

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Visual Basic i DirectX. Programowanie gier w Windows

Autor: Wayne S. Freeze

Tłumaczenie: Jarosław Dobrzański

ISBN: 83-7197-806-5

Tytuł oryginału: [Window Game Programming with Visual Basic and DirectX](#)

Format: B5, stron: 410



Książka opisuje krok po kroku tworzenie gry symulacyjnej rozgrywanej się w czasie rzeczywistym, podobnej do słynnej gry SimCity. Gra wykorzystuje Direct3D do wyświetlania grafiki trójwymiarowej oraz DirectInput do komunikacji z użytkownikiem. Opisane zostały interfejsy DirectSound i DirectMusic w stopniu umożliwiającym użycie efektów dźwiękowych i muzyki w grze. Nauczysz się więc tworzenia w pełni trójwymiarowej grafiki a także odtwarzania plików dźwiękowych w formacie MP3. Książka przeznaczona jest dla osób, które już programowały w Visual Basicu.

W książce opisano:

- Wyświetlanie grafiki 3D za pomocą Direct3D
- Sterowanie obrazem: obracanie, przybliżanie i przewijanie
- Tworzenie animowanych postaci
- Operowanie czasem w symulacjach
- Odtwarzanie muzyki i efektów dźwiękowych przy użyciu DirectAudio
- Interakcje użytkownik – program za pomocą DirectInput
- Syntezę mowy przy użyciu interfejsu Microsoft Speech API
- Przygotowywanie grafiki 2D i 3D oraz plików dźwiękowych na potrzeby gry

Autor, Wayne S. Freeze, jest programistą z 25 letnim doświadczeniem i autorem licznych publikacji na tematy związane z językiem Visual Basic, bazą danych Microsoft SQL Server i technologią ASP.



Spis treści

O Autorze	13
Wstęp	15
Tworzenie gry komputerowej	16
Rodzaje gier	16
Visual Basic	17
DirectX.....	18
Gra „Podwodne zakupy”.....	18
Wyzwanie dla czytelnika	19
Dla kogo jest ta książka.....	19
Wymagania sprzętowe	20
Co zawiera książka.....	21
Czego książka nie zawiera	21
Konwencje stosowane w książce	22
Pytania i komentarze	22
Pozostałe zasoby	23
Część I Zanurzenie się w grę	25
Rozdział 1. Od pomysłu do gry	27
Projektowanie gry	27
Projekt gry „Podwodne zakupy”	28
Idea.....	28
Fabuła.....	29
Możliwości gry	30
Obiekty w grze.....	30
Interakcja z użytkownikiem	31
Aspekty kontrolowane przez symulację	33
Zmienne w grze.....	34
Tajne kody	34
Niespodzianki	35
Podsumowanie	35
Rozdział 2. Wprowadzenie do DirectX	37
Czym jest DirectX?	38
Dlaczego DirectX?.....	38
Usługi DirectX	39
Grafika 3D.....	40
Układ współrzędnych dwuwymiarowych.....	40
Układ współrzędnych trójwymiarowych	41

Świat zbudowany z trójkątów	42
Kreślenie wielu trójkątów	43
Kolory	44
Oświetlenie i punkty widzenia	44
Jestem sobie czajniczkiem	45
Instalacja DirectX	46
Uruchamianie programu	47
Inicjalizacja obiektów DirectX	48
Inicjalizacja DirectX3D	51
Ustawianie świateł	54
Tworzenie obiektu 3D	56
Wyświetlanie grafiki	56
Renderowanie pojedynczego kadru	57
Podsumowanie	59
Rozdział 3. Tworzenie grafiki 3D	61
Wprowadzenie do programu trueSpace	61
Wprowadzenie do programów Photoshop i Illustrator	62
Tworzenie grafiki 3D	63
Podstawowe pojęcia	63
Korzystanie z trueSpace	64
Narzędzia trueSpace	65
Tworzenie centrum handlowego	66
Projektowanie wnętrza	66
Konstruowanie pustego wnętrza	67
Podsumowanie	80
Rozdział 4. Grafika dwuwymiarowa w świecie 3D	81
Tworzenie grafiki dwuwymiarowej	81
Przetwarzanie obrazu w Photoshopie	82
Wprowadzenie do Photoshopa	82
Edycja obrazów	84
Korzystanie z warstw	86
Wycinanie fragmentów obrazu	88
Filtry i pozostałe funkcje	89
Tworzenie sklepu	90
Tworzenie prostej tekstury w Photoshopie	91
Tworzenie złożonej tekstury w Photoshopie	93
Tworzenie pustego sklepu w trueSpace	95
Podsumowanie	98
Rozdział 5. Wczytywanie danych gry	99
Szczegóły projektowe	99
Dane na dysku	99
Format pliku inicjalizacyjnego	101
Dane w pamięci	103
Wczytywanie zapisanej gry	103
Klasa Game	103
Korzystanie z klasy Game	104
Zmienne na poziomie modułu w klasie Game	105
Metoda LoadGame	106
Czerpanie danych z pliku z zapisaną grą	107
Odczyt danych za pomocą klasy SwimFile	110
Wczytywanie informacji o centrum handlowym i sklepach	112
Tworzenie sklepów i pozostałych elementów centrum handlowego	113

Właściwości supermarketu	117
Zarządzanie kolekcjami obiektów Anchors	117
Podsumowanie	119
Rozdział 6. Wyświetlanie grafiki 3D	121
Inicjalizacja grafiki	121
Ładowanie centrum handlowego	122
Inicjalizacja DirectX	123
Ładowanie obiektu 3D do pamięci	125
Wyświetlanie informacji graficznych	127
Renderowanie centrum handlowego	128
Główna pętla programu	128
Renderowanie grafiki Direct3D	129
Wyświetlanie tekstu na ekranie	134
Wybieranie obiektów za pomocą klasy DXGraphics	136
Wykrywanie kliknięcia	136
Szukanie wybranego obiektu 3D	137
Sprawdzanie pojedynczego obiektu	137
Uruchamianie programu	139
Podsumowanie	140
Część II Symulowanie rzeczywistości	141
Rozdział 7. Liczby losowe nie są losowe	143
Definicja liczby losowej	143
Liczby losowe w komputerach	144
Generowanie liczb losowych	145
Liczby losowe a Visual Basic	147
Randomize i Rnd	147
Generowanie liczb losowych za pomocą Rnd	147
Rzut kośćmi	149
Zakresy liczb losowych	151
Skalowanie wartości losowych ciągłych	152
Generowanie wartości losowych skokowych	153
Rozkład wartości losowych	153
Rozkład równomierny	153
Rozkład nierównomierny	154
Generowanie skokowych wartości losowych o rozkładzie nierównomiernym	154
Generowanie ciągłych wartości losowych o rozkładzie nierównomiernym	156
Generowanie nierównomiernie rozłożonych liczb losowych	157
Podsumowanie	159
Rozdział 8. Symulowanie rzeczywistości	161
Komputerowe symulacje	161
Symulacje i modele	162
Typy symulacji	162
Elementy modelu symulacyjnego	163
Klienci, serwery, kolejki	163
Czas oczekiwania i obsługi	164
Częstotliwość odwiedzin	164
Systemy wielu serwerów i kolejek	165
Złożone symulacje	165
Inne warianty kolejek	166

Zegar symulacji	167
Zegar postępujący	167
Zegar wyzwalany zdarzeniami	168
Statystyki symulacji	169
Programowanie symulacji	169
Generatory liczb losowych	170
Kolejki	170
Klienci jako obiekty person	172
Restauracja Kelp-Fil-A jako obiekt store	172
Uruchamianie symulacji	173
Inicjalizacja symulacji	173
Sekunda w symulacji	175
Tworzenie klienta	176
Odświeżanie głównego zegara	177
Wyświetlanie statystyk	177
Uruchomienie symulacji taktowanej zegarem	178
Podsumowanie	179
Rozdział 9. Symulacja centrum handlowego	181
Istota symulacji	181
Budowanie szkieletu symulacji	182
Uruchamianie symulacji	182
Kroki symulacji	183
Kroki w centrum handlowym	185
Sterowanie prędkością symulacji	187
Osiedla i konkurencja	188
Budowanie osiedla	188
Tworzenie konkurencji	190
Inicjalizacja symulacji	191
Kluczowe wskaźniki centrum handlowego	191
Wczytywanie godzin otwarcia	192
Monitorowanie centrum handlowego	193
Podsumowanie	198
Rozdział 10. Symulacja klientów	201
Klasa Customer	201
Potrzeby klienta	202
Potrzeby gastronomiczne	204
Zliczanie potrzeb	204
Kiedy pojawią się klienci?	205
Tworzenie klientów	207
Wybór najlepszego centrum	209
Kolejki priorytetowe	211
Przebudowa kolejki	212
Dodawanie elementów do kolejki	212
Symulowanie klienta w centrum	214
Projektowanie automatu skończonego	214
Implementacja automatu skończonego	215
Podsumowanie	217
Rozdział 11. Symulacja sklepów i pieniędzy	219
Sklepy a potrzeby	219
Przechowywanie informacji o sklepie	220

Klienci i sklepy.....	221
Szukanie sklepu	221
Wejście do sklepu	223
Obliczanie momentu wyjścia ze sklepu.....	223
Wyjście ze sklepu	224
Utrzymanie sklepu.....	225
Koniec dnia	225
Koniec miesiąca	226
Wyświetlanie informacji o sklepie.....	226
Zarządzanie budżetem centrum handlowego	229
Pobieranie czynszu.....	229
Koniec miesiąca w centrum handlowym	230
Koniec roku w sklepie	232
Koniec roku w centrum handlowym.....	233
Podsumowanie	234

Część III Od symulacji do gry.....235

Rozdział 12. Sterowanie grą poprzez DirectInput 237

Czym jest DirectInput?.....	237
Korzystanie z DirectInput	238
Integrowanie DirectInput z grą	238
Inicjalizacja DirectInput.....	239
Inicjalizacja DirectInput.....	240
Obsługa zdarzeń DirectInput.....	243
Odczytywanie klawiatury	244
Konwersja kodów klawiatury na ASCII	246
Obsługa myszy.....	247
Przywracanie myszy.....	252
Zamykanie DirectInput	253
Podsumowanie	253

Rozdział 13. Wydawanie poleceń w grze 255

Uruchamianie gry.....	255
Poruszanie się po centrum	256
Sterowanie szybkością gry.....	256
Wybieranie sklepów.....	257
Polecenia menu	257
Polecenia w grze.....	257
Stany gry	258
Nowe funkcje graficzne	259
Tworzenie poleceń	261
Konwersja informacji z klawiatury na polecenia	262
Konwersja informacji z myszy na polecenia.....	266
Wykonywanie poleceń	268
Modyfikacja głównej pętli	268
Wykonywanie poleceń.....	269
Wykonywanie poleceń w stanie OkayState	270
Wykonywanie poleceń w stanie NormalState	270
Wyświetlanie informacji graficznych	272
Kreślenie kursora	272
Kreślenie okienka dialogowego.....	275
Podsumowanie	278

Rozdział 14. Dźwięki i muzyka	279
Dźwięki w centrum handlowym	279
Wprowadzenie do DirectX Audio	280
Udźwiękowanie gry	280
Inicjalizacja DXAudio	281
Odtwarzanie muzyki w tle	283
Ładowanie plików z muzyką do odtwarzania w tle	283
Odtwarzanie muzyki w tle	284
Zatrzymywanie odtwarzania	286
Wstrzymywanie odtwarzania w tle	286
Przywracanie odtwarzania	287
Ustawianie głośności muzyki w tle	287
Pierwszoplanowe efekty dźwiękowe	288
Sprawdzanie statusu odtwarzania	289
Tworzenie efektów dźwiękowych	290
Formaty plików dźwiękowych	290
Poszukiwanie dźwięków	291
Wprowadzenie do programu Cool Edit	291
Digitalizacja dźwięku	292
Nagrywanie dźwięku	294
Edycja pliku dźwiękowego	294
Zarządzanie plikami dźwiękowymi	296
Obróbka dźwięku	297
Podsumowanie	299
Rozdział 15. Mapa centrum i klienci	301
Odwzorowywanie centrum	301
Tworzenie mapy	302
Definiowanie mapy	302
Ładowanie danych mapy	304
Inicjalizacja mapy	306
Zaznaczanie obszarów	307
Diagnostyka mapy	309
Tworzenie klientów	310
Kreślenie szkieletu i łuski	310
Rybie oczy i kolory	313
Zapisywanie ryby	315
Podsumowanie	315
Rozdział 16. Spacer po centrum handlowym	317
Wyświetlanie klientów	317
Ładowanie siatek klientów	318
Zmiany w klasie Customer	318
Zmiany w klasie Queue	319
Wyświetlanie klientów	320
Spacer po centrum handlowym	321
Zarządzanie polami	321
Znajdowanie wejścia	322
Znajdowanie wolnego pola	323
Znajdowanie wolnego pola w sąsiedztwie	323
Sterowanie ruchem klientów	325
Chodzenie po centrum	328
Droga do sklepu	329
Poruszanie się po sklepie	330

Opuszczanie centrum handlowego	331
Obracanie klienta	332
Podsumowanie	332
Część IV Dopracowanie szczegółów gry.....	335
Rozdział 17. Gazeta „Wiadomości Podwodne”	337
Czytanie „Wiadomości Podwodnych”	337
Zapisywanie artykułów	338
Redagowanie artykułów	339
Pobieranie wiadomości	340
Drukowanie gazety	341
Wyświetlanie gazety	342
Przeglądanie gazety	343
Pożyczanie pieniędzy z banku	344
Pożyczanie pieniędzy	344
Wprowadzanie ciągów tekstowych z klawiatury	344
Wyświetlanie okna dialogowego	346
Sterowanie oknem dialogowym	348
Raymond przemawia ludzkim głosem	349
Synteza mowy	349
Wybór głosu	350
Raymond mówi	350
Śledzenie wypowiedzi Raymonda	351
Wyświetlanie Raymonda	352
Podsumowanie	354
Rozdział 18. Uruchamianie centrum handlowego	355
Tworzenie struktury poleceń	355
Projektowanie automatu skończonego	356
Dodawanie definicji	357
Odczyt informacji z wejść	359
Wykonywanie poleceń	360
Zarządzanie sklepami	361
Generowanie listy sklepów	361
Wyświetlanie informacji o sklepie	363
Wyświetlanie listy zaspokajanych potrzeb	365
Rozwiązywanie umowy dzierżawy	369
Obsługa polecenia rozwiązania umowy	369
Likwidacja sklepu	370
Podsumowanie	372
Rozdział 19. Koniec jest dopiero początkiem.....	373
Zapisywanie gry	373
Wprowadzanie nazwy pliku	373
Zapisywanie gry w formacie SwimFile	374
Zapis informacji o grze	375
Zmiany w klasie SwimFile	376
Ładowanie zapisanej gry	377
Uruchamianie gry	377
Wybieranie zapisanej gry	378
Oszukiwanie w „Podwodnych zakupach”	381
Wykrywanie polecenia CheatInputCommand	382
Przetwarzanie kodów	383

Ukrywanie niespodzianek	385
Szukanie niespodzianek	385
Wyświetlanie niespodzianek.....	387
Podsumowanie	388
Skorowidz.....	391

Rozdział 6.

Wyświetlanie grafiki 3D

W rozdziale:

- ◆ Inicjalizacja grafiki
- ◆ Wczytywanie informacji graficznych
- ◆ Wyświetlanie informacji graficznych
- ◆ Renderowanie centrum handlowego
- ◆ Wybieranie obiektów za pomocą klasy DXGraphics
- ◆ Uruchamianie programu
- ◆ Podsumowanie

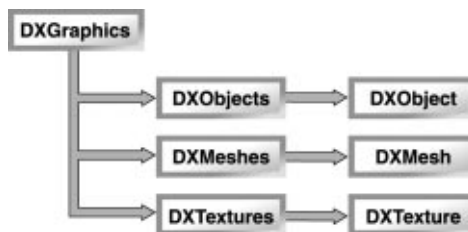
Po stworzeniu trójwymiarowej grafiki centrum handlowego i paru sklepów oraz załadowaniu kilku plików inicjalizacyjnych pora wrócić do DirectX i wyświetlić ją na ekranie.

Inicjalizacja grafiki

Po załadowaniu danych centrum handlowego czas załadować do pamięci grafikę, używając informacji inicjalizacyjnych, wczytanych do obiektów omawianych w poprzednim rozdziale. Obiekty te zawierają nazwy plików obiektów 3D, które mają zostać wyświetlone, wraz z ich względnym położeniem w świecie gry.

Chcąc odizolować grafikę od reszty gry używam oddzielnego zestawu obiektów do przechowywania rzeczywistych obiektów, wyświetlanych na ekranie. Używając klasy `DxGraphics` z rozdziału 2. — „Wprowadzenie do DirectX” — zdefiniowałem sześć nowych klas, które będą przechowywać dane graficzne (zobacz rysunek 6.1).

Rysunek 6.1.
*Informacje graficzne,
 przechowywane
 w kolekcji klas
 Visual Basic*



Centrum handlowe składa się z zestawu obiektów graficznych, z których każdy został oddzielnie stworzony za pomocą narzędzia do modelowania trójwymiarowego, np. trueSpace. Obiekty te to samo centrum handlowe, sklepy, restauracje i wszystkie inne obiekty, wyświetlane w ramach centrum. Każdy z obiektów jest przechowywany w klasie DXObject.

DxObject zawiera informacje niezbędne do wyświetlenia obiektu graficznego na ekranie, w tym informacje z pliku DirectX typu *.X*, zwane również *siatką*, oraz tekstury, którymi zostaną pokryte siatki. W celu zminimalizowania ilości wykorzystywanych zasobów, siatki i tekstury są ładowane tylko raz i przechowywane we własnych klasach. Umożliwia to stworzenie pojedynczego obiektu, który będzie potem używany wielokrotnie bez konieczności każdorazowego ładowania.

Siatki są przechowywane w klasie DXMesh wraz z materiałami i nazwami plików związanych z teksturami, wyświetlanymi na siatkach. Kolekcja DXMeshes jest zbudowana podobnie, jak kolekcje omawiane wcześniej w rozdziale 5. (Anchors, Foods, MallObjects i Stores) i służy wyłącznie do zarządzania zestawem siatek.

Tekstury, podobnie jak siatki, są przechowywane we własnym obiekcie DXTexture, a kolekcje tekstur zawierają się w obiekcie DXTextures. Aby przywołać daną teksturę, potrzebna jest tylko nazwa jej pliku. Wymusza to pewne ograniczenie w sposobie projektowania grafiki. Jako że nazwa pliku musi być niepowtarzalna w obrębie kolekcji, nie można stworzyć dwóch tekstur o tych samych nazwach i przechowywać ich w odrębnych katalogach, chyba że zawierają ten sam obraz.

Przykładowo, nie można korzystać z tekstury o nazwie BlueStripes.BMP, która zawiera wzór z pionowymi paskami i jest przechowywana w katalogu Stores wraz z inną teksturą o nazwie BlueStripes.BMP, zawierającej wzór z ukośnymi paskami i przechowywanej w katalogu Anchors. Zostanie wtedy użyta tekstura z pierwszego z załadowanych plików, a drugi z nich zostanie uznany za duplikat pierwszego.

Ładowanie centrum handlowego

Wyświetlanie grafiki przez DirectX można podzielić na dwie fazy. Najpierw grafika musi zostać załadowana do pamięci, a potem wyświetlona na ekranie. Ładowanie grafiki to dość skomplikowany proces, jednak po sprowadzeniu go do postaci zawierającej tylko kluczowe elementy nie wygląda już tak strasznie.

Inicjalizacja DirectX

W metodzie `InitGame` podsystem graficzny został zainicjalizowany poprzez wywołanie `InitGraphics` (zobacz wydruk 6.1). `InitGraphics` odpowiada za dwie rzeczy: po pierwsze musi stworzyć nową instancję klasy `DXGraphics` i wywołać metodę `InitDX`, aby zainicjalizować urządzenie `Direct3D`; po drugie, musi załadować pliki `.X` na podstawie załadowanej informacji inicjalizacyjnej.

Wydruk 6.1. *Metoda Form1.InitGraphics*

```
Function InitGraphics() As Long

    Dim Result As Long
    Dim a As Anchor
    Dim f As Food
    Dim m. As MallObject
    Dim s As Store

    Set dx = New DXGraphics
    Set dx.DebugObject = GameDebugger

    dx.Windowed = True
    dx.hWnd = Picture1.hWnd
    dx.ShowFrameRate = True
    dx.Height = Picture1.Height / Screen.TwipsPerPixelY
    dx.Width = Picture1.Width / Screen.TwipsPerPixelX
    dx.DefaultCameraPoint = GameObj.CameraPoint
    dx.DefaultViewPoint = GameObj.ViewPoint
    dx.DefaultRotation = GameObj.Rotation

    Result = dx.InitDX
    if Result <> 0 Then
        GameDebugger.WriteLine "Inicjalizacja urządzenia Direct3D nie powiodła się".
        ➔Result
        Unload Me
    End If

    With GameObj.Mall
        dx.LoadMeshFromDisk App.Path & "\Malls", .DXFilename, _
            .Location, .Rotation, DXObjectMall
    End With

    For Each a In GameObj.Mall.Anchors
        dx.LoadMeshFromDisk App.Path & "\Anchors", a.DXFilename, a.Location, _
            a.Rotation, dxobjectanchor
    Next a

    For Each f In GameObj.Mall.Foods
        dx.LoadMeshFromDisk App.Path & "\Foods", f.DXFilename, f.Location, _
            f.Rotation, dxobjectfood
    Next f
```

```

For Each m In GameObj.Mall.MallObjects
    dx.LoadMeshFromDisk App.Path & "\Objects", m.DXFilename, m.Location, _
        m.Rotation, dxobjectfood

Next m

For Each s In GameObj.Mall.Stores
    dx.LoadMeshFromDisk App.Path & "\Stores", s.DXFilename, s.Location, _
        s.Rotation, dxobjectstore

Next s

End Function

```

Procedura `InitGraphics` na początku tworzy nowy obiekt `DXGraphics`, ustawia różne właściwości i wywołuje metodę `InitDX` z klasy `DXGraphics`. W odróżnieniu od wersji tej procedury z rozdziału 2. zawiera ona właściwość `Windowed`, która decyduje o tym, czy program jest wyświetlany w okienku (`True`) czy na pełnym ekranie (`False`).



Okienko czy pełny ekran

Mimo, że `DXGraphics` obsługuje tryb pełnoekranowy, nie polecam korzystania z niego, ponieważ jedyne elementy sterujące interfejsu użytkownika pozostają wtedy poza obszarem wyświetlania `DirectX`, co uniemożliwia sterowanie programem.

Teraz już można przyporządkować wartości dla `ViewPoint`, `CameraPoint` i `Rotation` załadowanego pliku z zapisaną grą. Można również określić szerokość i wysokość ekranu w pikselach za pomocą odpowiednio: `Width` i `Height`. Jest to istotne dla określenia perspektywy, wyświetlanej w oknie o zmiennym rozmiarze, jak i do ustalenia rozmiaru ekranu gry w trybie pełnoekranowym.

Zakładając, że `DirectX` zostało poprawnie zainicjalizowane wywołaniem `InitDX`, można załadować poszczególne pliki `.X`. Wykonuje to pętla `For Each`, która dla każdego pojedynczego obiektu 3D centrum handlowego wykonuje metodę `LoadMeshFromDisk`, aby załadować go do pamięci.

Kiedy obiekt zostanie załadowany do pamięci, do jego wyświetlenia na ekranie potrzebne są, poza nazwą pliku, trzy dodatkowe informacje: lokalizacja obiektu w centrum handlowym, czyli właściwości `X` i `Z` obiektu, informacja o tym, jak obracać obiekt, aby prawidłowo umieścić go w centrum handlowym i typ obiektu.



Skąd się wzięło Z?

Przy tworzeniu obiektów w programie `trueSpace`, a następnie ich wyświetlaniu poprzez `DirectX`, można zauważyć, że osie `Y` i `X` zostają zamienione. W `trueSpace` oś `Z` wskazuje górę, podczas gdy w `DirectX` górę wskazuje oś `Y`. Wymaga to od programisty zamiany wartości `Y` i `Z` podczas konwersji współrzędnych z `trueSpace` do `DirectX`.

Ładowanie obiektu 3D do pamięci

Metoda `LoadMeshFromDisk` (patrz wydruk 6.2) ładuje do pamięci obiekty 3D, stworzone w programie `trueSpace`, aby umożliwić ich późniejsze wyświetlenie. Wymaga ona kilku parametrów, takich jak nazwa siatki, miejsce w przestrzeni trójwymiarowej, gdzie siatka ma zostać wyświetlona, sposób obracania siatką, umożliwiający jej prawidłowe umieszczenie w centrum handlowym i typ wyświetlanego obiektu. Wszystkie te informacje zostaną zapisane w `DXObject` i dodane do kolekcji `DXObjects`, będącej elementem obiektu `DXGraphics`.

Wydruk 6.2. Metoda `LoadMeshFromDisk`

```
Public Function LoadMeshFromDisk (MeshPath As String, MeshFile As String, _
    Location As D3DVECTOR, Rotation As Single, Typ As DXObjectTypes) As Long

    Dim mtrlbuffer As D3DXBuffer
    Dim i As Long
    Dim fname As String
    Dim o As DXObject
    Dim m As DXMesh
    Dim t As DXTexture
    Dim nMaterials As Long
    Dim MeshMat As D3DMATERIAL8

    On Error Resume Next

    If Len(MeshFile) = 0 Then
        LoadMeshFromDisk = 0
        Exit Function

    End If

    fname = MeshPath & "\" & MeshFile

    DXDebugger.WriteLine "Ładowanie siatki: " & fname
    Set o = New DXObject
    o.Name = MeshFile
    o.mType = TexturedMesh
    o.oType = Typ
    o.Location = Location
    o.Rotation = Rotation
    o.MeshFilename = MeshFile

    If DXmshs(MeshFile) Is Nothing Then
        Set m = New DXMesh
        Set m.Mesh = d3dx.LoadMeshFromX(fname, D3DXMESH_MANAGED, d3dDevice, _
            Nothing, mtrlbuffer, nMaterials)
        If m.Mesh Is Nothing Then
            DXDebugger.WriteDXErr "Nie można załadować siatki". Err
            LoadMeshFromDisk = Err.Number
            Exit Function
        End If

        m.Name = MeshFile
        m.Materials = nMaterials
```

```

For i = 0 to nMaterials - 1
    d3dx.BufferGetMaterial mtrlbuffer, i, MeshMat
    MeshMat.Ambient = MeshMat.diffuse
    m.MeshMaterials(i) = MeshMat
    fname = d3dx.BufferGetTextureName(mtrlbuffer, i)
    m.MeshTextures(i) = fname

    If Len(fname) > 0 Then
        If Dxtexs(fname) Is Nothing Then
            Set t = New DXTexture
            Set t.Texture = d3dx.CreateTextureFromFile(d3dDevice, MeshPath _
                & "\" & fname
            t.Name = fname

            DXtexs.Add t

        End If

    End If

Next i

DXmshs.Add m.

End If

DXobs.Add o

LoadMeshFromDisk = 0

End Function

```

Na początku metoda sprawdza, czy nazwa pliku nie jest pusta. Jeżeli jest, opuszcza funkcję, niczego nie ładując. Następnie wysyła echo z nazwą ładowanego pliku do *Debug.LOG*. Potem konstruuje kompletną nazwę pliku, łącząc z sobą parametry *MeshPath* i *MeshFile* oraz tworzy nowy egzemplarz obiektu *DXObject* do przechowywania nowej siatki. Następnie ustawia właściwości stworzonej kopii obiektu *DXObject*, przyporządkowując im wartości związane z ładowanym obiektem 3D.

W dalszej kolejności metoda przesyła nazwę pliku siatki do obiektu *DXmshs*, aby zorientować się, czy należy załadować plik siatki z dysku. Jako że domyślną metodą dla tej klasy jest *Item*, nie trzeba jej wymieniać po nazwie klasy, aby została użyta. W ten sposób, kiedy *DXmshs(MeshFile)* zwraca *Nothing*, wiadomo, że trzeba załadować siatkę. Jeżeli siatka jest już załadowana, funkcja dodaje nowy obiekt *DXObject* do kolekcji *DXObjects* i wraca do programu wywołującego.

Aby załadować siatkę, należy stworzyć nową instancję obiektu *DXMesh*, a następnie użyć *LoadMeshFromX* w celu załadowania pliku *.X* do pamięci. Metoda ta wymaga paru dodatkowych parametrów poza nazwą pliku do załadowania. *D3DXMESH_MANAGED* oznacza, że dane zostaną załadowane do obszaru pamięci zarządzanego przez *Direct3D*, co poprawi ogólną wydajność. Parametry *mtrlbuffer* i *nMaterials* zawierają informacje o sposobie wyświetlenia powierzchni czołowej każdej siatki.

**Nie powiodło się!**

Najczęstszym błędem podczas ładowania plików `.X` jest próba załadowania nieistniejącego pliku. Jeżeli obiekty 3D nie wyświetlają się na ekranie, należy sprawdzić zawartość pliku `Debug.LOG`. Podczas ładowania siatki nazwa jej pliku zostaje dodana do pliku dziennika. Jeżeli wystąpił problem, informacja o błędzie będzie zapisana tuż po nazwie pliku, do którego się odnosi.

Po załadowaniu siatki trzeba wydobyć parę informacji z bufora tekstur. Konkretnie chodzi o nazwę pliku tekstury, którą uzyskujemy, wykorzystując metodę `BufferGet-TextureName` i informację o oświetleniu, otrzymywaną za pomocą metody `BufferGet-Material.mtrlbuffer` zawiera tablicę wartości, dlatego należy użyć indeksu, aby odwołać się do potrzebnej wartości.

Po uzyskaniu informacji o oświetleniu trzeba ustawić atrybut `Ambient` na taką samą wartość, jak `diffuse`. Pomaga to w prawidłowym wyświetlaniu kolorów. Wtedy można zapisać tę informację do właściwości `MeshMaterials` obiektu `DXMesh`.

W ramach tej samej pętli można również otrzymać nazwę pliku, związaną z daną teksturą. Nazwa posłuży do sprawdzenia w kolekcji tekstur (`Dxtexs`), czy dana tekstura została już wcześniej załadowana. Jeżeli nie, tworzy się nową instancję obiektu `DXTexture` i za pomocą metody `CreateTextureFromFile` otrzymuje się teksturę. Następnie można dodać teksturę do obiektu `DXTextures` i powtarzać całą operację, dopóki wszystkie informacje z bufora `mtrlbuffer` zostaną przetworzone.

Na koniec funkcja dodaje `DXObject` do kolekcji `DXObjects`, ustawia wartość zwrótną na zero, co oznacza, że operacja się powiodła i opuszcza metodę.

**Działa, ale zbyt wolno**

Mimo że opisana technika działa, wstępne przetworzenie wszystkich plików, związanych z centrum handlowym, zajmuje dużo czasu. Jest wiele technik, z których można skorzystać, aby przyspieszyć ten proces, jednak nigdy nie pozbedziemy się pewnego opóźnienia podczas ładowania grafiki. To dlatego większość gier podczas ładowania wyświetla atrakcyjne obrazy lub filmy. Pomaga to zająć czymś gracza, podczas gdy w tle ładują się niezbędne informacje.

Wyświetlanie informacji graficznych

Proces wyświetlania grafiki 3D jest stosunkowo prosty w teorii. Na początku należy zdefiniować otoczenie trzema macierzami — światła, widoku i rzutu. Potem trzeba wyczyścić ekran i dodać każdy ze zdefiniowanych obiektów do świata. W końcu należy wyświetlić grafikę. Mimo że proces ten wydaje się prosty, w praktyce może się niezwykle skomplikować.

Macierz światła opisuje, gdzie znajduje się początek układu współrzędnych świata gry. Jest to zwykle najprostszą z trzech macierzy — przynajmniej w sensie inicjalizacji. Jednakże jej wartości będą się zmieniać podczas dodawania nowych obiektów do świata gry.

Macierz widoku opisuje lokalizację kamery, przez którą widziany jest świat. Kamera ta działa podobnie, jak oko (*eye*) w programie trueSpace, z tą różnicą, że oko jest nieruchome i widziany świat porusza się względem niego, podczas gdy współrzędne kamery oznaczają jej umiejscowienie w świecie gry, a widok (*viewpoint*) reprezentuje miejsce w przestrzeni, na które skierowana jest kamera.

Macierz rzutu opisuje, jak współrzędne świata gry są przyporządkowane współrzędnym ekranu. Sposób przyporządkowania decyduje o tym, czy świat będzie widoczny jak w teleskopie czy jak w obiektywie szerokokątnym, albo czy obiekty będą widziane jako wysokie i wąskie czy jako niskie i szerokie. Przypadki te to oczywiście wartości skrajne. Tylko ich właściwe ustawienie może uczynić grę ciekawszą.

Kolejnym problemem jest ustawienie oświetlenia. Praca w trueSpace wymagała od czasu do czasu poruszania czerwonymi gwiazdkami oznaczającymi punkty świetlne, aby lepiej widzieć tworzony właśnie obiekt. DirectX również umożliwia niezależne oświetlenie każdego obiektu, jak również stosowanie pełnego oświetlenia, które spowoduje, że obiekty będą oświetlone tak samo z każdej strony. Co prawda dzięki temu staną się bardziej widoczne, ale pozbawienie ich strony ciemnej i jasnej osłabi efekt trójwymiarowości.

Renderowanie centrum handlowego

Po załadowaniu wszystkich informacji dotyczących centrum handlowego można przystąpić do jego wyświetlania. W tym celu należy pobrać informacje graficzne, załadowane wcześniej metodą `LoadMeshFromDisk` i wyświetlić je na ekranie. Dodatkowo trzeba ustalić sposób przesuwania widoku na ekranie i kontrolować pozostałe aspekty wyświetlania świata 3D.

Główna pętla programu

Procedura `PlayGame` (zobacz wydruk 6.3) to główna pętla programu. Wywołuje ona metodę `Render` z klasy `DXGraphics`, a następnie procedurę `DoEvents`, umożliwiającą systemowi Windows wykonywanie innych zadań. Bez tego odwołania gra będzie blokować inne aplikacje, co może doprowadzić do zawieszenia się systemu.

Wydruk 6.3. Procedura `Form1.PlayGame`

```
Sub PlayGame()  
  
Do While GameActive  
    dx.Render  
    DoEvents  
  
Loop  
  
End Sub
```

Renderowanie grafiki Direct3D

Metoda `Render` to kolejna z procedur, które na pierwszy rzut oka wyglądają groźnie (zobacz wydruk 6.4). Jednakże w praktyce nie jest zbyt skomplikowana. Na początku deklaruje kilka tymczasowych zmiennych i wyłącza kontrolę błędów. W ten sposób kontrola błędów przechodzi w gestię programisty.

Wydruk 6.4. Metoda `DXGraphics.Render`

```
Public Function Render() As Long

    On Error Resume Next

    Dim matTemp As D3DMATRIX
    Dim matTrans As D3DMATRIX
    Dim vecTemp As D3DVECTOR
    Dim vp As D3DVIEWPORT8
    Dim mm As D3DMATERIAL8
    Dim o As DXObject
    Dim i As Long

    D3DXMatrixIdentity matWorld
    d3dDevice.SetTransform D3DTS_WORLD, matWorld

    D3DXMatrixIdentity matView
    D3DXMatrixRotationY matView, Rotation
    D3DXMatrixLookAtLH matTemp, CameraPoint, Viewpoint, vec3(0, 1, 0)
    D3DXMatrixMultiply matView, matView, matTemp
    d3dDevice.SetTransform D3DTS_VIEW, matView

    D3DXMatrixPerspectiveFovLH matProj, pi / 4, Height / Width, 0.1, 100
    d3dDevice.SetTransform D3DTS_PROJECTION, matProj

    D3DXVec3Subtract vecTemp, CameraPoint, ViewPoint
    d3dDevice.GetViewport vp
    D3DXVec3Unproject vecLight, vecTemp, vp, matProj, matView, matWorld
    D3DXVec3Scale vecLight, vecLight, -1
    d3dLight.Direction = vecLight
    d3dDevice.SetLight 0, d3dLight
    d3dDevice.LightEnable 0, 1

    d3dDevice.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, &H404040, 1#, 0

    d3dDevice.BeginScene

    For Each o In DXObs
        D3DXMatrixIdentity matTemp
        D3DXMatrixRotationY matTemp, o.Rotation
        D3DXMatrixTranslation matTrans, -o.Location.X, -o.Location.Y, -o.Location.Z
        D3DXMatrixMultiply matTemp, matTemp, matTrans
        D3dDevice.SetTransform D3DTS_WORLD, matTemp

        If o.mType = SimpleMesh Then
            DXmshs(o.Name).Mesh.DrawSubset 0
        End If
    Next o
End Function
```

```
ElseIf o.mType = TexturedMesh Then
    With DXmshs(o.MeshFilename)
        For i = 0 To .Materials -1
            d3dDevice.SetMaterial .MeshMaterials(i)
            d3dDevice.SetTexture 0, DXtexs(.MeshTextures(i)).Texture
            .Mesh.DrawSubset i

        Next i
    End With

ElseIf o.mType = SelectedMesh Then
    With DXmshs(o.MeshFilename)
        For i = 0 To .Materials - 1
            mm.Ambient.a = 1
            mm.Ambient.r = 1
            mm.Ambient.b = 0.5
            mm.Ambient.g = 0.5
            mm.Diffuse = mm.Ambient
            d3dDevice.SetMaterial mm
            d3dDevice.SetTexture 0, _
                DXtexs(.MeshTextures(i)).Texture
            .Mesh.DrawSubset i

        Next i
    End With

End If

Next o

If ShowFrameRate Then
    DrawFrameRate
    DrawViewPoint

End If

d3dDevice.EndScene

Err.Clear
d3dDevice.Present ByVal 0, ByVal 0, 0, ByVal 0
If Err.Number <> 0 Then
    Render = ResetDX

End If

End Function
```

Definiowanie funkcji przekształceń

Pierwszym krokiem podczas wyświetlania grafiki trójwymiarowej jest zainicjalizowanie macierzy świata. Metoda `D3DXMatrixIdentity` definiuje macierz jednostkową *i*, korzystając z metody `SetTransform`, przyporządkowuje jej wartości macierzy świata. Jako że najlepiej zaczynać od początku, współrzędne zostały ustawione na punkt (0, 0, 0).

Ustawianie widoku jest trochę bardziej złożone. Program zaczyna się od przyporządkowania widoku do macierzy jednostkowej, następnie używa wartości `Rotation`, by określić, jak obrócić widok względem osi `Y`. Zamiast używać skomplikowanej matematyki do tego typu obliczeń, po prostu korzysta z procedury `D3DXMatrixRotationY`, która obraca macierzą widoku. Należy pamiętać, że kąty obrotu są wyrażone w radianach, a nie w stopniach.



Stopnie czy radiany?

Pełny okrąg ma 360 stopni albo 2π radianów. Tak więc aby zamienić wartość kąta w stopniach na radiany, należy pomnożyć jego wartość przez $\pi/180$. Z kolei aby zamienić kąt wyrażony w radianach na stopnie, należy pomnożyć go przez $180/\pi$.

Następnym krokiem jest utworzenie tymczasowej macierzy za pomocą procedury `D3DXMatrixLookatLH`, która korzysta z wartości `CameraPoint` i `ViewPoint` wraz ze specjalnym wektorem, opisującym, która z osi skierowana jest do góry. W tym przypadku wektor `vec3(0, 1, 0)` wskazuje na oś `Y`. Ostateczną macierz, opisującą bieżący widok, uzyskujemy mnożąc ze sobą powyższe macierze przy użyciu procedury `D3DXMatrixMultiply`.

Macierz rzutu zostaje utworzona przy użyciu procedury `D3DXMatrixPerspectiveFovLH`. Tworzy ona macierz, zawierającą informacje o perspektywie i polu widzenia. Pole to jest typowo opisywane jako liczba π , dzielona przez pewną wartość. $\pi/4$ oznacza widok normalny, porównywalny do obrazu widzianego przez 50 mm obiektyw w 35 mm kamerze. Mniejsze wartości, czyli π/x dla x większego od 4, dadzą efekt teleskopowy, a większe — efekt obiektywu szerokokątnego. Należy zwrócić uwagę, że wartości różne od 4 powodują zniekształcenie widzianych obiektów, dlatego wypada używać ich z rozwagą.

Parametr `Height/Width` to stosunek szerokości ekranu do jego wysokości. Pomaga on w prawidłowym wyświetlaniu obrazu na ekranie, który nie jest kwadratowy. Po ustawieniu tego parametru na 1 przy ekranie o szerokości większej niż jego wysokość, obraz zostanie rozciągnięty w szerokość, sprawiając, że obiekty staną się za niskie i za szerokie. Jeżeli szerokość ekranu jest mniejsza niż jego wysokość, wyświetlane elementy staną się zbyt wysokie i zbyt wąskie. Użycie odpowiedniej wartości `Height/Width` umożliwi DirectX prawidłową kompensację, aby obiekty były widziane w takich proporcjach, w jakich zostały zaprojektowane.

Dwa ostatnie parametry, `.1` i `100`, opisują granice widoczności. Wszystko, co znajduje się bliżej kamery, niż `.1` lub dalej niż `100`, nie będzie widoczne.

Ustawienie oświetlenia

Ustawienie oświetlenia jest nieco skomplikowane, ponieważ program korzysta z kilku wektorów i wykorzystuje operacje na macierzach do obliczenia pozycji punktów świetlnych. Na początku odejmuje wektor `CameraPoint` od wektora `ViewPoint` i zapisuje wynik do tymczasowego wektora. Następnie korzysta z procedury `D3DXVec3Unproject`, konwertującej tymczasowy wektor do wektora opisującego lokalizację punktu świetlnego, korzystając z transformaty widoku, która została ustawiona wcześniej.

Otrzymanej wartości można użyć do ustawienia kierunku padania światła. Odpowiada za nią właściwość `Direction` obiektu `d3dLight`. Metoda `SetLight` dodaje punkt świetlny do świata gry jako punkt o numerze 0, a metoda `LightEnable` po prostu włącza światło.

Poprzez ustawienie światła w taki sposób, aby skierowane było od kamery w kierunku jej widoku, zapewniamy, że wszystkie obiekty oglądane przez gracza są odpowiednio oświetlone.

Wyświetlenie pojedynczego ujęcia

Urządzenie Direct3D (`d3dDevice`) zostało zainicjalizowane w taki sposób, że grafika jest przechowywana w dwóch buforach. Podczas gdy jeden z nich jest aktywnie wyświetlany, drugi jest udostępniony w celu załadowania danych do wyświetlania. Po załadowaniu obiektów do drugiego bufora można je szybko zamienić. Wówczas świeżo załadowany bufor staje się aktywnie wyświetlanym buforem, a bufor, który był aktywny, zostaje udostępniony do odświeżenia danych wyprowadzanych na ekran.

Proces wyświetlania rozpoczyna się od wywołania metody `Clear` urządzenia Direct3D. Powoduje ona wymazanie istniejącej w buforze informacji. Gdybyśmy ominęli ten krok, stara informacja pozostałaby na ekranie. Jedną z użytecznych funkcji tej metody jest możliwość ustawienia koloru tła. Ja zdecydowałem się na kolor ciemnoszary (&H404040), jednak w każdej chwili można zmienić go na inny.



Kolory tła

Kolory są opisywane jako wartości RGB, gdzie R to czerwony, G — zielony, a B — niebieski. Każda z wartości może być z zakresu od 0 do 255 (wyrażone dziesiętnie) lub od 00 do FF (wyrażone szesnastkowo). Dobrym sposobem na ustalenie koloru jest wybranie go w Photoshopie przy użyciu narzędzia *color swatches* albo *color sliders*. Potem wystarczy zamienić wartości z każdego pola koloru na szesnastkowe i użyć ich jako parametr metody `Clear`.

Kolejnym krokiem w procesie wyświetlania grafiki jest wywołanie metody `BeginScene`. Służy ona do oznaczenia początku lokalnego przetwarzania. Każde wywołanie `BeginScene` wymaga odpowiadającego mu wywołania `EndScene`, w innym wypadku program wygeneruje błąd.

Po rozpoczęciu ujęcia można dodać obiekty 3D do bufora. W tym przypadku polecenie `For Each` wykonuje iterację w ramach kolekcji wartości `DXObjects` przynależnych kolekcji `DXObs`.

Aby wyświetlić obiekt, należy odpowiednio ustawić początek układu współrzędnych świata gry. Jest to konieczne, jako że każdy z obiektów został stworzony względem własnego początku układu współrzędnych przestrzennych (0, 0, 0). Przesunięcie to umożliwi prawidłowe wyświetlenie obiektu. Proces ten lepiej wyobrazić sobie jako poruszanie światem gry w celu dopasowania go do obiektu, niż poruszanie obiektem, aby dopasować go do świata.

Przesuwanie początku układu współrzędnych świata polega na obliczeniu jego nowych współrzędnych tą samą techniką, jaka została użyta do stworzenia macierzy widoku. A więc najpierw tworzy się macierz jednostkową i używa się procedury `D3DXMatrixRotation` w celu obrócenia początku układu do wartości `Rotation` z pliku inicjalizacyjnego. Następnie oblicza się przesunięcie względem rzeczywistego początku układu, korzystając ze stałych opisujących lokalizację obiektu, zawartych w jego pliku inicjalizacyjnym. Na koniec, za pomocą metody `SetTransform`, przywraca pierwotne współrzędne świata gry.

`DXObject` współpracuje z trzema różnymi typami obiektów 3D. Pierwszy to siatka bez tekstury bądź informacji o materiale (`SimpleMesh`). Przykładem takiego obiektu jest czajniczek z rozdziału 2. Pozostałe dwa typy (`TexturedMesh` i `SelectedMesh`) umożliwiają wyświetlenie obiektów 3D wraz z teksturą. Jedyna różnica polega na tym, że obiekt `SelectedMesh` będzie podświetlony na czerwono, aby umożliwić graczowi identyfikację obiektu w centrum handlowym, który jest aktualnie wybrany.

Obiekt z teksturą zwykle składa się z paru części, które muszą być oddzielnie narysowane. Każda część, która pokryta jest inną teksturą lub kolorem, musi być potraktowana oddzielnie. Do wyświetlania kolejnych części zastosowana została pętla `For`. Wykonuje ona iteracje w ramach listy materiałów i za pomocą metod `SetMaterial` i `SetTexture` kolejno je uaktywnia, by następnie wywołać metodę `DrawSubset`, rysującą część obiektu, pokrytą daną teksturą.

Do narysowania obiektu wybranego przez gracza stosuje się ten sam proces, z tym że przed wywołaniem `SetMaterial` odrobinę zmienia się atrybuty tekstury. Poprzez zmianę informacji o kolorach we właściwościach `Ambient` i `diffuse` obiekt będzie podświetlony na czerwono. Jest to idealny sposób na graficzne podkreślenie wybranego obiektu.

Właściwości kolorów w `DirectX` zawierają zestaw typowych kolorów, czyli czerwony, niebieski i zielony, a także współczynnik przenikalności, zwany alfa. Tak samo, jak w przypadku kolorów, alfa jest wartością typu `single` z zakresu od 0 do 1. Zero oznacza, że materiał jest doskonale przezroczysty, a jeden, że jest doskonale nieprzezroczysty. Normalnie atrybuty: alfa, czerwony, zielony i niebieski będą ustawione na 1, co oznacza, że tekstura będzie namalowana na białej powierzchni. Ustawienie atrybutów kolorów niebieskiego i zielonego na 0,5 spowoduje wyświetlenie obiektu na czerwonej powierzchni. W ten sposób uzyskamy czerwoną poświatę dla wybranego obiektu.

Po wyświetleniu wszystkich obiektów 3D można jeszcze wyświetlić informacje o ilości wyświetlanych klatek na sekundę i aktualnym punkcie widzenia. Na koniec wywołana zostaje metoda `EndScene`, kończąca ujęcie.

Najczęstszą okolicznością występowania błędów jest wyświetlanie informacji na ekranie za pomocą metody `Present`. Mimo że problemy te występują z wielu przyczyn, zwykle lekarstwem okazuje się powtórna inicjalizacja urządzenia `Direct3D` i związanych z nim obiektów. Zajmuje się tym procedura `ResetDX`, wywołując po prostu metodę `InitDX` w celu odbudowania informacji związanych z `DirectX`.



With przychodzi z pomocą

Instrukcja `With` może być niezwykle przydatna podczas korzystania z obiektów zagnieżdżonych w innych obiektach. Ustanawia ona tymczasowy wskaźnik na określony obiekt. Wówczas każde odwołanie, które rozpoczyna się od kropki (`.`), będzie działało w obrębie tego obiektu. Spowoduje to nie tylko uproszczenie kodu, ale przede wszystkim szybsze działanie aplikacji, w związku z brakiem konieczności szukania obiektu za każdym razem, gdy występuje do niego odwołanie. Typowym zastosowaniem `With` jest korzystanie z głęboko zagnieżdżonych obiektów lub kolekcji obiektów. W przypadku zagnieżdżania samych instrukcji `With` aktywna będzie tylko najbardziej zagnieżdżona z instrukcji. Dopuszczenie więcej niż jednej aktywnej instrukcji `With` uniemożliwiłoby Visual Basicowi poprawne kojarzenie metod i właściwości z obiektami, do których przynależą.

Wyświetlanie tekstu na ekranie

Jedną z przydatnych funkcji DirectX jest wyświetlanie tekstu w obrębie ekranu DirectX3D w czasie wykonywania programu. W ten sposób można wyświetlać tak istotne informacje, jak ilość klatek na sekundę. DirectX ułatwia to zadanie, udostępniając metodę `DrawText`, jednak zanim będzie można z niej skorzystać, należy zainicjalizować obiekt, zawierający informacje o tekście.

Wyświetlanie tekstu umożliwi specjalny obiekt, zwany `D3DXFont`, który zawiera informacje o czcionce i jej atrybutach. Na poziomie modułu `DXGraphics` został zdefiniowany obiekt klasy `D3DXFont`, zwany `GameFont`. W wydruku 6.5 zawarty jest kod, inicjalizujący czcionkę używaną w grze. (Wywołanie tej procedury następuje z poziomu `InitDX`).

Wydruk 6.5. Funkcja `DXGraphics.InitDXFonts`

```
Private Function InitDXFonts() As Long

    Dim FntDesc As IFont
    Dim Fnt As StdFont

    Set Fnt = New StdFont
    Fnt.Name = "Arial"
    Fnt.Size = 8
    Fnt.Bold = True
    Set FntDesc = Fnt
    Set GameFont = d3dx.CreateFont(d3dDevice, FntDesc.hFont)

    DXDebugger.WriteErr "Inicjalizacja czcionki GameFont", Err

    InitDXFonts = Err.Number

End Function
```

W celu zainicjalizowania obiektu `D3DXFont` należy utworzyć obiekt `StdFont`, przechowujący wszelkie informacje związane z czcionką. W tym przypadku dla wyświetlanego tekstu wybrano pogrubioną czcionkę „Arial” o rozmiarze 8 punktów. Po zdefiniowaniu

obiektu `StdFont` zostaje on przyporządkowany obiektowi `IFont`, który z kolei zostaje przesłany metodzie `CreateFont`. Obiekt `StdFont` nie zawiera atrybutu `hFont`, który jest wymagany przez `CreateFont`, z kolei obiekt `FontDesc` nie może być poprawnie zainicjalizowany w Visual Basicu.

Procedura `DrawFrameRate` (zobacz wydruk 6.6) oblicza aktualną prędkość wyświetlania klatek i wyświetla ją na ekranie. Korzysta ona z procedury Win32 API `GetTickCount`, aby otrzymać wartość `Long`, zawierającą czas od włączenia komputera, wyrażony w milisekundach. Jeżeli minęła więcej niż sekunda (1000 milisekund) od ostatniego pobrania informacji, bieżąca prędkość zostaje zapisana do `LastTimeCheckFPS`, a bieżąca wartość `FramesDrawn` zapisana do zmiennej zwanej `FrameRate` i wyzerowana.

Wydruk 6.6. *Procedura `DXGraphics.DrawFrameRate`*

```
Private Sub DrawFrameRate()  
  
Dim TextRect As RECT  
  
If GetTickCount - LastTimeCheckFPS >= 1000 Then  
    LastTimeCheckFPS = GetTickCount  
    FrameRate = FramesDrawn  
    FramesDrawn = 0  
  
End If  
  
FramesDrawn = FramesDrawn + 1  
  
TextRect.Top = 1  
TextRect.Left = 1  
TextRect.Bottom = 16  
TextRect.Right = 200  
d3dx.DrawText GameFont, &HFFFFFF0000, "Bieżąca ilość kadrów na sekundę: " _  
    & FrameRate, TextRect, DT_TOP Or DT_LEFT  
  
End Sub
```

Po odświeżeniu informacji o szybkości wyświetlania obrazu wartość `FramesDrawn` zostaje zwiększona o jeden, zliczając w ten sposób kolejne wyświetlone klatki. Należy pamiętać, że opisywana procedura jest wywoływana tylko raz na każde wyświetlenie klatki.

Aby szybkość wyświetlania ukazała się na ekranie, należy zdefiniować prostokątne okno za pomocą współrzędnych ekranu, wyrażonych w pikselach, w miejscu, gdzie ma zostać wyświetlony tekst. W tym celu wykorzystano strukturę `RECT`, ustawiając jej właściwości `Top`, `Left`, `Bottom` i `Right` w taki sposób, aby tekst ukazywał się w lewym górnym rogu ekranu.

Potem wywołana zostaje metoda `DrawText` ze zdefiniowanym wcześniej obiektem `GameFont` i wartością kolor-przenikalność ustawioną na `&HFFFFFF0000`. Ta szesnastkowa wartość oznacza, że wyświetlane znaki będą nieprzezroczyste, koloru czerwonego. Pierwsze dwie cyfry heksadecymalne zawierają wartość współczynnika alfa, a następne sześć — wartości kolorów, kolejno po dwie na czerwony, zielony i niebieski.

Następnie określony zostaje łańcuch tekstowy do wyświetlenia, z obliczoną wcześniej wartością `FrameRate`, struktura `RECT` z informacją o pozycji i rozmiarze pola tekstowego, wreszcie informacje o pozycjonowaniu tekstu w obrębie pola. W tym przypadku tekst zostaje wyrównany do górnej i lewej krawędzi pola.



Przepenia się tylko co 49,7 dni

Jeżeli komputer jest cały czas włączony, należy liczyć się z tym, że wartość `TickCount` w końcu się przepeni i wyzeruje. Jako, że jest to wartość typu `Long`, powinno się to zdarzyć jedynie co 49,7 dni. Jest to mało prawdopodobne (Windows musiałby wytrzymać tyle czasu bez restartu), więc postanowiłem nie pisać procedury obsługującej to zdarzenie.

Wybieranie obiektów za pomocą klasy `DXGraphics`

Skoro zadałem sobie trud zaprogramowania wyświetlania obiektów wybranych, należałoby również napisać procedurę umożliwiającą graczowi wybór danego obiektu. Kod tej procedury składa się z dwóch części — zdarzenia `MouseDown` z obiektu `Form`, które wykrywa naciśnięcie przycisku myszki oraz metody `HitObject` z klasy `DXGraphics`.

Wykrywanie kliknięcia

Zdarzenie `MouseDown` powinno być dobrze znane większości programistom Visual Basic. Uruchamia się ono za każdym razem, gdy zostanie naciśnięty przycisk myszki, podczas gdy kursor znajduje się ponad którąś z kontrolki w formularzu. W wydruku 6.7 zawarta jest obsługa tego zdarzenia. Korzysta ono ze zmiennej na poziomie modułu, zwanej `SelectedObject`, która zawiera obiekt, wskazujący na aktualnie wybrany element na ekranie. Jeżeli żaden obiekt nie jest wybrany, `SelectedObject` ma wartość `Nothing`.

Wydruk 6.7. `Form1.Picture1_MouseDown`

```
Private Sub Picture1_MouseDown(Button As Integer, Shift As Integer, _  
    X As Single, Y As Single)  
  
    If Not SelectedObject Is Nothing Then  
        SelectedObject.mType = TexturedMesh  
  
    End If  
  
    Set SelectedObject = dx.HitObject(X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY)  
  
    If Not SelectedObject Is Nothing Then  
        SelectedObject.mType = SelectedMesh  
  
    End If  
  
End Sub
```

Procedura rozpoczyna od ustawienia wskaźnika aktualnie wybranego obiektu na Nothing. Następnie wywołuje metodę HitObject, uprzednio konwertując wartości X i Y z jednostek Twips na piksele. Jeżeli HitObject zwróci wskaźnik do istniejącego obiektu DXObject, ustawia właściwość mType na SelectedMesh.

Szukanie wybranego obiektu 3D

Procedura HitObject (patrz wydruk 6.8) za pomocą pętli For Each wykonuje iterację w ramach pełnej kolekcji obiektów. Dla każdego obiektu wywoływana jest procedura HitAnObject, która sprawdza, czy współrzędne kursora leżą w obrębie obiektu. Jeżeli tak, zwraca odnośnik do wybranego obiektu i kończy działanie. W innym przypadku procedura zwraca Nothing.

Wydruk 6.8. Funkcja DXGraphics.HitObject

```
Public Function HitObject(X As Single, Y As Single) As DXObject

    Dim o As DXObject

    For Each o In DXObs
        If HitAnObject(o, X, Y) And o.oType <> dxobjectmall Then
            Set HitObject = o
            Exit Function
        End If
    Next o

    Set Hit Object = Nothing

End Function
```

Należy zwrócić uwagę, że w przypadku obiektu Mall, czyli samego centrum handlowego, funkcja nie zwraca odnośnika. Jako że centrum jest ładowane jako pierwsze, nie byłoby możliwości wybrania któregoś ze sklepów, ponieważ sklepy znajdują się w obrębie centrum. W związku z tym kliknięcie na sklep równałoby się kliknięciu na centrum.

Sprawdzanie pojedynczego obiektu

Największym problemem przy sprawdzaniu, czy dany obiekt zawiera punkt określony współrzędnymi X i Y, jest jego konwersja na prostą w przestrzeni trójwymiarowej. Późniejsze ustalenie, czy prosta przecina obiekt, jest już kwestią relatywnie łatwą. Utrudnieniem pozostaje fakt, że współrzędne przechowywane przez siatkę danego obiektu wskazują zawsze, że obiekt jest w centrum świata 3D.

Tak więc pierwszym krokiem będzie przesunięcie początku układu współrzędnych świata na początek układu współrzędnych obiektu. W tym celu wykorzystany zostanie ten sam kod, który służył do kreślenia obiektów w metodzie Render.

Wydruk 6.9. *Funkcja DX.Graphics.HitAnObject*

```

Private Function HitAnObject(o As DXObject, X As Single, Y As Single) As Boolean

    Dim vIn As D3DVECTOR
    Dim vNear As D3DVECTOR
    Dim vFar As D3DVECTOR
    Dim vDir As D3DVECTOR
    Dim viewport As D3DVIEWPORT8
    Dim hit As Long
    Dim faceindex As Long
    Dim retu As Single
    Dim retv As Single
    Dim dist As Single

    Dim matTemp As D3DMATRIX
    Dim matNewWorld As D3DMATRIX
    Dim matTrans As D3DMATRIX

    D3DXMatrixIdentity matTemp
    D3DXMatrixRotationY matTemp, o.Rotation
    D3DXMatrixMultiply matTemp, matWorld, matTemp
    D3DXMatrixTranslation matTrans, -o.Location.X, -o.Location.Y, -o.Location.Z
    D3DXMatrixMultiply matNewWorld, matTemp, matTrans

    d3dDevice.GetViewport viewport

    vIn.X = X
    vIn.Y = Y

    vIn.Z = 1
    D3DXVec3Unproject vNear, vIn, viewport, matProj, matView, matNewWorld

    vIn.Z = -100
    D3DXVec3Unproject vFar, vIn, viewport, matProj, matView, matNewWorld

    D3DXVectSubstract vDir, vNear, vFar

    d3dx.Intersect DXmshs(o.MeshFilename).Mesh, vFar, vDir, hit, faceindex, retu, retv,
    ➔dist

    HitAnObject = (hit = 1)

End Function

```

Funkcja tworzy wektor ze zmiennych X i Y, przyjmując współrzędną Z jako 1. Potem procedura `D3DXVec3Unproject` konwertuje wektor do przestrzeni, w której obiekt jest centralny. W ten sposób tworzy się wektor `vNear`. Ten sam proces, powtórzony dla wartości $Z = -100$, definiuje wektor `vFar`.

Powstałe dwa wektory definiują prostą w przestrzeni trójwymiarowej. Odjęcie wektora `vNear` od wektora `vFar` daje wektor kierunkowy, który w połączeniu z wektorem `vFar` określa kierunek prostej.

Kolejnym krokiem jest wywołanie metody `Intersect`, by określić, czy prosta (zdefiniowana przez wektory `vNear` i `vFar`) krzyżuje się z daną siatką. Funkcja ta zwraca pięć różnych wartości. `Hit` przyjmuje wartość 1, jeżeli prosta przecina siatkę lub 0, jeżeli nie. `Faceindex` zawiera numer powierzchni obiektu, przeciętej prostą. `retu` i `retv` zawierają względne współrzędne punktu przecięcia w ramach tej powierzchni. (Ich wartości zawierają się w przedziale od zera do jedności). Wreszcie `dist` zawiera odległość od `vFar` do punktu przecięcia prostej z siatką.

Na koniec funkcja zamienia wartość `hit` na typ `Boolean` poprzez porównanie go do jedności. Wówczas, jeżeli `hit` ma wartość 1, funkcja zwróci `True`, a w innym wypadku zwróci `False`.

Uruchamianie programu

Czas na przetestowanie programu. Jego uruchomienie wymaga załadowania plików z katalogu `\VBGame\Rozdzial06` do Visual Basic'a i naciśnięcia klawisza `F5`. Dość szybko powinno otworzyć się okno formularza, a zaraz za nim centrum handlowe (zobacz rysunek 6.2). Rzeczywisty czas zależy w dużej mierze od szybkości procesora i karty graficznej. Im szybszy procesor, tym szybciej pojawi się obraz.

Rysunek 6.2.
Wygląd centrum handlowego w wersji z rozdziału 6

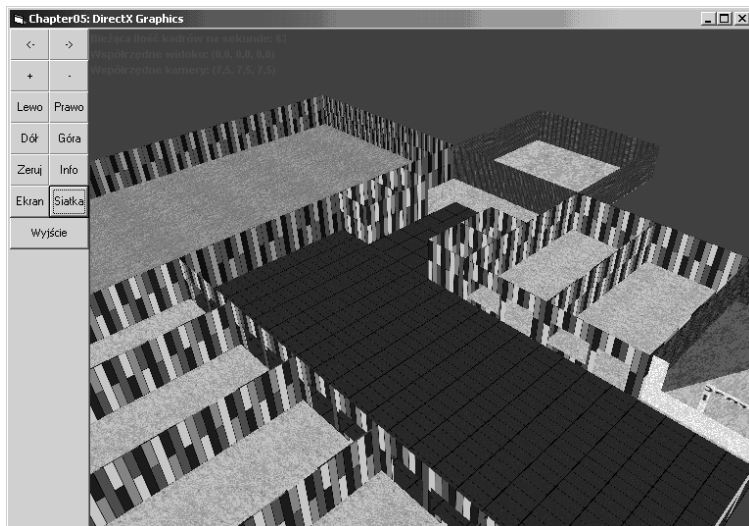


Program zawiera kilka przycisków z boku ekranu DirectX. Strzałki na górze umożliwiają obracanie centrum handlowym względem osi Y skokami co 45 stopni. Poniżej znajdują się przyciski `+` i `-`, pozwalające przybliżyć i oddalać widok oraz przyciski `Prawo`, `Lewo`, `Góra`, `Dół`, służące do przewijania widoku w określonym kierunku.

Przycisk `Zeruj` przywraca początkowy widok, jako że łatwo jest podczas przewijania i przybliżania stracić centrum handlowe z oczu. `Info` włącza i wyłącza wyświetlanie w lewym górnym rogu ekranu informacji o ilości klatek i punkcie widzenia.

Przycisk *Ekran* włącza tryb pełnoekranowy. Aby powrócić do wyświetlania w oknie, wystarczy nacisnąć *Esc*. *Siatka* służy do wybierania siatek. Po pierwszym naciśnięciu wybrany zostanie market. Dalsze naciskanie spowoduje wybieranie kolejnych obiektów z kolekcji `DXObject`s aż do osiągnięcia końca kolekcji i wybrania całego centrum handlowego. Przycisk *Wyjście* pozwala na powrót do Visual Basic. Kliknięcie myszką ponad któryś z obiektów spowoduje jego podświetlenie (patrz rysunek 6.3).

Rysunek 6.3.
Widok centrum handlowego po wybraniu marketu



Podsumowanie

Rozdział ten zawierał dużo matematyki. Na szczęście dokładne rozumienie wszystkich zawartych tu obliczeń nie jest konieczne — zawsze można po prostu skorzystać z istniejącego kodu, zostawiając wyższą matematykę profesjonalnym twórcom gier komputerowych.

Jednym z minusów procesu tworzenia grafiki, który został tu opisany, jest jego relatywnie mała wydajność, związana z dużą ilością pojedynczych siatek do wyświetlenia. Lepszą techniką byłoby skomasowanie podstawowych informacji o siatkach w kilku dużych obiektach, a następnie ich wyświetlenie. Jednak ilość kodu i skomplikowanych obliczeń, wymaganych do zastosowania tej techniki, mogłaby uczynić ten rozdział zbyt trudnym do zrozumienia.