# *Game Graphic Programming*

Kyung Hee University

Software Convergence

Prof. Daeho Lee

# MeshGeometry (1)

```
// struct MeshGeometry (1/2)
struct MeshGeometry {
    std::string Name;
    Microsoft::WRL::ComPtr<ID3DBlob> VertexBufferCPU = nullptr;
    Microsoft::WRL::ComPtr<ID3DBlob> IndexBufferCPU  = nullptr;
    Microsoft::WRL::ComPtr<ID3D12Resource> VertexBufferGPU = nullptr;
    Microsoft::WRL::ComPtr<ID3D12Resource> IndexBufferGPU = nullptr;
    Microsoft::WRL::ComPtr<ID3D12Resource> VertexBufferUploader = nullptr;
    Microsoft::WRL::ComPtr<ID3D12Resource> IndexBufferUploader = nullptr;

    UINT VertexByteStride = 0;
    UINT VertexBufferByteSize = 0;
    DXGI_FORMAT IndexFormat = DXGI_FORMAT_R16_UINT;
    UINT IndexBufferByteSize = 0;
```

# MeshGeometry (2)

```
// struct MeshGeometry (2/2)

    std::unordered_map<std::string, SubmeshGeometry> DrawArgs;

    D3D12_VERTEX_BUFFER_VIEW VertexBufferView() const {
        return {VertexBufferGPU->GetGPUVirtualAddress(), VertexBufferByteSize,
            VertexByteStride };
    }

    D3D12_INDEX_BUFFER_VIEW IndexBufferView() const {
        return { IndexBufferGPU->GetGPUVirtualAddress(), IndexBufferByteSize,
            IndexFormat };
    }

    void DisposeUploaders() {
        VertexBufferUploader = nullptr;      IndexBufferUploader = nullptr;
    }
};
```

# MeshGeometry Objects (1)

```
class Applicaton : public D3DApp {
// …
    std::unordered_map<std::string, std::unique_ptr<MeshGeometry>> mGeometries;

    RenderItem* mApplicationRitem = nullptr;
    std::vector<std::unique_ptr<RenderItem>> mAllRitems;
    std::vector<RenderItem*> mRitems;  // for rendering of mAllRitems
// …
};

struct RenderItem{
// …
    MeshGeometry* Geo = nullptr;

    XMFLOAT4X4 World = MathHelper::Identity4x4();
    UINT ObjCBIndex = -1;
// …
};
```

# MeshGeometry Objects (2)

```
struct SubmeshGeometry{
    UINT IndexCount = 0;
    UINT StartIndexLocation = 0;
    INT BaseVertexLocation = 0;
    DirectX::BoundingBox Bounds; // for collision detection
};

void Applicaton::BuildGeometry() {
// …
    mGeometries["Name0"] = std::make_unique<MeshGeometry>();
    // Set CPU, GPU for vertex and index buffers of mGeometries["Name"]
    mGeometries["Name0"]->DrawArgs["Item0"]; // = SubmeshGeometry object
// …
};
```

# `RenderItem` Objects (1)

```
void Application::BuildRenderItems() {
// …
   int i = 0;
   mAllRitems.push_back(std::make_unique<RenderItem>());
   mAllRitems[i]->Geo = mGeometries["Name0"];
   mAllRitems[i]->StartIndexLocation
      = mAllRitems[i]->Geo->DrawArgs["box"].StartIndexLocation;
   mAllRitems[i]->BaseVertexLocation
      = mAllRitems[i]->Geo->DrawArgs["box"].BaseVertexLocation;

   mAllRitems[i]->ObjCBIndex; // = index, for root signature
   mAllRitems[i]->World;      // = world matrix
// …
   for(auto& e : mAllRitems)
       mRitems.push_back(e.get());
}
```

# RenderItem Objects (2)

```
void Application::DrawRenderItems(ID3D12GraphicsCommandList* cmdList,
    const std::vector<RenderItem*>& ritems) {
     UINT objCBByteSize
        = (sizeof(ObjectConstants) + 255) & ~255;
    Microsoft::WRL::ComPtr<ID3D12Resource> objectCB; // = UploadBuffer

    for(size_t i = 0; i < ritems.size(); ++i) {
         auto ri = ritems[i];
         cmdList->IASetVertexBuffers(0, 1, &ri->Geo->VertexBufferView());
         cmdList->IASetIndexBuffer(&ri->Geo->IndexBufferView());
         cmdList->IASetPrimitiveTopology(ri->PrimitiveType);

         UINT cbvIndex = ri->ObjCBIndex;
         auto cbvHandle = CD3DX12_GPU_DESCRIPTOR_HANDLE(
              mCbvHeap->GetGPUDescriptorHandleForHeapStart());
         cbvHandle.Offset(cbvIndex, mCbvSrvUavDescriptorSize);

         cmdList->SetGraphicsRootDescriptorTable(0, cbvHandle);

         cmdList->DrawIndexedInstanced(ri->IndexCount, 1,
           ri->StartIndexLocation, ri->BaseVertexLocation, 0);
     }
}
```