INFORMATYKA II

GRAFIKA KOMPUTEROWA

CEL LABORATORIUM

Celem zajęć jest zapoznanie uczestniczki/uczestnika z podstawami trójwymiarowej grafiki komputerowej. Przedstawiona zostanie przykładowa aplikacja stworzona za pomocą API OpenGL, do której dodane zostanie generowanie złożonych obiektów graficznych i oświetlenie.

CZYM JEST GRAFIKA KOMPUTEROWA?

Pod nazwą „grafika komputerowa" kryje się dyscyplina informatyki, która zajmuje się tworzeniem i przetwarzaniem cyfrowych treści wizualnych. Z efektami prac grafików komputerowych mamy do czynienia na co dzień, w postaci tapety na pulpicie naszego telefonu czy komputera, a także interfejsu użytkownika dowolnej aplikacji, systemu czy strony internetowej.

Rzadziej mamy do czynienia z grafiką trójwymiarową. Głównie w programach symulujących pewne obiekty lub zdarzenia albo w grach komputerowych. Grafikę trójwymiarową wyróżnia to, że przedstawia trójwymiarowe środowisko, oprócz wysokości i szerokości, oferujące również głębię. Istnieje szereg bibliotek i rozwiązań, które ułatwiają tworzenie takiej grafiki.

Wśród nich znajduje się opracowany przez Microsoft interfejs programistyczny DirectX, nowoczesny i niskopoziomowy Vulkan oraz OpenGL. Ten ostatni, ze względu na ogólnodostępność i prostotę, zostanie użyty do przedstawienia tematu.

ALGORYTM DZIAŁANIA APLIKACJI

Aplikacje korzystające z interfejsu programistycznego OpenGL, muszą być skonstruowane według konkretnego schematu. Uproszczoną wersję schematu działania takich aplikacji, można rozpisać za pomocą następującego algorytmu. Poszczególne punkty staną się jaśniejsze, wraz z postępami w realizacji instrukcji:

1. Ustaw opcje wyświetlania
2. Utwórz okno aplikacji
3. Odczytaj sterowanie
4. Dokonaj transformacji przestrzeni
5. Nanieś obiekty do bufora roboczego
6. Zamień bufory miejscami
7. Wyświetl zawartość bufora na ekranie
8. Jeżeli nie zakończono, wróć do punktu 3

OPENGL I GLUT – HISTORIA, NAZEWNICTWO, FUNKCJE

W celu uproszczenia pewnych rzeczy, korzystać będziemy z dodatkowej biblioteki zwanej *GLUT* (od ang. „*graphics library utility toolkit*", czyli „zestaw pomocnych narzędzi dla biblioteki graficznej"). OpenGL i GLUT nie korzystają z klas, ale za to udostępniają funkcje. Wszystkie funkcje biblioteki OpenGL, mają nazwy zaczynające się od „gl". Po tym prefiksie następuje krótka nazwa informująca czym funkcja się zajmuje, a następnie liczba i litera, które informują jakiego typu i ile argumentów funkcja przyjmuje. Dla przykładu, żeby ustalić kolor obiekty, wywołamy funkcję „`glColor3f`", która przyjmuje trzy argumenty typu `float`.

Dla odróżnienia, wszystkie funkcje biblioteki GLUT, zaczynają się od prefiksu „glut". Listę funkcji OpenGL i biblioteki GLUT, z których będziemy korzystać w trakcie zajęć, znajdziesz w poniższej tabeli. Lista jest posortowana alfabetycznie.

Funkcja	**Opis**
`glBegin`	Rozpoczyna fragment kodu definiujący wielokąt
`glClear`	Czyści bufor przekazany jako argument
`glClearColor`	Wypełnia zawartość bufora pikselami o danym kolorze
`glColor3f`	Ustala kolor kolejnego wielokąta
`glEnable`	Włącza żądaną opcję
`glEnd`	Kończy fragment kodu definiujący wielokąt
`glLightfv`	Nadaje wartość wybranej cechy, wybranego źródła światła
`glLoadIdentity`	Nadpisuje macierz transformacji macierzą jednostkową
`glNormal3f`	Definiuje wektor normalny do powierzchni
`glRotatef`	Dodaje do aktualnej macierzy transformacji obrót
`glShadeModel`	Służy do wyboru techniki cieniowania
`glTranslatef`	Dodaje do aktualnej macierzy transformacji przesunięcie
`gluPerspective`	Tworzy nową macierz perspektywy
`glutCreateWindow`	Tworzy nowe okno
`glutDisplayFunc`	Pozwala ustalić funkcję, która będzie wywoływana podczas każdego odświeżenie ekranu
`glutInit`	Inicjalizuje działanie biblioteki GLUT
`glutInitDisplayMode`	Pozwala wybrać i zainicjalizować tryb wyświetlania
`glutInitWindowSize`	Ustala wysokość i szerokość tworzonego okna
`glutKeyboardFunc`	Pozwala ustalić funkcję, która będzie wywoływana podczas każdego wciśnięcia klawisza
`glutMainLoop`	Rozpoczyna główną pętlę przetwarzania komunikatów
`glutPostRedisplay`	Wymusza zastosowanie zmian na buforze
`glutReshapeFunc`	Pozwala ustalić funkcję, która będzie wywoływana podczas każdego przypadku, gdy rozmiar okna ulegnie zmianie
`glutSwapBuffers`	Zamienia miejscami bufor ekranu i bufor roboczy
`glVertex3f`	Definiuje nowy punkt wielokąta
`glViewport`	Ustala obszar okna służący do wyświetlania grafiki

PRACA ZDALNA – INSTALACJA OPENGL

Jeżeli niniejszą instrukcję realizujesz na stanowisku w laboratorium, **OpenGL powinno być zainstalowane**, a więc nie powinno być wymagane pobieranie i instalowanie dodatkowych sterowników. Jeżeli pracujesz na własnym komputerze, konieczna może się okazać instalacja OpenGL. Ta będzie przebiegać inaczej, w zależności od systemu, na którym pracujesz.

Na systemach z rodziny ***nix**, należy otworzyć terminal i wykonać następujące instrukcje:

```
sudo apt update
sudo apt install python3-opengl
```

Na systemach z rodziny **Windows**, instalacja będzie wymagać pobrania API OpenGL ze strony Khronos Group:

Pobieranie: https://www.khronos.org/opengl/wiki/Getting_Started#Downloading_OpenGL
Specyfikacja: https://www.opengl.org/sdk/

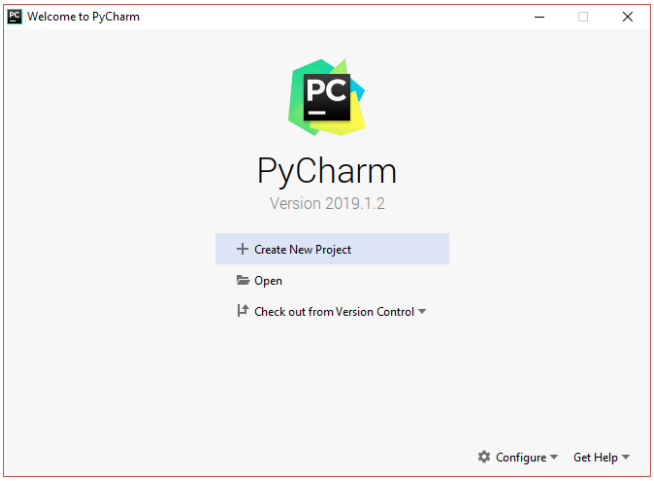
Dodatkowo należy pobrać bibliotekę freeglut ze strony twórców:

Pobieranie: http://freeglut.sourceforge.net/index.php#download
Specyfikacja: http://freeglut.sourceforge.net/docs/api.php

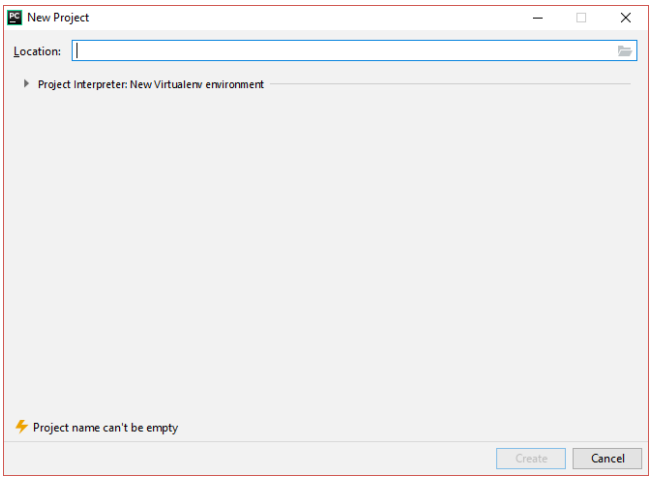
TWORZENIE NOWEGO PROJEKTU W ŚRODOWISKU PYCHARM (PRZYPOMNIENIE)

Uruchom środowisko JetBrains **PyCharm**. Możesz również skorzystać z własnego komputera lub z innych środowisk obecnych na komputerach w laboratorium. Upewnij się, że Twoje środowisko obsługuje język Python w wersji 3.6.1 lub nowszej. Stwórz nowy projekt.

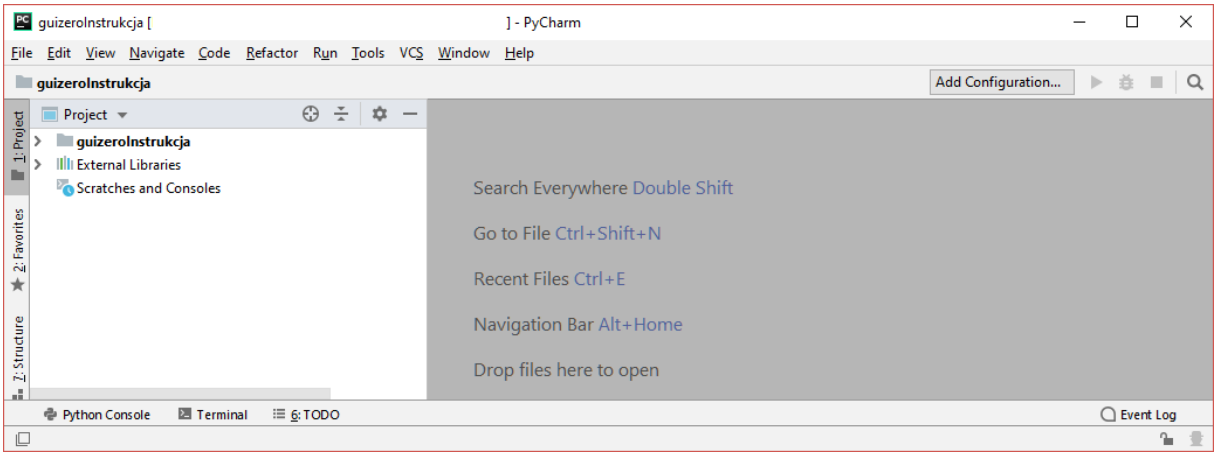
Krok 1. W oknie startowym wybierz opcję „Create New Project"



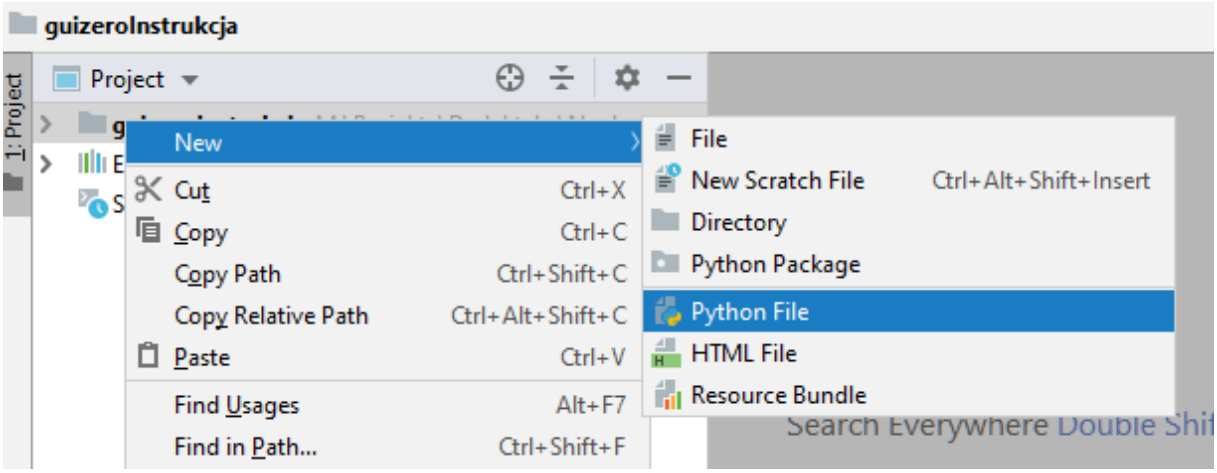
Krok 2. Wybierz miejsce zapisu i nazwę projektu. W nazwie zawrzyj swoje imię i nazwisko oraz aktualną datę, np. „Adam Mickiewicz 11 02 2020"



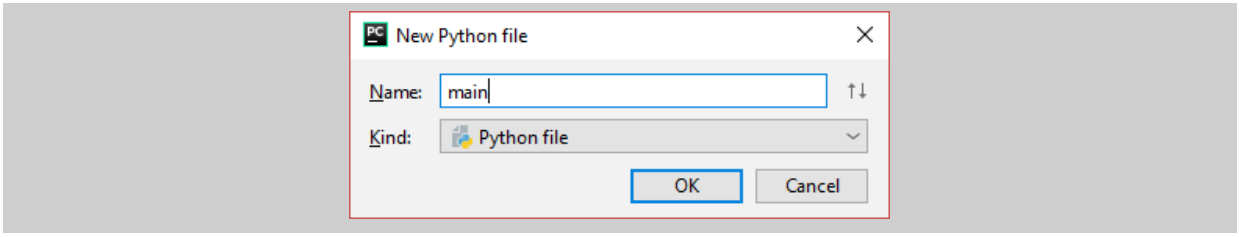
Krok 3. Gdy środowisko skończy tworzenie projektu, ukaże Ci się okno edytora.



Krok 4. Kliknij prawym przyciskiem myszy na nazwę projektu. Ukaże Ci się menu kontekstowe. Najedź kursorem myszy na pozycję „New", po czym wybierz opcję „Python File"



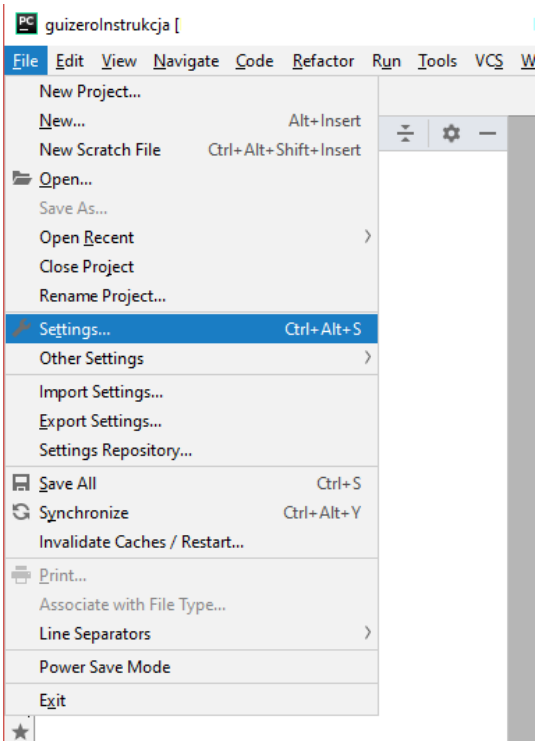
Krok 5. Nadaj plikowi nazwę *main* (z rozszerzeniem *.py!), po czym wciśnij przycisk „OK". Plik powinien zostać automatycznie otwarty do edycji. Będzie to główny plik naszej aplikacji



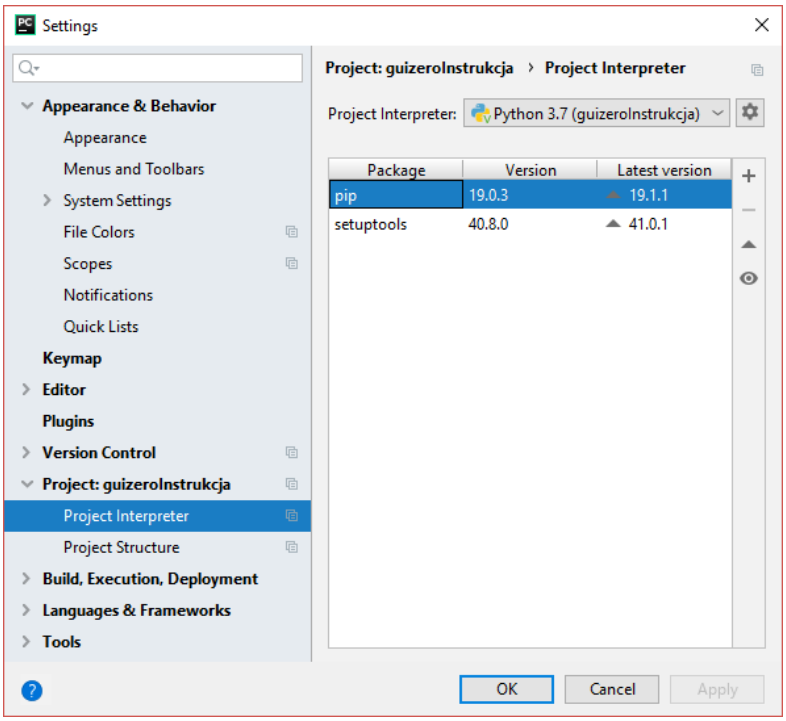
INSTALACJA PAKIETU PYOPENGL

Kolejnym krokiem jest integracja pakietu „PyOpenGL" z projektem. Aby to zrobić, wykonaj następujące kroki:

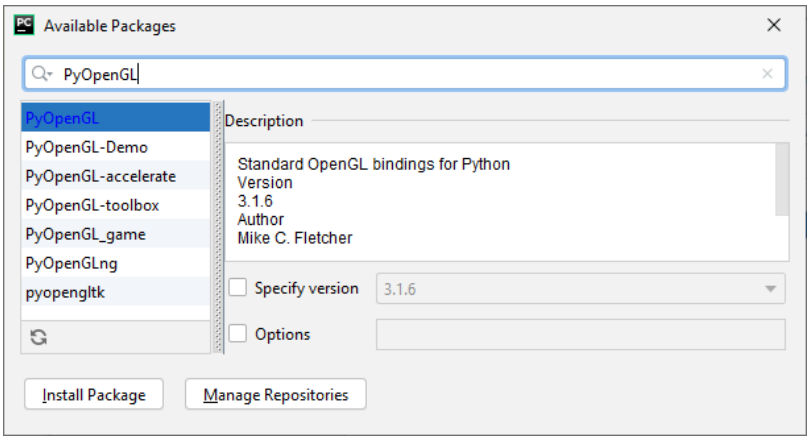
Krok 1. Z paska menu wybierz pozycję „File", a następnie „Settings"



Krok 2. W oknie ustawień rozwiń zakładkę „Project: nazwa projektu" (na poniższym rysunku projekt nazwano „guizerolnstrukcja"), po czym wybierz pozycję „Project Interpreter". W oknie po prawej powinny pokazać się dwie pozycje. Kliknij na ikonkę „+" znajdującą się po prawej stronie nagłówka tabeli.



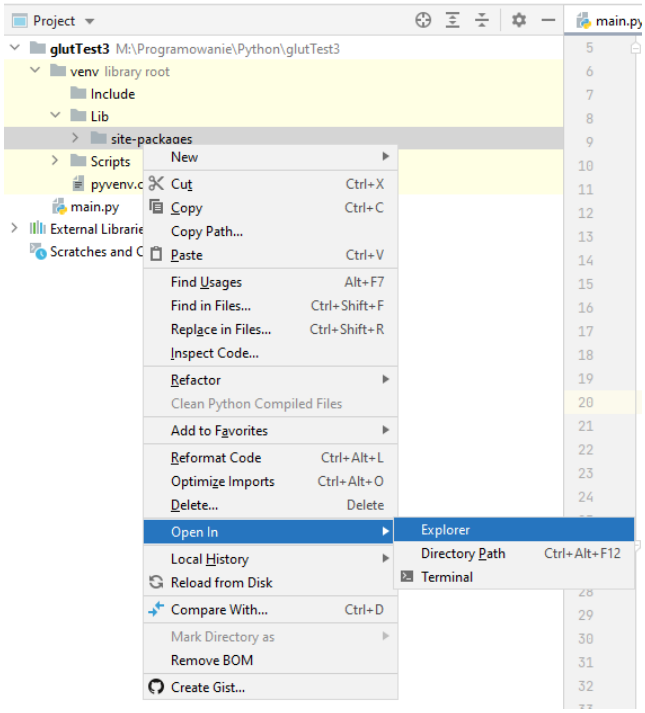
Krok 3. W oknie wyszukiwania wpisz „PyOpenGL", wybierz z listy pakiet „PyOpenGL" (bez żadnych dopisków), po czym wciśnij przycisk „Install Package" i poczekaj aż środowisko zainstaluje pakiet. W oknie „Available Packages" wyświetli się informacja. Na komputerach o mniejszej mocy obliczeniowej, może to zająć do kilku minut.



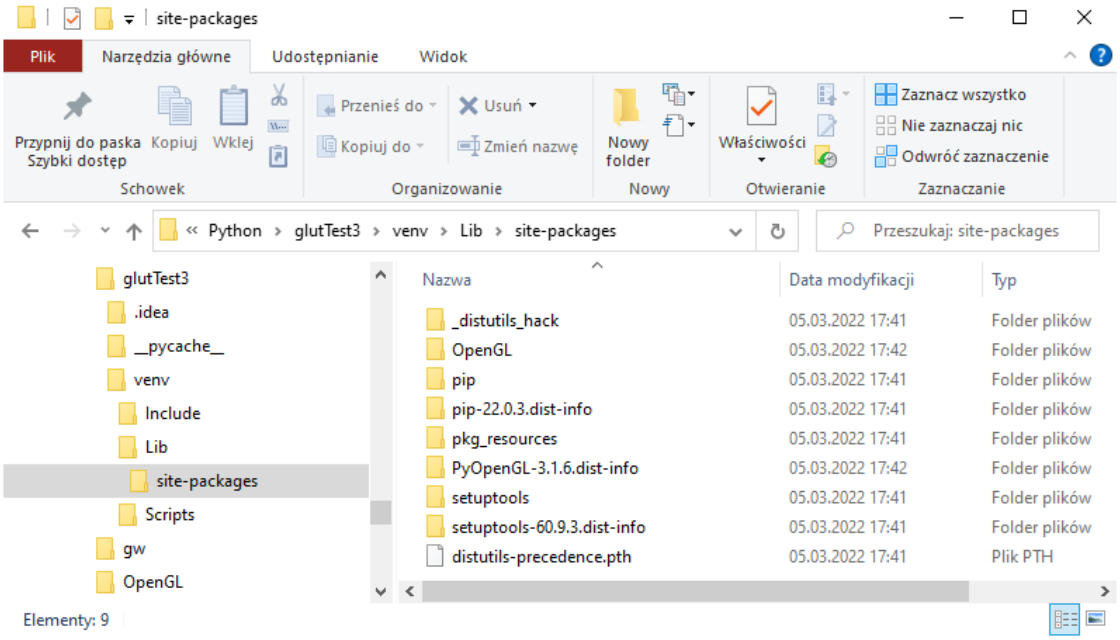
Krok 4. Jeżeli projekt tworzysz na systemie z rodziny ***nix**, po zainstalowaniu pakietu możesz zamknąć okno pakietów oraz okno ustawień, wrócić do okna edytora i rozpocząć czytanie kolejnej sekcji instrukcji. API OpenGL zostało zainstalowane i korzystanie z niego powinno być możliwe. Jeżeli projekt tworzysz na systemie z rodziny **Windows**, należy wykonać dodatkowe kroki.

Krok 5. Pobierz plik znajdujący się pod następującym adresem: http://www.stawarz.edu.pl/informatyka2/L05_pyopengl316.zip

Krok 6. Rozwiń katalog „venv" znajdujący się w widoku projektu. Następnie rozwiń katalog „Lib", a później kliknij prawym przyciskiem myszy na katalog „site packages" i wybierz z menu kontekstowego opcję „Open In"→„Explorer".



Krok 7. Wypakuj zawartość archiwum pobranego w kroku 5, **do wnętrza** katalogu site-packages, który został otwarty w kroku 6. Jeżeli system zapyta o podmianę plików, **potwierdź**. Ostatecznie, w katalogu site-packages, powinny znajdować się następujące podkatalogi:



ZAWARTOŚĆ PLIKU MAIN

Do pliku main.py skopiuj zawartość znajdującą się pod adresem: http://www.stawarz.edu.pl/informatyka2/lab05/main.py. Kod, który znajduje się pod tym adresem, zostanie omówiony w kolejnych rozdziałach instrukcji.

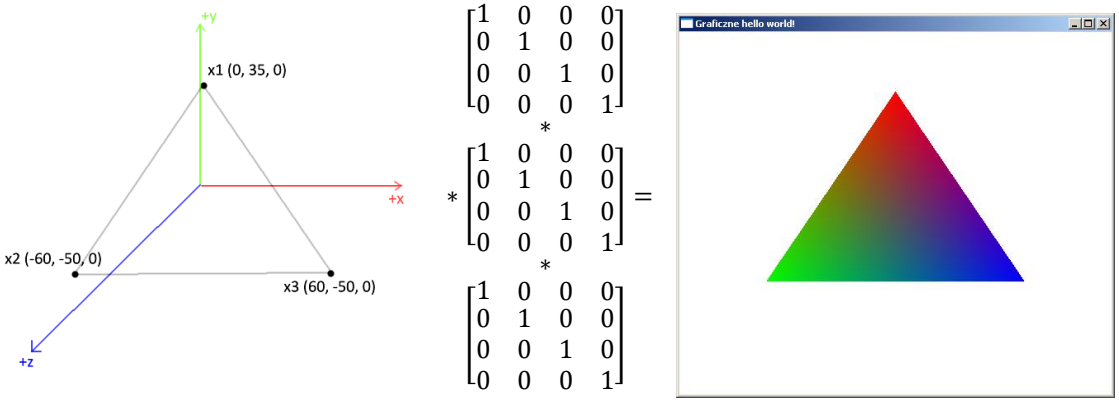
REPREZENTACJA OBIEKTÓW W PRZESTRZENI TRÓJWYMIAROWEJ

Istnieją dwa sposoby reprezentacji obiektu w przestrzeni trójwymiarowej. Pierwszym sposobem jest reprezentacja za pomocą wzoru matematycznego, który pozwala określić strukturę obiektu z dowolną dokładnością. Sposób ten możliwy jest do zastosowania wyłącznie w bardzo niewielkiej liczbie przypadków, w których takim wzorem dysponujemy. Przeważnie, w ten sposób reprezentowane są proste figury czy krzywe.

Częstszym sposobem reprezentacji jest wykreowanie obiektu za pomocą pewnej skończonej liczby wierzchołków, których pozycja jest ściśle określona w przestrzeni. Poszczególne wierzchołki są łączone, tworząc wielokąty płaskie, które następnie są poddawane odpowiednim transformacjom przestrzennym i rzutowane na dwuwymiarową płaszczyznę ekranu. Najprostszym wielokątem, który można w ten sposób zaprezentować, jest oczywiście trójkąt.

Transformacja obiektu, a więc jego przesunięcie, obrót lub skalowanie (a także połączenie dowolnych lub wszystkich tych rodzajów transformacji), również jest operacją czysto matematyczną. Współrzędne punktu są zapisywane do wektora, który jest następnie mnożony przez pewną macierz, opisującą transformację. Macierz taka jest zwaną macierzą transformacji. Wynikiem mnożenia jest nowa pozycja punktu, uwzględniająca wszystkie zapisane w macierzy transformacje.

OpenGL przechowuje trzy macierze transformacji. Pierwsza z nich, zwana macierzą modelu, przelicza współrzędne lokalne, na współrzędne globalne. Druga, zwana macierzą widoku, przelicza współrzędne globalne, na współrzędne obserwatora. Ostatnia, macierz projekcji, przelicza współrzędne obserwatora, na współrzędne ekranu.



Tworzenie wielokątów za pomocą API OpenGL jest bardzo łatwe. Definicja wielokąta rozpoczyna się od wywołania funkcji `glBegin`, a kończy wywołaniem funkcji `glEnd`. Jedynym

argumentem `glBegin` jest typ wielokąta, który będziemy tworzyć. Możesz przyjąć, że **zawsze** przekazujemy tam wartość `GL_POLYGON`. Pomiędzy wywołaniem tych dwóch funkcji, znajduje się informacja o punktach tworzących wielokąt. Każdy punkt jest zdefiniowany poprzez wywołanie funkcji `glVertex3f`, z argumentami, których wartość reprezentuje współrzędne (x, y, z) punktu. Powyższy trójkąt, można więc zdefiniować w następujący sposób:

```
glBegin(GL_POLYGON)
glVertex3f(0.0, 35.0, 0.0)
glVertex3f(-60.0, -50.0, 0.0)
glVertex3f(60.0, -50.0, 0.0)
glEnd()
```

Każdy wierzchołek można wzbogacić o dodatkowe parametry. Kolor wierzchołka można ustalić za pomocą funkcji `glColor3f`. Przyjmuje ona trzy wartości – natężenie składowej czerwonej, zielonej i niebieskiej. Wszystkie składowe muszą być z zakresu od 0, do 1. Wartość 1 oznacza, że dana składowa ma maksymalną wartość.

Podobnie proste jest tworzenie macierzy transformacji. Służą do tego trzy funkcje OpenGL: `glTranslatef`, `glRotatef` i `glScalef`. Funkcja `glTranslatef` przyjmuje trzy argumenty – przesunięcie w osi x, y i z. Funkcja `glRotatef` przyjmuje cztery argumenty, z czego pierwszy jest kątem o jaki należy dokonać obrotu, a pozostałe wskazują w której osi odbędzie się obrót. Ostatecznie, kod dokonujący transformacji i definicji wielokąta, mógłby więc wyglądać następująco:

```
# Zresetuj macierz transformacji
glLoadIdentity()

# Utworz macierz perspektywy
gluPerspective(50.0, szerokoscOkna / wysokoscOkna, 1.0, 1500.0)

# Przesun "kamere"
glRotatef(25,1,0,0)
glTranslatef(0,-55.0,zKamery)
glRotatef(xKamery,0,1,0)

# Narysuj kolorowy trojkat
glBegin(GL_POLYGON)
glColor3f(1.0, 0.0, 0.0)
glVertex3f(0.0, 35.0, 0.0)
glColor3f(0.0, 1.0, 0.0)
glVertex3f(-60.0, -50.0, 0.0)
glColor3f(0.0, 0.0, 1.0)
glVertex3f(60.0, -50.0, 0.0)
glEnd()
```

ZMIENNE POMOCNICZE

Zanim rozpoczniemy tworzenie logiki aplikacji, należy rozważyć i zadeklarować wszystkie zmienne, które będą współdzielone pomiędzy poszczególnymi elementami. W przestrzeni globalnej potrzebne będą parametry okna, takie jak jego szerokość i wysokość

oraz „uchwyt" służący do komunikacji. Jeżeli planowane jest poruszanie kamerą, należy przechować również pozycję i obrót kamery.

```
# Ustawienia okna
szerokoscOkna = 800
wysokoscOkna = 600
uchwytDoOkna = 0

# Aktualna pozycja i obrot kamery
xKamery = 0
zKamery = -150
```

FUNKCJA MAIN

Przyjęło się, że każdy program powinien posiadać funkcję `main`. Ciało tej funkcji będzie wyjątkowo proste. Najpierw wywołamy funkcję pomocniczą `inicjalizuj` (realizującą kroki 1 i 2 algorytmu przedstawionego na początku), a następnie funkcję `glutMainLoop`, która zajmie się całą resztą:

```
# Funkcja main
def main():
    # Inicjalizuj aplikacje
    inicjalizuj()

    # Pozwol bibliotece GLUT zajac sie obsluga grafiki
    glutMainLoop()
```

Dodatkowo, zadbamy o to by funkcja była automatycznie wywołana przez nasz program:

```
# Uruchom funkcjÄ main
if __name__ == "__main__":
    main()
```

FUNKCJA INICJALIZUJ

Przyjdźmy do analizy funkcji `inicjalizuj`. Jej zadaniem jest zainicjalizować okno aplikacji i przygotować je do wyświetlania grafiki. Inicjalizacja okna wymaga wielu linijek kodu, więc zostanie wyodrębniona do osobnej funkcji, którą nazwiemy `InicjalizacjaOkna`. Dzięki takiemu podziałowi, łatwiej będzie znaleźć odpowiedni fragment kodu, jeżeli zajdzie konieczność modyfikacji.

```
# Definicje funkcji pomocniczych
def inicjalizuj():
    # Zainicjalizuj okno
    inicjalizacjaOkna()

    # Wyczysc bufor koloru
    glClearColor(1.0, 1.0, 1.0, 0.0)

    # Dodatkowe opcje
    glShadeModel(GL_SMOOTH)           # Cieniowanie miekkie
    glEnable(GL_DEPTH_TEST)           # Testowanie glebi
```

Funkcja `glClearColor` wypełnia cały bufor obrazu za pomocą koloru o określonych parametrach. W tym wypadku, wypełniamy bufor kolorem białym. Funkcja `glShadeModel` z argumentem o wartości `GL_SMOOTH` oznacza, że nasza aplikacja będzie stosować cieniowanie miękkie. To jest, każdy wierzchołek będzie mógł być oświetlony osobno, światłem o innym natężeniu. Ostatnią instrukcją jest `glEnable` z parametrem `GL_DEPTH_TEST`. Sama funkcja `glEnable` służy do aktywacji pewnych API OpenGL, natomiast `GL_DEPTH_TEST` mówi o tym, że przed narysowaniem obiektu na ekranie, należy się upewnić czy nic go nie przesłania.

Funkcja `inicjalizacjaOkna` wyglądać będzie następująco:

```
def inicjalizacjaOkna():
    # Uruchom biblioteke GLUT
    glutInit()

    # Ustaw wielkosc okna
    glutInitWindowSize(szerokoscOkna, wysokoscOkna)

    # Ustaw tryb wyswietlania
    glutInitDisplayMode(
        GLUT_DEPTH              # Uzywamy bufora glebi,
        | GLUT_DOUBLE           # podwojne buforowanie
        | GLUT_RGBA             # i schemat kolorow RGBA
    )

    # Utworz okno
    uchwytDoOkna = glutCreateWindow("Graficzne hello world!")

    # Jezeli utworzenie okna sie nie powiodlo, zakoncz aplikacje
    if uchwytDoOkna < 1:
        exit(1)

    # Ustaw dodatkowe funkcje obslugi zdarzen
    glutReshapeFunc(zmienRozmiarOkna)                # (1)
    glutKeyboardFunc(obslugaKlawiatury)              # (2)
    glutDisplayFunc(rysujOkno)                       # (3)
```

Komentarze wskazują zadania poszczególnych linijek kodu. Nie musisz zapamiętać konkretnych instrukcji, ale warto zwrócić uwagę na kolejność wywołania instrukcji. Najwięcej uwagi wymagają linijki oznaczone komentarzami „(1)", „(2)" i „(3)". Służą one do wskazania funkcji, które będą użyte w przypadku zajścia konkretnych zdarzeń.

Funkcja `zmienRozmiarOkna`, z linijki z komentarzem „(1)", będzie wywołana, gdy użytkownik zmieni nazwę okna. Rozmiar można zmienić tak, jak rozmiar dowolnej innej aplikacji. Linijka (2) ustala funkcję, która będzie wywołana, gdy nastąpi wciśnięcie klawisza na klawiaturze. Ostatnia linijka, oznaczona jako (3), będzie wywoływana okresowo, gdy trzeba będzie odświeżyć zawartość okna. Omówmy te funkcje.

FUNKCJA OBSŁUGUJĄCA ZMIANĘ ROZMIARU OKNA

Pierwszą pomocniczą funkcją, jest funkcja `zmienRozmiarOkna`. Biblioteka GLUT wywoła ją automatycznie (gdyż tak ustaliliśmy w linijce „(1)"., gdy użytkownik zmieni rozmiar okna aplikacji i automatycznie przekazuje nowe wymiary jako argumenty.

W przypadku omawianej aplikacji, cel jest dość prosty: nadpisać wartość odpowiednich zmiennych globalnych i utworzyć nowy obszar wyświetlania grafiki, o nowych wymiarach. Realizuje to poniższy kod:

```
def zmienRozmiarOkna(nowaSzerokosc, nowaWysokosc):
    # Poinformuj funkcję, że istnieją zmienne globalne
    global szerokoscOkna
    global wysokoscOkna

    # Przypisz nowe wymiary do zmiennych przechowujacych aktualne wymiary
    szerokoscOkna = nowaSzerokosc
    wysokoscOkna = nowaWysokosc

    # Utworz nowy widok
    glViewport(0, 0, szerokoscOkna, wysokoscOkna)
```

FUNKCJA OBSŁUGUJĄCA KLAWIATURĘ

Następną pomocniczą funkcją jest funkcja `obslugaKlawiatury`. Biblioteka GLUT wywoła ją automatycznie, jeżeli wykryje wciśnięcie klawisza na klawiaturze (gdyż tak ustaliliśmy w linijce „(2)"). Jako pierwszy argument otrzymywany jest klawisz, a dwa pozostałe argumenty, wskazują aktualną pozycję kursora myszy na ekranie. Jeżeli więcej niż jeden klawisz jest wciśnięty naraz, jako argument przekazywany jest wyłącznie ostatni.

W projektowanej w trakcie zajęć aplikacji, interesujące są wyłącznie cztery klawisze. Klawisz 'w', będzie służyć do poruszenia kamery wprzód (a więc w stronę pozytywnej części osi Z), a klawisz 's', będzie służyć do poruszania kamery do tyłu (a więc w stronę negatywnej części osi Z). Klawisze 'a' i 'd', będą służyć odpowiednio do obrotu w lewo i w prawo, wzdłuż osi X. Po oczytaniu stanu klawiszy, należy wymusić odświeżenie ekranu.

```
def obslugaKlawiatury(klawisz, x, y):
    # Poinformuj funkcję, że istnieją zmienne globalne
    global xKamery
    global zKamery

    # Podejmij inna akcje, w zaleznosci od tego, jaki klawisz wcisnieto
    if klawisz == b'w':
        zKamery+=1
    elif klawisz == b's':
        zKamery-=1
    elif klawisz == b'a':
        xKamery-=2
    elif klawisz == b'd':
```

```
        xKamery+=2

    # Wymus odswiezenie okna
    glutPostRedisplay()
```

Oczywiście w zaprezentowanym fragmencie nie poruszamy kamerą. Zmieniamy po prostu wartość zmiennych globalnych. Sam ruch kamerą można wykonać za pomocą odpowiedniego użycia funkcji `glTranslatef` i `glRotatef`, ale to nie jest zadanie dla obecnej funkcji.

FUNKCJA OBSŁUGUJĄCA ODŚWIEŻENIE EKRANU

Ostatnią i być może najważniejszą funkcją pomocniczą, wywoływaną automatycznie przez bibliotekę GLUT, jest funkcja odświeżająca zawartość ekranu. W naszym programie, nadaliśmy jej nazwę `rysujOkno` i poinstruowaliśmy GLUT, żeby ją wywoływał automatycznie, w linijce oznaczonej komentarze „(3)". Ciało funkcji powinno się zacząć od wywołania funkcji `glClear` i wyczyszczenia buforów koloru i głębi:

```
# Wyczyść bufory koloru i glebi
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

Następnie należy zresetować macierz transformacji. Żeby zresetować transformację, należy nadpisać aktualną macierz transformacji, macierzą jednostkową. Dowolny wektor pomnożony przez macierz jednostkową, nie ulega zmianie. OpenGL udostępnia w tym celu funkcję `glLoadIdentity`:

```
# Zresetuj macierz transformacji
glLoadIdentity()
```

Kolejnym krokiem jest załadowanie do macierzy transformacji, macierzy reprezentującej rzut perspektywiczny, dostosowany do aktualnych wymiarów okna aplikacji. Ostatnie dwie wartości oznaczają minimalną i maksymalną odległość wyświetlanych obiektów:

```
# Utworz macierz perspektywy
gluPerspective(50.0, szerokoscOkna / wysokoscOkna, 1.0, 1500.0)
```

Jeżeli planujemy w jakikolwiek sposób poruszyć kamerą, to teraz przyszedł na to czas. W naszej aplikacji, najpierw obrócimy kamerę o 25 stopni w osi X (czyli popatrzymy „w dół"), następnie przesuniemy kamerę o 55 punktów do góry i o pewną wartość, wskazywaną przez zmienną `zKamery`, w tył. Później obrócimy kamerę o `xKamery` stopni w osi Y. Wartość zmiennych `zKamery` oraz `xKamery`, ustalana jest w funkcji `obslugaKlawiatury`.

```
# Przesun "kamere"
glRotatef(25,1,0,0)
glTranslatef(0,-55.0,zKamery)
glRotatef(xKamery,0,1,0)
```

Gdy ustalimy już pozycję obiektów na scenie, można je „narysować". Na razie narysujemy trójkąt, którego rogi będą miały różne kolory. Zrobimy to za pomocą kodu pokazanego w sekcji „*reprezentacja obiektów w przestrzeni trójwymiarowej*".

Ostatnim krokiem jest zamiana miejscami buforu roboczego z buforem ekranu i wyświetlenie grafiki:

```
# Zamien bufor tymczasowy z buforem ekranu i wymus odswiezenie
glutSwapBuffers()
glutPostRedisplay()
```

ZADANIA DO SAMODZIELNEGO WYKONANIA

ZADANIE 1

Uruchom projekt. Spróbuj wykonać ruch kamerą we wszystkich dostępnych osiach, a także zmienić rozmiar okna aplikacji.

ZADANIE 2

Do projektu dodaj nowy plik i nazwij go „Teren.py". Wklej do niego zawartość dostępną pod następującym adresem: http://www.stawarz.edu.pl/informatyka2/lab05/Teren.py.

Na górze pliku main.py, tuż pod pozostałymi poleceniami importu, dodaj linijkę:

```
from Teren import *
```

Zapoznaj się z definicją klasy znajdującą się w pliku `Teren.py` oraz z definicją funkcji składowych.

ZADANIE 3

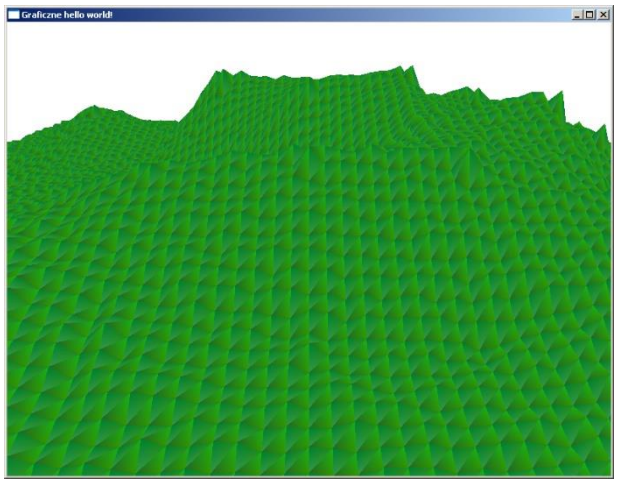
W pliku `main.py`, w miejscu, w którym tworzone są zmienne globalne, zaraz pod definicją zmiennej globalnej `zKamery`, dodaj następujący fragment kodu:

```
# Zmienna przechowujaca teren
wzgorze = Teren()
```

Z funkcji `rysujOkno`, usuń fragment kodu znajdujący się pomiędzy komentarzami „*Tutaj umieszczamy instrukcje tworzace grafike*" i „*Zamien bufor tymczasowy z buforem ekranu i wymus odswiezenie*". Fragment ten rysuje pokolorowany trójkąt. Zamiast niego, wstaw informację o zmiennej globalnej „wzgorze" oraz wywołanie funkcji `rysuj` klasy `Teren`, na obiekcie `wzgorze`:

```
// Tutaj umieszczamy instrukcje tworzace grafike
global wzgorze
wzgorze.rysuj()
```

Uruchom program. Jeżeli wszystko jest poprawne, widok powinien być podobny do następującego:



ZADANIE 4

W pliku `Teren.py`, tuż nad definicją funkcji `rysuj`, dodaj definicje następujących funkcji:

```
def obliczWektorNormalny(self, h):
      # Zmienne pomocnicze
      vP1 = [15.0, h[3] - h[0], 15.0]
      vP2 = [-15.0, h[2] - h[1], 15.0]

      # Wyznacz wektor prostopadly
      wynik = [0,0,0]
      wynik[0] = vP1[1]*vP2[2] - vP1[2]*vP2[2]
      wynik[1] = vP1[2]*vP2[0] - vP1[0]*vP2[2]
      wynik[2] = vP1[0]*vP2[1] - vP1[1]*vP2[0]

      # Oblicz dlugosc wektora
      dlugosc = sqrt(wynik[0]*wynik[0] + wynik[1]*wynik[1]
                           + wynik[2]*wynik[2])

      # Znormalizuj wektor
      wynik[0] /= -dlugosc
      wynik[1] /= -dlugosc
      wynik[2] /= -dlugosc

      # Zwroc wynik
      return wynik

def pobierzWysokoscSasiadow(self, x, y):
      # Zmienne pomocnicze
      xPrev = max(x-1,0)
      yPrev = max(y-1,0)
      xNext = min(x+1,self.pointsPerRow-1)
```

```
        yNext = min(y+1,self.pointsPerRow-1)

        # Zmienne pomocnicze
        hSasiadow = [0,0,0,0]

        # Gora-lewo
        hSasiadow[0] = self.wysokosc[xPrev][yPrev]
        # Gora-prawo
        hSasiadow[1] = self.wysokosc[xNext][yPrev]
        # Dol-lewo
        hSasiadow[2] = self.wysokosc[xPrev][yNext]
        # Dol-prawo
        hSasiadow[3] = self.wysokosc[xNext][yNext]

        # Zwroc wynik
        return hSasiadow
```

Funkcja `obliczWektorNormalny`, służy do wyznaczenia wektora normalnego (a więc prostopadłego) do płaszczyzny otaczającej wierzchołek, którego sąsiadujące wierzchołki mają zadaną wysokość. Jak zapewne pamiętasz z fizyki, **kąt odbicia światła jest równy kątowi padania na płaszczyznę**. Wyznaczenie wektora normalnego, pozwala ustalić, pod jakim kątem na daną płaszczyznę pada światło.

Mając płaszczyznę opisaną trzema punktami, v_1, v_2 i v_3, wektor normalny można wyznaczyć z następującego wzoru: $v_{normalny} = (v_2 - v_1) \times (v_3 - v_1)$. Upewnij się czy po wprowadzeniu zmian, program kompiluje się i uruchamia poprawnie.

ZADANIE 5 (PLUS MOŻE ZDOBYĆ KAŻDA OSOBA, KTÓRA SKOŃCZY ZADANIE)

Zmień definicję funkcji `rysuj` klasy `Teren`, na następującą:

```
def rysuj(self):
    glPolygonMode(GL_FRONT, GL_FILL)
    offset = round(self.pointsPerRow/2)
    glColor3f(0.4, 0.4, 0.4)

    for x in range(self.pointsPerRow-1):
      for y in range(self.pointsPerRow-1):
        # Zmienne pomocnicze
        v1=[self.plotSize*(-offset+x),self.wysokosc[x][y], self.plotSize * (-offset+y)]
        v2=[self.plotSize*(-offset+x),self.wysokosc[x][y+1], self.plotSize * (-(offset-1)++y)]
        v3=[self.plotSize*(-(offset-1)+x),self.wysokosc[x+1][y], self.plotSize * (-offset+y)]
        v4=[self.plotSize*(-(offset-1)++x),self.wysokosc[x+1][y+1], self.plotSize * (-(offset-1)++y)]

        glBegin(GL_POLYGON)

        # Narysuj trojkat
        wysokoscSasiadow = self.pobierzWysokoscSasiadow(x,y)
        wektorNormalny = self.obliczWektorNormalny(wysokoscSasiadow)
        glNormal3f(wektorNormalny[0], wektorNormalny[1], wektorNormalny[2])
        glVertex3f(v1[0], v1[1], v1[2])

        wysokoscSasiadow = self.pobierzWysokoscSasiadow(x,y+1)
        wektorNormalny = self.obliczWektorNormalny(wysokoscSasiadow)
        glNormal3f(wektorNormalny[0], wektorNormalny[1], wektorNormalny[2])
        glVertex3f(v2[0], v2[1], v2[2])

        wysokoscSasiadow = self.pobierzWysokoscSasiadow(x+1,y)
        wektorNormalny = self.obliczWektorNormalny(wysokoscSasiadow)
        glNormal3f(wektorNormalny[0], wektorNormalny[1], wektorNormalny[2])
```

```
        glVertex3f(v3[0], v3[1], v3[2])
        glEnd()

        glBegin(GL_POLYGON)

        # Narysuj trojkat
        wysokoscSasiadow = self.pobierzWysokoscSasiadow(x+1,y)
        wektorNormalny = self.obliczWektorNormalny(wysokoscSasiadow)
        glNormal3f(wektorNormalny[0], wektorNormalny[1], wektorNormalny[2])
        glVertex3f(v3[0], v3[1], v3[2])

        wysokoscSasiadow = self.pobierzWysokoscSasiadow(x,y+1)
        wektorNormalny = self.obliczWektorNormalny(wysokoscSasiadow)
        glNormal3f(wektorNormalny[0], wektorNormalny[1], wektorNormalny[2])
        glVertex3f(v2[0], v2[1], v2[2])

        wysokoscSasiadow = self.pobierzWysokoscSasiadow(x+1,y+1)
        wektorNormalny = self.obliczWektorNormalny(wysokoscSasiadow)
        glNormal3f(wektorNormalny[0], wektorNormalny[1], wektorNormalny[2])
        glVertex3f(v4[0], v4[1], v4[2])

        glEnd()
```

W pliku `main.py`, pod deklaracją zmiennej `wzgorze`, dodaj następujący fragment kodu.

```
# Ustawienia oswietlenia
light_ambient = [0.0, 0.0, 0.0, 1.0]        # Kolor swiatla otoczenia
light_diffuse = [0.95, 0.95, 0.75, 1.0]     # Kolor swiatla rozproszonego
light_specular = [0.95, 0.95, 0.95, 1.0]    # Kolor swiatla rozproszonego
lightPos = [450.0, 450.0, 0.0, 1.0]         # Pozycja zrodla swiatla
```

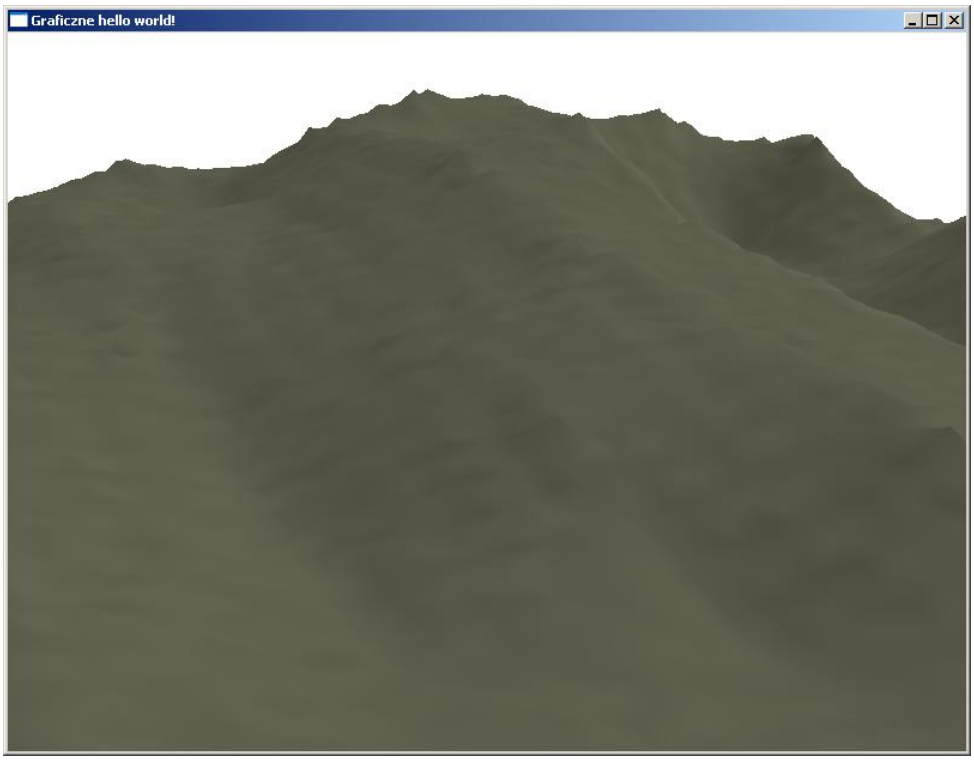
Do funkcji `inicjalizuj`, tuż pod wywołaniem funkcji `glEnable` z argumentem `GL_DEPTH_TEST`, dodaj następujący fragment kodu:

```
        glEnable(GL_LIGHTING)       # Aktywuj obliczanie oswietlenia

        # Aktywuj ustawienia oswietlenia
        glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient)
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse)
        glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular)
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos)

        # Aktywuj zrodlo swiatla
        glEnable(GL_LIGHT0)
```

Jeżeli zmiany zostały wykonane poprawnie, teren powinien być oświetlony tak, jak pokazuje poniższy zrzut ekranu. Obróć kamerę, sprawdź, jak wygląda teren pod różnym kątem.


Graficzne hello world!