Programowanie Grafiki Komputerowej

Microsoft DirectX cz. 2 - organizacja przestrzeni 3D, Z-Bufor, transformacje 3D, opis obiektu

Grzegorz Łukawski

Katedra Informatyki, Politechnika Świętokrzyska w Kielcach



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



Politechnika Świętokrzyska

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY



Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

" Program Rozwojowy Potencjału Dydaktycznego Politechniki Świętokrzyskiej w Kielcach: *kształcenie w nowoczesnych obszarach techniki* "

Program Operacyjny Kapitał Ludzki, Priorytet IV Szkolnictwo wyższe i nauka, Działanie 4.1. Wzmocnienie i rozwój potencjału dydaktycznego uczelni oraz zwiększenie liczby absolwentów kierunków o kluczowym znaczeniu dla gospodarki opartej na wiedzy, Poddziałanie 4.1.1. Wzmocnienie potencjału dydaktycznego uczelni.

Umowa UDA–POKL.04.01.01-00395/09-00

Spis treści

1. Organizacja przestrzeni 3D
   - Z-Bufor
   - Bryła widzenia
   - Położenie obserwatora
2. Opis obiektu
   - FVF
   - Definicja obiektu
   - Rendering obiektów
3. Transformacje 3D
   - Podstawowe transformacje
   - Składanie przekształceń
   - Przykładowy program (dx-p3)
4. Indeksowane obiekty 3D
   - Tabele ścian i tabele wierzchołków
   - Przykładowy program (dx-p4)
5. Koniec

Uwagi

- Animacje są uruchamiane, gdy plik PDF jest wyświetlany na pełnym ekranie.
- Animacja jest uruchamiana dopiero po naciśnięciu spacji (program Adobe Reader).
- Klatki animacji wyświetlane są co 2 sekundy.
- Symulacje są uruchamiane po kliknięciu w ikonę z odpowiednim systemem operacyjnym. Przeglądarka wyświetli okno z ostrzeżeniem, należy go zaakceptować. W systemie Linux należy dodatkowo kliknąć dwukrotnie w nazwę programu.

Wymagania

Prezentowane programy wymagają dołączenia następujących bibliotek:

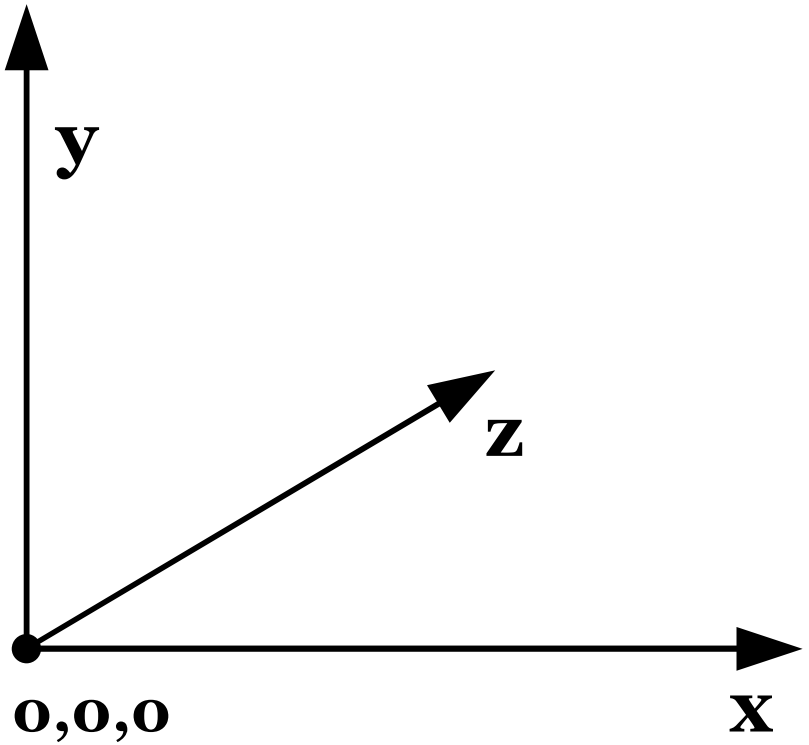
include

```
#include <windows.h>
#include <d3d9.h>
#include <d3dx9.h>

#pragma comment (lib, "d3d9.lib")
#pragma comment (lib, "d3dx9.lib")
```

Układ współrzędnych

DirectX korzysta z **kartezjańskiego, lewoskrętnego** układu współrzędnych:



Inicjalizacja Z-Bufora

Z-Bufor (bufor głębokości) to podstawowy, sprzętowy mechanizm określający widoczność obiektów 3D. Rozwiązuje problemy z wzajemnym zasłanianiem obiektów, niezależnie od kolejności rysowania[1].

Inicjalizacja:

```
// Informacja o użyciu Z-Bufora:
d3dpp.EnableAutoDepthStencil = TRUE;
// Format Z-Bufora - 16 bitów:
d3dpp.AutoDepthStencilFormat = D3DFMT_D16;

// Włączenie Z-Bufora:
d3ddev->SetRenderState(D3DRS_ZENABLE, TRUE);
```

[1] Algorytm działania Z-Bufora przedstawiony został na wykładzie z podstaw grafiki komputerowej 3D.

Czyszczenie Z-Bufora

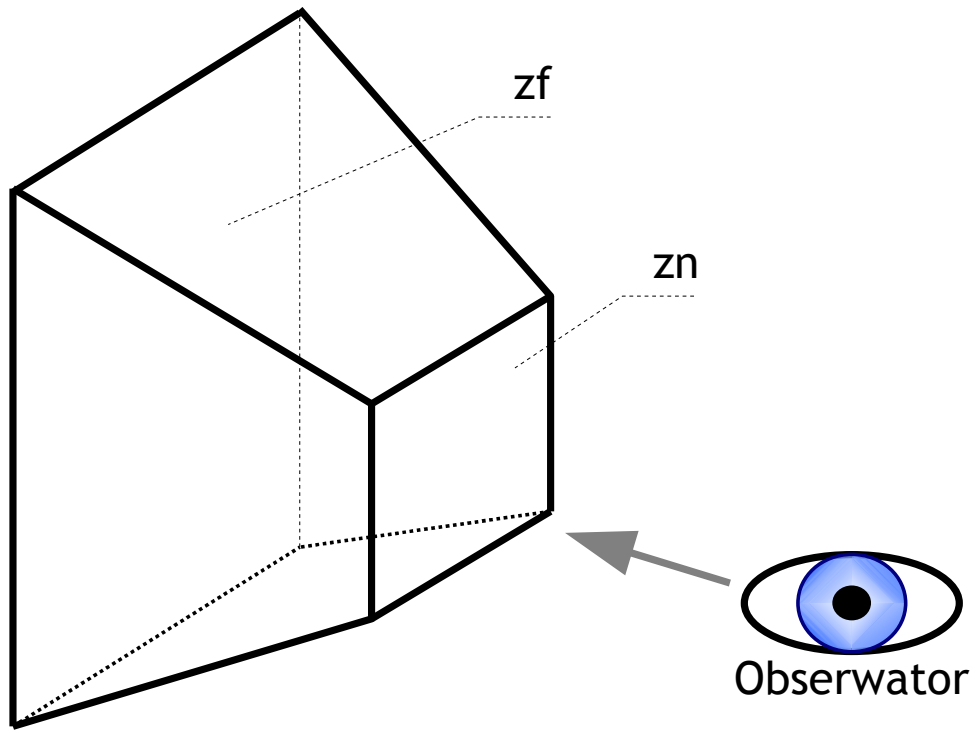
Przed narysowaniem każdej ramki animacji, Z-Bufor musi zostać wyczyszczony (wypełniony czarnym kolorem):

Czyszczenie Z-Bufora:

```
d3ddev->Clear(0, NULL, D3DCLEAR_ZBUFFER,
                    D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
```

Kształt bryły widzenia

Bryła widzenia w przypadku rzutowania perspektywicznego[2] ma kształt ściętego ostrosłupa:



[2]Rodzaje rzutowania przedstawione były na wykładzie z podstaw grafiki komputerowej 3D.

Macierz rzutowania

Rzutowanie perspektywiczne i **bryła widzenia** realizowane są z pomocą macierzy o rozmiarach 4x4. Macierz rzutowania wygenerować można z pomocą funkcji:

Macierz rzutowania:

```
// Macierz 4x4:
D3DXMATRIX matProj;

// Wygenerowanie macierzy rzutowania:
D3DXMatrixPerspectiveFovLH(&matProj,
    D3DXToRadian(45), // Kąt rozwarcia ścian ostrosłupa
    (FLOAT)640 / (FLOAT)480, // Proporcje okna
    1.0f, // Odległość do przedniej ściany (zn)
    100.0f); // Odległość do tylnej ściany (zf)
```

Kąt rozwarcia ścian ostrosłupa to typowo od 45 do 60 stopni.

Macierz rzutowania

Przygotowaną macierz rzutowania należy ustawić jako aktywną z pomocą metody:

Ustawienie macierzy rzutowania:

```
d3ddev->SetTransform(D3DTS_PROJECTION, &matProj);
```

Ustawienie punktu obserwacji

Położenie "kamery" może być kontrolowane z pomocą macierzy widoku i funkcji **D3DXMatrixLookAtLH()**.

Ustawienie punktu obserwacji:

```
D3DXMATRIX matView;

D3DXMatrixLookAtLH(&matView,
        &D3DXVECTOR3 (0.0f, 0.0f, -4.0f), // Pozycja kamery
        &D3DXVECTOR3 (0.0f, 0.0f, 0.0f), // Punkt docelowy
        &D3DXVECTOR3 (0.0f, 1.0f, 0.0f)); // Orientacja kamery

// Użycie przygotowanej macierzy widoku:
d3ddev->SetTransform(D3DTS_VIEW, &matView);
```

Struktura opisująca wektor:

```
typedef struct D3DXVECTOR3 {
        FLOAT x; FLOAT y; FLOAT z;
};
```

Flexible Vertex Format

Flexible Vertex Format (FVF) to sposób na przekazanie do Direct3D tylko potrzebnych danych związanych z definicją wierzchołków:

- `D3DFVF_XYZ (float, float, float)` – współrzędne wierzchołka w przestrzeni 3D (X-Y-Z).
- `D3DFVF_XYZRHW (float, float, float, float)` – współrzędne ekranowe wierzchołka (X-Y), dodatkowo współrzędna Z oraz współrzędna W wzajemnych jednorodnych (Reciprocal of Homogeneous W). Definicja przydatna przy tworzeniu własnych silników 3D.
- `D3DFVF_DIFFUSE (DWORD)` – kolor dla światła rozproszenia.
- `D3DFVF_SPECULAR (DWORD)` – kolor dla światła rozbłysków.
- `D3DFVF_TEX0 ... D3DFVF_TEX8 (float, float)` – współrzędne tekstury.

Definicja FVF

Przykładowa definicja własnego FVF:

```
#define MOJ_FVF (D3DFVF_XYZ | D3DFVF_DIFFUSE)

// Struktura zawiera definicję wierzchołka zgodnie
// z przyjętym formatem FVF:
struct MOJ_VERTEX {
    FLOAT x, y, z; // X-Y-Z
    DWORD color; // Kolor dla składowej DIFFUSE
};
```

Tabela wierzchołków

Definicja obiektu 3D składa się z odpowiedniej liczby wierzchołków, zależnie od przyjętego sposobu opisu obiektu (np. trójki wierzchołków tworzą trójkąty). Kolejne wierzchołki powinny być umieszczone w tabeli.

Obiekt złożony z dwóch rozłącznych trójkątów:

```
MOJ_VERTEX triang_vert[] = {
    {-1.0f, 0.8f, -0.5f, D3DCOLOR_XRGB(255, 0, 0)},
    { 1.3f, 0.5f, -0.5f, D3DCOLOR_XRGB(255, 255, 0)},
    { 0.2f, -1.0f, -0.5f, D3DCOLOR_XRGB( 0, 255, 0)},

    {-1.2f, 0.8f, 0.5f, D3DCOLOR_XRGB( 0, 0, 255)},
    { 0.9f, 0.4f, 0.5f, D3DCOLOR_XRGB( 0, 255, 255)},
    { 0.2f, -1.1f, 0.5f, D3DCOLOR_XRGB(255, 255, 0)},
};
```

Bufor wierzchołków

Bufor wierzchołków (vertex buffer) to obszar pamięci **karty graficznej** do której należy skopiować definicję obiektu. Bufor tworzony jest z pomocą metody **CreateVertexBuffer()**.

Następnie bufor należy zablokować metodą **Lock()** i skopiować dane w standardowy sposób, np. z pomocą funkcji **memcpy()**. Gotowy bufor należy odblokować metodą **Unlock()**.

Blokada:

Zablokowanie bufora zabezpiecza go przed dostępem innych procesów i procesora karty graficznej. Dzięki temu, między innymi, taki obszar nie zostanie nagle przesunięty w inne miejsce pamięci.

CreateVertexBuffer()

```
CreateVertexBuffer(UINT Length, DWORD Usage,
                   DWORD FVF, D3DPOOL Pool,
                   LPDIRECT3DVERTEXBUFFER9 ppVertexBuffer,
                   HANDLE* pSharedHandle);
```

- **Length** – długość bufora w bajtach.
- **Usage** – parametr przydatny jeżeli format wierzchołków odbiega od standardu.
- **FVF** – nasz specyficzny format wierzchołków.
- **Pool** – gdzie i jak DirectX ma utworzyć bufor?
  - `D3DPOOL_DEFAULT` – domyślnie, wybór automatyczny.
  - `D3DPOOL_MANAGED` – bufor zostanie utworzony w pamięci graficznej.
  - `D3DPOOL_SYSTEMMEM` – w pamięci systemowej (niedostępnej dla urządzenia Direct3D!).
  - `D3DPOOL_SCRATCH` – w pamięci systemowej, ale całkowicie bez dostępu przez Direct3D (np. na chwilowo nieużywane tekstury).
- **ppVertexBuffer** – wskaźnik na bufor docelowy.
- **pSharedHandle** – Nieużywane.

Lock()

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

- **OffsetToLock** – początek obszaru do zablokowania w buforze.
- **SizeToLock** – długość obszaru do zablokowania.
- **ppbData** – wskaźnik na wskaźnik na obszar wynikowy (rezultat działania metody).
- **Flags** – opcje zaawansowane lub zero.

Jeżeli jako dwa pierwsze parametry podano zera, zablokowany zostanie cały bufor.

Przykład

Utworzenie i wypełnienie bufora wierzchołków:

```
// Bufor w pamięci karty graficznej:
LPDIRECT3DVERTEXBUFFER9 v_buffer;

// Dwa trójkąty = 6 wierzchołków:
d3ddev->CreateVertexBuffer(6*sizeof(MOJ_VERTEX),
    0, MOJ_FVF, D3DPOOL_MANAGED, &v_buffer, NULL);

// Zwykły wskaźnik pamięci RAM:
VOID* pVoid;
// Blokada i uzyskanie adresu:
v_buffer->Lock(0, 0, (void**)&pVoid, 0);

// Skopiowanie definicji obiektu i zdjęcie blokady:
memcpy(pVoid, triang_vert, sizeof(triang_vert));
v_buffer->Unlock();
```

Strumienie wierzchołków

Rendering obiektu składa się z trzech etapów:

1. **SetFVF()** – określenie formatu opisu wierzchołków.
2. **SetStreamSource()** – wybranie tzw. "strumienia", czyli określenie którego bufora wierzchołków należy użyć do narysowanie kolejnego obiektu.
3. **DrawPrimitive()** – narysowanie prymitywu.

Uwaga:

W obrębie jednej sceny można stosować wiele formatów opisu wierzchołków i wiele strumieni. Można się między nimi przełączać w dowolnym momencie rysowania ramki animacji.

SetStreamSource()

```
SetStreamSource(UINT StreamNumber,
                LPDIRECT3DVERTEXBUFFER9 pStreamData,
                UINT OffsetInBytes, UINT Stride);
```

- **StreamNumber** – numer strumienia wierzchołków, lub zero (jeżeli jest tylko jeden strumień).
- **pStreamData** – wskaźnik na bufor wierzchołków.
- **OffsetInBytes** – offset w buforze wierzchołków, lub zero.
- **Stride** – wielkość danych opisujących jeden wierzchołek, zazwyczaj sizeof(MOJ_VERTEX).

DrawPrimitive()

```
DrawPrimitive(D3DPRIMITIVETYPE PrimitiveType,
              UINT StartVertex, UINT PrimitiveCount);
```

- **PrimitiveType** – rodzaj prymitywu, wybrane rodzaje:
  - `D3DPT_POINTLIST` – seria punktów,
  - `D3DPT_LINELIST` – seria niezależnych linii,
  - `D3DPT_LINESTRIP` – seria połączonych linii,
  - `D3DPT_TRIANGLELIST` – lista niezależnych trójkątów,
  - `D3DPT_TRIANGLESTRIP` – pasek trójkątów,
  - `D3DPT_TRIANGLEFAN` – wachlarz trójkątów.
- **StartVertex** – numer pierwszego wierzchołka od którego rozpocząć rysowanie.
- **PrimitiveCount** – liczba prymitywów w buforze, które należy narysować.

Przykład

Rendering obiektu:

```
d3ddev->SetFVF(MOJ_FVF);

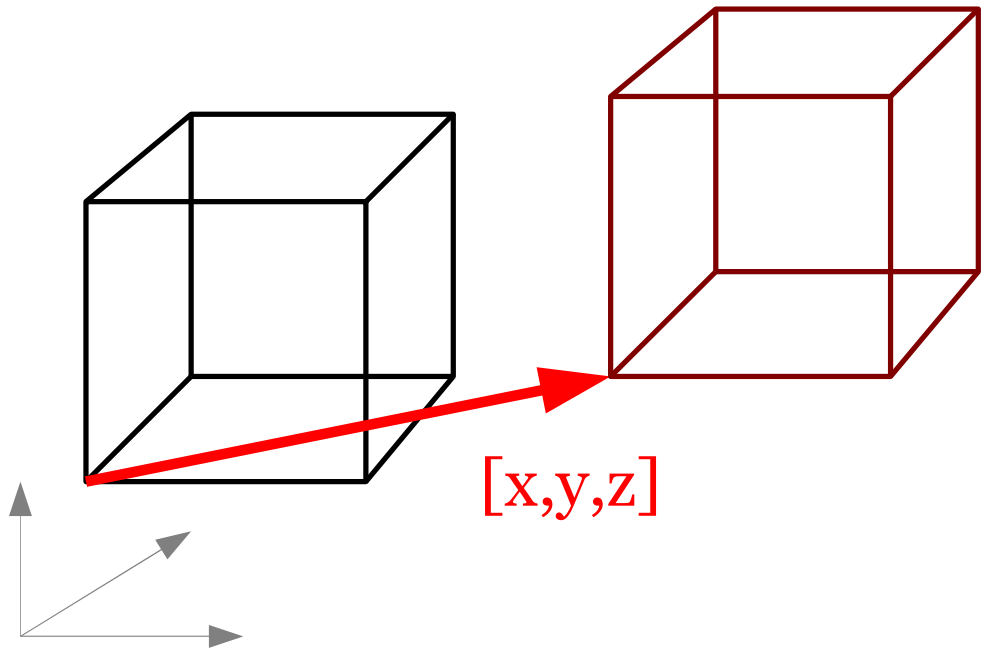
// Ustawienie strumienia:
d3ddev->SetStreamSource(0, v_buffer, 0,
                sizeof(MOJ_VERTEX));

// Narysowanie kształtu (dwa trójkąty):
d3ddev->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);
```

Translacja – przesunięcie o wektor

```
D3DXMatrixTranslation(D3DXMATRIX *pOut,
                      FLOAT x, FLOAT y, FLOAT z);
```

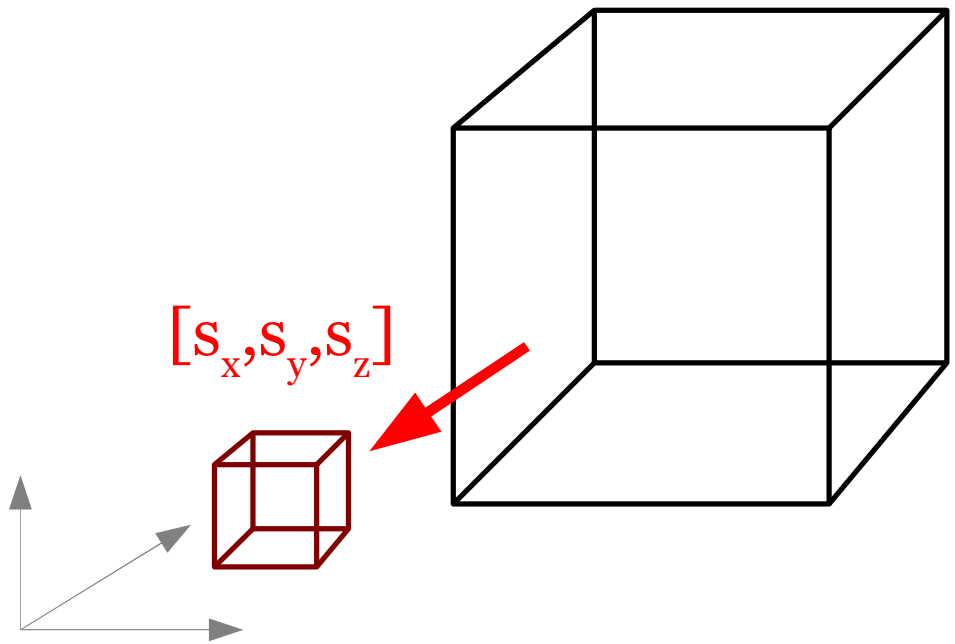
- **pOut** – wynikowa macierz,
- **x, y, z** – wektor translacji.



Jednokładność – skalowanie

```
D3DXMatrixScaling(D3DXMATRIX *pOut,
                                  FLOAT sx, FLOAT sy, FLOAT sz);
```

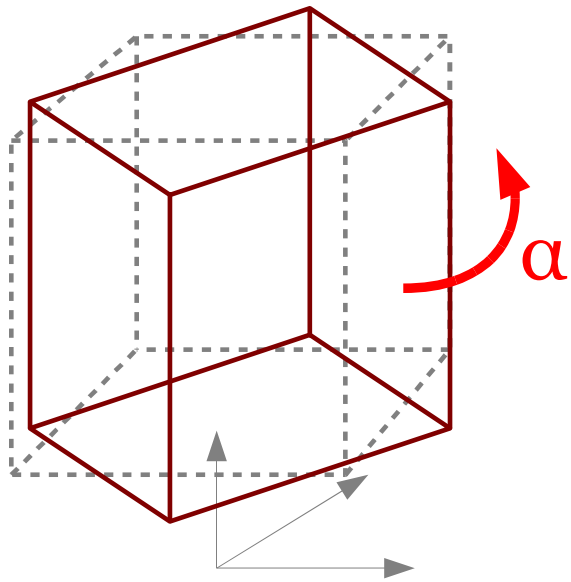
- **pOut** – wynikowa macierz,
- **sx, sy, sz** – współczynniki skalowania.



Obrót wokół osi X, Y, Z

```
D3DXMatrixRotationX(D3DXMATRIX *pOut, FLOAT Angle);
D3DXMatrixRotationY(D3DXMATRIX *pOut, FLOAT Angle);
D3DXMatrixRotationZ(D3DXMATRIX *pOut, FLOAT Angle);
```

- **pOut** – wynikowa macierz,
- **Angle** – kąt obrotu w radianach.



Obrót wokół dowolnej osi

```
D3DXMatrixRotationAxis(D3DXMATRIX *pOut,
                       const D3DXVECTOR3 *pV, FLOAT Angle);
```

- **pOut** – wynikowa macierz,
- **pV** – oś obrotu (jako wektor),
- **Angle** – kąt obrotu w radianach.

Operacje na macierzach

Przekształcenia można składać, tworząc bardziej złożone transformacje. Złożenie przekształceń odpowiada **mnożeniu przez siebie** odpowiednich macierzy.

Operacje na macierzach:

Na zmiennych typu `D3DXMATRIX` można bezpośrednio stosować (przeciążone) operatory: *, +, -, /, +=, *=, itp.

Generowanie **macierzy jednostkowej**, która może stać się bazą dla własnych przekształceń:

```
D3DXMatrixIdentity(D3DXMATRIX* pOut);
```

Użycie macierzy przekształceń

Przygotowaną macierz przekształceń można aktywować z pomocą metody:

```
d3ddev->SetTransform(D3DTS_WORLD, &macierz);
```

Od tej chwili, wszystkie obiekty, wierzchołki i kształty umieszczone na scenie zostaną poddane wskazanej transformacji.

Funkcja renderująca dwa trójkąty

```
// Wyczyszczenie ekranu i Z-Bufora:
d3ddev->Clear(0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0, 0, 80), 1.0f, 0);
d3ddev->Clear(0, NULL, D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);

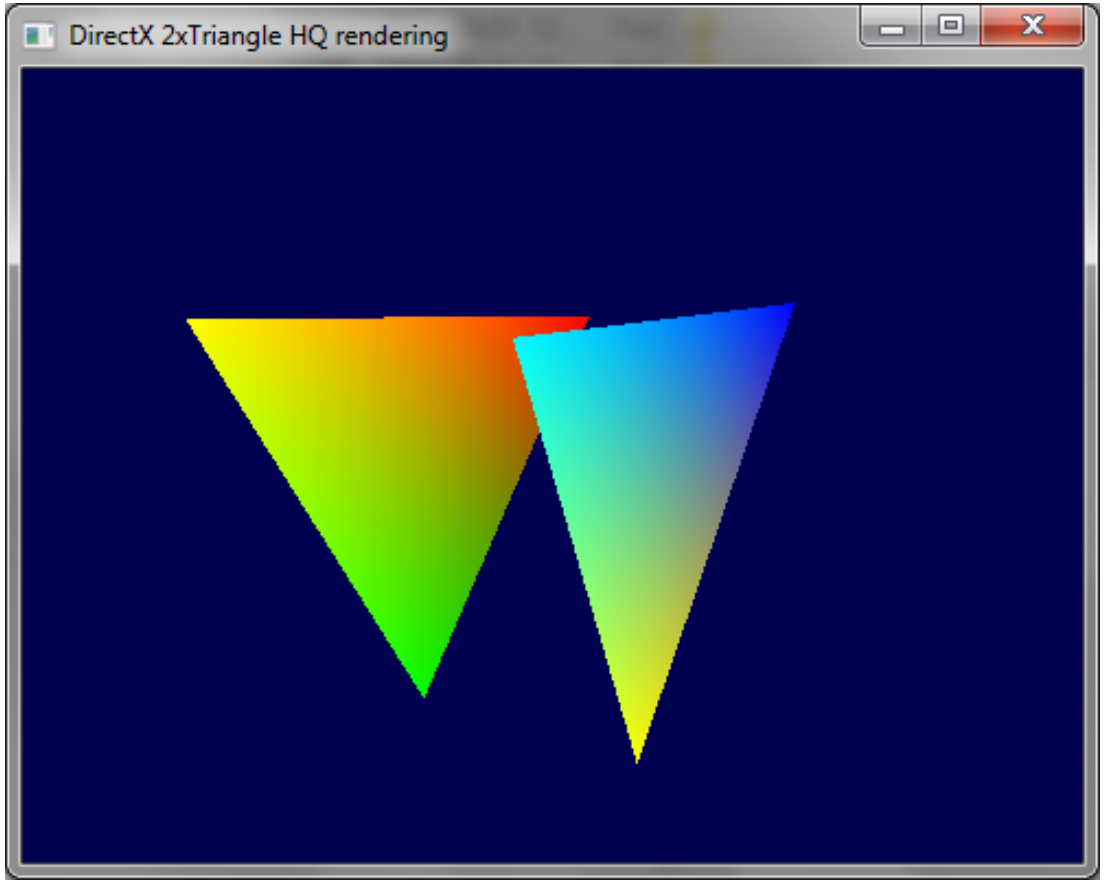
d3ddev->BeginScene(); // Rozpoczęcie rysowania sceny

D3DXMATRIX matRot;
static float index = 0.0f; index += 0.02f; // Animacja
D3DXMatrixRotationY(&matRot, index);
d3ddev->SetTransform(D3DTS_WORLD, &matRot);

// Ustawienie punktu obserwacji:
D3DXMATRIX matView;
D3DXMatrixLookAtLH(&matView,
    &D3DXVECTOR3(0.0f, 0.0f, -4.0f),
    &D3DXVECTOR3(0.0f, 0.0f, 0.0f),
    &D3DXVECTOR3(0.0f, 1.0f, 0.0f));
d3ddev->SetTransform(D3DTS_VIEW, &matView);

// Określenie strumienia wierzchołków:
d3ddev->SetFVF(MOJ_FVF);
d3ddev->SetStreamSource(0, v_buffer, 0, sizeof(MOJ_VERTEX));
d3ddev->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2); // Narysowanie dwóch trójkątów

d3ddev->EndScene();
d3ddev->Present(NULL, NULL, NULL, NULL);
```



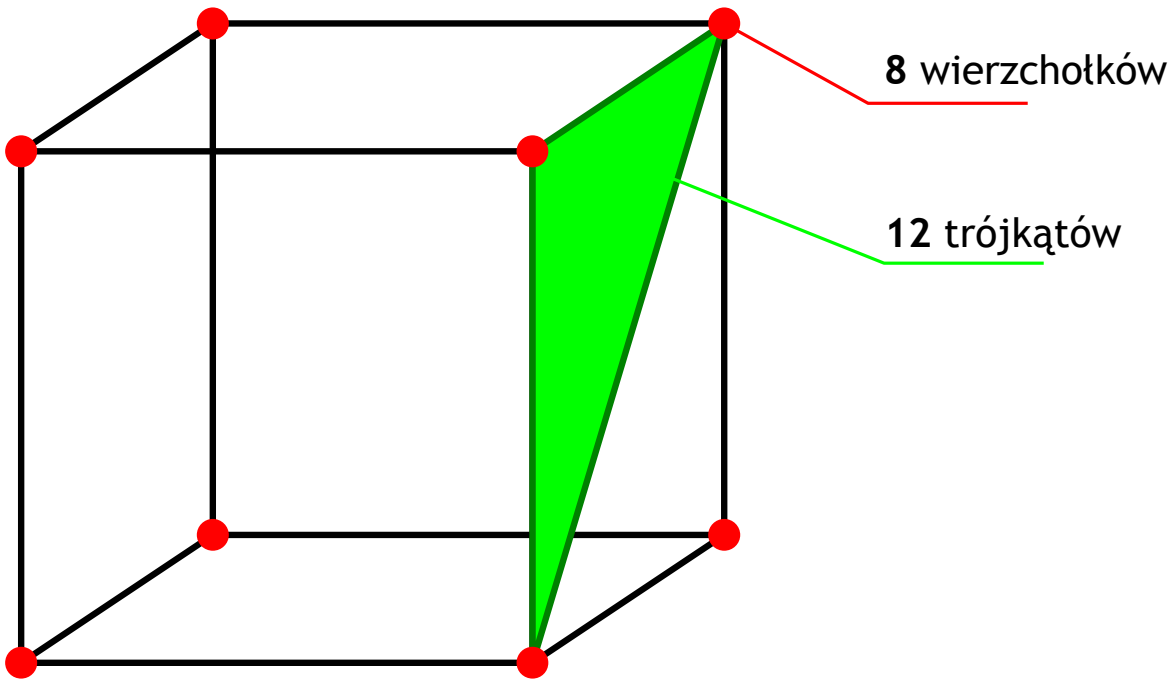
Przykładowy program

Windows

Tabela ścian

Bardziej efektywny opis obiektu 3D to zastosowanie **dwóch tabeli**:

1. **Tabeli wierzchołków (*vertex table*)**, zawierającej współrzędne X-Y-Z wszystkich wierzchołków obiektu.
2. **Tabeli ścian/indeksów (*index table*)**, zawierającej numery wierzchołków pomiędzy którymi rozpięte są ściany.



Limit indeksów

Maksymalna liczba ścian obiektu opisanego w ten sposób ograniczona jest przez możliwości biblioteki/sprzętu/sterownika. Jest to conajmniej 2^{16} ścian.

Odczyt maksymalnej liczby wierzchołków:

```
D3DCAPS9 capz;
d3d->GetDeviceCaps(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, &capz);
cout << "MaxVertexIndex = " << capz.MaxVertexIndex << endl;
```

Sześcian:

```
MOJ_VERTEX cube_vert[] = { // 8 wierzchołków sześcianu
    { -1.0f, 1.0f, -1.0f, D3DCOLOR_XRGB( 0, 255, 0), }, // 0
    { 1.0f, 1.0f, -1.0f, D3DCOLOR_XRGB(255, 255, 0), }, // 1
    { -1.0f, -1.0f, -1.0f, D3DCOLOR_XRGB( 0, 0, 0), }, // 2
    { 1.0f, -1.0f, -1.0f, D3DCOLOR_XRGB(255, 0, 0), }, // 3
    { -1.0f, 1.0f, 1.0f, D3DCOLOR_XRGB( 0, 255, 255), }, // 4
    { 1.0f, 1.0f, 1.0f, D3DCOLOR_XRGB(255, 255, 255), }, // 5
    { -1.0f, -1.0f, 1.0f, D3DCOLOR_XRGB( 0, 0, 255), }, // 6
    { 1.0f, -1.0f, 1.0f, D3DCOLOR_XRGB(255, 0, 255), }, // 7
};

short cube_ind[] = { // 12 trójkątów składających się na sześcian
    0, 1, 2,  2, 1, 3,
    4, 0, 6,  6, 0, 2,
    7, 5, 6,  6, 5, 4,
    3, 1, 7,  7, 1, 5,
    4, 5, 0,  0, 5, 1,
    3, 7, 2,  2, 7, 6,
};
```

Bufory indeksów

Do opisania kształtu obiektu z pomocą dwóch tabel, oprócz bufora wierzchołków, niezbędny jest **bufor indeksów (index buffer)**. Jest on tworzony przez wywołanie metody **CreateIndexBuffer()**.

Analogicznie, dostęp do bufora indeksów powinien być poprzedzony zablokowaniem obszaru pamięci (metoda **Lock()**). Tabelę ścian należy skopiować do pamięci karty graficznej z pomocą standardowych metod, jak funkcja **memcpy()**.

CreateIndexBuffer()

```
CreateIndexBuffer(UINT Length, DWORD Usage, D3DFORMAT Format,
                  D3DPOOL Pool,
                  LPDIRECT3DINDEXBUFFER9 ppIndexBuffer,
                  HANDLE* pSharedHandle);
```

- **Length** – długość bufora w bajtach.
- **Usage** – parametr przydatny jeżeli format wierzchołków odbiega od standardu.
- **Format** – ile miejsca w pamięci zajmuje każdy indeks?
  - `D3DFMT_INDEX16` – 16 bitów, zapewnia to dużą szybkość. Maksymalna liczba ścian to 2^{16}.
  - `D3DFMT_INDEX32` – 32 bity, indeksem jest typ "int". Niedostępne na starszych kartach graficznych.
- **Pool** – gdzie i jak DirectX ma utworzyć bufor? Znaczenie parametru takie samo jak dla metody **CreateVertexBuffer()**.
- **ppIndexBuffer** – wskaźnik na bufor docelowy.
- **pSharedHandle** – Nieużywane.

Przygotowanie buforów dla sześcianu:

```
LPDIRECT3DVERTEXBUFFER9 v_buffer; // Bufor wierzchołków
LPDIRECT3DINDEXBUFFER9 i_buffer; // Bufor indeksów

// Przygotowanie i wypełnienie bufora wierzchołków:
d3ddev->CreateVertexBuffer(8*sizeof(MOJ_VERTEX), 0,
      MOJ_FVF, D3DPOOL_MANAGED, &v_buffer, NULL);

VOID* pVoid;
v_buffer->Lock(0, 0, (void**)&pVoid, 0);
memcpy(pVoid, cube_vert, sizeof(cube_vert));
v_buffer->Unlock();

// Przygotowanie i wypełnienie bufora indeksów:
d3ddev->CreateIndexBuffer(3 * 12 * sizeof(short), 0,
      D3DFMT_INDEX16, D3DPOOL_MANAGED, &i_buffer, NULL);

i_buffer->Lock(0, 0, (void**)&pVoid, 0);
memcpy(pVoid, cube_ind, sizeof(cube_ind));
i_buffer->Unlock();
```

Rendering obiektów indeksowanych

Rendering obiektu opisanego tabelą ścian odbywa się w czterech etapach:

1. **SetFVF()** – określenie formatu opisu wierzchołków.
2. **SetStreamSource()** – określenie strumienia wierzchołków.
3. **SetIndices()** – określenie tabeli ścian prymitywu.
4. **DrawIndexedPrimitive()** – narysowanie indeksowanego prymitywu.

DrawIndexedPrimitive()

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

- **Type** – rodzaj prymitywu, analogicznie jak dla DrawPrimitive().
- **BaseVertexIndex** – numer pierwszego wierzchołka (zazwyczaj zero).
- **MinIndex** – najmniejszy używany indeks (zazwyczaj zero).
- **NumVertices** – liczba wierzchołków obiektu.
- **StartIndex** – numer początkowego indeksu (zazwyczaj zero).
- **PrimitiveCount** – liczba prymitywów (np. trójkątów).

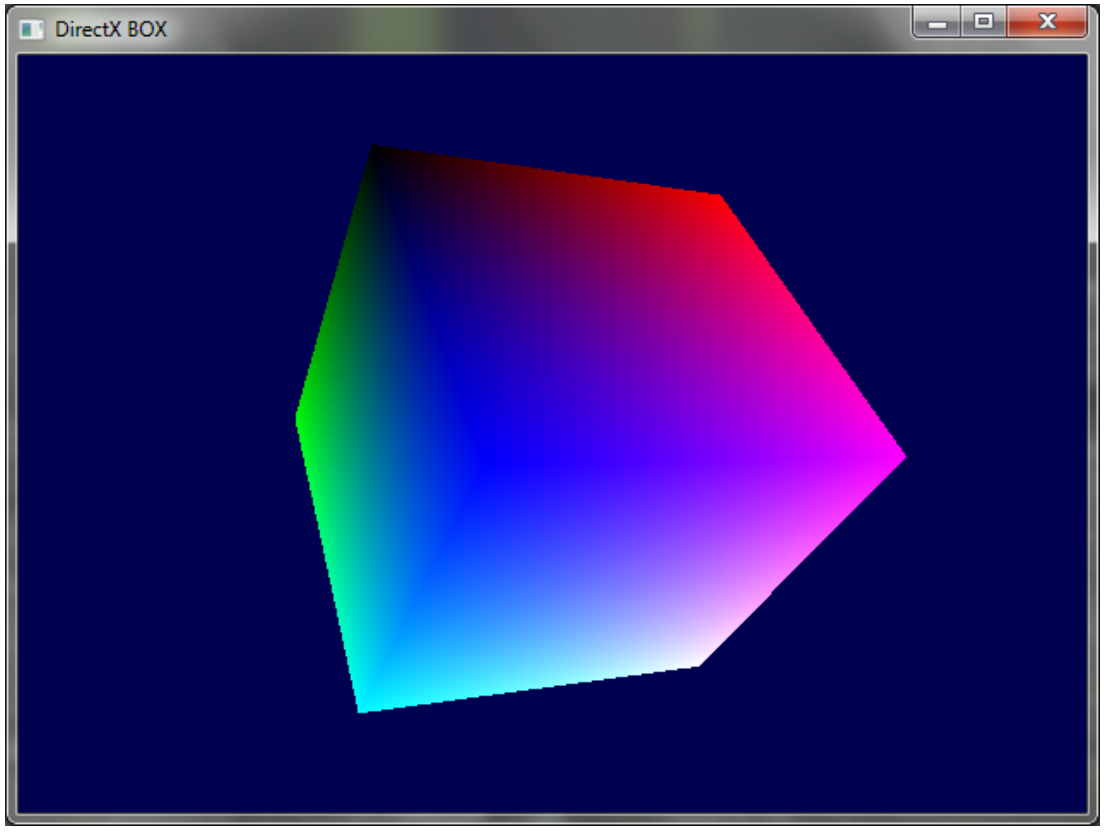
Indeksowany sześcian:

```
d3ddev->SetFVF(MOJ_FVF);

// Strumień wierzchołków:
d3ddev->SetStreamSource(0, v_buffer, 0,
                                    sizeof(MOJ_VERTEX));

// Ustawienie tabeli ścian:
d3ddev->SetIndices(i_buffer);

// Narysowanie indeksowanego sześcianu:
d3ddev->DrawIndexedPrimitive(D3DPT_TRIANGLELIST,
                                    0, 0, 8, 0, 12);
```



Przykładowy program


Windows




KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



Politechnika Świętokrzyska

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY



Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

" Program Rozwojowy Potencjału Dydaktycznego Politechniki Świętokrzyskiej w Kielcach: *kształcenie w nowoczesnych obszarach techniki* "

Program Operacyjny Kapitał Ludzki, Priorytet IV Szkolnictwo wyższe i nauka, Działanie 4.1. Wzmocnienie i rozwój potencjału dydaktycznego uczelni oraz zwiększenie liczby absolwentów kierunków o kluczowym znaczeniu dla gospodarki opartej na wiedzy, Poddziałanie 4.1.1. Wzmocnienie potencjału dydaktycznego uczelni.

Umowa UDA–POKL.04.01.01-00395/09-00