Exercise: Introduction to OpenGL Programming

This exercise is an introduction to the usage of the OpenGL graphics standard in the context of Windows application programming. As development environment Visual C++ is used. OpenGL is platform independent and can be run on a variety of operating systems. OpenGL commands are transfered to the graphics processing unit (GPU) and carried out by the graphics hardware. In order to provide functionality for drawing graphics output into a window (viewport) and for event handling we include the OpenGL Utility Toolkit (GLUT or FreeGLUT).

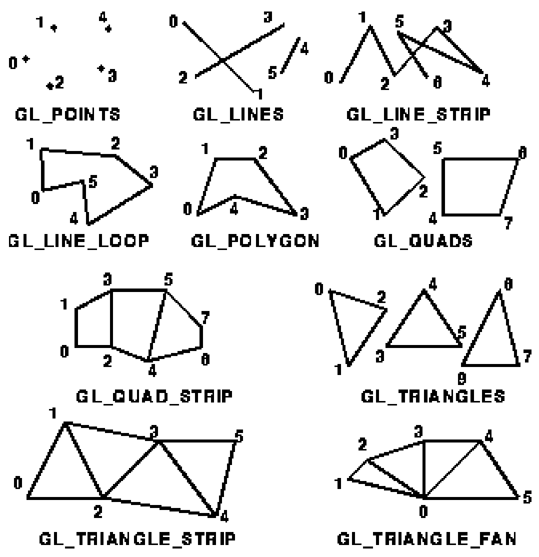
Building an OpenGL application with Visual C++

A detailed desription of defining an OpenGL project, compiling and debugging it with Visual Studio is given in the document *App_Programming_Env_VS*.pdf*.

OpenGL Functions for Drawing Primitives and Assigning Colours

OpenGL Drawing Primitives

OpenGL supports several basic primitive types, including points, lines, quadrilaterals, and geneneral polygons. All of these primitives are specified using a sequence of vertices. The diagram below shows the basic primitive types, where the numbers indicate the order in which the vertices have been specified. Note that for the `GL_LINES` primitive only every second vertex causes a line segment to be drawn. Similarly, for the `GL_TRIANGLES` primitive, every third vertex causes a triangle to be drawn. Note that for the `GL_TRIANGLE_STRIP` and `GL_TRIANGLE_FAN` primitives, a new triangle is produced for every additional vertex. All of the closed primitives shown below are solid-filled, with the exception of `GL_LINE_LOOP`, which only draws lines connecting the vertices.



The following code fragment illustrates an example of how the primitive type is specified and how the sequence of vertices are passed to OpenGL. It assumes that a window has already been opened and that an appropriate 2D coordinate system has already been established.

```
// draw several isolated points

GLfloat pt[2] = {3.0, 4.0};
glBegin(GL_POINTS);
glVertex2f(1.0, 2.0);     // x=1, y=2
glVertex2f(2.0, 3.0);     // x=2, y=3
glVertex2fv(pt);          // x=3, y=4
glVertex2i(4,5);          // x=4, y=5
glEnd();
```

The following code fragment specifies a 3D polygon to be drawn, in this case a simple square. Note that in this case the same square could have been drawn using the GL_QUADS and GL_QUAD_STRIP primitives.

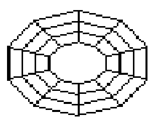
```
GLfloat p1[3] = {0,0,1};
GLfloat p2[3] = {1,0,1};
GLfloat p3[3] = {1,1,1};
GLfloat p4[3] = {0,1,1};

glBegin(GL_POLYGON);
glVertex3fv(p1);
glVertex3fv(p2);
glVertex3fv(p3);
glVertex3fv(p4);
glEnd();
```

Predefined Objects

The OpenGL utility library GLU provides a variety of predefined objects, such as sphere, cube, cylinder. They are called quadrics. With the command *gluNewQuadric* a new object is initialized. Its shape has to be defined. Examples are:

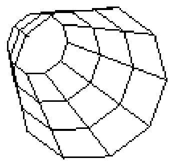
gluDisk (quadric, Ri, Ro, sec, circ)
defines a disc with inner radiusz *Ri*, outer radius *Ro*, *sec* sectors and *circ* circles.

```
   GLUquadricObj* newquadric=gluNewQuadric();
        gluQuadricDrawStyle(newquadric, GLU_LINE);
        gluDisk(newquadric, 0.2, 0.5, 10, 3);
```



gluCylinder (quadric, R0, R1, l, sec, seg)
defines a cylinder with end radii *R0* and *R1*, length *l*, *sec* sectors and *seg* subdivisions along the z-axis.

```
   GLUquadricObj* newquadric=gluNewQuadric();
        gluQuadricDrawStyle(newquadric, GLU_LINE);
        gluCylinder(newquadric, 0.2, 0.5, 1, 10, 3);
```



gluSphere (quadric, radius, lo, la)
defines a sphere with *lo* lines of longitude und *la* lines of latitude.

```
    GLUquadricObj* newquadric=gluNewQuadric();
         gluQuadricDrawStyle(newquadric, GLU_FILL);
         gluSphere(newquadric, 0.5, 15, 15);
```

Assigning Colours

OpenGL maintains a current drawing colour as part of its state information. The `glColor()` function calls are used to change the current drawing colour. assigned using the glColor function call. Like `glVertex()`, this function exists in various instantiations. Colour components are specified in the order of red, green, blue. Colour component values are in the range [0...1], where 1 corresponds to maximum intensity. For unsigned bytes, the range corresponds to [0...255]. All primitives following the fragment of code given below would be drawn in green, assuming no additional `glColor()` function calls are used.

```
    GLfloat mycolour[3] = {0,0,1}; // blue
    glColor3fv( mycolour );        // blue using vector of floats
    glColor3f(1.0, 0.0, 0.0);      // red using floats
    glColor3ub(0,255,0);           // green using unsigned bytes
```

If desired, a polygon can be smoothly shaded to interpolate colours between vertices. This is accomplished by using the GL_SMOOTH shading mode (the OpenGL default) and by assigning a desired colour to each vertex, as shown in the following example.

```
      glShadeModel(GL_SMOOTH);    // as opposed to GL_FLAT
      glBegin(GL_POLYGON);
      glColor3f(1.0, 0.0, 0.0);   // red
      glVertex3f(0.0, 0.0, 0.0);
      glColor3f(0.0, 1.0, 0.0);   // green
      glVertex3f(1.0, 0.0, 0.0);
      glColor3f(0.0, 0.0, 1.0);   // blue
      glVertex3f(1.0, 1.0, 0.0);
      glEnd();
```

A fourth value called *alpha* is often appended to the colour vector. This can be used assign a desired level of transparency to a primitive and finds uses in compositing multiple images together. An alpha value of 0.0 defines an opaque colour, while an alpha value of 1.0 corresponds to complete transparency.

The screen can be cleared to a particular colour as follows:

```
    glClearcolor(1.0, 1.0, 1.0, 0.0);   // sets the clear colour to white 
and opaque
    glClear( GL_COLOR_BUFFER_BIT);      // clears the colour frame buffer
```

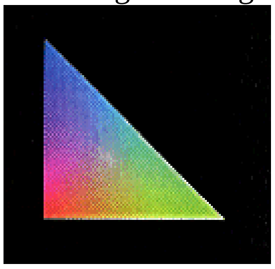
Preparation

1. Make yourself familiar with the OpenGL graphics standard.
2. Reconsider your knowledge of Windows application programming, particularly the subjects C++ and VisualStudio.
3. Answer the following questions:

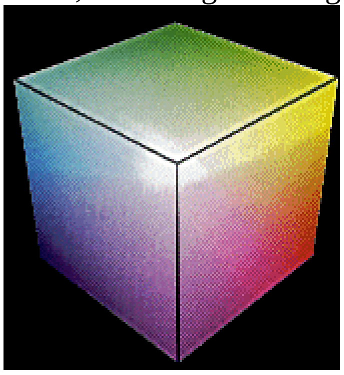
1. What does the notion *state-machine* mean?
2. Which functionality do the additional libraries GLUT and GLU provide?
3. Which company authored OpenGL?
4. What does an OpenGL driver, which usually is installed together with a graphics board, do?
5. Which graphics APIs can be used as an alternative to OpenGL?

Exercises

1. Write a program drawing a triangle filled with a solid colour.
2. Change your program, so that the triangle is smoothly shaded between 3 colours according to the figure



3. Define a perspective view of a cube in 2D with 3 quads. Write a program which draws a colour cube, interpolating the colours red, green, blue, cyan, magenta, yellow and white, according to the figure



4. Use the Visual C++ debugger to debug your programs.