Podstawowy projekt OpenGL utworzony przy użyciu środowisk programistycznych firmy Microsoft.

Autor: Radosław Płoszajczak

Spis treści

I. Wymagania i utworzenie projektu w Visual Studio 2005

Narzędzia Microsoft wymagane do utworzenia aplikacji OpenGL dla Windows:

- Microsoft Visual Studio,
- Microsoft Visual C++ lub
- Microsoft Visual C++ Express Edition + Microsoft Windows SDK (na dzień dzisiejszy najnowsza wersja to Windows® Server 2003 R2 Platform SDK ok.400MB)

Aby utworzyć projekt w Visual Studio 2005 należy z menu *File* wybrać *New→Project*, z listy *Project Type* wskazać *Visual C++ → Win32* i następnie z listy *Template* wybrać *Win32 Project*. Po wprowadzeniu nazwy i zatwierdzeniu typu projektu pojawi się okno kreatora. W zakładce *Application Settings* w polu *Application Type* należy wybrać *Windows Application* i w *Additional Options* zaznaczyć opcję *Empty Project*. W ustawieniach projektu *Project → Properties* z drzewa po lewej stronie wybieramy *Configuration Properties → Linker → Input* i w polu *Additional Dependiences* należy dopisać dodatkowe biblioteki:
`opengl32.lib glu32.lib`.

Domyślnie w Visual Studio 2005 programy kompilowane są dla kodowania Uncicode w Windows dlatego warto zmienić te ustawienie ponieważ biblioteka OpenGL w większości używa kodowania 1-bajtowego. Aby ustawić powyższe kodowanie należy w oknie ustawień projektu wybrać po lewej stronie *Configuration Properties → General* zmienić opcję *Character set* z *Use Unicode Character Set* na *Use Multibyte Character Set*.

W tak utworzonym projekcie należy kliknąć prawym przyciskiem na folderze *Source files* i z menu podręcznego wybrać *Add → New item...* i dodać do projektu nowy plik źródłowy *C++File (.cpp)*.

II. Absolutne minimum

Schemat postępowania:

1. Dodanie plików nagłówkowych

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
```

2. Stałe i zmienne globalne.

```
HGLRC hRC = NULL; //uchwyt kontekstu renderingu
HDC uchwytDC = NULL; //uchwyt prywatnego kontekstu urządzenia GDI
GLint ClientWidth; //szerokość obszaru roboczego okna
GLint ClientHeight; //wysokość obszaru roboczego okna
static LPCTSTR lpszAppName = "GLTemplate"; //nazwa okna i klasy okna
```

   Powyższe zmienne globalne zostaną wykorzystane w dalszej części tego tutorialu. Nazwę typu `LPCTSTR` można rozwinąć jako „long pointer to `TCHAR` string” czyli standardowy wskaźnik na tablicę znaków 1- lub 2-bajtowych (w zależności od ustawień środowiska) zakończonych `/0`.

3. Utworzenie funkcji `WinMain()`, która jest odpowiednikiem funkcji `main()` w aplikacjach konsolowych. Jej sygnatura jest następująca:

```
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine, int nCmdShow)
```

   Znaczenie poszczególnych parametrów:

   - `HINSTANCE hInstance` – identyfikator danej instancji aplikacji
   - `HINSTANCE hPrevInstance` – nie używane w aplikacjach Win32

- `LPSTR lpCmdLine` – wskaźnik na listę parametrów
- `int nCmdShow` – wartość określająca czy aplikacja ma zostać uruchomiona zminimalizowana, zmaksymilizowana itp.

4. Schemat funkcji `WinMain()`:
   1. Deklaracja struktur klasy okna (`WNDCLASS wc`) i komunikatu dla okna (`MSG msg`) oraz zmiennej przechowującej uchwyt okna (`HWND hWnd`).

   2. Wypełnienie struktyry klasy okna i jej zarejestrowanie dla danej aplikacji.

```
wc.style         = CS_HREDRAW | CS_VREDRAW | CS_OWNDC; //styl okna
wc.lpfnWndProc   = (WNDPROC)WndProc; //procedura okna
wc.cbClsExtra    = 0; //dodatkowe bajty zarezerwowane za klasą okna
wc.cbWndExtra    = 0; //dodatkowe bajty zarezerwowane za instancją okna
wc.hInstance     = hInstance; //instancja aplikacji
wc.hIcon         = NULL; //uchwyt ikony
wc.hCursor       = LoadCursor(NULL, IDC_ARROW); //uchwyt kursora
wc.hbrBackground = NULL; //uchwyt pędzla tła
wc.lpszMenuName  = NULL; //nazwa menu
wc.lpszClassName = lpszAppName; //nazwa klasy okna

//Rejestracja klasy okna
if(RegisterClass(&wc) == 0)
     return FALSE;
```

      - Styl klasy okna określa pewne jego specyficzne właściwości. Są to wartości zdefiniowane w stałych zaczynających się przedrostkiem `CS_` i mogą one być ze sobą łączone bitowym operatorem logicznym “lub” – `|`. W tym przypadku `CS_HREDRAW` i `CS_VREDRAW` wymuszają przerysowanie okna gdy zostanie zmieniony jego obszar roboczy na skutek zmiany rozmiaru okna lub pozycji okna (np. odsłonięcie jego części) odpowienio dla szerokości i wysokości. Ważne jest również aby okno posiadało własny unikalny kontekst urządzenia, którego nie utraci podczas całego czasu istnienia okna tej klasy (`CS_OWNDC`).
      - Pole procedury okna przechowuje wskaźnik do funkcji procedury okna, która zostanie omówiona w dalszej części tutorialu.
      - Dodatkowe bajty zarezerwowane za klasą i instanją okna wykożystywane są w innych bardziej zaawansowanych technikach programowania w WinAPI i w tym prostym przykładzie nie są potrzebne.
      - Instancja aplikacji musi być taka sama jak, tej która posiada procedurę okna. Jej wartość pobierana jest z paremetru funkcji `WinMain()`.
      - Uchwyt ikony pozwala na załadowanie własnej ikony dla okna. Wartośc `NULL` powoduje użycie domyślnej ikony Windows.
      - Uchwyt kursora załadowany jest poprzes funkcję `LoadCursor()`, która w tym przypadku wybiera domyślną strzałkę.
      - Uchwyt pędzla tła ustawiany jest na `NULL` co wymusza na aplikacji samodzielne odrysowanie tła. Wybranie jakiegokolwiek pędzla mogłoby spowodować migotanie przy odrysowywaniu sceny OpenGL.
      - Nazwa głównego menu jest wskaźnikiem na zakończony zerem ciąg znaków identyfikujący zasób aplikacji (resurce) zawierający definicję menu. W tym przypadku okno nie ma menu więc wartośc ta jest ustawiana na `NULL`.
      - Nazwa klasy okna jest wskaźnikiem na zakończony zerem łańcuch znaków i może on być taki sam jak nazwa okna.
      - Po wypełnieniu wszystkich pól struktury wc funkcja `RegisterClass()` rejestruje klasę okna i w przypadku niepowodzenia zwraca `0`.

3. Utworzenie okna przy pomocy zarejestrowanej jego klasy.
   Funkcja `CreateWindow()` tworzy okno i zwraca jego uchwyt:

```
hWnd = CreateWindow(
     lpszAppName, //nazwa klasy okna
     lpszAppName, //nazwa okna
     WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN | WS_CLIPSIBLINGS, //styl okna
     100, 100, //położenie okna (x,y)
     250, 250, //rozmiar okna (szerokość, wysokość)
     NULL, //uchwyt okna nadrzędnego (parent)
     NULL, //uchwyt menu
     hInstance, //uchwyt instancji aplikacji
     NULL); //parametr tworzenia okna

if(hWnd == NULL)
     return FALSE;
```

   - Nazwa klasy okna musi być taka sama jak ta zdefiniowana w strukturze zarejestrowanej klasy okna.
   - Nazwa okna jest wskaźnikiem na zakończony zerem łańcuch znaków i w przypadku gdy okno posiada pasek tutułowy jest na nim wyświetlana.
   - Styl okna podobnie jak styl klasy okna określa jego wygląd i pewne właściwości. Podobnie jak wcześniej definiuje się je łącząc wartości stałych lecz tym razem rozpoczynających się przedrostkiem `WS_`. Można tutaj ustalić czy okno ma posiadać pasek tytułowy, paski przewijania, przyciski minimalizacji, maksymalizacji, zamknięcia itp. Stała `WS_OVERLAPPEDWINDOW` określa połączenie kilku innych styli definiujących standardowe okno zawierające pasek tutułu, obramowanie, menu systemowe, przyciski maksymalizacji i minimalizacji itd. Najprostrze okno OpenGL musi posiadać dwa dodatkowe style aby aplikacja zawsze prawidłowo działała: `WS_CLIPCHILDREN` (wyklucza obszar zasłaniany przez okno podrzędne np. okno narzędziowe podczas odrysowywania okna głównego) oraz `WS_CLIPSIBLINGS`, które ma podobne znaczenie ale w odniesieniu do innych okien podrzędnych (“rodzeństwa”).
   - Położenie okna liczone jest w pikselach i określa współrzędną lewego, górnego rogu okna liczoną od lewego, górnego rogu ekranu.
   - Rozmiar okna podawany jest również w pikselach.
   - W przypadku gdyby tworzone okno było oknem podrzędnym należy określić uchwyt okna nadrzędnego. W tym przypadku dane okno jest oknem głównym i parametr ten przyjmuje wartość `NULL`.
   - Ponieważ okno nie ma posiadać menu uchwyt menu przyjmuje wartość `NULL`.
   - Uchwyt instancji aplikacji wskazuje moduł programu, do którego ma należeć okno. W powyższym przykładzie wartosc ta jest pobierana z parametru funkcji `WinMain()`.
   - Parametr tworzenia okna jest wskaźnikiem na parametr przekazywany procedurze okna `WndProc()` przez `lParam` podczas zajścia zdarzenia (`WM_CREATE`).
   - Jeżeli zwrócony przez funkcję uchwyt okna `hWnd` ma wartość `NULL` oznacza to błąd i okno nie zostanie utworzone.

4. Pokazanie i zaktualizowanie okna.

```
ShowWindow(hWnd,SW_SHOW);
UpdateWindow(hWnd);
```

   Obie funkcje przyjmują jako parametr uchwyt okna hWnd jednoznacznie identyfikujący okno. Stała `SW_SHOW` aktywuje i pokazuje okno w jego dotychczasowym rozmiarze. Możliwe jest też zastosowanie innych stałych z przedrostkiem `SW_` jak np. `SW_SHOWMAXIMIZED`, która to wyświetla zmaksymalizowane okno.

5. Pętla komunikatów.

```
while(GetMessage(&msg, NULL, 0, 0))
{
     TranslateMessage(&msg);
     DispatchMessage(&msg);
}
```

Jest to główne miejsce, które odbiera i przesyła komunikaty dla okna pochodzące od systemu Windows i użytkownika takich jak zmiana rozmiaru okna, potrzeba jego odrysowania, kliknięcia myszką, naciśniecia klawiszy itd. Komunikaty te są obsługiwane przez procedurę okna `WndProc()`, która zostanie opisana w dalszej części tego tutoriala.

- Funkcja `GetMessage()` pobiera kolejny komunikat z kolejki komunikatów przesłanych przez system operacyjny. W przypadku gdy funkcja odbierze komunikat `WM_QUIT` oznaczający zakończenie aplikacji zwraca wartość `FALSE` i przerywa pętle. Przyjmuje ona 4 parametry:
  - wskaźnik do struktury, w której zapisany zostanie komunikat
  - uchwyt okna, dla którego komunikat ma zostać pobrany – w tym przypadku wartość `NULL` oznacza, że komunikat ma zostać pobrany dla dowolnego okna aplikacji
  - ostatnie dwa parametry `wMsgFilterMin` i `wMsgFilterMax` określają zakres filtru wiadomości w kolejce i w przypadku gdy oba maja wartość `0` żaden filtr nie jest stosowany.
- Funkcja `TranslateMessage()` tłumaczy wirtualne kody klawiszy uwzględniając przy tym m.in. stan klawisza shift oraz ustawienia regionalne systemu na odpowiednie kody znakowe. Następnie wstawia w kolejkę komunikat `WM_CHAR` z odpowiednim znakiem. Ma to znaczenie gdy w aplikacji ma np. następować wprowadzanie tekstu z klawiatury.
- `DispatchMessage()` wysyła podany w parametrze komunikat do odpowiedniej procedury okna.

6. Zwrócenie wartości podczas zakończenia programu.

```
return msg.wParam;
```

Na konieć funkcji `WinMain()` warto zwrócić do systemu kod zakończenia aplikacji zawarty w parametrze komunikatu `WM_QUIT`.

5. Utworzenie procedury okna mającej za zadanie obsługę komunikatów wysyłanych przez system operacyjny do okna aplikacji. Jej deklaracja jest następująca:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
   LPARAM lParam);
```

Funkcja ta zwraca wartość typu `LRESULT` co odpowiada standardowemu typowi `long` i zazwyczaj wynosi `0`. Przyjmuje ona 4 parametry: uchwyt okna `hWnd`, którego aktualnie przetwarzany komunikat dotyczy, typ komunikatu `message` oraz dwa parametry tego komunikatu `wParam` i `lParam`.

6. Schemat podstawowej procedury okna OpenGL.

```
switch (message)
{
   case WM_CREATE: //Utworzenie okna
         uchwytDC = GetDC(hWnd);
         GL_UstalFormatPikseli(uchwytDC);
         //Utworzenie kontekstu renderowania i uczynienie go aktywnym
         hRC = wglCreateContext(uchwytDC);
         wglMakeCurrent(uchwytDC, hRC);
         GL_UstawienieSceny();
         break;
```

```
    case WM_DESTROY: //Usuwanie okna, czyszczenie
          //Usunięcie kontekstu renderowania
          wglMakeCurrent(uchwytDC,NULL);
          wglDeleteContext(hRC);
          //Wysłanie komunikatu zamknięcia aplikacji WM_QUIT
          PostQuitMessage(0);
          break;

    case WM_SIZE: //Zmiana rozmiaru okna
          RECT rect;
          GetClientRect(hWnd, &rect);
          ClientHeight = rect.bottom - rect.top;
          ClientWidth = rect.right - rect.left;
          GL_UstawienieSceny();
          break;

    case WM_PAINT: //Okno wymaga odrysowania
          GL_RysujScene();
          ValidateRect(hWnd,NULL);
          break;

    default:   //przetworzenie automatyczne pozostałych komunikatów
          return (DefWindowProc(hWnd, message, wParam, lParam));
}
    return (0L);
```

W najprostrzej postaci procedura okna posiada jedynie instrukcję `switch`, w której w zależności od rodzaju komunikatu wykonywane są odpowiednie czynności.

- Komunikat `WM_CREATE` przekazywany jest po utworzeniu okna i może on odpowiadać konstruktorowi klasy `GLForm`. Po jego odebraniu tworzony jest uchwyt urządzenia `uchwytDC` przy pomocy funkcji `GetDC()`, która jako parametr przyjmuje uchwyt okna dla którego jest on pobierany. Następnie na jego podstawie pobierany jest odpowiedni format pikseli przy pomocy funkcji `GL_UstalFormatPikseli()`, która ma taką sama postać jak ta w klasie `GLForm`. Kolejną czynnością jest utworznie kontekstu renderowania przy pomocy funkcji `wglCreateContext()` i uczynienie go aktywnym poprzez wywołanie funkcji `wglMakeCurrent()`. Gdy okno posiada już możliwość rysowania przy pomocy OpenGL można zainicjować ustawienia sceny funkcją `GL_UstawieniaSceny()` skopiowanej również z klasy GLForm.
- W momencie zamknięcia okna do procedury okna wysyłany jest komunikat `WM_DESTROY`. Analogicznie jak w poprzednim przypadku jest on odpowiednikiem destruktora klasy `GLForm` i przy jego obsłudze należy nieco "posprzątać". Najpierw należy sprawić aby kontekst renderowania nie był już aktywny podając w parametrze funkcji `wglMakeCurrent()` uchwyt kontekstu renderowania jako `NULL`. Następnie należy usunąć kontekst renderowania przy pomocy funkcji `wglDeleteContext()`. Kolejną istotną sprawą jest sprawienie aby zamknięcie okna spowodowało zamknięcie całej aplikacji czyli przerwanie pętli komunikatów. Służy temu funkcja `PostQuitMessage()` z parametrem o wartości `0` co oznacza brak błędów.
- W przypadku gdy istnieje możliwość zmiany rozmiaru okna trzeba obsłużyć jego wpływ na wygląd rysowanej sceny w komunikacie `WM_SIZE`. W tym celu należy obliczyć rozmiar obszaru roboczego okna (bez obramowania i pasku tytułu). Aby tego dokonać należy wywołać funkcję `GetClientRect()`, która pobiera współrzędne (lewa, prawa, góra, dół) prostąkąta roboczego okna. Następnie obliczając odpowienie różnice można otrzymać wysokość `ClientHeight` oraz szerokość `ClientWidth` obszaru roboczego. Wartości te bedą wykorzystywane przez funkcję `GL_UstawieniaSceny()` do określenia rozmiaru widoku.
- Ostatni komunikat `WM_PAINT` wysyłany jest za każdym razem gdy okno wymaga

odrysowania. Obsługa tego komunikatu polega na wywołaniu funkcji `GL_RysujScene()`. Wywołanie komunikatu `WM_PAINT` jest powiązane z unieważnieniem obszaru okna. Aby poprawnie odrysować okno należy na końcu zatwierdzić obszar roboczy okna. Służy temu funkcja `ValidateRect()`, która jako parametry przyjmuje uchwyt okna oraz wskaźnik do struktury `RECT` przechowującej współrzędne zatwierdzonego obszaru roboczego okna. Wartość `NULL` wymusza zatwierdzenie całej powierzchni roboczej.

7. Skopiowanie potrzebnych funkcji z klasy `GLForm`.
   Najpierw przed funkcją WinMain() należy zadeklarować odpowiednie funkcje:

```
bool GL_UstalFormatPikseli(HDC uchwytDC);
void GL_UstawienieSceny(void);
void __fastcall GL_RysujScene(void);
void __fastcall RysujScene(void);
```

   Następnie w dalszej części pliku źródłowego należy skopiować ciała tych funkcji. W przypadku funkcji `RysujScene()` należy w jej ciele zapisać dowolny kod rysujący OpenGL.
   Aby kompilacja aplikacji była możliwa należy wprowadzić kilka zmian w powyższych funkcjach. Dla uproszczenia tego opisu aplikacja zostanie również pozbawiona pewnych możliwości.
   - W funkcji `GL_UstawienieSceny()` należy wykomentować wywołanie fuknkcji `GL_Oswietlenie()`.
   - W funkcji `GL_RysujScene()` należy zmienić wywołanie funkcji `glClearColor()` tak aby nie używała zmiennej `kolorTla`:

```
glClearColor(0.0f, 0.0f, 0.0f, 0.0);
```

   - Kolejnym krokiem aby aplikacja mogła się skompilować jest zdefiniowanie i zainicjowanie na początku pliku zmiennych globalnych kamery wykorzystywanych w funkcji :

```
GLfloat kameraX = 0.0f;
GLfloat kameraY = 0.0f;
GLfloat tmpKameraX = 0.0f;
GLfloat tmpKameraY = 0.0f;
GLfloat kameraPhi = 0.0f;
GLfloat kameraTheta = 0.0f;
GLfloat tmpKameraPhi = 0.0f;
GLfloat tmpKameraTheta = 0.0f;
```

III. Wzbogacenie aplikacji o interakcję z użytkownikiem i animację.

1. Dodanie obrotu i przesunięcia kamery kontrolowanej przez mysz.
   Na początku procedury okna `WndProc()` należy zadeklarować statyczne zmienne, które będą przechowywać pozycję myszy w momencie kliknięcia prawego lub lewego przycisku, a później ostatnią pozycję myszy:

```
static int x0 = 0;
static int y0 = 0;
```

   Następnie należy do instrukcji switch wybierającej aktualnie przetwarzany komunikat dodać reakcję na naciśniecie lewego lub prawego przycisku myszy. Poniższe instrukcje zapiszą początkową pozycję myszy:

```
case WM_RBUTTONDOWN:
case WM_LBUTTONDOWN:
   x0 = LOWORD(lParam);
   y0 = HIWORD(lParam);
   break;
```

Makra `LOWORD()` i `HIWORD()` wyodrębniają mniej i bardziej znaczące bajty parametru `lParam` odpowiadające współrzędnych myszy x i y.
Kolejnym krokiem jest zaimplementowanie reakcji okna na ruch myszy co wymaga dodanie kolejnego zdarzenia do instrukcji `switch`:

```
case WM_MOUSEMOVE:
   {
         int x = LOWORD(lParam);
         int y = HIWORD(lParam);
         int dx = x - x0;
         int dy = y - y0;

         if (wParam == MK_RBUTTON)
         {
               const GLfloat czulosc = 50.0f;
               kameraX += (GLfloat)dx / czulosc;
               kameraY -= (GLfloat)dy / czulosc;
         }

         if (wParam == MK_LBUTTON)
         {
               const GLfloat czulosc = 5.0f;
               kameraPhi += (GLfloat)dx / czulosc;
               kameraTheta += (GLfloat)dy / czulosc;
         }

         x0 = x;
         y0 = y;
         GL_RysujScene();
   }
   break;
```

Na początku podobnie jak w przypadku naciśniecia przycisku myszy pobierana jest aktualna pozycja myszy. Następnie do zmiennych `dx` i `dy` zapisane są przesunięcia kursora myszy wzdłuż osi x i y okna. Parametr `wParam` tego zdarzenia zawiera informacje czy i jaki przycisk myszy był naciśnięty w czasie kiedy mysz została poruszona. W momencie gdy był naciśnięty prawy przycisk `MK_RBUTTON` następuje przesunięcie kamery proporcjonalnie do przesunięcia kursora. Lewy przycisk `MK_LBUTTON` powoduje obrót kamery. Na końcu aktualne wpółrzędne kursora są zapisywane, a scena odrysowywana.

2. Dodanie animacji.
   Animowanie kamery można zaimplementować z wykorzystaniem timera, który co jakiś określony odcinek czasu będzie ją obracał. Aby utworzyć timer wywoływany 50 razy na sekundę w procedurze okna `WndProc()` w komunikacie `WM_CREATE` należy dodać następującą instrukcję:

```
SetTimer(hWnd, 1, 20, NULL);
```

Pierwszy parametr jest uchwytem okna, z którym timer ma być powiązany. Drugi parametr jest identyfikatorem timera wykorzystywanym w przypadku gdy okno posiada wiele timerów. Trzeci parametr określa odstęp czasu wyrażony w milisekundach pomiedzy poszczególnymi wywołaniami komunikatu `WM_TIMER`. Ostatni parametr może posłużyć do przekazania wskaźnika do procedury timera, która zostanie wywołana za każdym razem gdy odliczanie się zakończy. Jest to alternatywa do zdarzenia `WM_TIMER` dlatego w tym przykładzie należy tu podać wartość `NULL`.
W momencie zamknięcia okna przez użytkownika należy usunąć czasomierz dodając przed instrukcją `PostQuitMessage(0)` następującą linię:

```
KillTimer(hWnd, 1);
```

Funkcja `KillTimer()` przyjmuje jako parametry uchwyt okna oraz identyfikator timera.
Na koniec należy napisać kod wywoływany podczas zakończenia odliczania. W tym celu w funkcji `WndProc()` należy dodać obsługę kolejnego zdarzenia dopisując do instrukcji `switch` kolejny przypadek:

```
case WM_TIMER:
   kameraPhi += 0.5f;
   GL_RysujScene();
   break;
```

Komunikat `WM_TIMER` przesyłany jest wraz ze identyfikatorem czasomierza zawartym w parametrze `wParam`, który jest niezbędny podczas obsługi wielu timerów. Parametr `lParam` zawiera wskaźnik do procedury czasomierza zdefiniowany w wywołaniu funkcji `SetTimer()`. Powyższy kod sprawia, że co 20ms kamera obraca się o 0,5°.