Kilka wstępnych uwag dotyczących biblioteki OpenGL

1. OpenGL jest układem dwóch bibliotek: GL i GLU. Biblioteki te są zaimplemantowane na praktycznie wszystkich istniejących systemach operacyjnych (niekomercyjna wersja OpenGL dla Unixów nazywa się Mesa - dostępna w sieci).

   Pliki źródłowe zawierające wywołania funkcji OpenGL mogą być kompilowane w dowolnym środowisku C, C++, Delphi itp. W zależności od systemu operacyjnego i środowiska kompilatora OpenGL używa określonej grupy plików, np. dla kompilatora języka C w środowisku Windows potrzebujemy następujących plików:

   - `gl.h, glu.h` - pliki nagłówkowe konwencjonalnie przechowywane w podkatalogu `GL` katalogu `include` danego kompilatora
   - `opengl32.lib, glu32.lib` (lub w wersji statycznej `opengl32.a, glu32.a` - np. w kompilatorze DevC++) - pliki biblioteczne zawierające funkcje GL i GLU, znajdujace się w katalogu `lib` danego kompilatora (nazwy tych bibliotek mogą się nieco róznić w różnych kompilatorach, np. `opengl.lib`)
   - `opengl32.dll, glu32.dll` - znajdują się w podkatalogu `System32` katalogu `Windows` - wbudowane w system operacyjny Windows.

2. *Zalecane kompilatory*

   - Windows: Visual C++ 6.0, Visual Net, DevC++ (dostępny za darmo w sieci - w razie używania proszę przegrać wersję instalacyjną zawierającą zintegowany kompilator - istnieje wersja środowiska DevC++ bez kompilatora), Borland Builder, Delphi itd.
   - Unix - gcc lub g++

   Na ćwiczeniach będziemy posługiwali się systemem Windows i kompilatorem Visual C++ 6.0.

3. Biblioteki GL i GLU **nie zawierają** żadnej funnkcji określającej okno. Zatem aby wyświetlić na ekranie monitora sceny opisane w OpenGL programista musi użyć zewnętrznej biblioteki okienkowej. Istnieje wiele bibilotek okienkowych przystosowanych do łatwej obsługi OpenGL, jednak w fazie początkowej nauki programowania w OpenGL powinno się używać biblioteki maksymalnie prostej i wymagającej niewielu linii kodu potrzebnego do wyświetlenia w oknie obrazu sceny (punkt ciężkości musi być skierowany na naukę programowania grafiki, a nie na uczenie się skomplikowanych bibilotek okienkowych !). Prostymi bibliotekami okienkowymi, napisanymi specjalnie dla OpenGL są np. biblioteki `GLUT` i `GLAUX`. W zależności od używanego kompilatora przynajmniej część z potrzebnych plików (`glut.h, glut32.lib, glut32.dll` itd.) trzeba dograć do właściwych katalogów kompilatora i systemu

operacyjnego (wszystkie te pliki są łątwo dostępne w sieci). Na ćwiczeniach będziemy posługiwali się biblioteką GLUT (kompilator Visual C++ 6.0 doskonale się z nią "rozumie", choć nie ma wbudowanych jej plików).

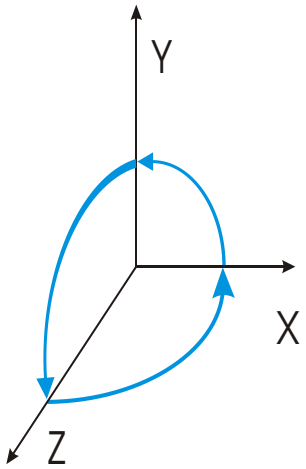
4. *Schemat małego programu przy użyciu biblioteki* GLUT *w sytemie Windows*

```
#include <windows.h> // nie w każdym kompilatorze potrzebne,
                     //wymagane np. w DevC++

#include <GL/glut.h> // dołączenie nagłówka biblioteki glut;
                     //nie ma potrzeby dołącznia nagłówków gl.h i glu.h,
                     //gdyż są one dołączane w kodzie glut.h

void display() //nazwa tej funkcji może być dowolona
{
  wywołania funkcji OpenGL opisującej scenę
}

void main(int argc, char** argv)
{
  glutInit(&argc, argv);

  //inicjacja wszystkich "‘stanów" bieżących biblioteki GLUT (tzn. wartości
  //początkowych zmiennych, którymi się posługuje). Dopóki nie zmieni się
  //któregoś stanu, jawnie wywołując daną funkcję, obowiązuje jego wartość
  //bieżąca; stanami są np. ilość bajtów opisująca kolor piksela,
  //rozdzielczość okna ekranowego, położenie okna na ekranie
  //(wszyskie funkcje biblioteki GLUT mają prefix glut)

  glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);

  //Wszystkie stałe  biblioteki GLUT mają prefix GLUT_. W funkcjach które
  //ich używają są one łączone przez operację sumy bitowej | w jedną wspólną
  //stałą. Wartości stałych podstawowych są na tyle unikalne, że suma
  //bitowa dowolnej ich ilości daje nową, również unikalną wartość.
  //Zapewnia to jednoznaczność opisu.
  //GLUT_RGB - okno będzie używało trójskladnikowego opisu koloru we
  //wspólrzędnych r,g,b (zakładam, że zasada tego opisu jest znana)
  //GLUT_SINGLE - okno będzie miało przydzieloną pamięć na pojedynczy
  //obraz (wielkośc pamięci to ilość pikseli okna razy pamięć na jeden
  //piksel - zob. też wywołania niżej). W przypadku animacji, dla
  //uniknięcia "migotania" potrzebujemy dwóch płaszczyzn obrazu -
```

```
    //przydzielamy wówczas podwójną pamięć - GLUT_DOUBLE

    glutInitWindowSize(400,300);

    //określnie rozmiaru okna w pikselach (szerokość, wysokość)

    glutInitWindowPosition(100,100);

    //określnie położenia lewego, górnego narożnika okna na ekranie
    //obie powyższe wielkości mają (jak wszystko w OpenGL) wartości
    //domyślne ustawione na starcie, ale kilka z nich warto ustawić
    //jawnie, np. żeby nie potrzeba pamiętać ustawień domyślnych

    glutCreateWindow("Scena testowa");

    //przygotowanie struktury okna w pamięci RAM - okno nie jest jeszcze
    //wyświetlone na ekranie ("scena testowa" to przyklad nazwy
    //pokazującej się na górnej belce okna)
    //ponadto nastepuje inicjacja wszystkich "‘stanów" bieżących biblioteki
    //OpenGl (tzn. wartości początkowych zmiennych, którymi się posługuje).
    //Dopóki nie zmieni się  któregoś stanu, jawnie wywołując daną funkcję,
    //obowiązuje jego wartość bieżąca; stanami są np. kolor rysowania
    //bieżącego obiektu, jego materiał, położenie obserwatora itp.

    glutDisplayFunc(display);

    //funkcja ta pobiera nazwę funkcji opisującej scenę ,która ma
    //być wyświetlona w zdefiniowanym oknie - w tym przypadku
    //funkcja nazywa się: display

    glutMainLoop();

    //uruchomienie pętli obsługi zdarzeń. tzn. w szczególności
    // - wyświetlenie zdefinowangeo wcześniej okna na ekranie
    // - załadowanie do okna obrazu sceny, przez wywolanie
    //   funkcji display
    // - oczekiwanie na kolejne zdarzenia (np. exit)
  }
```

5. Istnieje kilka innych istotnych funkcji biblioteki GLUT opisujących m. in.

   - scenę animowaną



Rysunek 1: Prawoskrętny układ używany przez OpenGL

- obsługę myszki
- obsługę klawiatury

Zostaną one omówione na kolejnych zajęciach.

6. *Opis sceny, czyli przykladowa treść funkcji* `display()`

   (a) OpenGL posługuje się kartezjańskim układem trójwymiarowym XYZ, przy czym osie zorientowane są prawoskrętnie, tzn. zgodnie z regułą śruby prawoskrętnej. Orientacja ta wyznacza również obroty dodatnie i ujemne wokół osi X, Y, Z. Obroty dodatnie zaznaczone są na Rysunku 1.

   (b) W skład typowego opisu sceny wchodzi opis:

   - rodzaju rzutowania sceny na obraz: perspektywiczny (podobny do sposobu w jaki patrzy człowiek), równoległy (aksonometryczny - używany często przez architektów)
   - obserwatora: położenie (punkt i orientacja pionowa), kierunek patrzenia
   - obiektów: geometria i właściwości powierzchni (kolor, materiał itp.)
   - źródeł światła: położenie, kolor jasność itp.

   W przypadku, gdy używamy źródła światła musimy wyspecyfikować własności materiałów. Zajmiemy się tym na dalszych zajęciach. Na razie w zakresie właściwości powierzchni będziemy się posługiwali tylko opisem koloru. Ponadto będziemy używać rzutu perspektywicznego jako bardziej intuicyjnego dla obserwatora ludzkiego.

(c) Obiekt w bibliotece OpenGL opisany jest przez swoją powierzchnię, która jest określona przez układ wielokątów, w sposób dokładny w przypadku wielościanów i w sposób przybliżony w przypadku pozostałych brył, np. kuli.

(d) *Opis wielokąta*

- Każdy wielokąt opisany jest w sposób jawny przez przez układ swoich wierzchołków w przestrzeni 3D. Wierzchołki danego wielokąta muszą być zdefiniowane tak, aby wszystkie należały do jednej płaszczyzny (w przeciwnym przypadku wielokąt jest błędnie zdefiniowany).
- W typowym przypadku wierzchołek zdefiniowany jest przez funkcję `glVertex3f(x,y,z)`
  Współrzędne `x,y,z` określają położenie wierzchołka w układzie XYZ.
- Wszystkie funkcje biblioteki GL mają prefix `gl`. Część tych funkcji ma wiele wariantów posiadających tę samą nazwę, ale różniących się ilością parametrów i (lub) typem parametrów. W przypadku `glVertex3f` liczba `3` oznacza, że podajemy trzy współrzędne wierzchołka, litera `f` oznacza, że używamy typu `float` do opisu współrzednych. Inne warianty:
  - `glVertex2f()`
  - `glVertex4f()`
  - `glVertex3d()` - litera `d` oznacza typ `double`
  - `glVertex3i()` - litera `i` oznacza typ `int`
  - itd.
- *Definicja wielokąta*

```
glBegin(stała określająca sposób połączenia listy wierzchołków);

  lista wierzchołków

glEnd();
```

  gdzie `stała określająca sposób połączenia listy wierzchołków` może przyjmować wiele wartości, np.
  - `GL_POLYGON` - wierzchołek pierwszy łączony jest z drugim, ten z trzecim itd. Ostatni łączony jest z pierwszym - w efekcie otrzymujemy `n`-kąt, gdzie `n` jest liczbą wierzchołków
  - `GL_QUADS` - pierwsze cztery wierzchołki tworzą pierwszy czworokąt, następne cztery drugi itd.
  - `GL_TRIANGLES` - pierwsze trzy wierzchołki tworzą pierwszy trójkąt, następne trzy drugi itd.

  Przykładowy trójkąt:

```
glBegin(GL_POLYGON);
```

```
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
glEnd();
```

Proszę przetestować podane powyżej i wszystkie inne możliwości posługując się dokumentacją funkcji `glBegin`.

(e) *Opis koloru*

Dany obiekt rysowany jest w kolorze aktualnie obowiązującym (tylko jeden kolor może być bieżący w danej chwili - na początku (`glutInit`) jako bieżący ustawiany jest kolor biały). Zmianę koloru wykonuje funkcja

`glColor3f(r,g,b)`,

gdzie `r,g,b` są wsólrzędnymi koloru i są znormalizowane do zakresu `[0,1]`.

Kolor jest przypisany do wierzchołków (musi być zdefiniowany przed definicją wierzchołka), a następnie wnętrze wielokąta jest wypełniane zgodnie z bieżącą regułą (domyślnie jako uśrednianie kolorów wierzchołków), np.

```
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_POLYGON);
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
glEnd();
```

W tym przypadku wszystkie wierzchołki mają taki sam kolor (czerwony), więc wnętrze wielokąta też będzie czerwone.

```
glBegin(GL_POLYGON);
  glColor3f(1.0, 0.0, 0.0);
  glVertex3f(0.0, 0.0, 0.0);
  glColor3f(0.0, 1.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glColor3f(0.0, 0.0, 1.0);
  glVertex3f(0.0, 1.0, 0.0);
glEnd();
```

W tym przypadku wierzchołki są różnego koloru (czerwony, zielony, niebieski), więc wnętrze wielokąta będzie (jako średnia) niejednolite.

Uwaga: Szczególnym przypadkiem jest zdefiniowanie (nie zmiana!) koloru tła okna. Wykonuje to funkcja

`glClearColor(r,g,b,1.0)`.

Sens czwartego parametru zostanie wyjaśniony na ćwiczeniach dalszych. Na razie wystarczy wiedzieć, że w ogólności kolor opisujemy czterema składowymi, przy czym dopóki czwartej nie używamy intencjonalnie, to przyjmuje ona wartość 1.

Zmiana koloru tła jest wykonywana przez funkcję

`glClear(GL_COLOR_BUFFER_BIT),`

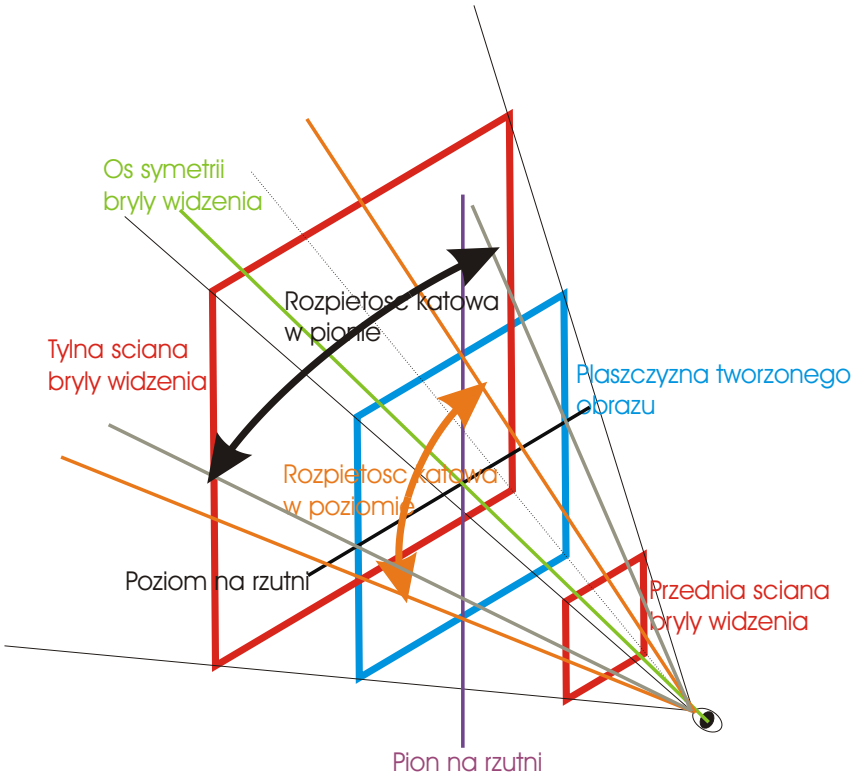
gdzie podany parametr (stała) identyfikuje pamięć okna przeznaczoną na zapis kolorów pikseli. Pamięć ta zostanie wypełniona kolorem wcześniej zdefiniowanym jako kolor tła.

(f) *Opis obserwatora - przypadek rzutu perspektywicznego*

- Domyślnie (dopóki tego jawnie nie zmienimy) obserwator jest w punkcie (0,0,0) i patrzy w kierunku ujemnej części osi `Z`, tzn. np. na punkt (0,0,-1).
- Przestrzeń, którą obejmuje wzrokiem jest określona tzw. *bryłą widzenia*, która w przypadku rzutu perspektywicznego jest ostrosłupem o podstawie prostokąta, osi symetrii wyznaczającej główny kierunek obserwacji oraz określonej rozpietości kątowej w pionie i w poziomie (Rysunek 2). Obserwator znajduje się w wierzchołku tej bryły. Obraz tworzony jest na rzutni równoległej do przedniej i tylnej ściany bryły widzenia.
- *Zmiana parametrów bryły widzenia*
  `gluPerspective(alfaY, aspect, f, b)`,
  gdzie `alfaY` to rozpiętość kątowa w pionie, `aspect` oznacza stosunek rozpiętości kątowej w poziomie do rozpietości w pionie, `f,b` są odpowiednio odległością obserwatora od przedniej i tylnej ściany bryły widzenia, przy czym obie muszą być dodatnie i $f < b$.
- *Zmiana obserwatora*
  `gluLookAt(ex,ey,ez, px,py,pz, dx,dy,dz)`,
  gdzie `ex,ey,ez` oznaczają położenie oka obserwatora, `px,py,pz` są współrzędnymi wyróżnionego punktu na linii patrzenia obserwatora, `dx,dy,dz` są współrzędnymi wektora[1] (nie punktu !), określającego nowy kierunek pionu obserwatora.
- Położenie obserwatora, jego kierunek patrzenia i pion muszą być adekwatne do sceny, na którą ma patrzeć. W szczególności nie może się znajdować wewnątrz jakiegoś obiektu, patrzeć w pustą przestrzeń itp.

(g) **Uwaga:** Ponieważ OpenGL jest układem typu *klient-serwer* (aplikacja jest klientem, biblioteka zainstalowana w danym systemie jest serwerem), więc po zdefiniwaniu całej sceny należy przekazać serwerowi żądanie wykonania wszystkich funkcji. Robi się to przez wywołanie

[1]Współrzędne wektora to różnica wartości pomiędzy jego punktem końcowym i początkowym. W aspekcie kierunku wektory [3,3,0], [1,1,0], [10,10,0] oznaczają to samo.



Rysunek 2: Bryła widzenia dla rzutu perspektywicznego

`glFlush()`.

(h) *Kolejność wywołań w funkcji opisującej scenę*

Z przyczyn, które zostaną wyjaśnione na ćwiczeniach nr 2, chwilowo należy zachować następującą kolejność wywołań omówionych wyżej aspektów sceny:

```
glClearColor(r,g,b,1);
glClear(GL_COLOR_BUFFER_BIT);
gluPerspective(alfaY, aspect, f, b);
gluLookAt(ex,ey,ez, px,py,pz, dx,dy,dz);
opis obiektów
glFlush();
```

(i) **Uwaga:** Jeżeli w trakcie przeprowadzanych testów zostanie zdefiniowana scena, w której obserwator widzi obiekty wzajemnie się zasłaniające, to z przyczyn, które zostaną wyjaśnione na wykładzie 1, przed powyższymi wywołaniami należy dodać linię

```
glEnable(GL_DEPTH_TEST); //umożliwia właściwe zasłanianie się obiektów
                         // z punktu widzenia obserwatora
```

oraz zmodyfikować dwie inne linie kodu:

- Zamiast
  `glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);`
  napisać
  `glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE | GLUT_DEPTH);`
- Zamiast
  `glClear(GL_COLOR_BUFFER_BIT);`
  napisać
  `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`

(j) *Przykładowa funkcja* `display()`

```
display()
{
glClearColor(1.0,1.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);
gluPerspective(45.0, 1.0, 0.1, 10.0);
gluLookAt(0.0, 0.0, 3.0,  0.0, 0.0, 0.0,  0.0, 1.0, 0.0);
glBegin(GL_POLYGON);
  glColor3f(1.0, 0.0, 0.0);
  glVertex3f(0.0, 0.0, 0.0);
  glColor3f(0.0, 1.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
```

```
  glColor3f(0.0, 0.0, 1.0);
  glVertex3f(0.0, 1.0, 0.0);
glEnd();
glFlush();
}
```

7. Zadania

   (a) Przeczytać dokumentację funkcji `glBegin` i przetestować wszystkie możliwe jej parametry tworząc obiekty płaskie.

   (b) Przetestować zmianę parametrów funkcji `gluPerspective` i `gluLookAt`, aż do osiągnięcia intuicji w kostruowaniu sceny (tzn. nabyciu umiejętności specyfikowania wzajemnego położenia obserwatora i sceny od razu poprawnie)

   (c) Zdefiniować czworościan (każda ściana w innym kolorze, jeden z wierzchołków w punkcie (0,0,0)) i obejrzeć go z 6 pozycji, wzdłuź osi współrzędnych układu.