# 4. Praca z *Direct3D*

W tym rozdziale będziemy się zajmować kolejnymi coraz bardziej zaawansowanymi elementami biblioteki *D3D* takimi jak oświetlanie i *teksturowanie*.

## 4.1 Oświetlanie

W każdej scenie *3D* występuje tzw. *światło otoczenia*. Znane jest ono w *Direct3D* jako *ambient light*. Przykład ustawiania *ambient light* podany jest na listingu 70.

```
SetRenderState( D3DRS_AMBIENT, 0x000000 );
```

**Listing 1 –** Ustawienie światła ambient na 0 - czarny, Źródło własne

Do dyspozycji mamy *8 akcelerowanych* sprzętowo (w przypadku kart ze sprzętowym *TnL*) świateł. Możemy użyć dowolnej liczbę, ale nie przekraczającą ośmiu równocześnie używanych świateł. Światła *kierunkowe* są *najszybsze*. *Najwolniejsze* są światła *SPOT*. Aby włączyć liczenie oświetlania:

```
SetRenderState( D3DRS_LIGHTING, TRUE );
```

**Listing 2 –** Włączenie przeliczania oświetlenia, Źródło własne

Teraz możemy przygotować struktury opisujące każde światło. Struktura *D3DLIGHT9* podana jest na listingu 72.

```
typedef struct _D3DLIGHT9 {
    D3DLIGHTTYPE Type;
    D3DCOLORVALUE Diffuse;
    D3DCOLORVALUE Specular;
    D3DCOLORVALUE Ambient;
    D3DVECTOR Position;
    D3DVECTOR Direction;
    float Range;
    float Falloff;
    float Attenuation0;
    float Attenuation1;
    float Attenuation2;
    float Theta;
    float Phi;
} D3DLIGHT9;
```

**Listing 3 –** *D3DLIGHT9*, Źródło *DirectX SDK*

- *Type* – to typ światła, *D3D* posiada *3* typy światła: *DIRECTIONAL* – kierunkowe, np. słońce padające na ziemie, *POINT* – punktowe, np. światło żarówki rozchodzące się we wszystkich kierunkach, *SPOT* – światło miejscowe, np. światło latarki,
- *Diffuse* – współczynnik *Diffuse* światła, zazwyczaj używa się go do określenia mocy światła, w czasie liczenia natężenia światła przemnażany jest przez wartość *Diffuse* materiału,
- *Ambient* – współczynnik *Ambient* światła, nie używa się go, zostawiając ten współczynnik dla światła otaczającego występującego w scenie,
- *Specular* – wartość *specular* światła, przemnażana przez wartość specular materiału w czasie liczenia oświetlenia,
- *Position* – aktualna pozycja światła (nie dotyczy świateł *DIRECTIONAL*),
- *Direction* – kierunek padania światła (nie dotyczy świateł *POINT*),
- *Range* – zasięg światła,
- *FallOff* – określa zanik światła między kątami *Theta* i *Phi* w świetle *SPOT*,
- *Atteniation0, 1, 2* – parametry zaniku światła wraz z odległością, pierwszy – stały, drugi – liniowy, trzeci – kwadratowy,
- *Theta* – w radianach, określa stożek wewnętrzny światła *SPOT,*
- *Phi* – w radianach, określa zewnętrzny światła *SPOT*.

Gdy już mamy przygotowane opisy świateł, należy włączyć kolejne światło i przekazać *D3D* właściwość tego światła, służy do tego metoda *IDirect3DDevice9::LightEnable* przedstawiona na listingu 73.

```
HRESULT LightEnable(
    DWORD LightIndex,
    BOOL bEnable
);
```

**Listing 4 –** *IDirect3DDevice9::LightEnable*, Źródło DirectX SDK

Pierwszy parametr metody *IDirect3DDevice9::LightEnable* to numer światła, od zera do siedmiu. Następny to stan: *TRUE* / *FALSE*. *TRUE* – włączone światło, *FALSE* – wyłączone. Przykład zastosowania znajduje się na listingu 74.

```
if ( D3D_OK != m_p3dDevice->LightEnable( wLight, bFlag ) )
      m_pDebugger->Log("","ERROR!: LIGHTENABLE\n");
```

**Listing 5 –** Użycie *LightEnable*, Źródło własne

Meotda *IDirect3DDevice::SetLight* przekazuje *D3D* strukturę opisującą światło. Przedstawiona jest na listingu 75.

```
HRESULT SetLight(
    DWORD Index,
    CONST D3DLIGHT9 *pLight
);
```

**Listing 6 –** *IDirect3DDevice9::SetLight*, Źródło *DirectX SDK*

Pierwszy parametr meotdy *IDirect3DDevice::SetLight* to numer światła, drugi to wskaźnik do struktury *D3DLIGHT9*. Aby oświetlanie zadziałało, musimy zdefiniować materiał i przypisać go obiektom. Ponownie tym będzie struktura *D3DMATERIAL9* przedstawiona na listingu 76.

```
typedef struct _D3DMATERIAL9 {
        D3DCOLORVALUE Diffuse;
        D3DCOLORVALUE Ambient;
        D3DCOLORVALUE Specular;
        D3DCOLORVALUE Emissive;
        float Power;
} D3DMATERIAL9;
```

**Listing 7 –** *D3DMATERIAL9*, Źródło *DirectX SDK*

gdzie *D3DCOLORVALUE* przedstawione jest na listingu 77.

```
typedef struct _D3DCOLORVALUE {
    float r;
    float g;
    float b;
    float a;
} D3DCOLORVALUE;
```

**Listing 8 –** *D3DCOLORVALUE*, Źródło *DirectX SDK*

*Ambient* to współczynnik odbijania światła otaczającego. *Diffuse* to współczynnik odbijania światła z innych świateł. *Specular* określa stopień odbijania rozbłysków światła. *Power* określa moc tych rozbłysków. *Emissive* określa stopień emisji światła. Zazwyczaj większość materiałów ma *Ambient* i *Diffuse* ustawione na *1.0, 1.0, 1.0, 1.0* i kolor światła przemnażany przez kolor tekstury. Wtedy tekstura „decyduje” o stopniu odbijania światła. Do ustawienia materiału używamy metody *IDirect3DDevice9::SetMaterial* przedstawionej na listingu 78.

```
HRESULT SetMaterial(
    CONST D3DMATERIAL9 *pMaterial
);
```

**Listing 9 –** *IDirect3DDevice9::SetMaterial*, Źródło *DirectX SDK*

Parametrem metody *IDirect3DDevice9::SetMaterial* jest wskaźnik do materiału. Przykład zastosowania znajduje się na listingu 79.

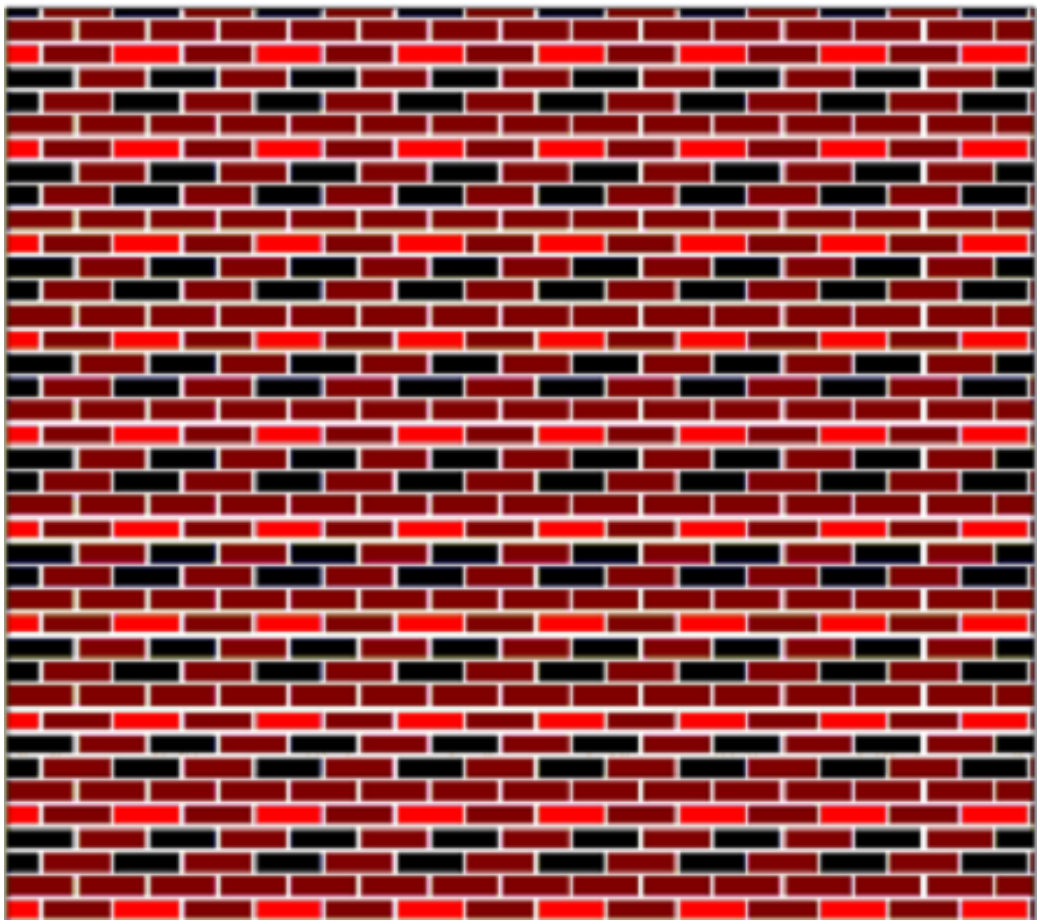
```
HRESULT hr = m_pD3DRenderer->GetD3DDevice()->SetMaterial( pMaterial );
```

**Listing 10 –** Ustawienie materiału, Źródło własne

## 4.2 Teksturowanie

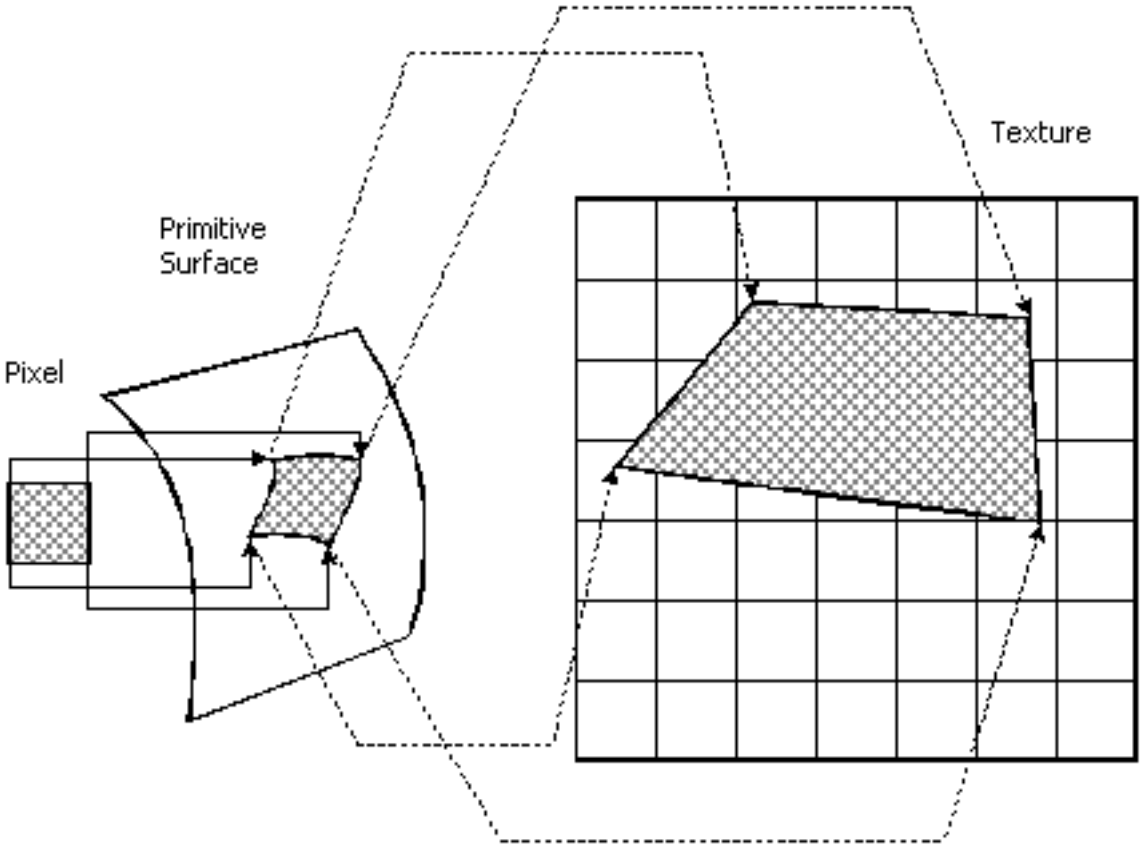
W tym rozdziale opiszemy co trzeba uczynić, aby na nasze trówymiarowe obiekty nałożyć *teksturę*, czyli płaski dwuwymiarowy obraz.

Definicja 1 - *tekstura* jest to obraz *2D*, który przy pomocy operacji matematycznej zostanie nałożony na obiekt *3D*. Przykład tekstury znajduje się na ilustracji 8.



**Ilustracja 1** - Przykład tekstury, Żródło *DirectX SDK*

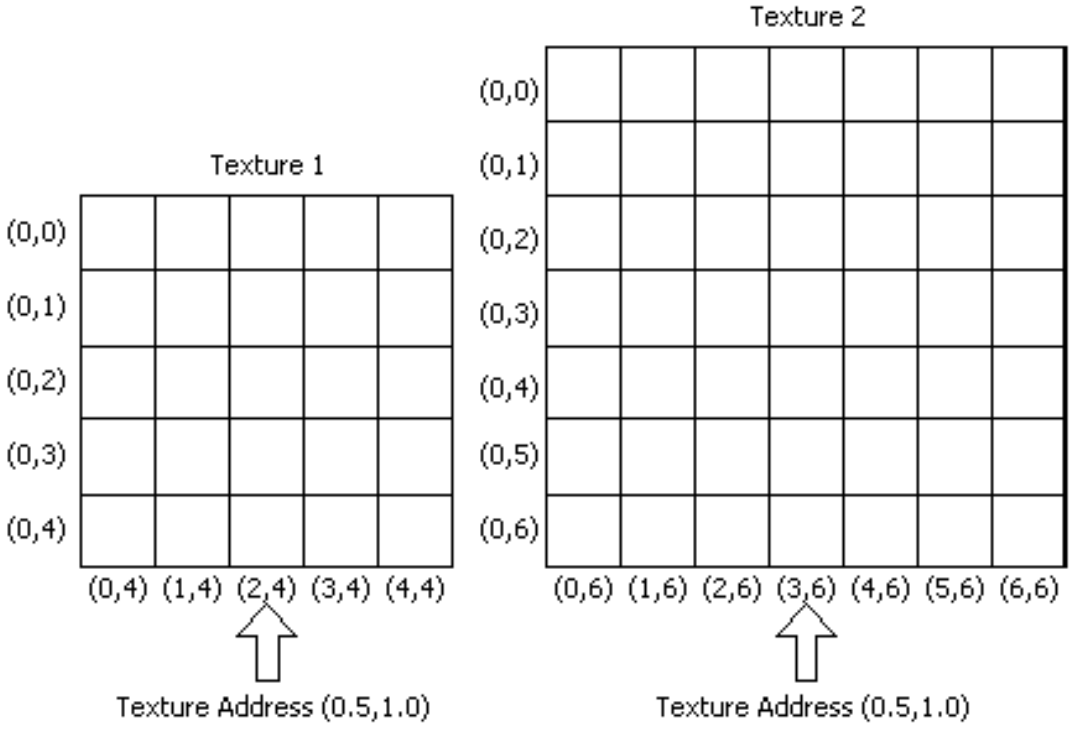
Rysnuek 27 przestawia nakładanie płaskiego obrazu na przestrzenną figurę.




**Rysunek 1 -** Schemat nakładania tekstury *2D* na obiekt *3D*, Źródło *DirectX SDK*

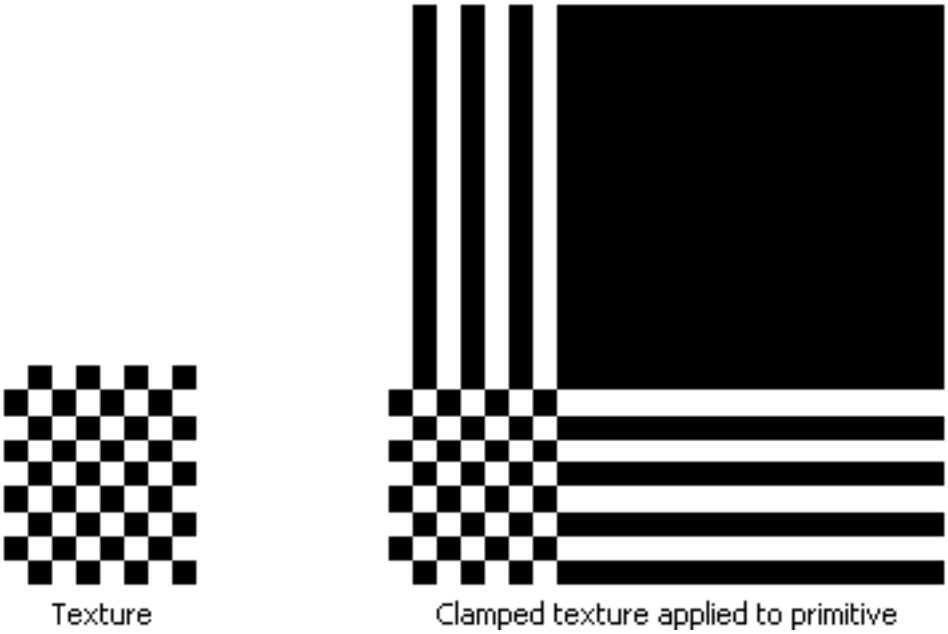
Definicja 2 - *Adresowanie. Współrzędne tekstury.* Tekstura jest zazwyczaj dwuwymiarowa. Lewy górny róg tekstury to punkt *0,0*. Natomiast prawy dolny to *1,1*. Współrzędne tekstury nazywają się *U* i *V*. *U* jest pozioma, *V* pionowa. W zależności od ustawień typu adresowania *U* i *V* mniejsze od zera albo większe od jedynki będą albo przyjmować najbliższą wartość z przedziału zero – jeden (*CLAMP*), albo tekstura będzie zawijana (*WRAP*). Tę opcje ustawia się przy pomocy metody *IDirect3DDevice::SetTextureStageState*. Rysunek 28 przedstawia różnicie między adresowaniem logicznym a fizycznym *tekstury*. *Teksturę* adresujemy używając współrzędnych logicznych, np. *(0.5, 1.0)*. Karta graficzna sama pobieże odpowiedni

*teksel* zamieniając współrzędne logiczne na fizyczne biorąc po uwagę rozmiar *samplowanej tekstury*.




**Rysunek 2 -** Adresowanie tekstury, Źródło *DirectX SDK*

Rysunek 29 przedstawia sposób adresowania *tekstury* przy użyciu trybu *CLAMP*.




**Rysunek 3** - Tryb *CLAMP* adresowania tekstury, Źródło *DirectX SDK*

Rysunek 30 przedstawia sposób adresowania *tekstury* przy użyciu trybu *WRAP*:

**Rysunek 4** - Tryb *WRAP* adresowania tekstury, Źródło *DirectX SDK*

*Filtrowanie tekstur.*

- *Nearest-Point Sampling* – filtrowanie (w momencie, kiedy *rasterizer* będzie chciał pobrać np. *teksel 0.5*, *0.5*, pobierze najbliższy sąsiadujący *teksel*),
- *Linear Filtering – bilinear filtering* – filtrowanie (w momencie, kiedy *rasterizer* będzie chciał pobrać np. *teksel 0.5*, *0.5*, wartość koloru zostanie uśredniona z otaczających *tekseli*),
- *Anisotropic Filtering* – jest to najbardziej zaawansowane filtrowanie polegające na uwzględnieniu konta nachylenia trójkąta do obserwatora.
- *Texture Filtering With Mipmaps – Trilinear filtering* – kolor zostaje nie tylko uśredniony z okolicznych tekseli, ale także z dwóch najbliższych *mipmap*.

Definicja 3 - *mipmapy*. *Mipmapy* to zestaw tekstur o różnych rozmiarach, od wielkości wyjściowej aż po teksturę *1x1* dzieląc wymiary tekstury przez *2* czyli np. *256x256, 128x128, 64x64* itd. *Mipmapy* stosuje się ponieważ trójkąt wraz ze zwiększaniem się kąta patrzenia na niego powinien tracić szczegóły tekstury. W momencie kiedy nakładamy jedną dużą teksturę, przybliżanie wartości tekseli nie daje zadowalających rezultatów. Często można zauważyć efekt „chodzących mrówek" albo „pływającej tekstury". Po użyciu mipmap obraz wygląda zadowalająco. Przykład mipmap znajduje się na rysunku 31.


**Rysunek 5** - Przykład *mipmap*, Źródło *DirectX SDK*

Aby utworzyć teksturę, należy użyć metody *IDirect3DDevice9::CreateTexture* podanej na listingu 80.

```
HRESULT CreateTexture(
    UINT Width,
    UINT Height,
    UINT Levels,
    DWORD Usage,
    D3DFORMAT Format,
    D3DPOOL Pool,
    IDirect3DTexture9** ppTexture,
    HANDLE* pHandle
);
```

**Listing 11 –** *IDirect3DDevice9::CreateTexture*, Źródło *DirectX SDK*

*Width* – szerokość tekstury, *Height* – wysokość. *Levels* – ilość mipmap, podanie *0* powoduje zbudowanie wszystkich poziomów mipmap od wartości zadanych aż do *1x1*.

*Usage* to jedna z flag:

- *D3DUSAGE_DEPTHSTENCIL*
- *D3DUSAGE_RENDERTARGET*

- *D3DUSAGE_DYNAMIC*
- *D3DUSAGE_AUTOGENMIPMAP*

albo *NULL*. Format to wybrany format koloru tekstury. *Pool* przedstawiony jest na listingu 81.

```
typedef enum _D3DPOOL {
    D3DPOOL_DEFAULT = 0,
    D3DPOOL_MANAGED = 1,
    D3DPOOL_SYSTEMMEM = 2,
    D3DPOOL_SCRATCH = 3,
    D3DPOOL_FORCE_DWORD = 0x7fffffff
} D3DPOOL;
```

**Listing 12 –** *D3DPOOL*, Źródło *DirectX SDK*

Ostatni parametr to wskaźnik do interfejsu tekstury. Dużo częściej będziemy korzystać z funkcji która czyta plik graficzny z dysku i automatycznie tworzy teksturę o zadanych parametrach – funkcja *D3DXCreateTextureFromFileEx* przedstawiona na listingu 82.

```
HRESULT D3DXCreateTextureFromFileEx(
    LPDIRECT3DDEVICE9 pDevice,
    LPCTSTR pSrcFile,
    UINT Width,
    UINT Height,
    UINT MipLevels,
    DWORD Usage,
    D3DFORMAT Format,
    D3DPOOL Pool,
    DWORD Filter,
    DWORD MipFilter,
    D3DCOLOR ColorKey,
    D3DXIMAGE_INFO *pSrcInfo,
    PALETTEENTRY *pPalette,
   LPDIRECT3DTEXTURE9 *ppTexture
);
```

**Listing 13 –** *D3DXCreateTextureFromFileEx*, Źródło *DirectX SDK*

- *pDevice* - to wskaźnik do interfejsu do karty graficznej dla której ma być stworzona tekstura,
- *pSrcFile* – to nazwa pliku,
- *Width* – żądana szerokość tekstury albo *D3DX_DEFAULT*, jeżeli wartość ta ma być przeczytana z pliku,
- *Height* - żądana wysokość tekstury albo *D3DX_DEFAULT*, jeżeli wartość ta ma być przeczytana z pliku,

- *MipLevels* – poziom mipmap, albo *D3DX_DEFAULT*, dla kompletnego łańcucha mipmap,
- *Usage* – kombinacja flag przedstawionych przy *IDirect3DDevice9::CreateTexture*,
- *Pool* – to samo znaczenie jak w *IDirect3DDevice9::CreateTexture*,
- *Filter* – rodzaj filtrowania: zalecane *D3DX_DEFAULT*,
- *MipFilter* – rodzaj filtrowania w czasie budowania mipmap: zalecane *D3DX_DEFAULT*,
- *ColorKey* – nie używane – *NULL*,
- *pSrcInfo* – wskaźnik do struktury *D3DXIMAGE_INFO* która otrzyma dane o pliku wejściowym,
- *pPalette* – paleta – *NULL*,
- *ppTexture* – wskaźnik który otrzyma interfejs do tekstury.

*D3DXIMAGE_INFO* znajduje się na listingu 83.

```
typedef struct _D3DXIMAGE_INFO {
    UINT Width;
    UINT Height;
    UINT Depth;
    UINT MipLevels;
    D3DFORMAT Format;
    D3DRESOURCETYPE ResourceType;
    D3DXIMAGE_FILEFORMAT ImageFileFormat;
} D3DXIMAGE_INFO;
```

**Listing 14 –** *D3DXIMAGE_INFO*, Źródło *DirectX SDK*

- *Width* – szerokość obrazu,
- *Height* – wysokość obrazu,
- *Depth* – głębokość koloru w bitach,
- *MipLevels* – ilość mipmap,
- *Format – D3DFORMAT* opisujący kolor obrazu,
- *ResourceType* – typ zasobu: *D3DRTYPE_TEXTURE*, *D3DRTYPE_CUBETEXTURE* ,
- ImageFileFormat – format wczytanego pliku, np.: *jpg*, *png*, *tga*.

Kiedy już mamy utworzone tekstury i chcemy je ustawić: użyjemy metody *IDirect3DDevice9::SetTexture* przedstawionej na listingu 84.

```
HRESULT SetTexture(
    DWORD Stage,
    IDirect3DBaseTexture9 *pTexture
```

```
);
```

**Listing 15 –** *IDirect3DDevice9::SetTexture*, Źródło *DirectX SDK*

Pierwszy parametr metody *IDirect3DDevice9::SetTexture* opisuje pod który *stage* chcemy podłączyć *teksturę*. Drugi parametr to wskaźnik do tekstury. S*tage* to numer tekstury. Każda nowoczesna karta graficzna ma *multitexturing*, czyli możliwość nakładania kilku tekstur na raz. *Stage* to właśnie numer takiej tekstury. *Stage* odnośnie tekstur określa się częściej jako *TextureStage*. *TextureStage* możemy nazwać kolejną akcelerowaną sprzętowo teksturę. Na przykład jeżeli urządzenie posiada *2* tekstury, to możemy mieć do czynienia ze *stage'em* numer *0* i *1*. *TMU* to skrót od *texture mapping unit* i jest to jednostka mapowania tekseli. Od ilości *TMU* zależy prędkość nakładania tekstury w karcie. Jeżeli np. karta ma cztery potoki i po jednym *TMU* na potok, wtedy użycie dwóch tekstur na piksel będzie dwa razy wolniejsze niż użycie jednej. Jeżeli karta będzie miała np. cztery potoki i po *2 TMU* na potok wtedy nie ma różnicy czy nakładana będzie jedna tekstura czy dwie. Operacje wspierane przez *SetTextureStageState* mają zazwyczaj dwa argumenty (nie mylić z parametrami metody *SetTextureStageState*). Meotda *IDirect3DDevice::SetTextureStageState* ma definicje podaną na listingu 85.

```
HRESULT SetTextureStageState(
    DWORD Stage,
    D3DTEXTURESTAGESTATETYPE Type,
    DWORD Value
);
```

**Listing 16 –** *IDirect3DDevice9::SetTextureStageState*, Źródło *DirectX SDK*

Pierwszy parametr to numer *stage'a*. Drugi to jeden z elementów listy poniżej. Trzeci - to wartość. Różne elementy mogą przyjmować różne wartości:

```
typedef enum _D3DTEXTURESTAGESTATETYPE {
    D3DTSS_COLOROP = 1,
    D3DTSS_COLORARG1 = 2,
    D3DTSS_COLORARG2 = 3,
    D3DTSS_ALPHAOP = 4,
    D3DTSS_ALPHAARG1 = 5,
    D3DTSS_ALPHAARG2 = 6,
    D3DTSS_BUMPENVMAT00 = 7,
    D3DTSS_BUMPENVMAT01 = 8,
    D3DTSS_BUMPENVMAT10 = 9,
    D3DTSS_BUMPENVMAT11 = 10,
    D3DTSS_TEXCOORDINDEX = 11,
    D3DTSS_BUMPENVLSCALE = 22,
    D3DTSS_BUMPENVLOFFSET = 23,
    D3DTSS_TEXTURETRANSFORMFLAGS = 24,
    D3DTSS_COLORARG0 = 26,
    D3DTSS_ALPHAARG0 = 27,
    D3DTSS_RESULTARG = 28,
    D3DTSS_CONSTANT = 32,
    D3DTSS_FORCE_DWORD = 0x7fffffff
} D3DTEXTURESTAGESTATETYPE;
```

**Listing 17 –** *D3DTEXTURESTAGESTATETYPE*, Źródło *DirectX SDK*

```
typedef enum _D3DTEXTUREOP {
    D3DTOP_DISABLE = 1,
    D3DTOP_SELECTARG1 = 2,
    D3DTOP_SELECTARG2 = 3,
    D3DTOP_MODULATE = 4,
    D3DTOP_MODULATE2X = 5,
    D3DTOP_MODULATE4X = 6,
    D3DTOP_ADD = 7,
    D3DTOP_ADDSIGNED = 8,
    D3DTOP_ADDSIGNED2X = 9,
    D3DTOP_SUBTRACT = 10,
    D3DTOP_ADDSMOOTH = 11,
    D3DTOP_BLENDDIFFUSEALPHA = 12,
    D3DTOP_BLENDTEXTUREALPHA = 13,
    D3DTOP_BLENDFACTORALPHA = 14,
    D3DTOP_BLENDTEXTUREALPHAPM = 15,
    D3DTOP_BLENDCURRENTALPHA = 16,
    D3DTOP_PREMODULATE = 17,
    D3DTOP_MODULATEALPHA_ADDCOLOR = 18,
    D3DTOP_MODULATECOLOR_ADDALPHA = 19,
    D3DTOP_MODULATEINVALPHA_ADDCOLOR = 20,
    D3DTOP_MODULATEINVCOLOR_ADDALPHA = 21,
    D3DTOP_BUMPENVMAP = 22,
    D3DTOP_BUMPENVMAPLUMINANCE = 23,
    D3DTOP_DOTPRODUCT3 = 24,
    D3DTOP_MULTIPLYADD = 25,
    D3DTOP_LERP = 26,
    D3DTOP_FORCE_DWORD = 0x7fffffff
} D3DTEXTUREOP;
```

**Listing 18 –** *D3DTEXTUREOP*, Źródło *DirectX SDK*

- *D3DTSS_COLOROP*
- *D3DTSS_ALPHAOP*

Te elementy ustawiają operacje wykonywane między kolorami albo kanałem alfa. Ich parametrami mogą być wartości przedstawione na listingu 87.

- *D3DTOP_ADD* – dodawanie kolorów
- *D3DTOP_SUBTRACT* – odejmowanie
- *D3DTOP_MODULATE* – mnożenie kolorów
- *D3DTOP_DISABLE* – wyłączenie operacji
- *D3DTOP_SELECARG1*- zamiast wykonania opracji wybierz to na co wskazuje argument pierwszy (opic argumentów poniżej)
- *D3DTOP_SELECTARG2* - zamiast wykonania opracji wybierz to na co wskazuje argument drugi

- *D3DTSS_COLORARG1*
- *D3DTSS_COLORARG2*

Służą do kontroli argumentów i mogą przyjmować następujące wartości:

- *D3DTA_CURRENT* – weź wynik z poprzedniego stage'a
- *D3DTA_DIFFUSE* – weź wynik oświetlania diffuse
- *D3DTA_SPECULAR* – weź wynik oświetlania specular
- *D3DTA_TEXTURE* – weź kolor tekstury

- *D3DTSS_ALPHAARG1*
- *D3DTSS_ALPHAARG2*

To samo co kolor, tyle że opisuje argumenty operacji *ALPHAOP* dla kanału alfa.

Przy pomocy *IDirect3Ddevice9::SetSampleState* możemy kontrolować w jaki sposób tekstura jest samplowana, czyli jak jest filtrowana, czy jest zawijana etc. Metoda *IDirect3Ddevice9::SetSampleState* podana jest na listingu 88.

```
HRESULT SetSamplerState(
    DWORD Sampler,
    D3DSAMPLERSTATETYPE Type,
    DWORD Value
);
```

**Listing 19 –** *IDirect3DDevice9::SetSamplerState*, Źródło *DirectX SDK*

Parametry przyjmuje analogicznie do *SetTextureStageState* z tym wyjątkiem, że drugi parametr może być jedną z wartości podaną na listingu 89.

```
typedef enum _D3DSAMPLERSTATETYPE {
    D3DSAMP_ADDRESSU = 1,
    D3DSAMP_ADDRESSV = 2,
    D3DSAMP_ADDRESSW = 3,
    D3DSAMP_BORDERCOLOR = 4,
    D3DSAMP_MAGFILTER = 5,
    D3DSAMP_MINFILTER = 6,
    D3DSAMP_MIPFILTER = 7,
    D3DSAMP_MIPMAPLODBIAS = 8,
    D3DSAMP_MAXMIPLEVEL = 9,
    D3DSAMP_MAXANISOTROPY = 10,
    D3DSAMP_SRGBTEXTURE = 11,
    D3DSAMP_ELEMENTINDEX = 12,
    D3DSAMP_DMAPOFFSET = 13,
    D3DSAMP_FORCE_DWORD = 0x7fffffff
} D3DSAMPLERSTATETYPE;
```

**Listing 20 –** *D3DSAMPLERSTATETYPE*, Źródło *DirectX SDK*

- *D3DTSS_ADDRESSU*
- *D3DTSS_ADDRESSV*

  Służa do adresowanie tekstury np. *CLAMP, WRAP*. Wszystkie możliwe parametry podane są na listingu 90.

```
typedef enum _D3DTEXTUREADDRESS {
    D3DTADDRESS_WRAP = 1,
    D3DTADDRESS_MIRROR = 2,
    D3DTADDRESS_CLAMP = 3,
    D3DTADDRESS_BORDER = 4,
    D3DTADDRESS_MIRRORONCE = 5,
    D3DTADDRESS_FORCE_DWORD = 0x7fffffff
} D3DTEXTUREADDRESS;
```

**Listing 21 –** *D3DTEXTUREADDRESS*, Źródło *DirectX SDK*

- *D3DTSS_MAGFILTER*
- *D3DTSS_MINFILTER*
- *D3DTSS_MIPFILTER*

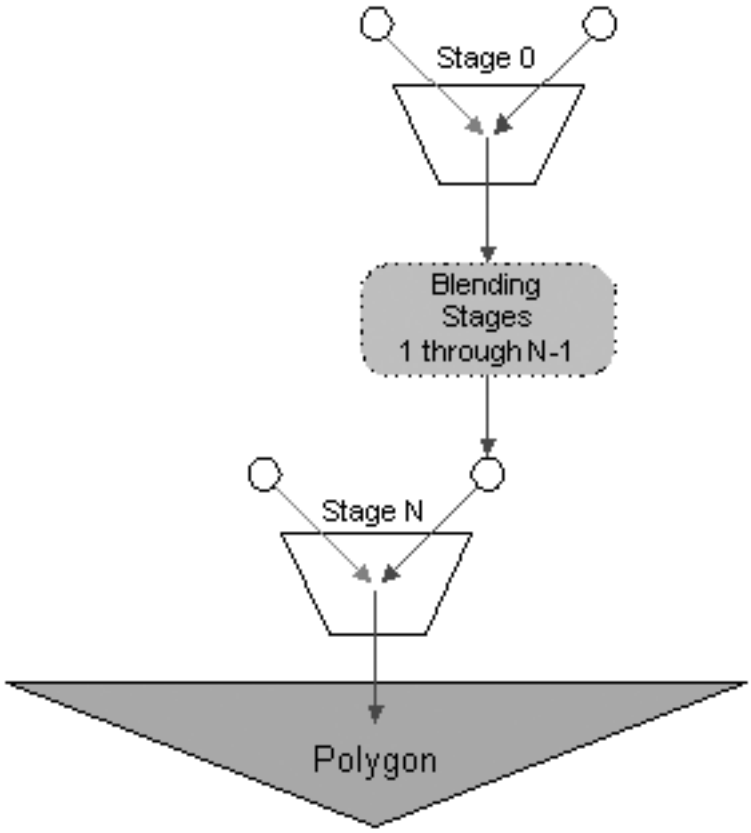
  Definiują operacje filtrowania o których mówiliśmy na początku rozdziału. Mogą przyjmować wartości podane na listingu 91.

```
typedef enum _D3DTEXTUREFILTERTYPE {
    D3DTEXF_NONE = 0,
    D3DTEXF_POINT = 1,
    D3DTEXF_LINEAR = 2,
    D3DTEXF_ANISOTROPIC = 3,
```

```
    D3DTEXF_PYRAMIDALQUAD = 6,
    D3DTEXF_GAUSSIANQUAD = 7,
    D3DTEXF_FORCE_DWORD = 0x7fffffff
} D3DTEXTUREFILTERTYPE;
```

**Listing 22 –** *D3DTEXTURFILTERTYPE*, Źródło *DirectX SDK*

Rysunek 32 przedstawia schemat operacji *multiteksturowania*.




**Rysunek 6** - *Multitexturing*, Źródło *DirectX SDK*

Każdy *TextureStage* ma 2 składniki i jeden wynik. Zatem wyobraźmy sobie sytuację, że mamy następujące argumenty:

- wynik oświetlania
- teksturę pierwszą
- teksturę drugą

Mamy pomnożyć wynik oświetlania przez pierwszą teksturę i dodać do tego drugą teksturę, natomiast alfę z oświetlania pomnożyć przez alfę w pierwszej teksturze i zignorować alfę w drugiej teksturze. Ustawienie *SetTextureStageState* przedstawione jest na listingu 92.

```
SetTextureStageState( 0, D3DTSS_ALPHAOP, D3DTOP_MODULATE );
SetTextureStageState( 0, D3DTSS_ALPHAARG1, D3DTA_DIFFUSE );
SetTextureStageState( 0, D3DTSS_ALPHAARG2, D3DTA_TEXTURE );
SetTextureStageState( 0, D3DTSS_COLOROP, D3DTOP_MODULATE );
SetTextureStageState( 0, D3DTSS_COLORARG1, D3DTA_DIFFUSE );
SetTextureStageState( 0, D3DTSS_COLORARG2, D3DTA_TEXTURE );
SetTextureStageState( 1, D3DTSS_ALPHAOP, D3DTOP_DISABLE );
SetTextureStageState( 1, D3DTSS_COLOROP, D3DTOP_ADD );
SetTextureStageState( 1, D3DTSS_COLORARG1, D3DTA_CURRENT );
SetTextureStageState( 1, D3DTSS_COLORARG2, D3DTA_TEXTURE );
```

**Listing 23 –** Przykładowe *SetTextureStageState*, Źródło własne