Руководство для рендеринга квадратной фигуры с текстурой на DerectX11

Для работы с данным руководством вам понадобится минимум уже созданное базовое приложение на "windows.h" то есть Windows APP

(Руководство по созданию есть в папке по пути ..\pong\DocAPI)

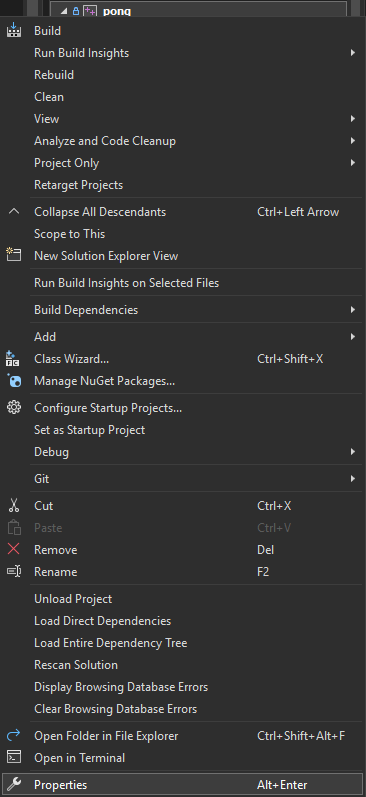
В данном руководстве не будут описаны методы оптимизации или организации работы d3d11, здесь будет представлен не абстрактный тип повествования с сырым исходным кодом, что позволит вам лучше понять работу базовых элементов в DerectX11.

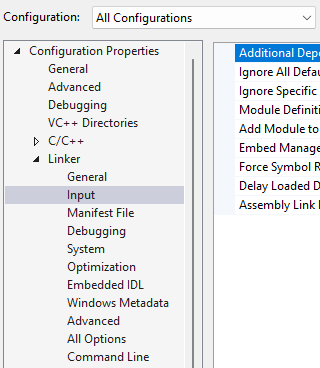
Для начала мы должны подключить библиотеку нашего d3d11

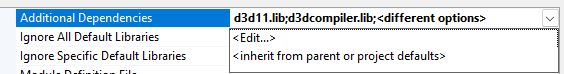
#include <d3d11.h>

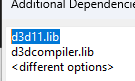
Но это еще не все сейчас нам нужно еще подключить библиотеки для корректной работы инклуда.

Запомните как подключать библиотеки это поможет вам потом подключать их самим:









Когда мы вписали библиотеку d3d11.lib можно начать писать наш код

Создадим класс нашего графического движка в отдельном хэдер файле

В public: части напишем конструктор и диструктор класса

GraphicEngine()

{}

~GraphicEngine()

{}

В {теле конструктора} будем создавать SwapChain устройства это определяет наше устройство (видео карту) и помогает в рендеринге сменять кадры и так же определяет область (наше окно виндорс) в которой мы будем что то рендерить

В первую очередь необходимо настроить дескриптор DXGI\_SWAP\_CHAIN\_DESC

Источник: https://learn.microsoft.com/en-us/windows/win32/api/dxgi/ns-dxgi-dxgi\_swap\_chain\_desc

DXGI\_SWAP\_CHAIN\_DESC SWdesc{};

SWdesc.BufferDesc.Width = 0;

SWdesc.BufferDesc.Height = 0;

SWdesc.BufferDesc.Format = DXGI\_FORMAT\_B8G8R8A8\_UNORM;

SWdesc.BufferDesc.RefreshRate.Numerator = 0;

SWdesc.BufferDesc.RefreshRate.Denominator = 0;

SWdesc.BufferDesc.Scaling = DXGI\_MODE\_SCALING\_UNSPECIFIED;

SWdesc.BufferDesc.ScanlineOrdering = DXGI\_MODE\_SCANLINE\_ORDER\_UNSPECIFIED;

SWdesc.SampleDesc.Count = 1;

SWdesc.SampleDesc.Quality = 0;

SWdesc.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;

SWdesc.BufferCount = 2;

SWdesc.OutputWindow = win.GetHWND();

SWdesc.Windowed = TRUE;

SWdesc.SwapEffect = DXGI\_SWAP\_EFFECT\_FLIP\_DISCARD;

SWdesc.Flags = 0;

Объяснение:

BufferDesc - режим отображения заднего буфера

Width - описывает ширину выходного окна (если 0 значит что он берет значение от нашего окна виндовс)

Height - тоже что и про Width только теперь высота

Format - формат отображения (тоесть по 32 бит на кждый пиксель)

Scaling - описывающий режим масштабирования

ScanlineOrdering - описывающий режим рисования сканлайна

RefreshRate - частота обновления в герц

Numerator - верхняя часть рационального числа

Denominator - нижняя часть рационального числа

SampleDesc - параметры

Count - Количество нескольких образцов на пиксель

Quality - качество отображения

BufferUsage - описывает использование поверхности и параметры доступа к процессору для заднего буфера

BufferCount - количество буферов в цепи свопов

OutputWindow - hWnd к выходному окну

Windowed - находится ли выход в оконном режиме

SwapEffect - описывает варианты обработки содержимого буфера

Flags - варианты поведения своп-цепочки

Мы создали на дескриптор цепочки свапов, теперь нам нужно его подключить к устройству для этого нам нужен дивайс и его образ

Есть 2 способа подключения при помощи указателя но при этом это нам нуждно будет указать кучу параметров для правильной его работы или при помощи универсального подключателя COM

1. Способ (пример)

private:

ID3D11Device\* pDevice = nullptr; - подключения в ручную

IDXGISwapChain\* pSwap = nullptr; -

ID3D11DeviceContext\* pContext = nullptr; -

ID3D11RenderTargetView\* pTarget = nullptr; -

ID3D11Resource\* pBackBuffer = nullptr; -

public:

pSwap->GetBuffer( - корректировка

0, - Буферный индекс нулевого уровня

\_\_uuidof(ID3D11Resource), - Тип интерфейса

reinterpret\_cast<void\*\*>(&pBackBuffer) - Указатель на интерфейс

);

1. Способ (мы будем использовать его)

#include <wrl.h>

using namespace Microsoft::WRL;

private:

HRESULT hr;

ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM

ComPtr<IDXGISwapChain> pGISwapChain{nullptr};

ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};

ComPtr<ID3D11Resource> pResource{ nullptr };

Мы подключили наши интерфейсы и теперь подключим нашу цепь к устройству Продолжаем в теле конструктора:

Источник: https://learn.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-d3d11createdeviceandswapchain

hr = D3D11CreateDeviceAndSwapChain - создание цепи

(nullptr, - Указатель на видеоадаптер (по умолчанию)

D3D\_DRIVER\_TYPE\_HARDWARE, - тип драйвера

nullptr, - Доступ к DLL

0, - слои времени выполнения для включения

nullptr, - уровней признаков(по умолчанию)

0, - Количество элементов для признаков

D3D11\_SDK\_VERSION, - Версия SDK

&SWdesc, - указатель на дескриптор цепи

&pGISwapChain, - указатель на интерфейс цепи

&pDevice, - указатель на интерфейс девайса

nullptr, - указатель на признаки

&pDeviceContext - указатель на интерфейс образа

)

Теперь подключим нашу цепь к ресурсам устройства

pGISwapChain->GetBuffer(0, IID\_PPV\_ARGS(&pResource)

И последним этапом получим доступ к ресурсам цели рендеринга

Источник: https://learn.microsoft.com/en-us/windows/win32/api/d3d12/nf-d3d12-id3d12device-createrendertargetview

private:

...

ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};

public:

Конструктор{

...

pDevice->CreateRenderTargetView(

pResource.Get(), - ресурс при таком действии не очищаются

nullptr, - дескриптор цели рендеринга

&pRenderTargetView - возвращаем данные для интерфейса

}

Мы закончили с конструктором, в деструкторы мы ничего не пишем.

Сразу создадим наш класс графического движка для использования в системе:

Класс графического движка

{

…

}d3dx;

Приступим к созданию функций которые уже будут что-то рендерить или рисовать на нашем устройстве и выводить на экран

1. Создадим и закрасим задний экран в какой либо цвет

Используем функцию очисти и закраски заднего экрана ClearRenderTargetView()

Источник: https://learn.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-id3d11devicecontext-clearrendertargetview

public:

void RenderClearBuffer(float red, float green, float blue)

{

const float color[] = { red,green,blue,1.0f };

pDeviceContext->ClearRenderTargetView(pRenderTargetView.Get(), color);

};

1. Синхронизирует отображение изображения на нашем устройстве с применением вертикальной синхронизации используем функцию Present()

Источник: https://learn.microsoft.com/en-us/windows/win32/api/dxgi/nf-dxgi-idxgiswapchain-present

public:

void Present(bool vSync)

{

if (vSync)

{

pGISwapChain->Present(1u, 0u);

}

else

{

pGISwapChain->Present(0u, 0u);

}

}

1. Мы можем теперь проверить что у нас все работает

void AppGame::Render()

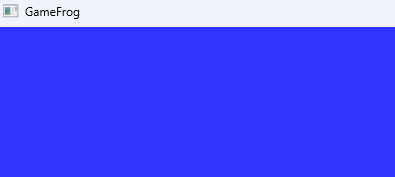
{

Sleep(16) - удаляем

d3dx.RenderClearBuffer(0.2f, 0.2f, 1.0f); - это будет синий цвет

d3dx.Present(true);

}



1. Начнем создание нашего первого квадрата с текстурой

Создадим функцию рисования объекта DrawObject():

public:

Void DrawObject(

float x, float y, float z,

float width, float height,

string filename)

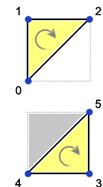
{тело функции};

Приступим к написанию тела функции

1. Создадим Вертексный буффер

Вертексы будут описывать нашу фигуру так как мы хотим описать квадрат он будет состоять из 2 треугольников и будет выглядеть это примерно так, можно заметить что указывать порядок вертексов нужно по часовой стрелке

Источник: https://learn.microsoft.com/ru-ru/windows/win32/direct3d9/rendering-from-vertex-and-index-buffers



Реализация:

Для начала создадим правило для указания вертексов, создадим структуру с координатами позиций и вертексов и позиции текстуры

(про текстуры мы поговорим еще потом)

class VEC3

{

public:

VEC3() = default;

VEC3(float x, float y,float z, float u, float v) :

x(x), y(y), z(z), u(u), v(v) {};

private:

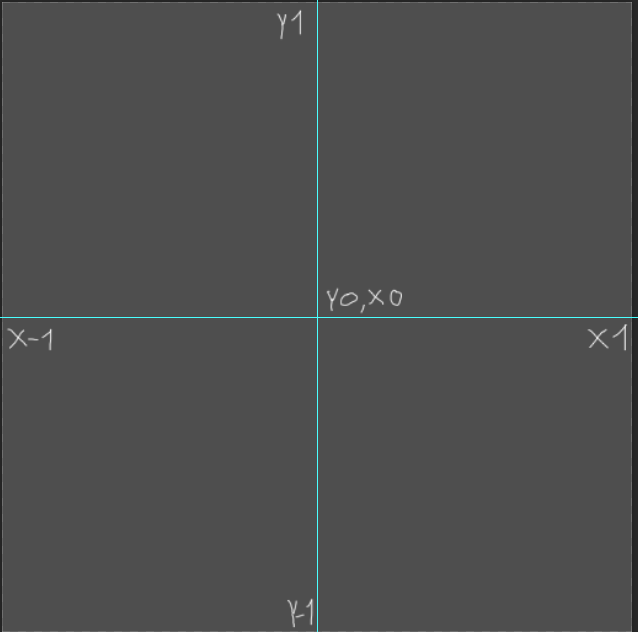
float x{}, y{}, z{}, u{}, v{};

};

Вернемся к реализации функции:

Полученные данные мы преобразуем в данные значения от -1 до 1

Это обусловлено особенностью расположения объектов в пространстве d3d11



Источник для D3D11\_BUFFER\_DESC: https://learn.microsoft.com/en-us/windows/win32/api/d3d11/ns-d3d11-d3d11\_buffer\_desc

Источник для D3D11\_SUBRESOURCE\_DATA: https://learn.microsoft.com/en-us/windows/win32/api/d3d11/ns-d3d11-d3d11\_subresource\_data

Источник для CreateBuffer: https://learn.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-id3d11device-createbuffer

Источник для IASetVertexBuffers: https://learn.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-id3d11devicecontext-iasetvertexbuffers

void DrawObject(x,y,z,width,height,filename)

{

float xLeft = (window.width / 2 - x) / (window.width / 2);

float xRight = (x + width - window.width / 2) / (window.width / 2);

float yTop = (window.height / 2 - y) / (window.height / 2);

float yBottom = (y + height - window.height / 2) / (window.height / 2);

float zBack = (z + width - window.width / 2) / (window.width / 2);

float zFront = (window.width / 2 - z) / (window.width / 2);

VEC3 VERTEX[] =

{

{ -xLeft,-yBottom,zFront, 0.0f,1.0f }, -левая нижняя часть

{ -xLeft,yTop,zFront, 0.0f,0.0f }, -левая верхняя

{ xRight,yTop,zFront, 1.0f,0.0f }, -правая верхняя

{ xRight,-yBottom,zFront, 1.0f,1.0f }, -правая нижняя

};

D3D11\_BUFFER\_DESC BD{};

BD.ByteWidth = sizeof(VERTEX); -размер в байтах всего массива

BD.StructureByteStride = sizeof(VEC3); -размер в байтах структуры

BD.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER; -тип буфуре

BD.Usage = D3D11\_USAGE\_DEFAULT; -способ расчета буфера

BD.CPUAccessFlags = 0u; -использование cpu

BD.MiscFlags = 0u; -дополнительные флаги

D3D11\_SUBRESOURCE\_DATA SD;

SD.pSysMem = VERTEX; -массив вертексов

pDevice->CreateBuffer(&BD, &SD, &pBuffer); -создание буфера

const UINT stride = sizeof(VEC3);

const UINT offset = 0u;

pDeviceContext->IASetVertexBuffers(

0u, - первый для связки вертекс

1u, - количество буферов

pBuffer.GetAddressOf(), - интерфейс буффера

&stride, - шаг по вертексам

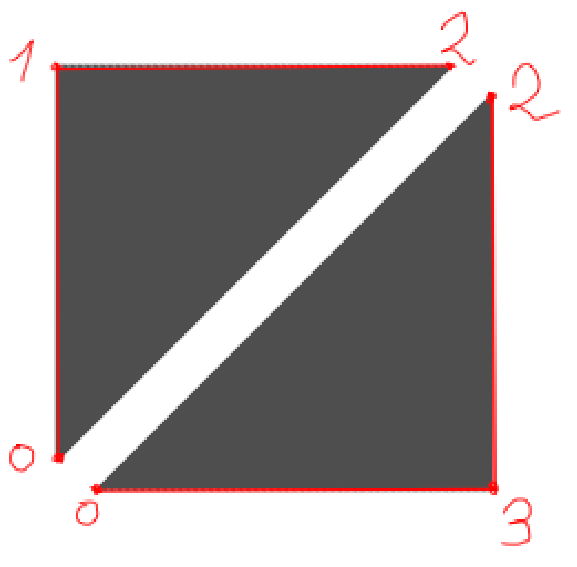
&offset - сдвиг по вртексам

);

}

1. Создадим индексный буфер:

Этот буфер нужен для правильного расположения вертексов в пространстве (то есть их последовательность отрисовки)



Индексы нужны для правильной обрисовки 2 треугольников с соединением их в 1 фигуру

Реализация:

void DrawObject(x,y,z,width,height,filename)

{

Вертексный буффер

...

unsigned short Index[] =

{

0,1,2,

2,3,0,

};

D3D11\_BUFFER\_DESC BD{};

BD.ByteWidth = sizeof(Index);

BD.StructureByteStride = sizeof(unsigned short);

BD.BindFlags = D3D11\_BIND\_INDEX\_BUFFER;

BD.Usage = D3D11\_USAGE\_DEFAULT;

BD.CPUAccessFlags = 0u;

BD.MiscFlags = 0u;

D3D11\_SUBRESOURCE\_DATA SD{};

SD.pSysMem = Index;

pDevice->CreateBuffer(&BD, &SD, &pIndexBuffer);

pDeviceContext->IASetIndexBuffer(

pIndexBuffer.Get(),

DXGI\_FORMAT\_R16\_UINT,

0u

);

UINT IndexCount = sizeof(Index)/sizeof(Index[0]) - количество точек

}

1. Создадим константный буффер, он нужен для реализации перемещения, позиционирования и манипулирования объектом в пространстве, для этого мы будем использовать матрицы

Подключаем #include <DirectXMath.h> для работы с матрицами

using namespace DirectX;

и переходим к реализации:

void DrawObject(x,y,z,width,height,filename)

{

Вертексный буффер

Индексный буффер

...

XMMATRIX Matrx =

{

XMMatrixTranspose( -умножает матрицы

XMMatrixTranslation(0,0,0) \* -матрица положения

XMMatrixPerspectiveLH(1,1,0.9f,30.0f) \* - перспектива

XMMatrixRotationZ(ZAngle) -матрица поворота

)

};

D3D11\_BUFFER\_DESC BD{};

BD.ByteWidth = sizeof(Matrx);

BD.StructureByteStride = 0u;

BD.BindFlags = D3D11\_BIND\_CONSTANT\_BUFFER;

BD.Usage = D3D11\_USAGE\_DYNAMIC;

BD.CPUAccessFlags = D3D11\_CPU\_ACCESS\_WRITE;

BD.MiscFlags = 0u;

D3D11\_SUBRESOURCE\_DATA SD{};

SD.pSysMem = Matrx;

pDevice->CreateBuffer(&BD, &SD, &pConstBuffer);

pDeviceContext->VSSetConstantBuffers(

0u,

1u,

pConstBuffer.GetAddressOf()

);

}

1. Создадим буффер текстуры