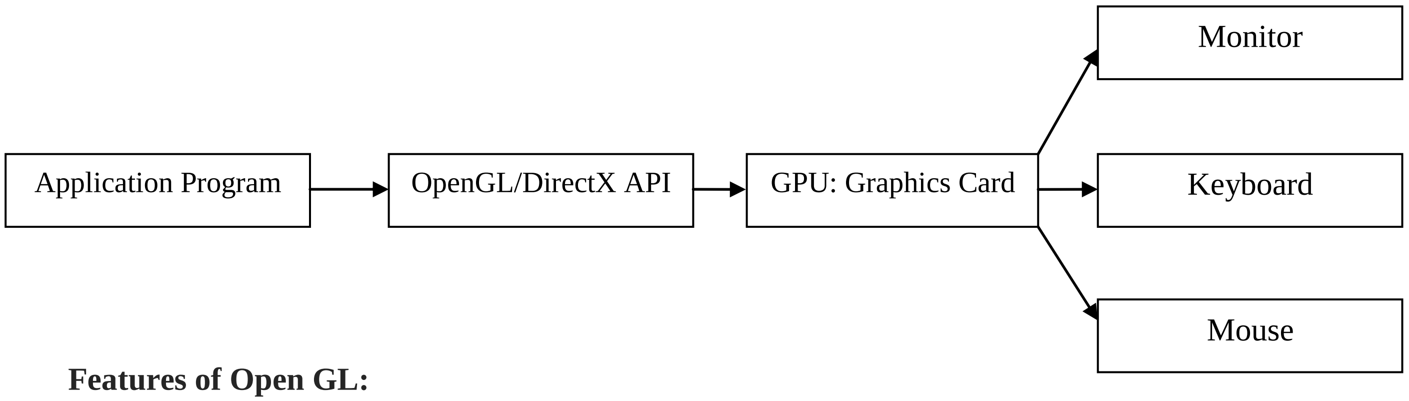
OPENGL

OpenGL is a **software interface** to graphics hardware that consists of **250 distinct commands** (200 in core OpenGL and 50 in OpenGL Utility Library) to produce interactive 3D applications It was designed as **hardware independent interface** to be implemented on different hardware platforms for building models that are built up from a small set of geometric primitives – points, lines, polygons

It is an **operating system and hardware platform independent graphics library** designed to be **easily portable** yet **rapidly executable**. It brings a standard 3D graphics library with the hardware enhanced ability to perform **lighting , shading, texture mapping, hidden surface removal and animation** on to the windows platform

Open GL is available on a **variety of hardware platforms and operating systems**. OpenGL was written with the express intention of becoming a **thin software interface to underlying graphics hardware** an arrangement of proven success in the graphics workstation market.



Features of Open GL:

Texture Mapping : The ability to apply an image to a graphics surface, this technique is used to rapidly generate realistic images without having to specify on excessive amount of detail regarding pixel coordinates, textures etc

Z-Buffering: The ability to calculate the distance from the viewer's location. this makes it easy for the program to automatically remove surface or parts of surfaces that are hidden from view

Double Buffering: Support for smooth animation using double buffering. A smooth animation sequence is achieved by drawing into the back buffer while displaying the front buffer and then swapping the buffers when ready to display the next animation sequence

Lighting Effects: The ability to calculate the effects on the lightness of a surface's color when different lighting models are applied to the surface from one or more light sources

Smooth Shading: The ability to calculate the shading effect that occurs when light hits a surface at an angle and results in subtle color differences across the surface. This effect is important for making model look realistic

Material Properties: The ability to specify the material properties of a surface. These properties modify the lighting effects on the surface by specifying such this as dullness or shininess of the surface

Alpha Blending: The ability to specify alpha or opacity value in addition to the regular red , green, blue values. The alpha component is used to specify opacity, allowing the full range from completely transparent to totally opaque. When used in combination with z buffer, Alpha blending gives the effect of being able to see through objects

Developer's Advantage:

Industry Standard: the Open GL architecture Review board an independent consortium guides the Open GL specification Open GL is the only true open vendor-neutral, multiplatform graphics standard with a broad industry support

Stability: Updating to Open GL specification are carefully controlled and updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure the viability of existing applications

Portability: applications produce consistent visual display result on any OpenGL API compliant hardware regardless of OS or windowing system. So once a program is written for any platform, it can be ported for other platforms as well.

Evolving: New hardware innovations are accessible thru the API via the OpenGL extension mechanism Innovations are phased in to enable developers and hardware vendors to incorporate new features into their product release cycles

Scalability: Open GL applications can be scaled to any class of machine, everything from consumer electronics to PCS, workstations, Super Computers

Ease of Use: Efficient OpenGL routines typically result in applications with fewer lines of code than programs created with other graphics libraries or packages. OpenGL driver encapsulates the information about the underlying hardware so the application programmer does not need to b e concerned about having to design for specific hardware features

History

OpenGL was first created as an **open and reproducible alternative to Iris GL** which had been the proprietary graphics API on Silicon Graphics workstations.

Although OpenGL was initially similar in some respects to IrisGL the lack of a formal specification and conformance tests made Iris GL unsuitable for broader adoption. Mark Segal and Kurt Akeley authored the OpenGL 1.0 specification which tried to formalize the definition of a useful graphics API and made cross platform non-SGI 3rd party implementation and support viable. One notable omission from version 1.0 of the API was texture objects.

IrisGL had definition and bind stages for all sorts of objects including materials, lights, textures and texture environments. OpenGL avoided these objects in favor of incremental state changes with the idea that collective changes could be encapsulated in display lists. This has remained the philosophy with the exception that texture objects (glBindTexture) with no distinct definition stage are a key part of the API.

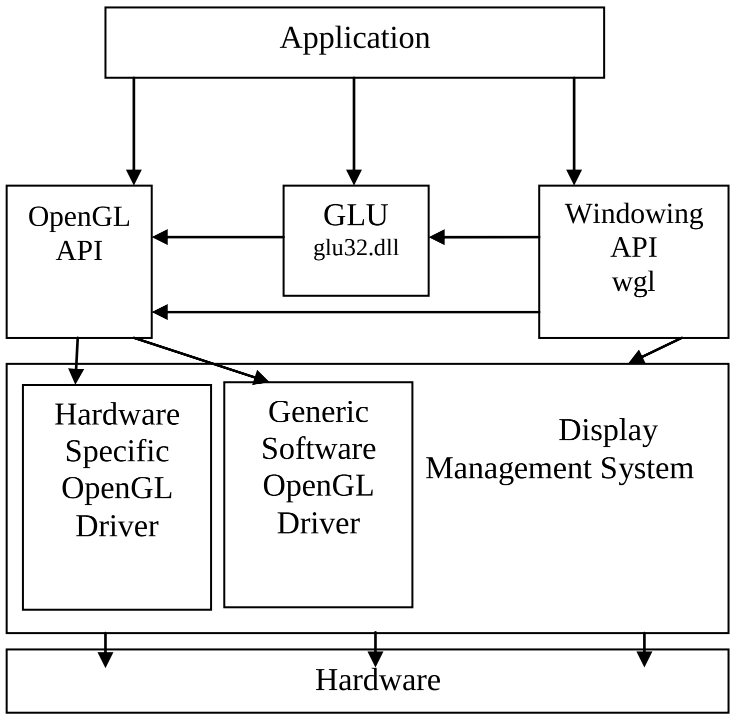
OpenGL has been through a number of revisions which have predominantly been incremental additions where extensions to the core API have gradually been incorporated into the main body of the API. For example OpenGL 1.1 added the glBindTexture extension to the core API.

OpenGL 2.0 incorporates the significant addition of the OpenGL Shading Language (also called GLSL), a C like language with which the transformation and fragment shading stages of the pipeline can be programmed.

OpenGL 3.0 adds the concept of deprecation: marking certain features as subject to removal in later versions. GL 3.1 removed most deprecated features, and GL 3.2 created the notion of core and compatibility OpenGL contexts.

Official versions of OpenGL released to date are 1.0, 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5.

General OpenGL Architecture


Application
OpenGL API
GLU glu32.dll
Windowing API wgl
Hardware Specific OpenGL Driver
Generic Software OpenGL Driver
Display Management System
Hardware

Block Diagram of Typical Application / OpenGL / OS interactions.

OpenGL is the definitive cross-platform 3D library.

- The **Application** module is developed by the programmer
- **Windowing API** functions specifically developed to support OpenGL are used to set up, shut down, and handles **event** signals from an OpenGL drawing area called a Rendering Context (RC) in a window or component . These functions may be defined by the operating system, third party widget set, or by a cross platform OpenGL library.
    - OS Defined Libraries:
        - Windows: WGL functions
        - XWindows (Linux, most UNIXs): GLX functions
        - Mac OS X: AGL, CGL, NSOpenGL classes
    - Cross Platform OpenGL Libraries
        - SDL
        - GLUT (pronounced gloot)
    - Cross Platform Widget Sets with OpenGL support
        - wxWidgets
        - fltk
        - tcl/tk
        - Qt

- **OpenGL API** functions defined in OpenGL.h and provided by the active OpenGL driver (specified by the RC) are used to draw.
    - There are some optional helper functions defined in glu.h which simplify some of the more complex OpenGL tasks.
    - Some of the Special OpenGL libraries also provide special drawing functions for certain shapes or text.
- **OpenGL Drivers** implement the details of OpenGL functions either by passing data directly to 3D hardware, performing computations on the main CPU or some combination of the two.
    - Whether the programmer will render with dedicated hardware or with a software implementation is determined by what OpenGL features the programmer requests with the Windowing API and what the computer's hardware supports.

Model-View-Controller Architecture

Interactive program design involves breaking the program into three parts:

- The ***Model***: all the data that are unique to the program reside there. This might be game state, the contents of a text file, or tables in a database.
- The ***View*** of the ***Model***: it is a way of displaying some or all of the data to user. Objects in the game state might be drawn in 3D, text in the file might be drawn to the screen with formatting, or queries on the database might be wrapped in HTML and sent to a web browser.
- The ***Controller***: the methods for user to manipulate the model or the view. Mouse and keystrokes might change the game state, select text, or fill in and submit a form for a new query.

Object-oriented programming was developed, in part, to aid with modularising the components of Model-View-Architecture programs. Most user interface APIs are written in an Object Oriented language.

Structuring programs to handle these three aspects of an interactive program. A 3D graphics:

- ***Model*** consists of descriptions of the geometry, positioning, apperance and lighting in your scene. Ideally these could be saved to and loaded from a file.
- ***View*** consists of the OpenGL rendering calls that set up your rendering area, interpret the document, and send things to be drawn.

Controller **is usually a set functions you write that respond to** event **signals sent to your program via the Windowing API.**

OpenGL Command Syntax

Uses the prefix `gl` and initial capital letters for each word making up the command name

e.g. `glClearColor(---------); glVertex3f(---------);`

OpenGL defined constants begin with GL and use all capital letters and underscores to separate words e.g. GL_COLOR_BUFFER_BIT

```
#include <necessary header files>
main(){
      InitializeWindow();
      Rendering operations();
      UpdateWindowAndCheckForEvents();
}
```

Data types

Data type	Corresponding C language	OpenGL Type Definition
8 bit integer	signed char	GLbyte
16 bit integer	short	GLshort
32 bit integer	int or long	GLint, GLsizei
32 bit float	float	GLfloat
64 bit float	double	GLdouble

GL Related Libraries

OpenGL Utility Library (GLU): Contains several routines that use lower level OpenGL Commands to perform tasks as setting up matrices, for specific viewing orientations and projections etc. e.g. `gluPerspective(…………..);`

OpenGL Utility Toolkit (GLUT): A window system independent toolkit written by Mark Kilgard to hide the complexities of differing system APIs

OpenGL contains **only rendering commands** and no commands for opening windows or reading events from keyboard or mouse

GLUT provides several routines for opening windows detecting input and creating complicated 3D objects like spehere , torus , teapot

GLUT routines use the prefix glut e.g. `glutCreateWindow(…); glutInit(…);`
`glutMouseFunc(…..); glutKeyboardFunc(…..);`

Windows Management

Five routines perform tasks necessary for initializing a window

`glutInit(int *argc, char **argv)`

- Initializes `glut` and processes any command line arguments

`glutInit();`

- `should be called before any other GLUT routine`

`glutInitDisplayMode(unsigned int mode)`

- specifies whether to use an RGBA or color-index model
- specifies whether to use a single or double buffered window and specify a depth buffer

`glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA|GLUT_DEPTH);`

`glutInitWindowPosition(int x, int y)`

- specifies screen location for the upper loft corner of your window

`glutInitWindowSize(int x, int y)`

- specifies the size in pixels of your window

`glutCreateWindow(char *string)`

- creates a window with the name that you supplies as the string (pointer) variable

The Display Callback

`glutDisplayFunc(void (*func)(void))`

- The first and most important event callback function
- Whenever GLUT determines that the contents of the window need to be redisplayed the call back function registered by `glutDisplayFunc()` is executed
- All the routines needed to redraw the scene are placed here

`glutMainLoop()`

- This is an infinite loop
- All the windows that have been created are now shown infinitely

Handling input events

Use the following routines to register callback commands that are invoked when specified events occur

`glutReshapeFunc(void (*func)(int w, int h))`

- Indicates what action should be taken when the window is resized

`glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))`

```
glutMouseFunc(void (*func)(int button, int state, int x, int y))
```

- Allow you to link a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released

Managing a background process

```
glutIdleFunc((*func)(void))
```

- Used to specify a function that's to be executed if no other events are pending
- This routine takes a pointer to a function as its only argument

Examples

```
#include <windows.h>
#include <gl/glut.h>
#include <gl/glu.h>
void display(){
}
void main(int argc, char **argv){
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(100,100);
 glutCreateWindow("saroj");
 glutDisplayFunc(display);
 glutMainLoop();
}
//CLEARING THE BACKGROUND
void display(){
 glClearColor(1.0,0.0,1.0,1.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glFlush();
}
```

```
// PLOTTING A POLYGON:  ADDED INSIDE
DISPLAY FUNCTION
glShadeModel(GL_SMOOTH);
 glBegin(GL_POLYGON);
  glVertex3f(0.25,  0.25, 0.0);
  glVertex3f(0.75,  0.25, 0.0);
  glVertex3f(0.75,  0.75, 0.0);
  glVertex3f(0.25,  0.75, 0.0);
 glEnd();

//ADDED TO DRAW A POLYGON
void display(){
  glClearColor(1.0,0.0,1.0,1.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glShadeModel(GL_FLAT);
  glColor3f(0.0,1.0,0.0);
// TO SPECIFY FILL COLOR
  glBegin(GL_POLYGON);
    glVertex3f(0.25,  0.25, 0.0);
    glVertex3f(0.75,  0.25, 0.0);
    glVertex3f(0.75,  0.75, 0.0);
    glVertex3f(0.25,  0.75, 0.0);
  glEnd();
 glFlush();
}
```

More Examples

To create a window

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <gl/glu.h>
void display(){

}
```

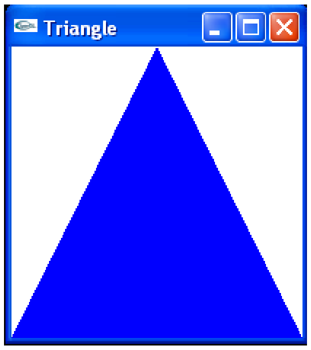
```
void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowPosition(111,111);
  glutInitWindowSize(200,100);
  glutCreateWindow("dsaroj");
  glutDisplayFunc(display);
  glutMainLoop();}
```

To clear the background to a desired color, add these code to the display function

```
void display(){
 glClearColor(1.0,0.0,1.0,1.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glFlush();
}
```

Drawing a Triangle

```
static GLfloat x = 0.15;
void display(){
 glClearColor(1.0,1.0,1.0,1.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(0.0,0.0,1.0);  //fill color
  glBegin(GL_POLYGON);
    glVertex3f(0.0,0.0,0.0);
    glVertex3f(50.0,0.0,0.0);
    glVertex3f(25.0,50.0,0.0);
  glEnd();
glFlush();
}
```



```
void reshape(int w, int h){
 glViewport(0,0,w,h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glOrtho(0,50,0,50,-1,1);
 //glOrtho(left,right,bottom,top,near,far);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
}
```

```
void main(int argc, char **argv){
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowPosition(200,200);
 glutInitWindowSize(200,200);
 glutCreateWindow("Triangle");
 glutDisplayFunc(display);
 glutReshapeFunc(reshape);
 glutMainLoop();
}
```

`GL_PROJECTION` : Applies subsequent matrix operations to the projection matrix stack.

`GL_MODELVIEW` : Applies subsequent matrix operations to the modelview matrix stack.

Getting input from a Mouse

```
GLfloat x = 0.15;
void display(){
 glClearColor(1.0,1.0,1.0,1.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(0.0,0.0,1.0);
  glBegin(GL_POLYGON);
    glVertex3f(0.25+x,0.25,0.0);
    glVertex3f(0.75+x,0.25,0.0);
    glVertex3f(0.75+x,0.75,0.0);
  glEnd();
 glFlush();
}
void trans(){
  x = x-0.001;
  glutPostRedisplay();
}
void mouse(int button, int state, int x, int y){
   switch(button){
       case GLUT_LEFT_BUTTON:
         if (state == GLUT_DOWN)
           glutIdleFunc(trans);
        break;
       case GLUT_RIGHT_BUTTON:
         if (state == GLUT_DOWN)
           glutIdleFunc(NULL);
        break;
}
void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowPosition(111,111);
  glutInitWindowSize(200,100);
  glutCreateWindow("saroj");
  glutDisplayFunc(display);
  glutMouseFunc(mouse);
  glutMainLoop();}
```

Getting input from a Keyboard

```
GLfloat x = 0.15;
void display(){
 glClearColor(1.0,1.0,1.0,1.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(0.0,0.0,1.0);
    glBegin(GL_POLYGON);
        glVertex3f(0.25+x,0.25,0.0);
        glVertex3f(0.75+x,0.25,0.0);
        glVertex3f(0.75+x,0.75,0.0);
    glEnd();
 glFlush();
}
```

```
void trans(){
  x = x-0.001;
  glutPostRedisplay();
}
void key(unsigned char key, int x, int y){
    switch(key){
       case 'a':
         glutIdleFunc(trans);
        break;
       case 'f':
         glutIdleFunc(NULL);
        break;
      }
}
```

To rotate a triangle

glRotatef(type angle, type x, type y, type z);

multiplies current matrix by a matrix that rotates an object in counter clockwise direction

glTranslatef(type x, type y, type z);

multiplies current matrix by a matrix that translates an object by th e give x, y , z values

glScalef(type x, type y, type z);

multiplies current matrix by a matrix that stretches or shrinks an object

Smooth Shading (Intensity Interpolation Scheme)

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
#include <stdlib.h>
#include <math.h>
```

Draw triangle with three vertices with different colors

```
void display(){
glClearColor(0.0,0.0,0.0,0.0);
//glShadeModel(GL_FLAT);       //CIS
glShadeModel(GL_SMOOTH);  //Gouroud Shading
glClear(GL_COLOR_BUFFER_BIT);
glutWireSphere(1.0,20,16);
glBegin(GL_TRIANGLES);
     glColor3f(0.0,1.0, 0.0); //green
     glVertex2f(25.0,  5.0);

     glColor3f(0.0,0.0, 1.0); //blue
     glVertex2f(5.0,  25.0);

     glColor3f(1.0,0.0, 0.0); //red
     glVertex2f(5.0,  5.0);
 glEnd();
 glFlush();
}
```

Define Coordinate System and create view port

```
void reshape(int w,int h){
 glViewport(0,0,(GLsizei)w,(GLsizei)h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
  if(w<= h)
    gluOrtho2D(0.0,250.0,0.0,250.0);
  else
    gluOrtho2D(0.0,30.0,0.0,30.0*(GLfloat)h/(GLfloat)w);
 glMatrixMode(GL_MODELVIEW);

}
```

```
int main(int argc, char* argv[]){
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
   glutInitWindowPosition(100,100);
   glutInitWindowSize(600,600);
   glutCreateWindow("Intensity Interpolation Scheme");
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
 return 0;
}
```

//Animation using Double Buffer

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
static GLfloat spins= 0.0;

//Create a rectangle
void display(){
  glClearColor(0.0,0.0,0.0,0.0);
  glShadeModel(GL_FLAT);
  glClear(GL_COLOR_BUFFER_BIT);
  glPushMatrix();
       glRotatef(spins,0.0,0.0,1.0);
       glColor3f(1.9,1.0,0.8);
       glRectf(-25.0,-25.0,25.0,25.0);
  glPopMatrix();
  glutSwapBuffers();
}
```

//Function for changing the value of spins

```
void spin(){
   spins = spins + 2;
    if( spins > 360.0 )
        spins = spins - 360.0;
    glutPostRedisplay();
}
```

//Trap mouse click event

```
void mouse(int button,int state, int x, int y){
    switch(button){
      case GLUT_LEFT_BUTTON:
         if(state == GLUT_DOWN)
      glutIdleFunc(spin);     //NULL to Pause
      break;
      }
}
```

//Specify coordinate system

```
void reshape(int w, int h){
 glViewport(0,0,(GLsizei)w,(GLsizei)h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glOrtho(-50.0,50.0,-50.0,50.0,-1.0,1.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
}
void main(int argc, char* argv[ ]){
   glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(100,100);
   glutCreateWindow("Animation");
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMouseFunc(mouse);
   glutMainLoop();
}
```

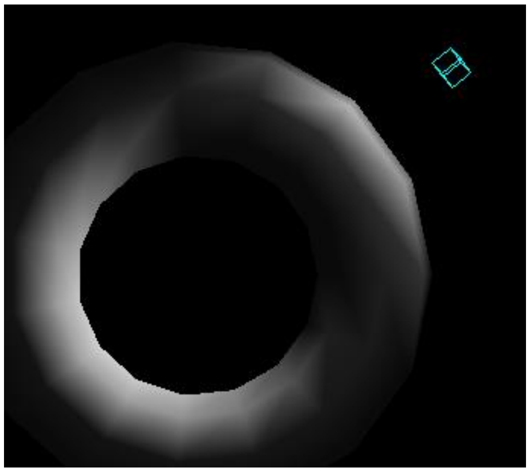
Font Generation

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <gl/glu.h>
GLubyte rasters[24] = {
0xc0,0x00, 0xc0,0x00, 0xc0,0x00, 0xc0,0x00, 0xc0,0x00,
0xff,0x00, 0xff,0x00, 0xc0,0x00, 0xc0,0x00, 0xc0,0x00,
0xff,0xc0, 0xff,0xc0,
};
void init(){
 glPixelStorei(GL_UNPACK_ALIGNMENT,1);
 glClearColor(0.0,0.0,0.0,0.0);
}

void display(void){
 glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1,1,1);
glFlush();
}
void show(){
glRasterPos2i(20,20);
glBitmap(10,12,0,0,11,0,rasters);
glBitmap(10,12,0,0,11,0,rasters);
glBitmap(10,12,0,0,11,0,rasters);
glFlush();
}
void reshape(int w, int h){
  glViewport(0,0,(GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0, w,0 , h, -1.0,1.0);
  glMatrixMode(GL_MODELVIEW);
}

void keyboard(unsigned char key, int x, int y){
    switch(key){
      case 'f':
        show();
        case 'a':
              exit(0);
}
 }

void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
}
```

Creating a lighting effect

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <gl/glu.h>
static int spin = 0;

void init(){
 glClearColor(0.0,0.0,0.0,0.0);
 glShadeModel(GL_SMOOTH);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glEnable(GL_DEPTH_TEST);
}

void display(){
GLfloat position[]={0.0,0.0,1.5,1.0};
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glPushMatrix();
      glTranslatef(0.0,0.0,-5.0);
      glPushMatrix();
           glRotated ((GLdouble)spin,1.0,0.0,1.0);
           glLightfv(GL_LIGHT0,GL_POSITION,position);
           glTranslated(0.0,0.0,1.5);
           glDisable(GL_LIGHTING);
```

```
          glColor3f(0.0,1.0,1.0);
          glutWireCube(0.1);
         glEnable(GL_LIGHTING);
     glPopMatrix();
          glutSolidTorus(0.275,0.85,8,15);
 glPopMatrix();
 glFlush();
}

void reshape(int w, int h){
  glViewport(0,0,(GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0,(GLfloat) w / (GLfloat)h, 1.0,20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

}
void mouse(int button, int state, int x, int y){
    switch(button){
      case GLUT_LEFT_BUTTON:
         if (button == GLUT_DOWN){
      spin = (spin + 30) % 360;
      glutPostRedisplay();
         }
         break;
         default:
            break;
    }
 }

void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutMainLoop();
}
```



```
glLightfv(GL_LIGHT0,GL_POSITION,position):
```

- Set light-source parameters.

```
void glLightfv(
  GLenum light,
  GLenum pname,
  const GLfloat *params
);
```

light

The identifier of a light

GL_LIGHTi where $0 \leq i <$ GL_MAX_LIGHTS

At least eight lights are supported.

pname

A single-valued light-source parameter for *light*

param

The value to which parameter *pname* of light source *light* will be set.