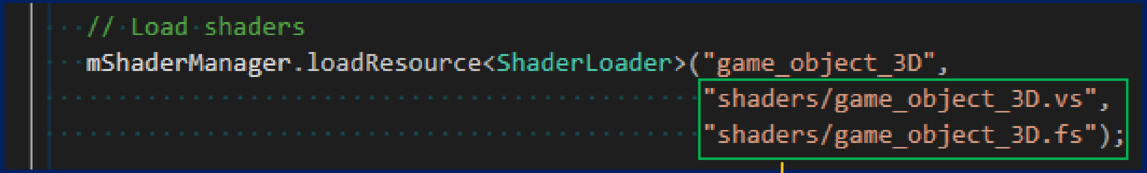


```cpp
template<typename TResource>
template<typename TResourceLoader, typename... Args>
std::shared_ptr<TResource> ResourceManager<TResource>::loadResource(const std::string& resourceID, Args&&... args)
{
   std::shared_ptr<TResource> resource{};

   auto it = mResources.find(resourceID);
   if (it == mResources.cend())
   {
      resource = TResourceLoader{}.loadResource(std::forward<Args>(args)...);

      if (resource)
      {
         mResources[resourceID] = resource;
      }
   }
   else
   {
      std::cout << "Warning - A resource with the following ID already exists: " << resourceID << "\n";
      resource = it->second;
   }

   return resource;
}
```

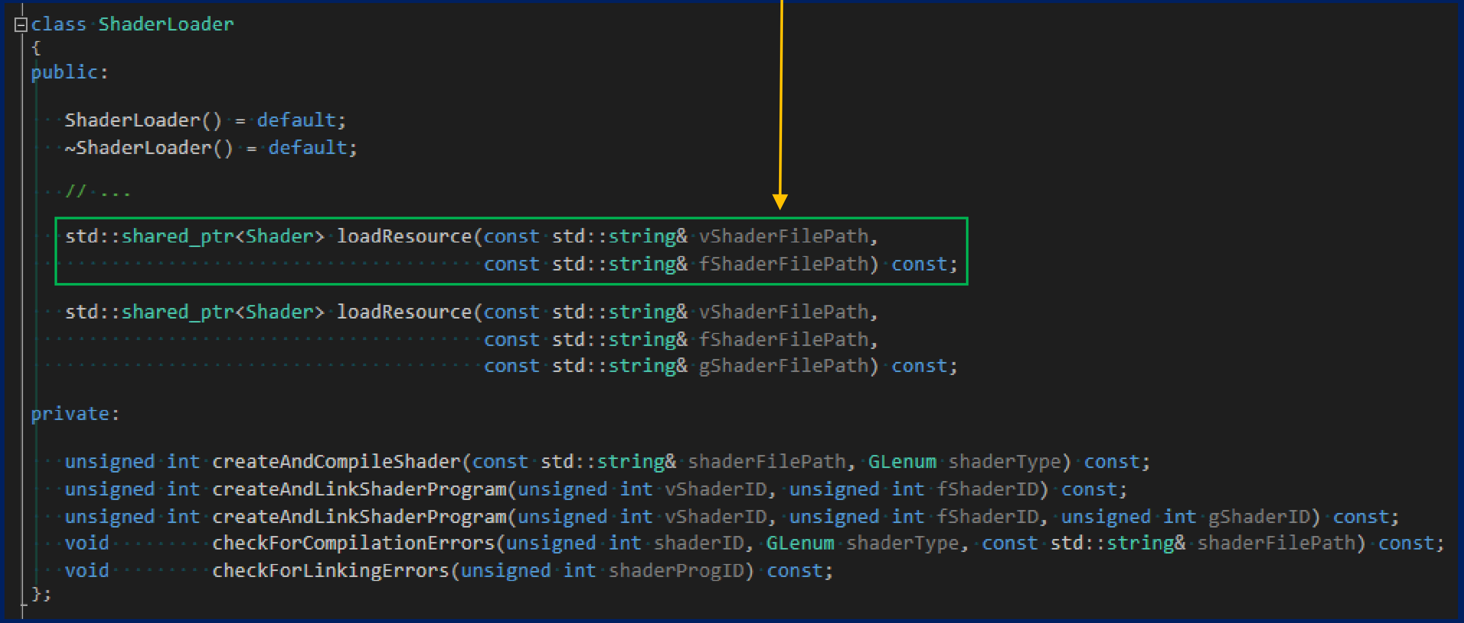
Implicit Requirement
All resource loaders must have a public function called loadResource that returns a std::shared_ptr

Modern C++ Features

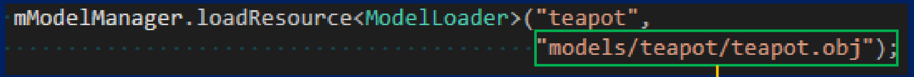
3) Variadic templates and perfect forwarding



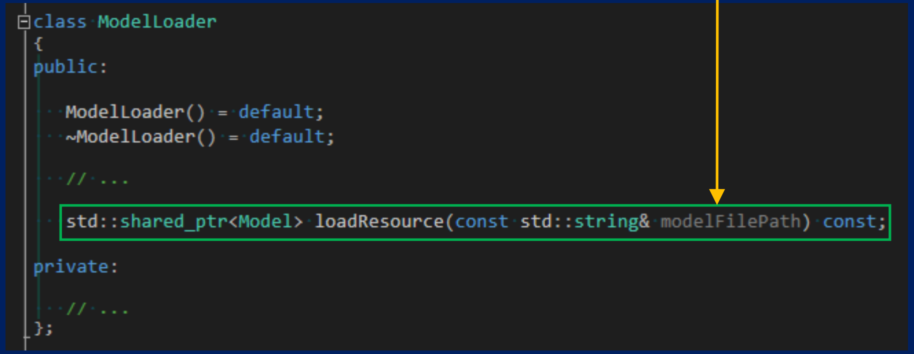
```cpp
// Load shaders
mShaderManager.loadResource<ShaderLoader>("game_object_3D",
                                          "shaders/game_object_3D.vs",
                                          "shaders/game_object_3D.fs");
```



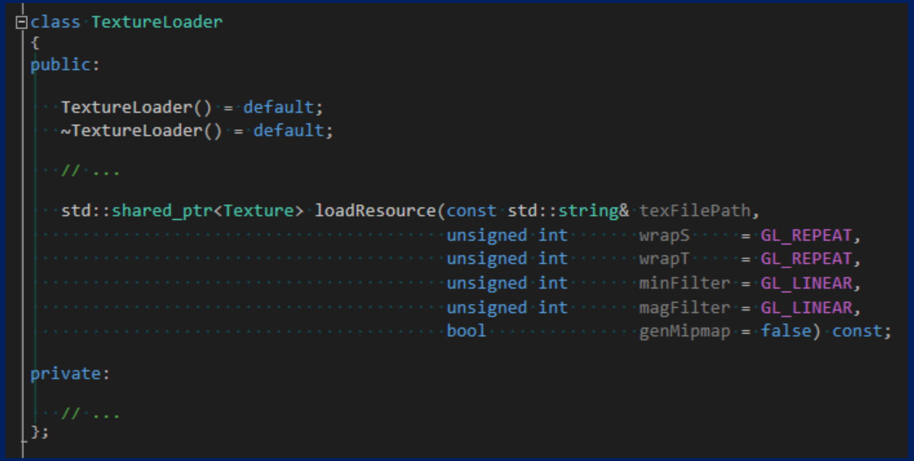
```cpp
class ShaderLoader
{
public:

   ShaderLoader() = default;
   ~ShaderLoader() = default;

   // ...

   std::shared_ptr<Shader> loadResource(const std::string& vShaderFilePath,
                                        const std::string& fShaderFilePath) const;

   std::shared_ptr<Shader> loadResource(const std::string& vShaderFilePath,
                                        const std::string& fShaderFilePath,
                                        const std::string& gShaderFilePath) const;

private:

   unsigned int createAndCompileShader(const std::string& shaderFilePath, GLenum shaderType) const;
   unsigned int createAndLinkShaderProgram(unsigned int vShaderID, unsigned int fShaderID) const;
   unsigned int createAndLinkShaderProgram(unsigned int vShaderID, unsigned int fShaderID, unsigned int gShaderID) const;
   void         checkForCompilationErrors(unsigned int shaderID, GLenum shaderType, const std::string& shaderFilePath) const;
   void         checkForLinkingErrors(unsigned int shaderProgID) const;
};
```



```cpp
mModelManager.loadResource<ModelLoader>("teapot",
                                        "models/teapot/teapot.obj");
```



```cpp
class ModelLoader
{
public:

   ModelLoader() = default;
   ~ModelLoader() = default;

   // ...

   std::shared_ptr<Model> loadResource(const std::string& modelFilePath) const;

private:

   // ...
};
```

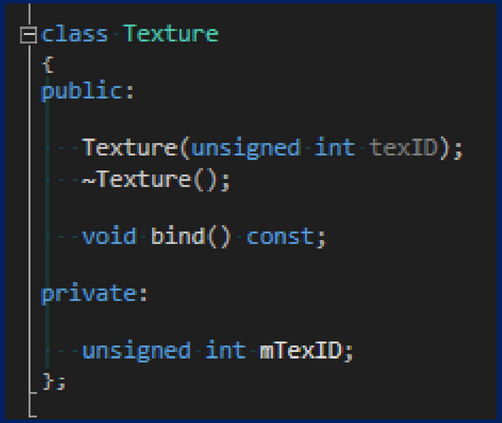


```cpp
class TextureLoader
{
public:

   TextureLoader() = default;
   ~TextureLoader() = default;

   // ...

   std::shared_ptr<Texture> loadResource(const std::string& texFilePath,
                                         unsigned int      wrapS     = GL_REPEAT,
                                         unsigned int      wrapT     = GL_REPEAT,
                                         unsigned int      minFilter = GL_LINEAR,
                                         unsigned int      magFilter = GL_LINEAR,
                                         bool              genMipmap = false) const;

private:

   // ...
};
```

Key Insights

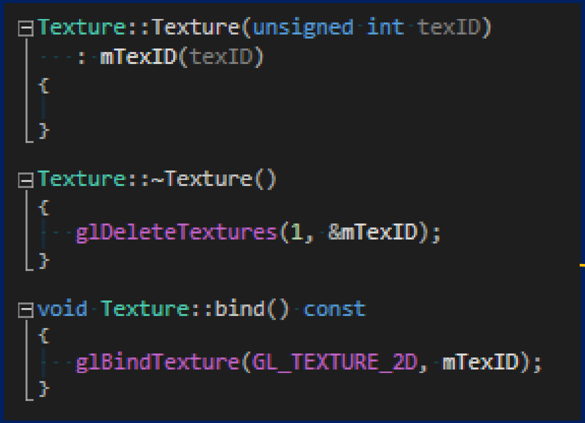
1) The object-oriented language problem - RAII and hidden destructor calls

texture.h



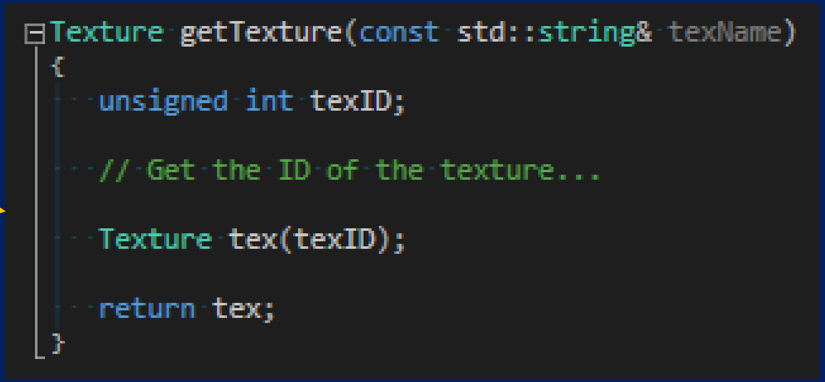
```cpp
class Texture
{
public:

    Texture(unsigned int texID);
    ~Texture();

    void bind() const;

private:

    unsigned int mTexID;
};
```

texture.cpp



```cpp
Texture::Texture(unsigned int texID)
    : mTexID(texID)
{
}

Texture::~Texture()
{
    glDeleteTextures(1, &mTexID);
}

void Texture::bind() const
{
    glBindTexture(GL_TEXTURE_2D, mTexID);
}
```

The problem

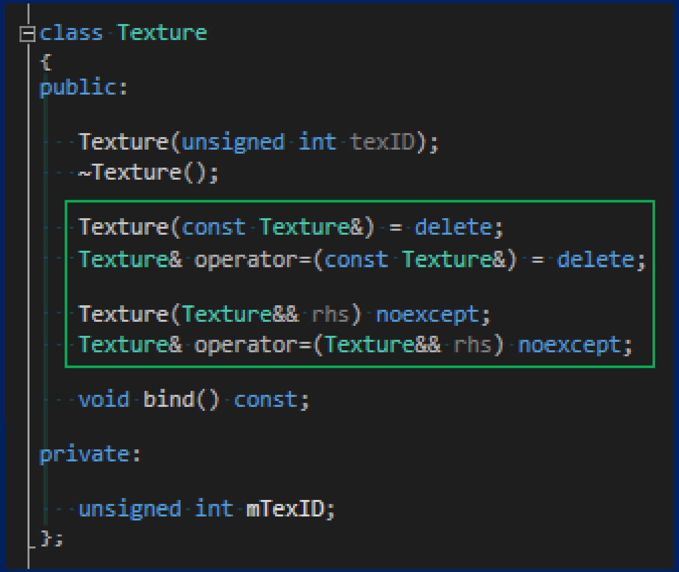


```cpp
Texture getTexture(const std::string& texName)
{
    unsigned int texID;

    // Get the ID of the texture...

    Texture tex(texID);

    return tex;
}
```

Key Insights

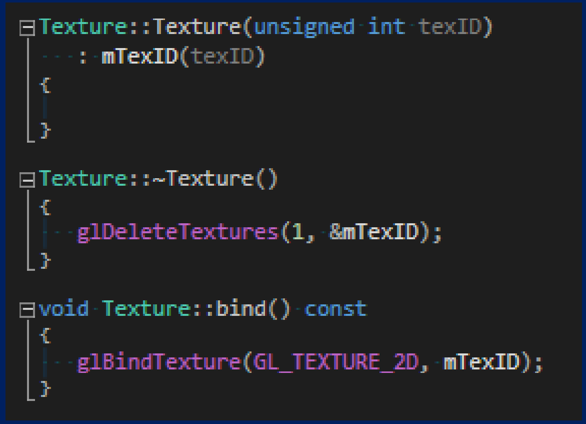
1) The object-oriented language problem - RAII and hidden destructor calls

texture.h

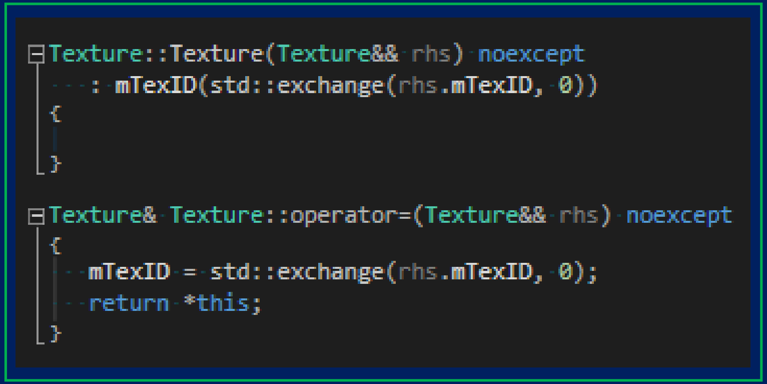


```cpp
class Texture
{
public:

    Texture(unsigned int texID);
    ~Texture();

    Texture(const Texture&) = delete;
    Texture& operator=(const Texture&) = delete;

    Texture(Texture&& rhs) noexcept;
    Texture& operator=(Texture&& rhs) noexcept;

    void bind() const;

private:

    unsigned int mTexID;
};
```

texture.cpp



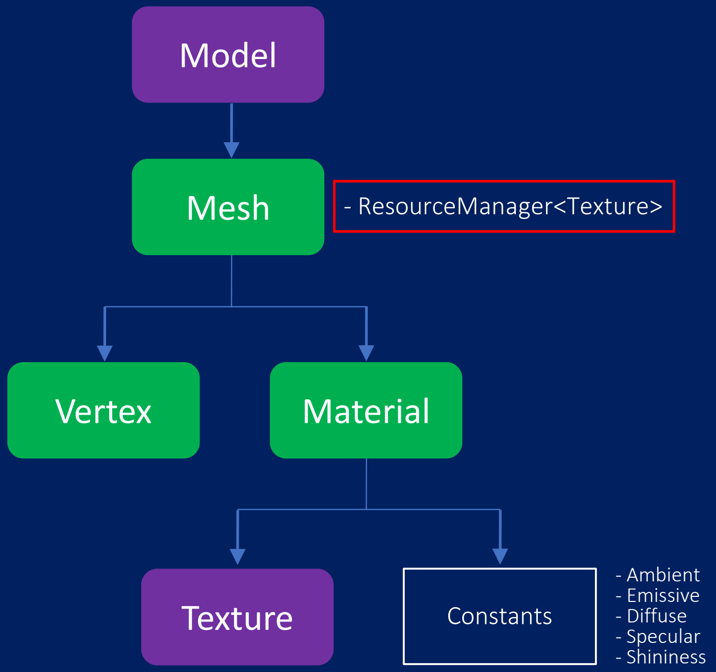
```cpp
Texture::Texture(unsigned int texID)
    : mTexID(texID)
{
}

Texture::~Texture()
{
    glDeleteTextures(1, &mTexID);
}

void Texture::bind() const
{
    glBindTexture(GL_TEXTURE_2D, mTexID);
}
```



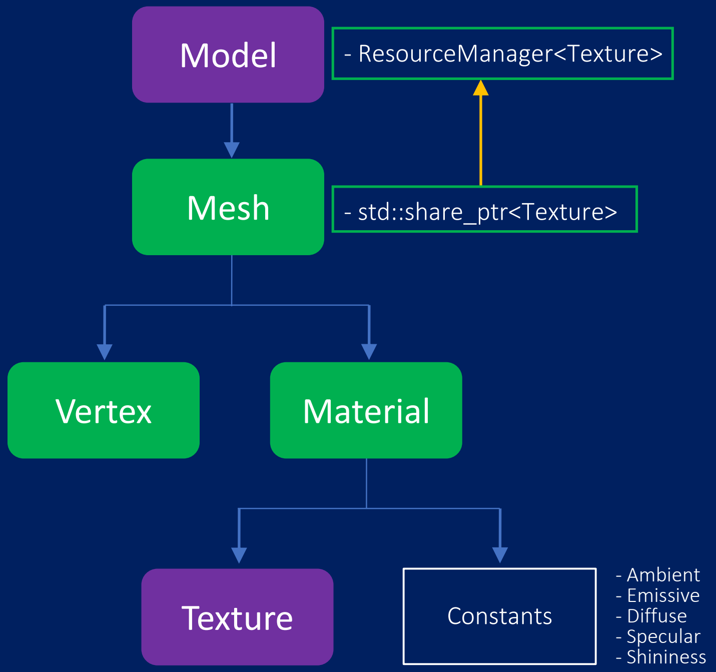
```cpp
Texture::Texture(Texture&& rhs) noexcept
    : mTexID(std::exchange(rhs.mTexID, 0))
{
}

Texture& Texture::operator=(Texture&& rhs) noexcept
{
    mTexID = std::exchange(rhs.mTexID, 0);
    return *this;
}
```

Key Insights

2) The repeated textures problem



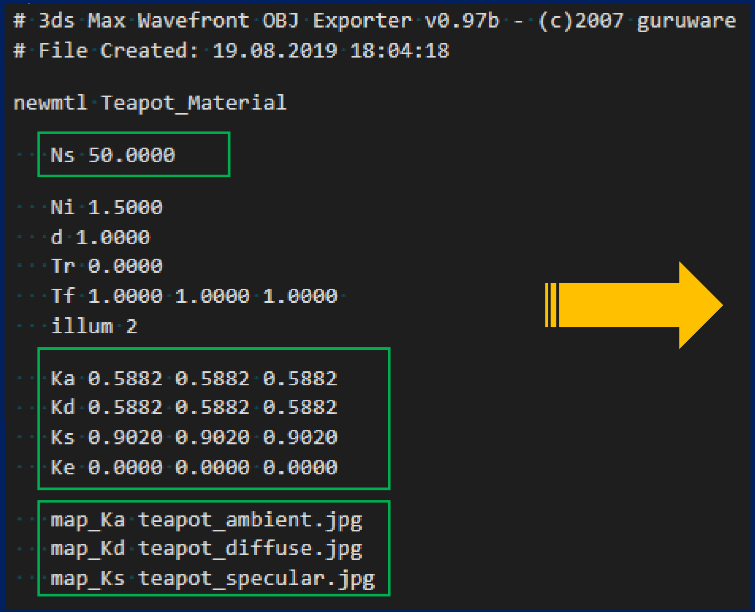
Avoid loading a texture more than once by allowing meshes to share them!



Key Insights

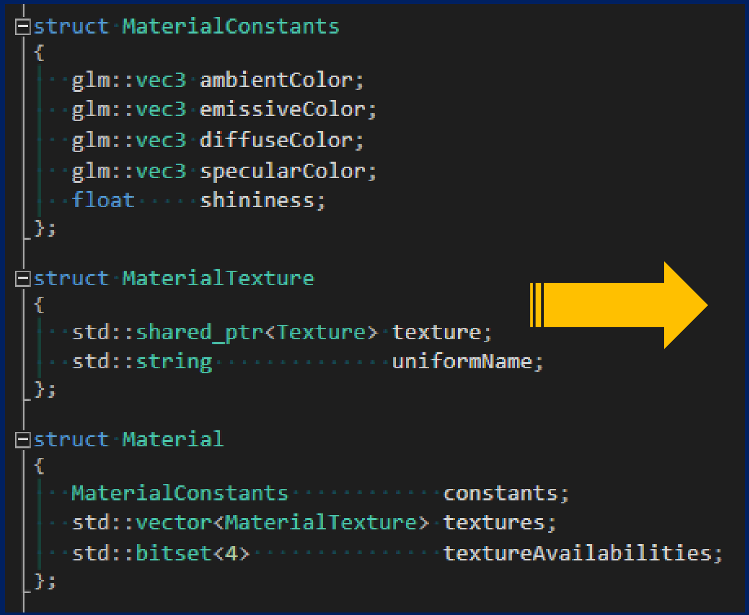
3) The constant VS texture problem

teapot.mtl



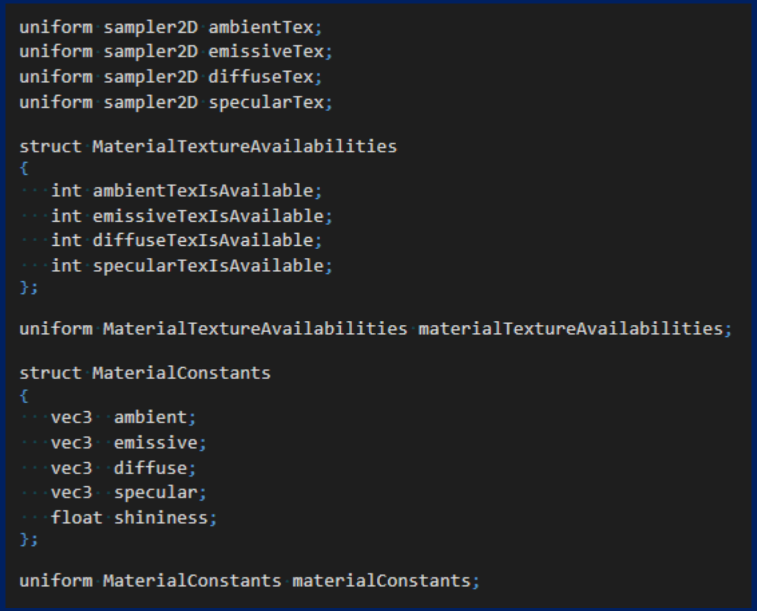
```
# 3ds Max Wavefront OBJ Exporter v0.97b - (c)2007 guruware
# File Created: 19.08.2019 18:04:18

newmtl Teapot_Material

    Ns 50.0000

    Ni 1.5000
    d 1.0000
    Tr 0.0000
    Tf 1.0000 1.0000 1.0000
    illum 2

    Ka 0.5882 0.5882 0.5882
    Kd 0.5882 0.5882 0.5882
    Ks 0.9020 0.9020 0.9020
    Ke 0.0000 0.0000 0.0000

    map_Ka teapot_ambient.jpg
    map_Kd teapot_diffuse.jpg
    map_Ks teapot_specular.jpg
```

mesh.h



```
struct MaterialConstants
{
    glm::vec3 ambientColor;
    glm::vec3 emissiveColor;
    glm::vec3 diffuseColor;
    glm::vec3 specularColor;
    float     shininess;
};

struct MaterialTexture
{
    std::shared_ptr<Texture> texture;
    std::string              uniformName;
};

struct Material
{
    MaterialConstants            constants;
    std::vector<MaterialTexture> textures;
    std::bitset<4>               textureAvailabilities;
};
```

game_object_3D.fs



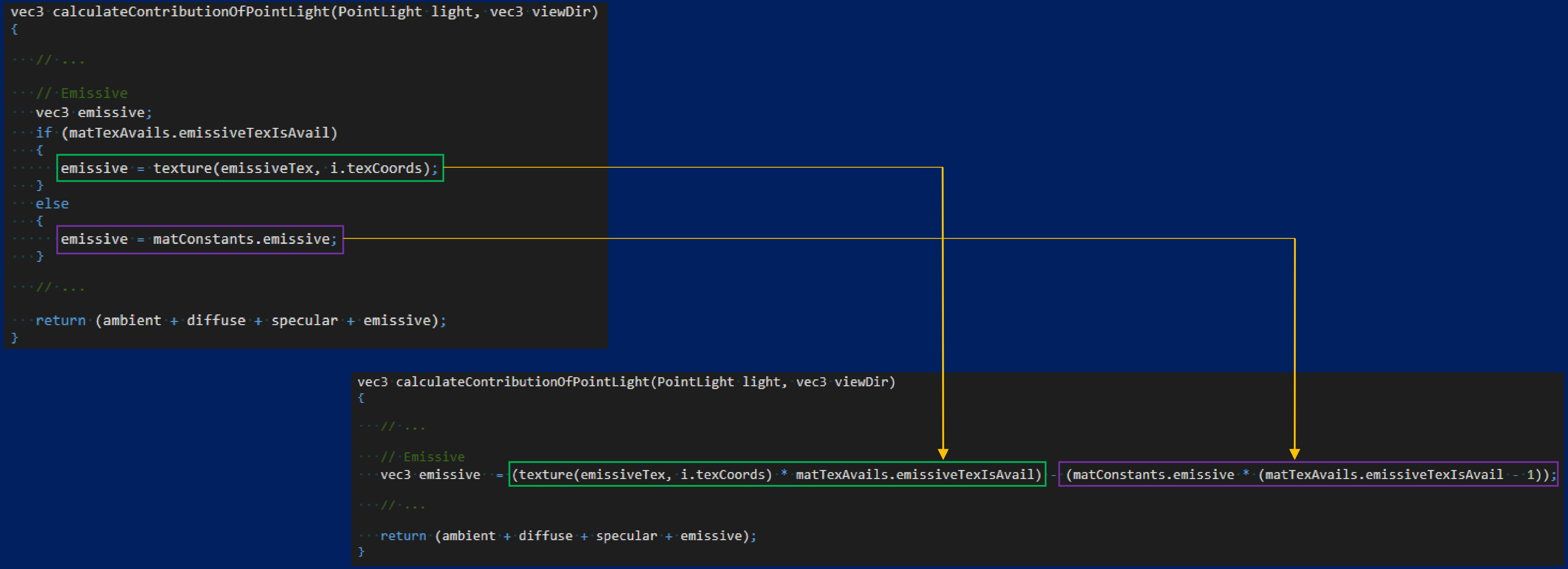
```
uniform sampler2D ambientTex;
uniform sampler2D emissiveTex;
uniform sampler2D diffuseTex;
uniform sampler2D specularTex;

struct MaterialTextureAvailabilities
{
    int ambientTexIsAvailable;
    int emissiveTexIsAvailable;
    int diffuseTexIsAvailable;
    int specularTexIsAvailable;
};

uniform MaterialTextureAvailabilities materialTextureAvailabilities;

struct MaterialConstants
{
    vec3  ambient;
    vec3  emissive;
    vec3  diffuse;
    vec3  specular;
    float shininess;
};

uniform MaterialConstants materialConstants;
```

How does the shader efficiently decide whether it should use a constant or a texture?

Key Insights

3) The constant VS texture problem

game_object_3D.fs

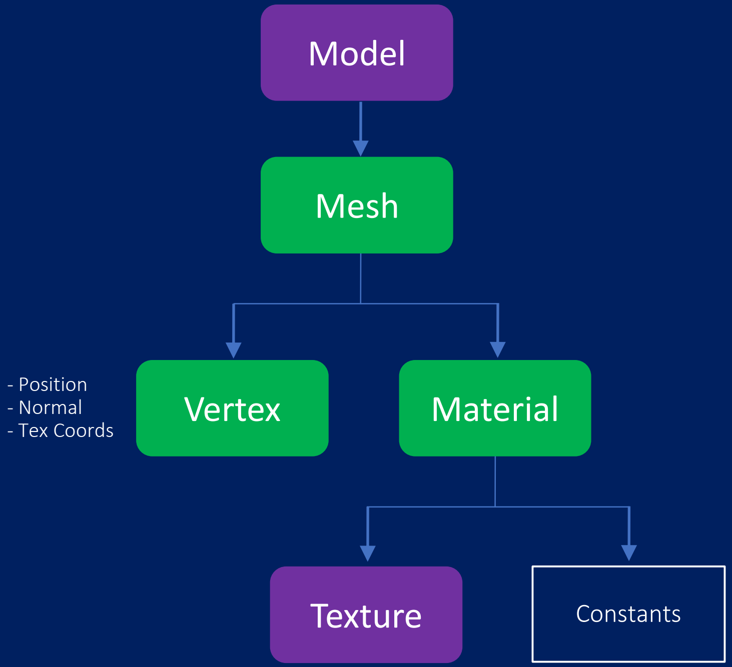


```
vec3 calculateContributionOfPointLight(PointLight light, vec3 viewDir)
{
    // ...

    // Emissive
    vec3 emissive;
    if (matTexAvails.emissiveTexIsAvail)
    {
        emissive = texture(emissiveTex, i.texCoords);
    }
    else
    {
        emissive = matConstants.emissive;
    }

    // ...

    return (ambient + diffuse + specular + emissive);
}
```

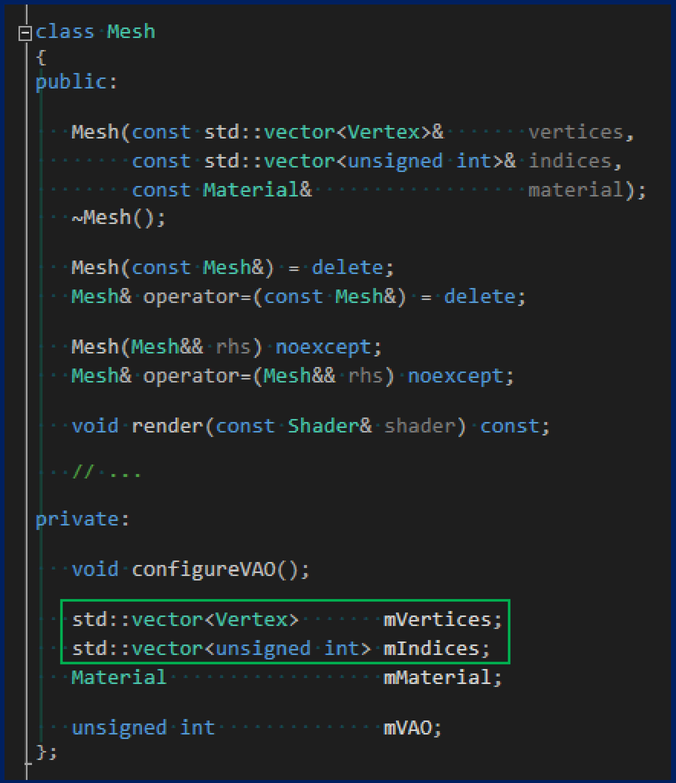
```
vec3 calculateContributionOfPointLight(PointLight light, vec3 viewDir)
{
    // ...

    // Emissive
    vec3 emissive  = (texture(emissiveTex, i.texCoords) * matTexAvails.emissiveTexIsAvail) - (matConstants.emissive * (matTexAvails.emissiveTexIsAvail - 1));

    // ...

    return (ambient + diffuse + specular + emissive);
}
```

Key Insights

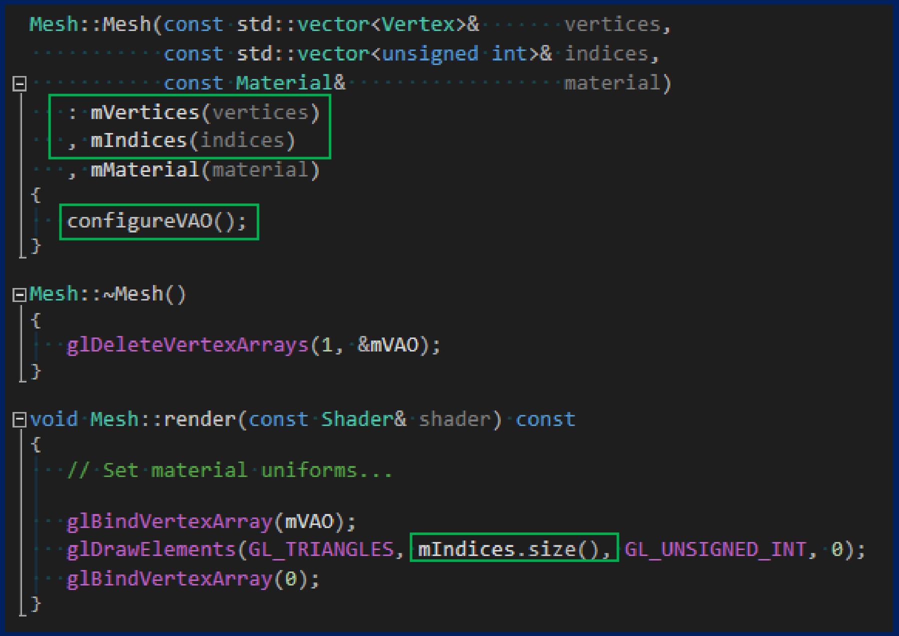
4) The expensive mesh problem



mesh.h


```cpp
class Mesh
{
public:

    Mesh(const std::vector<Vertex>&       vertices,
         const std::vector<unsigned int>& indices,
         const Material&                  material);
    ~Mesh();

    Mesh(const Mesh&) = delete;
    Mesh& operator=(const Mesh&) = delete;

    Mesh(Mesh&& rhs) noexcept;
    Mesh& operator=(Mesh&& rhs) noexcept;

    void render(const Shader& shader) const;

    // ...

private:

    void configureVAO();

    std::vector<Vertex>       mVertices;
    std::vector<unsigned int> mIndices;
    Material                  mMaterial;

    unsigned int              mVAO;
};
```

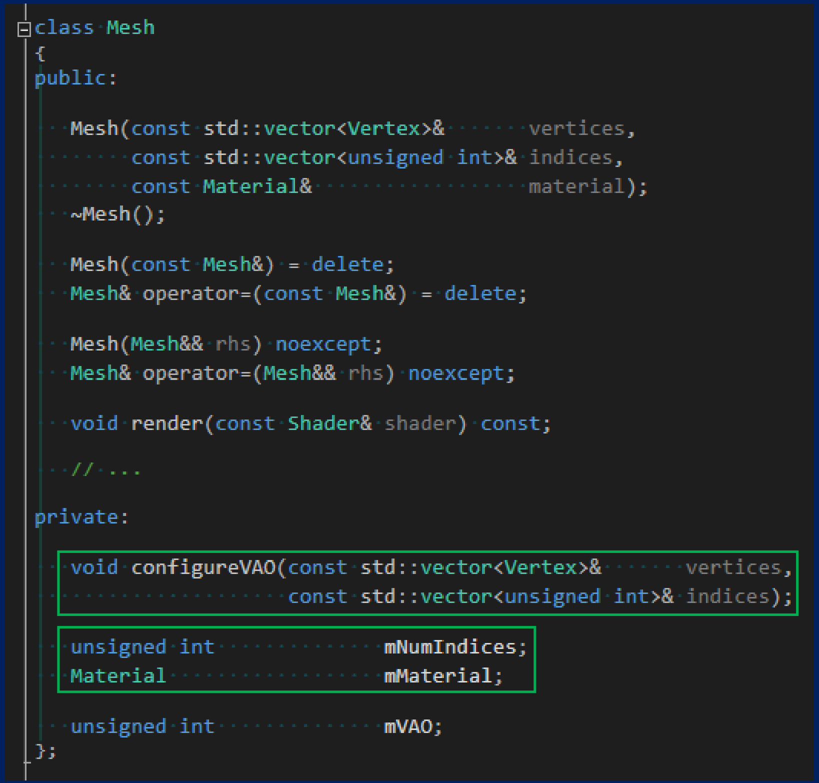
mesh.cpp


```cpp
Mesh::Mesh(const std::vector<Vertex>&       vertices,
           const std::vector<unsigned int>& indices,
           const Material&                  material)
    : mVertices(vertices)
    , mIndices(indices)
    , mMaterial(material)
{
    configureVAO();
}

Mesh::~Mesh()
{
    glDeleteVertexArrays(1, &mVAO);
}

void Mesh::render(const Shader& shader) const
{
    // Set material uniforms...

    glBindVertexArray(mVAO);
    glDrawElements(GL_TRIANGLES, mIndices.size(), GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
}
```

Key Insights

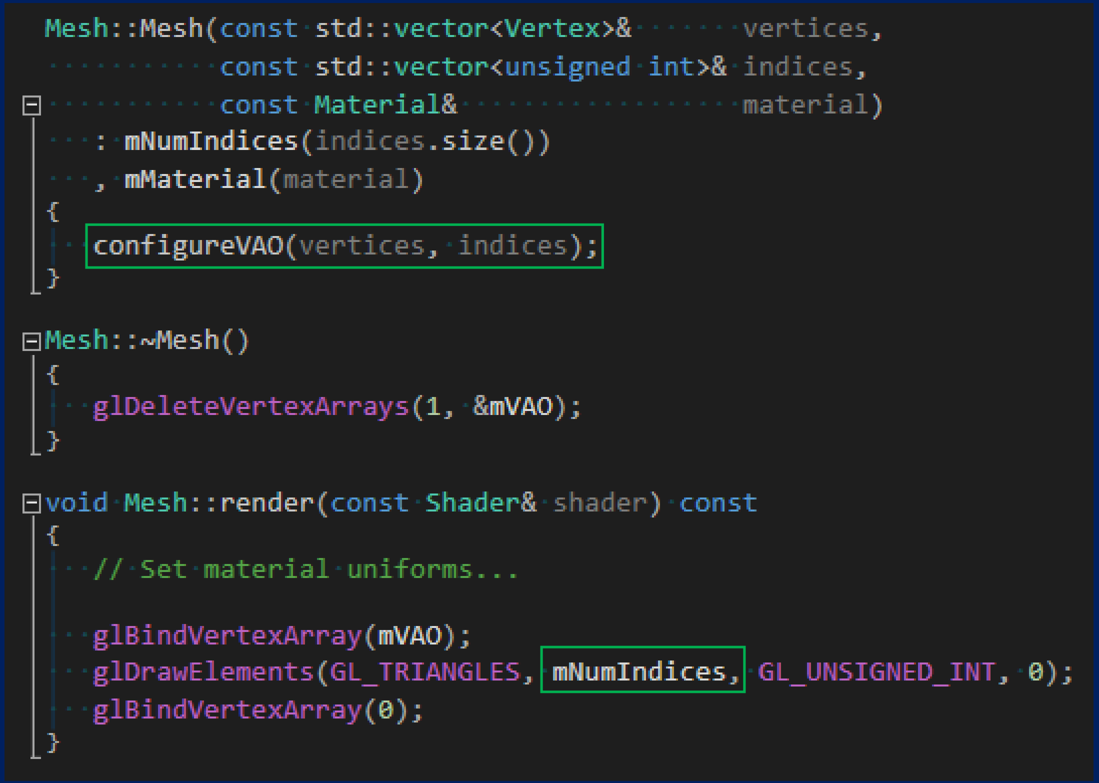
4) The expensive mesh problem

mesh.h



```
class Mesh
{
public:
    Mesh(const std::vector<Vertex>&       vertices,
         const std::vector<unsigned int>& indices,
         const Material&                  material);
    ~Mesh();

    Mesh(const Mesh&) = delete;
    Mesh& operator=(const Mesh&) = delete;

    Mesh(Mesh&& rhs) noexcept;
    Mesh& operator=(Mesh&& rhs) noexcept;

    void render(const Shader& shader) const;

    // ...

private:
    void configureVAO(const std::vector<Vertex>&       vertices,
                      const std::vector<unsigned int>& indices);

    unsigned int             mNumIndices;
    Material                 mMaterial;

    unsigned int             mVAO;
};
```

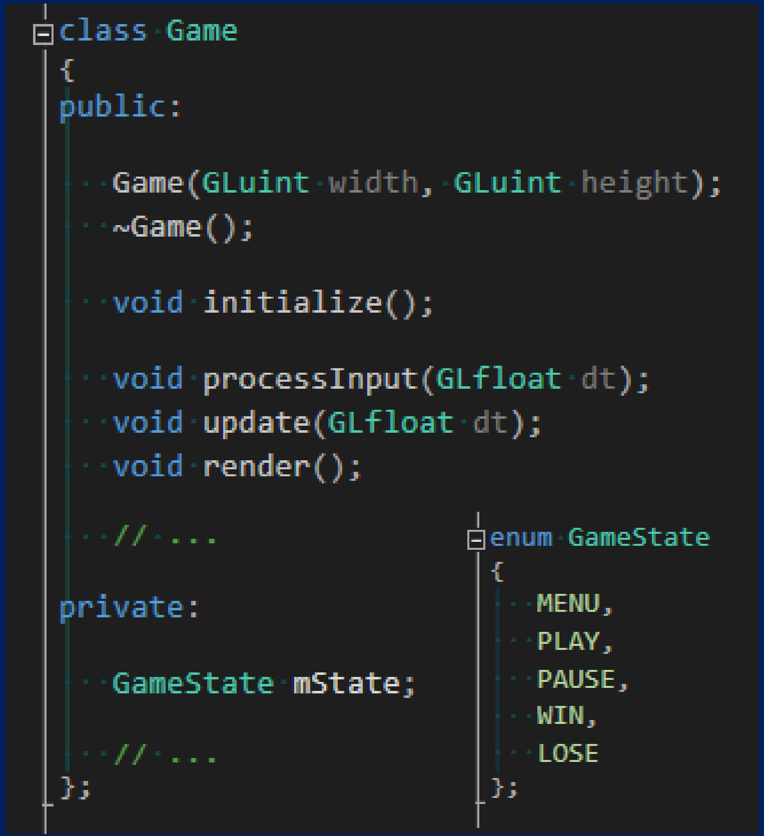
mesh.cpp



```
Mesh::Mesh(const std::vector<Vertex>&       vertices,
           const std::vector<unsigned int>& indices,
           const Material&                  material)
    : mNumIndices(indices.size())
    , mMaterial(material)
{
    configureVAO(vertices, indices);
}

Mesh::~Mesh()
{
    glDeleteVertexArrays(1, &mVAO);
}

void Mesh::render(const Shader& shader) const
{
    // Set material uniforms...

    glBindVertexArray(mVAO);
    glDrawElements(GL_TRIANGLES, mNumIndices, GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
}
```

Design Patterns

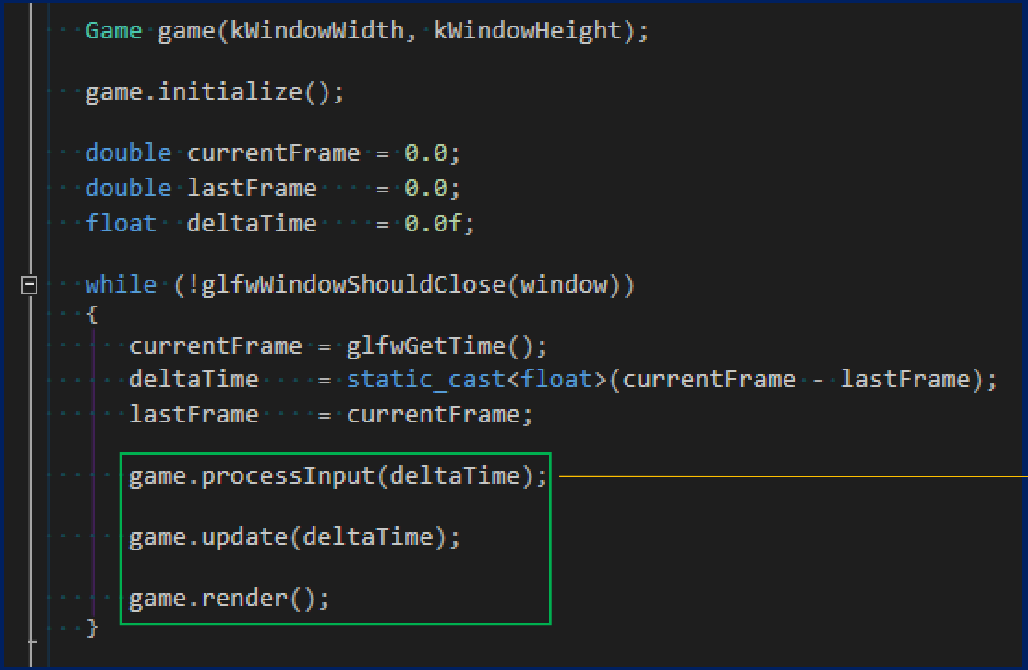
1) The state design pattern



```cpp
class Game
{
public:

    Game(GLuint width, GLuint height);
    ~Game();

    void initialize();

    void processInput(GLfloat dt);
    void update(GLfloat dt);
    void render();

    // ...

private:

    GameState mState;

    // ...
};
```

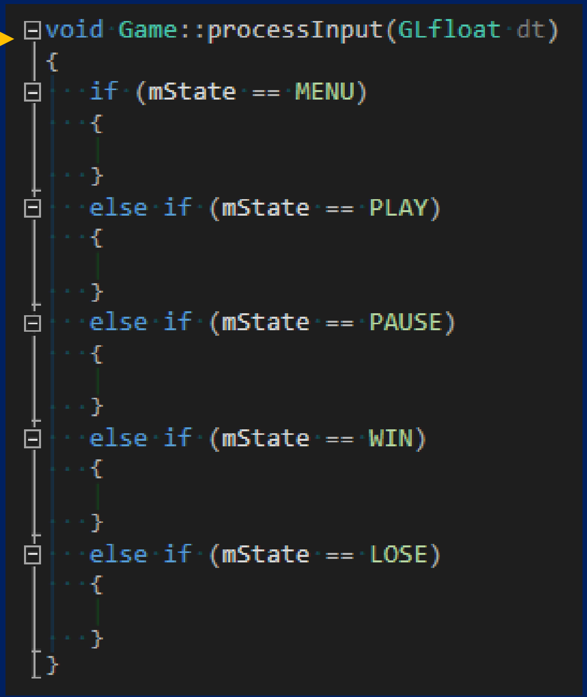
```cpp
enum GameState
{
    MENU,
    PLAY,
    PAUSE,
    WIN,
    LOSE
};
```

"Classic game loop"



```cpp
Game game(kWindowWidth, kWindowHeight);

game.initialize();

double currentFrame = 0.0;
double lastFrame    = 0.0;
float  deltaTime    = 0.0f;

while (!glfwWindowShouldClose(window))
{
    currentFrame = glfwGetTime();
    deltaTime    = static_cast<float>(currentFrame - lastFrame);
    lastFrame    = currentFrame;

    game.processInput(deltaTime);

    game.update(deltaTime);

    game.render();
}
```

"Classic state management"

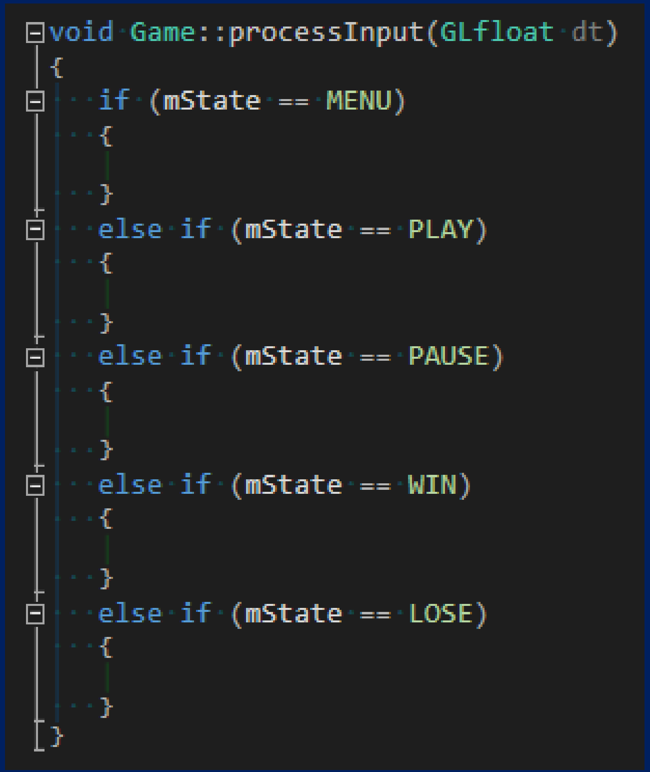


```cpp
void Game::processInput(GLfloat dt)
{
    if (mState == MENU)
    {

    }
    else if (mState == PLAY)
    {

    }
    else if (mState == PAUSE)
    {

    }
    else if (mState == WIN)
    {

    }
    else if (mState == LOSE)
    {

    }
}
```

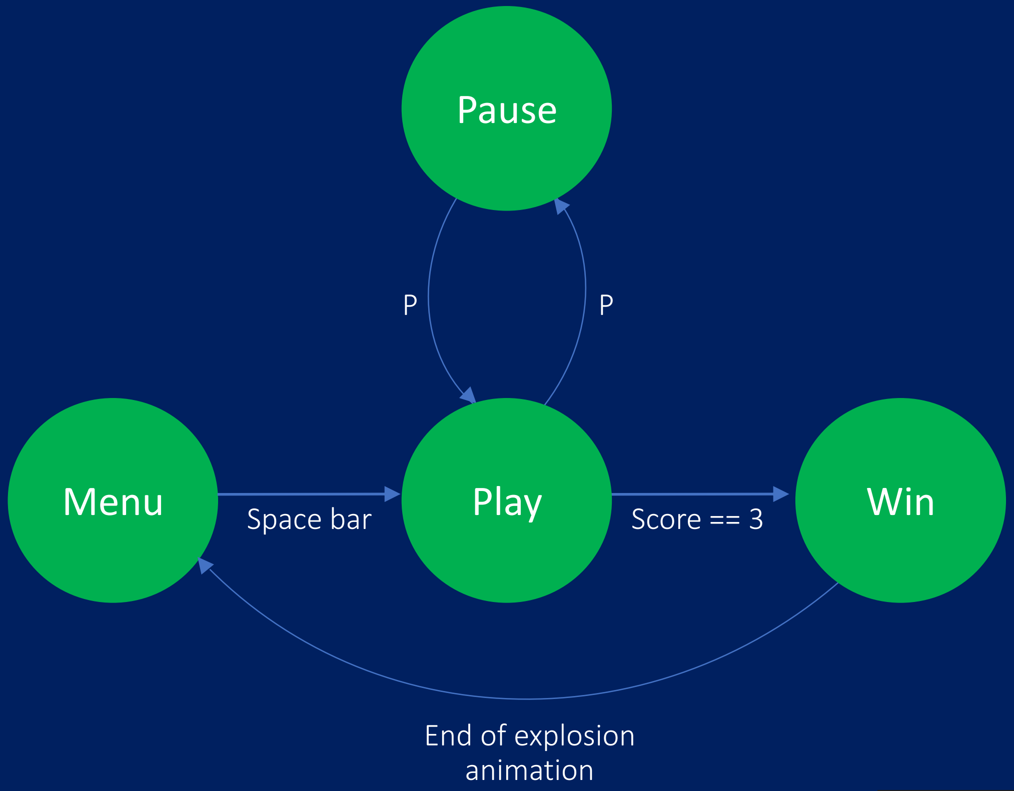
This does not scale without
a lot of tears :-(

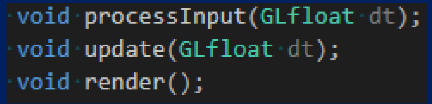
Design Patterns

1) The state design pattern



```
void Game::processInput(GLfloat dt)
{
    if (mState == MENU)
    {

    }
    else if (mState == PLAY)
    {

    }
    else if (mState == PAUSE)
    {

    }
    else if (mState == WIN)
    {

    }
    else if (mState == LOSE)
    {

    }
}
```

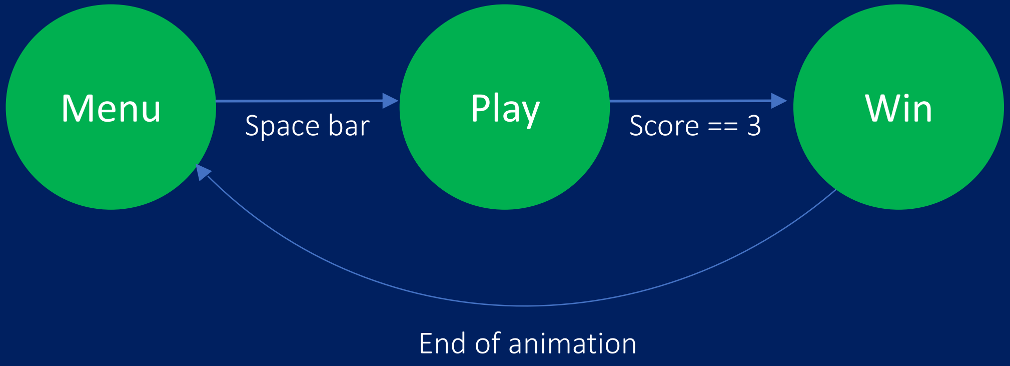




```
void processInput(GLfloat dt);
void update(GLfloat dt);
void render();
```

- Each state implements its own versions of:
- Each state only has access to what it needs, and it can share what it owns with other states.
- Each state checks the conditions that could lead to a state change, and notifies the FSM when necessary.

Design Patterns



1) The state design pattern

```cpp
class State
{
public:

    State() = default;
    virtual ~State() = default;

    // ...

    virtual void enter() = 0;
    virtual void execute(float deltaTime) = 0;
    virtual void exit() = 0;
};
```

```cpp
void processInput(GLfloat dt);
void update(GLfloat dt);
void render();
```

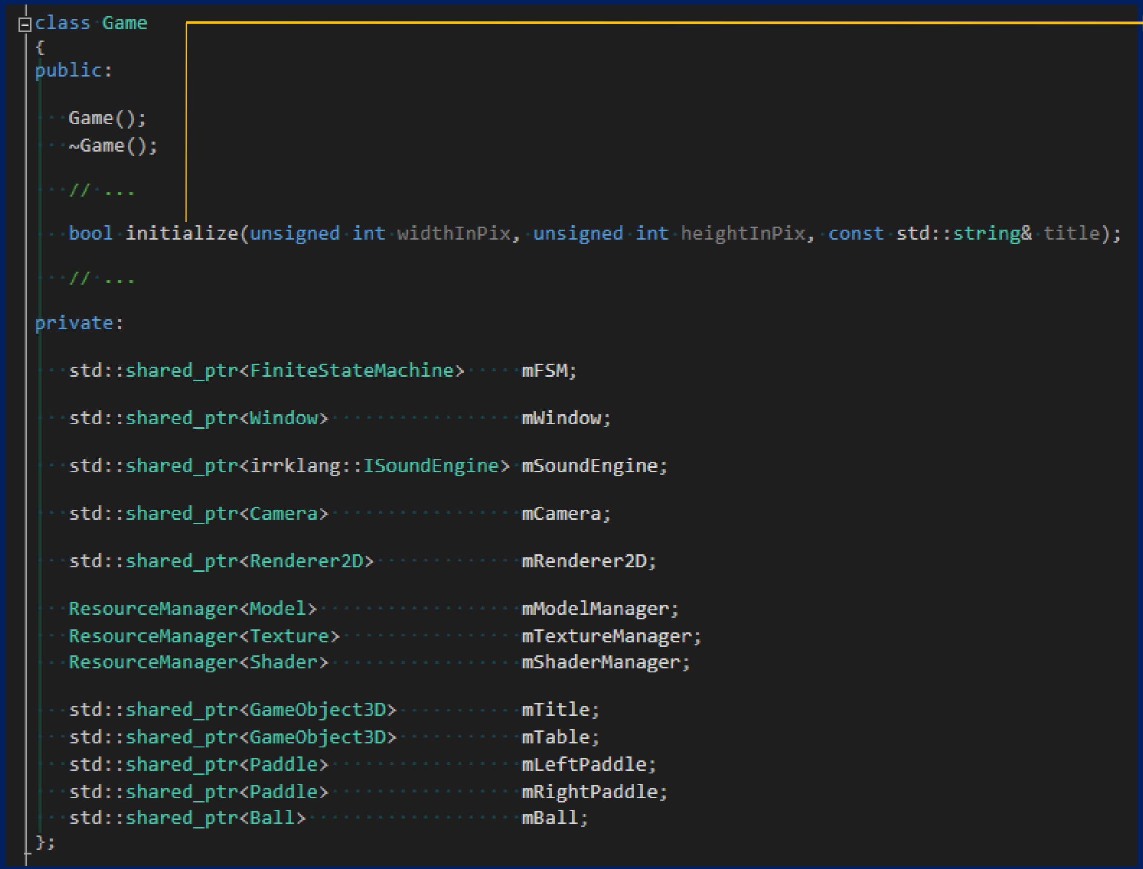
```cpp
class FiniteStateMachine
{
public:

    FiniteStateMachine() = default;
    ~FiniteStateMachine() = default;

    // ...

    void initialize(MapOfStates&&      states,
                    const std::string& initialStateID);

    void executeCurrentState(float deltaTime) const;

    void changeState(const std::string& newStateID);

    // ...

private:

    MapOfStates            mStates;

    std::shared_ptr<State> mCurrentState;

    // ...
};
```

```cpp
using MapOfStates = std::unordered_map<std::string, std::shared_ptr<State>>;
```

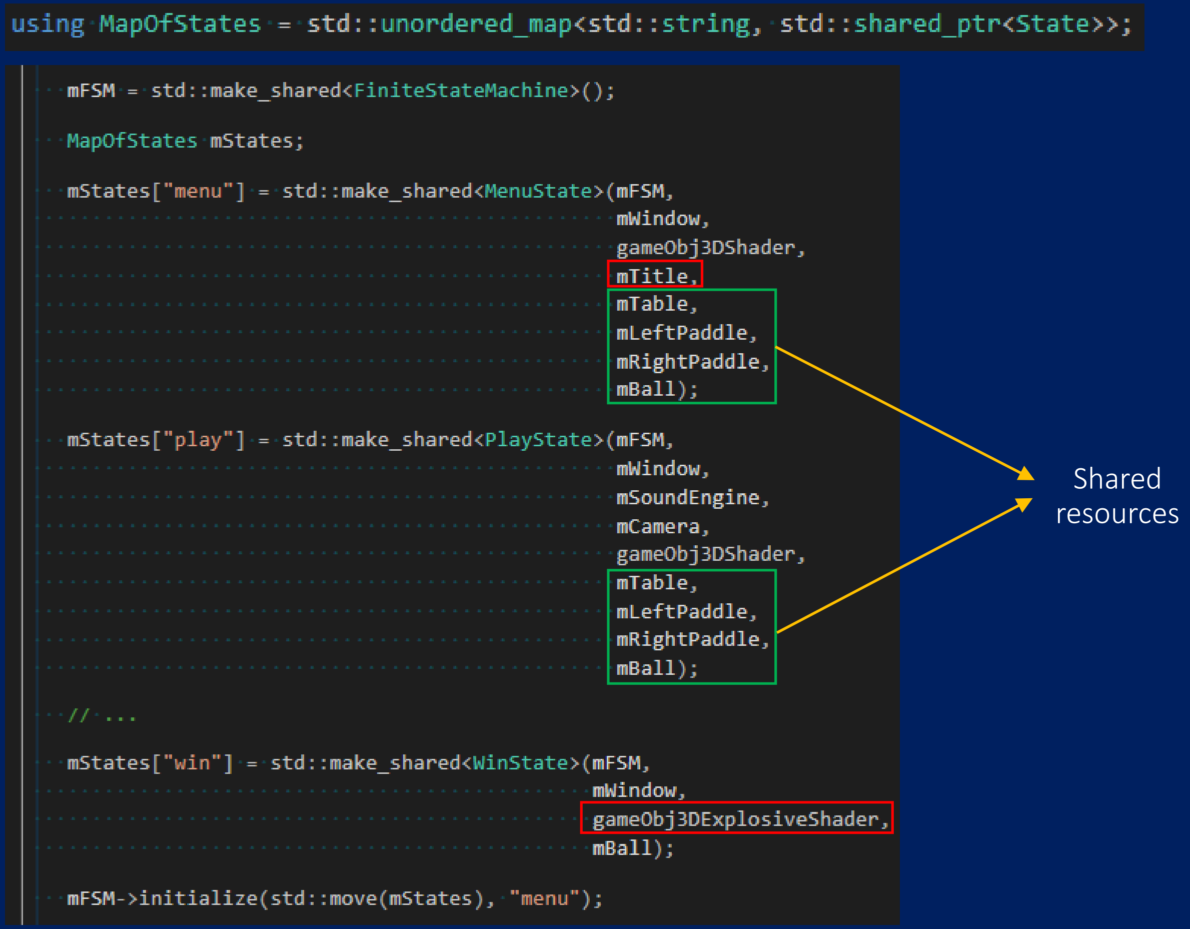
```cpp
void FiniteStateMachine::initialize(MapOfStates&&      states,
                                    const std::string& initialStateID)
{
    mStates = std::move(states);

    auto it = mStates.find(initialStateID);
    if (it != mStates.end())
    {
        mCurrentState = it->second;
        mCurrentState->enter();
    }
    else { /* ... */ }
}

void FiniteStateMachine::executeCurrentState(float deltaTime) const
{
    mCurrentState->execute(deltaTime);
}

void FiniteStateMachine::changeState(const std::string& newStateID)
{
    auto it = mStates.find(newStateID);
    if (it != mStates.end())
    {
        mCurrentState->exit();
        mCurrentState = it->second;
        mCurrentState->enter();
    }
    else { /* ... */ }
}
```

Design Patterns

1) The state design pattern



```cpp
class Game
{
public:

    Game();
    ~Game();

    // ...

    bool initialize(unsigned int widthInPix, unsigned int heightInPix, const std::string& title);

    // ...

private:

    std::shared_ptr<FiniteStateMachine>     mFSM;

    std::shared_ptr<Window>                 mWindow;

    std::shared_ptr<irrklang::ISoundEngine> mSoundEngine;

    std::shared_ptr<Camera>                 mCamera;

    std::shared_ptr<Renderer2D>             mRenderer2D;

    ResourceManager<Model>                  mModelManager;
    ResourceManager<Texture>                mTextureManager;
    ResourceManager<Shader>                 mShaderManager;

    std::shared_ptr<GameObject3D>           mTitle;
    std::shared_ptr<GameObject3D>           mTable;
    std::shared_ptr<Paddle>                 mLeftPaddle;
    std::shared_ptr<Paddle>                 mRightPaddle;
    std::shared_ptr<Ball>                   mBall;
};
```

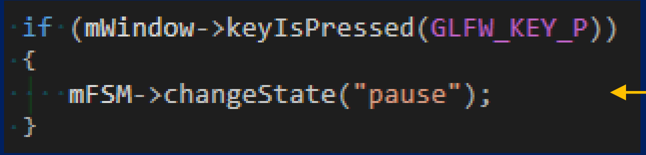


```cpp
using MapOfStates = std::unordered_map<std::string, std::shared_ptr<State>>;
```

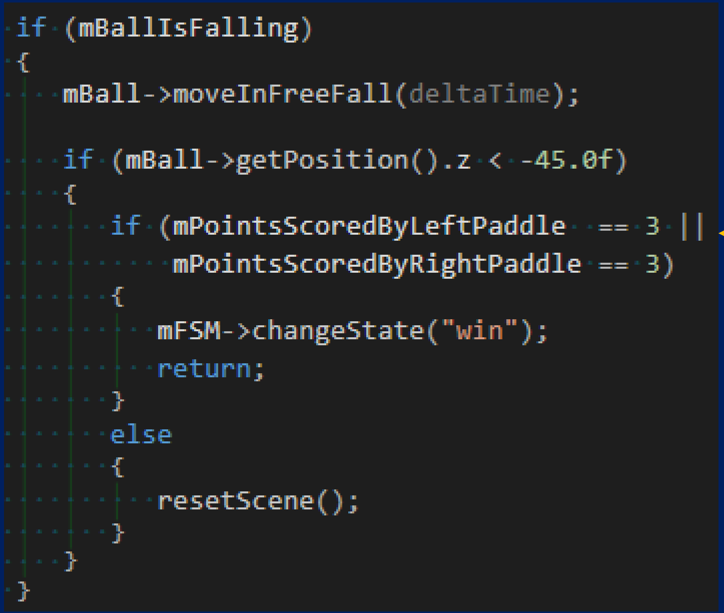
```cpp
mFSM = std::make_shared<FiniteStateMachine>();

MapOfStates mStates;

mStates["menu"] = std::make_shared<MenuState>(mFSM,
                                              mWindow,
                                              gameObj3DShader,
                                              mTitle,
                                              mTable,
                                              mLeftPaddle,
                                              mRightPaddle,
                                              mBall);

mStates["play"] = std::make_shared<PlayState>(mFSM,
                                              mWindow,
                                              mSoundEngine,
                                              mCamera,
                                              gameObj3DShader,
                                              mTable,
                                              mLeftPaddle,
                                              mRightPaddle,
                                              mBall);

// ...

mStates["win"] = std::make_shared<WinState>(mFSM,
                                            mWindow,
                                            gameObj3DExplosiveShader,
                                            mBall);

mFSM->initialize(std::move(mStates), "menu");
```

Design Patterns

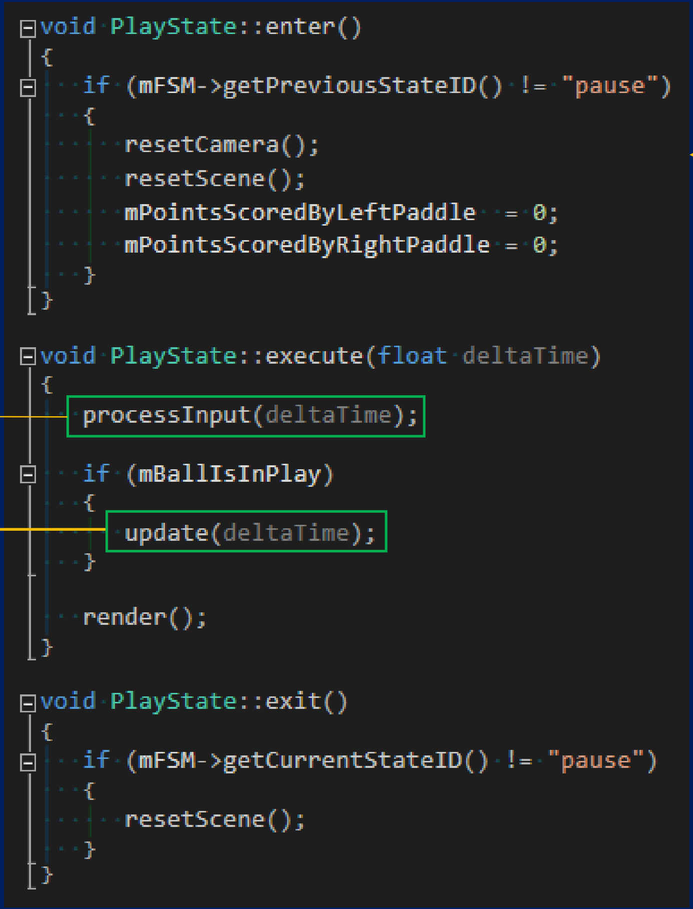
1) The state design pattern



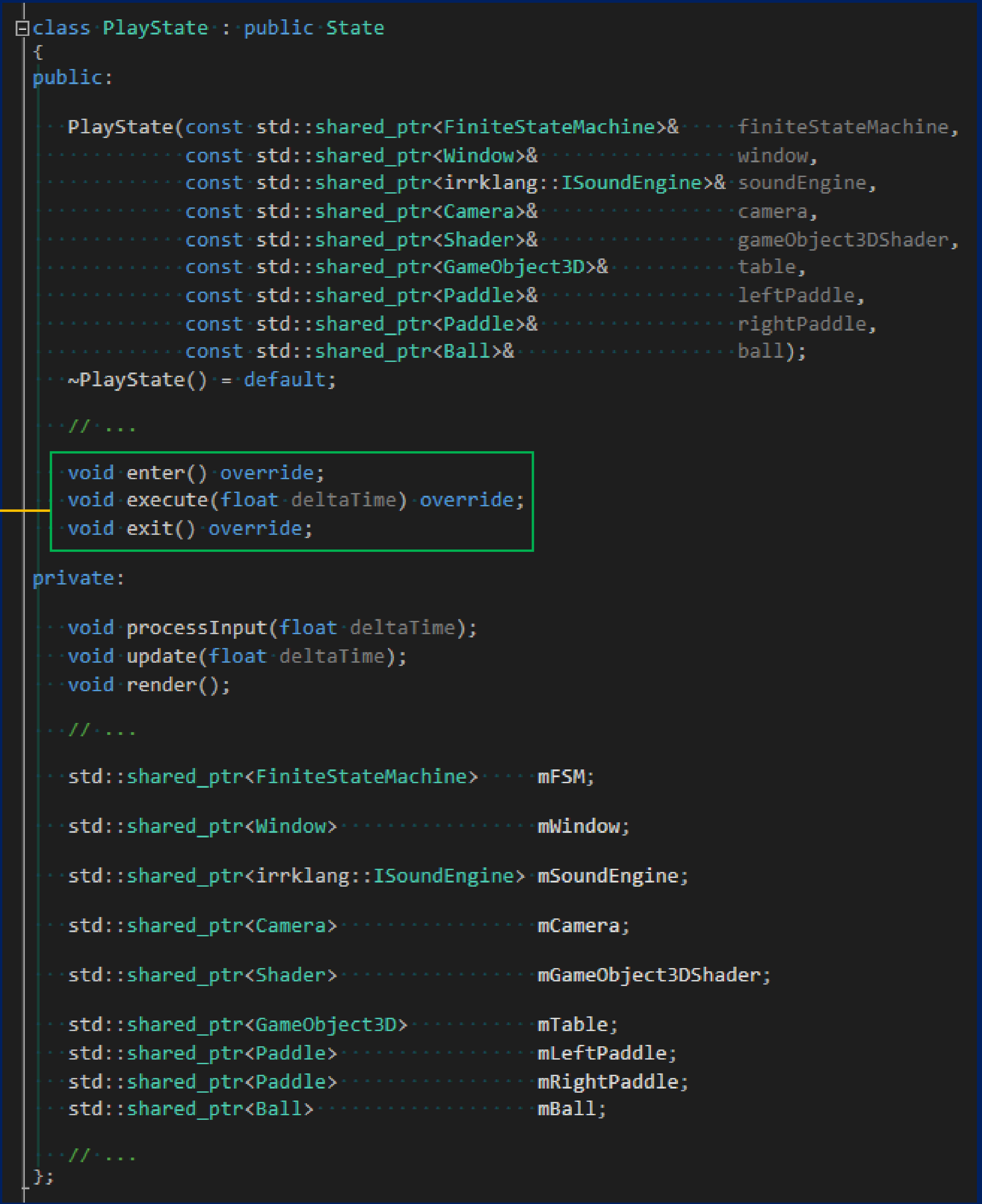
```cpp
if (mWindow->keyIsPressed(GLFW_KEY_P))
{
   mFSM->changeState("pause");
}
```



```cpp
if (mBallIsFalling)
{
   mBall->moveInFreeFall(deltaTime);

   if (mBall->getPosition().z < -45.0f)
   {
      if (mPointsScoredByLeftPaddle  == 3 ||
          mPointsScoredByRightPaddle == 3)
      {
         mFSM->changeState("win");
         return;
      }
      else
      {
         resetScene();
      }
   }
}
```



```cpp
void PlayState::enter()
{
   if (mFSM->getPreviousStateID() != "pause")
   {
      resetCamera();
      resetScene();
      mPointsScoredByLeftPaddle  = 0;
      mPointsScoredByRightPaddle = 0;
   }
}

void PlayState::execute(float deltaTime)
{
   processInput(deltaTime);

   if (mBallIsInPlay)
   {
      update(deltaTime);
   }

   render();
}

void PlayState::exit()
{
   if (mFSM->getCurrentStateID() != "pause")
   {
      resetScene();
   }
}
```

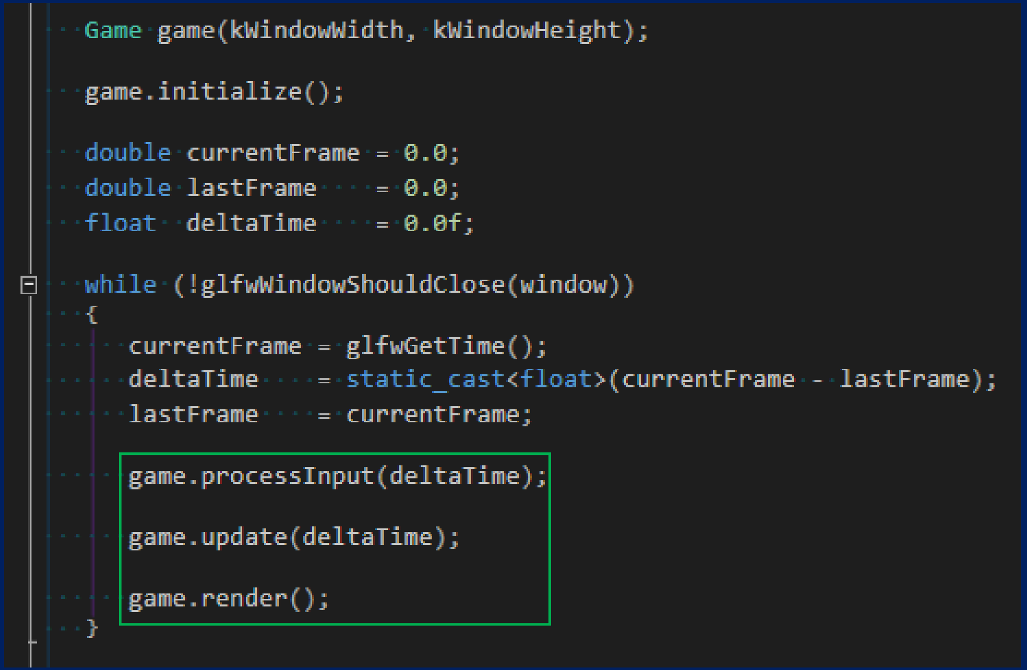


```cpp
class PlayState : public State
{
public:

   PlayState(const std::shared_ptr<FiniteStateMachine>&     finiteStateMachine,
             const std::shared_ptr<Window>&                 window,
             const std::shared_ptr<irrklang::ISoundEngine>& soundEngine,
             const std::shared_ptr<Camera>&                 camera,
             const std::shared_ptr<Shader>&                 gameObject3DShader,
             const std::shared_ptr<GameObject3D>&           table,
             const std::shared_ptr<Paddle>&                 leftPaddle,
             const std::shared_ptr<Paddle>&                 rightPaddle,
             const std::shared_ptr<Ball>&                   ball);
   ~PlayState() = default;

   // ...

   void enter() override;
   void execute(float deltaTime) override;
   void exit() override;

private:

   void processInput(float deltaTime);
   void update(float deltaTime);
   void render();

   // ...

   std::shared_ptr<FiniteStateMachine>     mFSM;

   std::shared_ptr<Window>                 mWindow;

   std::shared_ptr<irrklang::ISoundEngine> mSoundEngine;

   std::shared_ptr<Camera>                 mCamera;

   std::shared_ptr<Shader>                 mGameObject3DShader;

   std::shared_ptr<GameObject3D>           mTable;
   std::shared_ptr<Paddle>                 mLeftPaddle;
   std::shared_ptr<Paddle>                 mRightPaddle;
   std::shared_ptr<Ball>                   mBall;

   // ...
};
```

Design Patterns

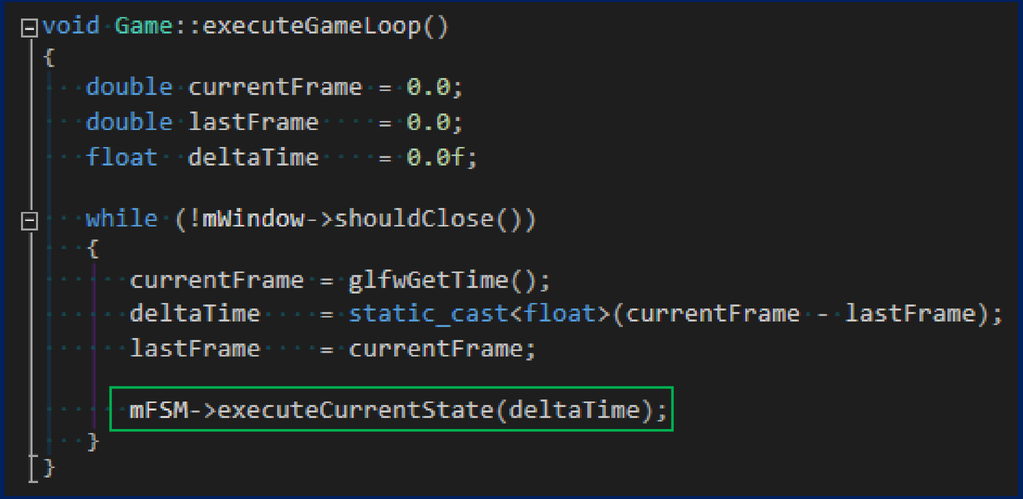
1) The state design pattern

Classic state management



```
Game game(kWindowWidth, kWindowHeight);

game.initialize();

double currentFrame  = 0.0;
double lastFrame     = 0.0;
float  deltaTime     = 0.0f;

while (!glfwWindowShouldClose(window))
{
    currentFrame  = glfwGetTime();
    deltaTime     = static_cast<float>(currentFrame - lastFrame);
    lastFrame     = currentFrame;

    game.processInput(deltaTime);

    game.update(deltaTime);

    game.render();
}
```

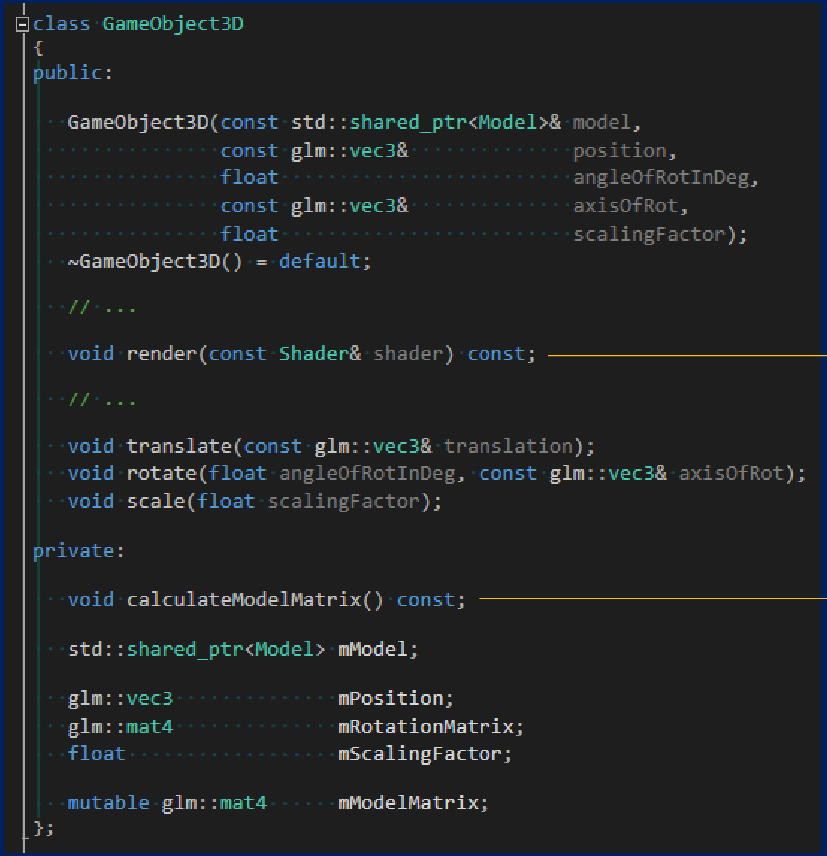
State design pattern



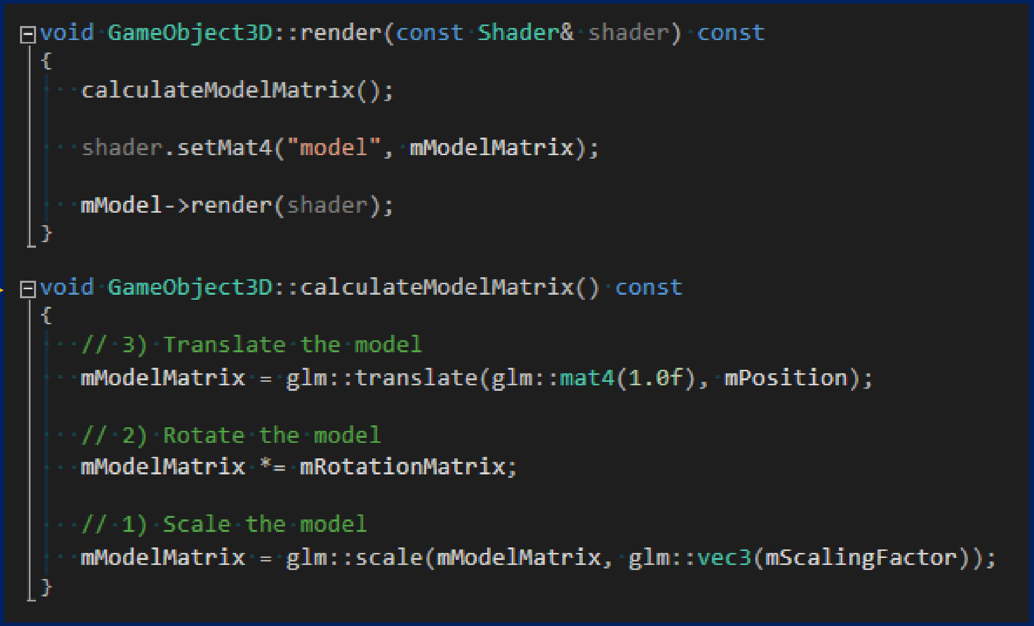
```
void Game::executeGameLoop()
{
    double currentFrame  = 0.0;
    double lastFrame     = 0.0;
    float  deltaTime     = 0.0f;

    while (!mWindow->shouldClose())
    {
        currentFrame  = glfwGetTime();
        deltaTime     = static_cast<float>(currentFrame - lastFrame);
        lastFrame     = currentFrame;

        mFSM->executeCurrentState(deltaTime);
    }
}
```

Things I'm Not Proud of

1) Premature optimizations... everywhere!



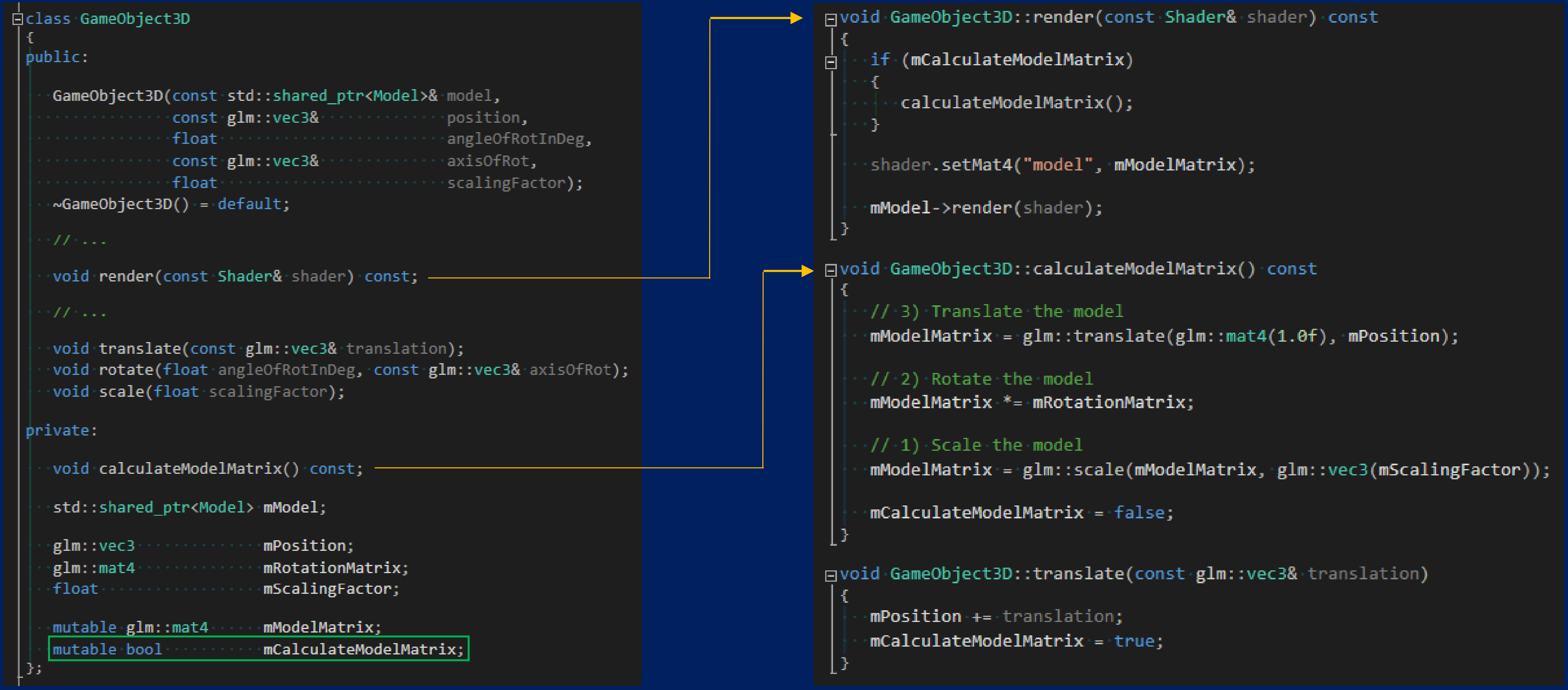
```cpp
class GameObject3D
{
public:

    GameObject3D(const std::shared_ptr<Model>& model,
                 const glm::vec3&              position,
                 float                         angleOfRotInDeg,
                 const glm::vec3&              axisOfRot,
                 float                         scalingFactor);
    ~GameObject3D() = default;

    // ...

    void render(const Shader& shader) const;

    // ...

    void translate(const glm::vec3& translation);
    void rotate(float angleOfRotInDeg, const glm::vec3& axisOfRot);
    void scale(float scalingFactor);

private:

    void calculateModelMatrix() const;

    std::shared_ptr<Model> mModel;

    glm::vec3          mPosition;
    glm::mat4          mRotationMatrix;
    float              mScalingFactor;

    mutable glm::mat4  mModelMatrix;
};
```



```cpp
void GameObject3D::render(const Shader& shader) const
{
    calculateModelMatrix();

    shader.setMat4("model", mModelMatrix);

    mModel->render(shader);
}

void GameObject3D::calculateModelMatrix() const
{
    // 3) Translate the model
    mModelMatrix = glm::translate(glm::mat4(1.0f), mPosition);

    // 2) Rotate the model
    mModelMatrix *= mRotationMatrix;

    // 1) Scale the model
    mModelMatrix = glm::scale(mModelMatrix, glm::vec3(mScalingFactor));
}
```

Things I'm Not Proud of

1) Premature optimizations... everywhere!

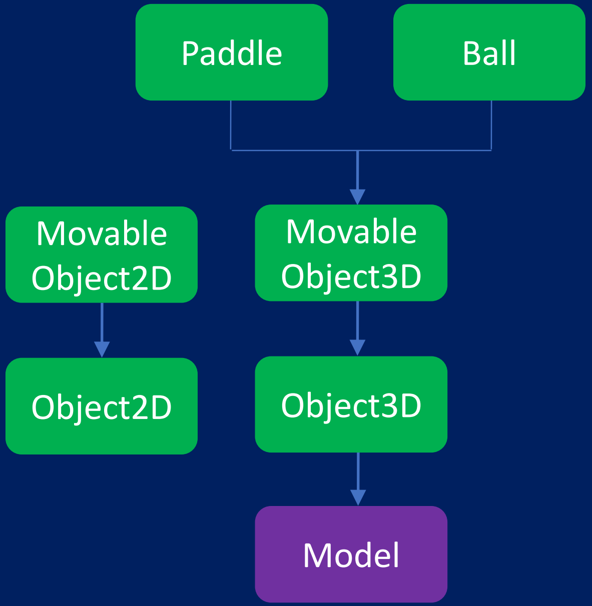


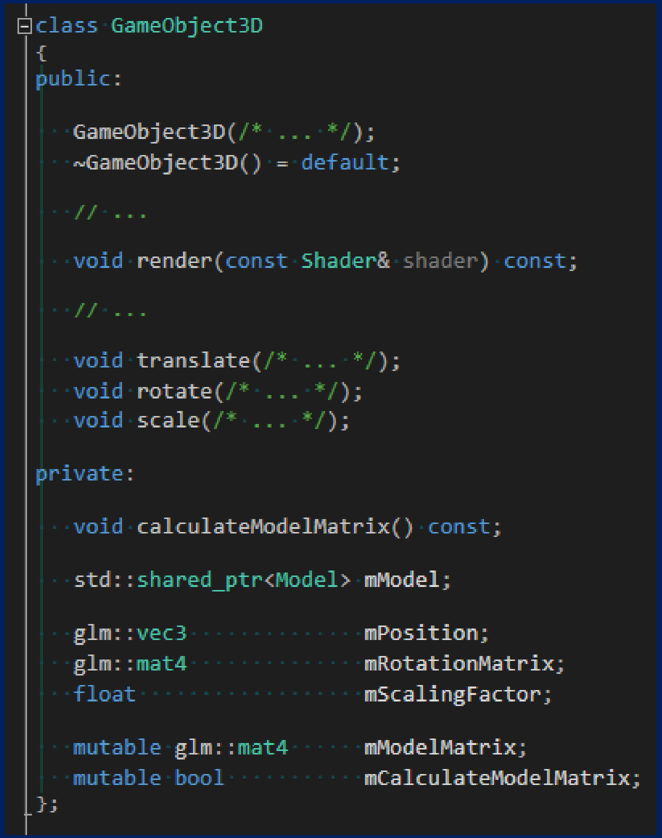
```cpp
class GameObject3D
{
public:

   GameObject3D(const std::shared_ptr<Model>& model,
                const glm::vec3&           position,
                float                      angleOfRotInDeg,
                const glm::vec3&           axisOfRot,
                float                      scalingFactor);
   ~GameObject3D() = default;

   // ...

   void render(const Shader& shader) const;

   // ...

   void translate(const glm::vec3& translation);
   void rotate(float angleOfRotInDeg, const glm::vec3& axisOfRot);
   void scale(float scalingFactor);

private:

   void calculateModelMatrix() const;

   std::shared_ptr<Model> mModel;

   glm::vec3            mPosition;
   glm::mat4            mRotationMatrix;
   float                mScalingFactor;

   mutable glm::mat4    mModelMatrix;
   mutable bool         mCalculateModelMatrix;
};
```

```cpp
void GameObject3D::render(const Shader& shader) const
{
   if (mCalculateModelMatrix)
   {
      calculateModelMatrix();
   }

   shader.setMat4("model", mModelMatrix);

   mModel->render(shader);
}

void GameObject3D::calculateModelMatrix() const
{
   // 3) Translate the model
   mModelMatrix = glm::translate(glm::mat4(1.0f), mPosition);

   // 2) Rotate the model
   mModelMatrix *= mRotationMatrix;

   // 1) Scale the model
   mModelMatrix = glm::scale(mModelMatrix, glm::vec3(mScalingFactor));

   mCalculateModelMatrix = false;
}

void GameObject3D::translate(const glm::vec3& translation)
{
   mPosition += translation;
   mCalculateModelMatrix = true;
}
```

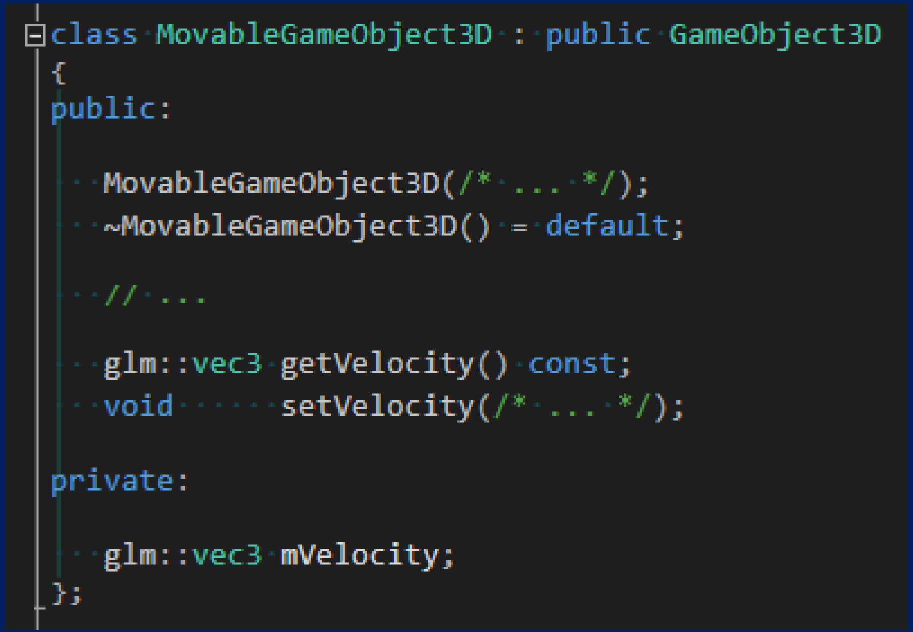
Things I'm Not Proud of

2) The use of inheritance to distinguish between static and movable objects





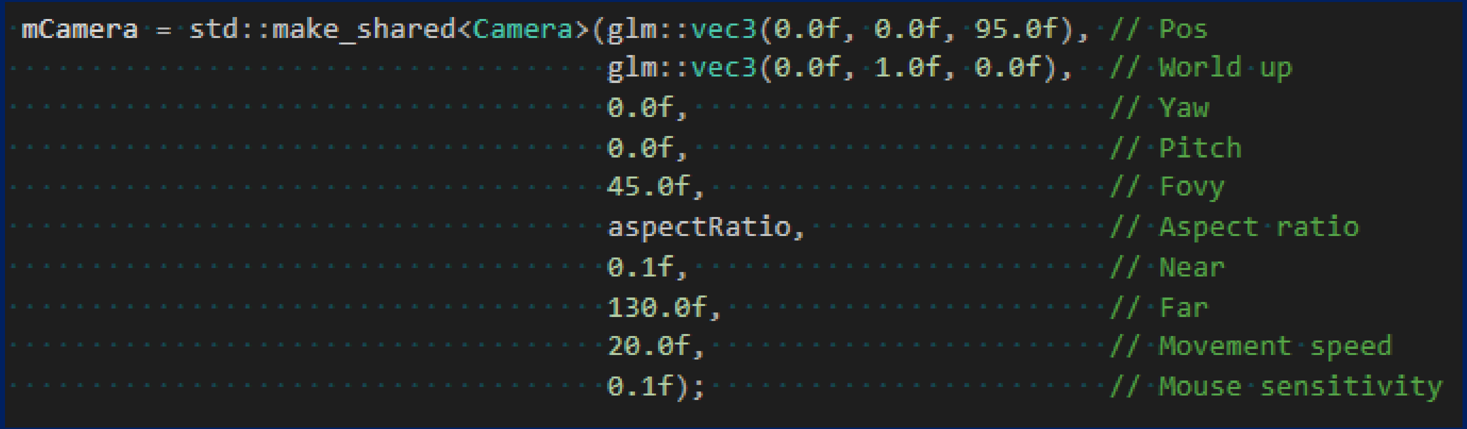
```cpp
class GameObject3D
{
public:

    GameObject3D(/* ... */);
    ~GameObject3D() = default;

    // ...

    void render(const Shader& shader) const;

    // ...

    void translate(/* ... */);
    void rotate(/* ... */);
    void scale(/* ... */);

private:

    void calculateModelMatrix() const;

    std::shared_ptr<Model> mModel;

    glm::vec3              mPosition;
    glm::mat4              mRotationMatrix;
    float                  mScalingFactor;

    mutable glm::mat4      mModelMatrix;
    mutable bool           mCalculateModelMatrix;
};
```



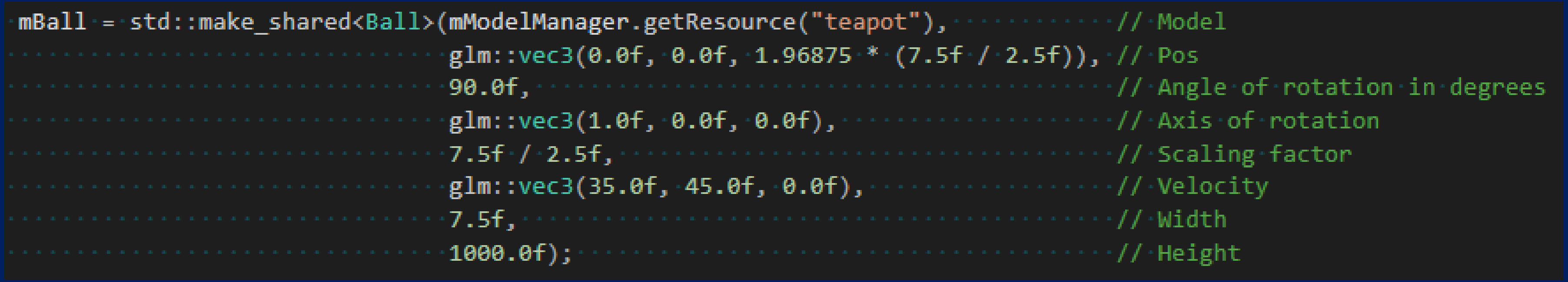
```cpp
class MovableGameObject3D : public GameObject3D
{
public:

    MovableGameObject3D(/* ... */);
    ~MovableGameObject3D() = default;

    // ...

    glm::vec3 getVelocity() const;
    void      setVelocity(/* ... */);

private:

    glm::vec3 mVelocity;
};
```

Things I'm Not Proud of

3) Hard coded constants in the game logic



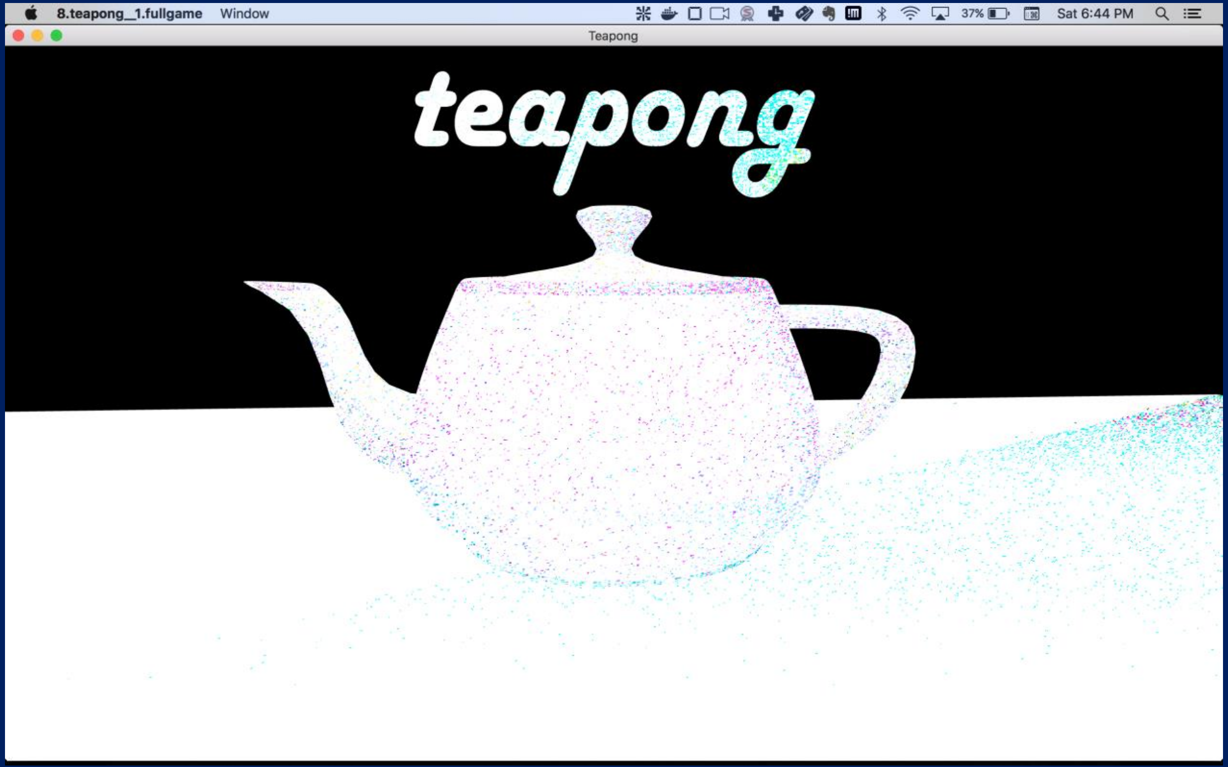
```cpp
mCamera = std::make_shared<Camera>(glm::vec3(0.0f, 0.0f, 95.0f), // Pos
                                   glm::vec3(0.0f, 1.0f, 0.0f),  // World up
                                   0.0f,                         // Yaw
                                   0.0f,                         // Pitch
                                   45.0f,                        // Fovy
                                   aspectRatio,                  // Aspect ratio
                                   0.1f,                         // Near
                                   130.0f,                       // Far
                                   20.0f,                        // Movement speed
                                   0.1f);                        // Mouse sensitivity
```



```cpp
mBall = std::make_shared<Ball>(mModelManager.getResource("teapot"),            // Model
                               glm::vec3(0.0f, 0.0f, 1.96875 * (7.5f / 2.5f)), // Pos
                               90.0f,                                          // Angle of rotation in degrees
                               glm::vec3(1.0f, 0.0f, 0.0f),                    // Axis of rotation
                               7.5f / 2.5f,                                    // Scaling factor
                               glm::vec3(35.0f, 45.0f, 0.0f),                  // Velocity
                               7.5f,                                           // Width
                               1000.0f);                                       // Height
```

The Subtlest Bug

"This looks as if we are reading garbage bytes and displaying them on the screen"



The Subtlest Bug

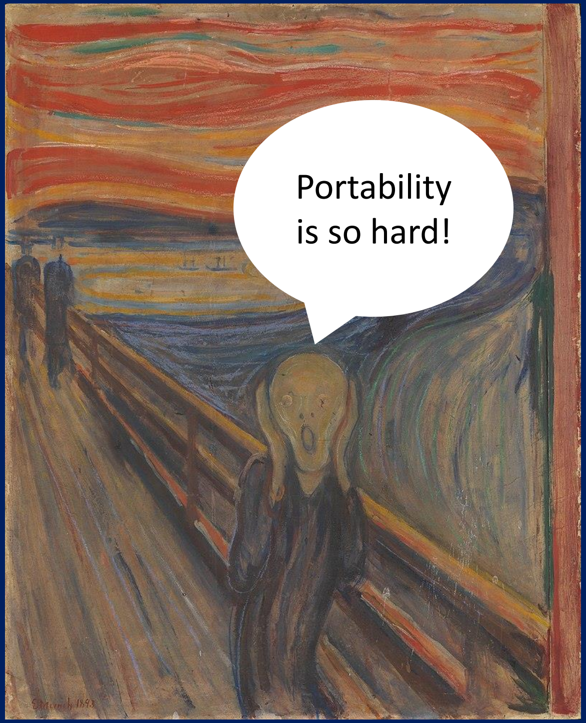
- GLenum glGetError(void);
  - GL_INVALID_VALUE
  - GL_INVALID_OPERATION
- OpenGL Reference Compiler (glslang)
  - The primary purpose of the reference compiler is to identify shader portability issues.
  - If glslang accepts a shader without errors, then all OpenGL implementations claiming to support the shader's language version should also accept the shader without errors.
  - glslangValidator game_object_3D.frag
- Could it be a problem related to the sampling of the textures?
  - Even with only material constants, the problem is still there!

The Subtlest Bug

Should I update the drivers?

What's your MacBook's OpenGL version?

Apple deprecated OpenGL and now uses Metal. Could that be the problem?



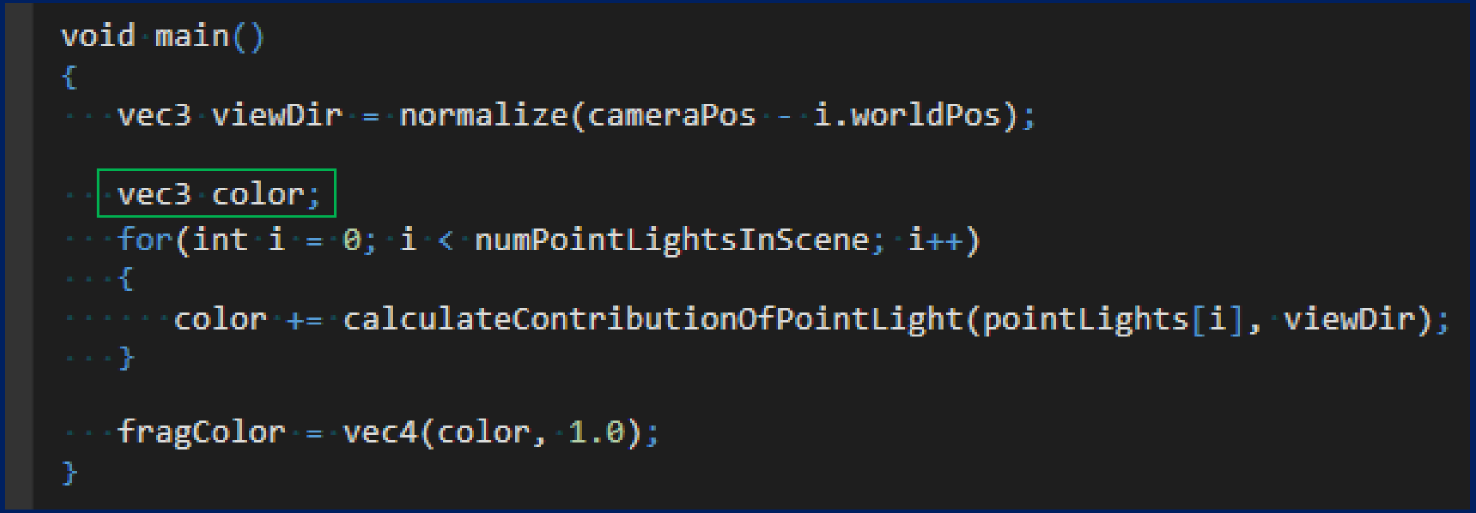
You compiled the project with Apple Clang. Maybe we should try GCC?

What's the brand of your GPU?

Should we rewrite the entire game in Assembly?

The Subtlest Bug

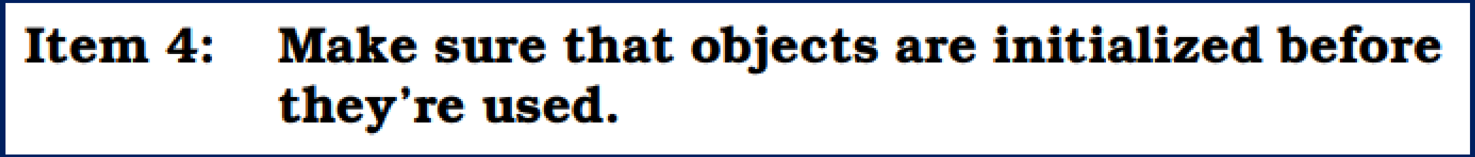
"This looks as if we are reading garbage bytes and displaying them on the screen"



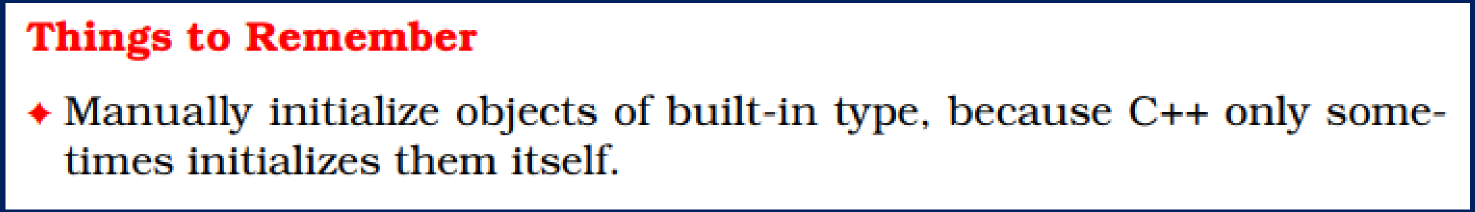
```
void main()
{
    vec3 viewDir = normalize(cameraPos - i.worldPos);

    vec3 color;
    for(int i = 0; i < numPointLightsInScene; i++)
    {
        color += calculateContributionOfPointLight(pointLights[i], viewDir);
    }

    fragColor = vec4(color, 1.0);
}
```

The Subtlest Bug

So the takeaway is: don't just read, practice!

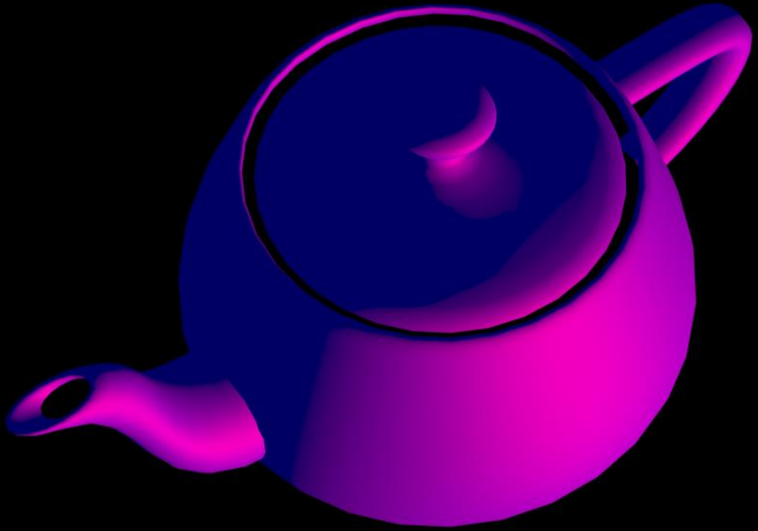


Item 4: Make sure that objects are initialized before they're used.



Things to Remember

- Manually initialize objects of built-in type, because C++ only some-times initializes them itself.



Thank you!