JSPM'S JAYAWANTRAO SAWANT COLLEGE OF ENGINEERING

Department of Computer Engineering

Second year Engineering

(SEM 1 - A.Y. 2020 – 2021)

210248: OOP and Computer Graphics Laboratory

PART 2: Computer Graphics Laboratory (CGL)

COMPUTER GRAPHICS

MINI PROJECT REPORT

Name:	**JOEL SILAS**
Class:	**SE (A.Y. 2020-2021)**
Division:	**B**
Roll no:	**2259**

Depiction of Sunrise and Sunset with motion control in OpenGL

Animation, simulation and other virtual real world experiences are now possible with the use of some recent graphics softwares.

They can create prototypes, models, study graphs and testing environments with reusability to create a working model as per application.

Real world experiences are now possible from any corner of this world.

By a similar intention this project can implement an animation clip of the environmental sunrise and sunset using the OpenGL language which is cost effective.

The atmospheric changes during sunrise and sunset and its effect on the scattering of light phenomenon can be perceived by this animation.

There is extensive use of Computer graphics concepts of 2D & 3D geometrical shapes, light and colour models, shading algorithms, 3D clipping and viewing transformations along with primary C++ programming skills in this animation.

CONTENTS

♦ACKNOWLEDGEMENTS♦

First and foremost, I would like to thank GOD ALMIGHTY for bringing this opportunity in my life to create this project.

I express my sincere gratitude to Prof. Mrinali M. Bhajibhakre who motivated me to develop this project.

I also thank her for all the support that she has provided me in order to bring about my creative, technical and analytical abilities in this project.

I thank my parents for being a continuous source of encouragement .

----------Abstract----------

Hundreds of years before the introduction of true graphics and animation, people from all over the world enjoyed shows with moving figures that were created and manipulated manually in puppetry, automata, shadow play and the magic lantern.

Cinematography eventually broke through in 1895 after animated pictures had been known for decades, the wonder of the realistic details in the new medium was seen as its biggest accomplishment.

Animations on film was not commercialized until a few years later by manufacturers of optical toys, with chromolithography film loops. The enormous success of Mickey Mouse is seen as the start of the golden age of **American animation**.

With introduction of computers by **Charles Babbage** and contributions in graphical user interfaces, Graphics processing units, computer graphics, animation softwares, special fantasy effects and simulations became possible.

Similarly, the advent of OpenGL API and its released versions by the **Khronos Group** along with its other extensions has uplifted the graphics rendering domain which has **trigerred the idea behind this project.**

----------Introduction----------

Animation is a method in which figures are manipulated to appear as moving images. In traditional animation, images are drawn or painted by hand on transparent celluloid sheets to be photographed and exhibited on film. Today, most animations are made with computer-generated imagery (CGI).

Computer animation can be very detailed 3D animation, while 2D computer animation can be used for stylistic reasons, low bandwidth or faster real-time renderings. Other common animation methods apply a stop motion technique to two and three-dimensional objects like paper cutouts, puppets or clay figures.

Commonly the effect of animation is achieved by a rapid succession of sequential images that minimally differ from each other.

Animation:

- It covers any change of appearance of any visual effect that is time based. It time based manipulation of a target element .
- It defines a mapping of the time to values for the target attribute. It includes change of position, transparency, time varying changes in shape & even changes of the rendering techniques.
- Animation classifies into 2 types viz. Frame animation and Sprite animation

The illusion—as in motion pictures in general—is thought to rely on **the phi phenomenon and beta movement**, but the exact causes are still uncertain.

Analog mechanical animation media that rely on the rapid display of sequential images include the phénakisticope, zoetrope, flip book, praxinoscope and film. Television and video are popular electronic animation media that originally were analog and now operate digitally. For display on the computer, techniques like animated GIF and Flash animation were developed.

Animation is more pervasive than many people realize. Apart from short films, feature films, television series, **animated GIFs and other media dedicated to the display of moving images, animation is also prevalent in video games, motion graphics, user interfaces and visual effects.**

The physical movement of image parts through simple mechanics—in for instance moving images in magic lantern shows—can also be considered animation. The mechanical manipulation of three-dimensional puppets and objects to emulate living beings has a very long history in automata. Electronic automata were popularized by Disney as animatronics.

OpenGL is a complex API which has many utilities and functionalities to support these phenomena.

OpenGL program to describe Sunrise and Sunset

Title	Implementation of simulation using OpenGL utilities
Aim/Problem Statement	To depict Sunrise and Sunset with motion control effects in OpenGL
CO Mapping	CO2: Define the concept of windowing and clipping and apply various algorithms to fill and clip polygons in 2D & 3D. CO3: Explain the core concepts of computer graphics in two and three dimensions as in viewing and projection. CO4: Explain the concepts of colour models, lighting, shading models and hidden surface elimination. CO5: Describe the concepts of animation.
Prerequisites	1. Basic programming skills of C++ and Graphics Library 2. OpenGL Library 3. 64-bit Open source Linux 4. Open Source C++ Programming tool like G++/GCC, OpenGL
Learning Objectives	The learner will be able to implement: 1. 3D viewing transformation 2. 3D clipping 3. Seed filling algorithms 4. Projection transformations 5. Illumination & colour models 6. Shading algorithms 7. Hidden / Back face detection & removal algorithms 8. Animation and Simulation 9. OpenGL predefined functions 10. Motion control using Keyboard keys in OpenGL

------Theory-----
Basics of OpenGL

- Open Graphics Library (OpenGL) is a cross-language (language independent), cross-platform (platform independent) API for rendering 2D and 3D Vector Graphics (use of polygons to represent image).
- OpenGL is a low-level, widely supported modeling and rendering software package, available across all platforms.
- It can be used in a range of graphics applications, such as games, CAD design, or modeling. OpenGL API is designed mostly in hardware so it is a hardware logic.

Features of OpenGL

- **Design**: This API is defined as a set of functions which may be called by the client program. Although functions are similar to those of C language but it is language independent.

- **Development**: It is an evolving API and Khronos Group regularly releases its new version having some extended feature compare to previous one. GPU vendors may also provide some additional functionality in the form of extension.

- **Associated Libraries:** The earliest version is released with a companion library called OpenGL

utility library. But since OpenGL is quite a complex process. So in order to make it easier other library such as OpenGL Utility Toolkit is added which is later superseded by freeglut. Later included library were GLEE, GLEW and glbinding.

- **Implementation:** Mesa 3D is an open source implementation of OpenGL. It can do pure software rendering and it may also use hardware acceleration on BSD, Linux, and other platforms by taking advantage of Direct Rendering Infrastructure.

OpenGL Installation

We need the following sets of libraries in programming OpenGL:

- **Core OpenGL (GL):** consists of hundreds of functions, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives, such as point, line, and polygon.

- **OpenGL Utility Library (GLU):** built on-top of the core OpenGL to provide important utilities and more building models (such as qradric surfaces). GLU functions start with a prefix "glu" (e.g., gluLookAt, gluPerspective)

OpenGL Utilities

OpenGL Utilities Toolkit (GLUT):

i. It provides support to interact with the Operating System (such as creating a window, handling key and mouse inputs); and more building models (such as sphere and torus).

ii. GLUT functions start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc).

iii. GLUT is designed for constructing small to medium sized OpenGL programs.

iv. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is simple, easy, and small.

v. Alternative of GLUT includes SDL.

Prerequisites for OpenGL

Since OpenGL is a graphics API and not a platform of its own, it requires a language to operate in and the language of choice is C++.

Getting started with an OpenGL program

- Main
- Open window and configure frame buffer (using GLUT for example)
- Initialize GL states and display (Double buffer, color mode, etc.)
- Loop
- Check for events

If window event (resize, unhide, maximize , etc)
modify the viewport and redraw
Else if input event (keyboard and mouse, etc.)
Handle the event (such as move the camera or change the state) and usually draw the scene

- Redraw
- Clear the screen (and buffers e.g., z-buffer)
- Change states (if desired)
- Render
- Swap buffers (if double buffer)

OpenGL order of Operations

- Construct shapes (geometric descriptions of objects – vertices, edges, polygons etc.)
- Use OpenGL to arrange shape in 3D (using transformations)
- Select your vantage point (and perhaps lights)
- Calculate color and texture properties of each object
- Convert shapes into pixels on screen

OpenGL Syntax

- All functions have the form: gl*
- glVertex3f() – 3 means that this function take three arguments, and f means that the type of those arguments is float.
- glVertex2i() – 2 means that this function take two arguments, and i means that the type of those arguments is integer

- All variable types have the form: GL*
- In OpenGL program it is better to use OpenGL variable types (portability)
- Glfloat instead of float
- Glint instead of int

OpenGL primitives

Drawing two lines:

```
glBegin(GL_LINES);
glVertex3f(_,_,_); // start point of line 1 glVertex3f(_,_,_);
// end point of line 1 glVertex3f(_,_,_); // start point of
line 2 glVertex3f(_,_,_);
// end point of line 2 glEnd();
```

We can replace GL_LINES with GL_POINTS, GL_LINELOOP, GL_POLYGON etc.

OpenGL states:

OpenGL is like a state machine which stores the details of the previous image drawn on it in its memory.

- On/off (e.g., depth buffer test)
- glEnable(GLenum)
- glDisable(GLenum)
- Examples:
- glEnable(GL_DEPTH_TEST);
- glDisable(GL_LIGHTING);
- Mode States

- Once the mode is set the effect stays until reset
- Examples:
- glShadeModel(GL_FLAT) or glShadeModel(GL_SMOOTH)
- glLightModel(…) etc.

Drawing in 3D:

Depth buffer (or z-buffer) allows to remove hidden surfaces. We use glEnable(GL_DEPTH_TEST) to enable it.

glPolygonMode(Face, Mode)

- Face: GL_FRONT, GL_BACK,
- GL_FRONT_AND_BACK
- Mode: GL_LINE, GL_POINT, GL_FILL

glCullFace(Mode)

- Mode: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
- glFrontFace(Vertex_Ordering) where Vertex Ordering: GL_CW or GL_CCW

Viewing transformation:

- glMatrixMode (Mode)

Mode: GL_MODELVIEW, GL_PROJECTION, or GL_TEXTURE

- glLoadIdentity()
- glTranslate3f(x,y,z)
- glRotate3f(angle,x,y,z)
- glScale3f(x,y,z)

Spheres & polygons in OpenGL

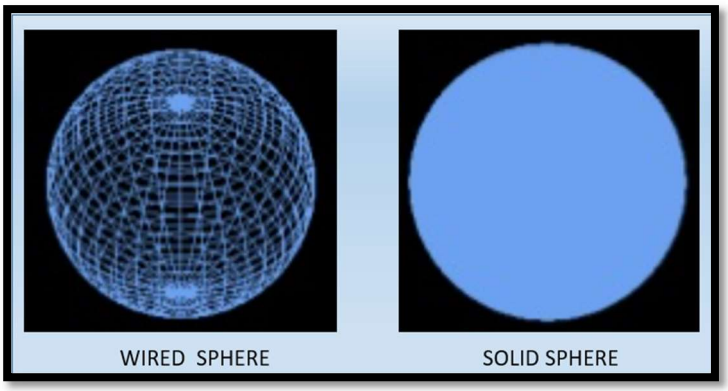
(A) To draw a circle / sphere:

`glutSolidSphere` and `glutWireSphere` render a solid or wireframe sphere respectively.

```
void glutSolidSphere(GLdouble radius,
                     GLint slices, GLint stacks);
void glutWireSphere(GLdouble radius,
                    GLint slices, GLint stacks);
```

`Radius` : Radius of the sphere.
`Slices` : Subdivisions around Z axis (longitudes).
`Stacks` : Subdivisions around Z axis (latitudes).

Renders a sphere centered at modeling coordinates origin of the specified `radius`.



(B) To draw a polygon: //void glBegin(Glenum mode);

Example: 4 sided 3D polygon (Quadrilateral)

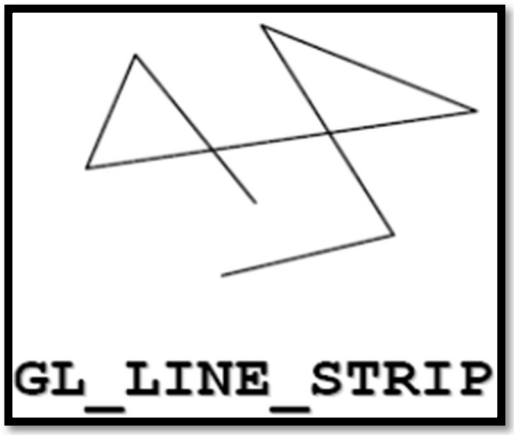
```
glBegin(GL_POLYGON);              //mode= shape to be drawn
      glVertex3f(0.0, 0.0, 0.0);      //glVertex2f(x1,y1,z1)
      glVertex3f(0.0, 3.0, 1.4);      //glVertex2f(x2,y2,z2)
      glVertex3f(4.0, 3.0, 5.3);      //glVertex2f(x3,y3,z3)
      glVertex3f(6.0, 1.5, 2.4);      //glVertex2f(x4,y4,z4)
 glEnd();
```

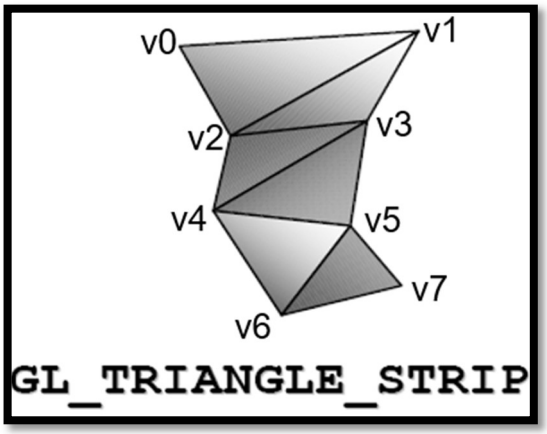
//enclose the shape to be drawn within glbegin() to start drawing & glend() to end drawing on the OpenGL console

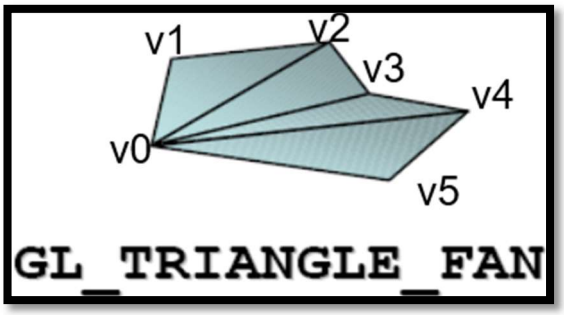
To draw different shapes

Value	Meaning
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_QUAD_STRIP	linked strip of quadrilaterals
GL_POLYGON	boundary of a simple, convex polygon

Some Examples







Writing an OpenGL Program with GLUT

An OpenGL program using the three libraries listed above must include the appropriate headers.

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

Before OpenGL rendering calls can be made, some initialization has to be done. With GLUT, this consists of initializing the GLUT library, initializing the display mode, creating the window, and setting up callback functions. The following lines initialize a full color, double buffered display: *glutInit(&argc, argv);*
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

Double buffering means that there are two buffers, a front buffer and a back buffer. The front buffer is displayed to the user, while the back buffer is used for rendering operations. This prevents flickering that would occur if we rendered directly to the front buffer.

Next, a window is created with GLUT that will contain the viewport which displays the OpenGL front buffer with the following three lines:

glutInitWindowPosition(px, py);
glutInitWindowSize(sx, sy);
glutCreateWindow(name);

To register callback functions, we simply pass the name of the function that handles the event to the appropriate GLUT function.

glutReshapeFunc(reshape);
glutDisplayFunc(display);

Here, the functions should have the following prototypes:
void reshape(int width,
int height); void
display();

In this example, when the user resizes the window, reshape is called by GLUT, and when the display needs to be refreshed, the display function is called. For animation, an idle event handler that takes no arguments can be created to call the display function to constantly redraw the scene with ***glutIdleFunc***. Once all the callbacks have been set up, a call to ***glutMainLoop*** allows the program to run.

In the display function, typically the image buffer is cleared, primitives are rendered to it, and the results are presented to the user. The following line clears the image buffer, setting each pixel color to the clear color, which can be configured to be any color:

glClear(GL_COLOR_BUFFER_BIT);

The next line sets the current rendering color to blue. OpenGL behaves like a state machine, so certain state such as the rendering color is saved by OpenGL and used automatically later as it is needed.

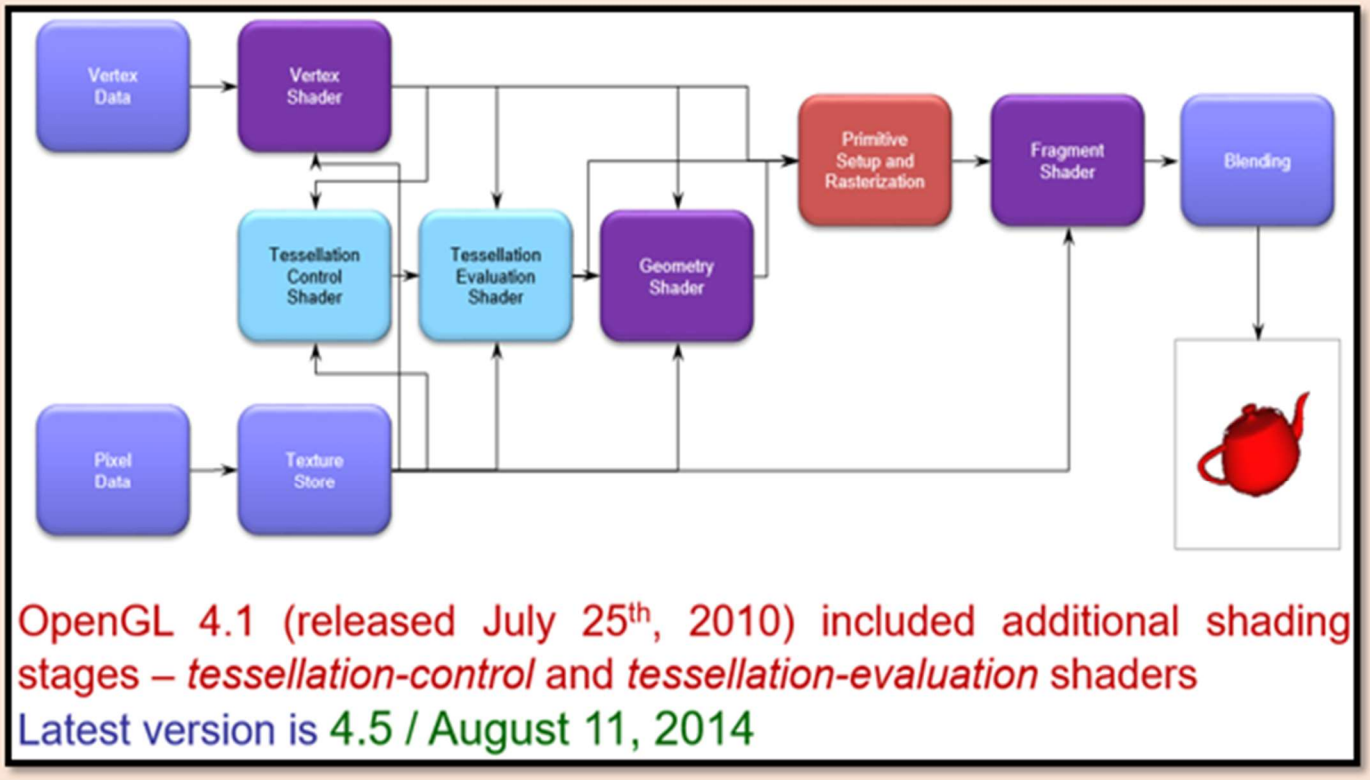
glColor3f(0.0f, 0.0f, 1.0f);

To render a primitive, such as a point, line, or polygon, OpenGL requires that a call to *glBegin* is made to specify the type of primitive being rendered.
glBegin(GL_LINES);

Only a subset of OpenGL commands is available after a call to ***glBegin***. The main command that is used is ***glVertex***, which specifies a vertex position. In GL LINES mode, each pair of vertices define endpoints of a line segment. In this case, a line would be drawn from the point at (x0, y0) to (x1, y1).
glVertex2f(x0, y0); glVertex2f(x1, y1);

A call to glEnd completes rendering of the current primitive. ***glEnd()***; Finally, the back buffer needs to be swapped to the front buffer that the user will see, which GLUT can handle for us: ***glutSwapBuffers();***

OpenGL Pipeline



Developer-Driven Advantages

- **Industry standard**

An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

- **Stable**

OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

- **Reliable and portable**

All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

- **Evolving**

Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in

a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

- **Scalable**

OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

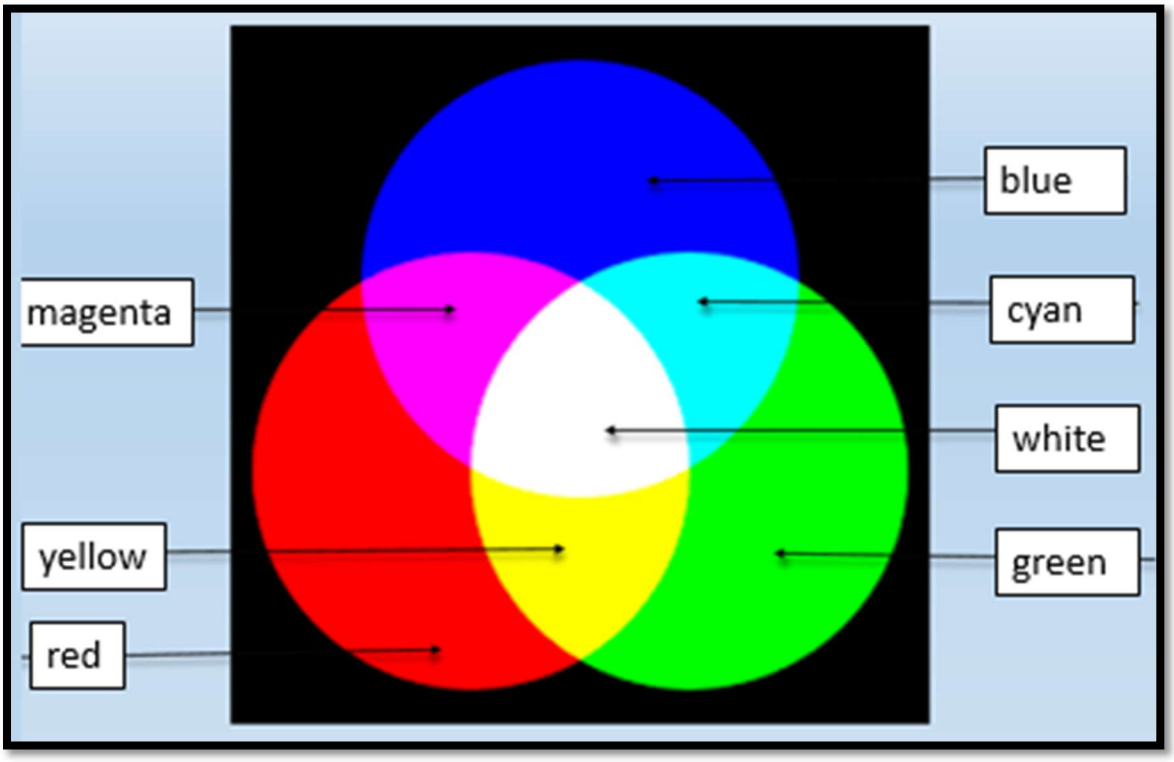
- **Easy to use**

OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

- **Well-documented**

A great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

RGB Colour Model



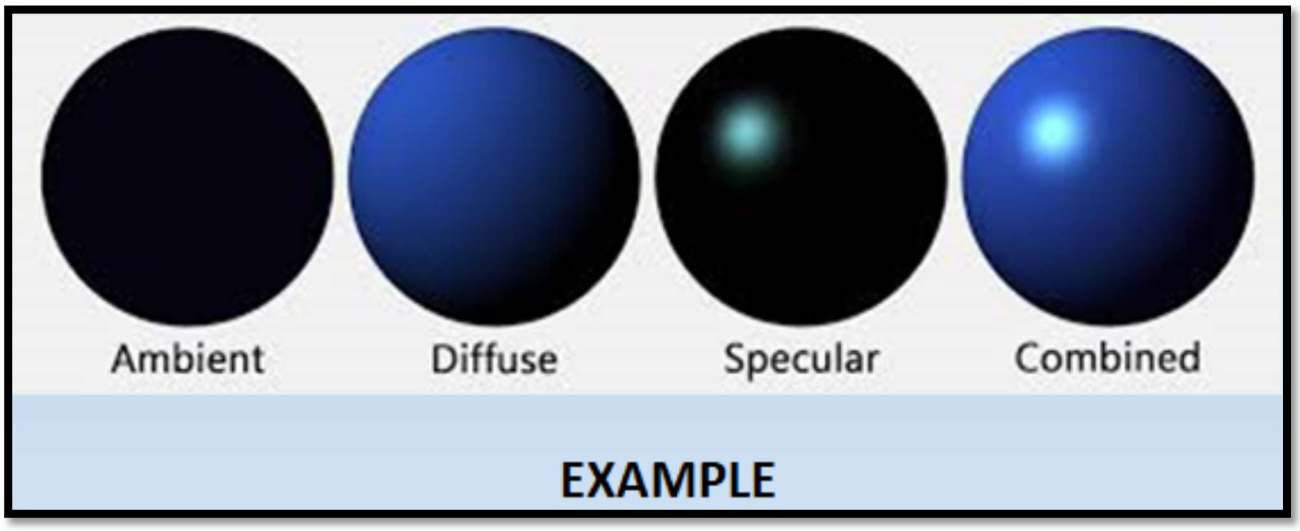
The three primaries in RGB colour model are red, green, blue. It is an additive colour model since colour gamuts are obtained by adding appropriate proportions of these three colours.

Light Models

The OpenGL light model presumes that light reaching our eyes from the polygon surface on the 2D screen arrives by four different mechanisms:

- ***AMBIENT*** - light is reflected from surrounding objects & scattered randomly in all directions equally & so it propagates in all directions to reach other objects in space.
- ***DIFFUSE*** - light that comes from a particular point source (like the Sun) and is incident on surfaces. It best defines the shape of 3D objects.

- ***SPECULAR*** - light comes from a point source, but is reflected more as if from a mirror & produces shiny highlights. It helps to distinguish between flat, dull surfaces like plastered walls and shiny surfaces like polished plastics and metals.
- ***EMISSION*** - light is actually emitted by the polygon, equally in all directions.



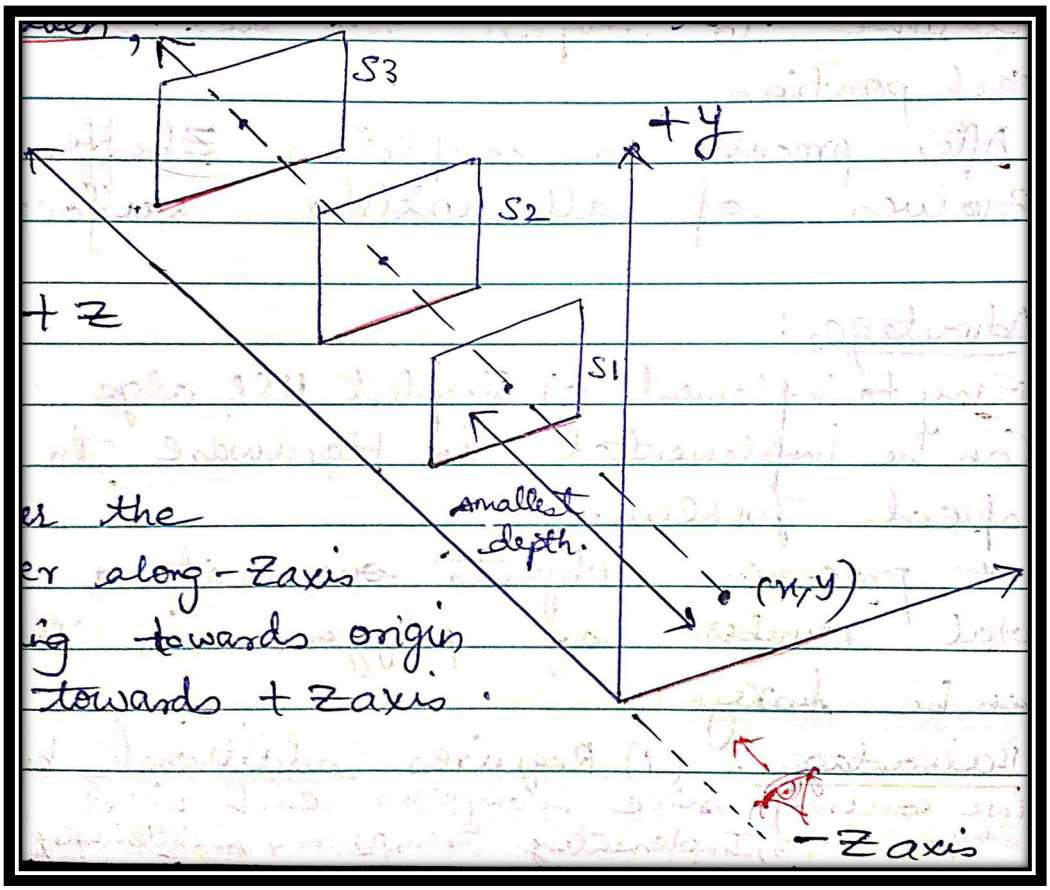
To enable directed light incident on objects in OpenGL:

// **void glEnable(GL_LIGHTING);**

following types of light will give a combination of white and black i.e. a gray shade to the objects visible in front view of user

- **glEnable(GL_LIGHT0);** //Enable white light #0
- **glEnable(GL_LIGHT1);** //Enable black light #1
- This command turns on a directional light that shines from the direction of the viewer into the scene.
- So it illuminates everything, without any specular highlights, that the user can see.

Z / Depth Buffer Algorithm



1) Frame buffer and depth(z) buffer are used.

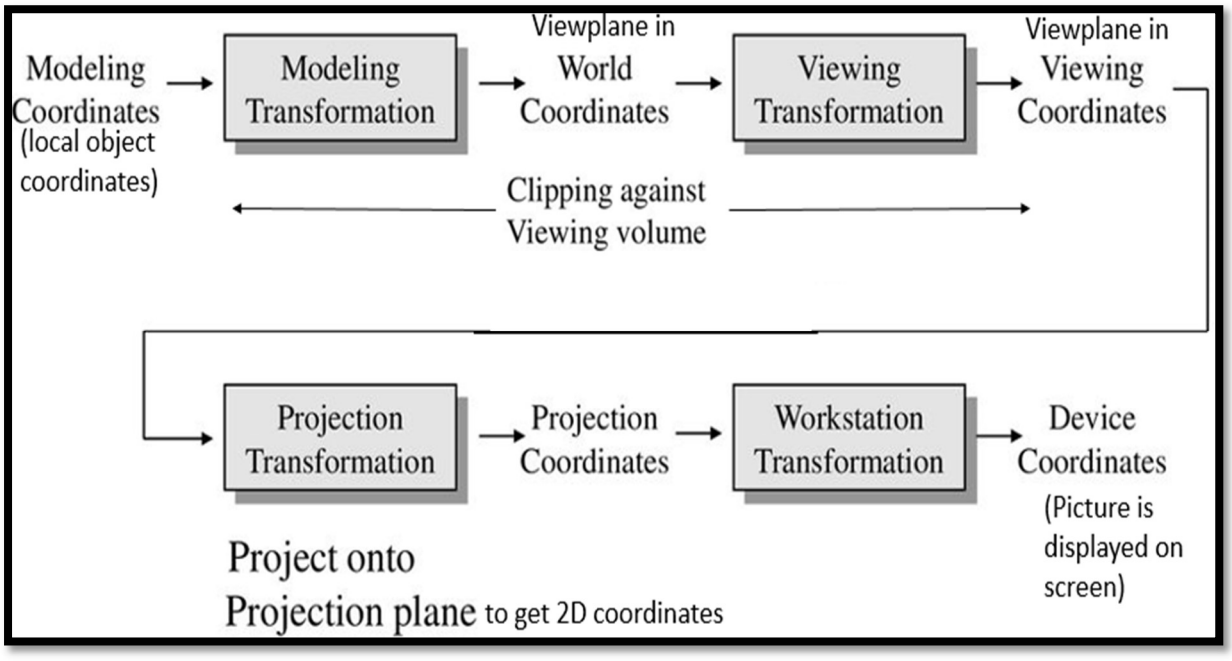
2) In z buffer the depth(z coordinate) of an object in a scene can be stored.

3) If a pixel whose position is stored in frame buffer is in front of the pixel previously stored in z buffer then z coordinate(depth) of this new pixel is stored in z buffer, else no action is taken and repeat the process for the next pixel.

4) After all the pixels have been considered once, z buffer contains z values only of the visible pixels i.e. only visible surfaces will be drawn as per the viewing direction of user and the distance of the object in the scene, from the user.

```
Iinitialize the depth of each pixel.
    d(i, j) = infinite (max length)
Initialize the color value for each pixel
    c(i, j) = background color
for each polygon:
{   for (each pixel in polygon's projection)
        {     find depth i.e, z of polygon
              at (x, y) corresponding to pixel (i, j)
              if (z < d(i, j))
              {         d(i, j) = z;
                        c(i, j) = color;
              }
        }
}
```

3D Viewing Transformation



Window to viewport mapping

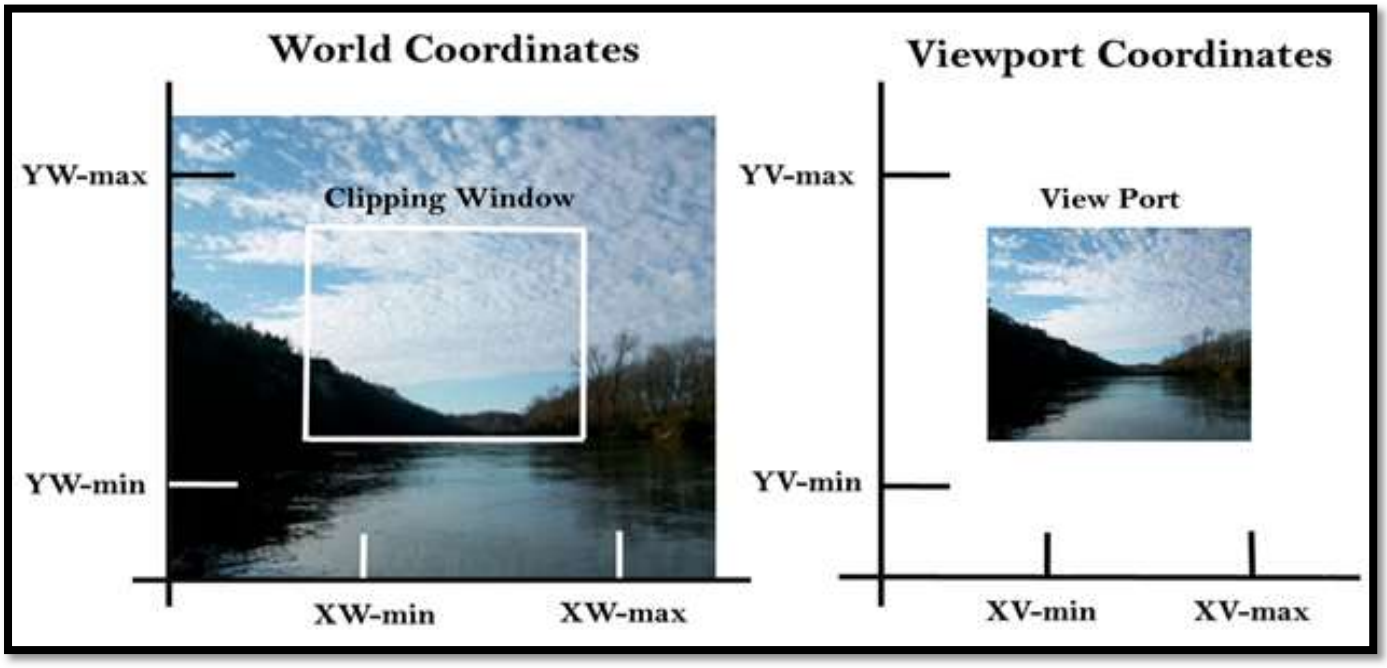
Modelling coordinates: These are the 3D local object coordinates in the real world.

World Coordinates: These are the 3D object coordinates relative to other objects in the real world.

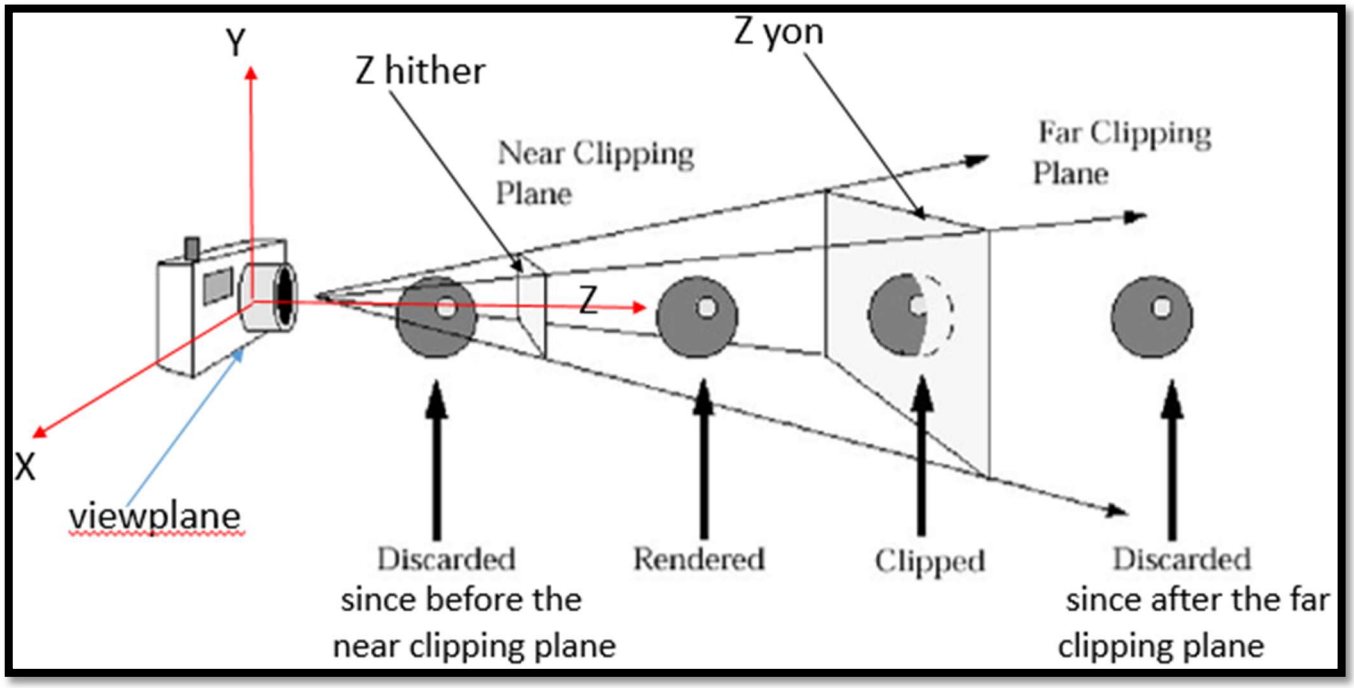
Window: The method of selecting and enlarging a portion of a drawing is called windowing. The area chosen for this display is called a window.

Viewport: It is the display area on the window (screen), which is measured in pixels in screen coordinates.

Projections: These are vectors extended from corners of the 3D objects to intersect a 2D projection plane to convert a 3D object to its 2D image on projection plane.



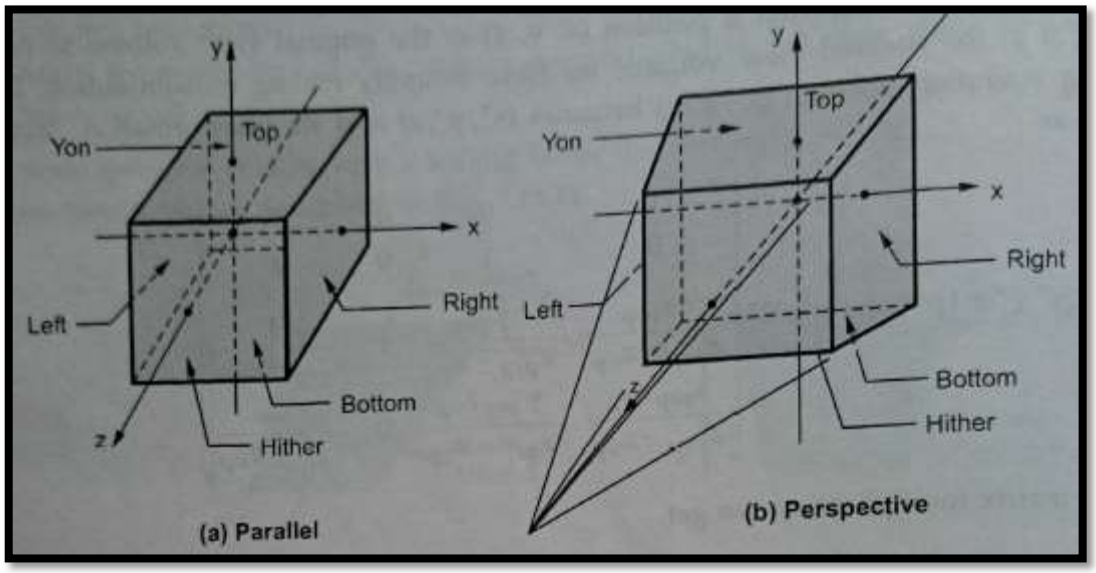
3D Clipping



Clipping:

- **It is a method to selectively enable or disable rendering operations within a defined region of interest.**
- **In 2D, a clip region may be defined so that pixels are only drawn within the boundaries of a window or frame.**

In 3D space we clip view volume. Common 3D clipping volumes are: rectangular parallelepiped and truncated pyramidal volume.



These volumes are six sided with sides: left, right, top, bottom, hither (near), and yon (far).

The 2D region codes can be extended to 3D by considering six sides and 6-bit code instead of four sides and 4-bit code

- A line segment is completely within the view volume if both endpoints have code 000000.
- If either endpoint has non-zero code perform the logical AND operation on both endpoint codes. If the result is nonzero then both endpoints are outside the view volume and line segment is completely invisible.
- If result is zero then line segment may be partially visible. so determine this intersection of the line with the clipping volume.

Motion Control using Keyboard Keys

```
void keyPress(int key, int x, int y)  // to control horizontal and vertical motion of objects
{
    if(key==GLUT_KEY_RIGHT)
        X += 0.05f;       //increment x to move towards the right
    if(key==GLUT_KEY_LEFT)
        X  -= 0.05f;      //decrement x to move towards the left
    if(key==GLUT_KEY_UP)
        Y += 0.05f;       //increment y to move upwards
    if(key==GLUT_KEY_DOWN)
        Y -= 0.05f;       //decrement y to move downwards

   glutPostRedisplay();
}
//Function call using OpenGL function is:
glutSpecialFunc(keyPress);
//Here, the user defined function name is passed as a parameter to the OpenGL function
```

---Executable OpenGL Program Code---

```
// header files or C++ standard libraries ie. preprocessor derectives

#include<iostream>        //for standard input/output in C++
#include<stdlib.h>        /*for exit(0) function, to stop program execution and exit
from
                           output console window*/

//if Mac is used
#ifdef __APPLE__
#include<openGL/openGL.h>
#include<GLUT/glut.h>

//if Windows is used          //Similarly, different for linux,etc...
#else
#include<GL/glut.h>
#endif

using namespace std;         // to use the namespace defined in C++ libraries
                             //and not any other or userdefined namespace, to avoid any
ambiguity

// Initial position or coordinates of the SUN when it is at bottom left side of
display screen
float ballX = -0.8f;     //x coordinate of sun
float ballY = -0.3f;     //y coordinate of sun
float ballZ = -1.2f;     //z coordinate of sun

// SUN colour in RGB  colour model
float colR=3.0;         // use appropriate quantities of RGB for the initial sun colour
float colG=1.5;
float colB=1.0;

// Initial background colour in RGB is black colour when SUN is at bottom left side of
display screen
float bgColR=0.0;      //0.0 indicates black colour  &&   1.0 indicates white colour
float bgColG=0.0;
float bgColB=0.0;

static int flag=1;  /* only one variable of the name flag is available for the entire
scope
                      of this program which is explicitly (i.e. by user) initialised to
1
                      else garbage value or 0 is assigned to flag
                      by compiler implicitly...depends on the compiler used
                     */

//TO DRAW SUN / BALL
void drawBall(void)   // user defined function
{
        glColor3f(colR,colG,colB); //set ball colour
        glTranslatef(ballX,ballY,ballZ); /*motion or translation ie. change in
position
                                          of the ball along the screen
                                         from L.H.S to R.H.S of screen
                                         */
        glutSolidSphere(0.05, 30, 30);  //create ball or sun    //explained by ppt
}

// TO DRAW MOUNTAINS
void drawAv(void)
```

```
{
    glBegin(GL_POLYGON);

        glColor3f(1.0,1.0,1.0);

        glVertex3f(-0.9,-0.7,-1.0);       //explained by ppt

        glVertex3f(-0.5,-0.1,-1.0);

        glVertex3f(-0.2,-1.0,-1.0);

        glVertex3f(0.5,0.0,-1.0);

        glVertex3f(0.6,-0.2,-1.0);

        glVertex3f(0.9,-0.7,-1.0);

    glEnd();
}


void keyPress(int key, int x, int y)  // to control horizontal and vertical motion of
the sun
                                      // (x,y) is current position of mouse pointer
{
      if(key==GLUT_KEY_RIGHT)
        ballX += 0.05f;         //increment x to move towards the right
      if(key==GLUT_KEY_LEFT)
        ballX  -= 0.05f;        //decrement x to move towards the left
      if(key==GLUT_KEY_UP)
        ballY += 0.05f;         //increment y to move upwards
      if(key==GLUT_KEY_DOWN)
        ballY -= 0.05f;         //decrement y to move downwards

    glutPostRedisplay();       /* Tells GLUT that the display has changed ie. state has
been updated
                              (after each iteration ,due to translation of sun)
                             so the scene needs to be redrawn and redisplayed to
reflect the new state.
                              */
}

void handleKeypress(unsigned char k, int x, int y)      // for exit(0) from program
                                                        // k is a non-
negative(unsigned) character
{
if(k==27)             //ASCII value of escape key =27
    exit(0);
}

void initRendering()   // Removal of hidden surfaces and shading algorithms
{
    glEnable(GL_DEPTH_TEST);      //allows SUN to move behind the mountains and not in
the front of them,
                                  // can use depth / Z Buffer algo
    glEnable(GL_COLOR_MATERIAL);  //to shade the sun as per its position on screen

//Enable lighting
glEnable(GL_LIGHTING);
```

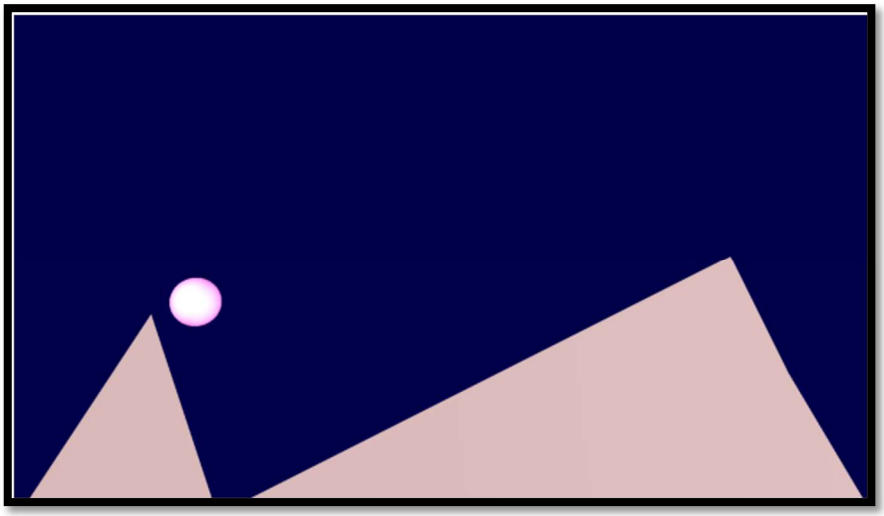
```
//following types of light will give a combination of white and black
//i.e. a gray shade to the objects visible in front view
    glEnable(GL_LIGHT0); //Enable white light #0
    glEnable(GL_LIGHT1); //Enable black light #1

//Optional shading effect
    glEnable(GL_NORMALIZE);  //Automatically normalize normals... for phong shading
    glShadeModel(GL_SMOOTH); //Enable smooth shading (like analog signal)
}


/* Instructs OpenGL to convert from x,y,z real world window coordinates to pixel
values of viewport
 since pixels and coordinates are different phenomena
 to adjust the scene/image position as per size of window
*/
void handleResize(int w, int h)   /* w=current width, h=current height i.e. current
size of window
                                     when the output console opens*/
{
    glViewport(0,0,w,h);    //viewport is a quadrilateral (polygon having 4 vertices)
                            //bottom left vertex is (0,0) and upper right vertex is (w,
h)

    glMatrixMode(GL_PROJECTION);     // for setting the camera perspective i.e front
view
                                     // helps resolve the overlapping images of objects
                                     //sets a matrix of polygon vertices
    //Set the camera perspective
    glLoadIdentity();       /*Reset the camera ie. reset the matrix of polygon
vertices
                             to identity matrix
                             Resetting the matrix generated above
                             since x,y,z coordinates change with the motion of sun
                             This function takes no arguments, returns no value*/

    gluPerspective(45.0,                    /*The camera angle, it decides the viewing
direction
                                              for user/viewer */

                   (double)w / (double)h, /*The width-to-height ratio (aspect ratio)
of window
                                           type casting done ,also increases precision
                                           from float to double */

                   1.0,       //The near z (hither) clipping coordinate i.e. starting
point for starting clipping
                   200.0);    //The far z (yon) clipping coordinate i.e. end point to
stop 3D clipping
                              //clipping the depths(z values) of objects outside the
window

}

// TO ADD COLOUR / SHADING TO THE OBJECTS OF THE SCENE
void drawScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);   //for clear screen, similar
to cleardevice();
    glClearColor(bgColR,bgColG,bgColB,0.0);              // clears the colour buffers
as per 4th parameter= alpha
```

```
                                                        //alpha=0.0 colours become
invisible (completely opaque),
                                                         //alpha=1.0 then colours are
visible (transparent)
    glMatrixMode(GL_MODELVIEW);                 // future matrix manipulations should
affect the modelview matrix

    glLoadIdentity();       //reset matrix to identity due to redisplaying the scene
repeatedly


   //1.0 purest maximum form of the colour, 0.0 is absence of colour

    //Add ambient / scattered light
    GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f}; //{ red , green , blue ,
transperancy/thickness of colour }
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);    // selects ambient light
for the object

// directed light ie. as per direction of light, shadow of the oject will be formed
    //Add directed light 1 ie. light source 1 is at a particular position
    GLfloat lightColor0[] = {0.5f, 0.5f, 0.5f, 1.0f};
    GLfloat lightPos0[] = {4.0f, 0.0f, 8.0f, 1.0f};
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightColor0);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);

    //Add directed light 2 ie.light source 2 is at a particular position
    GLfloat lightColor1[] = {0.5f, 0.2f, 0.2f, 1.0f}; //Color (0.5, 0.2, 0.2)
    //Coming from the direction (-1, 0.5, 0.5)
    GLfloat lightPos1[] = {-1.0f, 0.5f, 0.5f, 0.0f};
    glLightfv(GL_LIGHT1, GL_DIFFUSE, lightColor1);
    glLightfv(GL_LIGHT1, GL_POSITION, lightPos1);

    //drawing the SUN
    glPushMatrix();     // push matrix of coordinates of circle on the stack
        drawBall();     // circle is drawn as per matrix entries
    glPopMatrix();      // pop this matrix

    //drawing the Mountain
    glPushMatrix();      // similarly
        drawAv();
    glPopMatrix();

    glutSwapBuffers();      // implement double buffering
}

//Sun colour manipulation
void update(int value)
{
    if(ballX>0.9f)  // x coordinate of the sun becomes positive when sun crosses more
than half
                    //of the horizontal distance
    {
        ballX = -0.8f;        // for proper sunset
        ballY = -0.3f;
        flag=1;
        colR=2.0;       //When sun crosses more than half of the horizontal distance
increase red light for sun set time
        colG=1.50;      // accordingly use appropriate quantity of green and blue to
get orange coloured sun
```

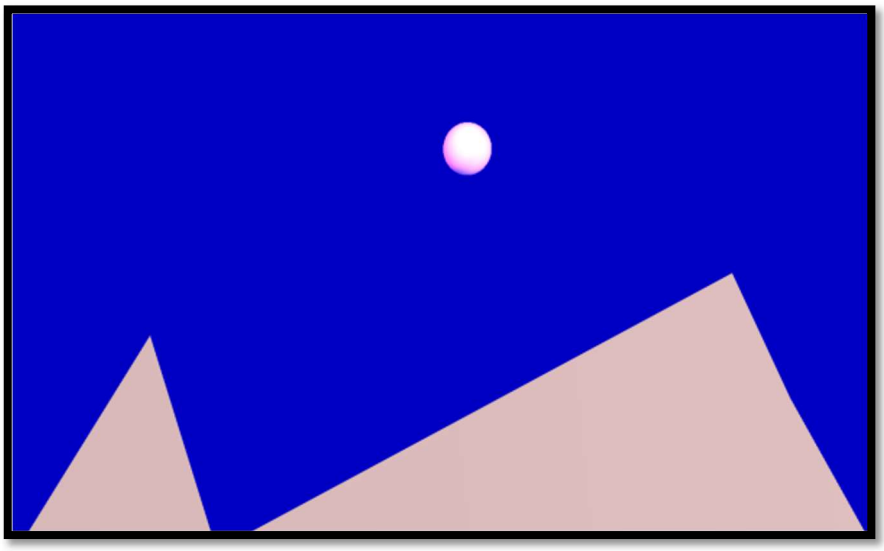
```
        colB=1.0;

        bgColB=0.0;      //initially, sky is blue so start making background (sky)
dark for sun set time
                //since 0.0 is black , 1.0 is white
    }

    if(flag)     // initially flag=1 && ballx < 0.9f
    {
    ballX += 0.001f;    //so move the ball to R.H.S
    ballY +=0.0007f;    //also move the ball upwards        .....for sunrise

    colR-=0.001;      // accordingly adjust sun colour       ....this is sunrise
    colB+=0.005;

    bgColB+=0.001;    //adjust background by making sky more blue      .....since
sunrise

       if(ballX>0.01)        //if ball crosses half of the screen
           flag=0;
    }
    if (!flag)              // initially !1 == 0 == false i.e. if statement is not
executed
               // after ball crosses half of the screen flag=0,  !0 == 1 ==true i.e.
if statement gets executed
    {
        ballX += 0.001f;    //to the R.H.S. sun sets ,increase x, decrese y ,sun is
moving downward and right side
        ballY -=0.0007f;
        colR+=0.001;        //to get orange coloured sun, adjust colours
        colB-=0.01;

        bgColB-=0.001;      //make sky darker by decreasing blue colour

        if(ballX<-0.3)     // after one complete cycle of sunrise & sunset
                  //to make flag=1 again and start sunrise again from L.H.S to set
proper colour for rising sun
           flag=1;
    }

    glutPostRedisplay(); /* Tells GLUT that the display has changed ie. state has been
updated
                        (after each iteration ,due to translation of sun)
                       so the scene needs to be redrawn and redisplayed to reflect the
new state.
                       */

    // Tell GLUT to call update again after 25 milliseconds, this acts as a loop
    glutTimerFunc(25, update, 0);   // glutTimerFunc(time in ms, function_name, number
of times this function will be called);

}

// Program Execution starts from main() function
 int main(int argc,char** argv)  //command line arguments necessary to initialise GLUT
{
    glutInit(&argc,argv);    //initialize GLUT needed for OpenGL programs

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);        //indicate use of
double buffering ,
```

```
                                                                    //RGB colour
model, depth checking
                                                                    // bitwise OR |
 ,  logical OR ||
   // To set/initialize the size of output console window
    glutInitWindowSize(1300,650);    // glutInitWindowSize(int width,int height);

   /* OpenGL output console window is created
   the name of that window is passed as a parameter to this built-in or predefined or
system
   defined or ready made function
   user only needs to be call/invoke this fnction and user does not need to define it
   */
    glutCreateWindow("sun rise and sun set");

    initRendering();     //function call to remove hidden surfaces and for shading

    glutDisplayFunc(drawScene);   //displays the scene drawn at the output window

    //glutFullScreen();   // To make OpenGL output console window fullscreen,
                          //but then exit from the program is inconvenient

    glutSpecialFunc(keyPress);         //Sun motion contol
    glutReshapeFunc(handleResize);    //real world window / display screen viewport
and objects within
                                      //them are spaced out
                                      //as per current size of output window

    glutKeyboardFunc(handleKeypress); // for exit(0) function

    // Tell GLUT to call update again after 25 milliseconds, this acts as a loop
    glutTimerFunc(25, update, 0);   // glutTimerFunc(time in ms, function_name, number
of times
                                    //this function will be called);

    glutMainLoop();    // acts as an infinte loop ie. sun will keep rising and setting
continuously
                       // enables us to use keyboard keys to control motion of objects
                       // helps in resizing the window
    return(0);
}
```

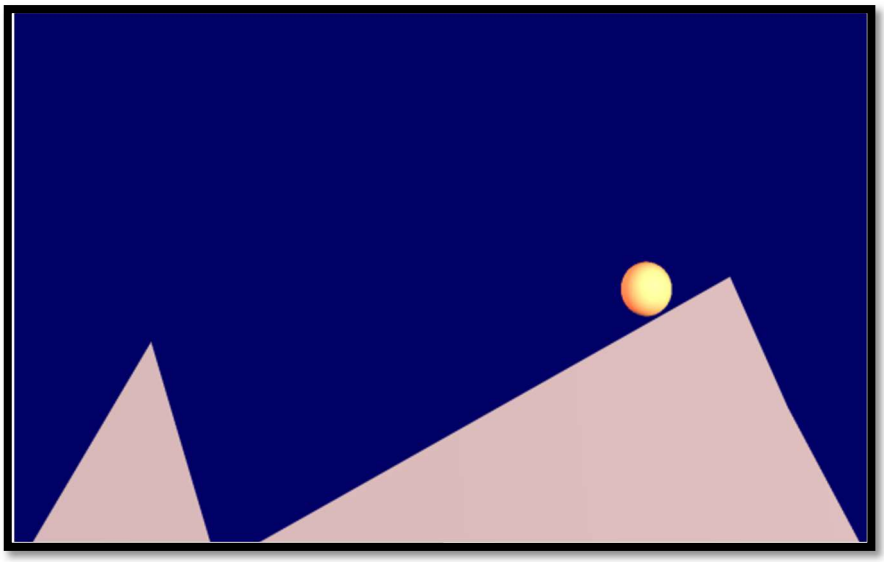
-----Sample Output Screenshots-----



Phase 1: The sun rises from lower left (coordinate axes origin) of the screen i.e. East, behind the mountains- possible by depth calculation using Z buffer algorithm. The sky is dark since sunrise has just begun.

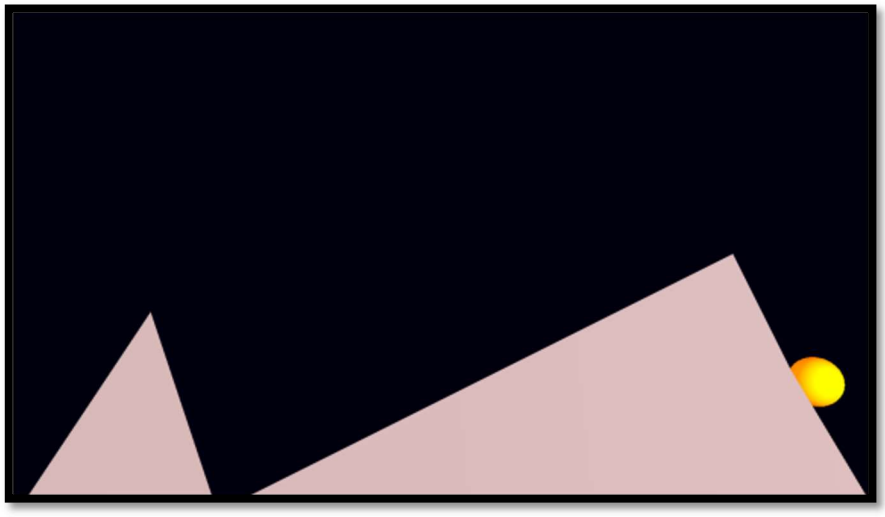


Phase 2: On further sunrise, x & y coordinates of the sun increase for it's horizontal & vertical advancement along the trajectory. The sky gets brighter eventually.



Phase 3: As sun crosses half of the horizontal distance, it starts setting and it's color becomes orangish. Color shading is possible due to RGB and light models. It's x coordinate increases but y coordinate decreases.

The sky once again starts getting darkergets brighter eventually.



Phase 4: Finally, the sun sets at the lower right of the screen i.e. West and it now has a darker orange shade. The sky is completely dark due to complete sunset.

:CONCLUSION:

1. Students are able to understand windowing and clipping and apply various algorithms to fill, clip or shade polygons in 2D & 3D.

2. The student can explain the core concepts of computer graphics using OpenGL in 2D & 3D as in viewing and projections.

3. Illustrating the concepts of colour models, lighting, shading models and hidden surface elimination is now possible.

4. Animation is described along with keyboard or mouse motion control at program run time.

References and useful links

1) **Dev C++ installation:**

https://sourceforge.net/projects/orwelldevcpp/

2) **freeglut installation for OpenGL:**

http://freeglut.sourceforge.net/index.php#download

3) **Examples of OpenGL 3D Graphics:**

https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_Examples.html

4) **OpenGL Functions:**

https://csciwww.etsu.edu/barrettm/4157/OpenGL%20Functions.htm

5) **RGB Colour Model:**

https://www.geeksforgeeks.org/computer-graphics-the-rgb-color-model/

6) **3D viewing transformations:**

https://www.gatevidyalay.com/tag/3d-viewing-transformation-in-computer-graphics/

7) **Depth Buffer Algorithm:**

https://www.javatpoint.com/computer-graphics-z-buffer-algorithm