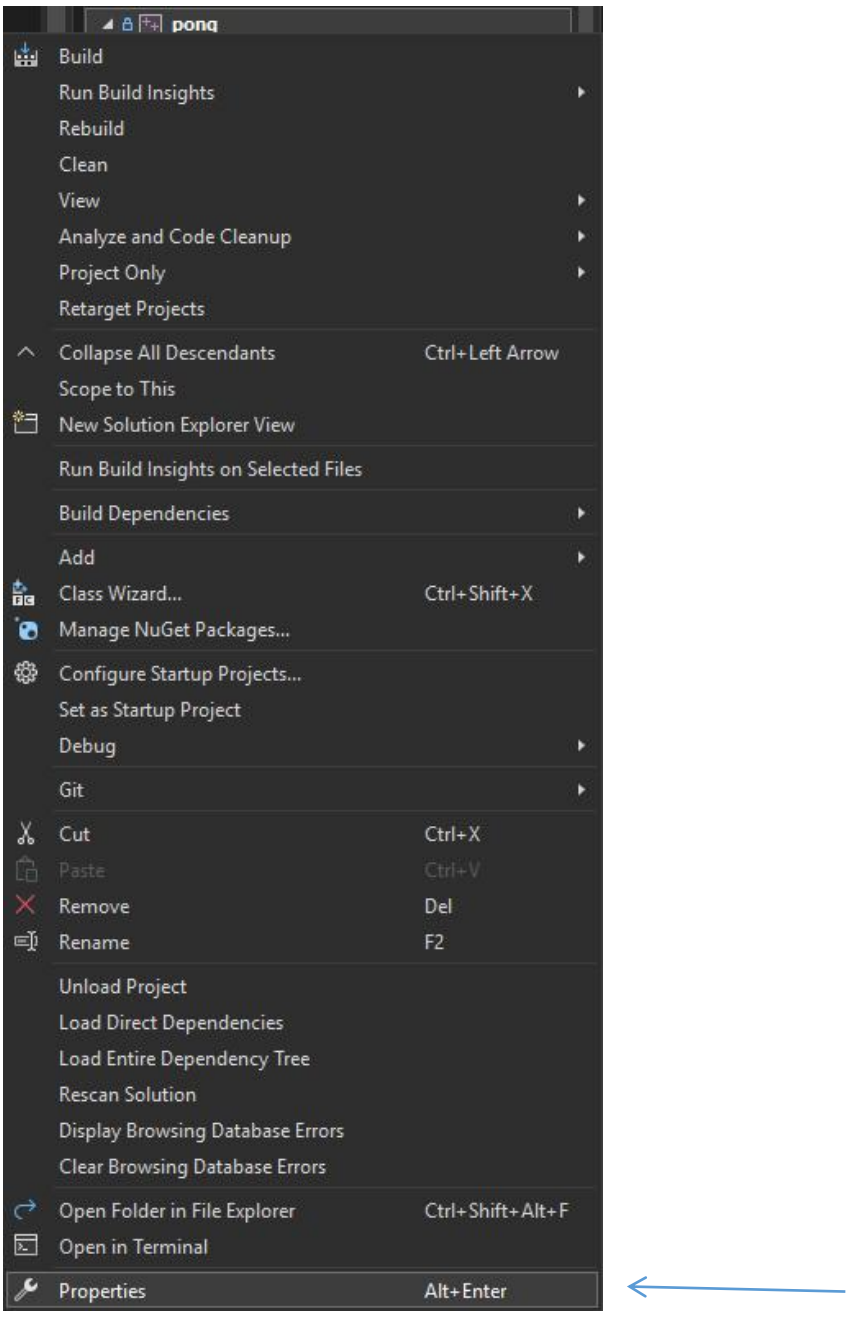
Руководство для рендеринга квадратной фигуры с текстурой на DerectX11

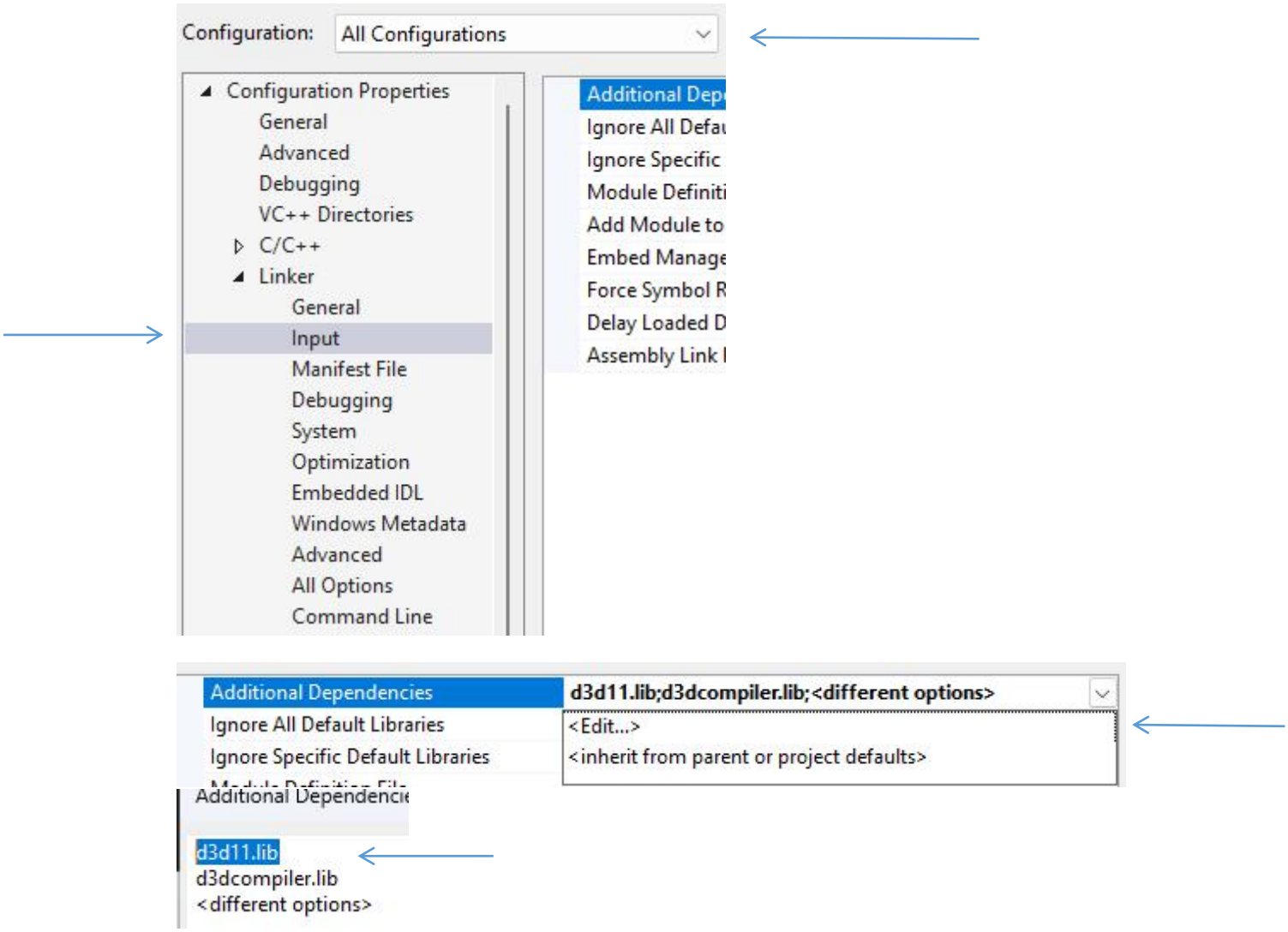
Для работы с данным руководством вам понадобится минимум уже созданное базовое приложение на "windows.h" то есть Windows APP
(Руководство по созданию есть в папке по пути ..\pong\DocAPI)

В данном руководстве не будут описаны методы оптимизации или организации работы d3d11, здесь будет представлен сырой исходный код к объяснением некоторых тонкостей, 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 подключить библиотеки для корректной работы инклуда `<d3d11.h>`.
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 (видеокарту) 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

```
Конструктор{
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 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 нужно 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
);
```

2. 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

```
Конструктор{
...
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));
```

```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr}; <--
```

И последним этапом получим доступ к ресурсам цели рендеринга

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

```
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 объект класса 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);
     };
```

2. 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);
	}
}
```

3. 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);
}
```

4. Начнем создание нашего первого квадрата с текстурой

Создадим функцию рисования объекта DrawObject():

```
#include <string>
using namespace std;
```

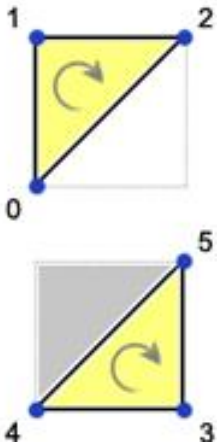
```
public:
Void DrawObject(
float x, float y, float z,
float width, float height)
{тело функции};
```

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

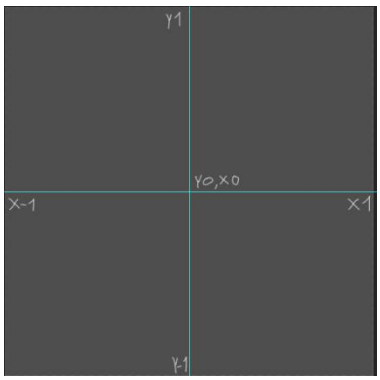


Реализация:

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

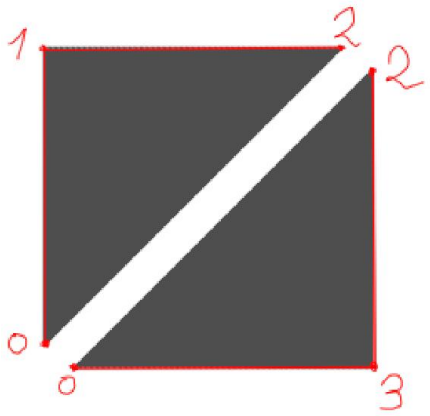
```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};
ComPtr<ID3D11Buffer> pBuffer{nullptr}; <--
```

```
void DrawObject(x,y,z,width,height)
{
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 BD1{};
BD1.ByteWidth = sizeof(VERTEX);           -размер в байтах всего массива
BD1.StructureByteStride = sizeof(VEC3);    -размер в байтах структуры
BD1.BindFlags = D3D11_BIND_VERTEX_BUFFER; -тип буфуре
BD1.Usage = D3D11_USAGE_DEFAULT;                -способ расчета буфера
BD1.CPUAccessFlags = 0u;                  -использование cpu
BD1.MiscFlags = 0u;                       -дополнительные флаги

D3D11_SUBRESOURCE_DATA SD1;
SD1.pSysMem = VERTEX;                       -массив вертексов
```

```
pDevice->CreateBuffer(&BD1, &SD1, &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
);

}
```

2. 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:

```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};
ComPtr<ID3D11Buffer> pBuffer{nullptr};
ComPtr<ID3D11Buffer> pIndexBuffer{nullptr}; <--
```

```
void DrawObject(x,y,z,width,height)
{
 Вертексный буффер
 ...

unsigned short Index[] =
{
   0,1,2,
   2,3,0,
};
```

```
D3D11_BUFFER_DESC BD2{};
BD2.ByteWidth = sizeof(Index);
BD2.StructureByteStride = sizeof(unsigned short);
BD2.BindFlags = D3D11_BIND_INDEX_BUFFER;
BD2.Usage = D3D11_USAGE_DEFAULT;
BD2.CPUAccessFlags = 0u;
BD2.MiscFlags = 0u;

D3D11_SUBRESOURCE_DATA SD2{};
SD2.pSysMem = Index;

pDevice->CreateBuffer(&BD2, &SD2, &pIndexBuffer);

pDeviceContext->IASetIndexBuffer(
pIndexBuffer.Get(),
DXGI_FORMAT_R16_UINT,
0u
);

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

}
```

3. 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:

```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};
ComPtr<ID3D11Buffer> pBuffer{nullptr};
ComPtr<ID3D11Buffer> pIndexBuffer{nullptr};
ComPtr<ID3D11Buffer> pConstBuffer{nullptr}; <--
```

```
void DrawObject(x,y,z,width,height)
{
 Вертексный буффер
 Индексный буффер
 ...

XMMATRIX Matrx[] =
{
      XMMatrixTranspose(                              -умножает матрицы
            XMMatrixTranslation(0,0,0) *              -матрица положения
            XMMatrixPerspectiveLH(1,1,0.9f,30.0f)     - перспектива
      )
};

D3D11_BUFFER_DESC BD3{};
```

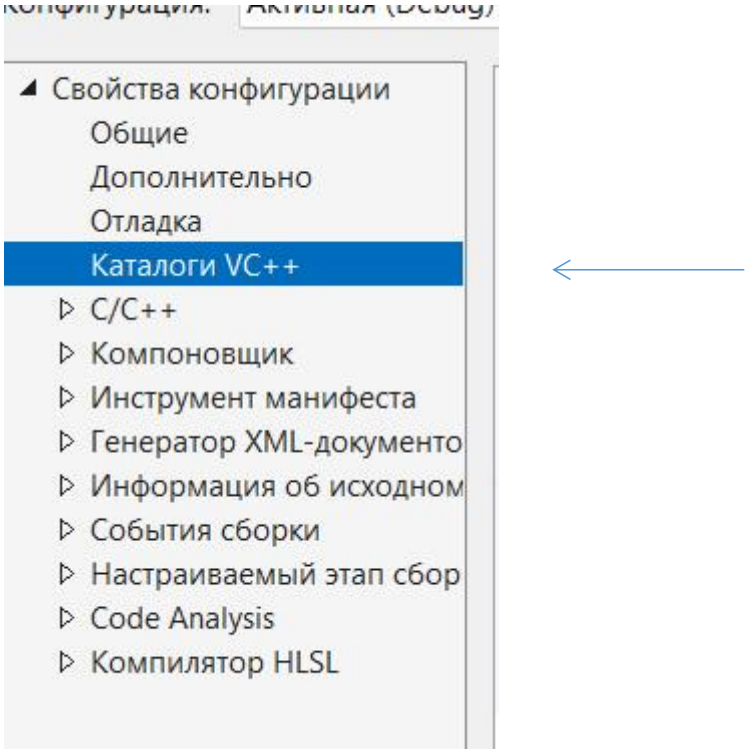
```
BD3.ByteWidth = sizeof(Matrx);
BD3.StructureByteStride = 0u;
BD3.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
BD3.Usage = D3D11_USAGE_DYNAMIC;
BD3.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
BD3.MiscFlags = 0u;

D3D11_SUBRESOURCE_DATA SD3{};
SD3.pSysMem = Matrx;

pDevice->CreateBuffer(&BD3, &SD3, &pConstBuffer);
pDeviceContext->VSSetConstantBuffers(
0u,
1u,
pConstBuffer.GetAddressOf()
);

}
```

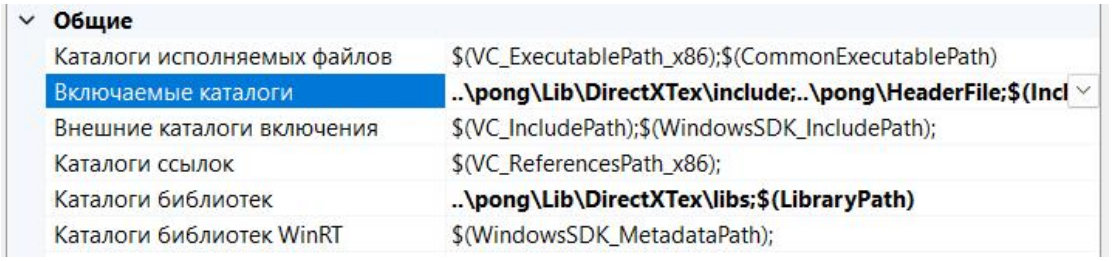
4. Создадим буффер текстуры

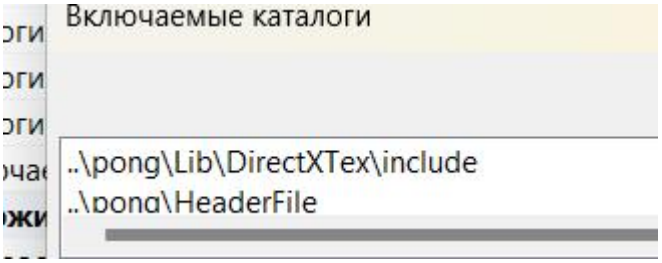
В первую очередь загрузить библиотеку и инклуд которых нету в проекте
Запомните как это делается и вы научитесь скачивать и линковать библиотеки с других репозиториев.

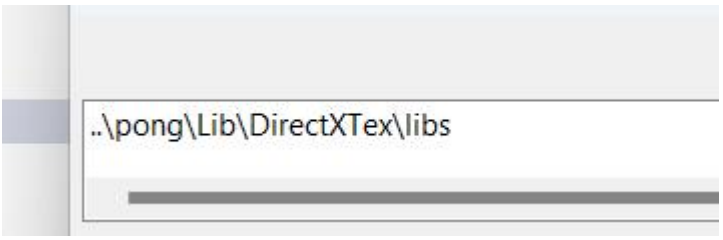
Нужно скачать библиотеку по данной ссылке: https://github.com/Liy-Egor/FROG_PONG_CLEAR/tree/Misha_new_enemy/pong/Lib/DirectXTex

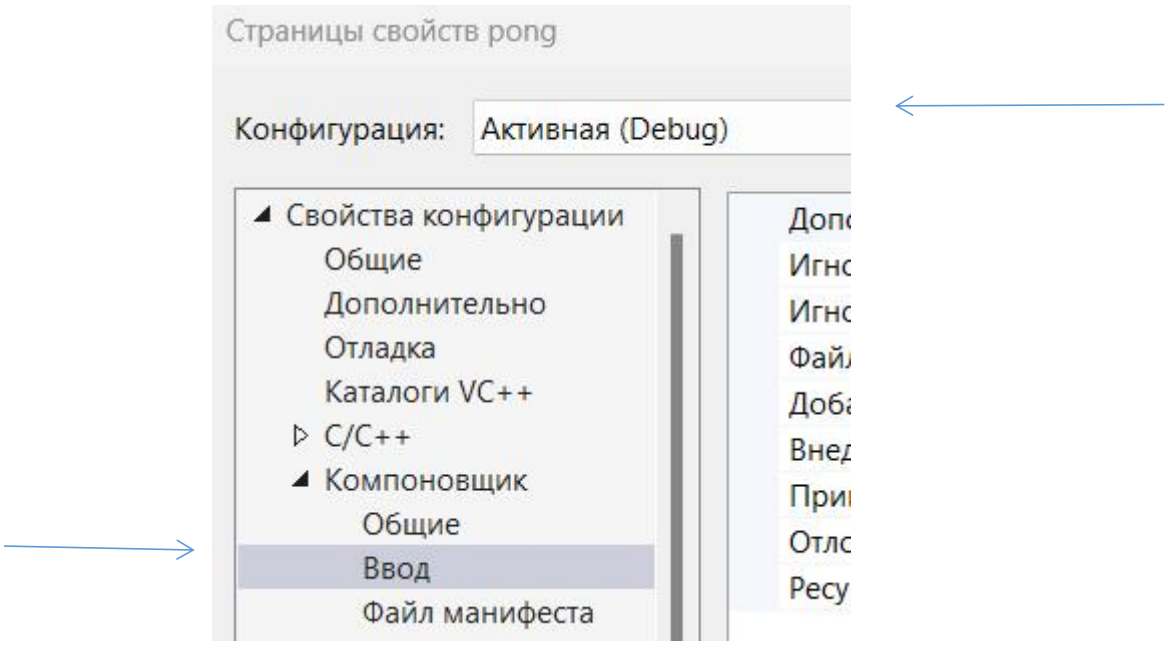
И закидываем в свой проект
Теперь данную библиотеку нужно залинковать:

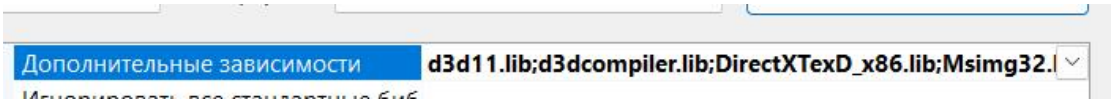






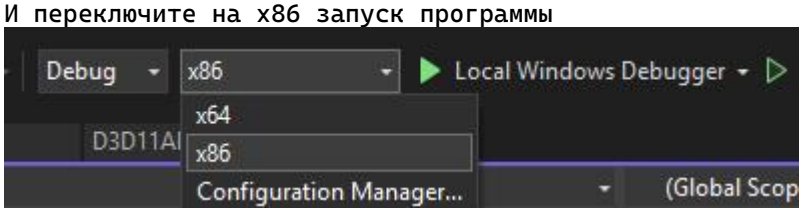






Теперь можно спокойно подключить инклуд в проект `#include "DirectXTex.h"`

И переключите на x86 запуск программы



Подготовим изображение которое будем использовать в виде текстуры

Имя изображения: image.png

Данную картинку мы закидываем в нашу папку с проектом

Теперь начнем реализацию кода:

Источник D3D11_TEXTURE2D_DESC: D3D11_TEXTURE2D_DESC (d3d11.h) - Win32 apps | Microsoft Learn

Источник D3D11_SHADER_RESOURCE_VIEW_DESC: D3D11_SHADER_RESOURCE_VIEW_DESC (d3d11.h) - Win32 apps | Microsoft Learn

Источник D3D11_SAMPLER_DESC: D3D11_SAMPLER_DESC (d3d11.h) - Win32 apps | Microsoft Learn

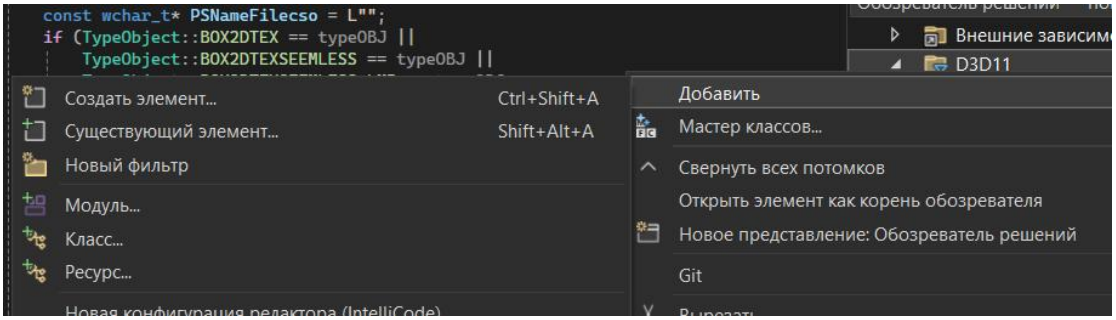
```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};
ComPtr<ID3D11Buffer> pBuffer{nullptr};
ComPtr<ID3D11Buffer> pIndexBuffer{nullptr};
ComPtr<ID3D11Buffer> pConstBuffer{nullptr};
ComPtr<ID3D11Texture2D> pTex2D{nullptr}; <--
ComPtr<ID3D11ShaderResourceView> pShaderResView{nullptr}; <--
ComPtr<ID3D11SamplerState> pSampler{nullptr}; <--
```

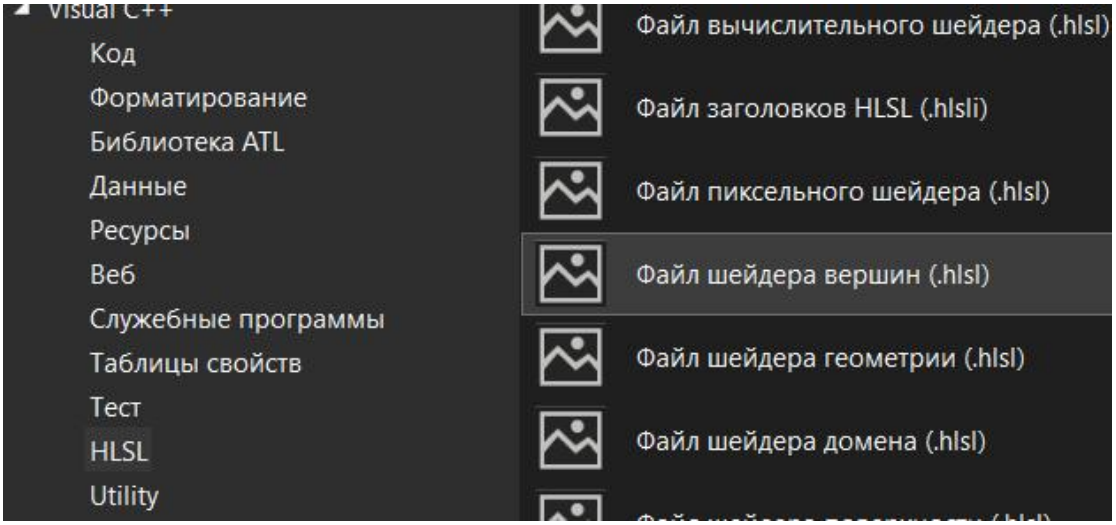
```
void DrawObject(x,y,z,width,height)
{
 Вертексный буффер
 Индексный буффер
 Константный буффер
 ...
```

```
ScratchImage IMAGE;             - контейнер для изображения
LoadFromWICFile(L"image.png", DirectX::WIC_FLAGS_NONE, nullptr, IMAGE);

      D3D11_TEXTURE2D_DESC TXDC{};
      TXDC.Width = IMAGE.GetImages()->width;    -размеры изображения
      TXDC.Height = IMAGE.GetImages()->height;  -
      TXDC.MipLevels = 1;                       - уровни мипмапов в текстуре
      TXDC.ArraySize = 1;                       - количество текстур
      TXDC.Format = DXGI_FORMAT_B8G8R8A8_UNORM;    - формат
      TXDC.SampleDesc.Count = 1;
      TXDC.SampleDesc.Quality = 0;
      TXDC.Usage = D3D11_USAGE_DEFAULT;
      TXDC.BindFlags = D3D11_BIND_SHADER_RESOURCE;
      TXDC.CPUAccessFlags = 0;
      TXDC.MiscFlags = 0;

      D3D11_SUBRESOURCE_DATA SD{};
      SD.pSysMem = IMAGE.GetImages()->pixels;            -массив пикселей
      SD.SysMemPitch = IMAGE.GetImages()->rowPitch;      -шаг

      D3D11_SHADER_RESOURCE_VIEW_DESC SRVD{};
      SRVD.Format = DXGI_FORMAT_B8G8R8A8_UNORM;
      SRVD.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D; -тип ресурска
      SRVD.Texture2D.MostDetailedMip = 0;
      SRVD.Texture2D.MipLevels = 1;

      D3D11_SAMPLER_DESC SDC{};
      SDC.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;  -фильтр отрисовки
      SDC.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;     -адресные разметки
      SDC.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
      SDC.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;

      pDevice->CreateTexture2D(&TXDC, &SD, &pTex2D);

      pDevice->CreateShaderResourceView(
      pTex2D.Get(),
      &SRVD,
      &pShaderResView);

      pDevice->CreateSamplerState(&SDC, &pSampler);

      ID3D11ShaderResourceView** pSrv = new ID3D11ShaderResourceView*;
      ID3D11SamplerState** pSs = new ID3D11SamplerState*;

      pDevice->CreateTexture2D(&TXDC, &SD, &pTex2D);
      pDevice->CreateShaderResourceView(pTex2D.Get(), &SRVD, pSrv);
      pDevice->CreateSamplerState(&SDC, pSs);

      pDeviceContext->PSSetShaderResources(0u, 1u, pSrv);
      pDeviceContext->PSSetSamplers(0, 1, pSs);
}
```

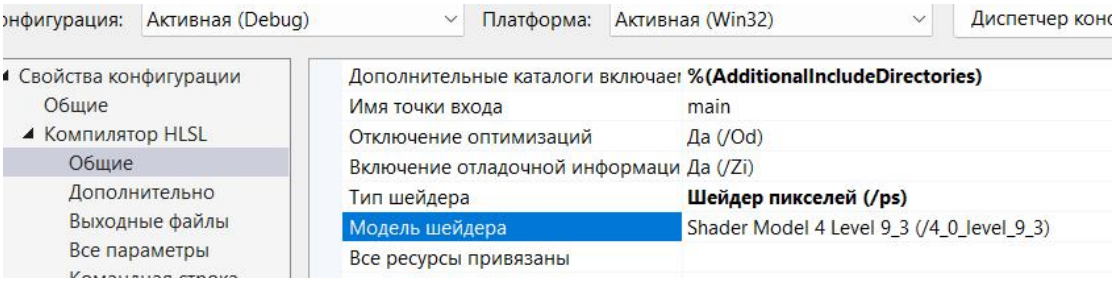
5. Когда все буфферы собраны можно начать работать с шейдерами

В первую очередь мы должны создать эти шейдеры, у нас будет 2 шейдера вертексный и пиксельный

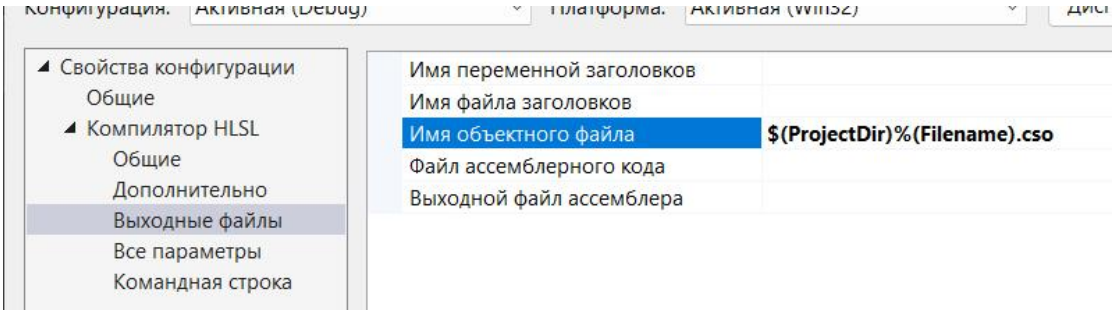




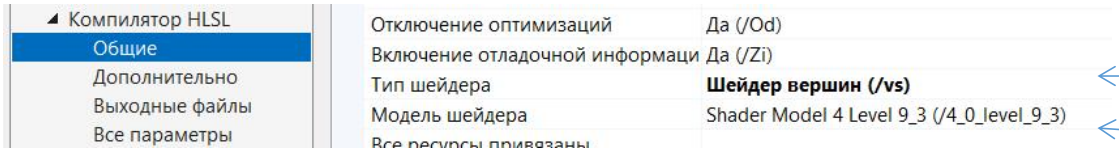
И так же их надо настроить в параметрах шейдеров



Для пиксельного шейдера



Для вертекстного шейдера



Теперь напишем код шейдеров внутри созданных шейдеров

Для вертексного шейдера VetexShader.hlsl

```
cbuffer Cbuf
{
    matrix transform;
};
struct VSOut
{
    float2 tex : TEXCOORD;
    float4 pos : SV_Position;
};

VSOut main(float3 pos : POSITION, float2 tex:TEXCOORD)
{
    VSOut vso;
    vso.pos = mul(float4(pos ,1.0f), transform); умножении текстуры на
                                                                позицию
    vso.tex = tex;
    return vso;
}
```

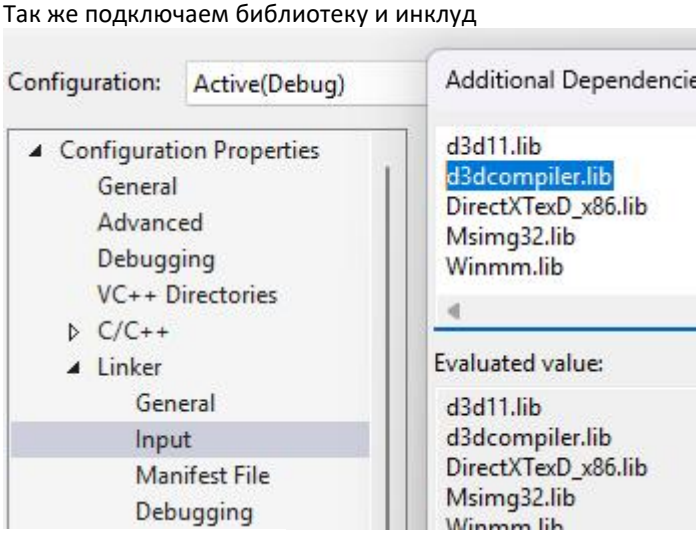
Для пиксельного шейдера PixelShader.hlsl

```
Texture2D tex;

SamplerState splr;

float4 main(float2 tc : TEXCOORD) : SV_TARGET
{
    return tex.Sample(splr, tc);
}
```

Теперь подключим шейдеры к нашему устройству, между зависимостями шейдеров осуществляется автоматически

```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};
ComPtr<ID3D11Buffer> pBuffer{nullptr};
ComPtr<ID3D11Buffer> pIndexBuffer{nullptr};
ComPtr<ID3D11Buffer> pConstBuffer{nullptr};
ComPtr<ID3D11Texture2D> pTex2D{nullptr};
ComPtr<ID3D11ShaderResourceView> pShaderResView{nullptr};
ComPtr<ID3D11SamplerState> pSampler{nullptr};
ComPtr<ID3DBlob> pBlobVS{nullptr}; <--
ComPtr<ID3DBlob> pBlobPS{nullptr}; <--
ComPtr<ID3D11VertexShader> pVertexShader{nullptr}; <--
ComPtr<ID3D11PixelShader> pPixelShader{nullptr}; <--
```

Так же подключаем библиотеку и инклуд



#include <d3dcompiler.h>

```
void DrawObject(x,y,z,width,height)
{
 Вертексный буффер
 Индексный буффер
 Константный буффер
 Текстуры
 ...

D3DReadFileToBlob(L"VertexShader.cso", &pBlobVS);
D3DReadFileToBlob(L"PixelShader.cso", &pBlobPS);

pDevice->CreateVertexShader(
pBlobVS->GetBufferPointer(),
pBlobVS->GetBufferSize(),
nullptr,
&pVertexShader);

pDeviceContext->VSSetShader(pVertexShader.Get(), 0, 0);

pDevice->CreatePixelShader(
pBlobPS->GetBufferPointer(),
pBlobPS->GetBufferSize(),
nullptr,
&pPixelShader);

pDeviceContext->PSSetShader(pPixelShader.Get(), 0, 0);
}
```

```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};
ComPtr<ID3D11Buffer> pBuffer{nullptr};
```

```
ComPtr<ID3D11Buffer> pIndexBuffer{nullptr};
ComPtr<ID3D11Buffer> pConstBuffer{nullptr};
ComPtr<ID3D11Texture2D> pTex2D{nullptr};
ComPtr<ID3D11ShaderResourceView> pShaderResView{nullptr};
ComPtr<ID3D11SamplerState> pSampler{nullptr};
ComPtr<ID3DBlob> pBlobVS{nullptr};
ComPtr<ID3DBlob> pBlobPS{nullptr};
ComPtr<ID3D11VertexShader> pVertexShader{nullptr};
ComPtr<ID3D11PixelShader> pPixelShader{nullptr};
ComPtr<ID3D11InputLayout> pInputLayout{nullptr}; <--
```

6. Теперь добавим слои которые определят как использовать шейдеры с нашими расчетами

```
void DrawObject(x,y,z,width,height)
{
 Вертексный буффер
 Индексный буффер
 Константный буффер
 Текстуры
 Шейдеры
 ...

D3D11_INPUT_ELEMENT_DESC EDSC [] =   -специальная разметка для шейдеров
{
      {"POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
D3D11_INPUT_PER_VERTEX_DATA, 0},

      {"TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, 12u,
D3D11_INPUT_PER_VERTEX_DATA, 0}
};

pDevice->CreateInputLayout(
EDSC,
sizeof(EDSC)/sizeof(EDSC[0]),
pBlobVS->GetBufferPointer(),
pBlobVS->GetBufferSize(),
&pInputLayout
);

pDeviceContext->IASetInputLayout(pInputLayout.Get());

}
```

```
private:
HRESULT hr;
ComPtr<ID3D11Device> pDevice{ nullptr }; - подключение COM
ComPtr<IDXGISwapChain> pGISwapChain{nullptr};
ComPtr<ID3D11DeviceContext> pDeviceContext{nullptr};
ComPtr<ID3D11Resource> pResource{ nullptr };
ComPtr<ID3D11RenderTargetView> pRenderTargetView{nullptr};
ComPtr<ID3D11Buffer> pBuffer{nullptr};
ComPtr<ID3D11Buffer> pIndexBuffer{nullptr};
ComPtr<ID3D11Buffer> pConstBuffer{nullptr};
ComPtr<ID3D11Texture2D> pTex2D{nullptr};
ComPtr<ID3D11ShaderResourceView> pShaderResView{nullptr};
ComPtr<ID3D11SamplerState> pSampler{nullptr};
ComPtr<ID3DBlob> pBlobVS{nullptr};
ComPtr<ID3DBlob> pBlobPS{nullptr};
ComPtr<ID3D11VertexShader> pVertexShader{nullptr};
ComPtr<ID3D11PixelShader> pPixelShader{nullptr};
```

```
ComPtr<ID3D11InputLayout> pInputLayout{nullptr};
ComPtr<ID3D11BlendState> pBlendState{nullptr}; <--
```

7. В качестве бонуса еще реализуем показ альфа канала, на нашем изображении конечно нету альфа канала то есть прозрачности но если вам нужна прозрачность то это уже будет реализовано

```
void DrawObject(x,y,z,width,height)
{
 Вертексный буффер
 Индексный буффер
 Константный буффер
 Текстуры
 Шейдеры
 Слои
 ...

D3D11_BLEND_DESC blendDesc;
ZeroMemory(&blendDesc, sizeof(D3D11_BLEND_DESC));
blendDesc.RenderTarget[0].BlendEnable = TRUE;
blendDesc.RenderTarget[0].SrcBlend = D3D11_BLEND_SRC_ALPHA;
blendDesc.RenderTarget[0].DestBlend = D3D11_BLEND_INV_SRC_ALPHA;
blendDesc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
blendDesc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
blendDesc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
blendDesc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
blendDesc.RenderTarget[0].RenderTargetWriteMask =
D3D11_COLOR_WRITE_ENABLE_ALL;
pDevice->CreateBlendState(&blendDesc, &pBlendState);
float blendFactor[4] = { 0.0f, 0.0f, 0.0f, 0.0f };

pDeviceContext->OMSetBlendState(pBlendState.Get(), blendFactor,
0xffffffff);
}
```

8. Последнее что остается, это указать отображение вьюпорта и тип отрисовки

```
void DrawObject(x,y,z,width,height)
{
 Вертексный буффер
 Индексный буффер
 Константный буффер
 Текстуры
 Шейдеры
 Слои
 Альфа канал
 ...
```

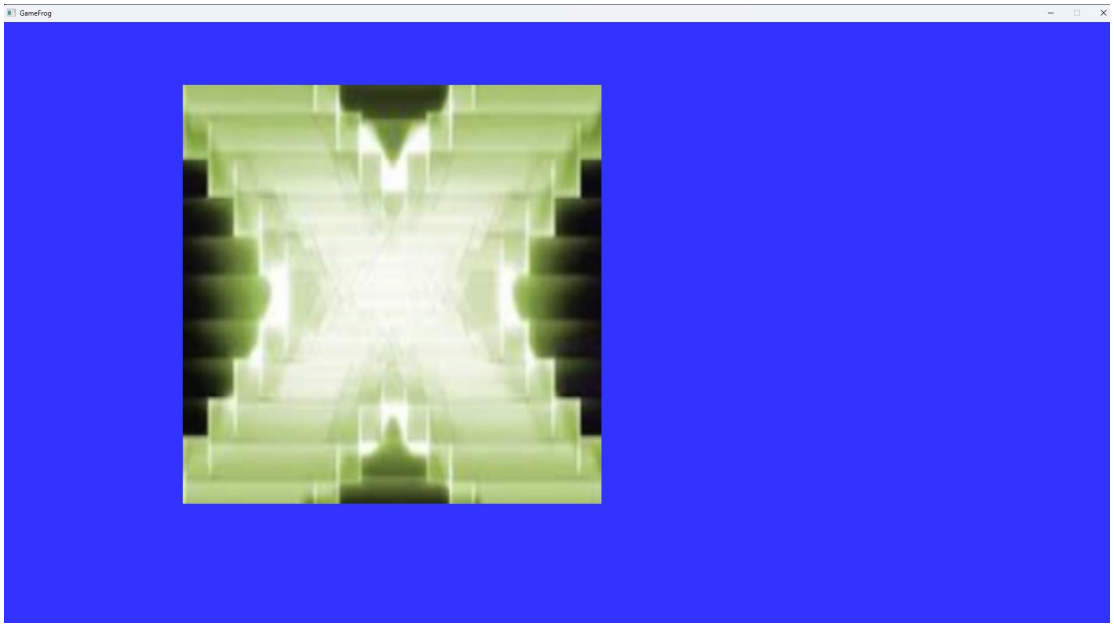
```
D3D11_VIEWPORT VP{};
VP.Width = window.width;
VP.Height = window.height;
VP.MinDepth = 0;
VP.MaxDepth = 1;
VP.TopLeftX = 0;
VP.TopLeftY = 0;
```

```
pDeviceContext->RSSetViewports(1u, &VP);

pDeviceContext->OMSetRenderTargets(
1u,
pRenderTargetView.GetAddressOf(),
nullptr
);

pDeviceContext->
IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);

pDeviceContext->DrawIndexed(IndexCount, 0u, 0u);
}
```

9. Все наша функция готова и теперь начнем рендерить наш квадрат

```
void AppGame::Render()
{
	d3dx.RenderClearBuffer(0.2f, 0.2f, 1.0f);

	d3dx.DrawObject(
	600, 300, 1,
	400, 400);

	d3dx.Present(true);
}
```

Что получилось по итогу руководства:



Репозиторий на исходный код программы:
https://github.com/wordlol/pract3d/tree/DirectX11