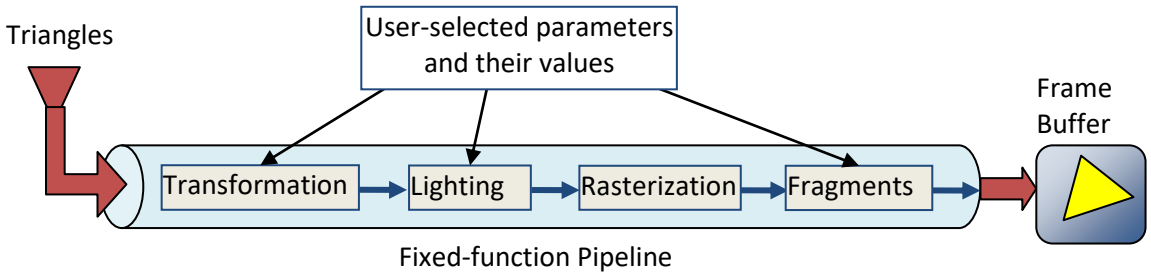
# 2. OpenGL Basics

*R. Mukundan*,
Department of Computer Science and Software Engineering
University of Canterbury.

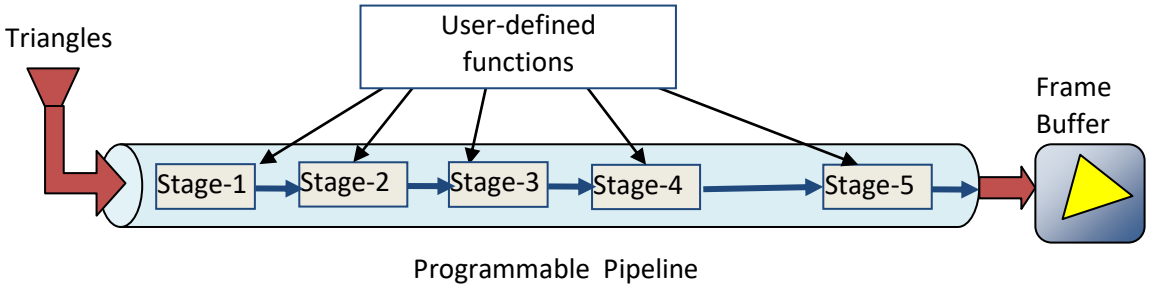
## 2.1 Traditional vs. Modern

OpenGL is a rendering library designed to provide a low-level application programmer's interface (API) with the graphics hardware. We can find two distinct types of programming models for rendering a scene in OpenGL: a model that uses the *fixed-function pipeline* and another that uses a more advanced *shader based programmable pipeline*. A rendering pipeline is nothing but a sequence of computations performed on the graphics hardware in generating the display of a three-dimensional scene comprising of a number of input primitives.

The fixed-function pipeline was supported by the early versions of OpenGL (v1, 2). It is also retained as a "compatibility profile" in later versions (v3, 4). This is a simple model that maps a set of dedicated fixed logic to various stages of the graphics processor. For example, in OpenGL v1, 2, the lighting model was hardcoded, and the user could only set the values of various parameters used for lighting calculations.




Fixed-function Pipeline




Programmable Pipeline

The programmable pipeline model replaced the built-in operations performed in the hardware with programmable stages. While this model allowed a great deal of flexibility to the user in defining own functions and in executing them directly on the hardware, it required a lot more coding effort than the previous model. In this model, it is necessary to develop separate code segments (shaders) for different stages, specifying the complete set of operations to be performed at each stage. The five shader

stages are the vertex shader, tessellation control shader, tessellation evaluation shader, geometry shader and the fragment shader.

Though considered obsolete, the fixed function pipeline provides an ideal framework for beginners to quickly learn the foundations of computer graphics and to gain a good understanding of the operations performed in the rendering pipeline, what it can do and what it cannot. The concepts related to the programmable pipeline and shader based implementations are introduced in the second half of the course.

## 2.2 A Simple OpenGL Program

A very basic structure of an OpenGL program written in C language is shown below. It uses the freeglut library (freeglut.sourceforge.net) for creating and managing display windows using the native windowing system. The header file `freeglut.h` is stored in the GL subfolder of the default include directory. Freeglut implements the GL Utility Toolkit (GLUT).

The sample OpenGL program consists of three parts (functions):

- The **`main()`** function uses glut functions to create a window named "Teapot" of size 600x600 pixels at position (0,0) on the screen. This position is the top-left corner of the screen. The function `glutInitDisplayMode()` is used to select a single colour buffer and a depth buffer. A set of such buffers used for generating the display is collectively called a *frame buffer*. The function `glutDisplayFunc()` registers a user-defined function `display()` as the function where all drawing commands to the current window are issued. The function `display()` then becomes a *callback* for the current window. That is, if the current display needs to be refreshed at any stage, this function will be called. Glut has a repeating event processing loop where such display events and other types of events (generated by input devices, timers etc.) are continuously processed. `glutMainLoop()` causes the process to enter this infinite loop (once called, the function will never return).

- The **`initialize()`** function is commonly used to define the initial values of various parameters and to enable the required OpenGL states. Data and images from files may also be loaded here. In this example, lighting, colour material and depth test features are enabled. It also sets the camera frustum (more about this later).

- The **`display()`** function was registered in `main()` as the display callback. This function contains all drawing commands. It clears the buffers, positions the camera and the light source, and draws a glut built-in object, the teapot. The next section describes the built-in objects in the glut library.

```
#include<GL/freeglut.h>

//--This is the main display module specifying the drawing functions
void display(void)
{
 float light_pos[4] = {0., 10., 10., 1.0};  //light's position

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

gluLookAt(0, 0, 12, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);  //Camera position
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);   //Set light's position

 glColor3f(0.0, 1.0, 1.0);   //Set the colour value
glutSolidTeapot(1.0);       //Draw a teapot
glFlush();
}

//-----------------------------------------------------------------------
void initialize(void)
{
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glEnable(GL_LIGHTING);             //Enable OpenGL states
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_DEPTH_TEST);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-5.0, 5.0, -5.0, 5.0, 10.0, 1000.0);   //Camera Frustum
}


//  ------- Main: Initialize glut window and register call backs -------
int main(intargc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_DEPTH);
glutInitWindowSize(600, 600);
glutInitWindowPosition(0, 0);
glutCreateWindow("Teapot");
 initialize();
glutDisplayFunc(display);
glutMainLoop();
 return 0;
}
```

## 2.3 Glut Built-in Objects

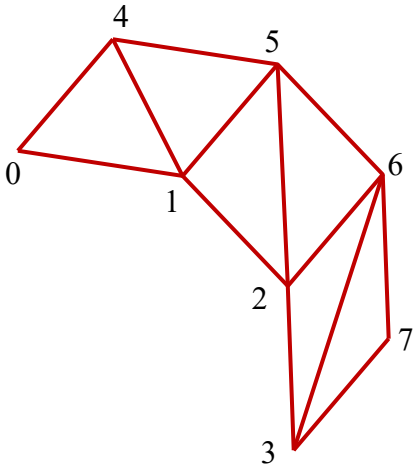
The GLUT (or freeglut) library contains a set of built-in models that can be easily added to a three-dimensional scene. The objects are always created at the origin of the coordinate reference frame. Each object has a set of parameters defining its shape and the number of subdivisions it should have. For a description of the functions and their parameters, please refer to the file GLUT-GLU-Objects.pdf (in Lab01).

There are a few important geometrical properties of GLUT objects that are worth noting here. The cone does not include its base, and is therefore a hollow cone. The teapot is generated using Bezier patches, and has clockwise winding of vertices which often causes problems when applications use the back-face culling feature. The five solids Tetrahedron, Cube, Octahedron, Dodecahedron and Icosahedron are called *platonic solids*. These are regular polyhedrons which form approximations of the sphere consisting of equilateral and equiangular faces. Only 5 platonic solids exist. The cube has 8 vertices and 6 faces, while the octahedron has 6 vertices and 8 faces. The cube and the octahedron are therefore called *dual surfaces*. Similarly, the dodecahedron (20 vertices, 12 faces) is the dual of the icosahedron (12 vertices, 20 faces).

The GLU library contains two useful object models: the cylinder and the disk. The cylinder model has a general representation that allows different radii to be defined for the base and the top. Thus we can use the model to create tapered cylinders or cones. The disk model is often used as the base for cylinders and cones.

## 2.4 3D Mesh Objects

The geometrical structure of a mesh object can be conveniently stored in two tables: the vertex table and the polygon table. The vertex table is a simple list of vertices with the positional information of each vertex stored as $x$, $y$, $z$ coordinate values. The polygon table gives the definitions of each polygon in terms of vertex indices. A vertex index is an integer value that is used as the array index for retrieving the coordinates from the vertex table. After vertex coordinates and polygon definitions, the next most important information associated with mesh data is the normal definition. Normal vectors are required for lighting computation. They may be attached to polygonal faces or vertices. The normal vectors for each triangle can be computed inside the rendering application using the data stored in vertex and polygon tables. However, some mesh files contain a separate "normal table" of pre-computed normal vector components which can simply be read-in by an OpenGL application.




Vertex Table

| | x | y | z |
|---|---|---|---|
| 0 | 0.0 | 1.75 | 0.5 |
| 1 | 1.0 | 1.75 | 0.5 |
| 2 | 1.6 | 1.0 | 0.5 |
| 3 | 1.6 | 0.0 | 0.5 |
| 4 | 0.0 | 1.75 | −0.5 |
| 5 | 1.0 | 1.75 | −0.5 |
| 6 | 1.6 | 1.0 | −0.5 |
| 7 | 1.6 | 0.0 | −0.5 |

Polygon Table

| i | j | k |
|---|---|---|
| 0 | 1 | 4 |
| 4 | 1 | 5 |
| 1 | 2 | 5 |
| 5 | 2 | 6 |
| 2 | 3 | 6 |
| 6 | 3 | 7 |

An example showing a simple mesh segment and the corresponding mesh data.

Complex mesh models with very large polygon counts are usually stored in compressed and binary formats to save file space. Simple mesh geometries on the other hand can be stored in ASCII formats, where the files can be easily viewed using commonly available text editors. Some of the popular ASCII file formats are OBJ, OFF and PLY. The Object (.OBJ) format was developed by Wavefront technologies. This format allows the definition of vertices in terms of either three-dimensional Cartesian coordinates or four-dimensional homogeneous coordinates. Polygons can have more than three vertices. In addition to a basic set of commands supporting simple polygonal mesh data, the .OBJ format also supports a number of advanced features such as grouping of polygons, material definitions and the specification of free-form surface geometries including curves and surfaces.

The Object File Format (.OFF) is another convenient ASCII format for storing 3D model definitions. It uses simple vertex-list and face-list structures for specifying a polygonal model. Unlike the .OBJ format, this format does not intersperse commands with values on every line, and therefore can be easily parsed to extract vertex coordinates and polygon indices. This format also allows users to specify vertices in homogeneous coordinates, faces with more than three vertex indices, and optional colour values for every vertex or face.

The Polygon File Format (.PLY) also organises mesh data as a vertex list and a face list with several optional elements. The format is also called the Stanford Triangle Format. Elements can be assigned a type (int, float, double, uint etc.). This file format supports several types of elements and data, and the complete specification is included in the file header. Parsing a PLY file is considerably complex than parsing an OBJ or OFF file. The PLY format was developed in the 90s by the Stanford Graphics Lab for storing models created using 3D scanners. One such popular model is the "Stanford Bunny" consisting of about 69000 triangles.

The following page provides a few links to websites where you could find downloadable 3D models in OFF/OBJ/PLY/3DS formats. Please make sure that the downloaded models are royalty-free.

## Websites containing 3D mesh models

http://www.aimatshape.net/ontologies/shapes/

http://www.holmes3d.net/graphics/offfiles/

http://nasa3d.arc.nasa.gov/models

http://graphics.stanford.edu/data/3Dscanrep/

http://archive3d.net/

http://www.amazing3d.com/

http://www.baument.com/restore/archives.html

http://www.buckrogers.demon.co.uk/3d/3d.htm

http://www.cmlab.csie.ntu.edu.tw/~robin/courses/cg03/model/

http://www.3dlinks.com/links.cfm?categoryid=9&subcategoryid=91

http://www.turbosquid.com/Search/3D-Models/free

http://tf3dm.com/

http://www.3dmodelfree.com/

http://www.the3dstudio.com/3d-models

http://www.3dtotal.com/

http://www.3dxtras.com/3dxtras-free-3d-models-list.asp?catid=-1

http://www.cgtrader.com/free-3d-models

http://www.sweethome3d.com/freeModels.jsp

http://www.creativecrash.com/3dmodels

http://www.3dcontentcentral.com/default.aspx

http://www.123dapp.com/3d-model-library

http://shape.cs.princeton.edu/benchmark/index.cgi

http://www.3dm3.com/modelsbank/

http://www.3dvia.com/search/?search%5Bfile_types%5D=1

http://www.3dcadbrowser.com/

http://graphics.cs.williams.edu/data/meshes.xml