

dfgldfjld,mhncv;lfdf;lfkgjldfbbcmvb,m;dlf /f/gfgbf mnlk;dfdlfgdkfgjklsgdhflskdjfhgkdgf;[iwerutwerkejrgeggb,vmf.,v

W  
i  
s  
d  
h  
.  
m  
s  
n  
d  
f  
d  
f  
s  
/  
s  
d  
f  
.  
.  
d  
f  
d  
/  
:  
^  
%  
^  
[  
]  
L  
H  
J  
H  
s  
f  
h  
s  
d  
f  
^  
B  
B  
L



OpenGL

# ***Библиотека OpenGL***

## ***Лекция № 6***

# Практика. Библиотека OpenGL

Open Graphic Library разработана фирмой Silicon Graphics.

Это индустриальный формат, поддерживаемый многими ОС и некоторыми графическими акселераторами.

При создании изображения с помощью OpenGL используется контекст отображения (reading context).

## *Общая схема программы:*

создание окна программы. Здесь необходимо установить стиль окна WS\_CLIPCHILDREN и WS\_CLIPSIBLINGS в CreateWindow.

Открываем контекст отображения (ко).Рекомендуется открытие его во время обработки сообщения WM\_CREATE.

Чтобы создать ко необходимо открыть контекст окна hdc, например функцией GetDC.

Для выяснения характеристик ко устанавливаем соответствующие значения полей структуры PIXELFORMATDESCRIPTOR и вызываем функцию ChoosePixelFormat. Эта функция возвращает номер пиксельного формата.

Вызовом функции SetPixelFormat задаем соответствующий пиксельный формат в hdc.

На основе контекста hdc создаем КО hglrc функцией wglCreateContext. Для переадресации текущего вывода графики OpenGL в hglrc необходимо вызвать функцию wglMakeCurrent.

В ходе работы программы выводим графические объекты в текущий контекст отображения.

Графический вывод можно осуществить во время обработки сообщения WM\_PAINT или других. Для этого используются функции для работы с графическими примитивами OpenGL.

Перед закрытием окна программы необходимо закрыть все открытые КО. Также закрыть все КГУ.

Это можно сделать в ходе обработки сообщения WM\_DESTROY вызовом функции ReleaseDC и wglDeleteContext.

# Библиотека OpenGL

## Особенности Winopgl.cpp

Чтобы использовать библиотеку OpenGL в среде разработки программ на С и С++ необходимо подключить соответствующие файлы:

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>
```

В параметры Link:  
opengl32.lib  
glu32.lib  
glaux.lib

Текст программы представлен в виде двух файлов:  
Winopgl.cpp                      Studex.cpp

### Особенности Winopgl.cpp:

Атрибуты окна WS\_CLIPCHILDREN / WS\_CLIPSIBLINGS в функции CreateWindow.

Инициализация OpenGL при обработке сообщения WM\_CREATE.

Инициализация размеров при обработке сообщения WM\_SIZE.

Инициализация контекста при обработке сообщения WM\_DESTROY.

# Библиотека OpenGL

Описания и функции в Studex.cpp

В OpenGL описаны собственные типы данных, например

GLubyte (беззнаковый байт), GLbyte - байт,  
GLshort (короткое целое), GLint – целое, GLfloat – действительное,  
GLdouble – действительное с двойной точностью,  
GLushort – короткое целое без знака, GLuint – беззнаковое целое.

В окне программы создаются некоторые примитивы. Каждый примитив в OpenGL оформлен парой функций:

```
glBegin(mode);
```

```
... // здесь перечислены варианты объекта  
glEnd();
```

Координаты вершин задаются функцией glVertexXX (XX-2f, 3f),  
glVertex2f (0,300), glVertex3f (0,3,0);



# Библиотека OpenGL

Описания и функции в Studex.cpp

mode:

GL\_POINTS

GL\_LINES

отрезком прямой

GL\_LINE\_STRIP

отрезков

GL\_LINE\_LOOP

GL\_TRIANGLES

GL\_TRIANGLE\_STRIP

GL\_TRIANGLE\_FAN

вершиной

GL\_QUADS

четырёхугольники

GL\_QUAD\_STRIP

GL\_POLYGON

Задание проекции отображения, используя несколько функций.

glOrtho(),

glOrtho2D() - ортографическая (фксометрическая),

gluPerspective() – центральная проекция.

Вывод:

Каждая вершина дается как отдельная точка

Каждая пара вершин соединяется

Полилиния из нескольких связанных

Замкнутая полилиния

Тройки вершин образуют треугольники

Связанные треугольники

Связанные треугольники с общей первой

Каждые четыре вершины образуют

Связанные четырехугольники

Один выпуклый полигон

# Работа с картинками

В библиотеке Gloux описана структура:

```
typedef struct _AUX_RGBImageRec
```

```
{  
    GL int sizeX,sizeY;  
    Unsigned char *data;  
} AUX_RGBImageRec;
```

Опишем переменную этого типа

```
AUX_RGBImageRec *image;
```

Для загрузки используем функцию:

```
Image = auxDIBImageLoad(“photo.bmp”);
```

Выводим на экран:

```
glRasterPos2d(-4.5, -3); // координаты нижнего левого угла
```

```
glPixelZoom(1,1); // шкалирование
```

```
glPixelStorei(GL_UNPACK_ALIGNMENT,1); // сохранить в память
```

```
glDrawPixels(image->sizeX,image->sizeY,GL_RGB,
```

```
GL_UNSIGNED_BYTE,image->data) // вывести на экран
```

# Создание текстуры

```
AUX_RGBImageRec *photo_image;
unsigned int photo_tex;
photo_image = auxDIBImageLoad("photo.bmp");
glGetTextures(1,&photo_tex); // создание идентификатора текстуры
glBindTexture(GL_TEXTURE_2D, photo_image->sizeX,
              photo_image->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
              photo_image->data); // привязываем картинку к текстуре
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_NEAREST);
glEnable(GL_TEXTURE_2D);
glCOLOR(1,1,1);
glBindTexture(GL_TEXTURE_2D, photo_tex);
glBegin(GL_QUADS);
glTexCoord2d(0,0);
glTexCoord2d(0,1);
glTexCoord2d(1,1);
glTexCoord2d(1,0);
glVertex3d(-5,-5,-0.1);
glVertex3d(-5,5,-0.1);
glVertex3d(5,5,-0.1);
glVertex3d(5,-5,-0.1);
glEnd();
```

# Текстуры

## *Текстура на сфере*

```
GLUquadricObj *quadObj;  
quadObj = gluNewQuadric();  
gluQuadricTexture(quadObj, GL_TRUE);  
gluQuadricDrawStyle(quadObj, GLU_FILL);  
glColor3d(1,1,1);  
glRotated(-90,1,0,0);  
gluSphere(quadObj,3,16,16);  
gluDeleteQuadric(quadObj);
```

## *Текстура на произвольном объекте:*

```
glEnable(GL_TEXTURE_2D);  
glEnable(GL_TEXTURE_GEN_S);  
glEnable(GL_TEXTURE_GEN_T);  
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);  
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);  
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);  
auxSolidTeapot(2);
```



# Функции Open GL

**glClearColor(r,g,b, $\alpha$ )** - очистка экрана с заполнением соответствующим цветом.

**glClear(mask)**

**glClearDepth(depth)** – установка начальной глубины.

**glEnable(cap)**

**cap:**    **GL\_DEPTH\_TEST** – включаем Z-buffer

**GL\_COLOR\_MATERIAL** – включает цветность

**GL\_LIGHTING** – включает освещение

**GL\_LIGHT0** – включает первый источник (всего 8)

**glLightf(GL\_LIGHT0, GL\_SPOT\_CUTOFF, angle)**

**gluPerspective(fovy, aspect, zNear, zFar)** – задает параметры центральной проекции

**glMatrixMode(GL\_MODELVIEW)** – указывает какую матрицу нужно преобразовать

**GL\_PROJECTION** – матрица проекций

**GL\_TEXTURE** – для положения текстур

**GL\_MODELVIEW** – видовая матрица

**glLoadIdentity()** – загрузить единичную матрицу

**glTranslatef(x,y,z)** – перенос начала координат

**glRotatef( $\alpha$ ,x,y,z)** – поворот на угол  $\alpha$  вокруг вектора (x,y,z)

**glScalef(x,y,z)** – сжатие/растяжение вдоль осей

# Создание GLU – объектов

```
GLUquadricObj *quadricObj;  
quadricObj = gluNewQuadric();  
if(quadricObj)  
{  
    glPushMatrix();  
    gluSphere(quadricObj,1,16,16);  
    glPopMatrix();  
    gluDeleteQuadric(quadricObj);  
}
```

**GLU – объекты:**

**gluCylinder(q0,R,r,H,slices,stacks);** - цилиндр

**gluSphere(q0,R,slices,stacks);** - сфера

**gluDisk(q0,r,R,slices,stacks);** - диск

**gluPartialDisk(q0,r,R,slices,stacks,A,SA);** - часть диска

# Объекты библиотеки Glaux

**auxSolidSphere(r);**  
**auxSolidBox(x,y,z);**  
**auxSolidTorus(r,R);**  
**auxSolidCylinder(r,H);**  
**auxSolidCone(r,H);**  
**auxSolidTeapot(r);**  
**auxSolidIcosahedron(r);**  
**auxSolidOctahedron(r);**  
**auxSolidCube(r);**  
**auxSolidTetrahedron(r);**  
**auxSolidDodecahedron(r);**