MeshData to Binary Conversion

Walter Kusnezow

August 2025

1 Content

2 Byte Array Structure

To save multiple arrays of different types into a single binary, a kind of header block is needed, to mark the byte offset, where data starts and ends. When writing Data into the Byte Array buffer, the size of the buffer must be pre calculated. Once we set the size of the Byte buffer, the pointer will stay valid.

$$Vertex_{bytes} = |VertexBuffer| \cdot sizeof(FVector)$$
$$Index_{bytes} = |IndexBuffer| \cdot sizeof(int32)$$
$$Normal_{bytes} = |NormalBuffer| \cdot sizeof(FVector)$$
$$UV_{bytes} = |UVBuffer| \cdot sizeof(FVector2D)$$

Since we are saving 4 arrays of data, the size of the header block is:

$$Header_{bytes} = 4 \cdot sizeof(int32)$$

The Total byteresult , which needs to be reserved in our byte array $TArray < unit8 >$ is:

$$Total_{bytes} = Header_{bytes} + Vertex_{bytes} + Index_{bytes} + Normal_{bytes} + UV_{bytes}$$

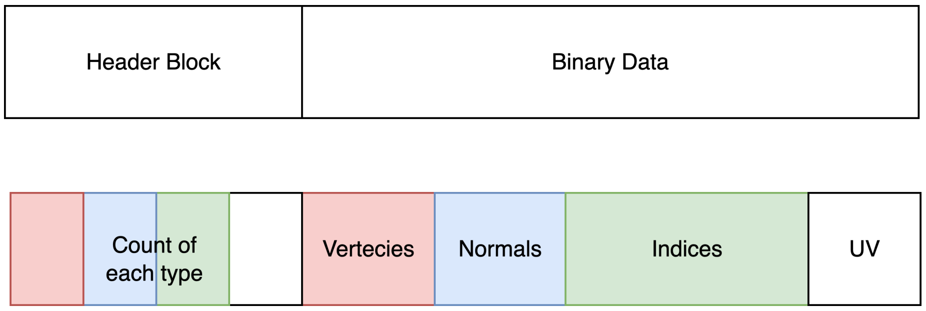

Figure 1: Binary Structure

3 Saving the Data

Once the Byte Array size is set, we can retrive a Data pointer to the array of type *uint*8∗. Each array size of the data to fill in, is copied inside the byte array through the pointer. The Pointer is increased by the bytes copied after each operation. The Pointer is passed by reference to keep the Pointer arithmetic.

```cpp
void StorageInterfaceMeshData::writeMeshData(...){
    ...
    int previousSize = Bytes.Num(); //if the data is
        appended to another buffer, the pervious size is of
        interest
    Bytes.SetNumUninitialized(previousSize + completeSize);
    uint8* Ptr = Bytes.GetData(); //get data pointer to work
         with (starts at front, needs to be at offset)
    Ptr += previousSize;

    writeInfoData(...)

    FMemory::Memcpy(Ptr, (uint8 *)Vertecies.GetData(),
        verteciesByteSize); //copy casted data
    Ptr += verteciesByteSize; //increase pointer adress

    FMemory::Memcpy(Ptr, (uint8 *)Normals.GetData(),
        normalsByteSize); //copy
    Ptr += normalsByteSize; //increase pointer adress

    FMemory::Memcpy(Ptr, (uint8 *)UV0.GetData(), uv0ByteSize
        ); //copy
    Ptr += uv0ByteSize; //increase pointer adress

    FMemory::Memcpy(Ptr, (uint8 *)Triangles.GetData(),
        trianglesByteSize); //copy

    ...
}
```

```



void StorageInterfaceMeshData::writeInfoData(
    int32 vertexCount,
    int32 normalCount,
    int32 uvCount,
    int32 triangleCount,
    uint8*& Ptr
){
    TArray<int32> info = {
        vertexCount,
        normalCount,
        uvCount,
        triangleCount
    };
    for (int i = 0; i < info.Num(); i++){
        int32 *infoCurrent = &info[i];
        /*
        FMemory::Memcpy (
            void* Dest,
            const void* Src,
            SIZE_T Count
        )
        */
        FMemory::Memcpy(Ptr, infoCurrent, sizeof(int32)); //
            copy
        Ptr += sizeof(int32); //increase pointer adress
    }
}
```

4 Multiple MeshData in One Binary

As visible in the above Method, the previous size of the Binary Buffer is kept. It is possible to concatenate Multiple MeshData Buffers in one Binary / Byte Buffer. The reading and write method can be called recursively.

5 Reading from Byte Array

The Data can be read from the Byte Array just like writing the buffer. All the sizes are extracted from the front bytes: $4 \cdot sizeof(int32)$. Once the array sizes are extracted from the front, the Pointer is moved to the start of the first array.

Once all sizes of the different arrays are extracted, the num is set for the arrays, so the Memcopy operation is valid. After Each Copy, the pointer is increased by the given byte count, for the type to copy and count of array, which was extracted: $Ptr+ = sizeof(T) \cdot count$

```

//can be called recursively, endReached flag will be marked
void StorageInterfaceMeshData::LoadIntoMeshBuffers(
    TArray<uint8> &Bytes, //buffer size is increased after
        append!
    uint8*& Ptr, //is increased after append, must be at
        correct offset starting with header bytes!
    TArray<FVector> &Vertecies,
    TArray<FVector> &Normals,
    TArray<FVector2D> &UV0,
    TArray<int32> &Triangles,
    bool &endReached
){
    //load info data
    int32 vertexCount = 0;
    int32 normalCount = 0;
    int32 uvCount = 0;
    int32 triangleCount = 0;

    loadInfoData(Ptr, vertexCount, normalCount, uvCount,
        triangleCount);

    //SET SIZE OF BUFFERS
    Vertecies.SetNumUninitialized(vertexCount);
    Normals.SetNumUninitialized(normalCount);
    UV0.SetNumUninitialized(uvCount);
    Triangles.SetNumUninitialized(triangleCount);

    //copy mesh data based on infoData (make void* its
        cooler)
    int infoBytesSize = getInfoBytesSize();
    int verteciesByteSize = getVertexBytesSize(vertexCount);
```

```cpp
    int normalsBytesSize = getNormalsBytesSize(normalCount);
    int uvBytesSize = getUVBytesSize(uvCount);
    int trianglesByteSize = getTrianglesBytesSize(triangleCount);

    //verify size
    int totalSize =
        infoBytesSize +
        verteciesByteSize +
        normalsBytesSize +
        uvBytesSize +
        trianglesByteSize;

    //if bytes exceeded, mesh data is broken.
    //if(totalSize != sizeof(uint8) * Bytes.Num()){
    if(totalSize > sizeof(uint8) * Bytes.Num()){ //check if reaching out of bounds with Pointer.
        DebugHelper::logMessage("Storage Interface MeshData Exceeded byte size");
        endReached = true;
        return;
    }
    if(totalSize == sizeof(uint8) * Bytes.Num()){
        endReached = true;
    }

    /*
    FMemory::Memcpy (
        void* Dest,
        const void* Src,
        SIZE_T Count
    )
    */

    FMemory::Memcpy((uint8 *)Vertecies.GetData(), Ptr, verteciesByteSize); // copy casted data
    Ptr += verteciesByteSize;                                                  // increase pointer adress


    FMemory::Memcpy((uint8 *)Normals.GetData(), Ptr, normalsBytesSize); // copy casted data
    Ptr += normalsBytesSize;                                                  // increase pointer adress

    FMemory::Memcpy((uint8 *)UV0.GetData(), Ptr, uvBytesSize); // copy casted data
    Ptr += uvBytesSize;
```

```
                                                              // increase
        pointer adress

    FMemory::Memcpy((uint8 *)Triangles.GetData(), Ptr,
        trianglesByteSize); // copy casted data
    Ptr += trianglesByteSize; // increase pointer adress (if
         next data loading needed)

}




void StorageInterfaceMeshData::loadInfoData(
    uint8 *& Ptr, //Ptr already at given offset, passed by
        reference to keep adress
    int32 &vertexCount,
    int32 &normalCount,
    int32 &uvCount,
    int32 &triangleCount
){
    TArray<int32 *> infoData = {
        &vertexCount,
        &normalCount,
        &uvCount,
        &triangleCount
    };

    /*
    FMemory::Memcpy (
        void* Dest,
        const void* Src,
        SIZE_T Count
    )
    */
    for (int i = 0; i < infoData.Num(); i++){
        FMemory::Memcpy(infoData[i], Ptr, sizeof(int32));
        Ptr += sizeof(int32);
    }
}
```

6 Code As graphic

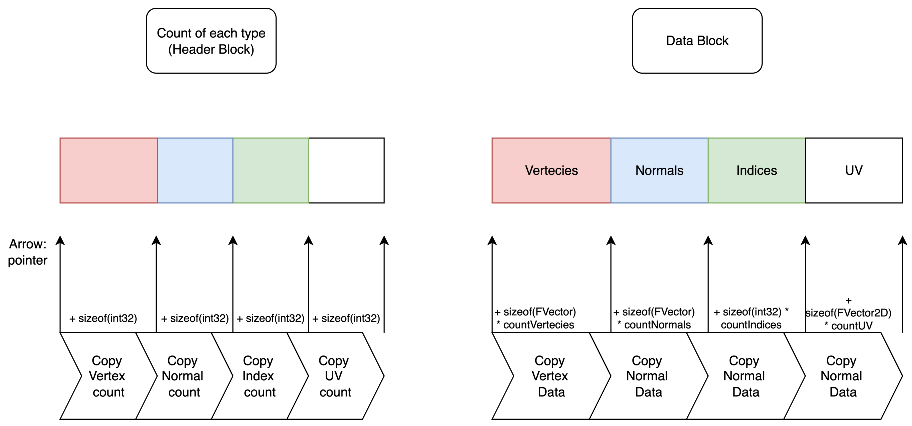

Figure 2: The Pointer Movement and Copy Operation visualized