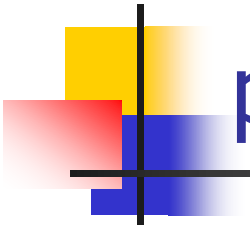
Wykład 3:

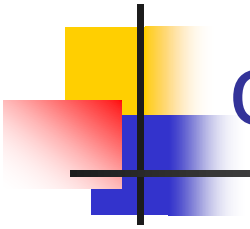
Biblioteka OpenGL elementy zaawansowane



przekształcenia geometryczne:

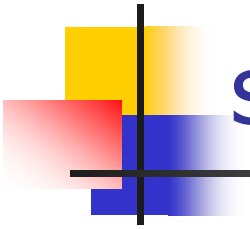
Biblioteka OpenGL umożliwia przekształcenia geometryczne obiektów.
Do podstawowych przekształceń należą:
- obrót
- skalowanie
- przesunięcie (translacja).

W przypadku, gdy zestaw standardowych przekształceń okazuje się niewystarczający, biblioteka OpenGL udostępnią możliwość realizacji przekształceń za pomocą operacji macierzowych.



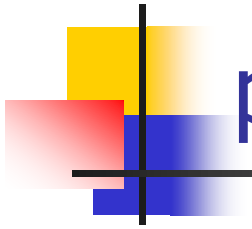
obrót:

Funkcje
void glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z)
void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z)
gdzie angle to kąt obrotu w stopniach, a x, y, z to współrzędne wektora określającego oś obrotu, określają obrót obiektu o kąt angle w kierunku przeciwnym do ruchu wskazówek zegara względem prostej wyznaczonej przez wektor [x, y, z] zaczepiony w początku układu współrzędnych.



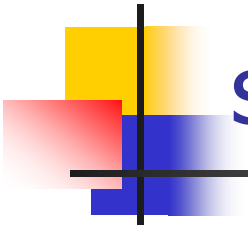
skalowanie:

Funkcje
void glScaled(GLdouble x, GLdouble y, GLdouble z)
void glScalef(GLfloat x, GLfloat y, GLfloat z)
void glScalex(GLfixed x, GLfixed y, GLfixed z)
gdzie x, y, z są współczynnikami skalowania względem kolejnych osi układu współrzędnych, określają operację skalowania obiektu zgodnie z podanymi współczynnikami skalowania dla poszczególnych osi.



przesunięcie:

Funkcje
void glTranslated(GLdouble x, GLdouble y, GLdouble z)
void glTranslatef(GLfloat x, GLfloat y, GLfloat z)
void glTranslatex(GLfixed x, GLfixed y, GLfixed z)
gdzie x, y, z są współrzędnymi wektora przesunięcia, określają operację przesunięcia obiektu o podany wektor.

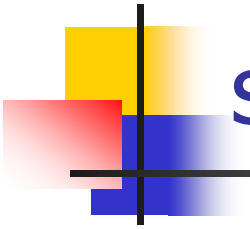


stosy macierzy:

Macierze w OpenGL opisujące przekształcenia, to **macierze 4 x 4**.

Macierze są przechowywane w trzech stosach macierzowych:

- **stos modelowania** - identyfikowany przez stałą **GL_MODELVIEW** przekształcenia opisują przejście między układem, w którym są podawane współrzędne punktów i wektorów, a układem, w którym następuje rzutowanie perspektywiczne lub równoległe
- **stos rzutowania** - identyfikowany przez stałą **GL_PROJECTION** przekształcenie reprezentowane przez macierz na tym stosie opisuje rzut perspektywiczny lub równoległy
- **stos tekstur** - identyfikowany przez stałą **GL_TEXTURE** przekształcenia na tym stosie opisują odwzorowanie tekstury

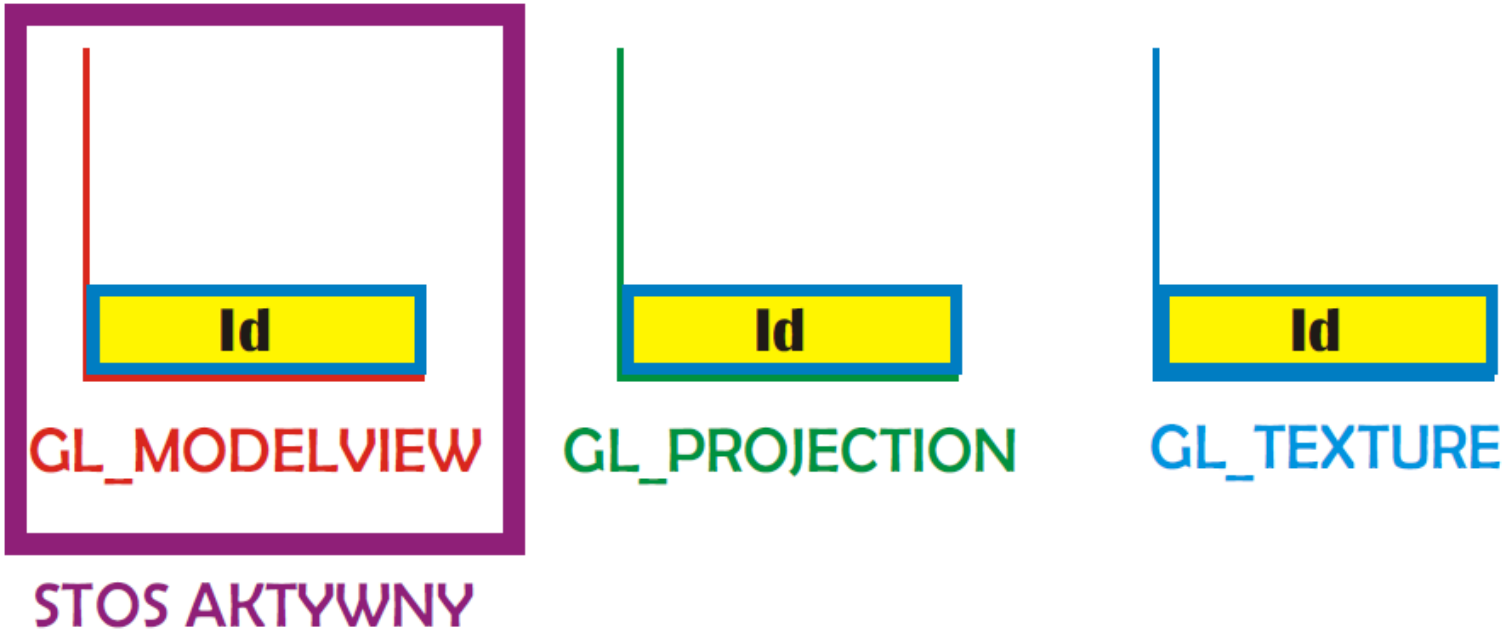


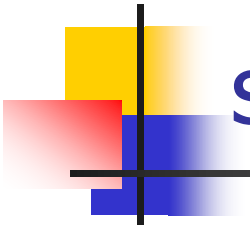
stosy macierzy:

Po zainicjowaniu biblioteki (**glutInit(...)**) na każdym z tych stosów powinna znaleźć się macierz jednostkowa **Id**

$$Id = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

stosy macierzy:





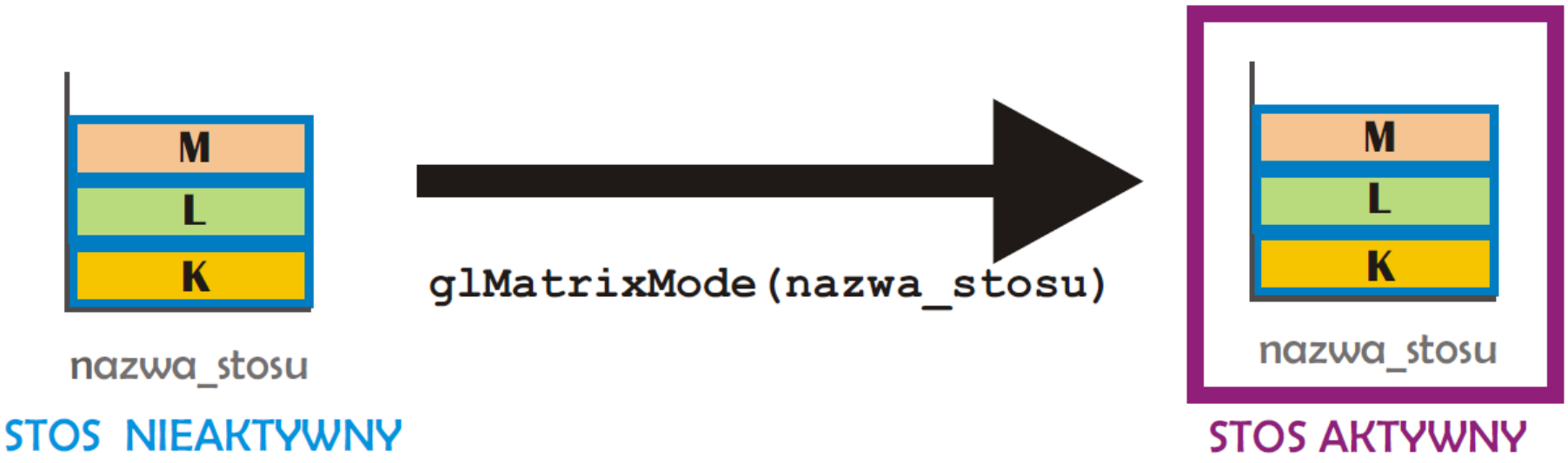
stosy macierzy:

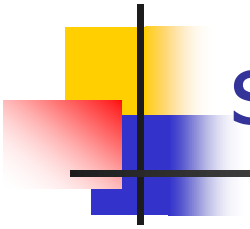
W danym momencie dostępny jest tylko jeden stos, tzw. **stos aktywny.** Pozostałe stosy nie są dostępne. Po zainicjowaniu biblioteki aktywny jest stos modelowania.

Funkcja
void glMatrixMode(GLenum nazwa)
gdzie nazwa jest jedną ze stałych GL_MODELVIEW, GL_PROJECTION lub GL_TEXTURE, określa, który ze stosów stanie się stosem aktywnym.

Funkcja
void glLoadIdentity()
powoduje zastąpienie wierzchołka aktywnego stosu macierzą Id

stosy macierzy:



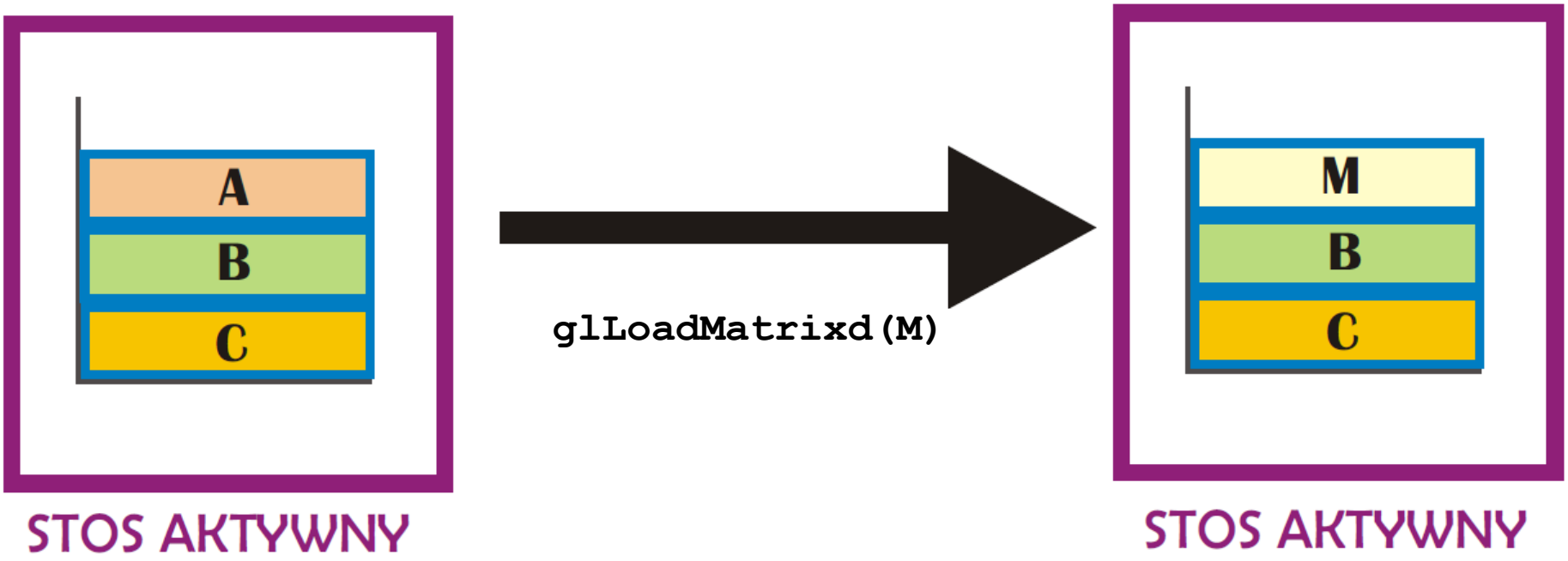


stosy macierzy:

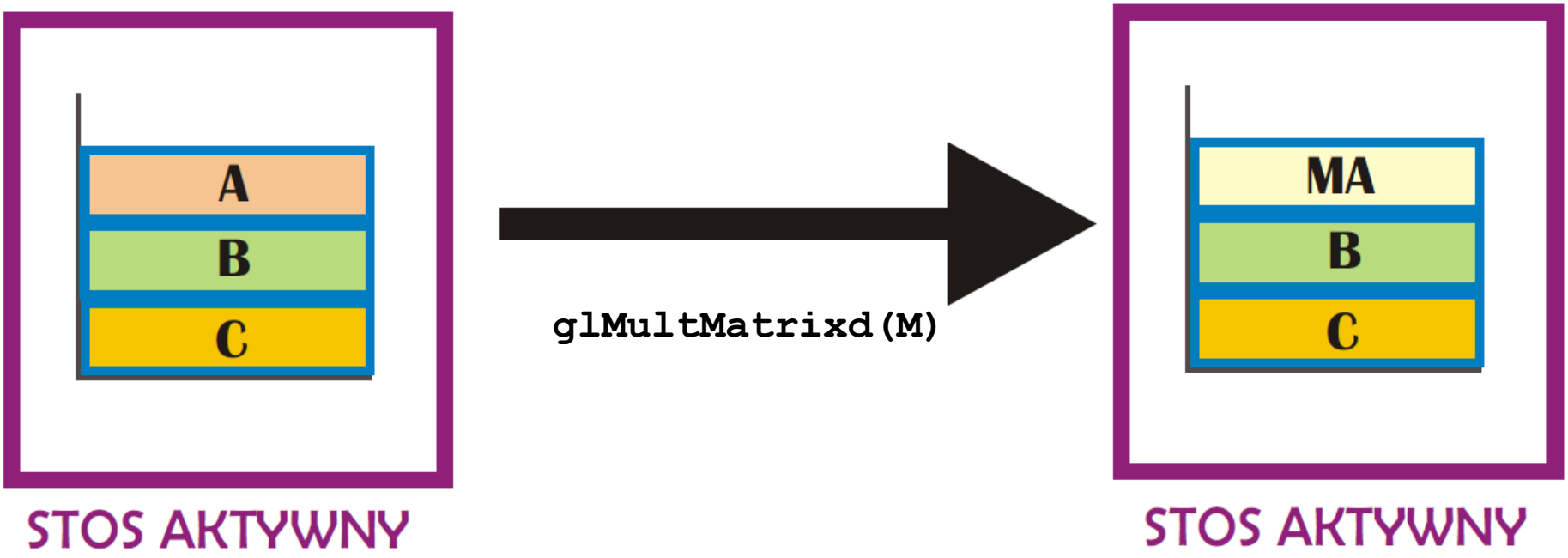
Funkcje
void glLoadMatrixd(const GLdouble * M)
void glLoadMatrixf(const GLfloat * M)
gdzie M jest macierzą zdefiniowaną przez programistę, powodują **zastąpienie wierzchołka** aktywnego stosu **macierzą M**.

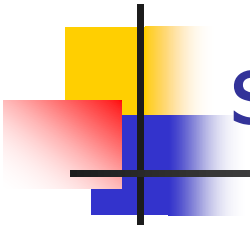
Funkcje
void glMultMatrixd(const GLdouble * M)
void glMultMatrixf(const GLfloat * M)
gdzie M jest macierzą zdefiniowaną przez programistę, powodują **zastąpienie wierzchołka** aktywnego stosu lewostronnym **iloczynem macierzy M i macierzy aktualnie znajdującej się na szczycie** aktywnego stosu.

stosy macierzy:



stosy macierzy:

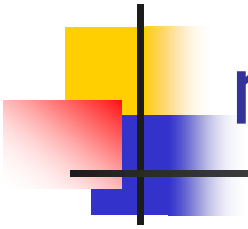




stosy macierzy:

Następujące funkcje powodują zastąpienie wierzchołka aktywnego stosu lewostronnym iloczynem odpowiedniej macierzy M (podanej jako parametr lub wygenerowanej automatycznie) i macierzy aktualnie znajdującej się na szczycie aktywnego stosu:

- glMultMatrixd, glMultMatrixf
- glRotated, glRotatef
- glScaled, glScalef, glScalex
- glTranslated, glTranslatef, glTranslatex
- glFrustum
- gluPerspective
- gluLookAt

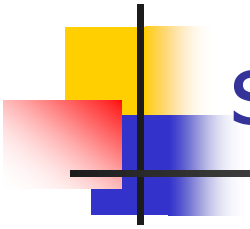


mnożenie macierzy:

Operacje w OpenGL związane z macierzami to:
- mnożenie macierzy przez macierz - w wyniku powstaje nowa macierz
- mnożenie macierzy przez wektor - w wyniku powstaje nowy wektor.

W bibliotece OpenGL mnożenia te wykonywane są **lewostronnie**, tzn. mnożenie danej macierzy M przez macierz N, będzie miało postać N*M (a nie M*N).
Podobnie dla wektora P i macierzy M, zostanie wykonane mnożenie P*M, a to znaczy, że wektor P, musi być **wektorem wierszowym**.

Aby operacje mnożenia macierzy i wektorów były właściwie wykonane, **macierze** też muszą być **transponowane** (tzn. wiersze muszą zostać zamienione na kolumny).



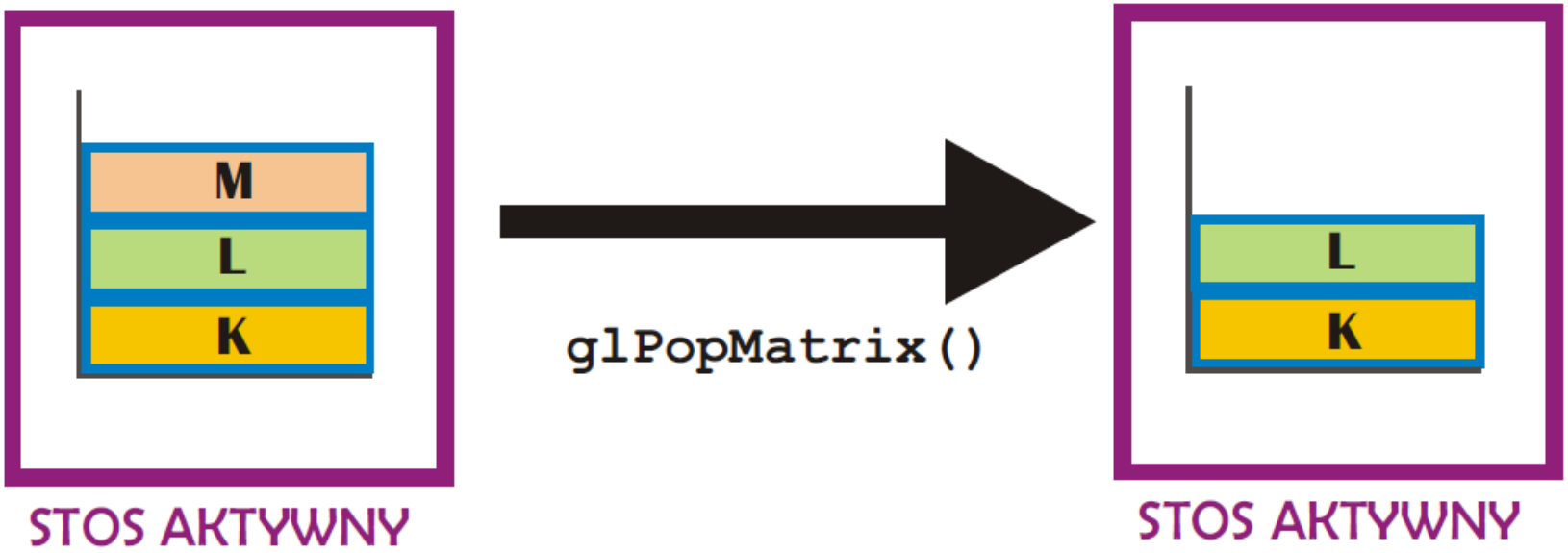
stosy macierzy:

Funkcja
void glPopMatrix()
usuwa macierz znajdującą się na wierzchołku aktywnego stosu.

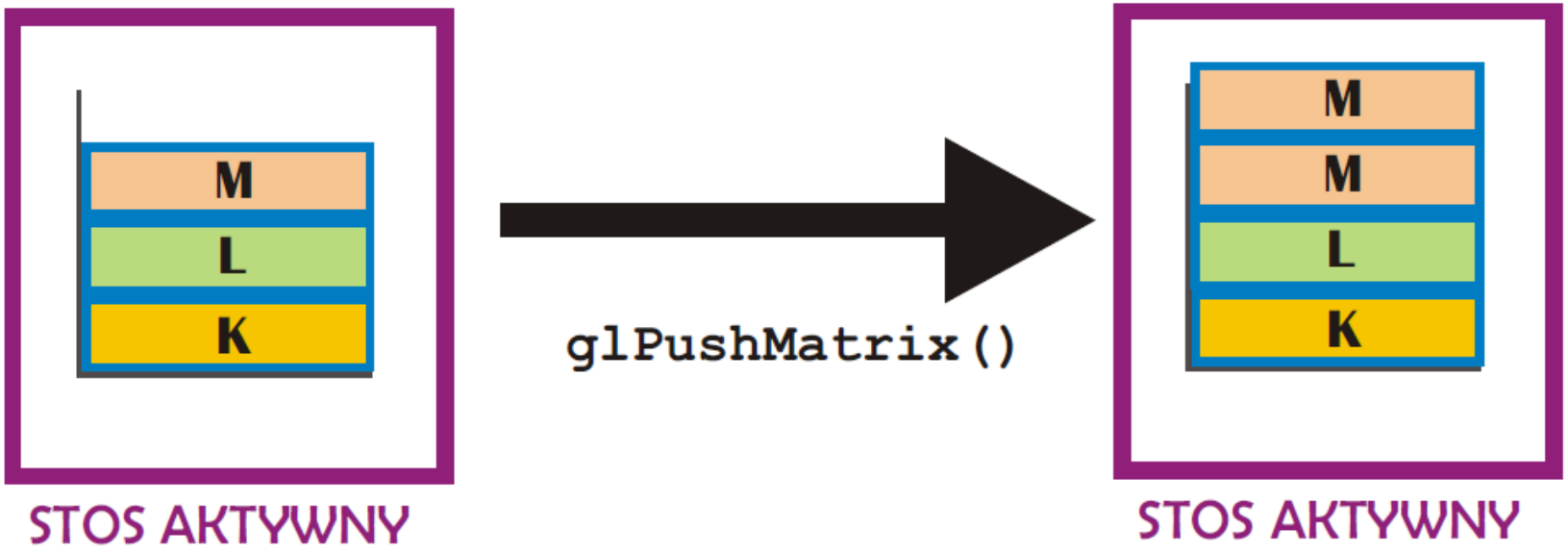
Funkcja
void glPushMatrix()
wykonuje kopię wierzchołka na aktywnym stosie.

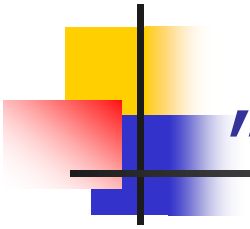
Nie ma innych funkcji umieszczających na stosie dowolną, zdefiniowaną przez programistę macierz.

stosy macierzy:



stosy macierzy:





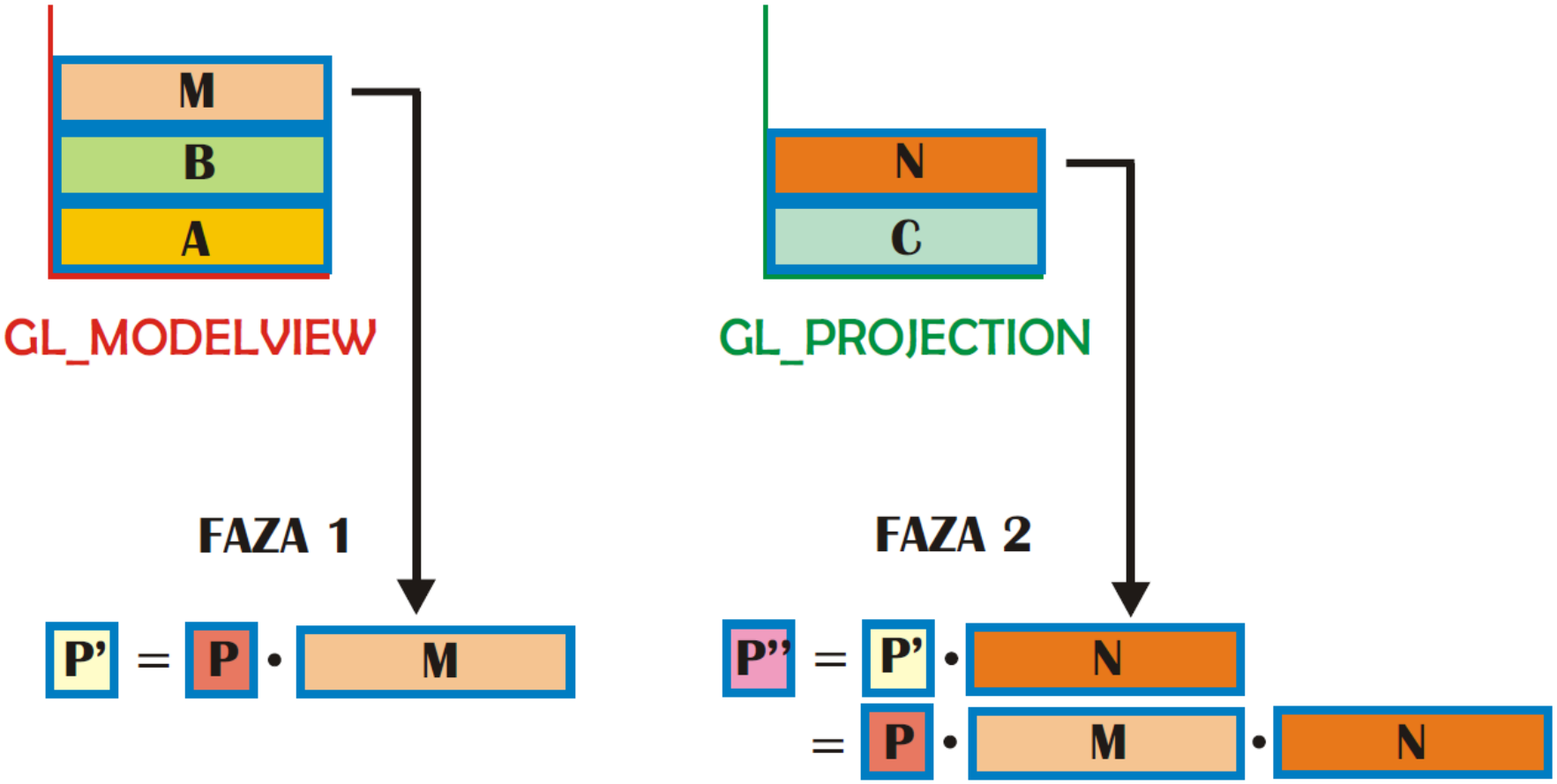
„przetwarzanie” punktu:

Na czym polega wykonanie funkcji glVertex3f(x, y, z) ?

Tak określony wierzchołek jest przechowywany w wektorze P = [x y z 1]. Niech na wierzchołku stosu modelowania znajduje się macierz M, a na wierzchołku stosu rzutowania jest macierz N. Wówczas:

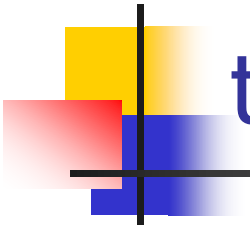
- wektor P jest mnożony przez wierzchołek stosu modelowania (FAZA 1), otrzymujemy wektor P’ = PM
- wektor P’ jest mnożony przez wierzchołek stosu rzutowania (FAZA 2), otrzymujemy wektor P” = P’N = (PM)N = P(MN)

„przetwarzanie” punktu:



Zmodyfikowana funkcja opisująca scenę:

```
void scene()
{
glClearColor(1.0, 1.0, 1.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST); //włączenie algorytmu zasłaniania
glMatrixMode(GL_PROJECTION); //aktywny stos rzutowania
glLoadIdentity(); //macierz jednostkowa inicjuje stos rzutowania
gluPerspective(60.0, 1.0, 0.1, 10.0);
gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glMatrixMode(GL_MODELVIEW); //aktywny stos modelowania
glLoadIdentity(); //macierz jednostkowa inicjuje stos modelowania
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(); //żądanie wykonania wszystkich określonych tutaj funkcji
}
```

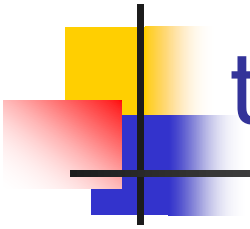


tablice wierzchołków:

Rysowanie brył (np. sześcianu) złożonych z wielu ścian (np. kwadratów) wymaga podania wiele razy współrzędnych wierzchołków (i ich kolorów) dla tych ścian.
Informacje o wierzchołkach możemy zawrzeć w tablicach (wierzchołków).
Przez użycie funkcji indeksujących te tablice będziemy mogli za pomocą jednego wywołania pobrać wszystkie dane dla określonego wierzchołka.

Konstrukcje definiujące tablice współrzędnych wierzchołków i ich kolorów są zgodne z zasadami deklarowania tablic w C++, np.:

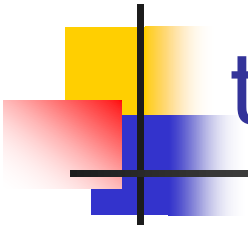
```
float wierzcholki[] = {0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0};
float kolory[] = {1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0};
```



tablice wierzchołków:

Funkcja
void glEnableClientState(GLenum array)
gdzie array jest jedną ze stałych GL_VERTEX_ARRAY (współrzędne wierzchołków), GL_COLOR_ARRAY (składowe kolorów) itd. włącza odpowiednią tablicę (wszystkie tablice są domyślnie nieaktywne), która będzie używana przy renderingu sceny.

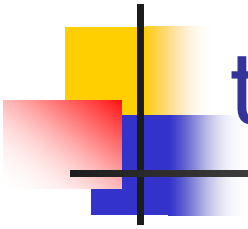
Funkcja
void glDisableClientState(GLenum array)
gdzie array jest jedną ze stałych GL_VERTEX_ARRAY (współrzędne wierzchołków), GL_COLOR_ARRAY (składowe kolorów) itd. wyłącza podaną tablicę.



tablice wierzchołków:

Funkcja
void glVertexPointer(GLint size, GLenum type, Glsizei offset, const void * pointer)
gdzie size określa ilość danych przypadającą na wierzchołek, type jest stałą określającą typ danych zawartych w tablicy (np. GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE), offset definiuje przesunięcie pomiędzy poszczególnymi elementami tablicy, a pointer oznacza wskaźnik na element tablicy, od którego mają być pobierane dane, definiuje dane dotyczące geometrii wierzchołków, czyli odpowiednie powiązanie przygotowanej w programie tablicy z danymi.

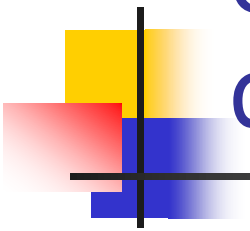
Przykład:
glVertexPointer(3, GL_FLOAT, 0, wierzcholki);



tablice wierzchołków:

Analogicznie funkcja
void glColorPointer(GLint size, GLenum type, Glsizei offset, const GLvoid * pointer)
gdzie size określa ilość danych przypadającą na wierzchołek, type jest stałą określającą typ danych zawartych w tablicy (np. GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE), offset definiuje przesunięcie pomiędzy poszczególnymi elementami tablicy, a pointer oznacza wskaźnik na element tablicy, od którego mają być pobierane dane, definiuje dane dotyczące kolorów wierzchołków, czyli odpowiednie powiązanie przygotowanej w programie tablicy z danymi.

Przykład:
glColorPointer(3, GL_FLOAT, 0, kolory);

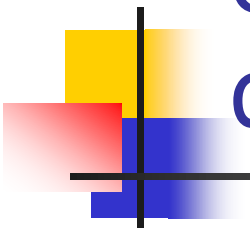


odwoływanie się do tablic wierzchołków:

Funkcja
void glArrayElement(GLint i)
gdzie i jest indeksem elementu, do którego się odwołujemy, umożliwia odwołanie się do konkretnego elementu i pobranie wszystkich danych (geometria, kolor itd.) o tym elemencie.

Przykład:

```
glBegin(GL_QUADS);
  glArrayElement(0);
  glArrayElement(1);
  glArrayElement(2);
  glArrayElement(3);
glEnd();
```



odwoływanie się do tablic wierzchołków:

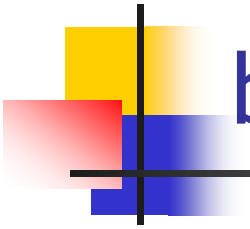
Funkcja
void glDrawArrays(GLenum mode, GLint first, GLsizei count)
gdzie mode przyjmuje takie same wartości, jak parametr mode funkcji glBegin, first to indeks wskazujący element tablic wierzchołków, od którego mamy wykorzystać kolejne elementy do rysowania, a parametr count określa ilość wykorzystywanych elementów, umożliwia odwołanie się do listy kolejnych elementów i pobranie wszystkich danych (geometria, kolor itd.) o tych elementach.

Przykład:
glDrawArrays(GL_QUADS, 0, 4);

odwoływanie się do tablic wierzchołków:

Funkcja
void glDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid * indices)
gdzie mode przyjmuje takie same wartości, jak parametr mode funkcji glBegin, count określa ilość indeksów znajdujących się w tablicy indeksów, do której wskaźnik zawiera parametr indices, a type jest typem danych tablicy indeksów (GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, GL_UNSIGNED_INT, indeksy numerowane są od 0), umożliwia odwołanie się do elementów i pobranie wszystkich ich danych (geometria, kolor itd.) zgodnie z tablicą indeksów tych elementów tworzących dany prymityw.

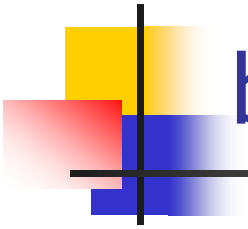
Przykład:
```
unsigned short indeksy[]={0, 1, 2, 3};
glDrawElements(GL_QUADS, 4, GL_UNSIGNED_SHORT, indeksy);
```



biblioteka GLUT - obsługa klawiatury:

Funkcja
void glutKeyboardFunc(void(*func)(unsigned char key, int x, int y))
gdzie func jest funkcją obsługi klawiatury mającą trzy parametry (key to kod ASCII ostatnio wciśniętego klawisza, x i y to współrzędne kursora myszki w chwili naciśnięcia klawisza klawiatury), włącza funkcję obsługi klawiatury odpowiadającą za obsługę znaków ASCII.

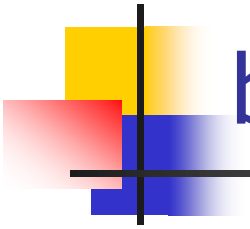
Funkcja obsługi klawiatury musi być wcześniej przygotowana.



biblioteka GLUT - obsługa klawiatury:

Przykład funkcji obsługi klawiatury odpowiadającej za obsługę znaków ASCII:

```
void klawiatura(unsigned char key, int x, int y)
{
    if( key == '+' )
        eyez -= 0.1;
    else

    if( key == '-' )
        eyez += 0.1;

    // odrysowanie okna
    Reshape(width, height);
}
```



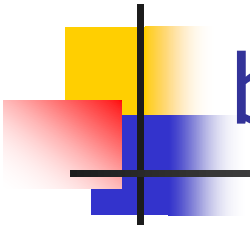
biblioteka GLUT - obsługa klawiatury:

Funkcja
void glutSpecialFunc(void(*func)(int key, int x, int y))
gdzie func jest funkcją obsługi klawiatury mającą trzy parametry (key to kod ostatnio wciśniętego klawisza specjalnego (klawisze kursora oraz przyciski funkcyjne), x i y to współrzędne kursora myszki w chwili naciśnięcia klawisza klawiatury), włącza funkcję obsługi klawiatury odpowiadającą za obsługę klawiszy specjalnych klawiatury.
Funkcja obsługi klawiatury musi być wcześniej przygotowana.
Kody klawiszy specjalnych reprezentowane są przez następujące stałe:
GLUT_KEY_F1, … , GLUT_KEY_F12, GLUT_KEY_LEFT, GLUT_KEY_UP, GLUT_KEY_RIGHT, GLUT_KEY_DOWN, GLUT_KEY_PAGE_UP, GLUT_KEY_PAGE_DOWN, GLUT_KEY_HOME, GLUT_KEY_END GLUT_KEY_INSERT

biblioteka GLUT - obsługa klawiatury:

Przykład funkcji obsługi klawiatury odpowiadającej za obsługę klawiszy specjalnych klawiatury:

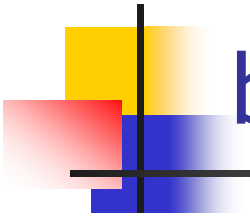
```
void specKlawisze(int key, int x, int y)
{
    switch(key)
    { case GLUT_KEY_LEFT:
        eyex += 0.1; break;
      case GLUT_KEY_UP:
        eyey -= 0.1; break;
      case GLUT_KEY_RIGHT:
        eyex -= 0.1; break;
      case GLUT_KEY_DOWN:
        eyey += 0.1; break;
    };
    Reshape(width, height);
}
```



biblioteka GLUT - obsługa myszy:

Funkcja
void glutMouseFunc(void(*func)(int button, int state, int x, int y))
gdzie func jest funkcją obsługi myszy mającą cztery parametry (button określa, który przycisk myszy został naciśnięty lub zwolniony (stałe GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON), state określa, czy przycisk myszy został naciśnięty (GLUT_UP), czy zwolniony (stała GLUT_DOWN), x i y to aktualne współrzędne kursora myszy), włącza funkcję obsługi myszy odpowiadającą za obsługę zdarzeń naciśnięcia i zwolnienia danego przycisku myszy.

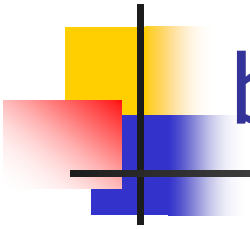
Funkcja obsługi myszy musi być wcześniej przygotowana.



biblioteka GLUT - obsługa myszy:

Przykład funkcji obsługi myszy odpowiadającej za obsługę zdarzeń naciśnięcia i zwolnienia danego przycisku myszy:

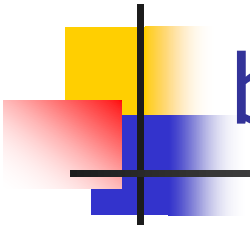
```
void myszPrzyciski(int button, int state, int x, int y)
{
    if( button == GLUT_LEFT_BUTTON )
    {
        button_state = state;

        if( state == GLUT_DOWN )
        {
            button_x = x;
            button_y = y;
        }
    }
}
```



biblioteka GLUT - obsługa myszy:

Funkcja
void glutMotionFunc(void(*func)(int x, int y))
gdzie func jest funkcją obsługi ruchu kursora myszy mającą dwa parametry (x i y to aktualne współrzędne kursora myszy), włącza funkcję obsługi ruchu kursora myszy odpowiadającą za obsługę zdarzeń ruchu kursora myszy przy wciśniętym danym przycisku myszy.

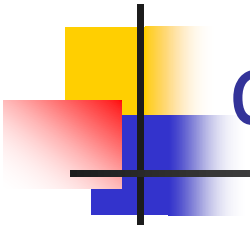
Funkcja obsługi ruchu kursora myszy musi być wcześniej przygotowana.



biblioteka GLUT - obsługa myszy:

Funkcja
void glutPassiveMotionFunc(void(*func)(int x, int y))
gdzie func jest funkcją obsługi ruchu kursora myszy mającą dwa parametry (x i y to aktualne współrzędne kursora myszy), włącza funkcję obsługi ruchu kursora myszy odpowiadającą za obsługę zdarzeń ruchu kursora myszy bez naciśnięcia jakiegokolwiek przycisku myszy

Funkcja obsługi ruchu kursora myszy musi być wcześniej przygotowana.

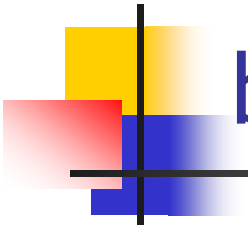


dodatkowe uwagi:

Wyłącznie danego rodzaju obsługi klawiatury lub myszy zachodzi przy wywołaniu którejkolwiek z pięciu omówionych funkcji z parametrem NULL, np. glutKeyboardFunc(NULL).

Włącznie (wyłączenie) danego rodzaju obsługi klawiatury lub myszy ma miejsce w ciele funkcji main.

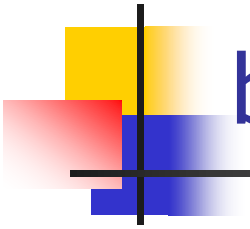
Przykłady:
glutKeyboardFunc(klawiatura);
glutSpecialFunc(specKlawisze);
glutMouseFunc(myszPrzyciski);



bryły 3D w bibliotece GLUT:

Rysowanie brył 3D za pomocą ciągów wywołań glVertex podanych między glBegin i glEnd jest dość uciążliwe.
W bibliotece GLUT mamy proste w użyciu funkcje rysujące wybrane bryły:
- sześcian, czworościan, ośmiościan, dwunastościan, dwudziestościan
- sfera (kula)
- stożek
- torus
- czajnik z Utah.

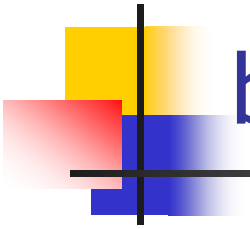
Biblioteka GLUT udostępnia obiekty 3D w wersji szkieletowej (Wire) oraz w wersji pełnej (Solid).



bryły 3D w bibliotece GLUT:

Funkcja
void glutWireCube(GLdouble size)
void glutSolidCube(GLdouble size)
rysuje sześcian o boku długości size i środku położonym w początku układu współrzędnych.

Funkcja
void glutWireTetrahedron() **void glutSolidTetrahedron()**
rysuje czworościan o środku położonym w początku układu współrzędnych.

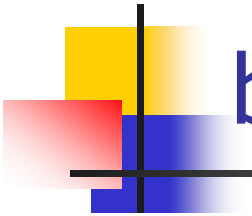


bryły 3D w bibliotece GLUT:

Funkcja
void glutWireOctahedron() **void glutSolidOctahedron()**
rysuje ośmiościan o środku położonym w początku układu współrzędnych.

Funkcja
void glutWireDodecahedron() **void glutSolidDodecahedron()**
rysuje dwunastościan o środku położonym w początku układu współrzędnych.

Funkcja
void glutWireIcosahedron() **void glutSolidIcosahedron()**
rysuje dwudziestościan o środku położonym w początku układu współrzędnych.



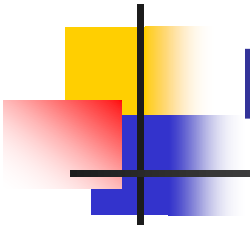
bryły 3D w bibliotece GLUT:

Funkcja
void glutWireSphere(GLdouble radius, GLint slices, GLint stacks)
void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks)
rysuje sferę o środku położonym w początku układu współrzędnych, promieniu radius; slices określa ilość „południków” a stacks ilość „równoleżników” (sfera zbudowana jest z czworokątów).
Funkcja
void glutWireCone(GLdouble base, GLdouble height,
GLint slices, GLint stacks)
void glutSolidCone(GLdouble base, GLdouble height,
GLint slices, GLint stacks)
rysuje stożek o środku podstawy położonym w początku układu współrzędnych i wierzchołku położonym na dodatniej półosi OZ; base jest promieniem podstawy, height wysokością stożka, slices określa ilość tworzących, a stacks ilość „warstw” (stożek zbudowany jest z czworokątów).

bryły 3D w bibliotece GLUT:

Funkcja
void glutWireTorus(GLdouble innerRadius, Gldouble outerRadius, GLint sides, GLint rings)
void glutSolidTorus(GLdouble innerRadius, Gldouble outerRadius, GLint sides, GLint rings)
rysuje torus o środku położonym w początku układu współrzędnych, z osią obrotu pokrywającą się z osią OZ, promieniem wewnętrznym innerRadius, promieniem zewnętrznym outerRadius; sides określa ilość bocznych ścian, z których składa się pojedynczy pierścień, a rings ilość pierścieni, z których składa się torus.

Funkcja
void glutWireTeapot(GLdouble size)
void glutSolidTeapot(GLdouble size)
rysuje czajnik z Utah o rozmiarze size, opracowany przez Martina Newella.



Literatura pomocnicza:

Kurs OpenGL, C++
http://cpp0x.pl/kursy/Kurs-OpenGL-C++/101

OpenGL Programming Guide (Addison-Wesley Publishing Company)
http://neo.dmcs.pl/tgk/redbook.pdf

OpenGL Programming Guide
http://www.glprogramming.com/red/index.html

Wojciech Kowalewski: Wykłady z OpenGL – materiały dostępne na Contact.dir

c. d. n.