CHAPTER 1
INTRODUCTION TO COMPUTER GRAPHICS

1.1 HISTORY

Computer Graphics (CG) was first created as a visualization tool for scientists and engineers in government and corporate research centers such as Bell Labs and Boeing in the 1950s.Later the tools would be developed at Universities in the 60s and 70s at places such as Ohio State University, MIT, University of Utah, Cornell, North Carolina and the New York Institute of Technology. The early breakthroughs that took place in academic centers continued at research centers such as the famous Xerox PARC in the 1970¹s. These efforts broke first into broadcast video graphics and then major motion pictures in the late 70¹s and early 1980¹s. Computer graphic research continues today around the world, now joined by the research and development departments of entertainment and production companies. Companies such as George Lucas¹s Industrial Light and Magic are constantly redefining the cutting edge of computer graphic technology in order to present the world with a new synthetic digital reality.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as computer-generated imagery (CGI).

The very first computer assisted graphics began in many different unrelated fields around the world. There is a very blurred line that is crossed somewhere between mechanical and analog computer assisted graphics, and the first directly digital computer-generated graphics that would associate with today as being true CG.

The very first radiosity image. While at MIT in the 1940s, Professors Parry Moon and Domina Eberle Spencer were using their field of applied mathematics to calculate highly accurate global lighting models which they called ³interflection reflection. The illumination algorithms were based on those by H. H. Higbie, published in his 1934 book, Lighting Calculations. The paper was cut out and ironed together by hand to create the image shown here in print for the first time in over 50 years. The images were first presented at the 1946 National Technical Conference of the Illuminating Engineering Society of North America and published two years later (in color) in the book: Lighting Design by Moon, P., and D. E. Spencer. 1948. (Addison Wesley. Cambridge, MA) The book was used for many years to teach lighting theory at MIT in the architecture curriculum there. Dr. Spencer went on to teach at Tufts, Brown, Rhode Island School of Design, and the University of Connecticut where she remains active today.

1.2 COMPUTER GRAPHICS

Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real-world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

1.3 PIONEERS IN GRAPHICS DESIGN

Charles Csuri

Charles Csuri is a pioneer in computer animation and digital fine art and created the first computer art in 1964. Csuri was recognized by Smithsonian as the father of digital art and computer animation, and as a pioneer of computer animation by the Museum of Modern Art (MoMA) and Association for Computing Machinery-SIGGRAPH.

Donald P. Greenberg

Donald P. Greenberg is a leading innovator in computer graphics. Greenberg has authored hundreds of articles and served as a teacher and mentor to many prominent computer graphic artists, animators, and researchers such as Robert L. Cook, Marc Levoy, Brian A. Barsky, and Wayne Lytle. Many of his former students have won Academy Awards for technical achievements and several have won the SIGGRAPH Achievement Award. Greenberg was the founding director of the NSF Center for Computer Graphics and Scientific Visualization.

A. Michael Noll

Noll was one of the first researchers to use a digital computer to create artistic patterns and to formalize the use of random processes in the creation of visual arts. He began creating digital art in 1962, making him one of the earliest digital artists. In 1965, Noll along with Frieder Nake and Georg Nees were the first to publicly exhibit their computer art. During April 1965, the Howard Wise Gallery exhibited Noll's computer art along with random-dot patterns by Bela Julesz.

1.4 VISUALIZATION

Visualization is any technique for creating images, diagrams, or animations to communicate a message. Visualization through visual imagery have been an effective way to communicate both abstract and concrete ideas since the dawn of man. Examples from history include Egyptian hieroglyphs, Greek geometry for engineering and scientific purposes. Visualization today has ever expanding applications in science, education, engineering (e.g. product visualization), interactive multimedia, medicine etc. Typical of a visualization application is the field of computer graphics. The invention of computer graphics may be the most important development in visualization since the invention of central perspective in the Renaissance period. The development of animation also helped advance visualization. The use of visualization to data plots for over a thousand years. Computer graphics has from its beginning been used to study scientific problems. The recent emphasis on visualization started in 1987 with the special issue of Computer Graphics on Visualization in Scientific Computing. Since then there have been several conferences and workspaces, co-sponsored by the IEEE Computer Society and ACCM SIGGRAPH, developed to the general topic, and special areas in the field, for example volume visualization. Scientific visualizations are computer-generated images that show real spacecraft in action, out in the void far beyond Earth, or on other planets.

CHAPTER 2

OVERVIEW OF OPENGL

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Silicon Graphics Inc., (SGI) started developing OpenGL in 1991 and released it in January 1992. Applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games. Since 2006 OpenGL has been managed by the non-profit technology consortium Khronos Group.

2.1 SPECIFICATION

One of the major accomplishment in the specification of OpenGL was the isolation of windows system dependencies from OpenGL's rendering model. The result is that OpenGL is windows system independent. At its most basic level OpenGL is specification, meaning it is simply a document that describes a set of functions and the precise behaviors that they must perform, from this specification, hardware vendors create implementations-libraries of functions created to match the functions stated in the OpenGL specification, making use of hardware acceleration where possible.

OpenGL (Open Graphics Library) is the interface between a graphic program and graphics hardware. It is streamlined. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.

- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation
- It is a rendering pipeline. The rendering pipeline consists of the following steps:
    - Defines objects mathematically.
    - Arranges objects in space relative to a viewpoint.
    - Calculates the color of the objects.
    - Rasterizes the objects.

2.2 OpenGL Related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. Several libraries enable you to simplify your programming tasks, including the following:

- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. Portions of the GLU are described in the OpenGL Reference Manual.
- For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix glX. For Microsoft Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix wgl. For IBM OS/2, the PGL is the Presentation Manager to OpenGL interface, and its routines use the prefix pgl. For Apple, the AGL is the interface for systems that support OpenGL, and AGL routines use the prefix agl.
- The OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs.GLUT routines use the prefix glut

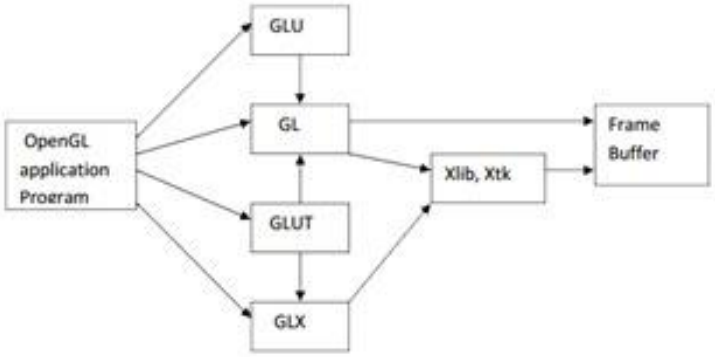


Fig.2.2.1 Library organization

#include <GLUT/glut.h> is sufficient to read in glu.h and gl.h. Although OpenGL is not object oriented, it supports a variety of data types through multiple forms for many functions. For example, we will use various forms of the function glUniform to transfer data to shaders. If we transfer a floating-point number such as a time value, we would use glUniform1f. We could use glUniform3iv to transfer an integer position in three dimensions through a pointer to a three-dimensional array of ints. Later, we will use the form glUniformMatrix4fv to transfer a 4 × 4 matrix of floats. We will refer to such functions using the notation glSomeFunction*(); where the * can be interpreted as either two or three characters of the formnt or ntv, where

n signifies the number of dimensions (2, 3, 4, or matrix); t denotes the data type, such as integer (i), float (f), or double (d); and v, if present, indicates that the variables are specified through a pointer to an array, rather than through an argument list. We will use whatever form is best suited for our discussion, leaving the details of the various other forms to the OpenGL Programming Guide. Regardless of which form an application programmer chooses, the underlying representation is the same, just as the plane on which we are constructing the gasket can be looked at as either a two-dimensional space or the subspace of a three-dimensional space corresponding to the plane z = 0. In Chapter 3, we will see that the underlying representation is four-dimensional; however, we do not need to worry about that fact yet. In general, the application programmer chooses the form to use that is best suited for her application.

2.3 Window Management of OpenGL

void glutInit(int *argc, char **argv):

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.glutInit also processes command line options, but the specific options parse are window system dependent.

void glutInitDisplayMode(unsigned int mode):

The initial display mode is used when creating top-level windows, subwindows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.Note that GLUT_RGBA selects the RGBA color model, but it does not request any bits of alpha (sometimes called an alpha buffer *or* destination alpha) be allocated. To request alpha, specify GLUT_ALPHA. The same applies to GLUT_LUMINANCE

.

void glutInitWindowSize(int width, int height):

This function allows you to request initial dimensions for future windows.There is a callback function to inform you of the new window shape (whether initially opened, changed by your glutReshapeWindow() request, or changed directly by the user).

int glutCreateWindow(char *name):

glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name .

2.4 CALLBACKS AND RUNNING THE PROGRAM

void glutDisplayFunc(void (*func)(void)):

glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters

void glutPostRedisplay(void):

Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

void glutMainLoop(void):

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

2.5 FEATURES OF OPENGL

The main features of OpenGL are

- It provides 2D and 3D geometric objects such lines, polygon, triangle, meshes, spheres, cubes, quadric surface and curve surface.
- It provides 3D modeling transformation and viewing functions to create views of 3d scenes using the idea of virtual camera.
- It supports high quality rendering of scenes including hidden surface removal, multiple light sources, material types, transparency, textures, blending, fog.
- It provides display list for creating graphics caches and hierarchal models. It also supports the interactive “picking” of object.
- It supports the manipulation of images as pixels, enabling frame buffer effects such as anti-aliasing, motion blur, depth of field and soft shadows.
- A key feature of the design of OpenGL is the separation of interaction from rendering. OpenGL itself is concerned only with graphics rendering.

2.6 COLOR SYSTEMS

2.6.1 RGB

The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue. The main purpose of the RGB color model is for the sensing, representation and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography. Before the electronic age, the RGB color model already had a solid theory behind it, based in human perception of colors.

RGB is a device-dependent color model: different devices detect or reproduce a given RGB value differently, since the color elements (such as phosphors or dyes) and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, or even in the same device over time. Thus, an RGB value does not define the same color across devices without some kind of color management.

2.6.1 RGBA

RGBA stands for red green blue alpha. While it is sometimes described as a color space, it is actually the combination of an RGB color model with an extra 4th alpha channel. Alpha indicates how opaque each pixel is and allows an image to be combined over others using alpha compositing, with transparent areas and anti-aliasing of the edges of opaque regions.

In computer graphics, pixels encoding the RGBA color space information must be stored in computer memory (or in files on disk), in well-defined formats. There are several ways to encode RGBA colors, which can lead to confusion when image data is exchanged. These encodings are often denoted by the four letters in some order (e.g. RGBA, ARGB, etc.). Unfortunately, the interpretation of these 4-letter mnemonics is not well established, leading to further confusion. There are two typical ways to understand a mnemonic such as "RGBA".

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS

Processor	1.5Ghz or Above, Intel Pentium 4 AMD
R.A.M	256MB internal RAM
Hard Disk	2.5GB free disk space
Monitor	1024*768 display resolution with true color
Keyboard	Standard 101 keyboard
Backup Media	Floppy/Hard Disk
Mouse	PS/2, USB Mouse

Table 3.1.1 Hardware Requirements

3.2 SOFTWARE REQUIREMENTS

This graphics has been designed for WINDOWS Platform and uses CODE BLOCKS integrated environment.

Operating System	Windows98,Windows XP or higher/UBUNTU
Language Tool	C++
Compiler	Turbo C Compiler/Borland C/C++ compiler
Libraries	GL, GLU, GLUT
Id	Visual C++

Table 3.2.1 Software Requirements

3.3 Functional Requirements

These are statements of services the system should provide and do how the system respond to particular inputs and how the system should behave in particular situations i.e. it describes what system should do.

- Press Left arrow key to move the car left
- Press Right arrow key to move the car right
- Press space key to move the football
- Press P key to park and unpark the car

CHAPTER 4

IMPLEMENTATION

4.1 User Defined Functions:

Function for fence:

fence ():

- This function is used to build the fence and
- The x and y coordinates are calculated and specified

code

```
void fence(GLint xco,GLint yco)
{
fencebar(0,yco+60);
fencebar(0,yco+40);
fencebar(0,yco+20);
glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex2d(xco,yco);
glVertex2d(xco,yco+130);
glVertex2d(xco+15,yco+150);
glVertex2d(xco+30,yco+130);
glVertex2d(xco+30,yco);
glEnd();

}
```

fencebar() :

- Here we have the railings for the fence.

Code

```
void fencebar(GLint a, GLint b)
{
        glColor3f(0.5,0.0,0.0);
        glBegin(GL_POLYGON);
        glVertex2d(a,b);
        glVertex2d(a,b+5);
        glVertex2d(a+1400,b+5);
        glVertex2d(a+1400,b);
        glEnd();}
```

Functions for the building:

makebuilding ():

- Here we have calculated the coordinates for the building

Code:

```
void makebuilding()
{
glColor3f(0.647059,0.164706,0.164706);
glBegin(GL_POLYGON);
glVertex2d(900,360);
glVertex2d(900,380);
glVertex2d(1360,380);
glVertex2d(1360,360);
glEnd();
glBegin(GL_POLYGON);
glVertex2d(930,380);
glVertex2d(930,400);
glVertex2d(1330,400);
glVertex2d(1330,380);
glEnd();
}
```

glasswindows():

- Here we construct the windows for the house

Code

```
void glasswindows(GLint x, GLint y)
{
glColor3f(0.53,0.12,0.47);
glBegin(GL_QUADS);
glVertex2d(x,y);
glVertex2d(x,y+40);
glVertex2d(x+40,y+40);
glVertex2d(x+40,y);
```

```
glEnd();
glColor3f(0.576471, 0.858824, 0.439216);
glBegin(GL_QUADS);
glVertex2d(x+5,y+5);
glVertex2d(x+5,y+35);
glVertex2d(x+35,y+35);
glVertex2d(x+35,y+5);
glEnd();
}
```

door():

- Here we construct the door

Code:

```
void door(GLint x, GLint y)
{
glColor3f(0.62352,0.372549,0.623529);
glBegin(GL_QUADS);
glVertex2d(x,y);
glVertex2d(x,y+100);
glVertex2d(x+50,y+100);
glVertex2d(x+50,y);
glEnd();
}
```

Wheel():

- Here we use the circle_draw function to draw the door knobs and the decoration objects.

Code:

```
void wheel(GLint x, GLint y)
{
for(GLint i = 0; i<= 20;i++)
        circle_draw(x,y,i);
}
```

Function for cars:

vehicle():

- Here we draw the car along with the headlights, then we call this function multiple times to display the multiple cars, and they also have different colors.

Code:

```
void vehicle(GLint x, GLint y, int status, GLfloat col1, GLfloat col2, GLfloat col3,GLfloat col4, GLfloat col5, GLfloat col6)
{
glColor3f(col1,col2,col3);
glBegin(GL_POLYGON);
glVertex2d(x,y);
glVertex2d(x+200,y);
glVertex2d(x+200,y+50);
glVertex2d(x,y+50);
glEnd();
}
```

Function for the students:

man():

- In this function we have the creation of the male students, and the function is called again to display multiple students, and the color of their clothes are changed by passing different parameters for the colors in RGB scheme. Their different parts like the face, neck, eyes, hands and legs are constructed here

Code:

```
void man(){
        glutInit(&argc, argv);
        glutInitWindowSize(width,height);
        glutInitWindowPosition(0,0);
        glutCreateWindow("OpenGlNature");
        init();
        glutDisplayFunc(startscreen);
        glutMouseFunc(mouse);
        glutSpecialFunc( SpecialKeyFunc );
        glutKeyboardFunc( otherkeys );
        glutMainLoop();}
```

4.2 BUILTIN FUNCTION

PushMatrix And PopMatrix

Syntax: glPushMatrix( ); glPopMatrix( );

Description:

Pushes the current transformation matrix onto the matrix stack. The glPushMatrix() function saves the current coordinate system to the stack and glPopMatrix() restores the prior coordinate system.

Solid Sphere

Syntax:

Void glutSolidSphere (GLdouble radius , GLint slices, GLint stacks);

Parameters:

Radius: The radius of the sphere.

Slices: The number of subdivisions around the Z axis (similar to lines of longitude).

Stacks: The number of subdivisions along the Z axis (similar to lines of latitude).

Description:

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

get Async KeyState Function :

Syntax: SHORT GetAsyncKeyState(int vKey);

Parameters: vKey [in] **int;**
Specifies one of 256 possible key codes. You can use left- and right-distinguishing constants to specify certain keys.

Description:

The GetAsyncKeyState function determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to GetAsyncKeyState**.**

Return Value: **SHORT**

Post Redisplay:

Syntax: void glutPostRedisplay( );

Description:

glutPostRedisplay marks the normal plane of current window as needing to be redisplayed. glutPostRedisplay may be called within a window's display or overlay display callback to remark that window for redisplay.

Timer Function :

Syntax: void glutTimerFunc(unsigned int msecs, void(*func), int value);

Parameters:

msecs : Number of milliseconds to pass before calling the callback.

func : The timer callback function.

value : Integer value to pass to the timer callback.

Description:

glutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc.

Bitmap Character :

Syntax*:* void glutBitmapCharacter(void *font , int character);

Parameters:

Font : Bitmap font to use.

Character : Character to render (not confined to 8 bits).

Description*:*

Without using any display lists, glutBitmapCharacter renders the character in the named bitmapfont. The available fonts are: GLUT_BITMAP_TIMES_ROMAN_24 : A 24-point proportional sed Times Roman font.

Raster Position :

Syntax: void glRasterPos3f(GLfloat x, GLfloat y, GLfloat z);

Parameters:

x*:* Specifies the x-coordinate for the current raster position. y*:* Specifies the y-coordinate for the current raster position. z*:* Specifies the z-coordinate for the current raster position.

Description:

OpenGL maintains a 3-D position in window coordinates. This position, called the raster position, is maintained with subpixel accuracy. It is used to position pixel and bitmap write operations.

Color Function

Syntax: void glColor3ub(GLubyte red, GLubyte green, GLubyte blue);

Parameters:

red: The new red value for the current color.

green: The new green value for the current color. blue*:* The new blue value for the current color.

Description:

This function randomly generates different color based on the rand() function.

Keyboard Function

Syntax: void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));

Parameters:

func: The new keyboard callback function.

Description:

glutKeyboardFunc sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback.

ShadeModel

Syntax*:* void glShadeModel(GLenum mode);

Parameters*:*

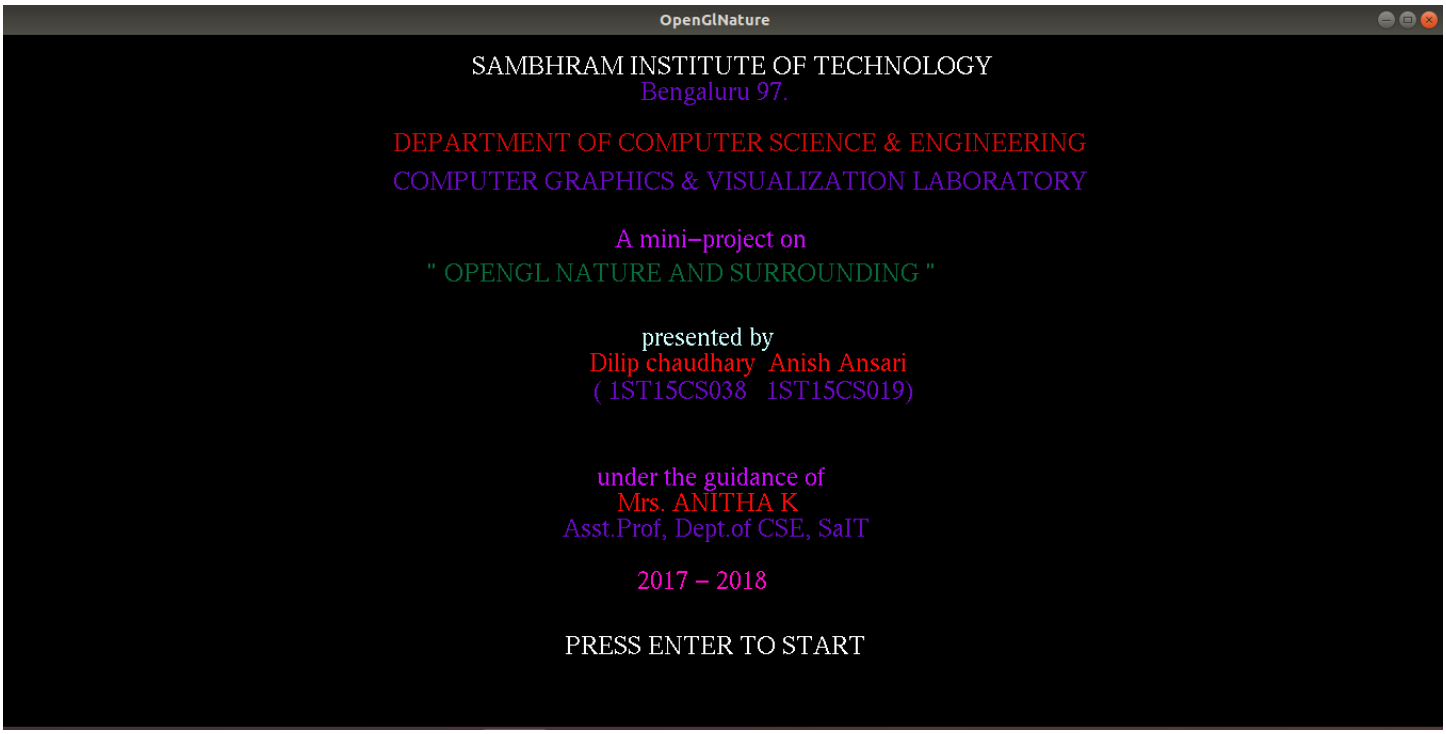
Mode: Specifies a symbolic value representing a shading technique. Accepted values are GL_FLAT and GL_SMOOTH. The initial value is GL_SMOOTH.

Description:

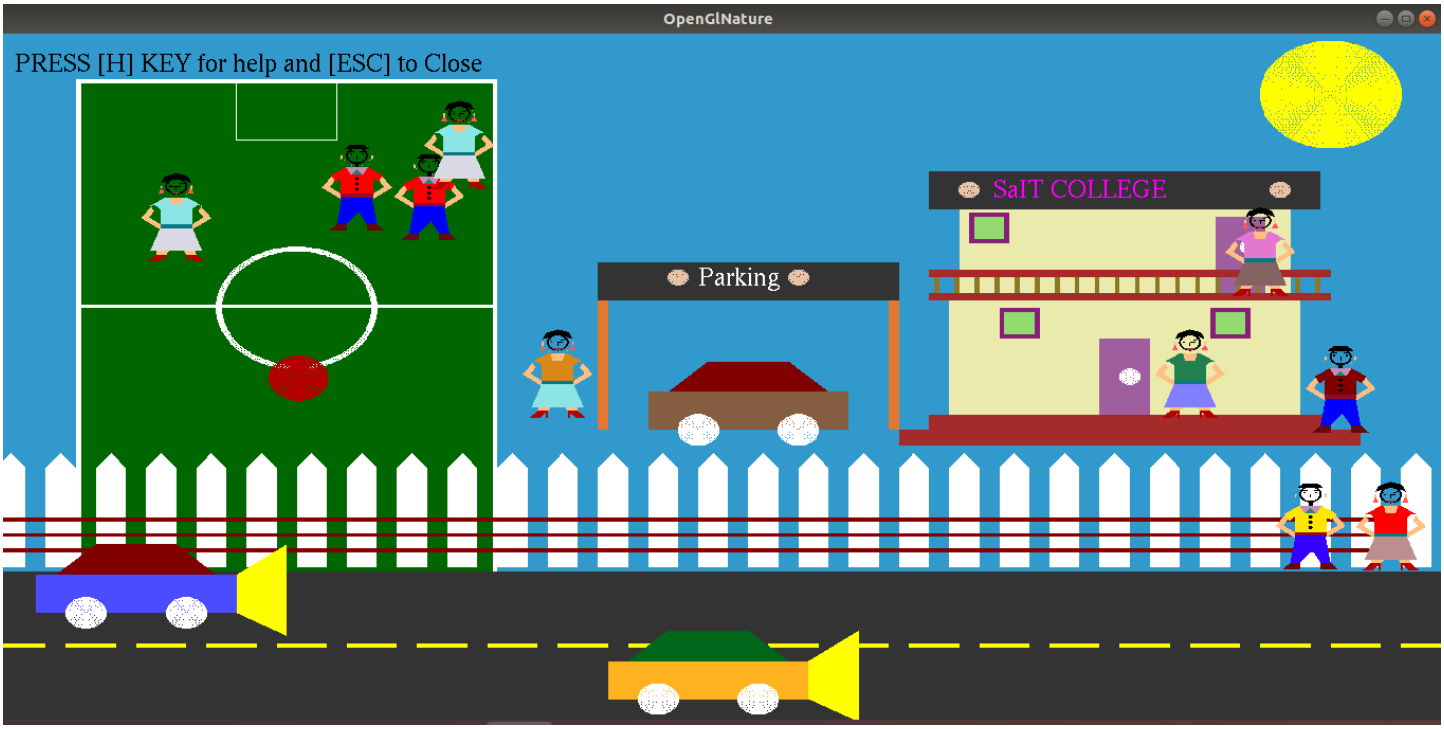
GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized typically assigning different colors to each resulting pixel fragment. Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.

CHAPTER 5

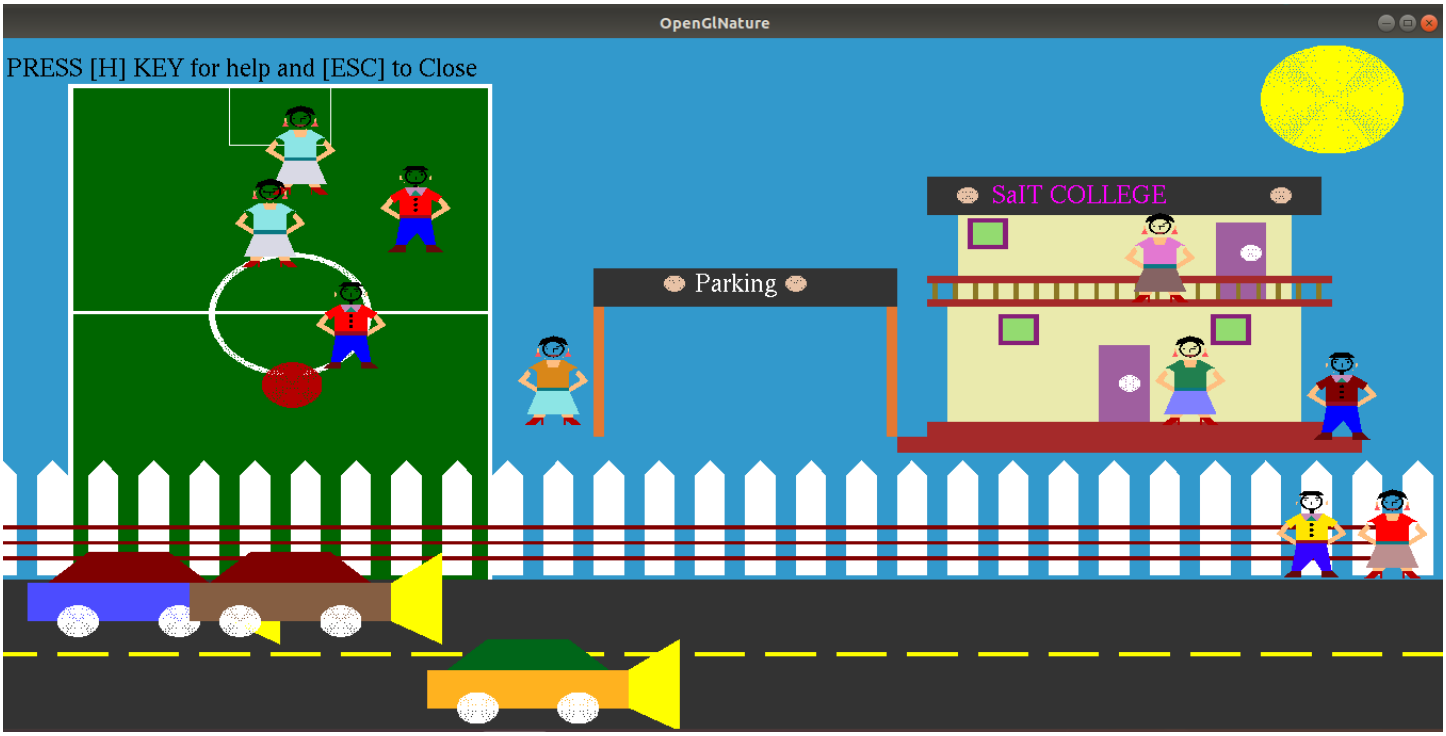
SNAPSHOTS



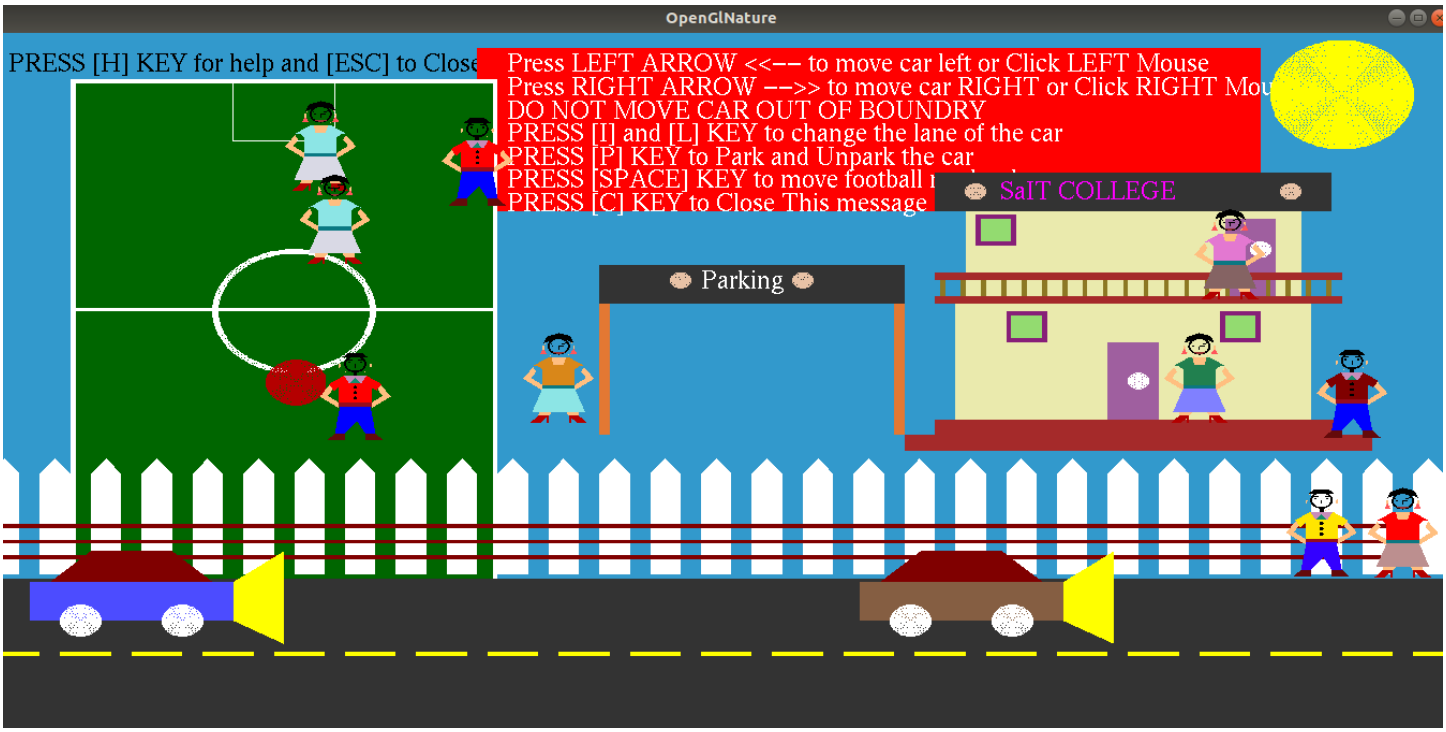
Snapshot 5.1: Home Page



Snapshot 5.2: Car parked at the parking area



Snapshot 5.3: Car unparked from the parking area



Snapshot 5.4: Demonstrating Help Menu

Snapshot 5.5: Demonstrating Context Menu

CONCLUSION AND FUTURE ENHANCEMENT

In this mini project we have simulated the Sambhram Institute of Technology using OpenGL library. We have simulated the components like students, building, players, basketball court, vehicle, road and so on. This simulation gives the idea to the viewer about the Sambhram Institute of Technology.

One of the primary advantages of the simulation is that, it enables the users to provide practical feedback when designing real world systems. This allows the designer to determine the correctness and efficiency of a design before the system is actually constructed. Consequently, the user may explore the merits of alternative deigns without actually, physically building the systems.

This simulation can be used as an effective means for teaching or demonstrating the related concepts to students. This is particularly true of simulators that make intelligent use of computer graphics visualization and also animation. Such simulators dynamically show the behavior and relationship of all the simulated system's components, thereby providing the user with meaningful understanding of the system's nature.

In future the following enhancements could be done:

- Providing Camera Movement. .
- Providing High Quality Graphics.
- Giving more functionalities to the cars.
- Improving the functions of the students and football players

BIBLOIOGRAPHY

- Donald Hearn & Pauline Baker: **"**Computer Graphics with OpenGL Version**"**,3rd / 4th Edition, Pearson Education,2011
- Edward Angel: **"**Interactive Computer Graphics- A Top Down approach with OpenGL**"**, 5th edition. Pearson Education, 2008
- https://www.openglprojects.in/2012/03/how-to-add-front-screen-in-your-opengl.html
- https://www.youtube.com/watch?v=cfiGI6l5sDU
- https://www.opengl.org/
- https://www.khronos.org/opengl/wiki/Getting_Started
- https://github.com/anishansari/openglnature-Simulation-of-SaIT