Write Up

Assignment 08 - GAMES6320

[Download Game]

Controls: WASD - Move Mesh, Arrow Keys - Move Camera

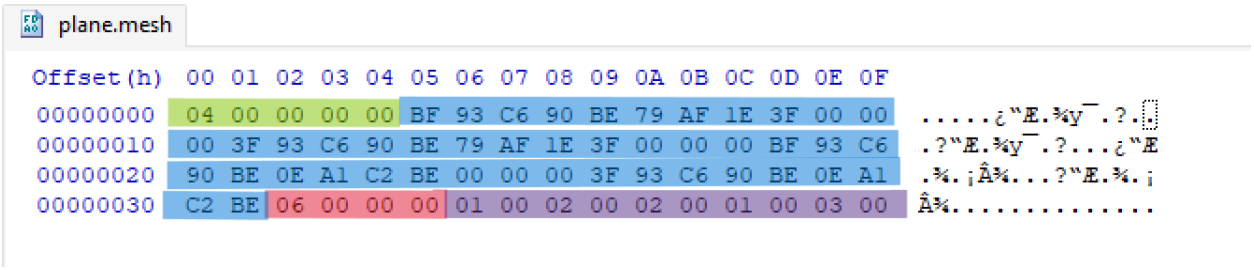
During Assignment 08, I learned a lot about the advantages of having binary files over human-readable files, especially for processing during run-time. Although I initially struggled to correctly write out binary data, I figured out the nuances and steps for the game engine to improve performance and speed through the generation and reading of binary files.

Default State Of Game

Binary Geometry File

The image below is an example of a binary geometry file generated by the MeshBuilder. This binary file represents a plane with four vertices and six indices. Although it was initially hard to read, looking back at the order of the four things that I wrote out helped me visualize the binary file.

The order that I chose was - the number of vertices, vertex data, number of indices, and index data. I went with this order because the number of vertices is needed first to allocate memory for the vertex data. Once the memory has been allocated, then the vertex positions can be loaded. Since the indices and index data also follow the same logic, this order made the most sense thinking from a computer's perspective.



One main advantage of a binary file format is that they are much more compact and cost less memory when compared to human-readable files. This is because they store data in a format that is directly understood by the system without extra characters like whitespace, commas, etc.

This, in turn, makes binary files much faster to read because there is no need to parse any data. Hence, it can drastically improve the performance of our program.

However, it is obviously very hard for a human to read binary files, and this is where human-readable files play a crucial role. To store the data in source control, we would want a human-readable file that is easy to understand and debug when something goes wrong. Here, we care more about readability than performance.

On the other hand, binary formats are the best option at run-time because we want to improve performance and memory usage. During run-time, we don't

need a human to read the files. The files would only be read by the computer, so we would want to make it as easy as possible for the system to read and process run-time files.

In terms of binary files for different platforms, I believe there should be different binary geometry files for Direct3D and OpenGL. This is to mainly account for the differences in coordinate systems and index winding order between the two platforms. As we know, Direct3D uses a left-handed (clockwise) order as opposed to OpenGL, which uses a right-handed (anti-clockwise) order. Hence, this affects the order in which index data is sorted and interpreted by the system. As I said before, we would want to make it as easy as possible for the computer to read these files at run-time, so I think it's better to provide it with different binary files depending on the platform, rather than letting it handle the winding order on its own.

Extracting Binary Data

```
{
    Mesh* mesh = nullptr;
    eae6320::Platform::sDataFromFile dataFromFile;
    cResult result = eae6320::Platform::LoadBinaryFile(pFilePath.c_str(), dataFromFile);
    if (result != eae6320::Results::Success)
    {
        EAE6320_ASSERTF(false, "Can't load binary file.");
        eae6320::Logging::OutputMessage("Can't load binary file.");
        return mesh;
    }

    // Initializing offsets
    auto currentOffset = reinterpret_cast<uintptr_t>(dataFromFile.data);
    const auto finalOffset = currentOffset + dataFromFile.size;

    // 1. Extracting the number of vertices (uint16_t)
    uint16_t vertexCount;
    memcpy(&vertexCount, reinterpret_cast<void*>(currentOffset), sizeof(vertexCount));
    currentOffset += sizeof(vertexCount); // Moving offset to the vertex data

    // 2. Extracting the vertex array (VertexFormats::sVertex_mesh)
    const auto* const vertexArray = reinterpret_cast<VertexFormats::sVertex_mesh*>(currentOffset);
    currentOffset += sizeof(VertexFormats::sVertex_mesh) * vertexCount; // Moving offset to number of indices

    // 3. Extracting the number of indices (uint16_t)
    uint16_t indexCount;
    memcpy(&indexCount, reinterpret_cast<void*>(currentOffset), sizeof(indexCount));
    currentOffset += sizeof(indexCount); // Moving offset to the index data

    // 4. Extracting the index array (uint16_t)
    const auto* const indexArray = reinterpret_cast<uint16_t*>(currentOffset);
    currentOffset += sizeof(uint16_t) * indexCount; // Now at the end of the binary chunk

    // At this point, we have the vertex and index data, so we can initialize the mesh
    GeometryData MeshInitData;
    MeshInitData.numVertices = vertexCount;
    MeshInitData.vertexData = new VertexFormats::sVertex_mesh[vertexCount];
    memcpy(MeshInitData.vertexData, vertexArray, sizeof(VertexFormats::sVertex_mesh) * vertexCount);
```

To extract the binary data at run-time, I followed the assignment instructions and kept track of the addresses by creating an offset and updating it after

extracting each of the four pieces of data. At first, I thought I needed to call the LoadBinaryFile() four times for each piece of data. But then I realized that the LoadBinaryFile() function loads the entire file as a single chunk of memory.

Comparison of Size and Speed

	Size on Disk	Speed of Loading and Extracting Data
Human - Readable Version	140 KB	3450 microseconds
Binary Version	48 KB	76 microseconds

Given above are the sizes and speeds of extracting data for a helix mesh generated by Maya. As you can see, the memory usage and speed of processing of the binary version at run-time is drastically better than that of a human-readable file.

On disk, the human-readable file for a helix mesh occupies almost 3 times the space of a binary version. When the scope of a game piles up, this can result in much larger storage space being consumed by the game build.

Similarly, the speed of extracting data from a binary file is orders of magnitude faster than loading and extracting data from a human-readable version. This is because the data is already in a format that the system understands, therefore leading to much greater performance speeds.

These advantages of binary files are really crucial, especially in game engines. The number of art assets and meshes in a game can be huge, depending on the scope and design. Hence, it is vital to use binary files so that performance can be maintained at run-time.

Download Game