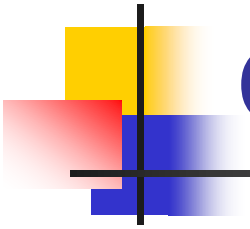
Wykład 2:

Biblioteka OpenGL elementy podstawowe

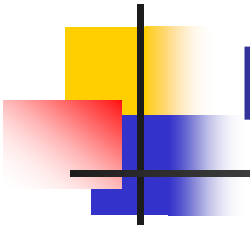


OpenGL wg Wikipedii:

OpenGL (Open Graphics Library) to specyfikacja wieloplatformowego, otwartego i uniwersalnego API do tworzenia grafiki. Zestaw funkcji składa się z 250 podstawowych wywołań, umożliwiających budowanie złożonych trójwymiarowych scen z podstawowych figur geometrycznych.

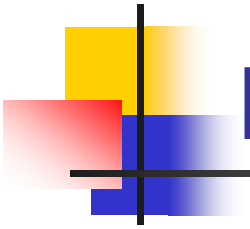
Głównym celem jest tworzenie grafiki. Dzięki temu, że polecenia są realizowane przez sprzęt, tworzenie grafiki następuje szybciej niż innymi sposobami.

Pierwotna wersja biblioteki wyewoluowała z biblioteki IRIS GL stworzonej przez firmę **Silicon Graphics Inc**. dla jej systemów graficznych (1992).



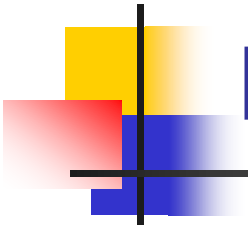
krótka historia:

1992 – specyfikacja standardu OpenGL 1.0
1992-2003 – kolejne wersje oznaczone numerami 1.1–1.5
2004 – specyfikacja standardu OpenGL 2.0
2006 – specyfikacja standardu OpenGL 2.1
2008 – specyfikacja standardu OpenGL 3.0
2010 – specyfikacja standardu OpenGL 4.1
2013 – specyfikacja standardu OpenGL 4.4
2014 – specyfikacja standardu OpenGL 4.5



podstawowe funkcjonalności:

- wykonywanie rysunków kreskowych, z cieniowaniem głębokości, obcinaniem głębokości i antyaliasingiem
- wyświetlanie scen zbudowanych z wielokątów z uwzględnieniem widoczności i oświetlenia, nakładaniem tekstury i efektami specjalnymi, takimi jak mgła, głębia ostrości, wspomaganie wyznaczania cieni
- ułatwienia w programowaniu animacji — można skorzystać z podwójnego buforowania obrazu, a także otrzymać rozmycie obiektów w ruchu
- obsługa list obrazowych, wspomaganie wyszukiwania obiektów w scenie
- programowanie bezpośrednio sprzętu graficznego (obecnie karty graficzne zawierają procesory o mocy kilkakrotnie większej niż główny procesor komputera); ten kierunek rozwoju standardu obecnie dominuje
- praca w sieci, w trybie klient-serwer.



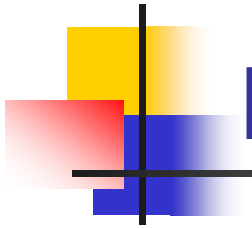
biblioteki OpenGL:

GL - funkcje „niskiego poziomu”, które mogą być realizowane przez sprzęt; figury geometryczne przetwarzane przez te funkcje, to punkty, odcinki i wielokąty wypukłe; nazwy wszystkich funkcji z tej biblioteki zaczynają się od liter gl; plik nagłówkowy funkcji z tej biblioteki można dołączyć do programu pisząc #include <GL/gl.h>

OpenGL jest interfejsem niskopoziomowym zyskując wsparcie w zestawie następujących bibliotek funkcji:

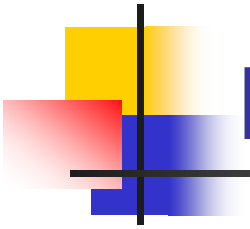
GLU - funkcje „wyższego poziomu”, w tym dające możliwości określania częściej spotykanych brył (sześcian, kula, stożek, krzywe i powierzchnie Béziera i NURBS); nazwy funkcji zaczynają się od glu, a plik nagłówkowy jest dołączany do programu poleceniem #include <GL/glu.h>

GLUT - funkcje całkowicie ukrywające szczegóły współpracy programu z systemem okienkowym; funkcje mają na początku nazwy przedrostek glut, a ich nagłówki umieszczamy w programie, pisząc #include <GL/glut.h> (dołączenie pliku glut.h spowoduje dołączenie poprzednich dwóch bibliotek).



podstawowe typy danych:

typ OpenGL	min ilość bitów	opis
GLboolean	1	typ logiczny
GLbyte	8	liczba całkowita ze znakiem
GLubyte	8	liczba całkowita bez znaku
GLchar	8	ciąg znaków tekstowych
GLshort	16	liczba całkowita ze znakiem
GLushort	16	liczba całkowita bez znaku
GLint	32	liczba całkowita ze znakiem
GLuint	32	liczba całkowita bez znaku
GLsizei	32	nieujemna liczba całkowita
GLenum	32	typ wyliczeniowy całkowity
GLintptr	ptrbits	wskaźnik na liczbę całkowitą
GLsizeiptr	ptrbits	wskaźnik na nieujemna liczbę całkowitą
GLbitfield	32	pole bitowe
GLfloat	32	liczba zmiennoprzecinkowa
GLclampf	32	liczba zmiennoprzecinkowa z przedziału [0, 1]
GLdouble	64	liczba zmiennoprzecinkowa
GLclampd	64	liczba zmiennoprzecinkowa z przedziału [0, 1]
GLvoid		typ pusty

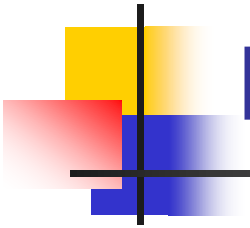


podstawowe ustalenia:

Biblioteka OpenGL stosuje **prawoskrętny układ współrzędnych** kartezjańskich, w którym oś OZ skierowana jest prostopadle do płaszczyzny monitora.

Biblioteka OpenGL wykorzystuje **model barw RGB**, w razie potrzeby uzupełniając składowe RGB o kanał alfa (RGBA).

Domyślnie obserwator w OpenGL:
- znajduje się jest w początku układu współrzędnych (0,0,0)
- skierowany jest w stronę ujemnej półosi OZ (patrząc na punkt 0,0,-100)
- jest tak zorientowany w przestrzeni, że kierunek „do góry” pokrywa się z kierunkiem osi OY.

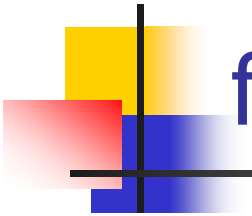


kolor tła, kolor obiektu:

Funkcja
void glClearColor(GLclampf r, GLclampf g, Glclampf b, GLclampf a)
gdzie parametry r, g, b to wartości składowych koloru RGB, natomiast parametr a to wartość składowej alfa, określa kolor tła. Wartości parametrów funkcji glClearColor powinny zawierać się w przedziale [0, 1].

Funkcja **void glClear(GLbitfield mask)**
gdzie parametr mask ma wartość GL_COLOR_BUFFER_BIT, określa zmianę koloru tła i wypełnienie stosownej pamięci kolorem wcześniej określonym jako kolor tła.

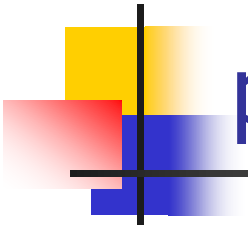
Funkcja **void glColor3f(GLfloat r, GLfloat g, GLfloat b)**
gdzie parametry r, g, b to wartości składowych koloru RGB, określa kolor obiektu. Wartości parametrów funkcji glColor3f powinny zawierać się w przedziale [0, 1], a składowa alfa przyjmuje domyślną wartość 1.0.



funkcje rysowania wierzchołków:

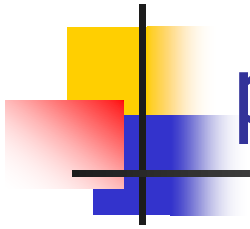
Każda z funkcji:
void glVertex3f(GLfloat x, GLfloat y, GLfloat z)
void glVertex3d(GLdouble x, GLdouble y, GLdouble z)
void glVertex3i(GLint x, GLint y, GLint z)
void glVertex2f(GLfloat x, GLfloat y)
void glVertex2d(GLdouble x, GLdouble y)
void glVertex2i(GLint x, GLint y)
...
gdzie x, y, z to wartości współrzędnych punktu w 3D (2D) określa wierzchołek o podanych współrzędnych w 3D lub 2D (dla 2D przyjmuje się współrzędną z=0)

Każdy wielokąt może być opisany w sposób jawny przez układ swoich wierzchołków w przestrzeni 3D. Wierzchołki danego wielokąta musza być zdefiniowane tak, aby wszystkie należały do jednej płaszczyzny.



podstawowe procedury rysowania:

Generowanie prymitywów graficznych rozpoczyna wywołanie funkcji:
void glBegin(GLenum mode)
gdzie parametr mode może przyjąć jedną z dziesięciu wartości:
GL_POINTS - kolejne punkty są traktowane indywidualnie
GL_LINES - każde kolejne dwa punkty są końcami odcinków
GL_LINE_STRIP - kolejne punkty są wierzchołkami łamanej otwartej
GL_LINE_LOOP - kolejne punkty są wierzchołkami łamanej zamkniętej
GL_TRIANGLES - kolejne trójki punktów są wierzchołkami trójkątów
GL_TRIANGLE_STRIP - taśma trójkątów; po wprowadzeniu dwóch pierwszych punktów każdy kolejny punkt powoduje wygenerowanie trójkąta, którego wierzchołki to ten punkt i dwa ostatnie wprowadzone wcześniej
GL_TRIANGLE_FAN - generowanie trójkątów; jednym z wierzchołków wszystkich trójkątów jest pierwszy punkt, dwa pozostałe to ostatni i przedostatni wprowadzony punkt

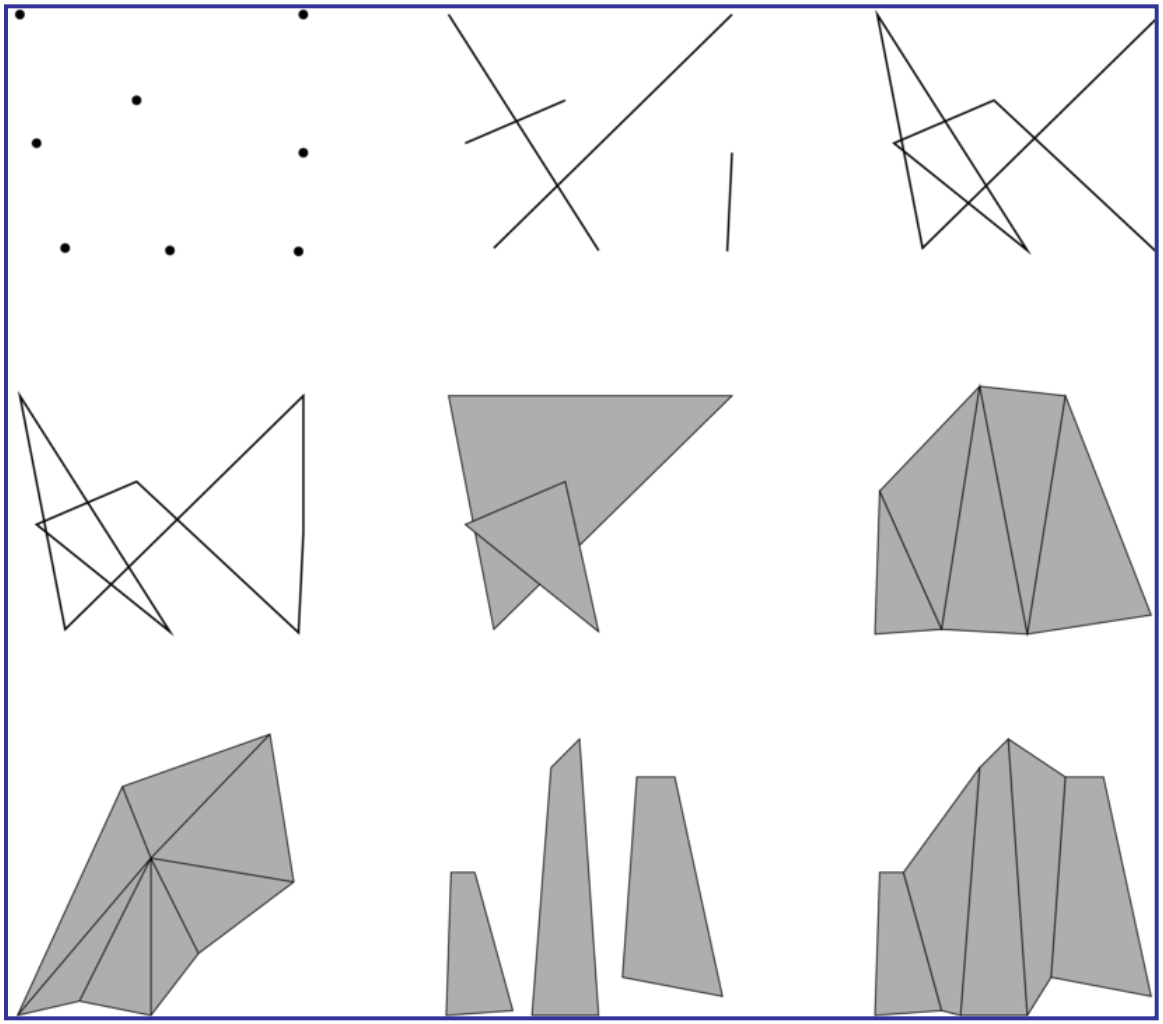


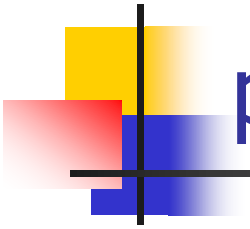
podstawowe procedury rysowania:

GL_QUADS - generowanie czworokątów wyznaczonych przez kolejne czwórki punktów
GL_QUAD_STRIP – taśma czworokątów, każda kolejna para punktów, z wyjątkiem pierwszej, powoduje wygenerowanie czworokąta, którego wierzchołkami są ostatnie cztery wprowadzone punkty
GL_POLYGON - wielokąt wypukły; wszystkie wierzchołki muszą leżeć w jednej płaszczyźnie.

Generowanie prymitywów graficznych kończy wywołanie funkcji:
void glEnd()

podstawowe procedury rysowania:





przykład:

```
glClearColor(1.0, 1.0, 1.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
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();
```

opis obserwatora rzut perspektywiczny:

Przestrzeń, którą obejmuje wzrokiem obserwator jest określona tzw. bryłą widzenia, która w przypadku **rzutu perspektywicznego** jest ostrosłupem o podstawie prostokąta, osi symetrii wyznaczającej główny kierunek obserwacji oraz określonej rozpiętości kątowej w pionie i w poziomie. Obserwator znajduje się w wierzchołku tej bryły. Obraz tworzony jest na rzutni równoległej do przedniej i tylnej ściany bryły widzenia.

Funkcja

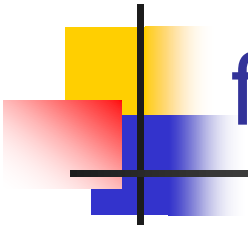
void gluLookAt(GLdouble ex, GLdouble ey, GLdouble ez,
GLdouble px, GLdouble py, GLdouble pz,
GLdouble dx, GLdouble dy, GLdouble dz)

gdzie ex, ey, ez to współrzędne położenia obserwatora, px, py, pz to współrzędne punktu, w kierunku którego zwrócony jest obserwator, dx, dy, dz to współrzędne wektora określającego kierunek „do góry”, pozwala zmienić na nowe dotychczasowe położenie obserwatora.

opis obserwatora rzut perspektywiczny:

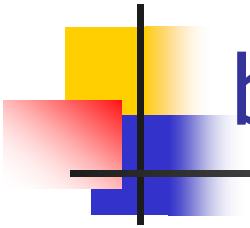
Funkcja
void glFrustum(GLdouble l, GLdouble r, GLdouble b, GLdouble t, GLdouble n, GLdouble f)
gdzie l, r, b, t wyznaczają współrzędne górnej podstawy ostrosłupa widzenia, natomiast n i f wyznaczają położenie odpowiednio górnej i dolnej podstawy ostrosłupa, które zawierają sie w płaszczyznach o równaniach z=-n i z=-f, (0 < n < f) określa parametry bryły widzenia.

Funkcja
void gluPerspective(GLdouble alfaY, GLdouble aspect, GLdouble n, GLdouble f)
gdzie alfaY to rozpietość kątowa w pionie, aspect to stosunek rozpiętości kątowej w poziomie do rozpiętości kątowej w pionie, n i f to odpowiednio odległości obserwatora od przedniej i tylnej ściany bryły widzenia, przy czym 0 < n < f, określa także parametry bryły widzenia.



funkcja opisująca scenę:

```
void scene() //nazwa funkcji może być dowolna
{
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(); //żądanie wykonania wszystkich określonych tutaj funkcji
}
```

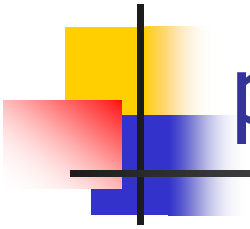


biblioteka GLUT:

Biblioteki GL i GLU **nie zawierają żadnej funkcji określającej okno.** Zatem aby wyświetlić na ekranie monitora sceny opisane w OpenGL, programista musi użyć zewnętrznej biblioteki okienkowej.

Prostą biblioteką okienkową, napisaną specjalnie dla OpenGL jest biblioteka **GLUT** (OpenGL Utility Toolkit). Zadaniem biblioteki GLUT jest ukrycie przed aplikacją wszystkich szczegółów interfejsu programowego systemu okien.

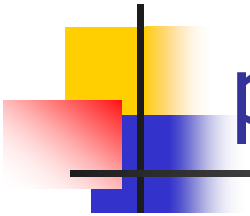
Okno renderingu (okno, w którym rysowana będzie scena) tworzone jest w funkcji main programu.



podstawowe funkcje biblioteki GLUT:

Wywołanie funkcji
glutInit(&argc, argv)
inicjuje wszystkie „stany” bieżących zmiennych biblioteki GLUT.
Parametrami są parametry wywołania programu podane w funkcji main.

Funkcja
void glutInitDisplayMode(unsigned int mode)
inicjuje bufor okna, w którym rysowana będzie scena. Wartością parametru mode jest suma bitowa odpowiednich stałych określających dane o oknie:
GLUT_RGB - okno będzie używało modelu barw RGB
GLUT_SINGLE - okno będzie używało jednego bufora obrazu
GLUT_DOUBLE - okno będzie używało dwóch buforów (płynna animacja)
GLUT_DEPTH - deklaruje używanie bufora głębokości
...

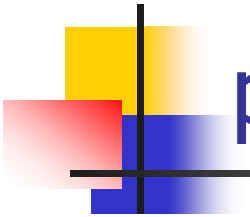


podstawowe funkcje biblioteki GLUT:

Funkcja
void glutInitWindowSize(int width, int height)
gdzie width i height oznaczają odpowiednio szerokość i wysokość okna, określa w pikselach początkowe wymiary okna renderingu.

Funkcja
void glutInitWindowPosition(int x, int y)
gdzie x i y oznaczają współrzędne piksela, określa położenie lewego, górnego narożnika okna na ekranie.

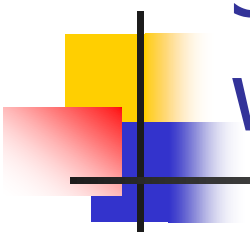
Funkcja
int glutCreateWindow(char *name)
gdzie name jest nazwą umieszczoną na górnej belce okna, dokonuje przygotowania struktury okna w pamięci RAM, okno nie jest jeszcze wyświetlone na ekranie.



podstawowe funkcje biblioteki GLUT:

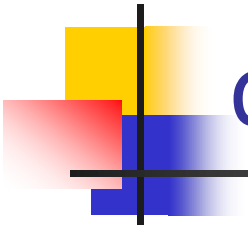
Funkcja
void glutDisplayFunc(void (* func)(void))
dokonuje rejestracji funkcji podanej jako parametr, która będzie wywoływana za każdym razem, gdy nastąpi konieczność odtworzenia (narysowania) zawartości okna, także gdy aplikacja zażąda wywołania tej funkcji za pomocą wywołania funkcji glutPostRedisplay().

Funkcja
void glutMainLoop()
powoduje uruchomienie pętli obsługi zdarzeń:
- wyświetlenie na ekranie wcześniej zdefiniowanego okna
- załadowanie do okna obrazu sceny, przez wywołanie funkcji opisującej scenę
- oczekiwanie na kolejne zdarzenia (np. exit).



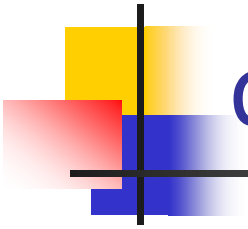
schemat programu wykorzystującego bibliotekę GLUT:

```
#include <GL/glut.h> // dołączenie nagłówka biblioteki GLUT;
void scene() {
//wywołania zestawu funkcji OpenGL opisujących scenę
}
void main(int argc, char *argv[]) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutInitWindowSize(400,300);
   glutInitWindowPosition(100,100);
   glutCreateWindow(”obraz 1”);
   glutDisplayFunc(scene);
   glutMainLoop();
}
```



dodatkowe uwagi:

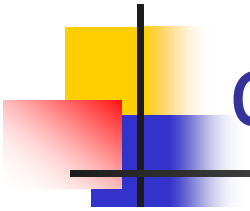
Jeśli zostanie zdefiniowana scena, w której obserwator widzi obiekty wzajemnie się zasłaniające, to w funkcji opisującej scenę należy dodać wywołanie:
glEnable(GL_DEPTH_TEST);
co umożliwia właściwe zasłanianie się obiektów z punktu widzenia obserwatora
oraz zmodyfikować linię kodu glClear(GL_COLOR_BUFFER_BIT);
pisząc
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
W funkcji main należy zastąpić wywołanie
glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
wywołaniem
glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE | GLUT_DEPTH);



dodatkowe uwagi:

Funkcja
void glutReshapeFunc(void(* func)(int width, int height))
gdzie width i height oznaczają odpowiednio szerokość i wysokość okna, dokonuje rejestracji funkcji podanej jako parametr, która będzie wywoływana za każdym razem, gdy nastąpi konieczność odtworzenia (narysowania) zawartości okna stosownie do wymiarów okna, przekazywanych jako parametry.

Funkcja
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)
gdzie x, y to współrzędne lewego, dolnego narożnika obszaru renderingu względem lewego dolnego narożnika okna, a width i height to szerokość i wysokość okna renderingu, umożliwia dynamiczną modyfikację rozmiarów okna renderingu.



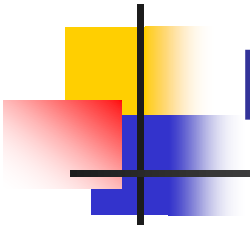
dodatkowe uwagi:

Gdy zmieniamy rozmiar okna przy pomocy myszki, scena może stracić proporcje. Ponieważ po zrzutowaniu sceny na okno rzutni, okno to jest przekształcane na okno renderingu (liniowo), wiec w przypadku gdy proporcje wymiarów tych okien nie są takie same, proporcje obiektów w oknie ekranowym są inne, niż w oknie rzutni.
Niech okno rzutni ma wymiary SizeX i SizeY, natomiast okno renderingu ma odpowiednio M (szerokość) i N (wysokość) pikseli. Aby obiekty w oknie renderingu miały takie same proporcje jak w oknie rzutni, musi zachodzić warunek: M/N = SizeX/SizeY.

Funkcję glViewport możemy wykorzystać do określenia w oknie renderingu podokna tak, aby width/height = SizeX/SizeY dla tego podokna. Warto również wycentrować podokno renderingu względem głównego okna.
Inne rozwiązanie zachowujące postulowany warunek proporcji to odpowiednie ustawienie drugiego parametru funkcji gluPerspective.

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
}
```



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.