Witam. Ponieważ mamy już podstawę do tego, aby móc kontynuować naszą naukę, więc dzisiaj zajmiemy się wierzchołkami. Dowiemy się też, po co nam **FVF**, czyli **Flexible Vertex Format**, jak definiować nasze własne typy wierzchołków, jak przypisywać im kolor oraz sposób oświetlenia. Poznamy bufor wierzchołków, powiemy trochę o tak zwanych prymitywach i sposobach ich wykorzystania. Na koniec wypełnimy pustą jak na razie przestrzeń pomiędzy parą metod urządzenia **BeginScene()** i **EndScene()**. Przystąpmy zatem do analizy programu (zajmiemy się szczegółowo tylko nowymi elementami, ponieważ część będzie nam już dobrze znana z poprzedniej lekcji). Aby jednak nie pozostawać gołosłownym, umieścimy cały potrzebny kod.

```
#include <d3d8.h>

LPDIRECT3D8             g_pD3D       = NULL;
LPDIRECT3DDEVICE8       g_pd3dDevice = NULL;
LPDIRECT3DVERTEXBUFFER8 g_pVB        = NULL;

struct CUSTOMVERTEX
{
  FLOAT x, y, z, rhw;
  DWORD color;
};

#define D3DFVF_CUSTOMVERTEX ( D3DFVF_XYZRHW | D3DFVF_DIFFUSE )
```

No i jak zwykle zacznijmy od początku.

```
LPDIRECT3D8             g_pD3D       = NULL;
LPDIRECT3DDEVICE8       g_pd3dDevice = NULL;
LPDIRECT3DVERTEXBUFFER8 g_pVB        = NULL;
```

Nasze zmienne, z których dwie już znamy (obiekt główny Direct3D i obiekt urządzenia). Nowy element to obiekt bufora wierzchołków (ang. *vertex buffer*). Bufor to wiadomo - miejsce w pamięci, w którym będziemy przechowywać pewne dane. W naszym przypadku będą to dane wierzchołków. Pod pojęciem "dane" możemy jednak rozumieć bardzo wiele rzeczy. Wiadomo, że bez pewnych na pewno nie będzie można się obejść (dla przykładu współrzędne), część będziemy mogli pominąć, gdyż nie będzie nam potrzebna. Charakterystyczną cechą DirectX-a jest to, że umożliwia nam tworzenie własnych typów wierzchołków. Co to oznacza? Dokładnie to, co napisałem powyżej. Będziemy mogli sobie w strukturze opisującej nasz własny wierzchołek zawrzeć zupełnie dowolne dane dla niego przeznaczone, które potem będziemy mogli wykorzystać w programie w dowolny, wygodny dla nas sposób. Dla potrzeb naszej pierwszej aplikacji będziemy potrzebować co następuje: po pierwsze - współrzędne wierzchołków. Bez tego na pewno nie da się pracować, i nigdy nic nie narysujemy nie mając choćby jednego wierzchołka, zdefiniowanego przez trzy współrzędne: *x*, *y*, *z*. W zasadzie wystarczyłoby to nam w zupełności, no ale figura złożona z wierzchołków jednego koloru może być po pierwsze mało atrakcyjna, a po drugie zdecydowanie mało czytelna, jeśli chodzi o bardziej skomplikowane kształty. My zażyczymy sobie więc posiadać jako dane wierzchołka jeszcze informację o jego kolorze, która pozwoli nam większe szaleństwo :-). Struktura taka będzie miała następującą postać:

```
struct CUSTOMVERTEX
{
  FLOAT x, y, z, rhw;
  DWORD color;
};
```

Tak jak zakładaliśmy, zawiera ona dane potrzebne do umieszczenia wierzchołków w przestrzeni (*x*, *y*, *z*), dane o kolorze i jedną daną potrzebną do operacji w buforze wierzchołków. Jest ona związana bezpośrednio z typem wierzchołków, ponieważ za chwilę się okaże, że użyjemy naszego własnego typu.

```
#define D3DFVF_CUSTOMVERTEX ( D3DFVF_XYZRHW | D3DFVF_DIFFUSE )
```

Ponieważ DirectX umożliwia tworzenie i manipulowanie wierzchołkami w bardzo elastyczny sposób, teraz zdefiniujemy sobie nasz typ wierzchołków. I to jest właśnie **FVF** czyli **Flexible Vertex Format**, o którym mówiłem na początku - czyli po prostu nasz własny typ, który złożony jest z typów podstawowych. Wiemy co będzie zawierał (jakie dane), teraz musimy określić jak te wierzchołki będą przekształcane i wyświetlane przez DirectX. W DirectX Graphics mamy wiele stałych, które określają sposób przekształcania wierzchołków przez funkcje wewnętrzne. Ogólna idea opiera się na tym, aby nam, użytkownikom pozwolić na pełną kontrolę nad tym, co z tymi wierzchołkami się dzieje na scenie. I tak możemy np. zażyczyć sobie, aby DirectX nie transformował wierzchołków (wszystkie obliczenia wykonamy sami), nie oświetlał (nie obliczał natężenia kolorów) i tym podobne. Ponieważ my jednak nie jesteśmy jeszcze aż takimi orłami, aby ze wszystkim radzić sobie sami, na pierwszy ogień każmy jednak odwalić DirectX-owi za nas brudną robotę :-) - użyjemy przetransformowanych wierzchołków. Co to oznacza? Załóżmy, że zapomnieliśmy nagle jak rzutuje się współrzędne 3D na ekran monitora (płaszczyznę 2D). Wydawać by się mogło, że w tym momencie będziemy mieli spory problem, jednak właśnie teraz pomocną rękę wyciąga do nas DirectX Graphics. Daje nam do dyspozycji możliwość użycia takiego formatu wierzchołków,

który spowoduje to, że wewnętrzne funkcje DirectX Graphics będą wiedziały jak zrzutować te współrzędne na ekran. Będzie to jakiś domyślny sposób, którego najczęściej się używa. Pokazana powyżej konstrukcja nakazuje aplikacji wykorzystanie wierzchołków przetransformowanych (stała **D3DFVF_XYZRHW**), czyli my nie będziemy się zajmować przeliczaniem współrzędnych wierzchołków, nie będziemy także liczyć natężenia koloru w poszczególnych punktach figury - zrobi to za nas obiekt urządzenia. Stała **D3DFVF_DIFFUSE** mówi urządzeniu, że w formacie wierzchołka będziemy podawać składową koloru **diffuse**, która oznacza składową rozproszoną (tą, która w praktyce ma największe znaczenie w wyglądzie końcowym renderowanej sceny). Ale o kolorach i składowych barw też jeszcze zdążymy sobie powiedzieć.
Nasz własny typ wierzchołków tworzymy po prostu przez złożenie (za pomocą operatora "|" - podobnie jak ma to miejsce na przykład przy stylach okna) typów zaimplementowanych w DirectX Graphics. Jest to niesamowicie wygodne i, jak nieraz się przekonamy, będzie można manipulować wierzchołkami w naprawdę przedziwne sposoby.

```
HRESULT InitVB()
{
    CUSTOMVERTEX g_Vertices[] =
    {
        { 150.0f, 50.0f, 0.5f, 1.0f, 0xffff0000, },
        { 250.0f, 250.0f, 0.5f, 1.0f, 0xff00ff00, },
        { 50.0f, 250.0f, 0.5f, 1.0f, 0xff00ffff, },
    };

    if( FAILED( g_pd3dDevice->CreateVertexBuffer( 3*sizeof(CUSTOMVERTEX), 0,
D3DFVF_CUSTOMVERTEX,
                                                  D3DPOOL_DEFAULT, &g_pVB ) ) )
    {
        return E_FAIL;
    }

    VOID* pVertices;
   if( FAILED( g_pVB->Lock( 0, sizeof(g_Vertices), (BYTE**)&pVertices, 0 ) ) )
        return E_FAIL;

    memcpy( pVertices, g_Vertices, sizeof(g_Vertices) );
    g_pVB->Unlock();

    return S_OK;
}
```

Teraz skupmy się, bo mamy dużo nowych rzeczy, wszystkie dotyczą oczywiście najważniejszej rzeczy w tej lekcji, jaką są wierzchołki.

```
    CUSTOMVERTEX g_Vertices[] =
    {
        { 150.0f, 50.0f, 0.5f, 1.0f, 0xffff0000, },
        { 250.0f, 250.0f, 0.5f, 1.0f, 0xff00ff00, },
        { 50.0f, 250.0f, 0.5f, 1.0f, 0xff00ffff, },
    };
```

Jak widać, tworzymy sobie tablicę 3 struktur typu **CUSTOMVERTEX**, który zdefiniowaliśmy na początku programu. Wypełniamy ją odpowiednimi wartościami, które zdefiniują nam poszczególne wierzchołki na scenie. Pierwsze trzy wartości to współrzędne wierzchołków w kolejności: *x*, *y*, *z*. Pewnie zastanowicie się skąd te wartości? Co sprytniejsi pewnie się już domyślili, ale krótki eksperyment przekona resztę. Ustawmy dla przykładu współrzędne pierwszego (czerwonego wierzchołka) na 0.0f, 0.0f, 0.0f - i gdzie wylądował? No to chyba dalej już nie muszę tłumaczyć :-). Następna wartość jest nam potrzebna do domyślnego przetwarzania (transformowania) wierzchołków. Ostatnia to składowa rozproszona koloru (ang. *diffuse*), będzie ona decydować o kolorze danego wierzchołka, a co za tym idzie również i figury umieszczonej na naszej scenie. Pierwszych czterech wartości nie muszę chyba nikomu tłumaczyć. Natomiast ostatnia wygląda dość koszmarnie. Ale jeśli przyjrzeć się jej tak dokładniej, to można dostrzec, że zawiera ona po prostu w jednej liczbie cztery wartości dla koloru (po dwa bajty). Pierwsze dwa to tak zwany **kanał alfa** (ang. *alpha channel*), który służy do definiowania przezroczystości. Ponieważ my nie będziemy tworzyć na razie żadnych szyb ani wody, więc ustawiamy tę wartość na 255 (0xff), czyli całkowicie nieprzezroczyste. Następne fragmenty liczby to odpowiednio składowe: czerwona (R - ***R**ed*), zielona (G - ***G**reen*), niebieska (B - ***B**lue*).

```
if( FAILED( g_pd3dDevice->CreateVertexBuffer( 3*sizeof(CUSTOMVERTEX), 0,
D3DFVF_CUSTOMVERTEX,
                                              D3DPOOL_DEFAULT, &g_pVB ) ) )
{
    return E_FAIL;
```

```
}
```

Dalej wykorzystujemy kolejną pożyteczną metodę obiektu urządzenia, jaką jest **CreateVertexBuffer()**. Pierwszy parametr tej funkcji określa rozmiar naszego bufora (w bajtach). U nas są trzy wierzchołki, więc mnożymy rozmiar naszej struktury przez właśnie tę liczbę. Dla wierzchołków reprezentujących elastyczny format (FVF), wielkość ta musi być dostatecznie duża, aby pomieścić choć jeden wierzchołek. Dodatkowo wartość ta nie jest sprawdzana dla wierzchołków, które korzystają z **FVF** czyli nie wbudowanych w Direct3D, stąd wniosek, że jeśli definiujemy własne formaty wierzchołków, to pojawia się kolejne potencjalne źródło błędów. Drugi parametr określa sposób postępowania z wierzchołkami w buforze, można ustalić, między innymi, aby wierzchołki nigdy nie były obcinane lub określić sposób przydziału pamięci. My jednak nie będziemy tu nic ustawiać, ponieważ nie interesuje nas w tym momencie żadna z tych właściwości. Trzeci parametr służy do określenia formatu wierzchołków. Podaje się tutaj kombinację flag opisujących elastyczny format wierzchołków (u nas nasz własny **D3DFVF_CUSTOMVERTEX**) lub jeśli podamy zero to, bufor nie będzie przechowywał wierzchołków posiadających właściwości **FVF**. Czwarty parametr definiuje sposób wykorzystania pamięci (jej rodzaj) do przechowywania wierzchołków. Wartość **D3DPOOL_DEFAULT** określa lokalizację domyślną, najwygodniejszą dla sterownika, najczęściej jest to pamięć karty wideo. Można też tworzyć bufor wierzchołków w innych obszarach pamięci (to znaczy systemowej), ale po dokładniejszy opis, jeśli kogoś interesuje, odsyłam do dokumentacji SDK. Ostatnim parametrem jest, tak samo jak w większości funkcji czy metod obiektów, które posiadają w nazwie słowo "Create", wskaźnik na nowo tworzony obiekt - u nas obiekt bufora wierzchołków.

```
    VOID* pVertices;
    if( FAILED( g_pVB->Lock( 0, sizeof( g_Vertices ), (BYTE**)&pVertices, 0 ) ) )
        return E_FAIL;

    memcpy( pVertices, g_Vertices, sizeof(g_Vertices) );
    g_pVB->Unlock(); return S_OK;
```

Skoro mamy już przydzieloną pamięć na wierzchołki w naszym nowym buforze, możemy przystąpić do jego wypełniania potrzebnymi nam danymi. Ponieważ bufor wierzchołków może być w pamięci urządzenia (karty graficznej), musimy najpierw uzyskać do niego dostęp używając metody **Lock()** obiektu bufora wierzchołków. Pierwszym parametrem w tej funkcji jest offset (inaczej przesunięcie, znawcy programowania typu segment-offset wiedzą o co chodzi) w pamięci wierzchołków, który określi nam do których danych chcemy uzyskać dostęp, w naszym przypadku jest to początek bufora, wartość jest taka, a nie inna ponieważ wypełniamy bufor nowymi wartościami. Następny parametr to ilość bajtów jaką rezerwujemy na nasze dane (u nas będzie to rozmiar tablicy zawierającej dane wierzchołków) - tyle bajtów zostanie zapisane w buforze. Trzeci parametr jest wskaźnikiem na miejsce w pamięci, w którym będzie się zaczynał bufor wierzchołków. Trzeci parametr to flaga określająca sposób blokowania naszego bufora. Zero w naszym przypadku mówi funkcji, że będziemy zapisywać do niego, stosując inne wartości (patrz help) możemy ustawić go na przykład tylko do odczytu. Jeśli zablokowanie bufora wierzchołków się nie powiedzie (tzn. nie będzie można uzyskać do niego dostępu), nastąpi wyjście z funkcji. Następnie zwykłą funkcją **memcpy()** kopiujemy dane pod adres wskazywany przez zmienna "pVertices", czyli najprawdopodobniej gdzieś do pamięci karty graficznej, z tablicy zawierającej dane wierzchołków. W tym momencie wypełniamy nasz bufor potrzebnymi danymi. Po zakończeniu kopiowania musimy oczywiście odblokować bufor wierzchołków metodą **Unlock()**, ponieważ potem moglibyśmy mieć problemy z ponownym uzyskaniem do niego dostępu (np. przy próbie odczytania danych potrzebnych do renderingu).

```
VOID Cleanup()
{
    if( g_pVB != NULL )
        g_pVB->Release();
    if( g_pd3dDevice != NULL )
        g_pd3dDevice->Release();
    if( g_pD3D != NULL )
        g_pD3D->Release();
}
```

W funkcji czyszczącej (wywoływanej na końcu aplikacji) zwalniamy obiekt bufora wierzchołków. Dobrym zwyczajem, w programowaniu z wykorzystaniem obiektów COM, jest zwalnianie obiektów w kolejności odwrotnej do ich tworzenia, skoro więc utworzyliśmy obiekt bufora jako ostatni, zwolnijmy go jako pierwszy. Najważniejsze jest jednak to, aby obiekt **Direct3D** był zwalniany na końcu, jeszcze raz o tym przypominam, bo to bardzo ważne!!!

```
VOID Render()
{
    g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,255), 1.0f, 0
);
    g_pd3dDevice->BeginScene();
    g_pd3dDevice->SetStreamSource( 0, g_pVB, sizeof(CUSTOMVERTEX) );
    g_pd3dDevice->SetVertexShader( D3DFVF_CUSTOMVERTEX );
    g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLELIST, 0, 1 );
```

```
    g_pd3dDevice->EndScene();
}
```

No i w końcu przechodzimy do tego co najważniejsze, czyli rysowania naszej figury. Pierwszy krok, jaki musimy wykonać, to określić źródło, skąd będą pobierane dane dla metody **DrawPrimitive()**. Robimy to za pomocą metody **SetStreamSource()** obiektu urządzenia. Pierwszym parametrem tej metody jest numer strumienia, który będziemy wykorzystywać, a ponieważ u nas jest niewiele danych i są one umieszczone w jednym buforze, to ustawiamy go na 0. Następny parametr jest wskaźnikiem do obiektu bufora wierzchołków, który zostanie skojarzony ze strumieniem. Ponieważ my utworzyliśmy tylko jeden obiekt bufora, więc przekazujemy tutaj jego adres. Trzeci parametr określa krok, co jaki przeskakuje wskaźnik w strumieniu przy odczycie danych, a ponieważ używamy własnego typu wierzchołków, więc tutaj musi być rozmiar jednej struktury naszego typu. Następnym krokiem, jaki powinniśmy wykonać, jest podpowiedzenie Direct3D, jakiego ma użyć vertex shadera. Co to takiego vertex shader, będzie jeszcze okazja powiedzieć, napiszę na pewno na jego temat oddzielny tutorial, bo jest to super rzecz, którą po prostu trzeba znać! Jest to zagadnienie dosyć rozbudowane, ale mówiąc w skrócie, taki shader określa sposób przetwarzania wierzchołków ładowanych na scenę. W naszym przypadku w metodzie **SetVertexShader()** mówimy Direct3D z jakim typem wierzchołków ma do czynienia.
No i w końcu mamy naszą funkcję, na którą tak długo czekaliśmy :-). Metoda **DrawPrimitive()** obiektu urządzenia renderującego jest ta metoda, która spowoduje, że w końcu ujrzymy coś na ekranie. Pierwszy parametr określa rodzaj rysowanych prymitywów. Może ich być kilka, podstawowe to: punkty, linie, trójkąty, pasy i wachlarze trójkątów. Rysowanie każdego rodzaju prymitywu jest związane z pewnymi warunkami, najważniejszym z nich jest ilość wierzchołków odpowiednia dla danego prymitywu. Dokładny opis oczywiście znajduje się w dokumentacji SDK, ale pokrótce mówiąc, po to aby rysować punkty, potrzebne są dla każdego z nich pojedyncze wierzchołki, co jest oczywiste. Aby narysować linie potrzeba nam parzystej liczby wierzchołków (każda para wierzchołków to końce linii), dla trójkątów muszą być co najmniej trzy wierzchołki, co chyba oczywiste. Pasy i wachlarze trójkątów są bardzo pożyteczne ponieważ aby narysować ciąg wielu trójkątów, nie trzeba definiować każdego z nich po kilka razy. Wystarczy spojrzeć na rysunki a wszystko się wyjaśni. Tylko dla pierwszego trójkąta podajemy jawnie wszystkie trzy wierzchołki. Każdy następny korzysta z dwóch wierzchołków poprzedniego trójkąta w wachlarzu lub pasie, a dokładany jest tylko jeden nowy wierzchołek. Jest to bardzo pożyteczna metoda, ponieważ może czasem znacznie ograniczyć ilość ładowanych na scenę wierzchołków, co oczywiście jest ściśle związane z szybkością wyświetlania grafiki. Tak więc zwróćmy na to uwagę. Drugi parametr tej metody to numer pierwszego wierzchołka ładowanego ze strumienia, u nas 0, czyli zaczynamy od samego początku. Ostatni parametr to ilość prymitywów jakie zostaną narysowane. Ponieważ spełniamy warunek ilości wierzchołków dla prymitywu trójkąta (bo taki przecież rysujemy), podanie liczby 1 spowoduje narysowanie jednego trójkąta - prawda, że proste? :-) Gdybyśmy podali jako prymityw np. punkty i podali 1, to zostałby narysowany punkt we współrzędnych, które jako pierwsze znajdują się w strumieniu. Gdybyśmy podali jako ilość prymitywów 3, to zostałyby narysowane punkty we wszystkich wierzchołkach trójkąta. Czy już domyślacie się jak to będzie w przypadku linii? Jeśli nie - spróbujcie a na pewno dojdziecie do pouczających wniosków :-).

I to byłoby na tyle w tej lekcji. Dowiedzieliśmy się jak działa bufor wierzchołków, do czego służy i co najważniejsze jak wyświetlić coś na ekranie. W porównaniu do poprzednich wersji Direct3D używanie strumieni danych i buforów wierzchołków jest na pewno dużo prostsze i przyjemniejsze, a co najważniejsze, daje dużo większe możliwości w manipulowaniu danymi przeznaczonymi do wyświetlenia.

Po skompilowaniu projektu i uruchomieniu programu powinniśmy otrzymać na ekranie załączony poniżej obrazek.