Vademecum programowania Direct3D dla kodu nienadzorowanego

Włodzimierz O. Kubera

Dynamiczny rozwój grafiki komputerowej obserwowany jest już od dłuższego czasu. W latach osiemdziesiątych zadziwiał teledysk „Money for Nothing" zespołu Dire Straits, który jako pierwszy zawierał grafikę cyfrową wizualizującą muzykę teledysku. Współcześnie technika z każdym rokiem upodabnia rzeczywistość wirtualną do realnego świata, a niniejszy artykuł ma na celu przybliżenie czytelnikowi metody programowania umożliwiającej uzyskanie ciekawych efektów multimedialnych za pomocą technologii DirectX firmy Microsoft. Innym celem tego artykułu jest próba popularyzowania środków służących pisaniu gier, wygaszaczy ekranu itp. Obejmuje on prezentację podstawowych pojęć z różnych płaszczyzn grafiki i programowania. Przydatnym czynnikiem procesu poznawczego jest znajomość podstaw języków C++ i C#. W nich są prezentowane przykładowe programy dla DirectX.

Grafika jest jedną z bardziej ekscytujących dziedzin, do której zaprzęgnięto komputery. Analogicznie jest z przetwarzaniem dźwięku w komputerze. Kierunkiem ich rozwoju jest dążenie do odzwierciedlenia rzeczywistości lub tworzenie świata nieistniejącego z coraz większą dokładnością. Według firmy Microsoft, za kilka lat wirtualny świat grafiki będzie w stanie bardzo dokładnie odzwierciedlić rzeczywistość. Już w dniu dzisiejszym, takie efekty jak HDR (metoda na bardziej dokładne odzwierciedlanie kolorów), czy efekt Dopplera w dźwięku karty muzycznej powodują, iż osoba korzystająca z komputera nie zauważa tych symulacji, na tyle są one dokładnym modelem świata rzeczywistego. Ludzkie zmysły przekazują do mózgu wiele informacji. Efektu Dopplera nie rejestrujemy (jest dla człowieka zbyt naturalny). To wynik dużej złożoności urządzeń multimedialnych w warstwie sprzętowej i w oprogramowaniu.

Kodem nienadzorowanym można określić aplikację dostępną w postaci kodu maszynowego, w tym przypadku plik o rozszerzeniu .EXE. Aplikacje kodu nadzorowanego (artykuł ich nie obejmuje) są zgodne z systemem .NET i mogą być wykonywane przez CLR (ang. *Common Language Runtime*, Wspólna Platforma Uruchomieniowa). Aplikacje kodu nadzorowanego są dodatkowo konsolidowane z kodu pośredniego w maszynie, na której są uruchamiane.

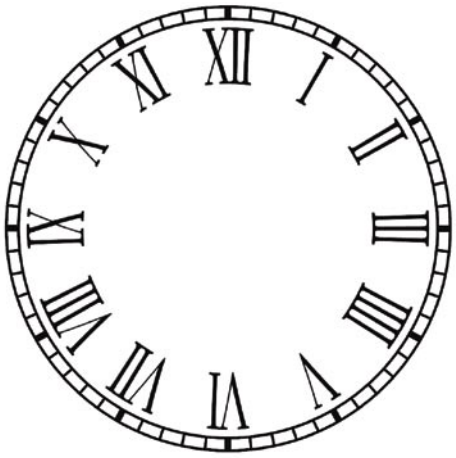
DirectX jest znakiem towarowym firmy Microsoft. Jest to zbiór funkcjonalności wspierających działanie aplikacji multimedialnych oraz wielu innych. Direct3D jest zbiorem narzędzi obsługi grafiki trójwymiarowej pod kontrolą współczesnych wersji systemu Windows. Jest on oczywiście integralnym elementem DirectX. 3D (3 - Dimensional, trójwymiarowej) oznacza, iż narzędzie operuje na trzech wymiarach w osiach X, Y oraz Z. Zet jest oczywiście osią głębi w obrazie. Nie zachodzi konieczność zakładania, czy przyrost wartości na osi Z oddala, czy przybliża punkt do-od użytkownika. Można podczas pracy aplikacji zdecydować, czy ów świat jest „lewo-", czy „prawo-ręczny"

Najbardziej podstawowe pojęcia grafiki komputerowej

W artykule zastosowano wiele intuicyjnych uproszczeń. Wierzchołki oraz figury są modelami obiektów geometrycznych o określonych właściwościach, dlatego poniżej opisane elementy geometryczne mogą być półprzeźroczyste, zawierać kolor, bądź teksturę, a nadal będą nazywane wierzchołkami, trójkątami itp. Pojęcia te zostały opisane z uwagi na ich wagę oraz występowanie w innych technologiach grafiki.

- Piksel (ang. *pixel*) – punkt grafiki komputerowej, często o równej szerokości i wysokości.
- Kolor jest nierzadko zdefiniowany jako RGB (*red* – czerwony, *green* – zielony, *blue* – niebieski). Zazwyczaj występuje jeszcze jeden parametr dla koloru (ARGB, ang. *alpha-blending RGB*). Parametr alfa przyjmuje wartości od 0 do 255. Jest to transparentność materiału, dla którego zdefiniowano barwę). 255 oznacza brak przeźroczystości, 128 oznacza w połowiczną przeźroczystość itd. Dla kolorów występuje analogiczny przedział wartości. Ich intensywność jest zależna od odpowiednich wartości barw źródłowych. Można napotkać inne formaty opisu barw w grafice.
- Tekstura (ang. *Texture*) lub faktura, jest to obraz grafiki, mapa bitowa lub inny obiekt (plik, zasób), który może zostać naniesiony na figurę (zobacz Rysunek 2.). Na Rysunku 2 przedstawiona została przykładowa tekstura przechowywana jako plik graficzny i przedstawiona pierwotnej postaci. Tekstura może zostać naniesiona na figurę o trzech wymiarach, jak widać. Jest to „skórka", która zostanie umieszczona na figurze 3D. Figu-

Autor jest programistą oraz po części administratorem baz danych. Ukończył magisterskie studia informatyczne w Uniwersytecie Warszawskim. Pierwsze publikacje autora w czasopiśmie PC-Kurier pochodzą z pierwszej połowy lat dziewięćdziesiątych. W wolnym czasie zajmuje się m in. pisaniem aplikacji z użyciem różnych technologii DirectX.
Kontakt z autorem: *wlodzimierz@kubera.info*

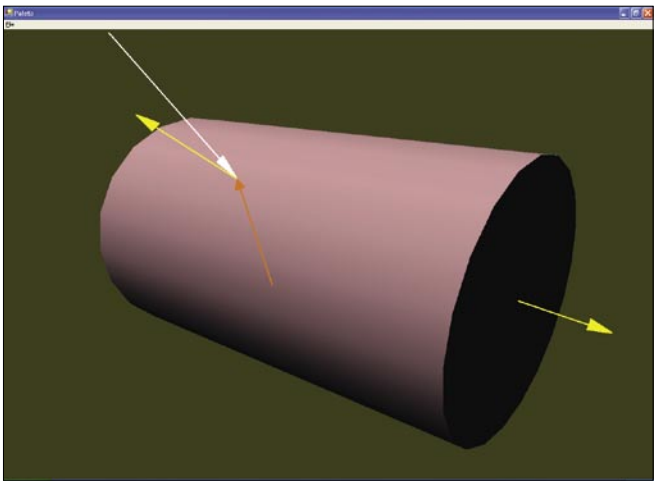


Rysunek 1. *Przykładowa tekstura jako przechowywana jako plik graficzny i przedstawiona pierwotnej postaci*

ra odkształci fakturę, nadając jej kształt oraz powodując reakcję na padające światło.

- Pozycjonowanie tekstury – Tekstura po załadowaniu do systemu jest dostępna jako kwadrat o boku 1.0f. Oznacza to, iż środkiem tekstury jest punkt `<0.5f , 0.5f>`.
- Najmniejszą atomową cechą tekstury jest teksel (ang. *Texel*). Teksel jest to punkt tekstury, z tekseli składa się tekstura.
- Wektor (ang. *Vector*) – w artykule, uporządkowany zbiór wartości zmiennoprzecinkowych (dwu-, trzy- i czterowymiarowych).
- Wierzchołek (ang. *vertex, l. m. vertices*) – punkt łączący boki trójkąta, wierzchołek może być wspólny dla wielu trójkątów jednocześnie.
  - Współrzędne wierzchołka (zazwyczaj trzy wartości typu float), umożliwiają rozmieszczenie wierzchołków w świecie trójwymiarowym.
  - Wektor normalny wierzchołka – wektor o długości jednostkowej (o długości zawsze równej 1.0f) wskazujący kierunek prostopadły do modelowanej płaszczyzny figury w danym wierzchołku, np. dla kuli będzie prostopadły do stycznej płaszczyzny tej kuli. Zazwyczaj służy on obliczeniu intensywności odbitego światła w zależności od usytuowania obserwatora i źródła światła (Rysunek 3.).

    Wektor normalny jest właściwością umowną i może wskazywać dowolny kierunek. Umożliwia to dokładniejsze modelowanie powierzchni zakrzywionych, gdyż projektant obiektu decyduje, o kierunku ustawienia wierzchołeka.
  - Kolor wierzchołka – jest kolorem przypisanym do danego wierzchołka.
  - Pozycja tekstury na wierzchołku jest reprezentowana dwoma wartościami (na dwóch osiach), jest to punkt zaczepienia tekstury.
- Trójkąt (ang. *triangle*) – podstawowa figura trójwymiarowej grafiki; z trójkątów składa się modele dowolnych obiektów, kul, ludzi i innych. Z punktu widzenia obserwatora, trójkąt o bokach (A, B, C) jest różną figurą od (C, B, A) – jedna z figur posiada wierzchołki posortowane z biegiem wskazówek zegara, druga - przeciwnie. Dla systemu trójwymiarowego taka informacja definiuje, iż jedna figura jest podana jako ściana widoczna z zewnątrz a druga, jako ściana od strony wewnętrznej. Trójkątów od wewnątrz modelu często wcale się nie rysuje.
- Podstawowa figura (ang. *primitive*) – zbiór trójkątów zawarty w spójnej strukturze danych (zob. Rysunek 4.).
- Siatka (ang. *mesh*) jest to złożona definicja modelu zawierająca jego charakterystyczne dane. Zazwyczaj są w niej zdefiniowane zbiory trójkątów, czasem materiały oraz nazwy tekstur (Rysunek 5.)
- Flexible Vertex Format (FVF – zmienny format wierzchołka) jest to rozwiązanie umożliwiające reprezentowanie różnych formatów wierzchołka. Należą do niego zazwyczaj pozycja oraz wektor normalny. Jeśli dla danego wierzchołka potrzebna jest też pozycja tekstury na tym wierzchołku, to będzie to kolejny atrybut struktury. Innym przypadkiem jest rysowanie figury nieodbijającej światła. Dla takiej figury jest zbędny wektor normalny. Oznacza to, że podstawowe figury mogą różnić się między sobą. Przed narysowaniem danej figury należy przekazać deklarację, z jakiej struktury rekordów składa się jej tablica, czyli jaki format posiadają aktualnie rysowane wierzchołki.
- Światło (ang. *light*), rodzaje źródeł światła:
  - Światło rozproszone, widoczne w każdym miejscu świata 3D,
  - Światło punktowe, światło padające z określonego punktu,
  - Światło kierunkowe jest to źródło znajdujące się nieskończenie daleko od obiektów grafiki.
- Model jest pojęciem abstrakcyjnym i może być np. siatką lub zbiorem powiązanych logicznie figur podstawowych. Cechami modelu, oprócz odzwierciedlenia obiektu jako takiego, są inne właściwości samodzielne: położenie, kierunek ustawienia figury oraz skala tegoż modelu.
- Macierz (ang. *Matrix*, l. m. *Matrices*) jest prostokątną tablicą zawierającą liczby zmiennoprzecinkowe, którą stosuje się do przechowywania danych o przestrzeni. Stosowanymi macierzami są macierze o rozmiarach: 3 x 3, 3 x 4, 4 x 3 oraz 4 x 4. Oznacza to, iż mają trzy kolumny oraz trzy wiersze itd. Wszelkie przekształcenia w świecie 3D można wykonać przez utworzenie dla nich odpowiednich macierzy przekształceń, a później przemnożenie tych macierzy w wymaganej przez autora kolejności. Wykonywanie mnożeń odpowiada składaniu przekształceń. Macierz



Rysunek 2. *Wektor normalny (żółty), źródło światła (białe) oraz obserwator (brąz)*

rozszerzona składa się z czterech kolumn i czterech wierszy. Jest to tzw. macierz afiniczna. Transformacje w świecie trójwymiarowym można wykonać mnożąc między sobą macierze przekształceń. Pomimo, iż w świecie 3D wystarczą 3 wiersze i 3 kolumny do tzw. przekształceń liniowych, to jednak stosuje się macierze afiniczne, gdyż operacje na macierzach mniejszego rzędu nie umożliwiają wykonania, np. translacji (przesunięcia o dowolny wektor). Obliczenia na macierzach są proste w implementacji, gdyż jest do tego przystosowany podsystem DirectX oraz współczesne karty graficzne. Przekształcenie w grafice polega na złożeniu dowolnej ilości operacji na macierzach. Typowym przypadkiem jest złożenie trzech macierzy (świata, perspektywy oraz projekcji na płaszczyźnie 2D). Macierze te są często rozszerzone do rozmiaru 4 x 4. Umożliwia to wykonanie translacji (przemieszczenia) modelu lub wykonanie na nim kilku przekształceń jednocześnie. Poszczególne macierze pozwalają modyfikować pozycję modelu, dramatyzm perspektywy i wiele innych.

Co to jest Shader?

Wraz z rozwojem kart graficznych następują zmiany w technikach programowania tych urządzeń. Pierwsze karty graficzne programowało się np. modyfikując bity obrazu w ich pamięci wideo. W późniejszym czasie pojawiły się karty zdolne do odtwarzania obrazów 3D. Były to urządzenia w dużym stopniu odtwórcze. Otrzymywały obiekty 3D do odrysowania. Bardzo wiele efektów było otrzymywanych przez rozwiązania trikowe (np. cień narysowany na teksturze) lub poprzez pracę procesora komputera. Karta starszego typu, nie jest zdolna do wykonywania podprogramów na bazie dostarczanych im informacji przez DirectX. Nowe urządzenie jest zdolne nie tylko do wyświetlania, ale i przetwarzania informacji krótkimi programami i jest zbudowane z programowalnego strumienia graficznego (ang. *Programmable Graphic Pipeline*). Przykładem działania Shader-a może być animowanie powierzchni metalowych. Niewielkie zmiany kąta padania światła, np. z powodu ruchu modelu, powodują duże zmiany na powierzchni metalowej. Dla drewnianego modelu, zmiany te nie są tak łatwo zauważalne. Widać więc, że drewno i metal powinny być z animowane różnymi technikami (powinny być „renderowane" oddzielnie). Różne właściwości fizyczne opisuje się przez różne podprogramy. Na tym polega praca programowalnej karty graficznej. Im wyższy numer wersji (Shader Model), tym sprawniejszy, bardziej funkcjonalny i zdolny do wykonywania dłuższych podprogramów.

W starszych wersjach Direct3D (DirectX 8) takie programowanie polegało na opracowywaniu kodu w wirtualnym assemblerze. Posługując się rejestrami oraz wbudowanymi funkcjami programiści uzyskiwali ciekawe efekty w karcie graficznej. W serii dziewiątej nastąpił przełom, gdyż Microsoft wprowadził nowe narzędzia. Stworzono dodatkowo język wysokiego poziomu HLSL (ang. *High-Level Shader Language*). Jest to język, którego programy mogą być kompilowane zgodnie z dowolnym Shader Model-em. Jest jednak wymagane, aby bieżący DirectX oraz posiadana karta graficzna były do tego dostosowane. Opis funkcji z Listingu 1.:

- Argumentem funkcji jest wektor (cztery wartości float – patrz: nazwa typu) przekazywany przez wartość (modyfikator in).
- Jako wynik zwraca analogiczny wektor.
- Do obydwu wektorów przypisano semantykę (kolor).
- Zwracana wartość, to proste operacje arytmetyczne.

Tabela 1. *Wymienione makra wraz z opisem*

Nazwa makra	Kompilacja w trybie Debug	Kompilacja w trybie Release	Zwracana wartość
`V(x)`	Przypisuje wartość zwróconą przez x zmiennej o nazwie hr. Jeśli x zwróciło błąd, to do okienka MessageBox trafia opis wywołania, które jest źródłem błędu.	Przypisuje wartość zwróconą przez x zmiennej o nazwie hr.	`void`, lecz wykona przypisanie
`V_RETURN(x)`	J. w., z tym, że w przypadku błędu następuje opuszczenie podprogramu, a zwracaną wartością funkcji jest x. Makro można używać do przerwania wykonania ciągu operacji po błędzie krytycznym, np przy niemożności załadowania niezbędnego modelu.	J. w., z tym, że w przypadku błędu następuje opuszczenie procedury, a zwróconą wartością funkcji jest x. Makro można używać do przerwania wykonania ciągu operacji po błędzie krytycznym.	`HRESULT`
`SAFE_DELETE(x)`	Usuwa obiekt utworzony operatorem new (wykonując delete) i przypisuje zmiennej opisywanej przez x wartość `NULL`.		`void`, lecz wykona przypisanie
`SAFE_RELEASE(x)`	Zwalnia referencję do danego komponentu (wołając metodę Release tego komponentu, co jest zgodne ze stylem COM – *Component Object Model*, czyli systemem komponentowym firmy Microsoft), a zmiennej opisywanej przez x przypisuje `NULL`.		`void`, lecz wykona przypisanie
`FAILED(Status)`	Informacja, czy Status jest informacją o błędzie. Dobrym rozwiązaniem jest sprawdzanie statusu tą metodą.		`bool`
`SUCCEEDED(Status)`	Informację, czy operacja się powiodła. Nie wszystkie funkcje zwracają `S_OK` dla poprawnego wykonania, czasem przyjmują inne wartości nieujemne. Funkcja ta, również wtedy zwróci wtedy true. Dobrym rozwiązaniem jest stosowanie tego makra zawsze, gdy konkretne wartości statusów nieujemnych są nieznaczące.		`bool`

Listing 1. *Przykład prostej funkcji w HLSL*

```
float4 mono(in float4 InRGB : COLOR) : COLOR
{
    float bw = (InRGB.r + InRGB.g + InRGB.b) / 3.0f;
    return float4(bw * 0.7f, bw * 1.0f, bw * 0.7f, InRGB.a);
}
```

Aktualnie są dostępne trzy poziomy dynamicznego programowania karty:

- `Vertex Shader Stage` (Faza Wierzchołków) – poziom, dla którego są przetwarzane wszystkie wierzchołki figury podstawowej. Można na tym etapie zmienić ich kolor, pozycję itp.
- `Geometry Shader Stage` (Faza geometrii), na tym poziomie można operować na całych trójkątach, możne je również dodawać lub anulować z procesu rysowania. Jest opcja nowa i dostępna aktualnie tylko w Windows Vista.
- `Pixel Shader Stage` (Faza Przetwarzania Pikseli) – po przeprowadzeniu wierzchołków przez wszelkie transformacje następuje proces odrysowania ich w postaci pikseli. Jest to programowalna faza przetwarzania i umożliwia np. uzyskanie efektu oświetlenia z dokładnością do jednego piksela.

Wymagane paramadygmaty języka programowania

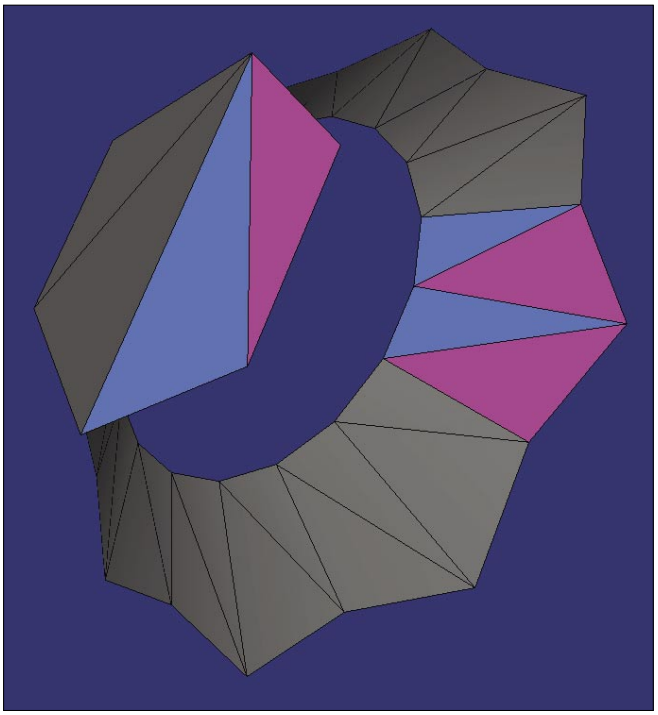
Paradygmaty języka programowania to zbiór ogólnych reguł definiujących styl i semantykę programowania w danym języku. Definiują one metodykę lub wiele metodyk dostępnych dla realizacji programu. Paradygmaty wymienione w tym artykule to:

- Programowanie imperatywne, zazwyczaj jest oparte na zbiorze zmiennych i podprogramów, które mogą być rozmieszczone w różnych pakietach. Innymi słowy, jest to programowanie proceduralne. Wykonanie takich programów polega na realizacji szeregu instrukcji, jedna po drugiej, z możliwością wykonania sekcji sterujących (warunki, pętle, rekurencje oraz skoki bezpośrednie). Przykładem takiego języka jest język C.
- Programowanie obiektowe, programowanie bazujące na modelowaniu rzeczywistości tzw. klasami. Klasy są zbiorami różnych właściwości. Właściwościami są zmienne zadeklarowane wewnątrz (nazywane atrybutami) klas oraz metody mogące się do nich odwoływać. Każda klasa powinna odzwierciedlać jakiś obiekt (np. samochód, faktura itp.). Z klas korzysta się zazwyczaj poprzez jej metody. Są to funkcje przywiązane do danej klasy. Dwie najważniejsze cechy klas to dziedziczenie oraz hermetyzacja. Zalecane jest zapoznanie się z programowaniem imperatywnym oraz obiektowym przed dokładniejszym poznaniem tego typu literatury. Największe korzyści przy tworzeniu takich programów wykazuje znajomość Visual C++ firmy Microsoft. Jako dobrą literaturę na temat obiektowości można polecić książkę [COAD], która uczy poprawnego formułowania kodu obiektowego z użyciem języka C++, edukując jednocześnie o jego podstawach.
- Programowanie komponentowe. Jest metodyka zbliżona do programowania obiektowego. Zmianie jednak ulega dostępność kodu. Często programiści korzystający z komponentów nie mają dostępu do kodu źródłowego komponentu. Uzyskują tylko obiektowe funkcje API oraz zmienne. Utworzenie obiektu nie odbywa się explicite słowem kluczowym new. Referencję do obiektu uzyskuje się z pewnej funkcji API lub z innego komponentu. Komponenty, gdy już się z nich nie korzysta, powinny być zwalniane. Nierzadko elegancką metodą zwalniania jest wykonanie tej operacji w odwrotnej kolejności, niż odbyło się ich tworzenie. W ten sposób giną obiekty, tworzone przez swoje „fabryki", a potem owe fabryki bez martwych referencji (czyli wskaźników na nieistniejące obiekty).

Środowisko programistyczne do kompilowania przykładowej aplikacji

Rozdział ten może dotyczyć osób, które nie zorganizowały sobie własnego środowiska programistycznego W celu skompilowania przykładów są niezbędne następujące elementy:

- Windows XP lub nowszy; w celu uruchamiania przykładów DirectX10 niezbędny jest system Windows Vista.
- Visual C++ Express Service Pack 1; *http://msdn.microsoft.com/vstudio/express/visualc/download/default.aspx*
- DirectX 2007 February; *http://msdn.microsoft.com/directx/sdk/*
- Platformę SDK najlepiej pobrać i zainstalować zgodnie ze wskazówkami na stronie MSDN dotyczącej pobierania i instalowania VC++ Express.



Rysunek 3. *Rodzaje zbioru trójkątów; wachlarz trójkątów po lewej stronie (ang. triangle fan) oraz serpentyna trójkątów (ang. triangle span). Oba warianty ukazują trójkąty ze wspólnymi wierzchołkami, co umożliwia krótszy zapis takiego modelu.*

Visual C++ Express jest darmowym środowiskiem programistycznym typu ang. *Rapid Application Development* (Szybkie Środowiska Rozwijania Aplikacji). Jest to język C++, który może posłużyć tworzeniu (kompilowaniui konsolidacji) praktycznie dowolnej aplikacji napisanej w C i C++. Niestety narzędzie zostało okrojone. Nie ma w nim, np. eksploratora zasobów, tzn. ikonek, map bitowych oraz innych zasobów, które mają być wbudowane do tworzonej aplikacji. Korzystanie z niego wiąże się z tego powodu z problemami. Nie zadziała też opcja tworzenia instalatora aplikacji kodu nienadzorowanego. Chcąc w pełni wykorzystać możliwości aplikacji Windows, należy stosować triki zastępujące opcje niedostępne w tym środowisku. Z tego i kilku innych powodów należy wykonywać regularne kopie zapasowe własnych projektów DirectX, ażeby powrót do działającego projektu nie wiązał się z nadmiarem pracy. W dniu publikacji najprawdopodobniej będzie dostępna nowsza wersja DirectX SDK. Można ją pobrać, a w właściwościach projektu należy umieścić stosowne ścieżki dla kompilatora oraz konsolidatora.

Dla osób, które pierwszy raz się stykają z DirectX można polecić samouczki (ang. *Tutorials*). Są to przystępne aplikacje w stylu *Hello, World*.

Przydatne makrodefinicje DirectX oraz Platform SDK

Makrodefinicje są popularne w C++, a ich częste użycie pochodzi jeszcze z czasów popularności języka C. Przez znawców języków programowania są one krytykowane, gdyż ich stosowanie prowadzi do błędów, niewydajnego użycia języka i utraty czytelności kodu. Jednak w aplikacjach systemowych są bardzo przydatne. Popularna funkcja `D3DXCreateFont` jest w rzeczywistości makrem, które dla aplikacji z zadeklarowanym użyciem UNICODE woła funkcję `D3DXCreateFontW`, a w przeciwnym przypadku woła `D3DXCreateFontA` (ANSI). Znajomość pewnych makr może znacząco wpłynąć na jakość kodu pisanego w VC++. Kod taki jest jaśniejszy, stabilniejszy i bardziej poprawny. Makra takie wołają, bądź zastępują pożyteczne funkcje. Poniższe akapity są wstępem do opisu wybranych makr.

Każdy wskaźnik powinien wskazywać na jakiś obiekt w ramach uruchomionego programu 3D. Oznacza to, iż należy wskaźnikowi przypisać wartość zwracaną przez słowo kluczowe new lub referencję w przypadku przywoływanego komponentu, który dla ułatwienia można przyjąć, iż jest też jest obiektem. Wskaźnik, który nie wskazuje na żaden obiekt, powinien być zawsze równy `NULL`. Czyli przed pierwszym użyciem wskaźnika należy mu przypisać wartość `NULL` lub od razu poprawną wartość. Zwalniając daną klasę lub komponent,

Rysunek 4. *Siatka palety „HEMTEX" jako model rzeczywistego obiektu*

danemu wskaźnikowi należy przypisać NULL. W celu zapewnienia stabilności aplikacji należy, rzecz jasna, sprawdzać czy dany wskaźnik nie jest równy NULL. Stabilność jest kluczowa, np. dla wygaszaczy ekranu, gdyż mogą one odsłonić zawartość ekranu wskutek błędu. W pierwszej kolejności można zalecić wykorzystanie tej metody, a dopiero w drugiej pełną obsługę takiego przypadku. Błędy wskaźników mogą być zagrożeniem dla aplikacji lub nawet systemu operacyjnego.

Istotnym elementem w pisaniu i rozumieniu kodu jest wartość zwracana przez wiele funkcji i metod. Jest ona typu HRESULT. Jest to wartość całkowitoliczbowa ze znakiem. Jeśli zawiera ona S_OK (czyli zero), oznacza to poprawne wykonanie danej funkcji. Jeśli wartość jest dodatnia niesie ona dodatkową informację, np. ostrzeżenie wygenerowane przez komponent. Konkretna wartość ujemna jest kodem błędu. Oczywiście wiele z tych wartości jest zastrzeżonych przez Windows oraz DirectX. Wartą uwagi umiejętnością, ułatwiającą zrozumienie kodu w DirectX SDK, jest znajomość tych makr, gdyż wszystkie one są stosowane w aplikacjach przykładowych. Oto wymienione pewne makra wraz z opisem:

Standardowy proces życia programu Direct3D

Dla omówienia działania typowej aplikacji w Direct3D załączono w tekście uproszczony schemat algorytmu na Rysunku 6. Aplikacja w pierwszej kolejności inicjalizuje tzw. urządzenie Direct3D (ang. *device*). Zdarzenie takie nazywa się OnCreateDevice. Urządzenie na tym etapie już istnieje i można załadować do niego tekstury, macierze czy pewne ustawienia. Jeżeli funkcja użytkownika zwróci wartość błędu (ujemną liczbę). Wówczas proces powstawania zostanie przerwany. Urządzenie istnieje aż do jego zwolnienia. Uwagi wymaga fakt, że obiekty, które są na tym etapie ładowane, muszą być niezależne od stanu urządzenia Direct3D. Komponenty takie są tworzone w trybie automatycznie zarządzanym (ang. *managed*). Jest to deklaracja, iż Direct3D musi przeźroczyście odświeżać stan tych zasobów.

W drugim etapie po powstaniu urządzenia jest wołana funkcja OnResetDevice. Tu można załadować tekstury i inne obiekty zarządzane przez aplikację użytkownika. Jeśli ta operacja zostanie wykonana poprawnie, to system jest gotowy do rozpoczęcia przetwarzania animacji.

Przed narysowaniem każdej klatki platforma SDK woła funkcję OnFrameMove. Służy ona wykonaniu zadań niepowiązanych bezpośrednio z podsystemem grafiki komputerowej. Może to być zaplanowanie stanu modeli zależnych od sztucznej inteligencji lub czasu itp. Kolejną czynnością jest wołanie podprogramu OnFrameRender. Wewnątrz tej funkcji odbywa się rysowanie całego świata 3D. Po jej opuszczeniu najprawdopodobniej zostanie wykonana kolejny raz funkcja OnFrameMove i cykl ten trwa, aż okienko nie otrzyma jakiegoś komunikatu (np. WM_CLOSE). Jeśli okienko odczyta komunikat to obsłuży go standardowo. Przy braku komunikatów animacja jest wykonywana w sposób ciągły wraz z poprzedzającym wywołaniem OnFrameMove. Jeżeli urządzenie zostanie uszkodzone (stan programu), np. zmianą trybu graficznego, to jest wywoływana funkcja OnDeviceLost. W tej funkcji należy zwolnić zasoby, które zostały „uszkodzone". Jeżeli aplikacja nadal nie otrzymała polecenia zakończenia się, to rozpocznie proces ponownie od funkcji OnResetDevice, jest to pętla zdarzeń.

W przypadku zamknięcia programu jest wołana jeszcze funkcja OnDestroyDevice. W tym miejscu trzeba zwolnić zasoby automatycznie zarządzane. Można również zamknąć pliki itp., zależnie od aplikacji.

Krótka nota dla średniozaawansowanych programistów

Microsoft w swojej metodyce sprzed dwóch lat zalecił „przesiadkę" programistów na Windows XP. Chodzi o użycie nowszych technologii bez wsparcia dla poprzednich wersji Windows. Zaletą takiego postępowania jest pisanie prostszych aplikacji, korzystanie z UNICODE oraz kilkuletnie wsparcie ze strony producentów. Dużą zaletą Windows XP jest jego przygotowana obsługa UNICODE. Umożliwia to tworzenie aplikacji wielojęzycznych bez „krzaczków" zamiast liter. Wiadomo, że Microsoft nie planuje wprowadzenia DirectX 10 w Windows poprzedzających Vistę. Czy można pisać aplikacje tak, żeby działały na obu platformach? Jest to możliwe na dwa sposoby:

- Przykłady w SDK są dualne, jedna aplikacja może zostać uruchomiona z DirectX 9 lub z DirectX 10 zależnie od warunków pracy. Dla starszych wersji Windows zadziała zawsze DirectX 9.
- Jest możliwe pisanie aplikacji DirectX 9c tak, ażeby działały względnie optymalnie również w Viście. Vista posiada zaimplementowane dobre wsparcie dla „dziewiątki". Możliwe jest pisanie aplikacji zgodnie z tą wersją systemu. Problemem może być fakt, iż aktualnie Pixel Shader 1.X nie są już dostępne i być może należałoby zmodyfikować taką aplikację. Ładowanie zasobów do systemu DirectX lub bezpośrednio do karty graficznej należy wykonywać również w konkretny sposób. Najpopularniejsze tryby ładowania takich danych to:
- D3DPOOL_DEFAULT – jest to opcja, która ładuje teksturę, listę trójkątów i innych do karty graficznej. Jej semantyka (właściwości, zachowanie) spowoduje, że przy zmianie trybu graficznego, przy przejściu z trybu okienkowego do pełnoekranowego itd. nastąpi usunięcie tej tekstury z karty graficznej. Jest to restart urządzenia DirectX. W takim przypadku należy teksturę załadować jeszcze raz po jej uprzednim zwolnieniu. Jest to opcja popularna dla starszych aplikacji.

Rysunek 5. *Model polskiego zegara z przełomu XIX i XX wieku*

- `D3DPOOL_MANAGED` – wybranie takiej opcji spowoduje, iż DirectX przejmie kontrolę nad zmianami parametrów podsystemu graficznego i za każdym razem zapewni poprawne działanie przy odwołaniu do danego komponentu graficznego.

Dla aplikacji pisanych optymalnie dla Windows Vista, należy stosować drugą opcję. Zmiana trybu działania (okienkowy na pełnoekranowy itp.) w Viście nie wymaga restartu zasobów karty graficznej. Ów restart jest generowany wyłącznie dla kompatybilności wstecz. Ta wersja posiada nową architekturę sterowników graficznych, co jest bezpośrednim powodem takiego sposobu działania.

Jako jeden z przykładów napisanych w DirectX February 2007 można podać projekt Polskiego zegara z przełomu stuleci XIX/XX. Wygaszacz ten został opublikowany na licencji GNU GPL i jest dostępny na płycie czasopisma.

Aplikacja przykładowa

Załączony wygaszacz „Zegar" został zrealizowany głównie wg metodyki obiektowej i jest oparty na jednej hierarchii klas. Wszystkie one implementują obsługę najbardziej podstawowych zdarzeń platformy Direct3D. Klasy zostały zdefiniowane głównie w pliku „ogolne.h". Klasy dziedziczące mogą znajdować się w innych plikach. Każda "monada" posiada konstruktor, który definiuje stan początkowy. Zazwyczaj jest to przypisanie wartości `NULL` na jeszcze nie użyte wskaźniki lub zainicjalizowanie stałych, niezależnych od stanu urządzenia graficznego. W niektórych przypadkach nie zostało zaimplementowane zwalnianie zmiennych ze sterty. W typowym wygaszaczu wiele zmiennych zwalnia się raz, przed zamknięciem procesu. Operację tą w bardziej wydajny sposób wykona system Windows bez skomplikowanego poruszania się po klasach. W tym celu część klas nie posiada destruktorów lub nie zwalniają one sterty. Zmniejszane są natomiast liczniki referencji do komponentów należących do DirectX. Oto lista metod, które są wołane przez framework:

- `OnCreateDevice` – podprogram jest wołany raz w celu załadowania zasobów do systemu lub karty graficznej. Tu są wczytywane tekstury, modele i in.
- `OnFrameMove()` – procedura, która jest wołana przed procesem rysowania grafiki 3D. Służy ona wyliczeniu różnych wartości użytych w późniejszej fazie. Dodatkowo dla tego zdarzenia jest zaimplementowana obsługa wytwarzania dźwięku aplikacji, gdyż nie należy ona do procesu animacji wizji.
- `OnFrameRender` – jest to metoda służąca odrysowaniu świata 3D i musi być zaimplementowana w każdym miejscu hierarchii klas (abstrakcyjna metoda). W przypadku klasy `CFaktura` jest ustawiany materiał i tekstura, które są jej atrybutami. W przypadku klasy `CMesh` jest odrysowany zawarty model.
- `OnDestroyDevice` – Metoda wołana w celu zwolnienia wszelkich komponentów Direct3D i DirectSound3D. Jest wywoływana raz, podczas opuszczania wygaszacza.
- `OnCastShadow` – metoda służąca utworzenia modelu cienia techniką Shadow Volume,
- `GetFVF` – funkcja, która pobiera od klasy rodzaj formatu Flexible Vertex Format. Umożliwia to rozpoznanie, jaką strukturą posługuje się dana klasa. Domyślną wartością jest zero, które świadczy o braku jakiejkolwiek formy FVF. Ułatwia to implementowanie tych samych właściwości przez różne klasy i ich struktury danych.

Bibliografia

- [COAD] Peter Coad i Jill Nicola, Programowanie Obiektowe, Oficyna Wyd. READ ME 1993
- [SAFE] Michael Howard, David LeBlanc, Bezpieczny Kod-tworzenie i zastosowanie, Microsoft Press, 2002
- [FUND] Fundamentals of Audio and Video Programming for Games, Microsoft Corporation 2004
- [GRKOMP] John Lansdown, Grafika Komputerowa, WNT 1990
- [PGP] Kris Gray, Microsoft DirectX 9 Programmable Graphics Pipeline, Microsoft Corporation 2003
- [AV] Peter Turcan, Mike Wasson, Fundamentals Audio and Video Programming for Games, Microsoft Press, 2004
- [MSDN] *http://msdn.microsoft.com/*
- [CODE] Stave McConnell, Code Complete, Microsoft Press 2004

Klasy obudowują funkcjonalności w DirectX w mniej lub bardziej złożony sposób. Korzystanie z klas polega na użyciu zbliżonych, standardowych metod wirtualnych. Wyróżniającą się klasą jest `CKombajnGraf`. Jej atrybutem jest tablica wskaźników do innych obiektów bazujących na `CObiektGraf`. Do tej klasy można dodawać pojedynczo dowolne takie obiekty, a jej wszelkie standardowe metody wykonają analogiczne wywołania we wszystkich obiektach zawartych, np. `OnFrameRender` realizują metody `OnFrameRender` w klasach umieszczonych w tablicy. Posiadając taką implementację nie należy operować na każdym obiekcie z osobna. Inną zaletą jest możliwość łatwego przekształcenia takiej hierarchii w język skryptowy.

Zegar ładuje modele tworząc klasy `CMesh` lub dziedziczące część jej właściwości. Aplikacja generująca model w tablicach przez podprogramy C++ zajęłaby znacząco mniej miejsca. Skorzystanie z tej techniki służy poprawieniu stabilności aplikacji, gdyż część pracy procesu została przekazana do systemu operacyjnego, a modele są rysowane przez mniejszą ilość funkcji API z uniknięciem alokowania i oprogramowywania modeli w C++, co zwiększa też czytelność kodu.

W celu zdobycia lepszej wiedzy na temat pisania wygaszaczy ekranu można pobrać ze strony Microsoft starszą wersję DirectX SDK (9a). Znajduje się tam odpowiednia platforma wraz z dokumentacją.

Zakończenie

Próba realizacji programów z użyciem technologii DirectX wiąże się z trudnościami tworzenia oprogramowania na niskim poziomie, gdyż środowisko to może operować urządzeniami fizycznymi za pośrednictwem sterowników. Trudno przewidzieć wszystkie warianty, w których aplikacja może zostać uruchomiona. Przykłady DirectX SDK zostały przygotowane w celach dydaktycznych i pełne wykorzystanie multimedialnych technologii wymaga tworzenia własnych narzędzi, bibliotek, dokumentów i kodu. W stopce artykułu znajduje się lista interesujących książek, mogących wesprzeć programistów. Jest w Polsce kilka firm, które udostępniają katalogi Microsoft Press. Przeglądanie takiej literatury ma większe znaczenie, gdyż aktualnie pojawił się system operacyjny „Windows Vista", który wprowadza wiele nowych rozwiązań. ■