Laboratorium Grafiki Komputerowej i Animacji

Ćwiczenie IV

Biblioteka OpenGL - transformacje przestrzenne obiektów

Sławomir Samolej

Rzeszów, 1999

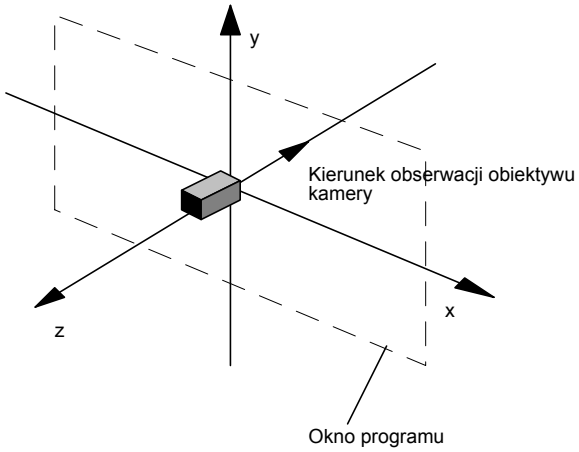
1. Wprowadzenie

Podstawowym zagadnieniem dotyczącym tworzenia scen graficznych jest umiejętność rozmieszczenia i orientowania wyświetlanych obiektów oraz definiowania wzajemnych zależności pomiędzy nimi. W standardzie OpenGL budowanie wzajemnych odniesień pomiędzy elementami sceny i obserwatorem polega na zdefiniowaniu zestawu transformacji matematycznych. Rolą transformacji jest przeniesienie trójwymiarowych definicji obiektów na płaski ekran monitora, a także umożliwienie programiście obracania, przemieszczania, oraz skalowania elementów animacji. W OpenGL nigdy nie dokonuje się bezpośredniej transformacji dotyczącej danego obiektu na nim samym. W rzeczywistości transformacje wykonują modyfikację układu współrzędnych, z którym dany obiekt jest związany. Jeśli przykładowo transformacja realizuje rotację (obrót) danego układu współrzędnych, to wszystkie obiekty przywiązane do niego automatycznie zmieniają swoje położenie zgodnie z nowymi współrzędnymi układu.

Opracowanie prezentuje interpretację i sposób realizacji podstawowych transformacji służących do manipulowania obiektami graficznymi na scenie. Omówione zostaną podstawowe funkcje standardu OpenGL pozwalające sytuować obiekty na scenie oraz umożliwiające zdefiniowanie sposobu ich obserwacji. Dołączona do opracowania dyskietka zawiera projekty gotowych aplikacji wykorzystywanych w opracowaniu jako przykłady ilustrujące kolejne aspekty omawianego zagadnienia.

2. Obserwacja, modelowanie, projekcja

Każdy zdefiniowany obiekt zanim pojawi się na ekranie monitora poddawany jest trzem podstawowym transformacjom: obserwacji (ang. viewing), modelowaniu (ang. modeling), projekcji (ang. projection). Obserwacja definiuje punkt w przestrzeni, w którym umieszcza się kamerę rejestrującą scenę. Domyślnie w OpenGL początkowym punktem obserwacji sceny jest środek bazowego układu współrzędnych. Bazowy układ współrzędnych OpenGL jest tak ustawiony, że środek układu znajduje się w środku okna, oś x wskazuje przesunięcie poziome, oś y - pionowe, natomiast oś z skierowana jest w kierunku obserwatora (rys 2.1)



Rys 2.1 Początkowa orientacja i kierunek obserwacji sceny w OpenGL

W chwili rozpoczęcia programu obiektyw kamery znajduje się w punkcie o współrzędnych (0,0,0) wyjściowego układu współrzędnych, a kierunek patrzenia obiektywu jest zgodny z ujemnym zwrotem osi z. Istnieje możliwość zdefiniowania położenia i kierunku patrzenia

kamery. Regułą jest, że transformacje przestrzenne określające położenie punktu obserwacji sceny wykonywane są jako pierwsze (zanim dokona się rozmieszczenia obiektów).

Modelowanie jest transformacją umożliwiającą manipulowanie tworzonymi obiektami należącymi do sceny. Transformacja ta pozwala na przesuwanie (translację) elementów sceny, ich obracanie (rotację) oraz przeskalowywanie ich rozmiarów (skalowanie). Końcowy wygląd sceny w głównej mierze zależy od kolejności wykonywania transformacji przestrzennych, ponieważ wykonanie kolejno przesunięcia i obrotu nie jest jednoznaczne z wykonaniem obrotu, a po nim przesunięcia (zagadnienie to zostanie zaprezentowane na przykładzie w dalszej części opracowania).

W rzeczywistości modelowanie i obserwacja, z punktu widzenia matematycznego i biblioteki OpenGL, są jednym typem transformacji. Prezentowane są one osobno dla uproszczenia rozumienia budowania scen graficznych. Dla wyjaśnienia można powiedzieć, że nie ma różnicy pomiędzy przesuwaniem jakiegoś obiektu w pewnym kierunku w danym układzie współrzędnych, a opisaniem tego samego zjawiska poprzez oddalanie od nieruchomego obiektu układu współrzędnych. Stąd w OpenGL, gdy dokonuje się transformacji przestrzennych obiektów mówi się o modyfikowaniu transformacji obserwacji-modelowania (ang. modelview transformation).

Projekcja określa fragment w przestrzeni, który obserwowany jest przez kamerę, oraz sposób odzwierciedlania przestrzeni na ekranie. W OpenGL zdefiniowane są dwa typy odzwierciedlania przestrzeni: prostopadły (ang. orthographic) oraz perspektywiczny (ang. perspective). W projekcji prostopadłej wszystkie obiekty rysowane są jak w rzucie prostokątnym na okno i zawsze zachowują swoje zdefiniowane rozmiary. Ten typ projekcji wykorzystywany jest przede wszystkim do aplikacji CAD oraz w przypadku, gdy wyświetla się pojedynczy obiekt o rozmiarach zbliżonych do odległości kamery od obiektu. Projekcja perspektywiczna służy do tworzenia scen, które mają oddawać rzeczywistość. Obiekty o tym samym rozmiarze położone dalej są mniejsze niż te które znajdują się bliżej w stosunku do kamery, a linie równoległe zachowują się zgodnie z zasadami perspektywy. Projekcja wyznacza także wycinek przestrzeni, w którym znajduje się scena. Jeśli dany obiekt sceny znajduje się poza wyznaczonym fragmentem przestrzeni przestaje on być widoczny.

Dodatkowa transformacja wycinająca (ang. viewport transformation) określa sposób przeniesienia sceny bezpośrednio na okno programu. Umożliwia ona obserwowanie zadanego fragmentu obrazu z całego obrazu docierającego do obiektywu kamery.

Każda z transformacji w OpenGL opisywana jest matematycznie przy pomocy odpowiednio skonstruowanej macierzy. Wyświetlenie pojedynczego punktu na ekranie polega w rzeczywistości na pomnożeniu współrzędnych punktu opisanego w notacji jednorodnej przez kolejno macierz obserwacji-modelowania, macierz projekcji, macierz wycięcia. Standard OpenGL przewiduje możliwość zmieniania każdej z macierzy i uzyskiwania dzięki temu zdolności modyfikowania zarówno obiektów rozmieszczonych na scenie jak i sposobu obserwacji sceny.

3. Definiowanie transformacji w OpenGL

W bibliotece OpenGL przemieszczanie obiektów graficznych po scenie polega na kolejnych, następujących po sobie modyfikacjach macierzy obserwacji-modelowania. OpenGL umożliwia dokonywanie trzech rodzajów transformacji przestrzennych: przesunięcia (translacji), obrotu (rotacji) oraz skalowania. Przykładowo jeśli zamiarem naszym jest wyrysowanie sześcianu przesuniętego o dziesięć jednostek w górę od początku układu współrzędnych musimy dokonać następujących operacji:

- skonstruować macierz przesuwającą o 10 jednostek w górę

- pomnożyć macierz obserwacji-modelowania przez stworzoną macierz przesunięcia
- wyrysować sześcian.

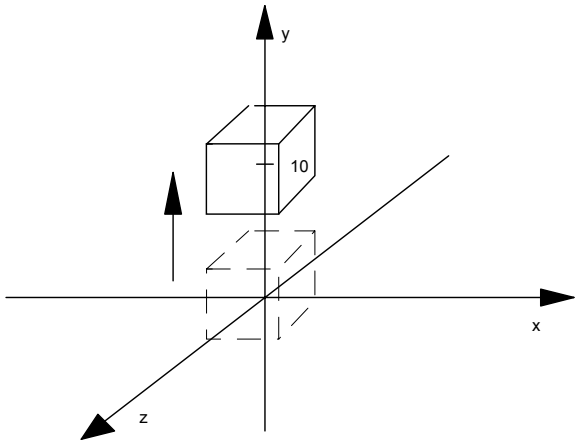
Fragment kodu prezentuje sposób rozwiązania postawionego problemu:

```
// Przesuń lokalny układ współrzędnych o 10 jednostek wzdłuż osi y
glTranslatef(0.0f, 10.0f, 0.0f);

// Rysuj sześcian
auxWireCube(10.0f)
```

Wyrysowaniem sześcianu może się zająć jedna z funkcji biblioteki AUX : auxWireCube(). Natomiast funkcja o prototypie:

```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

automatycznie generuje odpowiednią macierz przesunięcia i mnoży ją przez macierz obserwacji-modelowania. Jak łatwo zauważyć parametry wywołania funkcji x, y i z definiują współrzędne wektora translacji. Zamodelowaną transformację przestrzenną prezentuje rysunek 3.1.



Rys 3.1 Sześcian przesunięty o 10 jednostek w górę

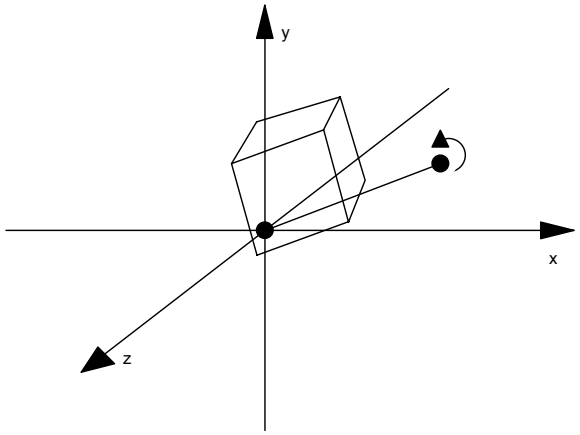
W celu dokonania obrotu danego obiektu wokół zdefiniowanej osi o określony kąt, należy utworzyć macierz rotacji, pomnożyć ją przez macierz obserwacji-modelowania i umieścić obiekt na scenie. Podobnie jak w przypadku przesunięcia, standard OpenGL definiuje pojedynczą funkcję, która automatycznie dokonuje opisaną transformację:

```
void glRotatef(GLfloat angle,GLfloat x, GLfloat y, GLfloat z);
```

Parametr angle określa kąt obrotu w stopniach, natomiast parametry x, y i z definiują wektor wokół którego ma nastąpić obrót. Fragment kodu:

```
// Obróć lokalny układ współrzędnych o 90 stopni wokół wektora
// o współrzędnych 1,1,1
glRotatef(90.0f, 1.0f, 1.0f, 1.0f);

// Rysuj sześcian
auxWireCube(10.0f)
```

dokonuje obrotu o 90 stopni wokół wybranej, osi a następnie wyrysowania zadanego sześcianu (rys 3.2).



Rys 3.1 Sześcian obrócony o 90 stopni wokół wektora o współrzędnych [1,1,1].

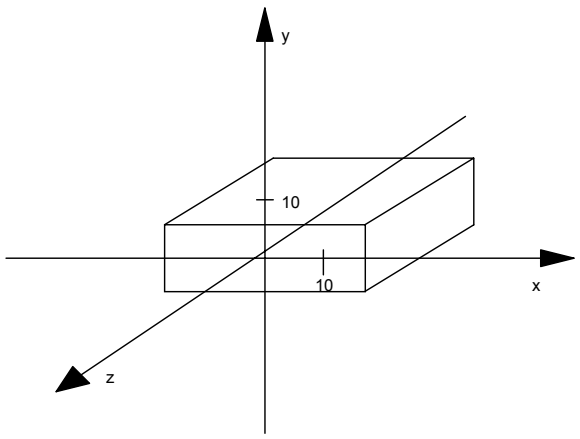
Skalowanie jest transformacją, która może zmienić rozmiar rysowanego obiektu poprzez zmianę odległości jednostkowych wzdłuż poszczególnych osi układu współrzędnych. Funkcja:

```
void glScalef(GLfloat z,GLfloat y,GLfloat z);
```

może służyć zarówno do "ściskania" jak i "rozciągania" obiektu wzdłuż osi układu współrzędnych. Fragment kodu:

```
// Rozciągnij dwukrotnie obiekt wzdłuż osi x i z
glScaleff(2.0f, 1.0f, 2.0f);

// Rysuj sześcian
auxWireCube(10.0f)
```

pokazuje sposób modyfikacji rozmiarów obiektów przy pomocy funkcji glScalef() (rys 3.3).

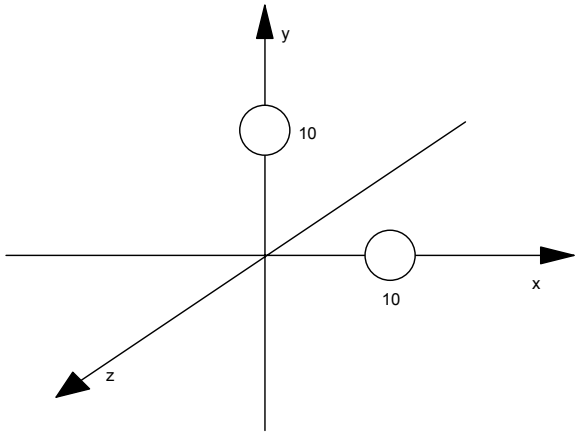


Rys 3.3 "Rozciągnięty" sześcian

4. Składanie transformacji

W standardzie OpenGL kolejne wywoływanie funkcji dokonujących transformacji przestrzennych (mnożących macierz obserwacji-modelowania przez macierz transformacji) powoduje kumulowanie się przekształceń. Dokonanie kolejnej transformacji polega na utworzeniu nowej macierzy przekształcenia i pomnożeniu bieżącej macierzy obserwacji-modelowania przez nowo powstałą.

Przeanalizujmy następujący przykład. Chcemy wyrysować dwie kule: jedną przesuniętą o 10 jednostek wzdłuż osi y, drugą przesuniętą o 10 jednostek wzdłuż osi x względem początku bazowego układu współrzędnych (rys. 4.1).

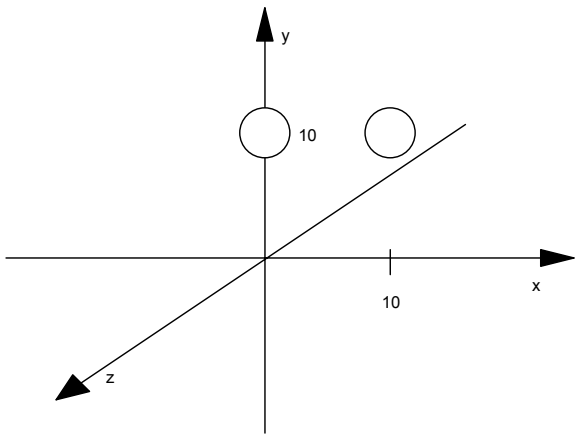


Rys. 4.1 Planowane położenie obiektów na scenie.

Najczęściej próbuje się wtedy tworzyć kod podobny do następującego:

```
// Przesuń się o 10 jednostek wzdłuż osi y
glTranslatef(0.0f, 10.0f, 0.0f);

// Rysuj pierwszą sferę
auxSoildSphere(1.0f);

// Przesuń się o 10 jednostek wzdłuż osi x
glTranslatef(10.0f, 0.0f, 0.0f);

// Rysuj drugą sferę
auxSoildSphere(1.0f);
```

Ponieważ jednak kolejne przekształcenia ulegają kumulacji efekt otrzymany w programie jest podobny do rysunku 4.2.



Rys. 4.2 Otrzymane położenie obiektów na scenie.

Łatwo zauważyć, że położenie drugiej kuli wynika ze złożenia dwu transformacji: przesunięcia o 10 jednostek w górę a następnie przesunięcie o 10 jednostek w prawo. Aby umieścić kulę w zadanym położeniu można by było dołożyć trzecie przesunięcie, tym razem o 10 jednostek w dół. W przypadku konstruowania bardziej złożonych scen taka próba manipulowania przekształceniami staje się zagadnieniem bardzo trudnym. Znacznie prostszym sposobem rozwiązania postawionego problemu wydaje się "zmuszenie" OpenGL do powrócenia do początkowego punktu obliczania transformacji po narysowaniu pierwszej kuli a następnie wyznaczenie nowego położenia drugiej kuli i narysowania jej. Z punktu widzenia matematycznego taki "powrót" do początku układu współrzędnych jest równoznaczny z załadowaniem macierzy jednostkowej jako macierzy obserwacji-modelowania. Fragment kodu:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

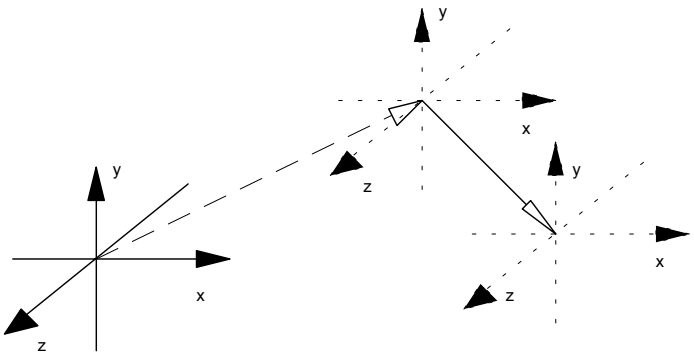
ładuje do macierzy obserwacji-modelowania macierz jednostkową. Pierwsza linia zaprezentowanego kodu informuje system OpenGL, że dokonana modyfikacja dotyczyć będzie macierzy obserwacji-modelowania. Funkcja glLoadIdentity() automatycznie ładuje do wyznaczonej macierzy macierz jednostkową. Skomentowania wymaga funkcja glMatrixMode(). Funkcja ta ustala, która z trzech podstawowych macierzy w systemie będzie aktualnie modyfikowana. Jeśli funkcja wywołana jest z parametrem **GL_MODELVIEW** to kolejne modyfikacje dotyczyć będą macierzy obserwacji-modelowania. Jeśli parametrem funkcji jest **GL_PROJECTION** to następne transformacje dotyczyć będą macierzy projekcji. Parametr **GL_TEXTURE** wskazuje, że modyfikacjom ulegać będzie macierz teksturowania (teksturowanie omówione zostanie w kolejnych opracowaniach). Do momentu kolejnego wywołania funkcji glMatrixMode() z innym parametrem każda modyfikacja będzie dotyczyć aktualnie ustalonej macierzy. Podany poniżej fragment kodu wykonuje prawidłowo postawione zadanie wyrysowania kul:

```
// Ustaw macierz obserwacji-modelowania jako bieżącą
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// Przesuń się o 10 jednostek wzdłuż osi y
glTranslatef(0.0f, 10.0f, 0.0f);

// Rysuj pierwszą sferę
auxSoildSphere(1.0f);

// Ustaw macierz obserwacji-modelowania w początkowy stan
glLoadIdentity();

// Przesuń się o 10 jednostek wzdłuż osi x
glTranslatef(10.0f, 0.0f, 0.0f);

// Rysuj drugą sferę
auxSoildSphere(1.0f);
```

Istotne znaczenie w budowaniu scen graficznych ma umiejętność interpretowania przekształceń przestrzennych. Są dwa sposoby interpretacji przekształceń przestrzennych w standardzie OpenGL. W pierwszym z nich wszystkie przekształcenia odbywają się w jednym globalnym układzie współrzędnych (układzie bazowym). Sposób drugi pozwala

interpretować przekształcenia jako kolejne transformacje dokonywane na lokalnych układach współrzędnych. Na początku omówione zostanie drugie podejście:

Jeśli w OpenGL dokonywane jest jakiekolwiek przekształcenie to można potraktować je jako kolejną operację przestrzenną na lokalnym układzie współrzędnych. Na początku wykonywania programu lokalnym układem współrzędnych jest układ bazowy. Dokonanie translacji (lub jakiekolwiek innej transformacji) układu o określony wektor w opisywanym podejściu można potraktować jako przesunięcie pewnego lokalnego układu współrzędnych (początkowo pokrywającego się z układem bazowym) o zadany wektor (rys 4.3).



Rys. 4.3 Interpretacja przekształceń przestrzennych
jako ciąg transformacji lokalnego układu współrzędnych

Kolejne przekształcenia należy wtedy interpretować jako transformacje lokalnego układu współrzędnych do następnego położenia itd. Takie rozumienie przekształceń przestrzennych przekłada się bezpośrednio na kolejne wywoływania funkcji OpenGL. Przykładowo fragment kodu:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glTranslatef(0.0f, 0.0f, -20.0f);
glRotatef(-30.0f, 0.0f, 0.0f, 1.0f);
glTranslatef(10.0f, 0.0f, 0.0f);
auxWireCube(10.0f)
```

powoduje, że lokalny układ współrzędnych początkowo przesuwany jest o 20 jednostek w ujemnym kierunku osi z, następnie obracany o 30 stopni wokół nowej osi z, potem przesuwany o 10 jednostek wzdłuż nowej osi x. Na końcu rysowany jest sześcian.

Drugim podejściem do interpretowania położenia obiektów na scenie jest "obserwowanie" ich z punktu widzenia bazowego układu współrzędnych. Najprostszym sposobem interpretacji położenia obiektów jest wtedy odczytywanie ciągu dokonywanych transformacji od końca. To znaczy podany wcześniej kod z punktu widzenia centralnego układu współrzędnych można odczytać jako kolejno: przesunięcie obiektu o 10 jednostek wzdłuż bazowej osi x, obrót przesuniętego obiektu wokół bazowej osi z, a następnie przesunięcie o -20 jednostek wzdłuż bazowej osi z.

Podczas budowania scen graficznych operacja powracania do bazowego układu współrzędnych może być kłopotliwa. Wygodniejszą operacją jest możliwość zapamiętania chwilowego położenia lokalnego układu współrzędnych. W tym celu system OpenGL posiada wbudowany stos przeznaczony do przechowywania stanów macierzy obserwacji-modelowania. Zasada działania stosu macierzy jest identyczna jak zasada działania każdej struktury danych typu stos. Dostępne są dwie operacje: składanie na stos i pobieranie ze

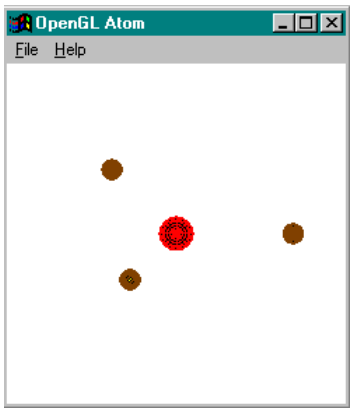
stosu. Z punktu widzenia OpenGL operację złożenia na stos macierzy obserwacji-modelowania można potraktować jako zapamiętanie chwilowego położenia lokalnego układu współrzędnych, a operację odczytu ze stosu jako załadowanie do macierzy obserwacji-modelowania ostatnio zapamiętanego położenia lokalnego układu współrzędnych. Do składania macierzy na stos służy funkcja:

```
glPushMatrix();
```

natomiast do ściągania ze stosu:

```
glPopMatrix().
```

Poniżej zaprezentowany zostanie fragment programu ATOM, wykorzystującego efekt składania transformacji do animacji modelu atomu. Kolejne klatki animacji obliczane są w obsłudze komunikatu Windows WM_TIMER. Jedną z klatek animacji prezentuje rysunek 4.4.



Rys 4.4 Okno programu ATOM.

```
// Funkcja rysująca scenę
void RenderScene(void)
      {
      // Kąt obrotu wokół jądra atomu
      static float fElect1 = 0.0f;

      // Wyczyść okno bieżącym kolorem tła
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      // Resetuj macierz obserwacji-modelowania
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();


      // Przesuń scenę 100 jednostek od kamery
      // (początkowa transformacja obserwacji)
      glTranslatef(0.0f, 0.0f, -100.0f);

      // Czerwone jądro atomu
      glRGB(255, 0, 0);
      auxWireSphere(10.0);

      // Żółte elektrony
      glRGB(255,255,0);

      // Pierwsza orbita elektronu
      // Zachowaj położenie lokalnego układu współrzędnych
```

```
glPushMatrix();

// Obróć się o wyznaczony kąt obrotu
glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

// Przesuń na orbitę
glTranslatef(90.0f, 0.0f, 0.0f);

// Wyrysuj elektron
auxWireSphere(6.0);

// Odzyskaj zachowane położenie lokalnego układu współrzędnych
glPopMatrix();

// Obliczenie orbity drugiego elektronu
glPushMatrix();
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glRotatef(fElect1, 0.0f, 1.0f, 0.0f);
glTranslatef(-70.0f, 0.0f, 0.0f);
auxWireSphere(6.0);

glPopMatrix();

// Obliczenie orbity trzeciego elektronu
glPushMatrix();
glRotatef(360.0f-45.0f,0.0f, 0.0f, 1.0f);
glRotatef(fElect1, 0.0f, 1.0f, 0.0f);
glTranslatef(0.0f, 0.0f, 60.0f);
auxWireSphere(6.0);
glPopMatrix();

// Zmodyfikuj poprzedni kąt obrotu elektronu
fElect1 += 10.0f;
if(fElect1 > 360.0f)
      fElect1 = 0.0f;

glFlush();
}
```

Sposób animacji pojedynczego elektronu realizowany jest w następujący sposób:
- zachowywane jest położenie lokalnego układu współrzędnych:

```
glPushMatrix();
```

- lokalny układ współrzędnych obracany jest wokół zadanego wektora:

```
glRotatef(fElect1, 0.0f, 1.0f, 0.0f);
```

- nowy lokalny układ współrzędnych jest następnie przesuwany o zadany wektor:

```
glTranslatef(90.0f, 0.0f, 0.0f);
```

- następnie wyrysowywany jest elektron:

```
auxWireSphere(6.0);
```

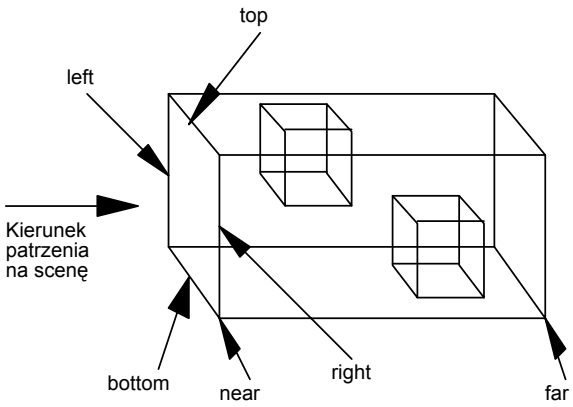
- oraz do macierzy obserwacji-modelowania załadowywany jest lokalny układ współrzędnych, który był usytuowany w jądrze atomu:

```
glPopMatrix();
```

5. Definiowanie projekcji

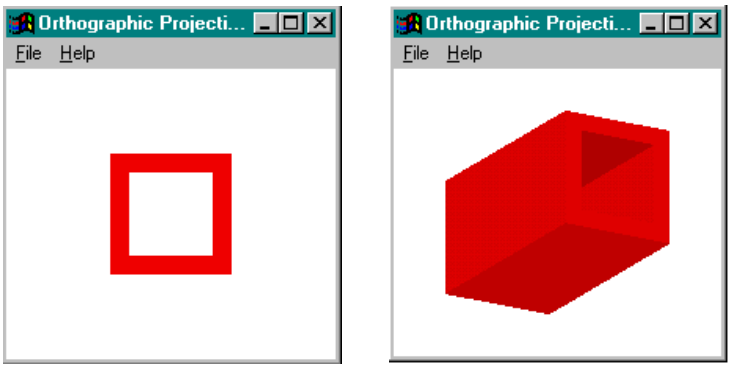
Macierz projekcji określa kształt i rozmiar obszaru przestrzeni, w którym obserwowana jest scena graficzna. Jak już wspomniano w OpenGL zdefiniować dwa typy projekcji: prostopadłą i perspektywiczną.

W projekcji prostopadłej (ang. orthographic projection) obserwowany wycinek przestrzeni ma kształt prostopadłościanu (rys 5.1).



Rys. 5.1 Opis projekcji prostopadłej OpenGL

Obiekty znajdujące się w obserwowanym wycinku przestrzeni odtwarzane są na ekranie komputera w rzucie prostopadłym. W programie ORTHO do wyświetlenia obiektu w postaci wydrążonego prostopadłościanu posłużono się projekcją prostopadłą (rys 5.2).



Rys 5.2 Wyświetlanie obiektów w projekcji prostopadłej OpenGL

Sposób wyświetlania sceny ściśle związany jest z proporcjami okna programu stąd transformacja określająca projekcję znajduje się zwykle we fragmentach kodu odpowiedzialnych za dostosowywanie obrazu do rozmiarów okna programu (funkcja ChangeSize() wywoływana pod wpływem komunikatu WM_SIZE). W programie ORTHO fragment kodu definiującego transformację projekcji ma postać:

```
void ChangeSize(GLsizei w, GLsizei h)
      {
      GLfloat nRange = 120.0f;
      // Zapobiegaj dzieleniu przez 0
      if(h == 0)
            h = 1;

      // Ustaw fragment obrazu docierającego do "obiektywu"
      // ,który ma być wyświetlony w oknie
      glViewport(0, 0, w, h);

      // Resetuj macierz obserwacji-modelowania
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();

      // Ustal obserwowany w projekcji prostopadłej fragment przestrzeni
      if (w <= h)
            glOrtho (   -nRange, nRange,
                        -nRange*h/w, nRange*h/w,
                        -nRange*2.0f, nRange*2.0f);
      else
            glOrtho (   -nRange*w/h, nRange*w/h,
                         -nRange, nRange,
                         -nRange*2.0f, nRange*2.0f);

      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      }
```

Po każdej zmianie rozmiarów okna programu następuje przedefiniowanie macierzy projekcji. Funkcja o prototypie:

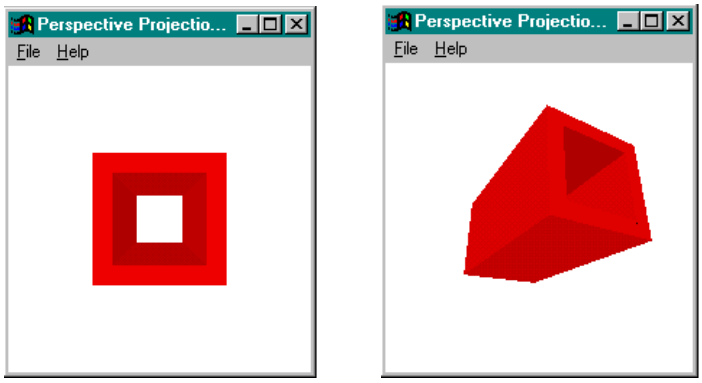
```
void glOrtho(     GLdouble left, GLdouble right, GLdouble bottom,
                  GLdouble top, GLdouble near, GLdouble far );
```

definiuje prostopadłościenny wycinek przestrzeni, który będzie widoczny przez obiektyw kamery obserwującej scenę. Poszczególne parametry funkcji odpowiadają wyszczególnionym na rysunku 5.1 krawędziom prostopadłościanu widzenia. Uzależnienie rozmiarów prostopadłościanu widzenia od proporcji wysokości do szerokości okna programu umożliwia zachowanie proporcji w kształtach rysowanych obiektów (Nadmierne "wydłużenie" okna programu nie powoduje przykładowo "spłaszczenia" figur, lub zachwiania ich proporcji wymiarowych w jakikolwiek inny sposób).
Linia kodu:

```
      glViewport(0, 0, w, h);
```

definiuje omawianą wcześniej transformację wycinającą. Kolejne parametry funkcji glViewport() wyznaczają prostokąt przez który obserwowana jest scena. Funkcja ta pozwala nakładać na obiektyw kamery obserwującej scenę pewnego rodzaju przesłonę ograniczającą widoczność. W omawianym przykładzie nie następuje żadne obcięcie obrazu, ponieważ transformacja wycinająca pozwala obserwować scenę przez okno równe rozmiarom okna programu.

Projekcja perspektywiczna definiuje obszar widzenia w postaci ściętego ostrosłupa. Wykorzystuje się ją do scen graficznych mających odzwierciedlać rzeczywistość (np. przedmioty znajdujące się dalej są mniejsze i linie proste zbiegają się w jednym punkcie). Przykładowy program PERSPECT wyświetla, podobnie jak program ORTHO, wydrążony prostopadłościan ale wykorzystuje do tego projekcję perspektywiczną (rys 5.3).



Rys 5.3 Wyświetlanie obiektów w projekcji perspektywicznej OpenGL

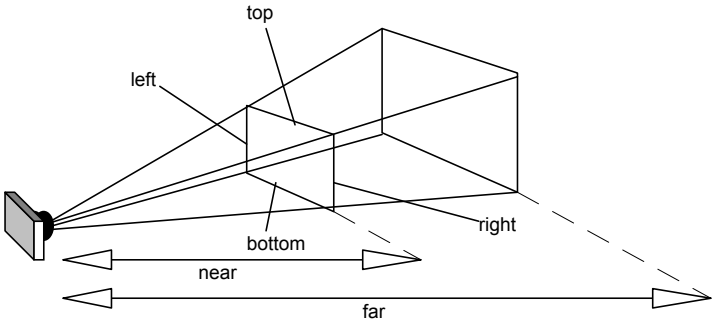
Fragment kodu definiującego projekcję perspektywiczną ma postać:

```
void ChangeSize(GLsizei w, GLsizei h)
      {
      GLfloat fAspect;

      // Prevent a divide by zero
      if(h == 0)
            h = 1;

      // Set Viewport to window dimensions
      glViewport(0, 0, w, h);

      fAspect = (GLfloat)w/(GLfloat)h;

      // Reset coordinate system
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();

      // Produce the perspective projection
      gluPerspective(60.0f, fAspect, 1.0, 400.0);

      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      }
```

Biblioteka OpenGL umożliwia zdefiniowanie ostrosłupa widzenia na dwa sposoby. W pierwszym przypadku do definicji wykorzystać można funkcję:

```
void glFrustum(   GLdouble left, GLdouble right, GLdouble bottom,
                  GLdouble top, GLdouble znear, GLdouble zfar );
```

Poszczególne parametry funkcji odpowiadają wielkościom zaznaczonym na rysunku 5.4. Funkcja nie jest zbyt często wykorzystywana do definicji transformacji projekcji, ponieważ trudno jest wyznaczyć jej parametry wywołania w sposób intuicyjny.

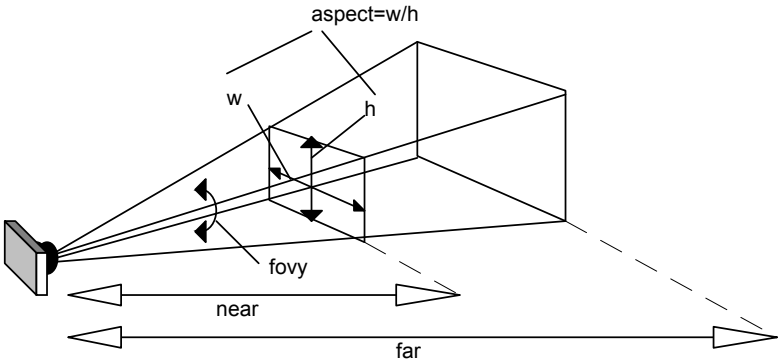


Rys 5.4 Projekcja perspektywiczna opisywana przez funkcję glFrustum()

Bardziej popularny sposób wyznaczenia transformacji projekcji perspektywicznej polega na wykorzystaniu funkcji wchodzącej w skład biblioteki GLU:

```
void gluPerspective(    GLdouble fovy, GLdouble aspect,
                        GLdouble zNear, GLdouble zFar );
```

Parametry funkcji odpowiadają wartościom wskazanym na rysunku 5.5.



Rys 5.5 Projekcja perspektywiczna opisywana przez funkcję gluPerspective()

Parametr aspect jest stosunkiem szerokości do wysokości, parametr fovy - kątem rozwarcia przy wierzchołku ostrosłupa widzenia, parametry znear i zfar określają odległości podstaw ściętego ostrosłupa od miejsca, w którym położona jest kamera obserwująca scenę.

6. Definiowanie własnych transformacji macierzowych

Standard OpenGL dopuszcza możliwość tworzenia własnych macierzy przekształceń jednorodnych. Dostępne są w tym celu dwie funkcje:

```
void glLoadMatrixd(const GLdouble *m );

void glMultMatrixd( const GLdouble *m );
```

Pierwsza z nich umożliwia załadowanie zdefiniowanej przez użytkownika macierzy do macierzy obserwacji-modelowania. Druga zaś umożliwia pomnożenie bieżącej macierzy obserwacji-modelowania przez zdefiniowaną macierz. Fragment kodu:

```
GLdouble m[]={    1.0, 0.0, 0.0, 0.0,
                  0.0, 1.0, 0.0, 0.0,
```

```
                0.0, 0.0, 1.0, 0.0,
                0.0, 0.0, 0.0, 1.0,
            };

glMatrixMode(GL_MODELVIEW);
glMultMatrix(m);
```

powoduje przemnożenie bieżącej macierzy obserwacji-modelowania przez zdefiniowaną przez użytkownika macierz m.

Bibliografia:

[1] R. S. Wright Jr., M. Sweet, *OpenGL Superbible*, The Waite Group Press, 1996
[2] *OpenGL Programming Guide*, Addison-Wessley, 1993
[3] E. Angel, *Interactive Computer Graphics*, Addison-Wesley, 1997
[4] R. Fosner, *OpenGL Programming for Windows and Windows NT*, Addison-Wesley, 1997
[5] R. Leniowski, *Wykłady z przedmiotu Grafika Komputerowa i Animacja*
[6] *Guide to OpenGL® on Windows® From Silicon Graphics®*, 1997