Shark Aquarium

Yuvraj Tilotia
CMP502 – Programming for Games
2200518@uad.ac.uk

Summary – DirectX 12 is a collection of APIs (Application Programming Interfaces) developed by Microsoft for use in Windows operating systems. It is primarily used to create and run high-performance graphics and multimedia applications, such as video games and video editing software. A scene in the context of DirectX 12 refers to a collection of 3D objects and other elements that make up a visual environment in a 3D application. These elements can include models of 3D objects, textures, lighting, and other visual effects. Scene rendering in DirectX 12 involves the process of taking all these elements and combining them to create the final image that is displayed to the user. The project uses DirectX 12 to build a box imitating an aquarium for Sharks. The model for the shark is made through Blender which is then called multiple times in the aquarium. The scene has lighting and movement animation.

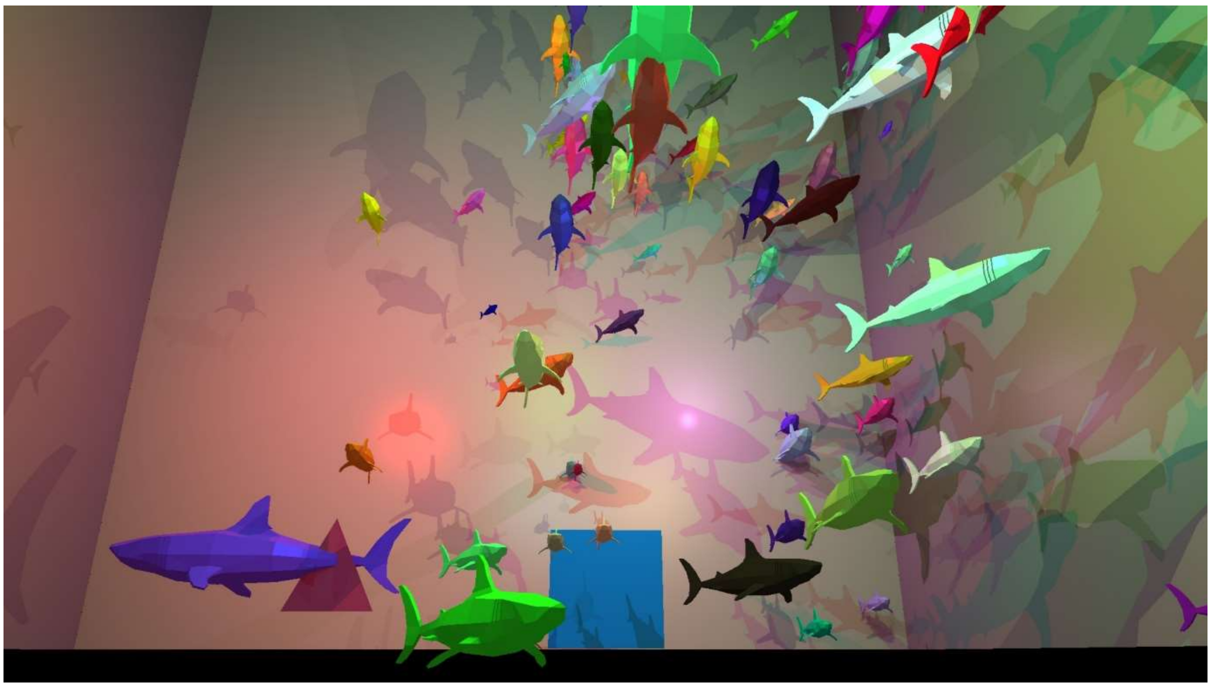


Fig 1 – Shark Aquarium

1. USER CONTROLS

   The scene initially puts the camera in the middle of the scene. Now the user can do all the following functions by continuously pressing the Right-Mouse Button and then pressing the following keys for their respective functions –

a) Movement ->
   W – Move Forward
   A – Move Leftward

S – Move Backward
D – Move Rightward
Q – Move Downward
E – Move Upward

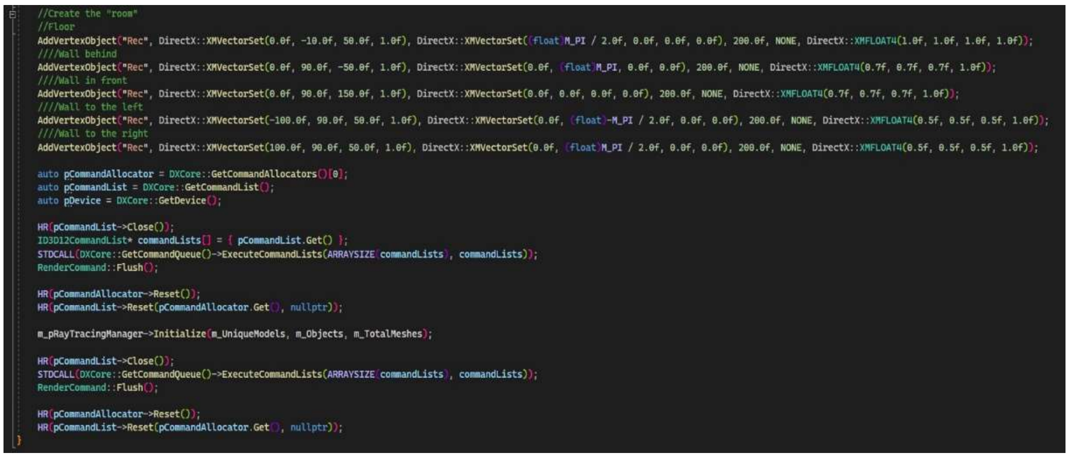
b) Camera Rotation ->
The camera has mouse-controlled movement.
Mouse Movement Forward – Camera Movement Upward
Mouse Movement Backward – Camera Movement Downward
Mouse Movement Leftward – Camera Movement Leftward
Mouse Movement Rightward – Camera Movement Rightward

c) Toggling Shadows ->
The shadows are initially active. You can disable the shadows for better performance by pressing key R.

2. TECHNICAL INSIGHT

a) Design Idea ->
The project idea was inspired by the basic lab tutorials for the module. The scene has multiple copies of the Shark object modelled in Blender. The aquarium was made to justify the multiple copies and provide an aesthetic element to the scene.

b) Process ->
The initial step was building the environment i.e., in this case, the aquarium. Then lighting was added to the walls to illuminate the whole box. The next step was to add functionality to the Camera element. Also, to import the Shark object, the "Model.cpp" file is used. Continuing the same path, Descriptor Heap files were built. The primary purpose of a descriptor heap is to encompass the bulk of memory allocation required for storing the descriptor specifications of object types that shaders reference for as large of a window of rendering as possible (ideally an entire frame of rendering or more). [1]

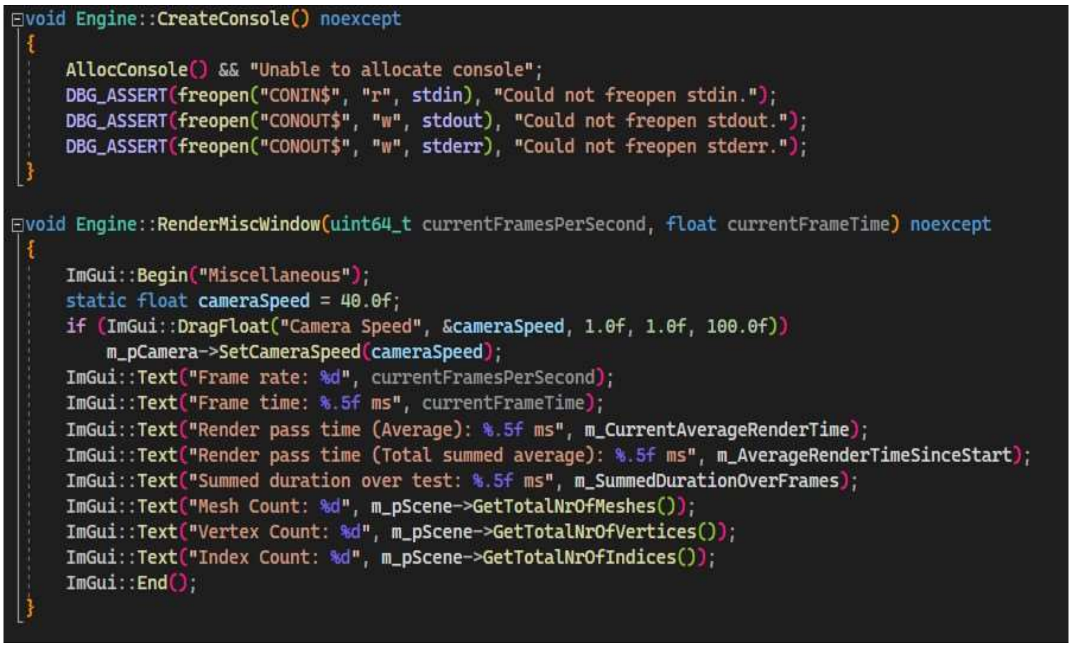


```cpp
    //Create the "room"
    //Floor
    AddVertexObject("Rec", DirectX::XMVectorSet(0.0f, -10.0f, 50.0f, 1.0f), DirectX::XMVectorSet((float)M_PI / 2.0f, 0.0f, 0.0f, 0.0f), 200.0f, NONE, DirectX::XMFLOAT4(1.0f, 1.0f, 1.0f, 1.0f));
    ////Wall behind
    AddVertexObject("Rec", DirectX::XMVectorSet(0.0f, 90.0f, -50.0f, 1.0f), DirectX::XMVectorSet(0.0f, (float)M_PI, 0.0f, 0.0f), 200.0f, NONE, DirectX::XMFLOAT4(0.7f, 0.7f, 0.7f, 1.0f));
    ////Wall in front
    AddVertexObject("Rec", DirectX::XMVectorSet(0.0f, 90.0f, 150.0f, 1.0f), DirectX::XMVectorSet(0.0f, 0.0f, 0.0f, 0.0f), 200.0f, NONE, DirectX::XMFLOAT4(0.7f, 0.7f, 0.7f, 1.0f));
    ////Wall to the left
    AddVertexObject("Rec", DirectX::XMVectorSet(-100.0f, 90.0f, 50.0f, 1.0f), DirectX::XMVectorSet(0.0f, (float)-M_PI / 2.0f, 0.0f, 0.0f), 200.0f, NONE, DirectX::XMFLOAT4(0.5f, 0.5f, 0.5f, 1.0f));
    ////Wall to the right
    AddVertexObject("Rec", DirectX::XMVectorSet(100.0f, 90.0f, 50.0f, 1.0f), DirectX::XMVectorSet(0.0f, (float)M_PI / 2.0f, 0.0f, 0.0f), 200.0f, NONE, DirectX::XMFLOAT4(0.5f, 0.5f, 0.5f, 1.0f));

    auto pCommandAllocator = DXCore::GetCommandAllocators()[0];
    auto pCommandList = DXCore::GetCommandList();
    auto pDevice = DXCore::GetDevice();

    HR(pCommandList->Close());
    ID3D12CommandList* commandLists[] = { pCommandList.Get() };
    STDCALL(DXCore::GetCommandQueue()->ExecuteCommandLists(ARRAYSIZE(commandLists), commandLists));
    RenderCommand::Flush();

    HR(pCommandAllocator->Reset());
    HR(pCommandList->Reset(pCommandAllocator.Get(), nullptr));

    m_pRayTracingManager->Initialize(m_UniqueModels, m_Objects, m_TotalMeshes);

    HR(pCommandList->Close());
    STDCALL(DXCore::GetCommandQueue()->ExecuteCommandLists(ARRAYSIZE(commandLists), commandLists));
    RenderCommand::Flush();

    HR(pCommandAllocator->Reset());
    HR(pCommandList->Reset(pCommandAllocator.Get(), nullptr));
}
```

Fig 2 – Aquarium Creation Code Snippet

The DXCore file is an adaptor enumeration API for DirectX devices. It enables the exposure of new device types to user mode, such as MCDM (Microsoft Compute Driver Model), for use with Direct3D 12, DirectML, and Windows Machine Learning. [2] To manage the rendering in the 3D pipeline, we use the "imguimanager.cpp" file. IMGUI is designed to enable fast iterations and to empower programmers to create content creation tools and visualization / debug tools (as opposed to UI for the average end-user). [3]

To implement ray tracing for shadows we build the "RayTracingManager.cpp" file. [4] Now, to control the camera movement, we build the keyboard and mouse input read files, and link the keypress to specific functions as listed in the previous section. The mesh and memory manager are now integrated into the scene. The final step is to build the render and the engine.



```
void Engine::CreateConsole() noexcept
{
    AllocConsole() && "Unable to allocate console";
    DBG_ASSERT(freopen("CONIN$", "r", stdin), "Could not freopen stdin.");
    DBG_ASSERT(freopen("CONOUT$", "w", stdout), "Could not freopen stdout.");
    DBG_ASSERT(freopen("CONOUT$", "w", stderr), "Could not freopen stderr.");
}

void Engine::RenderMiscWindow(uint64_t currentFramesPerSecond, float currentFrameTime) noexcept
{
    ImGui::Begin("Miscellaneous");
    static float cameraSpeed = 40.0f;
    if (ImGui::DragFloat("Camera Speed", &cameraSpeed, 1.0f, 1.0f, 100.0f))
        m_pCamera->SetCameraSpeed(cameraSpeed);
    ImGui::Text("Frame rate: %d", currentFramesPerSecond);
    ImGui::Text("Frame time: %.5f ms", currentFrameTime);
    ImGui::Text("Render pass time (Average): %.5f ms", m_CurrentAverageRenderTime);
    ImGui::Text("Render pass time (Total summed average): %.5f ms", m_AverageRenderTimeSinceStart);
    ImGui::Text("Summed duration over test: %.5f ms", m_SummedDurationOverFrames);
    ImGui::Text("Mesh Count: %d", m_pScene->GetTotalNrOfMeshes());
    ImGui::Text("Vertex Count: %d", m_pScene->GetTotalNrOfVertices());
    ImGui::Text("Index Count: %d", m_pScene->GetTotalNrOfIndices());
    ImGui::End();
}
```

Fig 3 – Engine Code Snippet

The project development technique used was Critical Path Method. The methodology entails breaking down the whole project into smaller and simpler modules. [5] This makes the project streamlined and helps in staying on track.

3. EVALUATION

The scene has a lot of potentials to build upon. The aquarium build was on point. The lighting and ray-tracing shadows work perfectly. When we turn off the ray tracing shadows, the performance improvement is significant and noticeable. The model import was easy and work efficiently. The Descriptor Heap, in addition to, Memory Manager helped immensely in delivering smooth working. The IMGUI worked well with the camera. There was no delay in model loading whenever there is a change in frame. The Mouse and Keyboard integration with camera movement is very smooth. There are no delays in response. The difficult part was the main Engine and Render amalgamation. This proved tricky to manage whenever there was a change in development files. The overall development of the scene took almost 34 hours including all research and viewing of tutorials online to help with any modules of the project.

There were a few areas in which the project was lacking. Even after trying countless times, we were not able to incorporate the whole number of sharks within the aquarium boundaries.

Some model calls placed the sharks outside the aquarium boundaries. The next point which might seem like a drawback but was done intentionally was the shark object model. As the number of calls the object will be made to in the scene was high, we decided to make it without any animation to counter-balance performance and framerate. Also, the lack of different object models was deliberate. The reason behind this too is the same as before i.e., performance efficiency is the priority.

4. FUTURE IMPROVEMENTS

There are many points on which the model can be improved. Some of them are as follows

a) Light Attenuation ->
This feature would allow for lights to have a drop-off. This would make point lights more realistic.

b) Animations ->
The shark object could be animated to swim around.

c) Water ->
Currently, there is no water in the aquarium. This could be improved through the addition of water and water textures around the animated sharks.

d) Terrain under the water line ->
Currently, there is no bottom terrain in the aquarium as it exists in the real world. This could be built or imported as an interactive object.

e) Shadows ->
Shadows add a lot of realism to the scene as they greatly affect lighting. The performance impact would be moderate..

5. REFERENCES

[1] Descriptor Heaps Overview (2021). Available at:

https://learn.microsoft.com/en-us/windows/win32/direct3d12/descriptor-heaps-overview (Accessed: 17 December 2022).

[2] DXCore Overview (2021). Available at:

https://learn.microsoft.com/en-us/windows/win32/dxcore/dxcore-enum-adapters (Accessed: 19 December 2022)

[3] IMGUI (Ongoing). Available at:

https://github.com/ocornut/imgui (Accessed: 11 December 2022)

[4] Ray Tracing in One Weekend (2020). Available at:

https://raytracing.github.io/books/RayTracingInOneWeekend.html (Accessed: 29 December 2022)

[5] Project Management Techniques (2021). Available at:

https://monday.com/blog/project-management/8-project-management-techniques-every-project-manager-should-know/ (Accessed: 8 December 2022).

a) Tutorials ->

   C++ 3D DirectX Programming (2020). Available at:

   https://www.youtube.com/playlist?list=PLqCJpWy5Fohd3S7ICFXwUomYW0Wv67pDD (Accessed: Multiple dates).

   C++ DirectX 11 Engine Tutorials (2019). Available at:

   https://www.youtube.com/playlist?list=PLcacUGyBsOIBlGyQQWzp6D1Xn6ZENx9Y2 (Accessed: Multiple dates).