**1. INTRODUCTION TO COMPUTER GRAPHICS**

* 1. **Computer graphics**

Graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software. The interaction and understanding of computers and interpretation of data has been made easier because of computer graphics. A computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies and the video game industry.

Typically, the term *computer graphics* refers to several different things:

* the representation and manipulation of image data by a computer
* the various technologies used to create and manipulate images
* the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Computer generated imagery can be categorized into several different types: two dimensional (2D),three dimensional (3D), and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component".

**1.2 OpenGL**

**OpenGL** (**Open G**raphics **L**ibrary) is a cross-language, multi-platform API for rendering 2D and 3D computer graphics. The API is typically used to interact with a GPU, to achieve hardware-accelerated rendering.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, we must work through whatever windowing system controls the particular hardware we’re using. Similarly, OpenGL doesn’t provide high-level commands for describing models of three-dimensional objects. Such commands might allow us to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules.

With OpenGL, we must build up our desired model from a small set of *geometric primitives* - points, lines, and polygons. A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation.

In OpenGL, Rendering is the process by which a computer creates images from models. These *models*, or objects, are constructed from geometric primitives - points, lines, and polygons - that are specified by their vertices. The final rendered image consists of pixels drawn on the screen; a pixel is the smallest visible element the display hardware can put on the screen. Information about the pixels (for instance, what color they’re supposed to be) is organized in memory into bitplanes. A bitplane is an area of memory that holds one bit of information for every pixel on the screen; the bit might indicate how red a particular pixel is supposed to be, for example. The bitplanes are themselves organized into a *framebuffer*, which holds all the information that the graphics display needs to control the color and intensity of all the pixels on the screen.

**1.3 GLUT**

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres and the Utah teapot. GLUT also has some limited support for creating pop-up menus.

The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. Getting started with OpenGL while using GLUT often takes only a few lines of code and does not require knowledge of operating system–specific windowing APIs.

The GLUT library supports the following functionality:

* Multiple windows for OpenGL rendering.
* Callback driven event processing.
* An ‘idle’ routine and timers.
* Utility routines to generate various solid and wire frame objects.
* Support for bitmap and stroke fonts.
* Miscellaneous window management functions.

**1.4 SOIL**

SOIL (Simple OpenGL Image Library) is a tiny C library used primarily for uploading textures into OpenGL. It is based on stb\_image version 1.16, the public domain code from Sean Barrett. It can load TGA and DDS files, and perform common functions needed in loading OpenGL textures. SOIL can also be used to save and load images in a variety of formats (useful for loading height maps, non-OpenGL applications, etc.)

You can start using it in your project by linking with SOIL and adding the src directory to your include path. Although SOIL includes functions to automatically create a texture from an image, it uses features that aren't available in modern OpenGL. Because of this we'll simply use SOIL as image loader and create the texture ourselves.

**Features:**

1. **Readable Image Formats:**

* BMP - non-1bpp, non-RLE (from stb\_image documentation)
* PNG - non-interlaced (from stb\_image documentation)
* JPG - JPEG baseline (from stb\_image documentation)

1. **Writeable Image Formats:**

* TGA - Greyscale or RGB or RGBA, uncompressed
* BMP - RGB, uncompressed
* DDS - RGB as DXT1, or RGBA as DXT5

1. **Can load an image file directly into a 2D OpenGL texture, optionally performing the following functions:**

* Can generate a new texture handle, or reuse one specified
* Can automatically rescale the image to the next largest power-of-two size
* Can automatically create MIPmaps
* Can scale the RGB values into the "safe range" for NTSC displays
* Can multiply alpha on load
* Can flip the image vertically
* Can compress and upload any image as DXT1 or DXT5
* Can convert the RGB to YCoCg color space
* Will automatically downsize a texture if it is larger than GL\_MAX\_TEXTURE\_SIZE
* Can directly upload DDS files. Note: directly uploading the compressed DDS image will disable the other options (no flipping, no pre-multiplying alpha, no rescaling, no creation of MIPmaps, no auto-downsizing)

# 2. PROJECT DESCRIPTION

**2.1 SOFTWARE & HARDWARE REQUIREMENTS**

Hand-Cricket is designed to run successfully on any computer manufactured within the last five years. However, due to the high precision requirement of Hand-Cricket, some processes may take considerable time.

Hand-Cricket will run with appreciable speed at high resolution on any computer that meets the following specifications:

* Intel® 1.5 GHz (Intel® Pentium® 4 or better) or AMD® (AMD® Athlon® XP or better) equivalent.
* 1 GB of free hard disk space.
* 1GB of graphic memory
* CD Drive / USB Port for installation source.
* Ubuntu 12.04 or Higher Operating System.
* Color Monitor capable of 24-bit color and 1366x768 Resolution.
* PS/2 mouse and USB Keyboard for navigation.

In addition to the above, the Code::Blocks (Version 10.4 or later) with OpenGL MESA libraries along with SOIL maybe required for viewing and editing the Hand-Cricket Source Code. Hand-Cricket will not execute on Windows® due to library requirements.

### 2.2 SYSTEM DESIGN

**2.3 ARCHITECTURE**

START

score[0]=0

User Input u

CPU Input c

YES

NO

u==c

score[0]+=u

target=score[0]

score[1]=0

User Input u

CPU Input c

YES

NO

u==c

score[0]+=u

1

1

YES

NO

score[0] >score[1]

User Wins

YES

NO

score[0]==score[1]

Draw

User Loses

END

**3. API’S USED**

1. **glBegin (GLenum mode);**

It defines the type of primitives that the vertices define. Each subsequent execution og glVertex3f specifies the x,y,z coordinates of a location in space.

1. **glEnd (void);**

It ends the list of vertices.

1. **glClear (GLbitfield mask);**

It is used to make the screen solid and white.

1. **glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);**

The background color is set with this API where the last argument specifies a degree of transparency.

1. **glColor3f (GLfloat red, GLfloat green, GLfloat blue);**

It sets color to current drawing.

1. **glEnable (GLenum cap);**

We enable different algorithms by this function call.

1. **glDisable (GLenum cap);**

We disable different algorithms by this function call.

1. **glLoadIdentity (void);**

To initialize the current transform matrix to the identity transform.

1. **glMatrixMode (GLenum mode);**

It switches matrix mode between the two matrices –

MODEL\_VIEW (GL\_MODELVIEW) & PROJECTION (GL\_PROJECTION)

1. **glPushMatrix (void);**

It is used to push matrix on the stack.

1. **glPopMatrix (void);**

It is used to pop matrix from the stack.

1. **glRasterPos3f (GLfloat x, GLfloat y, GLfloat z);**

It is used to alter the shape, size, position of pixels.

1. **glTranslatef (GLfloat x, GLfloat y, GLfloat z);**

It is used to alter the model view or projection matrices.

1. **glVertex3f (GLfloat x, GLfloat y, GLfloat z);**

It is used to represent vertex.

1. **glViewport (GLint x, GLint y, GLsizei width, GLsizei height);**

It specifies that the viewport will have lower left corner(x,y) in screen co-ordinates and will be width pixels wide and height pixels high.

1. **glutBitmapCharacter(void \*font, int character);**

The character is placed at the present raster position on the display, is measured in pixels and can be altered by the various forms of the function glRasterPos\*.

1. **glutCreateWindow(const char \*title);**

It creates and opens OpenGL window.

1. **glutDisplayFunc(void (GLUTCALLBACK \*func)(void));**

It sends graphics to screen.

1. **glutInit(int \*argcp, char \*\*argv);**

It initiates interaction between windowing system and OpenGL.

1. **glutInitDisplayMode(unsigned int mode);**

This function specifies how the display should be initialized. The constants GLUT\_SINGLE and GLUT\_RGB, which are ORed together, indicate that a single display buffer should be allocated and the colors are specified using desired amount of red, green and blue.

1. **glutInitWindowSize(int width, int height);**

It sets the size of created window.

1. **glutKeyboardFunc(void (GLUTCALLBACK \*func)(unsigned char key, int x, int y));**

The keyboard event is generated when the mouse is in the window and one of the key is pressed or released. This GLUT function is the call bach for event generated by pressing a key.

1. **glutMainLoop(void);**

It causes the program to begin an event-processing loop.

1. **glutReshapeFunc(void (GLUTCALLBACK \*func)(int width, int height));**

The reshape event is generated whenever the window is resized, such as by a user interaction.

1. **glutSwapBuffers(void);**

We can swap the front and back buffers at will from the application programs.

1. **glBindTexture(GLenum target, GLuint texture);**

We can bind a named texture to a texturing target.

1. **glTexParameteri(GLenum target, GLenum pname, GLint param);**

We can assign a value or a set of values to the texture parameter.

1. **glScalef(GLfloat x,GLfloat y,GLfloat z);**

We can produce a non-uniform scaling along the x, y, and z axes. The three parameters indicate the desired scale factor along each of the three axes.

1. **glutMouseFunc(void (\*func)(int button, int state,int x, int y));**

We can set the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback.

1. **gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);**

We can set up a two-dimensional orthographic viewing region.

**4. IMPLEMENTATION**

The implementation of Hand Cricket begins with the declaration of the global variables:

* sw & sh which represent the screen resolution
* texID1,texID2,texID3,texID4, texID5,texID6,texID7,texID8,texID9, texIDg, texIDc, texIDhc, texIDp which represent the various textures which hold different images
* xRaster & yRaster which represent the positions for graph
* p[3] which holds the scoring mechanism
* stat[60] holds the various runs scored in each ball
* ran()

This will generate a random number which will be used as run hit by computer

* drawText()

This we are using to draw different texts with different scale this function tweeks original bitmap character function with scaling factor.

* drawNumber()

This we are using to draw numbers on our screen.

* GLuint LoadTexturehc(char\* file)

We use this to load texture to our window which takes file name as input parameter here we are loading handcricket logo

* GLuint LoadTexturep(char\* file)

We use this to load texture to our window which takes file name as input parameter here we are loading pes logo

* LoadMainGLTextures()

This we are using to load images to the textures and set them their respective ones.

* gamedisp1()

This function we are used to load main game screen with different textures in it like hand score and input screen.

* gamedisp2()

This we are using to draw the numbers through which user gives input.

* gamedisp3()

This we are using to draw scores and target at right bottom of the screen.

* howtoplay()

This function is used to load the how to play instructions for the user.

* splash()

This function will load first screen with all its textures defined.

* barchart()

This function is used to draw statistical bar graph at the end.

* credits()

This is used to call credits screen with some delay.

* game()

This used to call main game screen in order with delay.

* anim()

This is used to change the hand textute according to user input.

* action()

Will display the run scored by the user and computer.

* myreshape()

Used to reshape window whenever the user presses maximize button.

* main()

First we set the display mode to double. Then we initialize the window. After that we create the output window called “HandCricket”. One by one we are calling keyboard, mouse, display and reshape function.

# 5. SOURCE CODE

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

HAND-CRICKET

VERSION 1.0

BY Nagabharan.N & Naveen Kumar.M

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

HEADER FILES

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

#include <stdlib.h>

#include <iostream>

#include <stdio.h>

#include <math.h>

#include <GL/glut.h>

#include "SOIL.h"

using namespace std;

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

GLOBAL DECLARATIONS

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

int sw = 1370, sh = 768;

GLuint texID1,texID2,texID3,texID4, texID5,texID6,texID7,texID8,texID9, texIDg, texIDc, texIDhc, texIDp;

GLint xRaster=25,yRaster=150;

GLubyte label[30]={ '1' ,'2' ,'3' ,'4', '5', '6' ,'7' ,'8', '9' ,'10' ,'11', '12','13', '14' ,'15' ,'16' ,'17', '18' ,'19' ,'20' ,'21','22' ,'23', '24' ,'25' ,'26' ,'27' ,'28' ,'29' ,'30' };

int p[3],target,o,score,w1,w2,w3,out,e,lb=31, stat[60]={0},bc=0,i,m=0,n=0;

int ran(){

int a;

do{

a=rand();

a=a\*rand()%1000+rand()%100;

a=a/rand()%1000+rand()%100;

a=a\*rand()%1000+rand()%100;

a%=6;

a++;

}while(a==0||a<0);

return a;

}

GLuint LoadTexturehc(char\* file) {

GLuint texIdhc = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texIdhc);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texIdhc;

}

GLuint LoadTexturep(char\* file) {

GLuint texIdp = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texIdp);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texIdp;

}

GLuint LoadTexturec(char\* file) {

GLuint texIdc = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texIdc);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texIdc;

}

GLuint LoadTextureg(char\* file) {

GLuint texIdg = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texIdg);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texIdg;

}

GLuint LoadTexture1(char\* file) {

GLuint texId1 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId1);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId1;

}

GLuint LoadTexture2(char\* file) {

GLuint texId2 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId2);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId2;

}

GLuint LoadTexture3(char\* file) {

GLuint texId3 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId3);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId3;

}

GLuint LoadTexture4(char\* file) {

GLuint texId4 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId4);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId4;

}

GLuint LoadTexture5(char\* file) {

GLuint texId5 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId5);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId5;

}

GLuint LoadTexture6(char\* file) {

GLuint texId6 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId6);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId6;

}

GLuint LoadTexture7(char\* file) {

GLuint texId7 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId7);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId7;

}

GLuint LoadTexture8(char\* file) {

GLuint texId8 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId8);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId8;

}

GLuint LoadTexture9(char\* file) {

GLuint texId9 = SOIL\_load\_OGL\_texture(file, SOIL\_LOAD\_AUTO, SOIL\_CREATE\_NEW\_ID, SOIL\_FLAG\_INVERT\_Y | SOIL\_FLAG\_MIPMAPS);

glBindTexture(GL\_TEXTURE\_2D, texId9);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

return texId9;

}

void LoadMainGLTextures() {

char \*a="img/grnd.png";

texIDg = LoadTextureg((char\*)(a));

}

void LoadcrGLTextures() {

char \*a="img/cr.png";

texIDc = LoadTexturec((char\*)(a));

}

void LoadspGLTextures() {

char \*a="img/hc.png",\*b="img/pes.png";

texIDhc = LoadTexturehc((char\*)(a));

texIDp = LoadTexturep((char\*)(b));

}

void LoadGameGLTextures() {

char \*a="img/0l.png",\*b="img/0r.png",\*c="img/cpu.png",\*d="img/human.png";

texID1 = LoadTexture1((char\*)(a));

texID2 = LoadTexture2((char\*)(b));

texID3 = LoadTexture3((char\*)(c));

texID4 = LoadTexture4((char\*)(d));

texID5 = LoadTexture5((char\*)("img/win.png"));

texID6 = LoadTexture6((char\*)("img/lose.png"));

texID7 = LoadTexture7((char\*)("img/out.png"));

texID8 = LoadTexture8((char\*)("img/draw.png"));

texID9 = LoadTexture9((char\*)("img/gameover.png"));

}

void drawNumber(int n,float x,float y,float z,float a){

char \*c,string[50];

glPushMatrix();

glTranslatef(x-19.5,y-10,z);

glScalef(.5\*a,.5\*a,z);

int b=sprintf (string, " %d ",n);

for (c=string; \*c != '\0'; c++)

glutStrokeCharacter(GLUT\_STROKE\_ROMAN , \*c);

glPopMatrix();

}

void drawText(char\*string,int x,int y,int z,float a){

char \*c;

glPushMatrix();

glTranslatef(x,y,z);

glScalef(.5\*a,.5\*a,z);

for (c=string; \*c != '\0'; c++)

glutStrokeCharacter(GLUT\_STROKE\_ROMAN , \*c);

glPopMatrix();

}

void spdisp11(){

glColor4f(1,1,1,1);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texIDp);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(400,600);

glTexCoord2i(1,0); glVertex2i(400,200);

glTexCoord2i(0,0); glVertex2i(100,200);

glTexCoord2i(0,1); glVertex2i(100,600);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor4f(1,1,1,1);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texIDhc);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(1200,400);

glTexCoord2i(1,0); glVertex2i(1200,300);

glTexCoord2i(0,0); glVertex2i(600,300);

glTexCoord2i(0,1); glVertex2i(600,400);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(0,0,0);

glLineWidth(4);

drawText("PRESENTS",700,500,0,1);

}

void spdisp1(){

glColor3f(1,1,1);

spdisp11();

}

void hdisp1(){

glColor4f(1,1,1,1);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texIDg);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(sw,sh);

glTexCoord2i(1,0); glVertex2i(sw,0);

glTexCoord2i(0,0); glVertex2i(0,0);

glTexCoord2i(0,1); glVertex2i(0,sh);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(1,1,0);

glLineWidth(2);

drawText("By default the user bats first for 30 balls.",50,500,0,.5);

drawText("You can select 1 of 6 numbers present on the right side.",50,400,0,.5);

drawText("After the game your statistics is displayed",50,300,0,.5);

drawText("You can exit the game anytime by using ESC key",50,200,0,.5);

}

void crdisp11(){

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texIDc);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(sw,sh);

glTexCoord2i(1,0); glVertex2i(sw,0);

glTexCoord2i(0,0); glVertex2i(0,0);

glTexCoord2i(0,1); glVertex2i(0,sh);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(1,1,0);

glLineWidth(3);

drawText("CREDITS",550,650,0,1);

drawText("handcricket",500,550,0,1);

drawText("Written by",550,450,0,.75);

drawText("Nagabharan N ( 1PE10CS054 )",450,400,0,.5);

drawText("Naveen Kumar M ( 1PE10CS056 )",450,350,0,.5);

drawText("Special Thanks",500,250,0,.75);

drawText("Prof. Sarasvathi V",550,200,0,.5);

drawText("Friends",550,150,0,.5);

}

void crdisp1(){

glColor3f(1,1,1);

crdisp11();

}

void gamedisp11(){

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texID1);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(450,500);

glTexCoord2i(1,0); glVertex2i(450,200);

glTexCoord2i(0,0); glVertex2i(50,200);

glTexCoord2i(0,1); glVertex2i(50,500);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texID2);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(900,500);

glTexCoord2i(1,0); glVertex2i(900,200);

glTexCoord2i(0,0); glVertex2i(500,200);

glTexCoord2i(0,1); glVertex2i(500,500);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texID3);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(200,625);

glTexCoord2i(1,0); glVertex2i(200,525);

glTexCoord2i(0,0); glVertex2i(300,525);

glTexCoord2i(0,1); glVertex2i(300,625);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texID4);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(750,625);

glTexCoord2i(1,0); glVertex2i(750,525);

glTexCoord2i(0,0); glVertex2i(650,525);

glTexCoord2i(0,1); glVertex2i(650,625);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(1,0,0);

drawText("CPU",150,650,0,1.25);

drawText("CPU",151,650,0,1.25);

drawText("CPU",152,650,0,1.25);

drawText("HUMAN",570,650,0,1.25);

drawText("HUMAN",571,650,0,1.25);

drawText("HUMAN",572,650,0,1.25);

drawText("BALLS LEFT",330,550,0,.75);

drawText("BALLS LEFT",331,550,0,.75);

drawText("BALLS LEFT",332,550,0,.75);

drawNumber(30-bc,400,625,0,1);

if(out){

glEnable(GL\_TEXTURE\_2D);

glColor4f(1,1,1,1);

glBindTexture(GL\_TEXTURE\_2D, texID7);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(150,180);

glTexCoord2i(1,0); glVertex2i(150,20);

glTexCoord2i(0,0); glVertex2i(50,20);

glTexCoord2i(0,1); glVertex2i(50,180);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(1,0,0);

drawText("OUT",155,75,0,1);

drawText("OUT",156,75,0,1);

drawText("OUT",157,75,0,1);

}

if(w1){

glEnable(GL\_TEXTURE\_2D);

glColor4f(1,1,1,1);

glBindTexture(GL\_TEXTURE\_2D, texID5);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(900,180);

glTexCoord2i(1,0); glVertex2i(900,20);

glTexCoord2i(0,0); glVertex2i(700,20);

glTexCoord2i(0,1); glVertex2i(700,180);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(1,0,0);

drawText("USER WIN",355,75,0,1);

drawText("USER WIN",356,75,0,1);

drawText("USER WIN",357,75,0,1);

e=1;

}

if(w2){

glEnable(GL\_TEXTURE\_2D);

glColor4f(1,1,1,1);

glBindTexture(GL\_TEXTURE\_2D, texID6);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(900,180);

glTexCoord2i(1,0); glVertex2i(900,20);

glTexCoord2i(0,0); glVertex2i(700,20);

glTexCoord2i(0,1); glVertex2i(700,180);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(1,0,0);

drawText("CPU WIN",325,75,0,1);

drawText("CPU WIN",326,75,0,1);

drawText("CPU WIN",327,75,0,1);

e=1;

}

if(w3){

glEnable(GL\_TEXTURE\_2D);

glColor4f(1,1,1,1);

glBindTexture(GL\_TEXTURE\_2D, texID8);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(900,180);

glTexCoord2i(1,0); glVertex2i(900,20);

glTexCoord2i(0,0); glVertex2i(700,20);

glTexCoord2i(0,1); glVertex2i(700,180);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glColor3f(1,0,0);

drawText("DRAW",325,75,0,1);

drawText("DRAW",326,75,0,1);

drawText("DRAW",327,75,0,1);

e=1;

}

}

void gamedisp21(){

glBegin(GL\_POLYGON);

glVertex2d(1000,650);

glVertex2d(1125,650);

glVertex2d(1125,700);

glVertex2d(1000,700);

glEnd();

glBegin(GL\_POLYGON);

glVertex2d(1175,650);

glVertex2d(1300,650);

glVertex2d(1300,700);

glVertex2d(1175,700);

glEnd();

glBegin(GL\_POLYGON);

glVertex2d(1000,550);

glVertex2d(1125,550);

glVertex2d(1125,600);

glVertex2d(1000,600);

glEnd();

glBegin(GL\_POLYGON);

glVertex2d(1175,550);

glVertex2d(1300,550);

glVertex2d(1300,600);

glVertex2d(1175,600);

glEnd();

glBegin(GL\_POLYGON);

glVertex2d(1000,450);

glVertex2d(1125,450);

glVertex2d(1125,500);

glVertex2d(1000,500);

glEnd();

glBegin(GL\_POLYGON);

glVertex2d(1175,450);

glVertex2d(1300,450);

glVertex2d(1300,500);

glVertex2d(1175,500);

glEnd();

}

void gamedisp22(){

drawNumber(1,1050,675,0,.5);

drawNumber(2,1220,675,0,.5);

drawNumber(3,1050,575,0,.5);

drawNumber(4,1220,575,0,.5);

drawNumber(5,1050,475,0,.5);

drawNumber(6,1220,475,0,.5);

drawText("SELECT NUMBER",1015,720,0,.5);

drawText("SELECT NUMBER",1016,720,0,.5);

drawText("SELECT NUMBER",1017,720,0,.5);

}

void gamedisp31(int a,int b,int q){

drawText("SCORE",1000,300,0,1.5);

drawText("SCORE",1001,300,0,1.5);

drawText("SCORE",1002,300,0,1.5);

drawNumber(a,950,150,0,2);

if(b){

drawText("TARGET",950,50,0,.75);

drawText("TARGET",951,50,0,.75);

drawText("TARGET",952,50,0,.75);

drawNumber(q,1100,50,0,1.5);

}

}

void gamedisp1(){

glColor3f(0,0.5+(float)rand()/(float)RAND\_MAX,(float)rand()/(float)RAND\_MAX/.5);

glBegin(GL\_POLYGON);

glVertex2d(0,0);

glVertex2d(930,0);

glVertex2d(930,sh);

glVertex2d(0,sh);

glEnd();

glColor3f(1,1,1);

gamedisp11();

}

void gamedisp2(){

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2d(930,400);

glVertex2d(sw,400);

glVertex2d(sw,sh);

glVertex2d(930,sh);

glEnd();

glColor3f(0,0,0);

gamedisp21();

glColor3f(1,1,1);

gamedisp22();

}

void gamedisp3(){

glColor3f(0,.5,0);

glBegin(GL\_POLYGON);

glVertex2d(930,0);

glVertex2d(sw,0);

glVertex2d(sw,400);

glVertex2d(930,400);

glEnd();

glColor3f(1,1,1);

gamedisp31(score,o,target+1);

}

void newdisp(){

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnable(GL\_TEXTURE\_2D);

glColor4f(1,1,1,1);

glBindTexture(GL\_TEXTURE\_2D, texID9);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(sw,sh);

glTexCoord2i(1,0); glVertex2i(sw,0);

glTexCoord2i(0,0); glVertex2i(0,0);

glTexCoord2i(0,1); glVertex2i(0,sh);

glEnd();

glDisable(GL\_TEXTURE\_2D);

}

void splash(){

glClear(GL\_COLOR\_BUFFER\_BIT);

spdisp1();

glutSwapBuffers();

}

void howtoplay(){

glClear(GL\_COLOR\_BUFFER\_BIT);

if(!n){

splash();

sleep(3);

n=1;

}

hdisp1();

glutPostRedisplay();

}

void barchart(void) {

GLint month,k;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1,0,0);

drawText("STATISTICS",300,550,0,2);

drawText("Runs scored each ball",400,500,0,.5);

for(k=0;k<30;k++){

if(stat[k]==1) glColor3f(1.0,0.0,0.0);

if(stat[k]==2) glColor3f(0.0,1.0,0.0);

if(stat[k]==3) glColor3f(0.0,0.0,1.0);

if(stat[k]==4) glColor3f(1.0,1.0,0.0);

if(stat[k]==5) glColor3f(1.0,0.0,1.0);

if(stat[k]==6) glColor3f(0.0,1.0,1.0);

if(stat[k]==0) glColor3f(1.0,1.0,1.0);

if(stat[k]==-2) glColor3f(0.0,0.0,0.0);

glRecti(270+k\*30,180,280+k\*30,200+40\*stat[k]);

}

glColor3f(0.0,0.0,0.0);

xRaster=270;

for(month=0;month<30;month++){

glRasterPos2i(xRaster,yRaster);

for(k=month;k<month+1;k++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12,label[k]);

xRaster+=30;

}

glutSwapBuffers();

}

void credits(){

glClear(GL\_COLOR\_BUFFER\_BIT);

crdisp1();

glutSwapBuffers();

}

void game(){

glClear(GL\_COLOR\_BUFFER\_BIT);

if(!m){

howtoplay();

glutSwapBuffers();

sleep(13);

m=1;

} glLineWidth(2);

gamedisp1();

gamedisp2();

gamedisp3();

glutSwapBuffers();

if(e){

sleep(2);

barchart();

sleep(5);

credits();

sleep(4);

newdisp();

glutSwapBuffers();

}}

void anim(int n1,int n2){

char a[30],b[30];

switch(n1){

case 1:

sprintf(a,"img/%dl.png",n1);

break;

case 2:

sprintf(a,"img/%dl.png",n1);

break;

case 3:

sprintf(a,"img/%dl.png",n1);

break;

case 4:

sprintf(a,"img/%dl.png",n1);

break;

case 5:

sprintf(a,"img/%dl.png",n1);

break;

case 6:

sprintf(a,"img/%dl.png",n1);

break;

default:

cout<<"Invalid File\n";

}

switch(n2){

case 1:

sprintf(b,"img/%dr.png",n2);

break;

case 2:

sprintf(b,"img/%dr.png",n2);

break;

case 3:

sprintf(b,"img/%dr.png",n2);

break;

case 4:

sprintf(b,"img/%dr.png",n2);

break;

case 5:

sprintf(b,"img/%dr.png",n2);

break;

case 6:

sprintf(b,"img/%dr.png",n2);

break;

default:

cout<<"Invalid File\n";

}

texID1 = LoadTexture1((char\*)(a));

texID2 = LoadTexture2((char\*)(b));

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texID1);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(450,500);

glTexCoord2i(1,0); glVertex2i(450,100);

glTexCoord2i(0,0); glVertex2i(50,100);

glTexCoord2i(0,1); glVertex2i(50,500);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, texID2);

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

glBegin(GL\_POLYGON);

glTexCoord2i(1,1); glVertex2i(900,500);

glTexCoord2i(1,0); glVertex2i(900,100);

glTexCoord2i(0,0); glVertex2i(500,100);

glTexCoord2i(0,1); glVertex2i(500,500);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glutPostRedisplay();

}

void action(int n){

int c=ran();

switch(n){

case 1: anim(c,1); cout<<c<<" & 1\n"; break;

case 2: anim(c,2); cout<<c<<" & 2\n"; break;

case 3: anim(c,3); cout<<c<<" & 3\n"; break;

case 4: anim(c,4); cout<<c<<" & 4\n"; break;

case 5: anim(c,5); cout<<c<<" & 5\n"; break;

case 6: anim(c,6); cout<<c<<" & 6\n"; break;

default: cout<<"Invalid\n";

} if(c!=n){

out=0;

if(!o){

p[o]+=n;

stat[i]=n;

i++;

bc++;

if(bc==lb){

out=1;

bc=0;

target=p[0];

stat[i]=-2;

if(o==0)

score=0;

o++;

}

}

else{

p[o]+=c;

bc++;

if(bc==lb){

out=1;

bc=0;

o++;

}

}

score=p[o];

if(o && p[1]>p[0]){

cout<<"Game Over p2 won";

w2++;

for(int j=0;j<=i;j++) cout<<stat[j];

}

}

else{

out=1;

bc=0;

target=p[0];

stat[i]=-2;

if(o==0)

score=0;

o++;

}

if(o){

if(o>1&&p[0]>p[1]){

cout<<"Game Over p1 won";

w1++;

for(int j=0;j<=i;j++) cout<<stat[j];

}

if(o>1&&p[0]==p[1]){

cout<<"Game Over draw";

w3++;

for(int j=0;j<=i;j++) cout<<stat[j];

}

}

}

void mouse(int btn,int state,int x,int y){

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN){

if(!w1&&!w2&&!w3){

y=(sh-y);

if((x<=1125)&&(x>=1000)){

if((y<=700)&&(y>=650))

action(1);

else if((y<=600)&&(y>=550))

action(3);

else if((y<=500)&&(y>=450))

action(5);

}

if((x<=1300)&&(x>=1175)){

if((y<=700)&&(y>=650))

action(2);

else if((y<=600)&&(y>=550))

action(4);

else if((y<=500)&&(y>=450))

action(6);

}

}

}

if(btn==GLUT\_RIGHT\_BUTTON && state==GLUT\_DOWN && e){

cout<<"Cant quit";

exit(0);

}

}

void keyboard(unsigned char key, int x, int y){

switch(key){

case 27: cout<<"Exiting game\n";

exit(0);

break;

case 32 : if(e) exit(0);

else{

cout<<"Cant quit";

break;

}

default: cout<<"Invalid";

}

}

void winReshape(GLint newWidth,GLint newHeight) {

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,(GLdouble)newWidth,0.0,(GLdouble)newHeight);

}

void reshape(int w, int h){

glViewport (0, 0, (GLsizei) w, (GLsizei) h);

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ();

gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);

}

void myinit(){

LoadMainGLTextures();

LoadspGLTextures();

LoadGameGLTextures();

LoadcrGLTextures();

glEnable(GL\_TEXTURE\_2D);

glClearColor(1.0,1.0,1.0,1.0);

glPointSize(1.0);

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,sw,0.0,sh);

}

int main( int argc,char \*\*argv){

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);

glutInitWindowSize(sw,sh);

glutInitWindowPosition(0,0);

glutCreateWindow("HandCricket");

myinit();

glutDisplayFunc(game);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutReshapeFunc(reshape);

glutFullScreen();

glutMainLoop();

return 0;

}

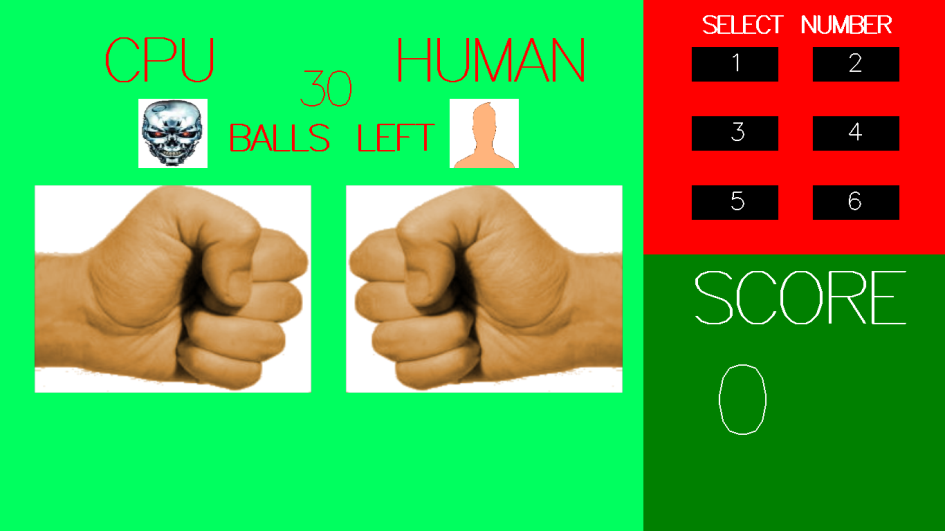
# 6. SAMPLE OUTPUTS



INTRODUCTION TO THE GAME



GAMEPLAY INSTRUCTIONS



MAIN GAME SCREEN



NUMBER SELECTION



SAMPLE OUT SITUATION



SAMPLE WIN SITUATION



CREDITS



GAME OVER

**7. CONCLUSION**

This project helps us to understand the various concepts used in Computer Graphics and Visualization lab like transformation (translation and scaling) and textures. We have also learnt how OpenGL uses built-in functions and algorithms to create complex graphical objects and render it to display.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. Most initial GLUT state is defined and the initial state is reasonable for simple programs. The GLUT routines also take relatively few parameters. No pointers are returned.

The SOIL library API’s are very user-friendly and easy to implement. They can be easily manipulated according to user convenience. They require relatively less parameter.

We, the developers, tried hard to integrate whatever we wished into Hand-Cricket to make it as interactive and user-friendly as possible. However, we encountered a few setbacks during the development stage which propelled us into modifying or altogether removing some features. We had initially planned to have the gameplay with the choice of toss where user can choose to bat or bowl. We wish to add more features to this project by providing multiplayer mode and test mode. We also planned to introduce various other statistical comparisons for giving a better view to the user.

# 8. BIBLIOGRAPHY

**Books referred:**

* Interactive Computer Graphics By Edward Angel (5th edition)
* Computer graphics using OpenGL: F.S.Hill, Jr

**Websites referred:**

* www.opengl.org/
* www.lonesock.net/soil.html
* nehe.gamedev.net/
* stackoverflow.com/