3. Inicjalizacja *D3D*

W niniejszym rozdziale prześledzimy krok po kroku wszystkie ważniejsze polecenia *IDirect3D9* i *IDirect3Ddevice9*, dzięki którym będziemy mogli zainicjalizować bibliotekę *D3D*, przeprowadzić enumerację (sprawdzenie) kart graficznych, utworzyć interfejs do urządzenia i rozpocząć rysowanie.

3.1 Enumeracja urządzeń

Na początku pobieramy interfejs przy pomocy którego będziemy kontrolować pracę całej biblioteki. Funkcja pobierająca interfejs *D3D* przedstawiona jest na listingu 18.

```
IDirect3D9 *Direct3DCreate9(
    UINT SDKVersion
);
```

Listing 1 – *Direct3DCreate9*, Źródło *DirectX SDK*

Przykład implementacji znajduje się na listingu 19.

```
if ( NULL == ( m_pD3D9 = Direct3DCreate9( D3D_SDK_VERSION ) ) )
      return FALSE;
```

Listing 2 – Użycie *Direct3DCreate9*, Źródło własne

Jako parametr podajemy stałą zdefiniowaną w każdym *SDK*. Wartość zwracana przez tę funkcję to interfejs do *D3D*.

Następnym krokiem jest *enumeracja* wszystkich istniejących i kompatybilnych kart graficznych z *Direct3D*, do której służy metoda *IDirect3D9::GetAdapterCount* przedstawiona na listingu 20.

```
UINT GetAdapterCount(VOID);
```

Listing 3 – *IDirect3D9::GetAdapterCount*, Źródło *DirectX SDK*

Przykład implementacji znajduje się na listingu 21.

```
if ( 0 == ( m_iAdapterCount = m_pD3D9->GetAdapterCount() ) )
      return FALSE;
```

Listing 4 – Użycie *GetAdapterCount*, Źródło własne

Zmienna *m_iAdapterCount* będzie zawierać liczbę wszystkich kart w systemie, jeżeli będzie się równać *0* to znaczy, że nie ma żadnej kompatybilnej karty. Jeżeli już wiemy, ile jest kart graficznych w systemie, pobieramy nazwę każdej z nich oraz sprawdzamy wersję sterownika, służy do tego metoda *IDirect3D9::GetAdapterIdentifier* przedstawiona na listingu 22.

```
HRESULT GetAdapterIdentifier(
    UINT Adapter,
    DWORD Flags,
    D3DADAPTER_IDENTIFIER9 *pIdentifier
);
```

Listing 5 – *IDirect3D9::GetAdapterIdentifier*, Źródło *DirectX SDK*

Przykład implementacji znajduje się na listingu 23.

```
if (D3D_OK!=pD3D9->GetAdapterIdentifier(iDeviceNumber,NULL,&m_D3Did))
{
      m_pDebugger->Log( "", "    - Can't Get Adapter Identifier\n");
      return;
}
```

Listing 6 – Użycie *GetAdapterIdentifier*, Źródło własne

Pierwszy parametr jest numerem urządzenia. Numer urządzenia musi być większy bądź równy *0*, a mniejszy niż liczba wszystkich urządzeń w systemie, może też być *D3DADAPTER_DEFAULT*, czyli stanowić urządzenie domyślne. Drugi parametr może być *0* albo *D3DENUM_WHQL_LEVEL*. Jeżeli użyjemy tej flagi, to system może się połączyć z witryna *Microsoftu* w celu uzyskania najnowszych sterowników posiadających certyfikat WHQL. Trzeci parametr to wskaźnik do struktury *D3DADAPTER_IDENTIFIER9*, która otrzyma dane na temat karty. Listing 24 przedstawia tą strukture.

```
typedef struct _D3DADAPTER_IDENTIFIER9 {
char Driver[MAX_DEVICE_IDENTIFIER_STRING];
char Description[MAX_DEVICE_IDENTIFIER_STRING];
char DeviceName[32];
LARGE_INTEGER DriverVersion;
DWORD DriverVersionLowPart;
DWORD DriverVersionHighPart;
DWORD VendorId;
DWORD DeviceId;
DWORD SubSysId;
DWORD Revision;
GUID DeviceIdentifier;
DWORD WHQLLevel;
} D3DADAPTER_IDENTIFIER9;
```

Listing 7 – *D3DADAPTER_IDENTIFIER9*, Źródło *DirectX SDK*

Struktura z listingu 24 zawiera informacje, które można zaprezentować użytkownikowi naszego programu, np.:

- *Driver* - nazwa sterownika,
- *DeviceName* – nazwa urządzenia,
- *DriverVersion* – wersja sterownika,
- *WHQLLevel* – czy sterowniki przeszły test *WHQL*, *1* – tak, *0* – nie.

Następnie musimy pobrać numer wszystkich trybów graficznych, użyjemy do tego metody przedstawionej na listingu 25.

```
UINT GetAdapterModeCount(
    UINT Adapter,
    D3DFORMAT Format
);
```

Listing 8 – *IDirect3D9::GetAdapterModeCount*, Źródło *DirectX SDK*

Przykład użycia *IDirect3D9::GetAdapterModeCount* przedstawiony jest na listingu 26.

```
if ( 0 == ( m_iModesNumber = pD3D9->GetAdapterModeCount( DeviceNumber,
           D3DFMT_X8R8G8B8 ) ) )
      return;
```

Listing 9 – Użycie *GetAdapterModeCount*, Źródło własne

Pierwszy parametr to numer urządzenia albo *D3DADAPTER_DEFAULT*. Drugi - to format *back-buffera*. To, co zwraca funkcja, to ilość wszystkich trybów graficznych na danej karcie.

Kolejnym krokiem jest pobranie listy obsługiwanych trybów graficznych. Służy do tego *IDirect3D9::EnumAdapterModes* przedstawiony na listingu 27.

```
HRESULT EnumAdapterModes(
    UINT Adapter,
    D3DFORMAT Format,
    UINT Mode,
    D3DDISPLAYMODE* pMode
);
```

Listing 10 – *IDirect3D9::EnumAdapterModes*, Źródło *DirectX SDK*

Pierwszy parametr to numer urządzenia albo *D3DADAPTER_DEFAULT*, drugi - to numer trybu, większy bądź równy *0*, ale mniejszy od wszystkich trybów dostępnych.

Trzeci parametr do format *back-buffera*. Ostatni parametr to wskaźnik do struktury *D3DDISPLAYMODE*, która otrzyma informacje na temat trybu. Struktura *D3DDISPLAYMODE* jest przedstawiona na listingu 28.

```
typedef struct _D3DDISPLAYMODE {
    UINT Width;
    UINT Height;
    UINT RefreshRate;
    D3DFORMAT Format;
} D3DDISPLAYMODE;
```

Listing 11 – *IDirect3D9::EnumAdapterModes*, Źródło *DirectX SDK*

Z listingu 28, *Width* to szerokość w *pikselach* trybu, *Height* to wysokość, *RefreshRate* to częstotliwość odświeżania ekranu, a *Format* to format koloru w jakim pracuje tryb (lista formatów kolorów była podana w poprzednim rozdziale). Przykład zastosowania *EnumAdapterModes* znajduje się na listingu 29.

```
if (NULL==(m_pDisplayModes = new D3DDISPLAYMODE[ m_iModesNumber ] ) )
      return;

for ( UINT i = 0; i < m_iModesNumber; i++ )
{
      if ( D3D_OK != pD3D9->EnumAdapterModes( iDeviceNumber,
                        D3DFMT_X8R8G8B8, i, &m_pDisplayModes[i] ) )
            return;
}
```

Listing 12 – Zastosowanie *IDirect3D9::EnumAdapterModes*, Źródło własne

Gdy już wiemy ile jest trybów urządzenia, jakie są wersje sterownika itd. należy sprawdzić urządzenie pod kątem jego możliwości renderowania grafiki. Posłużymy się: *IDirect3D::GetDeviceCaps* przedstawionym na listingu 30.

```
HRESULT GetDeviceCaps(
    UINT Adapter,
    D3DDEVTYPE DeviceType,
    D3DCAPS9 *pCaps
);
```

Listing 13 – Zastosowanie *IDirect3D9::GetDeviceCaps*, Źródło *DirectX SDK*

Pierwszy parametr to numer urządzenia albo *D3DADAPTER_DEFAULT*, następny to tryb urządzenia *D3DDEVTYPE*. Typ *D3DDEVTYPE* przedstawiony został na listingu 31.

```
typedef enum _D3DDEVTYPE {
    D3DDEVTYPE_HAL = 1,
    D3DDEVTYPE_REF = 2,
    D3DDEVTYPE_SW = 3,
    D3DDEVTYPE_FORCE_DWORD = 0xffffffff
} D3DDEVTYPE;
```

Listing 14 – *D3DDEVTYPE*, Źródło *DirectX SDK*

HAL – to akceleracja sprzętowa, *REF* – Reference Rasterizer, *SW* – implementacja software'owa (nie używana). Trzecim parametrem metody *IDirect3D::GetDeviceCaps* jest wskaźnik do struktury *D3DCAPS9*, która otrzyma dane na temat karty. Struktura *D3DCAPS9* przedstawiona została na listingu 32.

```
typedef struct _D3DCAPS9 {
    D3DDEVTYPE DeviceType;
    UINT AdapterOrdinal;
    DWORD Caps;
    DWORD Caps2;
    DWORD Caps3;
    DWORD PresentationIntervals;
    DWORD CursorCaps;
    DWORD DevCaps;
    DWORD PrimitiveMiscCaps;
    DWORD RasterCaps;
    DWORD ZCmpCaps;
    DWORD SrcBlendCaps;
    DWORD DestBlendCaps;
    DWORD AlphaCmpCaps;
    DWORD ShadeCaps;
    DWORD TextureCaps;
    DWORD TextureFilterCaps;
    DWORD CubeTextureFilterCaps;
    DWORD VolumeTextureFilterCaps;
    DWORD TextureAddressCaps;
    DWORD VolumeTextureAddressCaps;
    DWORD LineCaps;
    DWORD MaxTextureWidth;
    DWORD MaxTextureHeight;
    DWORD MaxVolumeExtent;
    DWORD MaxTextureRepeat;
    DWORD MaxTextureAspectRatio;
    DWORD MaxAnisotropy;
    float MaxVertexW;
    float GuardBandLeft;
    float GuardBandTop;
    float GuardBandRight;
    float GuardBandBottom;
    float ExtentsAdjust;
    DWORD StencilCaps;
    DWORD FVFCaps;
    DWORD TextureOpCaps;
```

```
    DWORD MaxTextureBlendStages;
    DWORD MaxSimultaneousTextures;
    DWORD VertexProcessingCaps;
    DWORD MaxActiveLights;
    DWORD MaxUserClipPlanes;
    DWORD MaxVertexBlendMatrices;
    DWORD MaxVertexBlendMatrixIndex;
    float MaxPointSize;
    DWORD MaxPrimitiveCount;
    DWORD MaxVertexIndex;
    DWORD MaxStreams;
    DWORD MaxStreamStride;
    DWORD VertexShaderVersion;
    DWORD MaxVertexShaderConst;
    DWORD PixelShaderVersion;
    float PixelShader1xMaxValue;
    DWORD DevCaps2;
    float MaxNpatchTesselationLevel;
    float MinAntialiasedLineWidth;
    float MaxAntialiasedLineWidth;
    UINT MasterAdapterOrdinal;
    UINT AdapterOrdinalInGroup;
    UINT NumberOfAdaptersInGroup;
    DWORD DeclTypes;
    DWORD NumSimultaneousRTs;
    DWORD StretchRectFilterCaps;
    D3DVSHADERCAPS2_0 VS20Caps;
    D3DPSHADERCAPS2_0 PS20Caps;
    DWORD VertexTextureFilterCaps;
    DWORD MaxVShaderInstructionsExecuted;
    DWORD MaxPShaderInstructionsExecuted;
    DWORD MaxVertexShader30InstructionSlots;
    DWORD MaxPixelShader30InstructionSlots;
    DWORD Reserved2;
    DWORD Reserved3;
} D3DCAPS9;
```

Listing 15 – *D3DCAPS9*, Źródło *DirectX SDK*

Omówimy wybrane pola tej struktury, które będą nam potrzebne do pracy:

- *Caps2* między innymi posiada flagi:
  - *D3DCAPS2_CANAUTOGENMIPMAP*

    Urządzenie może generować automatycznie mipmapy (szczegółowo omówione w rozdziale poświęconemu teksturom i mipmapom).
  - *D3DCAPS_DYNAMICTEXTURE,*

    Urządzenie posiada dynamiczne tekstury.
- *PresentationInterval* może zawierać:
  - *D3DPRESENT_INTERVAL_IMMEDIATE* co oznacza, że urządzenie nie musi czekać na powrót plamki, żeby wyświetlić zawartość drugiego bufora.
- *DevCaps*:

- *D3DDEVCAPS_HWRASTERIZATION*

  Urządzenie jest w stanie rasteryzować trójkąty sprzętowo.

- *D3DDEVCAPS_HWTRANSFORMANDLIGHT*

  Urządzenie posiada sprzętową jednostkę transformacji i oświetlania.

- *D3DDEVCAPS_PUREDEVICE*

  Urządzenie może być zadeklarowane jako *PURE*.

- *PrimitiveMiscCaps*:
  - *D3DPMISCCAPS_BLENDOP*

    Urządzenie posiada możliwość łączenia kolorów (np. do efektu przezroczystości).

  - *D3DPMISCCAPS_CULLCW*

    Urządzenie posiada możliwość odrzucania trójkątów których wierzchołki są zapisane wg ruchu wskazówek zegara (opisanie operacji zwane *CULLINGiem* będzie w części opisowej *RenderStates*).

  - *D3DPMISCCAPS_CULLCCW*

    Urządzenie posiada możliwość odrzucania trójkątów, których wierzchołki są zapisane w przeciwna stronę niż ruchu wskazówek zegara.

  - *D3DPMISCCAPS_CULLNONE*

    Urządzenie posiada możliwość nie odrzucania trójkątów.

- *RasterCaps*:
  - *D3DPRASTERCAPS_DITHER*

    Urządzenie posiada możliwość ditheringu czyli poprawiania jakości obrazu w trybie o małej ilości kolorów np. *16bpp*.

  - *D3DPRASTERCAPS_ZTEST*

    Urządzenie może wykonywać *Z* test.

- *ZCmpCaps*
  - *D3DPCMPCAPS_ALWAYS*

    Tryb pracy *ZBufora*. Piksele zawsze pozytywnie przechodzą przez test.

  - *D3DPCMPCAPS_GREATER*

    Tryb pracy *Zbufora*. Piksele przechodzą przez test, gdy ich wartość *Z* jest większa niż w *ZBuforze*.

  - *D3DPCMPCAPS_GREATEREQUAL*

    Tryb pracy *Zbufora*. Piksele przechodzą przez test, gdy ich wartość *Z* jest większa bądź równa niż ta w *ZBuforze*.

- *D3DPCMPCAPS_EQUAL*

  Tryb pracy *Zbufora*. Piksele przechodzą przez test, gdy ich wartość *Z* jest równa tej w *ZBuforze*.

- *D3DPCMPCAPS_LESS*

  Tryb pracy *ZBufora*. Piksele przechodzą przez test, gdy ich wartość *Z* jest mniejsza niż w *ZBuforze*.

- *D3DPCMPCAPS_LESSEQUAL*

  Tryb pracy *ZBufora*. Piksele przechodzą przez test, gdy ich wartość *Z* jest mniejsza bądź równa do tej w *ZBuforze*.

- *D3DPCMPCAPS_NEVER*

  Tryb pracy *ZBufora*. Piksele nigdy nie przechodzą przez test.

- *D3DPCMPCAPS_NOTEQUAL*

  Tryb pracy *ZBufora*. Piksele przechodzą przez test, gdy ich wartość *Z* nie jest równa tej w *ZBuforze*.

- *SrcBlendCaps*
  - *D3DPBLENDCAPS_INVSRCALPHA*

    Współczynnik łączenia kolorów pikseli może być równy *(1-As, 1-As, 1-As, 1-As)*, gdzie *As* to wartość *alfa* dla piksela źródłowego.

  - *D3DPBLENDCAPS_ONE*

    Współczynnik łączenia kolorów pikseli może być równy *(1, 1, 1, 1)*.

  - *D3DPBLENDCAPS_SRCALPHA*

    Współczynnik łączenia kolorów pikseli może być równy *(As, As, As, As)*.

  - *D3DPBLENDCAPS_ZERO*

    Współczynnik łączenia kolorów pikseli może być równy *(0, 0, 0, 0)*.

- *DestBlendCaps*

  Te same wartości jak powyżej.

- *ShadeCaps*
  - *D3DPSHADECAPS_COLORGOURAUDRGB*

    Interpolacja liniowa koloru w trójkącie między wierzchołkami. Jeżeli np. na jednym wierzchołku mamy wartość *0*, a na drugim *1*, to piksele między nimi będą miały wartość wyinterpolowaną liniowo między tymi wartościami.

- *TextureCaps*
  - *D3DPTEXTURECAPS_ALPHA*

Tekstura może mieć kanał alfa.

- *D3DPTEXTURECAPS_MIPMAP*

  Tekstura może mieć *mipmapy* (dalej jest rozdział poświęcony *teksturom* i *mipmapom*).

- *TextureFilterCaps*
  - *D3DPTFILTERCAPS_MAGFLINEAR*

    Urządzenie ma możliwość *biliniowego* filtrowania *tekstury*.
  - *D3DPTFILTERCAPS_MINFLINEAR*

    Urządzenie ma możliwość *biliniowego* filtrowania *tekstury*.
  - *D3DPTFILTERCAPS_MIPFLINEAR*

    Urządzenie ma możliwość trójliniowego filtrowania *tekstury* między *mipmapami*.
  - *D3DPTFILTERCAPS_MAGFPOINT*

    Urządzenie ma możliwość zaokrąglania do najbliższego piksela w czasie filtrowania *tekstury*.
  - *D3DPTFILTERCAPS_MIPFLINEAR*

    Urządzenie ma możliwość zaokrąglania do najbliższego piksela w czasie filtrowania *tekstury* w czasie używania *mipmap*.
  - *D3DPTFILTERCAPS_MINFLINEAR*

    Urządzenie ma możliwość zaokrąglania do najbliższego piksela w czasie filtrowania tekstury.
- *TextureAddressCaps*
  - *D3DPTADDRESSCAPS_CLAMP*

    Wszystkie wartości współrzędnych tekstury większych od jeden albo mniejszych od zera będą przyjmować wartości na odpowiednich krawędziach tekstury.
  - *D3DPTADDRESSCAPS_WRAP*

    Urządzenie ma możliwość „zawijania” tekstury kiedy współrzędne tekstury są większe od jednego albo mniejsze od zera.
- *MaxTextureWidth*

  Maksymalna szerokość tekstury.
- *MaxTextureHeight*

  Maksymalna wysokość tekstury.

- *TextureOpCaps*
  - *D3DTEXOPCAPS_ADD*

    Urządzenie ma możliwość dodawania kolorów w operacjach multiteksturowania.
  - *D3DTEXOPCAPS_MODULATE*

    Urządzenie ma możliwość mnożenia kolorów w operacjach multiteksturowania.
- *MaxSimulatenousTextures*

  Ilość tekstur które mogą być jednocześnie nakładane na piksel.
- *VertexProcessingCaps*
  - *D3DVTXCAPS_DIRECTIONALLIGHTS*

    Urządzenie ma sprzętową akceleracje świateł kierunkowych (*DIRECTIONAL*).
  - *D3DVTXCAPS_POSITIONALLIGHT*

    Urządzenie ma sprzętową akceleracje świateł pozycyjnych (*POINT*/*SPOT*).
  - *D3DVTXCAPS_TEXGEN*

    Urządzenie ma sprzętową akceleracje generowania współrzędnych tekstury.
- *MaxActiveLight*

  Liczna akcelerowanych sprzętowo świateł.

Gdy już wiemy jakie są urządzenia posiada możliwości, musimy sprawdzić z jakimi formatami koloru może pracować ekran i tylny bufor. Służy do tego metoda *IDirect3D9::CheckDeviceType* przedstawiona na listingu 33.

```
HRESULT CheckDeviceType(
    UINT Adapter,
    D3DDEVTYPE DeviceType,
    D3DFORMAT DisplayFormat,
    D3DFORMAT BackBufferFormat,
    BOOL Windowed
);
```

Listing 16 – *IDirect3D9::CheckDeviceType*, Źródło *DirectX SDK*

Pierwszy parametr jest numerem urządzenia, drugi to jego typ (*HAL, REF, SW*), kolejny to format ekranu. Większość urządzeń posiada następujące dwa formaty: *D3DFMT_X8R8G8B8* albo *D3FMT_R5G6B5*. Czwarty parametr jest formatatem tylnego bufora. Powinien być równy pod względem ilości bitów, jak format ekranu, np. w przypadku *D3DFMT_X8R8G8B8* możliwe formaty bufora to *D3DFMT_X8R8G8B8* i *D3DFMT_A8R8G8B8*. Jeżeli format nie jest równy, wtedy nie możemy skorzystać z *buffer-flipping*. Następnie możemy sprawdzić, jakie rodzaje zasobów ma urządzenie

i jakie funkcje te zasoby mogą przyjmować metodą *IDirect3D8::CheckDeviceFormat* przedstawioną na listingu 34.

```
HRESULT CheckDeviceFormat(
    UINT Adapter,
    D3DDEVTYPE DeviceType,
    D3DFORMAT AdapterFormat,
    DWORD Usage,
    D3DRESOURCETYPE RType,
    D3DFORMAT CheckFormat
);
```

Listing 17 – *IDirect3D9::CheckDeviceFormat*, Źródło *DirectX SDK*

Pierwszy parametr jest numerem urządzenia, drugi to typ (*HAL, REF, SW*). Trzecim parametrem jest format zasobu. Czwarty identyfikuje sposób w jaki chcemy użyć zasób:

- *D3DUSAGE_DEPTHSTENCIL* – zasób będzie buforem *Z*,
- *D3DUSAGE_RENDERTARGET* – zasób winien posiadać możliwość renderowania do niego,
- *D3DUSAGE_AUTOGENMIPMAP* – zasób (zazwyczaj tekstura) może tworzyć mipmapy automatycznie przez kartę graficzną.

Piąty parametr jest typem zasobu, jednym z możliwych przedstawionych na listingu 35.

```
typedef enum _D3DRESOURCETYPE {
    D3DRTYPE_SURFACE = 1,
    D3DRTYPE_VOLUME = 2,
    D3DRTYPE_TEXTURE = 3,
    D3DRTYPE_VOLUMETEXTURE = 4,
    D3DRTYPE_CUBETEXTURE = 5,
    D3DRTYPE_VERTEXBUFFER = 6,
    D3DRTYPE_INDEXBUFFER = 7,
    D3DRTYPE_FORCE_DWORD = 0x7fffffff
} D3DRESOURCETYPE;
```

Listing 18 – *D3DRESOURCETYPE*, Źródło *DirectX SDK*

Z listingu 35:

- *Surface* to podstawowy surface *D3D*,
- *Texture* to zwykła tekstura *2D*,
- *VertexBuffer* to bufor dla punktów – vertexów,
- *IndexBuffer* to bufor dla indeksów,

Szósty parametr funkcji *IDirect3D8::CheckDeviceFormat* to żądany format zasobu.

Przykład zastosowania przedstawiony jest na listingu 36.

```
if ( ( SUCCEEDED( pD3D9->CheckDeviceType( iDeviceNumber,
            D3DDEVTYPE_HAL,D3DFMT_X8R8G8B8,D3DFMT_A8R8G8B8,FALSE)) ) &&
     ( SUCCEEDED( pD3D9->CheckDeviceFormat( iDeviceNumber,
            D3DDEVTYPE_HAL, D3DFMT_X8R8G8B8, D3DUSAGE_RENDERTARGET,
            D3DRTYPE_TEXTURE, D3DFMT_A8R8G8B8 ) ) ) &&
     ( SUCCEEDED( pD3D9->CheckDeviceFormat( iDeviceNumber,
            D3DDEVTYPE_HAL, D3DFMT_X8R8G8B8, D3DUSAGE_AUTOGENMIPMAP,
            D3DRTYPE_TEXTURE, D3DFMT_A8R8G8B8 ) ) ) )
{
      m_PixelFormat.wBackBufferFormat |= D3D9PIXELFMT_A8R8G8B8;
      m_PixelFormat.wRenderableTextureFormat |= D3D9PIXELFMT_A8R8G8B8;
}
```

Listing 19 – Przykład zastosowania *CheckDeviceType* i *CheckDeviceFormat*, Źródło własne

Następnie musimy sprawdzić jaki format buforu *Z* jest dopuszczalny do użytku, posłużymy się do tego metodą *IDirect3D9::CheckDepthStencilMatch* przedstawioną na listingu 37.

```
HRESULT CheckDepthStencilMatch(
    UINT Adapter,
    D3DDEVTYPE DeviceType,
    D3DFORMAT AdapterFormat,
    D3DFORMAT RenderTargetFormat,
    D3DFORMAT DepthStencilFormat
);
```

Listing 20 – *IDirect3D9::CheckDepthStencilMatch*, Źródło *DirectX SDK*

Pierwszy parametr to numer urządzenia. Drugi to typ. Trzeci to format ekranu. Czwarty - tylnego bufora, a piąty bufora *Z*. Przykład zastosowania metody metodą *IDirect3D9::CheckDepthStencilMatch* znajduje się na listingu 38.

```
if ( ( SUCCEEDED( pD3D9->CheckDepthStencilMatch( iDeviceNumber,
            D3DDEVTYPE_HAL, D3DFMT_X8R8G8B8, D3DFMT_A8R8G8B8,
            D3DFMT_D24S8 ) ) ) &&
     ( SUCCEEDED( pD3D9->CheckDepthStencilMatch( iDeviceNumber,
            D3DDEVTYPE_HAL, D3DFMT_X8R8G8B8, D3DFMT_X8R8G8B8,
            D3DFMT_D24S8 ) ) ) &&
     ( SUCCEEDED( pD3D9->CheckDeviceFormat( iDeviceNumber,
            D3DDEVTYPE_HAL, D3DFMT_X8R8G8B8, D3DUSAGE_DEPTHSTENCIL,
            D3DRTYPE_SURFACE, D3DFMT_D24S8 ) ) ) )
{
      m_PixelFormat.wZBufferFormat |= D3D9ZFMT_D24S8;
```

```
}
```

Listing 21 – Przykład sprawdzenia czy dostępny jest *Zbufor* o formacie *D3DFMT_D24S8*, Źródło własne

3.2 Inicjalizacja urządzenia

Aby uzyskać interfejs do konkretnego urządzenia musimy posłużyć się funkcją *IDirect3D9::CreateDevice* przedstawioną na listingu 39.

```
HRESULT CreateDevice(
    UINT Adapter,
    D3DDEVTYPE DeviceType,
    HWND hFocusWindow,
    DWORD BehaviorFlags,
    D3DPRESENT_PARAMETERS *pPresentationParameters,
    IDirect3DDevice9** ppReturnedDeviceInterface
);
```

Listing 22 – *IDirect3D9::CreateDevice*, Źródło *DirectX SDK*

Pierwszy parametr jest numerem urządzenia. Drugi jest jego typem. Trzeci paramter jest uchwyte, do okna z którym urządzenie będzie pracować. Czwarty jest kombinacją flag:

- *D3DCREATE_FPU_PRESERVE*

  Użycie tej flagi powoduje, że *D3D* nie zmienia *FPU* w tryb pojedynczej dokładności. Tryb pojedynczej dokładności jest idealny dla liczb zmiennoprzecinkowych pojedynczej dokładności i jest szybszy. *FPU* pracuje normalnie w trybie podwójnej dokładności.
- *D3DCREATE_HARDWARE_VERTEXPROCESSING*
- *D3DCREATE_SOFTWARE_VERTEXPROCESSING*
- *D3DCREATE_MIXED_VERTEXPROCESSING*

  Mamy możliwość zmiany w trakcie używania urządzenia trybu pracy: albo sprzętowe albo programowe transformowanie i oświetlanie.
- *D3DCREATE_PUREDEVICE*

  Tworzy urządzenie *PURE*
- *D3DCREATE_MULTITHREADED*

  Powoduje, że *D3D* jest bezpieczne na zmianę wątków. Powoduje jednak lekką degradację wydajności.

Piąty parametr jest strukturą przekazującą ustawienia urządzenia. Struktura ta przedstawiona jest na listingu 40.

```
typedef struct _D3DPRESENT_PARAMETERS_ {
    UINT BackBufferWidth, BackBufferHeight;
    D3DFORMAT BackBufferFormat;
    UINT BackBufferCount;
    D3DMULTISAMPLE_TYPE MultiSampleType;
    DWORD MultiSampleQuality;
    D3DSWAPEFFECT SwapEffect;
    HWND hDeviceWindow;
    BOOL Windowed;
    BOOL EnableAutoDepthStencil;
    D3DFORMAT AutoDepthStencilFormat;
    DWORD Flags;
    UINT FullScreen_RefreshRateInHz;
    UINT PresentationInterval;
} D3DPRESENT_PARAMETERS;
```

Listing 23 – *D3DPRESENT_PARAMETERS*, Źródło *DirectX SDK*

- *BackBufferWidth* – szerokość tylnego bufora i ekranu,
- *BackBufferHeight* – wysokość tylnego bufora i ekranu,
- *BackBufferFormat* – format koloru tylnego bufora,
- *BackBufferCount* – ilość tylnych buforów (*1* albo *2*),
- *MultiSampleType* – ustawia antyaliasing – nie będziemy używać,
- *SwapEffect* – informuje kartę czy ma wykonywać kopiowanie między buforami, czy zamieniać wskaźniki. Wartości jakie może przyjmować *SwapEffect* znajdują się na listingu 41.

```
typedef enum _D3DSWAPEFFECT {
    D3DSWAPEFFECT_DISCARD = 1,
    D3DSWAPEFFECT_FLIP = 2,
    D3DSWAPEFFECT_COPY = 3,
    D3DSWAPEFFECT_FORCE_DWORD = 0xFFFFFFFF
} D3DSWAPEFFECT;
```

Listing 24 – *D3DSWAPEFFECT*, Źródło *DirectX SDK*

Opis wartości z listingu 41: *Flip* – zamiana back buffer na front, *Copy* – kopiowanie z back buffer do front

- *hDeviceWindow* – uchwyt do okna z którym pracować ma urządzenie,
- *Windowed – TRUE/FALSE*,
- *EnableAutoDepthStencil – TRUE/FALSE* – automatyczne tworzonie *ZBufora*,
- *AutoDepthStencilFormat* – format *ZBufora*,
- *FullScreen_PresentationInterval* – może przyjmować:
  - *D3DPRESENT_INTERVAL_IMMEDIATE* – urządzenie nie musi synchronizować się z odświeżaniem ekranu. Uprzednio należy sprawdzić czy istnieje taka możliwość w *D3DCAPS9*.

Ostatni parametr *IDirect3D9::CreateDevice* to wskaźnik, gdzie będzie przekazany interfejs do urządzenia. Przykład zastosowania znajduje się na listingu 42.

```
m_d3dpp.EnableAutoDepthStencil = TRUE;
m_d3dpp.BackBufferWidth = m_pConfig->m_wWidth;
m_d3dpp.BackBufferHeight = m_pConfig->m_wHeight;
m_d3dpp.BackBufferCount = 1;
m_d3dpp.Windowed = TRUE;
m_d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
m_d3dpp.BackBufferFormat = m_pCurrentDevice->m_BackBufferFormat;
m_d3dpp.AutoDepthStencilFormat = m_pCurrentDevice->m_ZBufferFormat;
m_d3dpp.FullScreen_RefreshRateInHz = D3DPRESENT_RATE_DEFAULT;
m_d3dpp.MultiSampleType = D3DMULTISAMPLE_NONE;
m_d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_DEFAULT;
m_d3dpp.hDeviceWindow = m_pConfig->m_hWnd;

m_p3dDevice = NULL;

if ( FAILED( hr = m_pD3D9->CreateDevice(
            m_pConfig->m_dwD3DDeviceNumber, D3DDEVTYPE_HAL,
            m_pConfig->m_hWnd, D3DCREATE_HARDWARE_VERTEXPROCESSING |
            D3DCREATE_PUREDEVICE, &m_d3dpp, &m_p3dDevice ) ) )
      return FALSE;
```

Listing 25 – Inicjalizacja urządzenia w oparciu o parametry konfiguracyjne, Źródło własne

Teraz, kiedy już mamy utworzony interfejs urządzenia i podstawowe bufory (*tylny* i *Z*) możemy przygotować urządzenie do pracy.

Ustawienia *Viewport* informują urządzenie na jakim obszarze ekranu może renderować. Zazwyczaj ustawiamy *viewport* na rozmiar ekranu. Struktura *D3DVIEWPORT9* przedstawiona jest na listingu 43.

```
typedef struct _D3DVIEWPORT9 {
    DWORD X;
    DWORD Y;
    DWORD Width;
    DWORD Height;
```

```
    float MinZ;
    float MaxZ;
} D3DVIEWPORT9;
```

Listing 26 – *D3DVIEWPORT9*, Źródło *DirectX SDK*

Parametr *MinZ* z listingu 43 jest minimalną wartość *Z* – zazwyczaj *0*, *MaxZ* to maksymalna wartość *Z* – zazwyczaj *1*, *IDirect3Ddevice9::SetViewport* przedstawiony jest na listingu 44.

```
HRESULT SetViewport(
    CONST D3DVIEWPORT9 *pViewport
);
```

Listing 27 – *IDirect3DDevice9::SetViewport*, Źródło *DirectX SDK*

Przykład zastosowania znajduje się na listingu 45.

```
m_vp.X = 0;
m_vp.Y = 0;
m_vp.Width = m_pConfig->m_wWidth;
m_vp.Height = m_pConfig->m_wHeight;
m_vp.MinZ = 0.0f;
m_vp.MaxZ = 1.0f;

if ( FAILED( m_p3dDevice->SetViewport( &m_vp ) ) )
      AnErrorMsg("D3D9 Renderer Error","Can't Set Viewport");
```

Listing 28 – Zastosowanie *SetViewport*, Źródło własne

Następnie informujemy urządzenie, jak ma pracować: przy pomocy metody *IDirect3Ddevice9::SetRenderState* przedstawionej na listingu 46.

```
HRESULT SetRenderState(
    D3DRENDERSTATETYPE State,
    DWORD Value
);
```

Listing 29 – *IDirect3DDevice9::SetRenderState*, Źródło *DirectX SDK*

Pierwszy parametr to *RenderState*, a drugi to wartość jaką ma przyjmować. Omówimy tylko ważniejsze oraz te z których korzystać będziemy.

- *D3DRS_ZENABLE* może przyjmować wartości przedstawione na listingu 44.

```
typedef enum _D3DZBUFFERTYPE {
    D3DZB_FALSE = 0,
    D3DZB_TRUE = 1,
    D3DZB_USEW = 2,
    D3DZB_FORCE_DWORD = 0x7fffffff
} D3DZBUFFERTYPE;
```

Listing 30 – *D3DZBUFFERTYPE*, Źródło *DirectX SDK*

- *D3DRS_ZENABLE* włącza używanie *Z Bufora*.
- *D3DRS_FILLMODE* może przyjmować wartości przedstawione na listingu 45.

```
typedef enum _D3DFILLMODE {
    D3DFILL_POINT = 1,
    D3DFILL_WIREFRAME = 2,
    D3DFILL_SOLID = 3,
    D3DFILL_FORCE_DWORD = 0x7fffffff
} D3DFILLMODE;
```

Listing 31 – *D3DFILLMODE*, Źródło *DirectX SDK*

D3DRS_FILLMODE włącza tryb wypełniania trójkątów, gdzie *Solid* – wyjściowy – normalne rysowanie trójkątów. *WireFrame* – tylko linie, *Point* – same punkty.

- *D3DRS_SHADEMODE* może przyjmować wartości przedstawione na listingu 46.

```
typedef enum _D3DSHADEMODE {
    D3DSHADE_FLAT = 1,
    D3DSHADE_GOURAUD = 2,
    D3DSHADE_PHONG = 3,
    D3DSHADE_FORCE_DWORD = 0x7fffffff
} D3DSHADEMODE;
```

Listing 32 – *D3DSHADEMODE*, Źródło *DirectX SDK*

D3DRS_SHADEMODE ustawia tryb interpolowania koloru między vertexami., gdzie *Flat* – jednym kolorem, *Gouraud* – interpolowanie liniowe między kolorami w vertexach. *Phong* – nie używane.

- *D3DRS_ZWRITEENABLE*

  Włącza / wyłącza zapisywanie do bufora *Z*, przyjmuje wartości *TRUE / FALSE.*
- *D3DRS_SRCBLEND*, *D3DRS_DESTBLEND* mogą przyjmować wartości przedstawione na listingu 47.

```
typedef enum _D3DBLEND {
    D3DBLEND_ZERO = 1,
    D3DBLEND_ONE = 2,
    D3DBLEND_SRCCOLOR = 3,
    D3DBLEND_INVSRCCOLOR = 4,
    D3DBLEND_SRCALPHA = 5,
    D3DBLEND_INVSRCALPHA = 6,
```

```
    D3DBLEND_DESTALPHA = 7,
    D3DBLEND_INVDESTALPHA = 8,
    D3DBLEND_DESTCOLOR = 9,
    D3DBLEND_INVDESTCOLOR = 10,
    D3DBLEND_SRCALPHASAT = 11,
    D3DBLEND_BOTHSRCALPHA = 12,
    D3DBLEND_BOTHINVSRCALPHA = 13,
    D3DBLEND_BLENDFACTOR = 14,
    D3DBLEND_INVBLENDFACTOR = 15,
    D3DBLEND_FORCE_DWORD = 0x7fffffff
} D3DBLEND;
```

Listing 33 – *D3DBLEND*, Źródło *DirectX SDK*

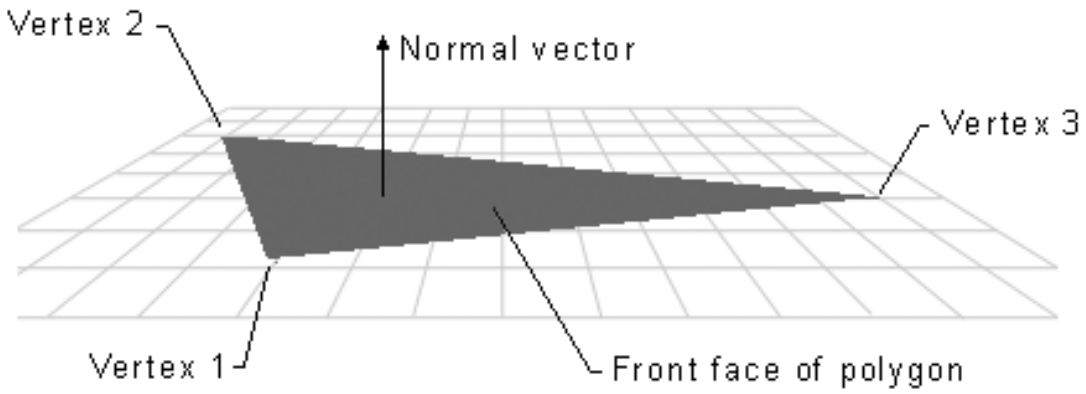
D3DRS_SRCBLEND, *D3DRS_DESTBLEND* ustawiają tryb pracy operacji łączenia kolorów, gdzie *Src* – piksel źródłowy, *Dest* – piksel docelowy. Możemy sprawdzić w *D3DCAPS9* jakie operacje są obsługiwane przez urządzenie.

- *D3DRS_CULLMODE* może przyjmować wartości przestawione na listingu 48.

```
typedef enum _D3DCULL {
    D3DCULL_NONE = 1,
    D3DCULL_CW = 2,
    D3DCULL_CCW = 3,
    D3DCULL_FORCE_DWORD = 0x7fffffff
} D3DCULL;
```

Listing 34 – *D3DCULL*, Źródło *DirectX SDK*

D3DRS_CULLMODE ustawia możliwość odrzucania trójkątów które są odwrócone tyłem do kamery. Zazwyczaj wszystkie trójkąty podawane są w kolejność – zgodnie z kirunkiem wskazówek zegara (*CW*), albo przeciw kierunkowi wskazówek zegara (*CCW*). Sytuacja przedstawiona jest na rysunku 26.



Rysunek 1 - znajdowanie trójkąta odwróconego przodem, Źródło *DirectX SDK*

Jeżeli podamy najpierw *Vertex1*, potem *2*, a potem *3*, to będzie to zgodnie z kierunkiem wskazówek zegara. Zatem definicja 1 – *Cullingiem* nazywamy operacje odrzucania trójkątów odwróconych tyłem do obserwatora (kamery). Taka operacja pozwala zaoszczędzić czas potrzebny do *wyrenderowania* efektu.

- *D3DRS_ZFUNC* może przyjmować wartości przedstawione na listingu 50.

```
typedef enum _D3DCMPFUNC {
    D3DCMP_NEVER = 1,
    D3DCMP_LESS = 2,
    D3DCMP_EQUAL = 3,
    D3DCMP_LESSEQUAL = 4,
    D3DCMP_GREATER = 5,
    D3DCMP_NOTEQUAL = 6,
    D3DCMP_GREATEREQUAL = 7,
    D3DCMP_ALWAYS = 8,
    D3DCMP_FORCE_DWORD = 0x7fffffff
} D3DCMPFUNC;
```

Listing 35 – *D3DCMPFUNC*, Źródło *DirectX SDK*

D3DRS_ZFUNC odpowiada za tryb porównywania wartości *Z*. (zobacz *D3DCAPS9*).

- *D3DRS_DITHERENABLE*

  Przyjmuje wartości *TRUE, FALSE*. Polepszenie jakości obrazu w trybach o małej ilości kolorów np. *16bpp*.

- *D3DRS_SPECULARENABLE*

  Przyjmuje wartości *TRUE, FALSE*. Włącza albo wyłącza liczenie rozbłysków na obiektach. Zazwyczaj wyłączone gdyż jest to trochę kosztowna operacja.

- *D3DRS_CLIPPING*

  Przyjmuje wartości *TRUE, FALSE*. Włącza albo wyłącza obcinanie trójkątów.

- *D3DRS_LIGHTING*

  Przyjmuje wartości *TRUE, FALSE*. Włącza albo wyłącza oświetlanie.

- *D3DRS_AMBIENT*

  Ustawia wartość światła otaczającego.

- *D3DRS_NORMALIZENORMALS*

Przyjmuje wartości *TRUE, FALSE*. Włącza albo wyłącza normalizowanie wektorów normalnych. W czasie transformacji może się zdarzyć, że normalne przestaną być z zakresu 0 – 1 np. przez skalowanie obiektu. Wtedy, żeby np. oświetlenie dawało poprawne wyniki należy znormalizować normalne. Operacja „za darmo" na kartach ze sprzętowym *TnL*.

- *D3DRS_BLENDOP* może przyjmować wartości przedstawione na listingu 51.

```
typedef enum _D3DBLENDOP {
    D3DBLENDOP_ADD = 1,
    D3DBLENDOP_SUBTRACT = 2,
    D3DBLENDOP_REVSUBTRACT = 3,
    D3DBLENDOP_MIN = 4,
    D3DBLENDOP_MAX = 5,
    D3DBLENDOP_FORCE_DWORD = 0x7fffffff
} D3DBLENDOP;
```

Listing 36 – *D3DBLENDOP*, Źródło *DirectX SDK*

D3DRS_BLENDOP ustawia operacje w czasie łączenia operacji kolorów

- *D3DRS_ALPHABLENDENABLE*

Przyjmuje wartości *TRUE, FALSE*. Włącza albo wyłącza operacje łączenia kolorów.

3.3 Format *Vertexów. Vertex Buffer.*

Kiedy już mamy zainicjalizowane urządzenie i ustawione *renderstate'y* należy przygotować geometrie do renderowania.

Musimy określić format naszego vertexa oraz ustawić flagi *FVF*. Przykład podany na listingu 52.

```
struct D3DVERTEX
{
    D3DXVECTOR3 p;
    D3DXVECTOR3 n;
    FLOAT       tu, tv;

    static const DWORD FVF;
};
const DWORD D3DVERTEX::FVF = D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1;
```

Listing 37 – *FVF*, Źródło *DirectX SDK*

Oznaczenia flag *FVF*:

- *D3DFVF_XYZ* – pozycja w formacie wektora x, y, z, gdzie każdy element to *float*,
- *D3DFVF_NORMAL* – wektor normalny w formacie wektora *x*, *y*, *z*, gdzie każdy element to *float*,
- *D3DFVF_TEX0 – TEX8* – współrzędne tekstury,
- *D3DFVF_TEXTUREFORMAT1 – 4* – ile *floatów* opisuje współrzędne.

Są to tylko podstawowe flagi, których będziemy używać. Kolejność flag nie jest przypadkowa. Dokładnie kolejność danych, jaka ma być w strukturze, podano w tabeli w poprzednim rozdziale. Teraz musimy stworzyć vertex bufor który będzie trzymał geometrie: metoda *IDirect3D9Device::CreateVertexBuffer* przedstawiona na listingu 53.

```
HRESULT CreateVertexBuffer(
    UINT Length,
    DWORD Usage,
    DWORD FVF,
    D3DPOOL Pool,
    IDirect3DVertexBuffer9** ppVertexBuffer,
    HANDLE* pHandle
);
```

Listing 38 – *IDirect3DDevice9::CreateVertexBuffer*, Źródło *DirectX SDK*

Gdzie:

- *Lenght* to wielkość bufora w bajtach,
- *Usage*:
    - *D3DUSAGE_SOFTWAREPROCESSING* – używane kiedy, będziemy korzystać z *TnL* programowego
    - *D3DUSAGE_DYNAMIC* – tworzy bufor dynamiczny, którego zawartość możemy zmieniać kiedy chcemy
    - *D3DUSAGE_WRITEONLY* – tworzy bufor tylko z możliwością zapisu – najlepsza opcja dla kart ze sprzętowym *TnL*
- *FVF* to kombinacja flag *FVF* opisująca *vertex*,
- *Pool*:

```
typedef enum _D3DPOOL {
    D3DPOOL_DEFAULT = 0,
    D3DPOOL_MANAGED = 1,
    D3DPOOL_SYSTEMMEM = 2,
    D3DPOOL_SCRATCH = 3,
```

```
    D3DPOOL_FORCE_DWORD = 0x7fffffff
} D3DPOOL;
```

Listing 39 – *D3DPOOL*, Źródło *DirectX SDK*

Default – utworzy bufor w pamięci karty graficznej w przypadku sprzętowego *TnL* (zalecane dla dynamicznych buforów), *Managed* – zleca *D3D* zarządzanie buforem (tego będziemy używac), *Systemmem* – bufor ma być umieszczony w pamięci *RAM*.

Następny parametr to wskaźnik pod który będzie zapisany interfejs do bufora.

- Ostatni parametr musi być *NULL*.

Przykład zastosowania metody *IDirect3D9Device::CreateVertexBuffer* przedstawiono na listingu 55.

```
if ( m_pD3DRenderer->GetCurrentDevice()->HasDeviceHardwareTnL() )
{
    if ( FAILED( hr == GetD3DDevice()->CreateVertexBuffer( dwSize *
           m_dwNumberOfVertices, D3DUSAGE_WRITEONLY, NULL,
           D3DPOOL_MANAGED, &pBuffer->ppStreams[ dwStream ]->pVB,
           NULL) ) )   AnErrorMsg("Can't Create Vertex Buffer”);
}
else
{
    if ( FAILED( hr == GetD3DDevice()->CreateVertexBuffer( dwSize *
           m_dwNumberOfVertices, D3DUSAGE_SOFTWAREPROCESSING, NULL,
           D3DPOOL_SYSTEMMEM, &pBuffer->ppStreams[ dwStream ]->pVB,
           NULL) ) )   AnErrorMsg("Can't Create Vertex Buffer”);
}
```

Listing 40 – Przykład tworzenia Vertex Buffera, Źródło własne

Aby zapisać coś do bufora musimy go *zalockować* (o *lockowaniu surface'ów*, zobacz poprzedni rozdział), służy do tego metoda *IDirect3DvertexBuffer9::Lock* przedstawiona na listingu 56.

```
HRESULT Lock(
    UINT OffsetToLock,
    UINT SizeToLock,
    VOID **ppbData,
    DWORD Flags
);
```

Listing 41 – *IDirect3DVertexBuffer9::Lock*, Źródło *DirectX SDK*

Pierwszy parametr to początek obszaru do *zalockowania*. Drugi to rozmiar, może być *NULL* – wtedy cały bufor będzie *zalockowany*. Trzeci to wskaźnik pod który będziemy

mogli zapisywać *vertexy* tak, jakbyśmy pisali do zwykłej tablicy *vertexów*. Czwarty to flagi (nie będziemy ich używać). Po wykonaniu zapisu należy bufor *zunlockować* służy do tego metoda *IDirect3DVertexBuffer8::Unlock* przedstawiona na listingu 57.

```
HRESULT Unlock(VOID);
```

Listing 42 – *IDirect3DVertexBuffer9::Unlock*, Źródło *DirectX SDK*

Przykład zastosowania znajduje się na listingu 58.

```
m_pDolphinVB1->Lock( 0, 0, (void**)&pDst, 0 );
pMeshSourceVB->Lock( 0, 0, (void**)&pSrc, 0 );
memcpy( pDst, pSrc, m_dwNumDolphinVertices * sizeof(D3DVERTEX) );
m_pDolphinVB1->Unlock();
pMeshSourceVB->Unlock();
```

Listing 43 – Przykład *lockowania* i *unlockowania* bufora, Źródło *DirectX SDK*

Teraz bufor jest gotowy do używania.

3.4 *Indeksy* i *Index Buffer*

Z index buforem i indexami postępujemy podobnie jak z *vertex* buforami: *IDirect3DDevice9::CreateIndexBuffer*

```
HRESULT CreateIndexBuffer(
    UINT Length,
    DWORD Usage,
    D3DFORMAT Format,
    D3DPOOL Pool,
    IDirect3DIndexBuffer9** ppIndexBuffer,
    HANDLE* pHandle
);
```

Listing 44 – *IDirect3DDevice9::CreateIndexBuffer*, Źródło *DirectX SDK*

Format zazwyczaj podajemy *D3DFMT_INDEX16*. Większość kart trzyma indexy jako 16 bitowe liczby całkowite. Tak więc po zalockowaniu który posiada wywołanie identyczne jak w przypadku *vertex* bufora, zapisujemy kolejne *indeksy*, jak do tablicy o elementach typu *short – 16 bitowa* liczba całkowita bez znaku. *Lockowanie* i *unlockowanie index* buffera odbywa się dokładnie na tej samej zasadzie, jak w przypadku *vertex* buffera. Teraz, aby narysować obiekt, musimy ustawić *vertex* bufor przy pomocy metody *IDirect3DDevice9::SetStreamSource* przedstawionej na listingu 60.

```
HRESULT SetStreamSource(
    UINT StreamNumber,
```

```
    IDirect3DVertexBuffer9 *pStreamData,
    UINT OffsetInBytes,
    UINT Stride
);
```

Listing 45 – *IDirect3DDevice9::SetStreamSource*, Źródło *DirectX SDK*

Stream to strumień *vertexów*. Takich strudmieni może być wiele. My zajmiemy się przypadkami kiedy występuje jeden strumień, czyli w pierwszym parametrze będziemy podawać 0. Następny parametr to *vertex buffer* z którego będziemy chcieli korzystać. Trzeci parametr to *offset* w bajtach względem początku *vertex buffera*, od którego *D3D* ma zacząć liczyć *vertexy*. Zazwyczaj będziemy ustawiać go na 0. Ostatni - to rozmiar w bajtach jednego *vertexa*. Przykład zastosowania znajduje się na listingu 61.

```
if ( FAILED( hr == m_pD3DRenderer->GetD3DDevice()->SetStreamSource( i,
            m_pActiveVB->ppStreams[i]->pVB, NULL,
            m_pActiveVB->ppStreams[i]->m_dwStride ) ) )
      m_pDebugger->Log("","ERROR! - SetStreamSource");
```

Listing 46 – Przykład użycia *SetStreamSource*, Źródło własne

Po ustawieniu bufora musimy poinformować *D3D* jaki format mają dane w buforze, służy do tego metoda *IDirect3DDevice9::SetFVF* przedstawiona na listingu 62.

```
HRESULT SetFVF(
    DWORD FVF
);
```

Listing 47 – *IDirect3DDevice9::SetFVF*, Źródło *DirectX SDK*

Jako parametr podajemy flagi *FVF* które opisują *vertex*. Następnie musimy ustawić index buffer przy pomocy metody *IDirect3Ddevic9::SetIndices* przedstawionej na listingu 63.

```
HRESULT SetIndices(
    IDirect3DIndexBuffer9 *pIndexData
);
```

Listing 48 – *IDirect3DDevice9::SetIndices*, Źródło *DirectX SDK*

Parametrem metody *IDirect3Ddevic9::SetIndices* jest wskaźnik do *index buffera*. Przykład znajduje się na listingu 64.

```
m_pD3DRenderer->GetD3DDevice()->SetIndices( pBuffer );
```

Listing 49 – Ustawienie bufora z indeksami, Źródło własne

Gdy już mamy ustawione bufory, musimy ustawić macierze transformacji; będziemy wyróżniać 4 typy transformacji:

D3DTS_VIEW – macierz widoku – macierz kamery. czyli macierz świat_do_kamery,

D3DTS_PROJECTION – macierz rzutowania,

D3DTS_WORLD – macierz obiekt_do_świata,

D3DTS_TEXTUREn – macierz przekształcająca współrzędne tekstury.

Ustawiamy macierze przy pomocy metody *IDirect3DDevice::SetTransform* przedstawionej na listingu 65.

```
HRESULT SetTransform(
    D3DTRANSFORMSTATETYPE State,
    CONST D3DMATRIX *pMatrix
);
```

Listing 50 – *IDirect3DDevice9::SetTransform*, Źródło *DirectX SDK*

Pierwszy parametr metody *IDirect3DDevice::SetTransform* to typ macierzy – czyli jedna z flag wymienionych powyżej jako typ transformacji. Drugi parametr to wskaźnik do macierzy. Przykład znajduje się na listingu 66.

```
m_p3dDevice->SetTransform( D3DTS_WORLD, &matWorld );
```

Listing 51 – Zastosowanie *SetTransform*, Źródło własne

Teraz możemy wywołać rysowanie używając metody *Direct3DDevice9::DrawIndexedPrimitive* podanej na listingu 67.

```
HRESULT DrawIndexedPrimitive(
    D3DPRIMITIVETYPE Type,
    INT BaseVertexIndex,
    UINT MinIndex,
    UINT NumVertices,
    UINT StartIndex,
    UINT PrimitiveCount
);
```

Listing 52 – *IDirect3DDevice9::DrawIndexedPrimitive*, Źródło *DirectX SDK*

Pierwszy parametr metody *Direct3DDevice9::DrawIndexedPrimitive* to typ rysowanych obiektów. Typy obiektów podane są na listingu 68.

```
typedef enum _D3DPRIMITIVETYPE {
```

```
    D3DPT_POINTLIST = 1,
    D3DPT_LINELIST = 2,
    D3DPT_LINESTRIP = 3,
    D3DPT_TRIANGLELIST = 4,
    D3DPT_TRIANGLESTRIP = 5,
    D3DPT_TRIANGLEFAN = 6,
    D3DPT_FORCE_DWORD = 0x7fffffff
} D3DPRIMITIVETYPE;
```

Listing 53 – *D3DPRIMITIVETYPE*, Źródło *DirectX SDK*

Omówiliśmy te typy na początku naszych rozważań o *D3D*. Zazwyczaj będziemy używać *TRIANGLELIST*. Drugi parametr *Direct3DDevice9::DrawIndexedPrimitive* to numer *indexa* w buforze *indexów*, który jest pierwszym *indexem* (zazwyczaj będziemy ustawiać na *0*). Trzeci parametr to numer *vertex* użytego do rysowania, np. jeżeli mamy *1000 vertexów*, a chcemy wyrenderować *10*, to w tym parametrze podajemy od którego *vertexa* zaczyna się te *10 vertexów*. W następnym parametrze podajemy ile *vertexów* chcemy *wyrenderować*. Czwarty parametr informuje od którego *indexa* w *index bufferze* ma zacząć rysowanie. Ostatni parametr podaje ile trójkątów ma *wyrenderować*. Zazwyczaj jest to liczba indeksów podzielona przez trzy. Przykład zastosowania znajduje sie na listingu 69.

```
if ( FAILED( hr == GetD3DDevice()->DrawIndexedPrimitive(
            D3DPT_TRIANGLELIST, iBaseVertexIndex, iMinIndex,
            iNumVertices, iStartIndex, iPrimitiveCount ) ) )
      m_pDebugger->Log("","ERROR! - DrawIndexedPrimitive\n");
```

Listing 54 – Użycie *DrawIndexedPrimitive*, Źródło własne