Grafika komputerowa

Wstęp do OpenGL

Zbigniew Szymański

z.szymanski@ii.pw.edu.pl

listopad 2008 - v1

listopad 2010 - v1d

kwiecień 2013 – v1e

Sprawy organizacyjne /1/

- Prowadzący zajęcia:
  - Zbigniew Szymański

    pok. 361

    z.szymanski@ii.pw.edu.pl

  - Rajmund Kożuszek

    pok. 308

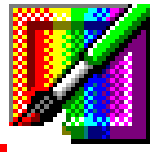
    r.kozuszek@ii.pw.edu.pl

  - Michał Kurowski

    pok. 304

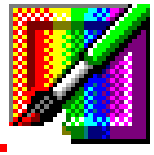
    m.kurowski@stud.elka.pw.edu.pl

Sprawy organizacyjne /2/



- Laboratorium nr 3 (czyli dzisiaj) – rozdanie projektów + wstęp do OpenGL
- Laboratorium nr 7 – oddanie gotowych rozwiązań (w dowolnym terminie lab. 7, jeśli prowadzący, który wydał projekt prowadzi dany termin)
- Dzisiejsze zajęcia nie są oceniane.
- Za projekt można uzyskać maksymalnie 10 pkt.
  - 7 pkt. - spełnienie narzuconych wymagań
  - 3 pkt. - implementacja dodatkowego efektu wizualnego
- **Uwaga – warunkiem koniecznym zaliczenia laboratorium jest uzyskanie min. 4 punktów z projektu OpenGL.**
- Deadline: ostatnie zajęcia nr 7 w harmonogramie laboratorium

Przydatne linki



- **http://www.opengl.org/**

  Najróżniejsze materiały dotyczące biblioteki OpenGL m.in. Specyfikacja, informacje o nowościach

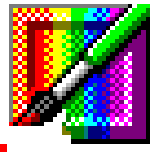
- **http://nehe.gamedev.net/**

  NeHe productions OpenGL tutorials – samouczki do programowania w OpenGL'u z wykorzystaniem biblioteki GLUT.

- **http://www.falloutsoftware.com/tutorials/gl/gl0.htm**

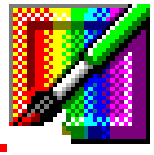
- **http://www.opengl.org/documentation/specs/glut/spec3/spec3.html**

Projekty



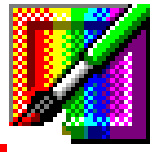
- Tematy projektów: proste programy wizualizujące lub proste gry zręcznościowe.
- Środowisko programistyczne: VisualStudio 2010, biblioteka OpenGL, biblioteka GLUT
- Należy przesłać pocztą elektroniczną parozdaniowy bardziej szczegółowy projekt.
- Celem zajęć jest zapoznanie się z biblioteką OpenGL – stosowanie wysokopoziomowych bibliotek nakładek (innych niż GLUT) na OpenGL nie jest dozwolone.
- Projekty powinny uruchamiać się i kompilować w sali 313. Oddawanie w innych salach lub na własnym sprzęcie nie jest możliwe.
- Gotowy projekt (kompilowalne źródła + wersja wykonywalna) należy wysłać do prowadzącego w formie jednego pliku *.zip.

Projekty – dobre rady



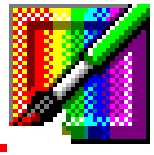
- Proszę nie czekać z pisaniem projekty do końca semestru – nie ma możliwości przesunięcia ostatecznego terminu.
- Przed oddaniem projektu proszę wcześniej sprawdzić czy się uruchamia i kompiluje w sali 313.

Projekty – co będzie oceniane



- Kod źródłowy (powinien być napisany zgodnie z zasadami sztuki programistycznej)
- Działanie programu.
- Estetyka wykonania m.in.:
  - Wizualizowane obiekty powinny zajmować centralną część ekranu o rozsądnym rozmiarze (a nie 10x10 pikseli w dolnym lewym rogu ekranu)
  - Wizualizowane obiekty nie powinny lewitować w kosmicznej próżni – należy dodać tło w scenie
  - Rozsądne użycie tekstur

Projekty – wymagania techniczne

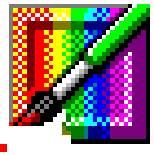


- Należy zastosować światła z możliwością regulacji intensywności.
- Należy pokazać umiejętność teksturowania (co nie oznacza konieczności teksturowania wszystkich obiektów w scenie).
- W scenie powinny znajdować się elementy ruchome.
- Ruch kamery / widza (niekoniecznie interaktywny).

Tematy zadań projektowych

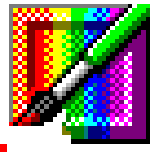
- Stonoga
- Pająk
- Węgorz
- Ślimak
- Meduza
- Rekin
- Żółw
- Płaszczka
- Krewetka
- Krab
- Ważka
- Motyl
- własne propozycje projektów

OpenGL



- Jest to API umożliwiające rysowanie grafiki 2D i 3D.
- Jest odpowiednikiem Microsoft Direct3D.
- Standardem zarządza konsorcjum Khronos.
- Oprócz standardowego OpenGL, istnieje także OpenGL ES (Embedded Standard) oraz OpenGL SC (Safety Critical)
- Implementacje OpenGL i OpenGL ES istnieją na platformach Windows, Unix, Solaris, Linux, Mac, BeOS/Haiku, Android, Bada, iOS, Symbian, PS3...
- Jest wykorzystywany w Unreal Engine 3, Unigine, id Tech 4 , Ogre 3D, World of Warcraft...

OpenGL – krotka historia

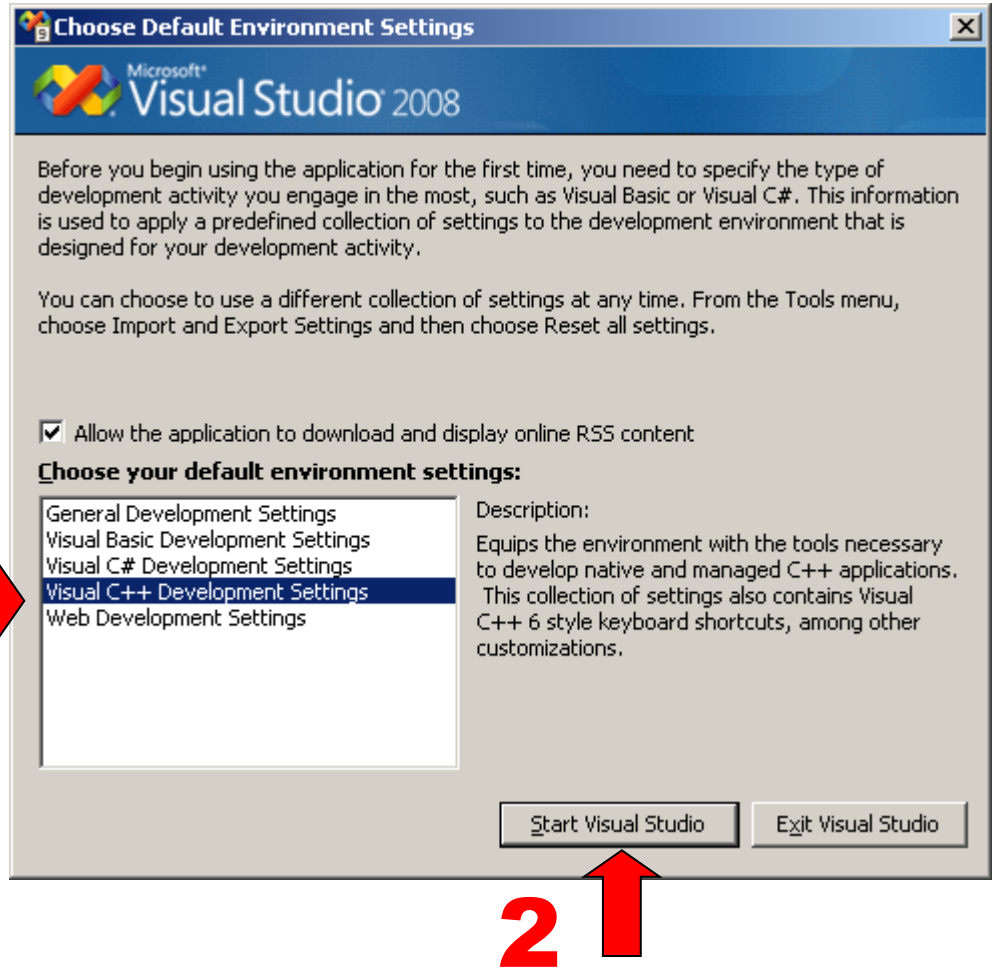


- 1992: 1.0; 1.1 - Vertex Arrays;
- 1998: 1.2 – 3D Textures; 1.2.1 – rozszerzenia ARB;
- 2001: 1.3 – Multisampling, Multitexturing;
- 2002: 1.4 – Depth Textures, Shadows;
- 2003: 1.5 – Buffer Objects;
- 2004: 2.0 – Shader Objects, Shader Programs, Multiple Render Targets, Non-power-of-two Textures;
- 2006: 2.1 – Pixel Buffer Objects;
- 2008: 3.0 – Framebuffer Objects;
- 2009: 3.2 - Geometry Shaders;
- 2010: 4.0 – Tesselation Control and Evaluation Shaders;

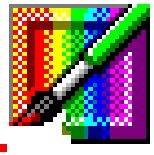
Prosty program z wykorzystaniem OpenGL /1/

- Uruchamiamy Visual Studio 2008

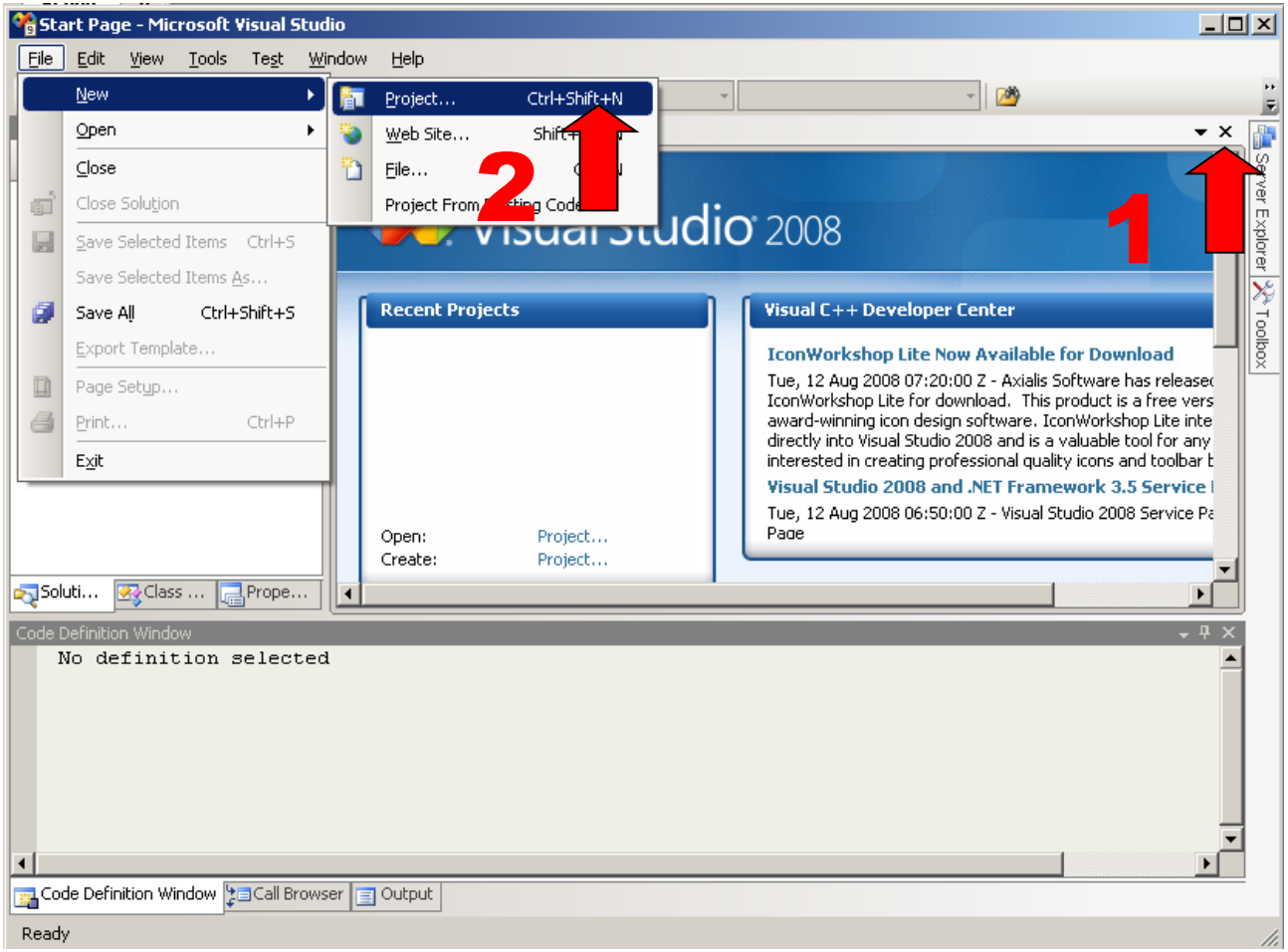
Menu Start | Programy | Microsoft Visual Studio 2008 | Microsoft Visual Studio 2008



Prosty program … /2/

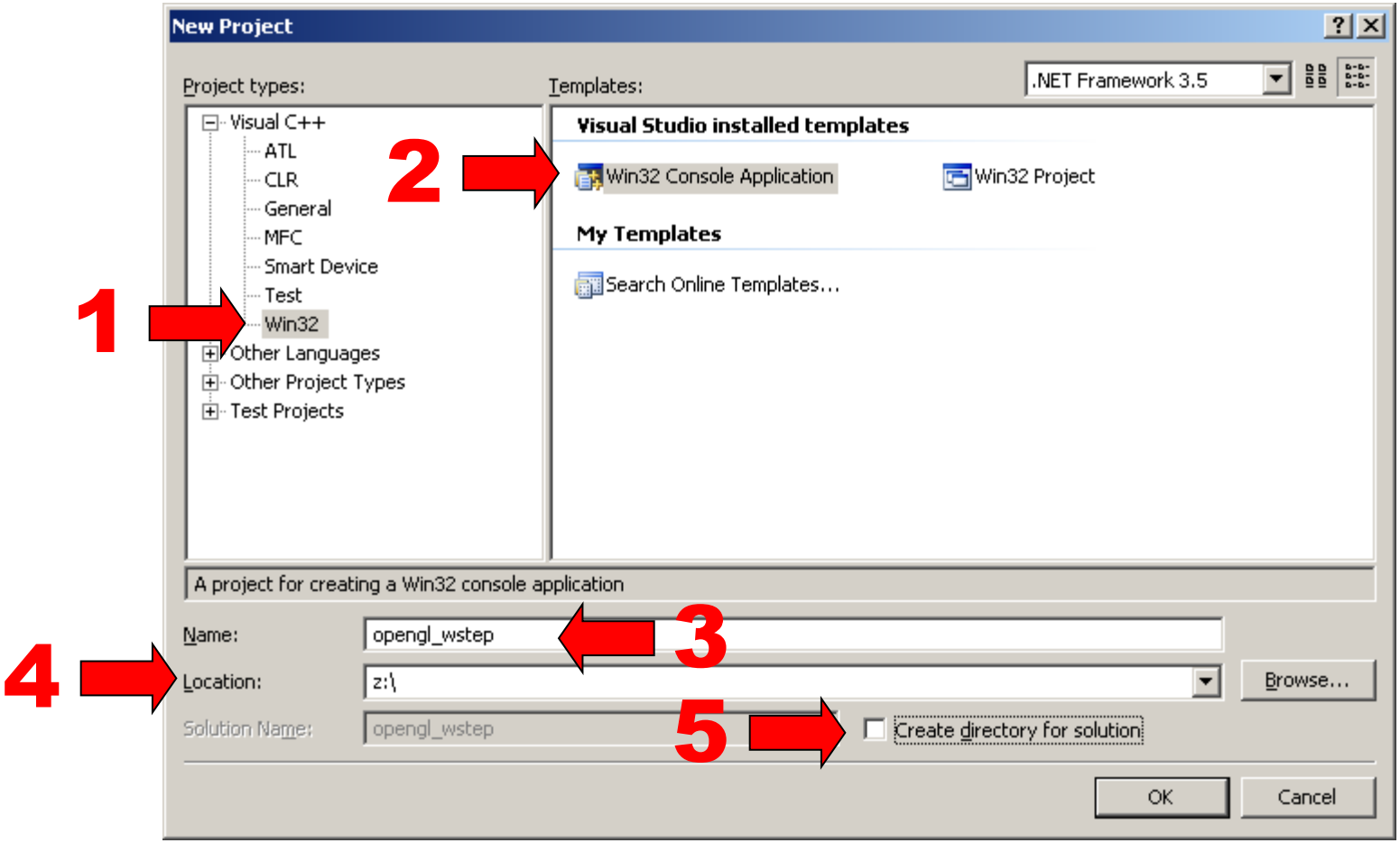


- Zamykamy „Start Page”
- Tworzymy nowy projekt: **`File | New | Project`**



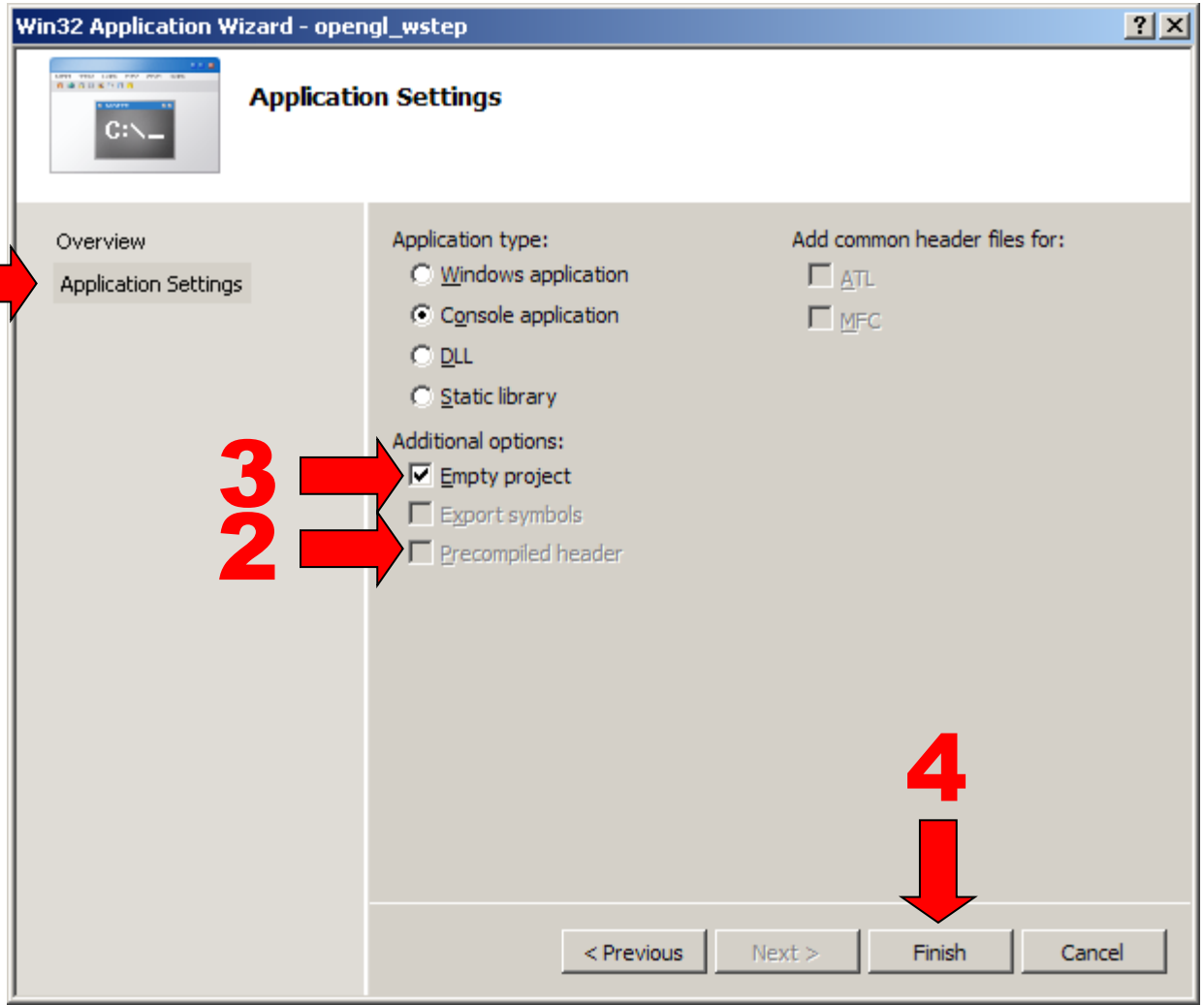
Prosty program ... /3/

- Typ projektu (Visual C++ | Win32 | Win32 Console App.),
- Nazwa (**`opengl_wstep`**)
- Położenie projektu (**`z:\`**), wyłączyć „create directory for solution”



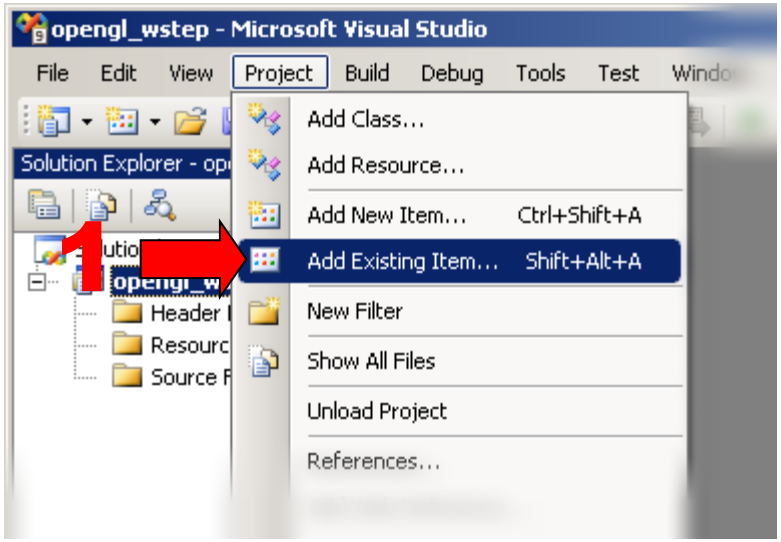
Prosty program … /4/

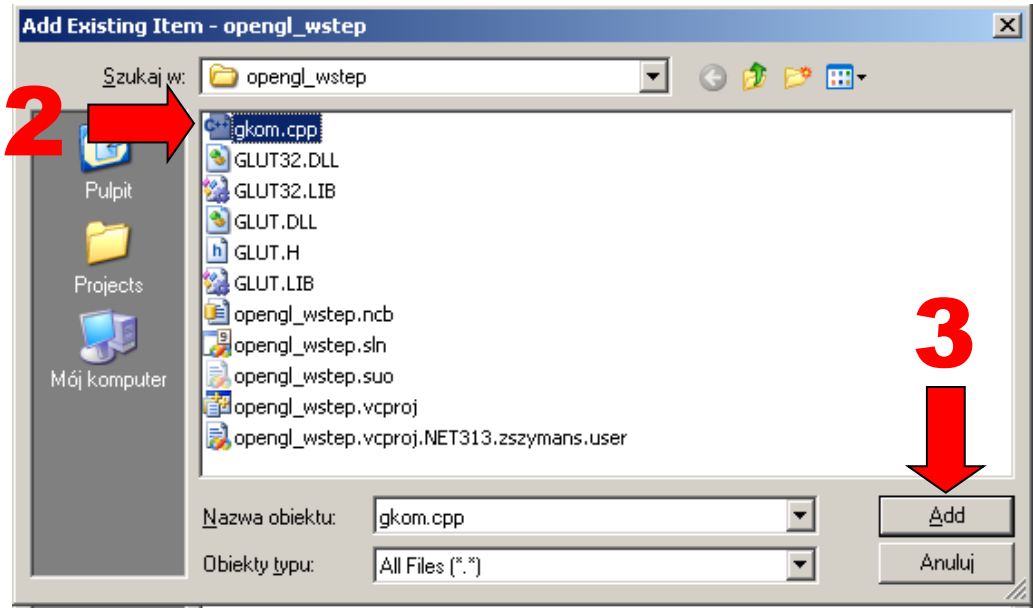
- W ustawieniach „Application settings” wyłączamy „Precompiled headers” i zaznaczamy „Empty project”



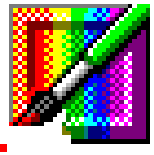
Prosty program … /5/

- Kopiujemy wszystkie pliki z katalogu:
  `\\NETGRAF\temporary\GKOM\OpenGL\zsz\`
  do katalogu z projektem:
  `z:\opengl_wstep`
- Dodajemy plik **`gkom.cpp`** do projektu
  `Project | Add Existing Item` …





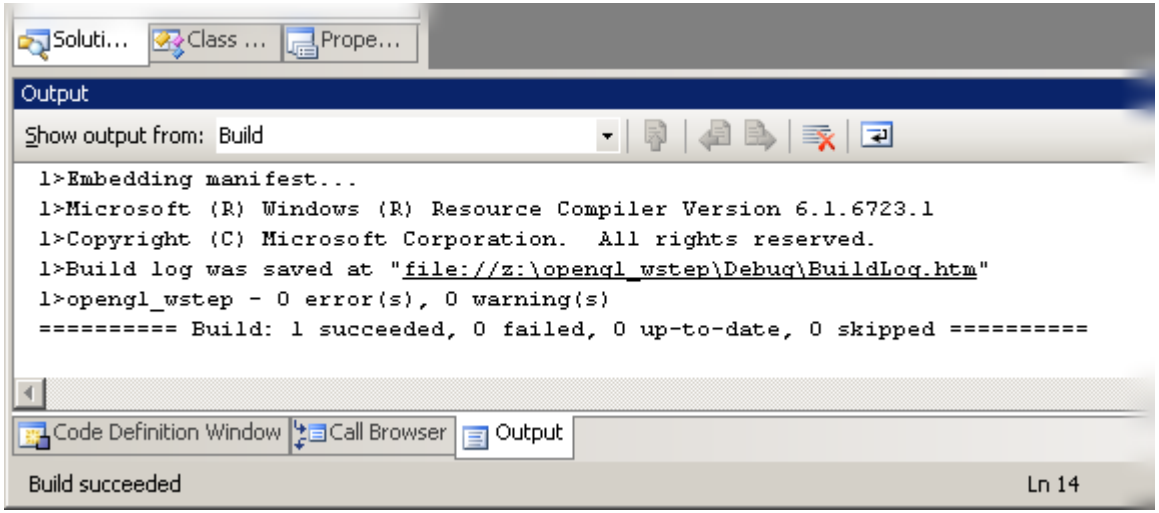
Prosty program … /6/



- Kompilacja projektu:

  `Build | Build opengl_wstep`

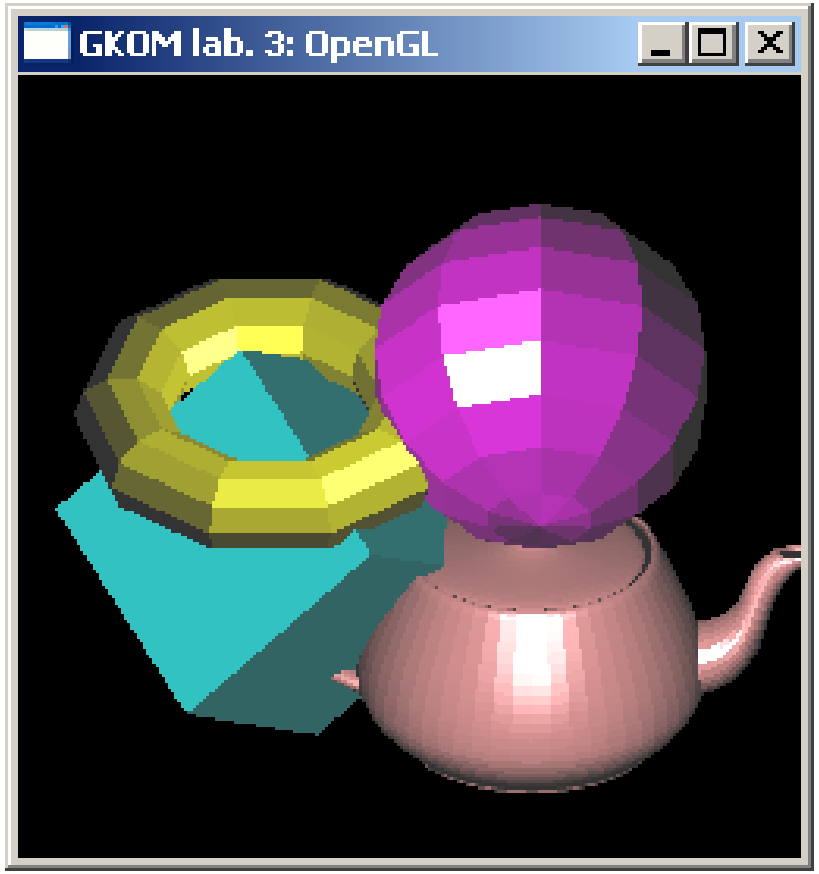
- Komunikaty dotyczące w kompilacji pojawiają się w oknie **`Output`**:



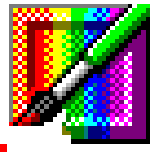
Prosty program … /7/

- Uruchomienie programu:

  `Debug | Start Debugging`



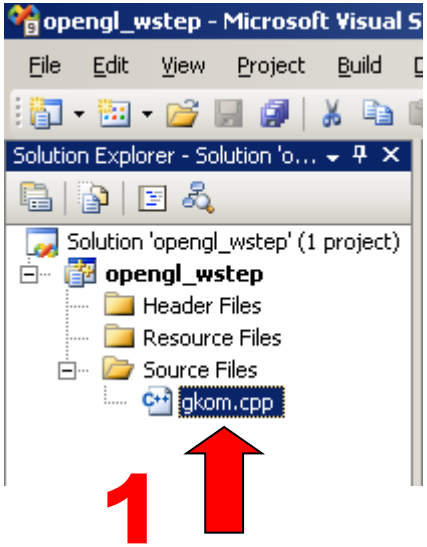
Biblioteka GLUT

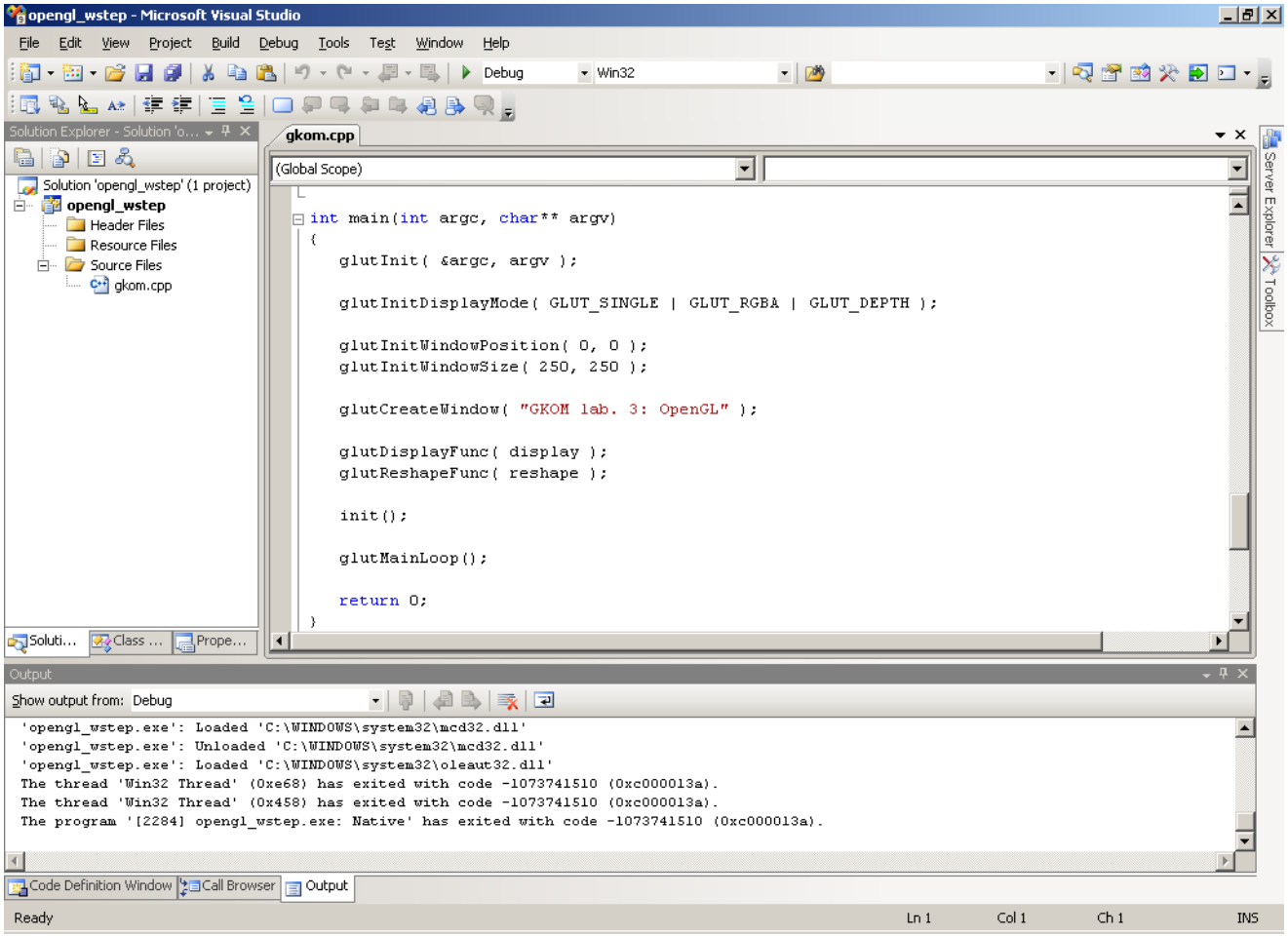


- Biblioteka GLUT (Open**GL** **U**tility **T**oolkit) umożliwia:
  - obsługę okien niezależną od systemu operacyjnego
  - obsługę zdarzeń
  - timery i funkcję „Idle” wykonywaną, gdy nic w systemie się nie dzieje
  - funkcje do generowania różnych brył
- Biblioteka GLUT jest dostępna dla różnych systemów operacyjnych (m.in. Windows i Linux)
- Nazwy funkcji z biblioteki GLUT posiadają przedrostek **`glut`**, a z biblioteki OpenGL przedrostek **`gl`**

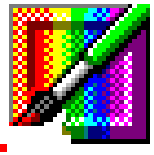
Oglądamy kod

- Otwieramy plik **gkom.cpp**





Funkcja `main`



```
glutInit( &argc, argv );
gl
                                          _DEPTH
gl
glutInitWindowSize( 250, 250 );
glutCreateWindow(
     "GKOM lab. 3: OpenGL" );
glutDisplayFunc( display );
glutReshapeFunc( reshape );
init();
glutMainLoop();
```

Funkcja dokonuje inicjalizacji biblioteki GLUT oraz inicjuje współpracę z systemem okienkowym.

Pytanie ?

Co to jest bufor głębokości (inaczej zwany z-buforem)?

Co to jest pojedyncze / podwójne buforowanie?

Funkcja `main`

```
glutInit( &argc, argv );
glutInitDisplayMode(
    GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH
    );
gl
gl
gl

gl
gl
in
gl
```

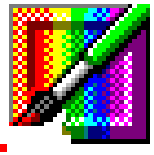
Ustawia tryb wyświetlania:

GLUT_SINGLE – maska bitowa wybierająca pojedyncze buforowanie okna.

GLUT_RGBA – tryb wyświetlania RGB (, a nie tryb indeksowany).

GLUT_DEPTH – tworzone okno będzie posiadało bufor głębokości.

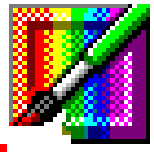
Funkcja `main`



```
glutInit( &argc, argv );
glutInitDisplayMode(
     GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH
     );
glutInitWindowPosition( 0, 0 );
glutInitWindowSize( 250, 250 );
gl
     GKOM lab. 3: OpenGL" );
glutDisplayFunc( display );
glutReshapeFunc( reshape );
init();
glutMainLoop();
```

Ustalenie położenia i rozmiaru okna.

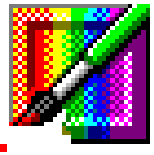
Funkcja `main`



```
glutInit( &argc, argv );
glutInitDisplayMode(
     GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH
     );
glutInitWindowPosition( 0, 0 );
glutInitWindowSize( 250, 250 );
glutCreateWindow(
     "GKOM lab. 3: OpenGL" );
gl
gl
in
glutMainLoop();
```

Utworzenie okna. Parametrem funkcji jest tytuł okna pojawiający się w pasku tytułu okna aplikacji.

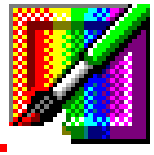
Funkcja `main`



```
glutInit( &argc, argv );
glutInitDisplayMode(
    GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH
gl
gl
gl
glutDisplayFunc( display );
glutReshapeFunc( reshape );
init();
glutMainLoop();
```

Rejestracja funkcji callback (wywoływanej gdy zajdzie określone zdarzenie) odpowiedzialnej za odrysowanie zawartości okna. Funkcja **display** jest zaimplementowana przez użytkownika.

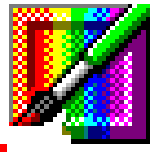
Funkcja **main**



```
glutInit( &argc, argv );
glutInitDisplayMode(
     GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH
     );
glutInitWindowPosition( 0, 0 );
gl
gl
gl
glutReshapeFunc( reshape );
init();
glutMainLoop();
```

Rejestracja funkcji callback wywoływanej gdy zmieni się rozmiar okna. Funkcja **reshape** jest zaimplementowana przez użytkownika.

Funkcja `main`

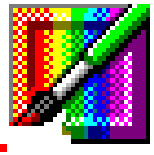


```
gl
gl
                                    DEPTH
gl
gl
gl
gl
gl
init();
glutMainLoop();
```

Funkcja **init** jest zaimplementowana przez użytkownika. Dokonuje ustawienia oświetlenia i spraw związanych z cieniowaniem.

Funkcja **glutMainLoop** rozpoczyna przetwarzanie zdarzeń przez system – odbiera informacje o zachodzących zdarzeniach i wywołuje stosowne procedury obsługi zdarzeń.

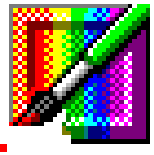
Funkcja **display**



```
glClear(
  GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT );
di
gl
```

Funkcja powodująca wyczyszczenie wskazanych buforów: bufora kolorów (przechowującego obraz sceny) oraz bufora głębokości (z-bufora).

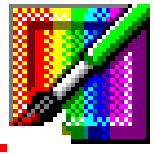
Funkcja **display**



```
glClear(
  GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT );
displayObjects();
gl
```

Nasza funkcja wyświetlająca scenę.

Funkcja **display**

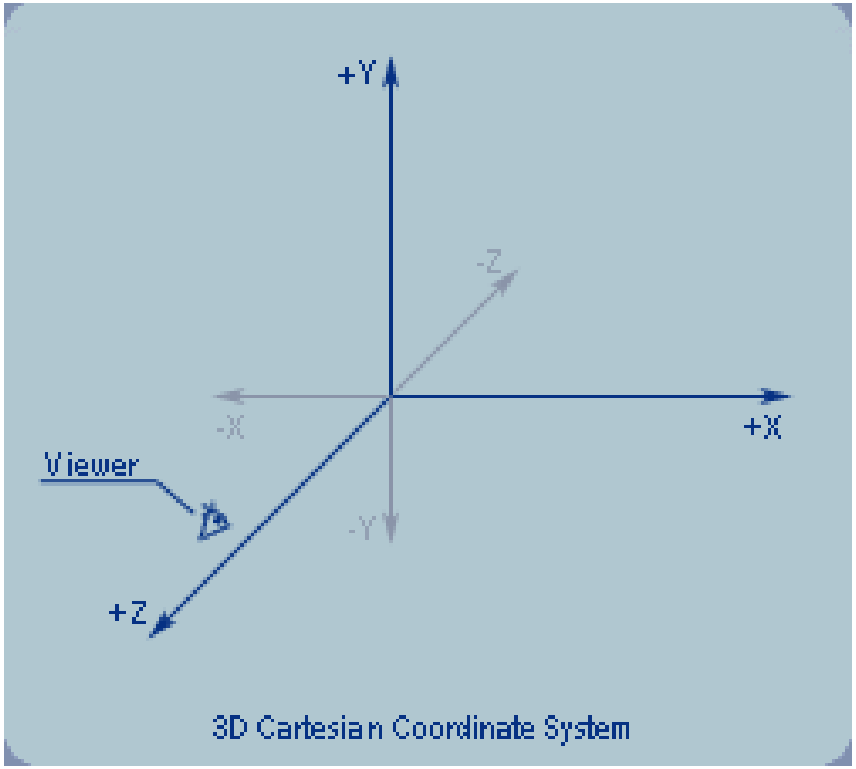


```
glClear(
  GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT );
displayObjects();
glFlush();
```

W różnych implementacjach OpenGL'a polecenia są buforowane. Funkcja glFlush powoduje opróżnienie tych buforów, co skutkuje wykonaniem wydanych poleceń tak szybko, jak umożliwia to 'silnik renderujący'.

Tworzenie nowych obiektów

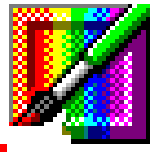
- Układ współrzędnych stosowany w OpenGL'u



[http://www.falloutsoftware.com/tutorials/gl/]

- Nowe obiekty tworzone są w środku układu współrzędnych.

Funkcja `displayObjects`



- W funkcji **`displayObjects`** znajdują się wywołania 4 funkcji tworzących obiekty w scenie:
  - **`glutSolidTorus( 0.275, 0.85, 10, 10 );`**

    Tworzy torus – parametry: promień wewnętrzny, promień zewnętrzny, liczba odcinków każdej sekcji radialnej, liczba sekcji radialnych
  - **`glutSolidCube( 1.5 );`**

    Tworzy sześcian o podanym boku.
  - **`glutSolidSphere( 1.0, 10, 10 );`**

    Tworzy kulę – parametry: promień, liczba podziałów wokół osi Z, liczba podziałów wzdłuż osi Z.
  - **`glutSolidTeapot( 1.0 );`**

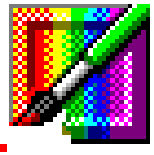
    Tworzy czajnik o podanym rozmiarze.

Pytanie ?

Co to są współrzędne jednorodne?

Co to są macierze przekształceń?

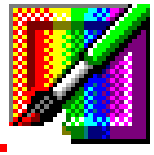
Transformacje na obiektach /1/



- Wszystkie transformacje geometryczne (obroty, translacje, skalowanie) wykonywane są OpenGL'u w formie działań na macierzach. W momencie wywołania funkcji tworzącej obiekt, jego wierzchołki są wymnażane przez macierz aktualnego przekształcenia.

- Wywołania funkcji dokonujących transformacji (obroty, translacje, skalowanie) powodują wymnożenie macierzy aktualnego przekształcenia przez macierz określonej transformacji.

Transformacje na obiektach /2/



- Przykład: Na wierzchołku o współrzędnych P chcemy dokonać trzech transformacji opisywanych przez macierze T_1, T_2, T_3. Policzyć współrzędne wierzchołka po tych transformacjach.

Współrzędne wierzchołka po transformacji T_1: $P_1=T_1*P$

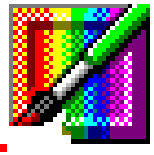
Współrzędne wierzchołka po transformacji T_2: $P_2=T_2*P_1=T_2*T_1*P$

Współrzędne wierzchołka po transformacji T_3: $P_3=T_3*P_2=T_3*T_2*T_1*P$

Macierz aktualnego przekształcenia

- Aby wykonać w kolejności transformacje T_1, T_2, T_3 ich macierze musimy wymnożyć w odwrotnej kolejności !

Funkcje dokonujące transformacji



- **`glRotatef( 100.0, 1.0, 0.0, 0.0 );`**

  kąt, współrzędne punktu definiującego oś obrotu

  Funkcja wymnażająca macierz aktualnego przekształcenia przez macierz reprezentującą obrót o kąt podany jako pierwszy parametr funkcji. Obrót dokonywany jest wokół osi przechodzącej przez punkt (0,0,0) i punkt, którego współrzędne są podane jako pozostałe trzy parametry funkcji.

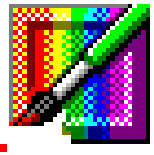
- **`glTranslatef( -0.80, 0.35, 0.0 );`**

  Funkcja wymnażająca macierz aktualnego przekształcenia przez macierz reprezentującą przesunięcie o podany wektor.

- **`glScalef(1.0,1.0,1.0);`**

  Funkcja wymnażająca macierz aktualnego przekształcenia przez macierz reprezentującą skalowanie.

Stos macierzy

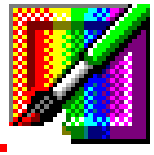


- Funkcja **`glPushMatrix()`** odkłada macierz aktualnego przekształcenia na stos.

- Funkcja **`glPopMatrix()`** zdejmuje ze stosu macierz przekształcenia. Macierz ta staje się macierzą aktualnego przekształcenia.

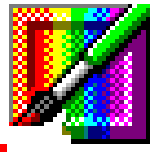
- Operacje te umożliwiają szybkie przywrócenie macierzy aktualnego przekształcenia bez konieczności wywoływania sekwencji funkcji glRotatef, glScalef, glTranslatef

Oglądamy kod



Na podstawie zawartości funkcji **`displayObjects`** proszę ustalić, jakie przekształcenia zostaną zastosowane do torusa (w jakiej kolejności)?

Torus /1/

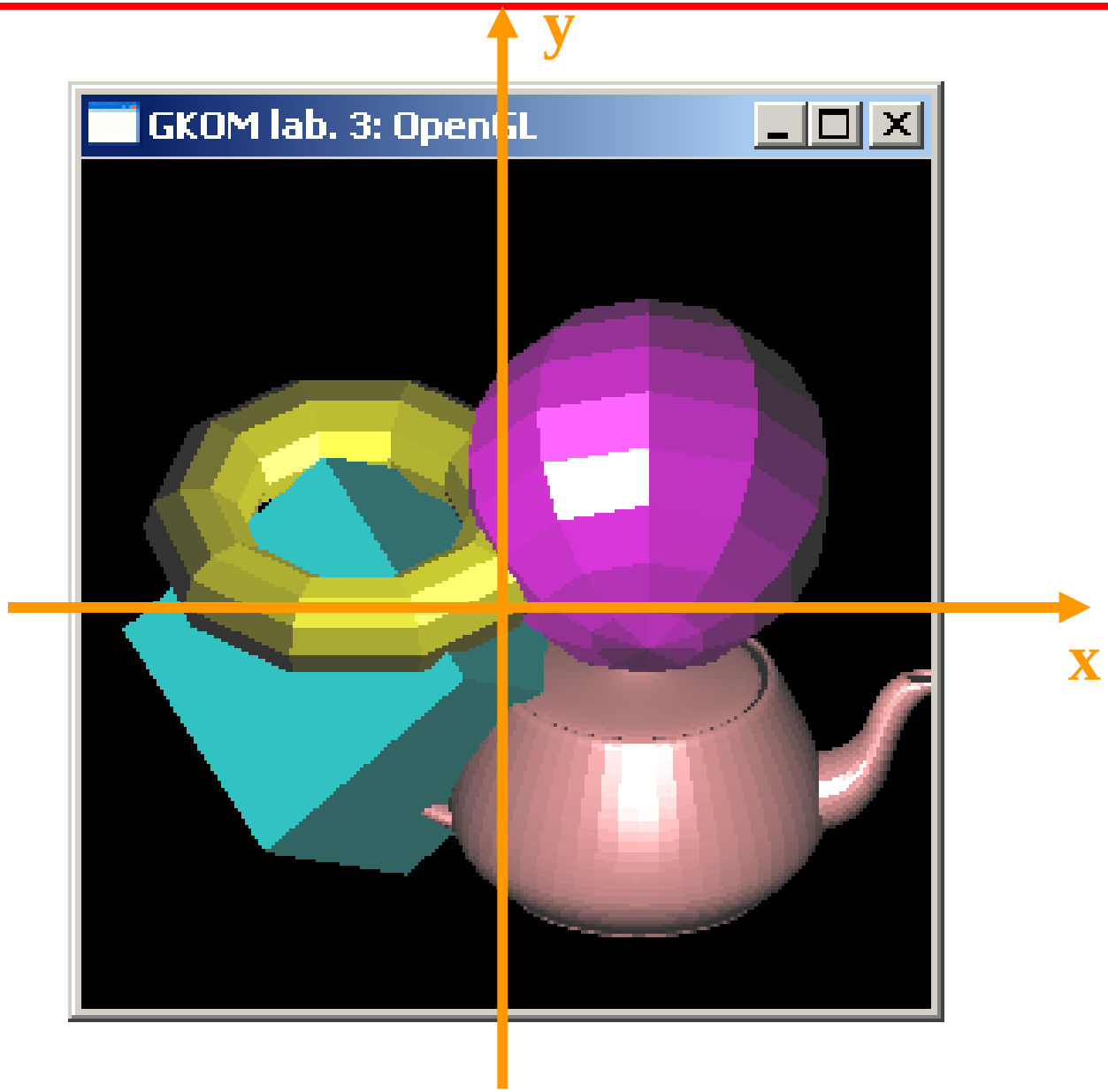


```
glRotatef( 30.0, 1.0, 0.0, 0.0 );
glTranslatef( -0.80, 0.35, 0.0 );
glRotatef( 100.0, 1.0, 0.0, 0.0 );
glutSolidTorus( 0.275, 0.85, 10, 10 );
```

Powyższą sekwencję można sobie wyobrazić jako:

1. Utworzenie torusa w środku układu współrzędnych w płaszczyźnie XY.
2. Obrót torusa o 100 stopni wokół osi X.
3. Przesunięcie torusa w lewo i do góry o wektor [-0.8, 0.35, 0.0]
4. Obrót torusa o 30 stopni wokół osi X.

Torus /2/



Włączenie podwójnego buforowania /1/

- Proszę włączyć tryb podwójnego buforowania dodając w funkcji **main** elementy zaznaczone na pomarańczowo.

```
glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA |
  GLUT_DEPTH );
glutInitWindowPosition( 0, 0 );
glutInitWindowSize( 250, 250 );
glutCreateWindow( "GKOM lab. 3: OpenGL" );
glutDisplayFunc( display );
glutReshapeFunc( reshape );
glutIdleFunc( display );
```

Funkcja Idle jest wywoływana, gdy system nie ma nic innego do wykonania – posłuży do animacji sceny.

- Proszę skompilować i uruchomić program. O czym zapomnieliśmy?

Włączenie podwójnego buforowania /2/

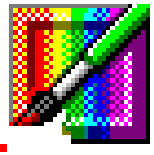
- Zapomnieliśmy o przełączaniu buforów. Rysowanie odbywało się na buforze, który nie był wyświetlany.

- Proszę włączyć przełączanie buforów dodając w funkcji **display** elementy zaznaczone na pomarańczowo.

```
glClear(
    GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
displayObjects();
glFlush();
glutSwapBuffers();
```

- Proszę skompilować i uruchomić program.

Zadanie



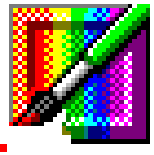
Wpraw torus w obrót dookoła własnej osi X.

Obrót torusa /1/

- Należy dodać w funkcji **display** elementy zaznaczone na pomarańczowo. Parametryzujemy funkcję **displayObjects** numerem ramki.

```
static int frame_no=0;
glClear(
     GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
if (frame_no<360) frame_no++; else frame_no=0;
displayObjects(frame_no);
glFlush();
glutSwapBuffers();
```

Zadanie



Proszę dwukrotnie zmniejszyć obiekty modyfikując parametry funkcji, które je tworzą.

Zmiana trybu cieniowania

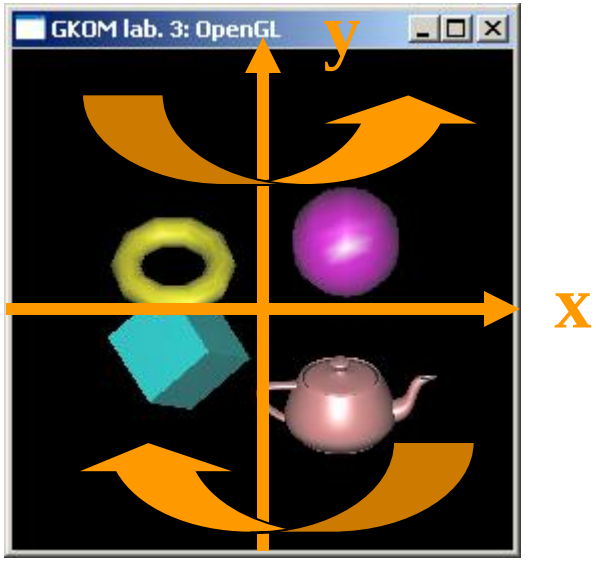
- Proszę włączyć tryb cieniowania interpolowanego zmieniając w funkcji **`init`** element zaznaczony na pomarańczowo.

```
glShadeModel( GL_SMOOTH );
```

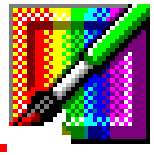
- Proszę skompilować i uruchomić program.

Zadanie

Wpraw kulę i torus w obrót wokół osi Y.
Wpraw czajnik i sześcian w obrót wokół osi Y w przeciwną stronę.



Rzutowanie



- Biblioteka OpenGL dokonując obliczeń związanych z rzutowaniem posługuje się drugą macierzą przekształcenia tzw. macierzą rzutowania.
- Modyfikacji tej macierzy można dokonać po przełączeniu trybu macierzy na **GL_PROJECTION** wywołując funkcję:

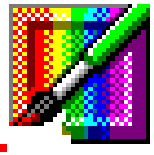
```
glMatrixMode( GL_PROJECTION );
```

- Powrót do trybu macierzy modelu dokonujemy wywołując funkcję:

```
glMatrixMode( GL_MODELVIEW );
```

- Można sobie wyobrazić, że przekształcenia dokonywane w trybie **GL_MODELVIEW** odnoszą się do obiektów w scenie, zaś w trybie **GL_PROJECTION** do obserwatora/kamery.

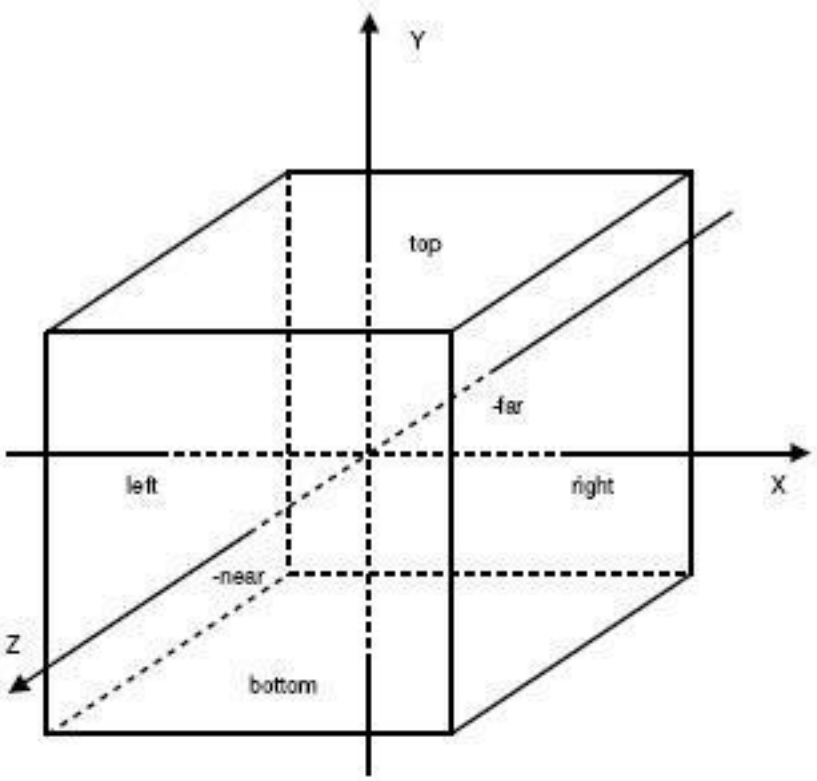
Rzutowanie równoległe



Macierz rzutowania dla rzutu równoległego ustawiana jest przy pomocy funkcji **`glOrtho`**.

```
glOrtho(x_left,x_right,y_bottom,y_top,z_near,z_far);
```

Parametry funkcji definiują (współrzędne) bryłę obcinania.



źródło: [http://kursy.ddt.pl/?LessonId=181]

Rzutowanie równoległe – funkcja **reshape**

```
if( h > 0 && w > 0 ) {
  glViewport( 0, 0, w, h );
  g
  g
  i
                -10.0, 10.0 );
    }
  else {
    glOrtho(-2.25*w/h,2.25*w/h,-2.25,2.25,
                -10.0, 10.0 );
    }
  glMatrixMode( GL_MODELVIEW );
  }
```

Ustala obszar okna przeznaczony do wykorzystania przez OpenGL. Parametry określają współrzędne wewnątrz okna.

Rzutowanie równoległe – funkcja **reshape**

```
if( h > 0 && w > 0 ) {
  glViewport( 0, 0, w, h );
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  if( w
    glO
    }
  else {
    glOrtho(-2.25*w/h,2.25*w/h,-2.25,2.25,
            -10.0, 10.0 );
    }
  glMatrixMode( GL_MODELVIEW );
  }
```

Włączenie trybu macierzy rzutowania.

Do macierzy rzutowania wpisywana jest macierz jednostkowa.

Rzutowanie równoległe – funkcja **reshape**

```
if( h > 0 && w > 0 ) {
  glViewport( 0, 0, w, h );
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  if( w <= h ) {
    glOrtho(-2.25,2.25,-2.25*h/w,2.25*h/w,
            -10.0, 10.0 );
    }
  else {
    glOrtho(-2.25*w/h,2.25*w/h,-2.25,2.25,
            -10.0, 10.0 );
    }
  glMa
  }
```

Ustalenie bryły obcinania, w zależności od tego czy okno jest dłuższe czy szersze.

Rzutowanie równoległe – funkcja **reshape**

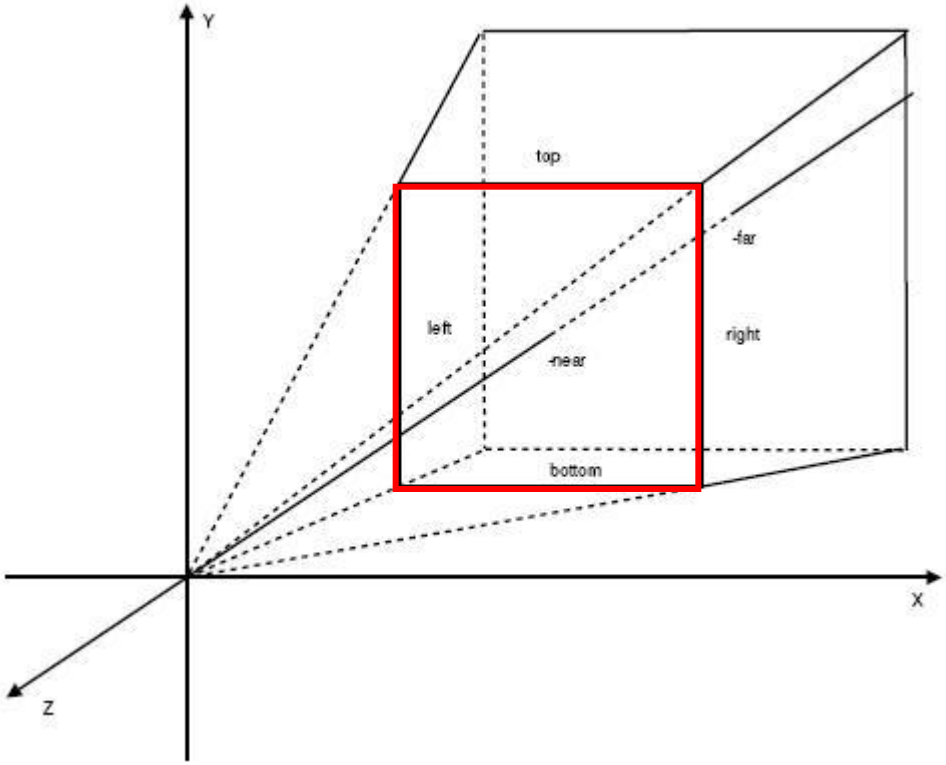
```
if( h > 0 && w > 0 ) {
  glViewport( 0, 0, w, h );
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  if( w <= h ) {
    glOrtho(-2.25,2.25,-2.25*h/w,2.25*h/w,
            -10.0, 10.0 );
    }
  else {
    glOrtho(-2.25*w/h,2.25*w/h,-2.25,2.25,
            -10.0, 10.0 );
    }
  glMatrixMode( GL_MODELVIEW );
  }
```

Włączenie trybu macierzy modelu.

Rzutowanie perspektywiczne

Macierz rzutowania dla rzutu perspektywicznego ustawiana jest przy pomocy funkcji **`glFrustum`**.

```
glFrustum(x_left,x_right,y_bottom,y_top,d_near,d_far);
```



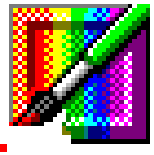
źródło: [http://kursy.ddt.pl/?LessonId=181]

Oko obserwatora jest początkowo umieszczone w punkcie (0,0,0).

Obraz jest rzutowany na płaszczyznę *near* (ekran komputera).

d_near, d_far są odległościami od oka obserwatora do płaszczyzn *near* i *far*. Muszą być dodatnie.

Zadanie



Włącz rzutowanie perspektywiczne

Zadanie

Zmodyfikuj program tak aby obserwator okrążał scenę w płaszczyźnie XZ.

Należy zmodyfikować funkcję display:

1. Przełącz tryb macierzy na tryb rzutowania
2. Zapamiętaj stan aktualnej macierzy
3. Dokonaj obrotu
4. Przełącz tryb macierzy na tryb modelu
5. Wyświetl obiekty
6. Przełącz tryb macierzy na tryb rzutowania
7. Odtwórz stan macierzy

Vertex Arrays – przykładowy program

- Proszę skopiować zawartość pliku **gkom.cpp** do innego pliku (jeśli ma być jeszcze później wykorzystany).
- Kopiujemy zawartość pliku:

  \\Netgraf2\temporary\GKOM\opengl\VAProject\gkom_va.cpp

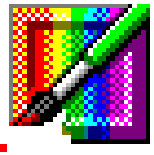
  do pliku:

  gkom.cpp
- Kopiujemy pliki girl.obj oraz viking.obj z katalogu

  \\Netgraf2\temporary\GKOM\opengl\VAProject

  do katalogu z projektem (nie dodajemy do projektu!).
- Proszę przekompilować i uruchomić projekt.

Vertex Arrays



- Umożliwiają rysowanie dowolnych wielokątów.
- Aby z nich skorzystać, należy włączyć odpowiednie opcje:

```
glEnableClientState( GL_VERTEX_ARRAY );
glEnableClientState( GL_NORMAL_ARRAY );
```

- Umożliwia to podanie odpowiednich wskaźników na dane:

```
glVertexPointer( 3, GL_FLOAT, 0, vs);
glNormalPointer( GL_FLOAT, 0, ns);
```

- Mając ustalone źródła danych, możemy przystąpić do rysowania podając indeksy kolejnych wierzchołków:

```
glDrawElements( GL_TRIANGLES, count,
GL_UNSIGNED_SHORT, is);
```

KONIEC