2D and 3D Graphics Processing Engine Design on LPC1769

AKHIL CHERUKURI
014525420
San Jose State University

Abstract— **The primary objective of the lab is to perform interfacing of the peripheral LCD Display module to the NXP LPC1769 LPCXpresso Board utilizing the Serial Peripheral Interface. The Program is designed in such way that the LPC1769 controls the pixel content displayed on the LCD. The program is implemented in 'C' programming language which displays 2D objects in the form of trees and 3D objects in the form of a 3D cube with a light source and a shadow on the LCD. Successful validation of 2D and 3D graphics processing engine design is shown on the LPC1769 LCD Module.**

I. INTRODUCTION

The main objective of this lab is to design and build a microprocessor unit that and control the LCD display using SPI Interface. The program code is written in 'C' language on an IDE and compiled. The compiled code is loaded on to the LPC1769 memory and is executed. The 2D and 3D objects defined in the code is seen as the output on the LCD node.

Graphics Processing Engine for 2D is drawing a patch of trees on the LCD display as a forest screensaver and for 3D it is showing the shadow of a 3D cube when a light source hits the object from a certain point.

This paper concentrates on providing detailed description of the software and hardware methodology implemented in the lab project. Along with this, it also includes details on the testing and verification. Diagrams and Images are also provided when needed to show a clear perspective or output.

The LPC module is connected to the computers USB port to get the required power supply. The LCD module draws in 3.3v from the LPC in built power supply. In LPC1769, pin numbers 5, 7, 8 are used for communication as we are using SPI port number 1 for our project code. The LCD peripheral is connected via appropriate pins with the LPC module's SPI port for communication.

II. METHODOLOGY

In this section, the system layout with hardware and software design is explained along with any technical difficulties and solutions encountered while developing the circuit or implementation of the program.

A. OBJECTIVES

The primary objective of the lab is to display the 2D and 3D images on the LCD with the help form LPC1769. It includes

1. Getting familiarized with the LPCXpresso IDE to test the LCD program that yields the result as expected.

2. Getting familiarized with the datasheets of NXP LPC1769 and Adafruit ST7735R LCD Module.

3. LCD module to SPI interface on the LPC1769.

4. Connect the pins between the LPC1769 and LCD module and checking connections to ensure correctness.

5. Understanding the pin functionalities of LPC1769 and LCD module, especially SPI pins that enable to display images on the LCD.

6. Understanding LCD hardware functionality of displaying images.

7. Thorough understanding of 2D and 3D graphics processing engine design.

8. 'C' language is required for programming the LPC1769 which controls the LCD module.

9. Synchronization establishment between the LCD and LPCXpresso 1769 board.

10. Debugging capability.

Technical challenges faced while building the project. They are as follows:

1. Displaying the images on the LCD Screen

2. Clearing the LCD display by user inputted interrupt.

B. SYSTEM LAYOUT

The power is supplied from the computer to LPC Module via USB Cable. The LCD Display Module is powered from the VCC and GND pins from the LPC1769 Module.

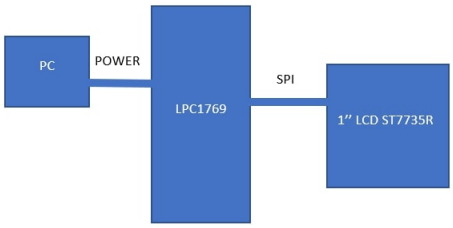

Fig. 1. SYSTEM LAYOUT

C. CIRCUIT DESIGN

LPCXpresso module is connected to the 1.8" 18 bit color TFT LCD display to ensure the functionality working as expected. This module is used to display images and shapes of geometric figures. It uses 4-wire SPI to communicate and has its own pixel addressable frame buffer. The resolution of the LCD display is 128 by 160. It includes a 3.3V regulator for low voltage drop out and 3/5 level shifter circuit so that we can have input power logic of 3V or 5V. The LPC module and LCD display are built on the wire wrapping board and are connected the required pin via connecting wires. Once the required hardware is built, MCUXpresso IDE Version 11 is utilized to program the LPC module.

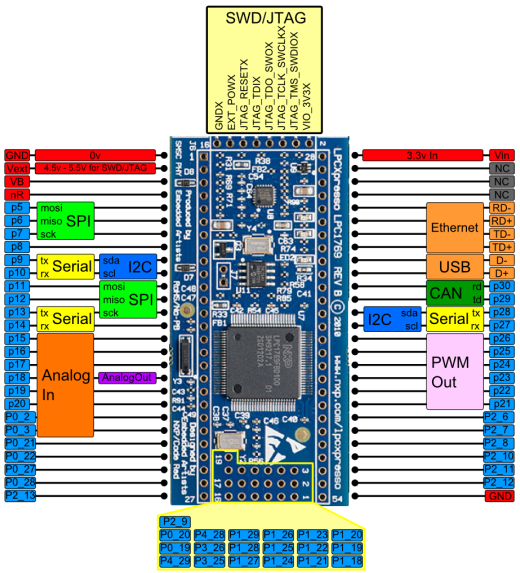

Fig. 2. LPC1769 PINOUT

Some of the techincal specifications for LCD module are as follows:

1. 1.8" diagonal LCD TFT display

2. 128x160 resolution

TABLE I
COMPONENT SPECIFICATIONS

Items	Description	Notes
CPU Module	LPC1769	Architecture
LCD Display	1.8" TFT 18bit Color LCD	Display
PC	Windows 10	MCUXpresso IDE

3. 18 bit color

4. ST7735R controlled with built in pixel-addressable video RAM buffer

5. 5V compatible power logic

6. Overall dimensions: 34 mm X 56mm X 6.5mm

III. SOFTWARE REQUIREMENT

Coding is done in which data is sent and processed by LCD in the form of frames. Every frame consists of a start of frame and end of frame for the LCD to differentiate the incoming data. Usually, the sequence would start with a header byte followed by command and then the data. At the end a frame tail would be present. On the whole, the data in each frame is limited ti 249 bytes maximum as the total including the header, command and tail will come up to 255 bytes.

The program is built using adafruit graphics library and NXPXpresso IDE. The software requirements involve initializing the LCD module using adafruit library for sending data buffer to the LCD as well as polling the GPIO pins for any external interrupt and based on the external interrupt taking actions for displaying content on the LCD screen.

We need the following to fullfill software requirement:

1. Windows/Mac/Linux

2. NXPXpresso IDE Version 11

3. External LCD Communication programming Opcode.

IV. CIRCUIT IMPLEMENTATION

The sample code is downloaded from Github and the zip file is loaded into the development environment. Later the code is modifed to set the pins and add functionality to the pins.

A. CIRCUIT DESIGN

First, We set up the connections between the LCD and LPC Module.

B. LCD INTERFACING CIRCUIT

The SPI interface circuit in built on the wire wrapping board of size 6x4 inches. It consists of external LCD and LPC1769 module. The circuit is connected to the computer using the USB cable. The detailed design and connections are show in the table below.

CPU MODULE	TFT LCD DISPLAY
PIN 28	LITE
NA	MISO
SCK1	SCK
MOSI0	MOSI
SSEL0	TFT_CS
NA	CARD_CS
PIN 23	D/C
PIN 23	RESET
PIN 28	VCC
PIN 1	GND

Fig. 3. Pin Connection between LCD and LPC

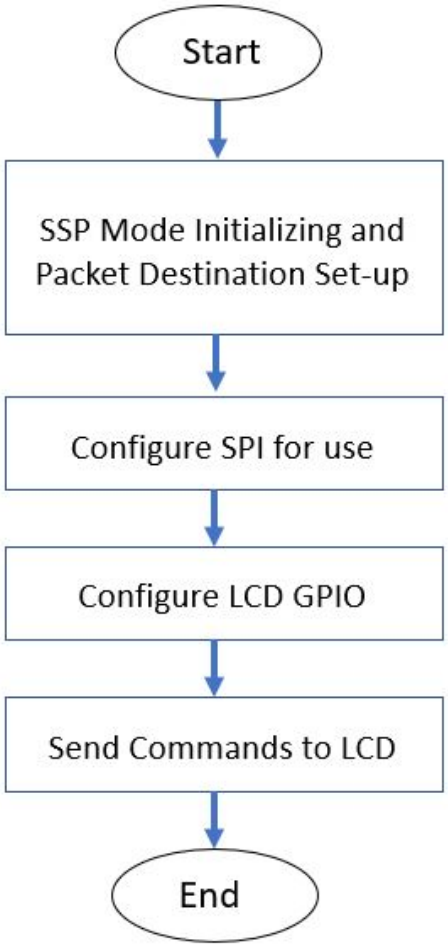

Fig. 4. ALORITHM FLOWCHART

V. ALGORITHM

Algorithm to control LCD functionality:

1. Initialize the SPI and the LCD.

2. Set the bit mask for the MOSI,SCK,SEEL pin of the SPI port

3. Define function opcode for LCD

4. Send commands to LCD to display desired image

Check Figure 4 for flowchart representation.

VI. TESTING AND VERIFICATION

Testing section mainly has to deal with testing the code and verifying the outputs. Procedures are followed to ensure working for the LCD. A multi-meter is used to ensure that no components are being shorted. This is done before debugging and actual testing is done.
When the program in debugged, the LCD module should be initialized and output should displayed. IDE needs to recognize the USB link for the programmed to the debugged and to generate actual output.

Random Trees: First, the center branch needed to be reduced by an expected portion. Second, we needed to create two other branches that are the center branch, but one is rotated counter-clockwise and the other clockwise. Once these characteristics created in a controlled manner, they could be reduced and rotated by a random reduced quantity and random angle, respectively.

Cube and Outlined Shadow: First, Cube was displayed. Second, was creating the point above the cube, which is treated as a source of light that creates the shadow below the cube. The four corners of the shadow are derived mathematically. All this is done in a 3D space, which is then projected onto a 2D space. This 2D space was displayed on the LCD display.

VII. CONCLUSION

Able to display 2D and 3D figures on the LCD using LPC.

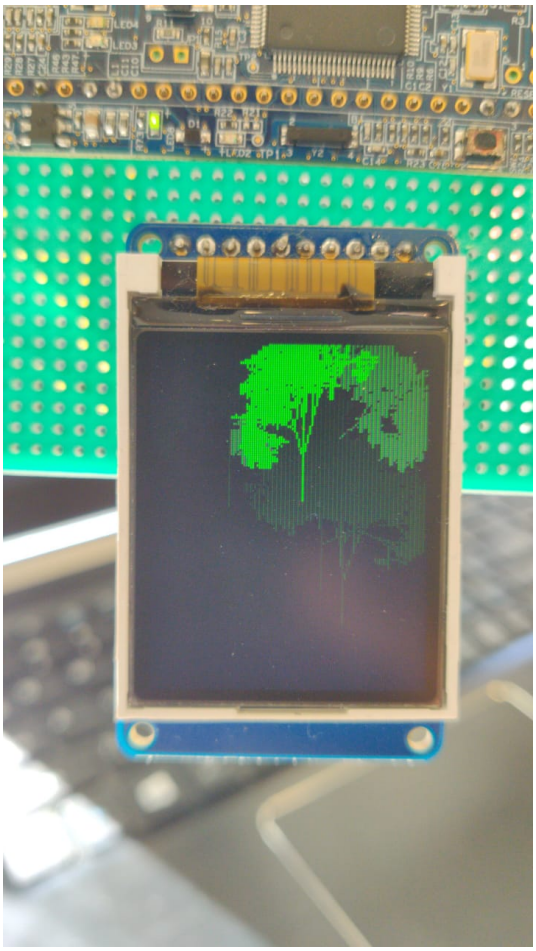

Fig. 5. Trees

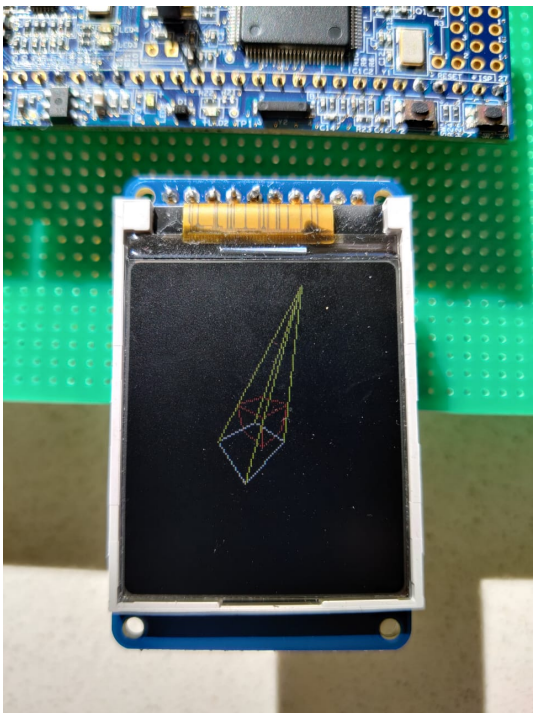

Fig. 6. 3D CUBE

VIII. ACKNOWLEDGMENT

The work and content described in this paper was achieved with the support of Dr. Harry Li. His lectures and discussion in class has contributed to completion of the lab assignments.

IX. REFERENCES

1. Dr. Harry Li, lecture notes of CMPE240 - Advanced Microcomputer Design, Computer Engineering Department, San Jose State University.

2. www.nxp.com LPC1769 Datasheet PDF

3.https://www.adafruit.com/product/358

4. UM10360 LPC176x/5x User Manual

5. github.com/hualili/CMPE240-Adv-Microprocessors

X. APPENDIX

Prototype Board:

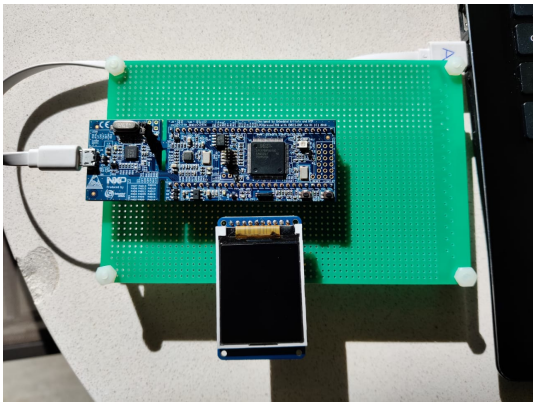

Fig. 7. Prototype board

Source code:

```
#include <cr_section_macros.h>
#include <NXP/crp.h>
#include "LPC17xx.h"                    /* LPC17xx definitions */
#include "ssp.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>



/* Be careful with the port number and location number, because

some of the location may not exist in that port. */

#define PORT_NUM          0


uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];


#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159

#define ST7735_CASET 0x2A
#define ST7735_RASET 0x2B
#define ST7735_RAMWR 0x2C
#define ST7735_SLPOUT 0x11
#define ST7735_DISPON 0x29



#define swap(x, y) {x = x + y; y = x - y; x = x - y ;}

// Defining Colors

#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF

// Self Defined Colors
#define YELLOW 0xFFFF00
#define BROWN 0xA52A2A
```

```
// Self Defined GREEN Shades
#define GREEN1 0x38761D
#define GREEN2 0x002200
#define GREEN3 0x00FC7C
#define GREEN4 0x32CD32
#define GREEN5 0x228B22
#define GREEN6 0x006400


int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;


void spiwrite(uint8_t c)

{

 int pnum = 1;

 src_addr[0] = c;

 SSP_SSELToggle( pnum, 0 );

 SSPSend( pnum, (uint8_t *)src_addr, 1 );

 SSP_SSELToggle( pnum, 1 );

}


void writecommand(uint8_t c)

{

 LPC_GPIO0->FIOCLR |= (0x1<<21);

 spiwrite(c);

}


void writedata(uint8_t c)

{

 LPC_GPIO0->FIOSET |= (0x1<<21);

 spiwrite(c);

}
```

```
void writeword(uint16_t c)
{
 uint8_t d;
 d = c >> 8;
 writedata(d);
 d = c & 0xFF;
 writedata(d);
}

void write888(uint32_t color, uint32_t repeat)
{
 uint8_t red, green, blue;
 int i;
 red = (color >> 16);
 green = (color >> 8) & 0xFF;
 blue = color & 0xFF;
 for (i = 0; i< repeat; i++) {
  writedata(red);
  writedata(green);
  writedata(blue);
 }
}

void setAddrWindow(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1)
{
```

```
 writecommand(ST7735_CASET);
 writeword(x0);
 writeword(x1);
 writecommand(ST7735_RASET);
 writeword(y0);
 writeword(y1);
}

void fillrect(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)
{
 //int16_t i;
 int16_t width, height;
 width = x1-x0+1;
 height = y1-y0+1;
 setAddrWindow(x0,y0,x1,y1);
 writecommand(ST7735_RAMWR);
 write888(color,width*height);
}

void lcddelay(int ms)
{
 int count = 24000;
 int i;
 for ( i = count*ms; i--; i > 0);
}

void lcd_init()
```

```
{

 int i;
 printf("Lab LPC-ARM Architecture Based 2D and 3D Graphics Processing Engine Design\n");
 // Set pins P0.16, P0.21, P0.22 as output
 LPC_GPIO0->FIODIR |= (0x1<<16);

 LPC_GPIO0->FIODIR |= (0x1<<21);

 LPC_GPIO0->FIODIR |= (0x1<<22);

 // Hardware Reset Sequence
 LPC_GPIO0->FIOSET |= (0x1<<22);
 lcddelay(500);

 LPC_GPIO0->FIOCLR |= (0x1<<22);
 lcddelay(500);

 LPC_GPIO0->FIOSET |= (0x1<<22);
 lcddelay(500);

 // initialize buffers
 for ( i = 0; i < SSP_BUFSIZE; i++ )
 {

   src_addr[i] = 0;
   dest_addr[i] = 0;
 }

 // Take LCD display out of sleep mode
 writecommand(ST7735_SLPOUT);
 lcddelay(200);

 // Turn LCD display on
 writecommand(ST7735_DISPON);
 lcddelay(200);

}




void drawPixel(int16_t x, int16_t y, uint32_t color)

{

 if ((x < 0) || (x >= _width) || (y < 0) || (y >= _height))

 return;

 setAddrWindow(x, y, x + 1, y + 1);

 writecommand(ST7735_RAMWR);
```

```
 write888(color, 1);

}

// Coordinate to Cartesian
int16_t xCartesian(int16_t x){
        x = x + (_width>>1);
        return x;
}

int16_t yCartesian(int16_t y){
        y = (_height>>1) - y;
        return y;
}

void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)

{

        // Cartesian Implementation
                x0 = xCartesian(x0);
                y0 = yCartesian(y0);
                x1 = xCartesian(x1);
                y1 = yCartesian(y1);

 int16_t slope = abs(y1 - y0) > abs(x1 - x0);

 if (slope) {

  swap(x0, y0);

  swap(x1, y1);

 }

 if (x0 > x1) {

  swap(x0, x1);

  swap(y0, y1);

 }

 int16_t dx, dy;

 dx = x1 - x0;

 dy = abs(y1 - y0);

 int16_t err = dx / 2;

 int16_t ystep;
```

```
 if (y0 < y1) {
  ystep = 1;
 }
 else {
  ystep = -1;
 }
 for (; x0 <= x1; x0++) {
  if (slope) {
   drawPixel(y0, x0, color);
  }
  else {
   drawPixel(x0, y0, color);
  }
  err -= dy;
  if (err < 0) {
   y0 += ystep;
   err += dx;
  }
 }
}


//2D Engine

#define DeltaX  0
#define DeltaY     0
#define pi 3.1416
// given lamda = 0.8
float lambda = 0.8;

// random lamda values
int rand_lambda = 5;
```

```
float array_lamda[] = {0.8-0.1, 0.8-0.05, 0.8, 0.8+0.05, 0.8+0.1};
int angles = 8;

// alpha degrees =5,10,15,20,30,40,50,60
float alpha[] = {pi/36, pi/18, pi/12, pi/9, pi/6, (2*pi)/9, (5*pi)/18, pi/3};

// colors for green leaves
uint32_t leaves[7] = {GREEN1, GREEN2, GREEN3, GREEN , GREEN4, GREEN5, GREEN6};


struct point{
        int x;
        int y;
};

//
struct point temp_point(int x, int y){
        struct point temp;
        temp.x = x;
        temp.y = y;

        return temp;
}

// Pre-processing

// Translate X Coordinate
// Tree branch reduction
// x= x1 + 0.8 * (x1-x0)

int16_t new_x(int16_t x0, int16_t x1){
        int16_t x;
        x = x1 + lambda*(x1 - x0);
        return x;
}

// Translate Y Coordinate
// Tree branch reduction
// y= y1 + 0.8 * (y1-y0)

int16_t new_y(int16_t y0, int16_t y1){
        int16_t y;
        y = y1 + lambda*(y1 - y0);
        return y;
}

struct point new_xy(struct point p0, struct point p1){
        struct point point;
        float z = array_lamda[rand()%rand_lambda];

        point = temp_point(p1.x + z * (p1.x - p0.x), p1.y + z * (p1.y - p0.y));

        return point;
```

```
}

// Random XY

struct point rand_xy(void){
return temp_point(rand()%64 - 32, rand()%80 - 60);
}

// CLOCKWISE BRANCH

struct point CW(struct point p0, struct point p1){
        struct point delta, point, prime, doubleprime, temp;
        // random angles '-'
        float deg = -alpha[rand()%angles];

        point = new_xy(p0, p1);
        delta = temp_point(-p1.x, -p1.y);
        prime = temp_point(point.x + delta.x, point.y + delta.y);
        doubleprime = temp_point(cos(deg)*prime.x - sin(deg)*prime.y, sin(deg)*prime.x + cos(deg)*prime.y);
        temp = temp_point(doubleprime.x - delta.x, doubleprime.y - delta.y);

        return temp;
}

// COUNTER CLOCKWISE BRANCH

struct point CCW(struct point p0, struct point p1){
   struct point delta, point, prime, doubleprime, temp;
   // random angles '+'
   float deg = alpha[rand()%angles];

   point = new_xy(p0, p1);
        delta = temp_point(-p1.x, -p1.y);
        prime = temp_point(point.x + delta.x, point.y + delta.y);
        doubleprime = temp_point(cos(deg)*prime.x - sin(deg)*prime.y, sin(deg)*prime.x + cos(deg)*prime.y);
        temp = temp_point(doubleprime.x - delta.x, doubleprime.y - delta.y);

        return temp;
}

// MIDDLE BRANCH

struct point NW(struct point p0, struct point p1){
        struct point point_nw;

        point_nw = temp_point(new_x(p0.x, p1.x), new_y(p0.y, p1.y));

        return point_nw;
}


void drawTree(){
// given least lvls=10
```

```
int lvls = 11;
struct point b[lvls + 2];
int z;
int ccw = 0, nw = 1, cw = 2, s = 3;
int al[lvls];

while(1){
struct point root = rand_xy();
z = rand()%4;

b[0] = temp_point(root.x, root.y);
b[1] = temp_point(root.x, root.y + 20);
drawLine(b[0].x,b[0].y,b[1].x,b[1].y, leaves[z]);

for(al[0] = 0; al[0] < s; al[0]++){
        if(al[0] == ccw){
                b[2] = CCW(b[0], b[1]);
                                        }
        else if(al[0] == nw){
                b[2] = NW(b[0], b[1]);
                                        }
        else if(al[0] == cw){
                b[2] = CW(b[0], b[1]);
                                        }
drawLine(b[1].x, b[1].y, b[2].x, b[2].y, leaves[z]);

for(al[1] = 0; al[1] < s; al[1]++){
        if(al[1] == ccw){
                b[3] = CCW(b[1], b[2]);
                                                }
        else if(al[1] == nw){
                b[3] = NW(b[1], b[2]);
                                                }
        else if(al[1] == cw){
                b[3] = CW(b[1], b[2]);
                                                }
drawLine(b[2].x, b[2].y, b[3].x, b[3].y, leaves[z]);

for(al[2] = 0; al[2] < s; al[2]++){
        if(al[2] == ccw){
                b[4] = CCW(b[2], b[3]);
                                                        }
        else if(al[2] == nw){
                b[4] = NW(b[2], b[3]);
                                                        }
        else if(al[2] == cw){
                b[4] = CW(b[2], b[3]);
                                                        }
drawLine(b[3].x, b[3].y, b[4].x, b[4].y, leaves[z]);

for(al[3] = 0; al[3] < s; al[3]++){
        if(al[3] == ccw){
                b[5] = CCW(b[3], b[4]);
```

```
                                                                    }
        else if(al[3] == nw){
                b[5] = NW(b[3], b[4]);
                                                                    }
        else if(al[3] == cw){
                b[5] = CW(b[3], b[4]);
                                                                    }

drawLine(b[4].x, b[4].y, b[5].x, b[5].y, leaves[z]);

for(al[4] = 0; al[4] < s; al[4]++){
        if(al[4] == ccw){
                b[6] = CCW(b[4], b[5]);
                                                            }
        else if(al[4] == nw){
                b[6] = NW(b[4], b[5]);
                                                            }
        else if(al[4] == cw){
                b[6] = CW(b[4], b[5]);
                                                            }
drawLine(b[5].x, b[5].y, b[6].x, b[6].y, leaves[z]);

for(al[5] = 0; al[5] < s; al[5]++){
        if(al[5] == ccw){
                b[7] = CCW(b[5], b[6]);
                                                                }
        else if(al[5] == nw){
                b[7] = NW(b[5], b[6]);
                                                                }
        else if(al[5] == cw){
                b[7] = CW(b[5], b[6]);
                                                                }
drawLine(b[6].x, b[6].y, b[7].x, b[7].y, leaves[z]);

for(al[6] = 0; al[6] < s; al[6]++){
        if(al[6] == ccw){
                b[8] = CCW(b[6], b[7]);
                                                                    }
        else if(al[6] == nw){
                b[8] = NW(b[6], b[7]);
                                                                }
        else if(al[6] == cw){
                b[8] = CW(b[6], b[7]);
                                                                    }
drawLine(b[7].x, b[7].y, b[8].x, b[8].y, leaves[z]);

for(al[7] = 0; al[7] < s; al[7]++){
        if(al[7] == ccw){
                b[9] = CCW(b[7], b[8]);
                                                                        }
        else if(al[7] == nw){
                b[9] = NW(b[7], b[8]);
                                                                        }
```

```
        else if(al[7] == cw){
                b[9] = CW(b[7], b[8]);
                                                                                                        }
drawLine(b[8].x, b[8].y, b[9].x, b[9].y, leaves[z]);

for(al[8] = 0; al[8] < s; al[8]++){
        if(al[8] == ccw){
                b[10] = CCW(b[8], b[9]);
                                                                                                        }
        else if(al[8] == nw){
                b[10] = NW(b[8], b[9]);
                                                                                                        }
        else if(al[8] == cw){
                b[10] = CW(b[8], b[9]);
                                                                                                        }
drawLine(b[9].x, b[9].y, b[10].x, b[10].y, leaves[z]);

for(al[9] = 0; al[9] < s; al[9]++){
        if(al[9] == ccw){
                b[11] = CCW(b[9], b[10]);

        }
        else if(al[9] == nw){
                b[11] = NW(b[9], b[10]);

        }
        else if(al[9] == cw){
                b[11] = CW(b[9], b[10]);

        }
drawLine(b[10].x, b[10].y, b[11].x, b[11].y, leaves[z]);

                                                                                                }
                                                                                        }
                                                                                }
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }
        }
}
}
}

// 3D ENGINE

float Lambda3D(float Zi,float Zs)
{
        float lambda;
        lambda = -Zi/(Zs-Zi);
        return lambda;
}
```

```
typedef struct{
	float_t x_value;
	float_t y_value;
	float_t z_value;
}Pts3D;

Pts3D ShadowPoint3D(Pts3D Pi, Pts3D Ps, float lambda)
{
	Pts3D pt;
	pt.x_value = (Pi.x_value + (lambda*(Ps.x_value-Pi.x_value)));
	pt.y_value = (Pi.y_value + (lambda*(Ps.y_value-Pi.y_value)));
	pt.z_value = (Pi.z_value + (lambda*(Ps.z_value-Pi.z_value)));

	return pt;
}

void drawShadow(){
#define UpperBD 52
#define NumOfPts 10
#define Pi 3.1415926
typedef struct{
		float X[UpperBD];
		float Y[UpperBD];
		float Z[UpperBD];

	}pworld;
typedef struct{
		float X[UpperBD];
		float Y[UpperBD];
		float Z[UpperBD];

	}pviewer;
typedef struct{
		float X[UpperBD];
		float Y[UpperBD];
	}pperspective;

// E(200,200,200),

float_t Xe=200;
float_t Ye=200;
float_t Ze=200;

float_t Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Xe,2));
float_t D_focal=40;
float_t L1,L2,L3,L4;

pworld WCS;
pviewer V;
pperspective P;

// X Y Z World Coordinate System
```

```
WCS.X[0]=0.0; WCS.Y[0]=0.0; WCS.Z[0]=0.0; //origin
WCS.X[1]=50.0; WCS.Y[1]=0.0; WCS.Z[1]=0.0;
WCS.X[2]=0.0; WCS.Y[2]=50.0; WCS.Z[2]=0.0;
WCS.X[3]=0.0; WCS.Y[3]=0.0;          WCS.Z[3]=50.0;

// Elevate Cube along Zw axis by 10

WCS.X[4]=100.0; WCS.Y[4]=0; WCS.Z[4]=10.0;
WCS.X[5]=0.0; WCS.Y[5]=100.0; WCS.Z[5]=10.0;
WCS.X[6]=0.0; WCS.Y[6]=0.0; WCS.Z[6]=110.0;

WCS.X[7]=100.0; WCS.Y[7]=0.0; WCS.Z[7]=110.0;
WCS.X[8]=100.0; WCS.Y[8]=100.0; WCS.Z[8]=10.0;
WCS.X[9]=0.0; WCS.Y[9]=100.0; WCS.Z[9]=110.0;

WCS.X[10]=100.0; WCS.Y[10]=100.0; WCS.Z[10]=110.0;

// POINT OF LIGHT SOURCE PS(-50, 50, 300)

WCS.X[11]=-50.0; WCS.Y[11]=50.0; WCS.Z[11]=300.0;


Pts3D Ps;
Ps.x_value = WCS.X[11]; Ps.y_value = WCS.Y[11]; Ps.z_value = WCS.Z[11];

Pts3D P1;
P1.x_value = WCS.X[6]; P1.y_value = WCS.Y[6]; P1.z_value = WCS.Z[6];

Pts3D P2;
P2.x_value = WCS.X[7]; P2.y_value = WCS.Y[7]; P2.z_value = WCS.Z[7];

Pts3D P3;
P3.x_value = WCS.X[9]; P3.y_value = WCS.Y[9]; P3.z_value = WCS.Z[9];

Pts3D P4;
P4.x_value = WCS.X[10]; P4.y_value = WCS.Y[10]; P4.z_value = WCS.Z[10];


// Shadow Points

Pts3D S1,S2,S3,S4;
L1 = Lambda3D(WCS.Z[6],WCS.Z[11]);
L2 = Lambda3D(WCS.Z[7],WCS.Z[11]);
L3 = Lambda3D(WCS.Z[9],WCS.Z[11]);
L4 = Lambda3D(WCS.Z[10],WCS.Z[11]);
S1 = ShadowPoint3D(P1,Ps,L1);
S2 = ShadowPoint3D(P2,Ps,L2);
S3 = ShadowPoint3D(P3,Ps,L3);
S4 = ShadowPoint3D(P4,Ps,L4);

WCS.X[12]=S1.x_value;          WCS.Y[12]=S1.y_value;          WCS.Z[12]=S1.z_value; //S1
WCS.X[13]=S2.x_value;          WCS.Y[13]=S2.y_value;          WCS.Z[13]=S2.z_value; //S2
```

```
WCS.X[14]=S3.x_value;          WCS.Y[14]=S3.y_value;          WCS.Z[14]=S3.z_value; //S3
WCS.X[15]=S4.x_value;          WCS.Y[15]=S4.y_value;          WCS.Z[15]=S4.z_value; //S4


float sPheta = Ye/sqrt(pow(Xe,2)+pow(Ye,2));
float cPheta = Xe/sqrt(pow(Xe,2)+pow(Ye,2));
float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/Rho;
float cPhi = Ze/Rho;

// WORLD TO VIEWER TRANSFROM

for(int i=0;i<=UpperBD;i++)
{
        V.X[i]=-sPheta*WCS.X[i]+cPheta*WCS.Y[i];
        V.Y[i] = -cPheta * cPhi * WCS.X[i]- cPhi * sPheta * WCS.Y[i]+ sPhi * WCS.Z[i];
        V.Z[i] = -sPhi * cPheta * WCS.X[i]- sPhi * cPheta * WCS.Y[i]-cPheta * WCS.Z[i] + Rho;
}

for(int i=0;i<=UpperBD;i++)
{
        P.X[i]=D_focal*V.X[i]/V.Z[i];
        P.Y[i]=D_focal*V.Y[i]/V.Z[i];

}

// CUBE DRAWLINES

drawLine(P.X[7],P.Y[7],P.X[4],P.Y[4],RED);
drawLine(P.X[7],P.Y[7],P.X[6],P.Y[6],RED);
drawLine(P.X[7],P.Y[7],P.X[10],P.Y[10],RED);
drawLine(P.X[8],P.Y[8],P.X[4],P.Y[4],RED);
drawLine(P.X[8],P.Y[8],P.X[5],P.Y[5],RED);
drawLine(P.X[8],P.Y[8],P.X[10],P.Y[10],RED);
drawLine(P.X[9],P.Y[9],P.X[6],P.Y[6],RED);
drawLine(P.X[9],P.Y[9],P.X[5],P.Y[5],RED);
drawLine(P.X[9],P.Y[9],P.X[10],P.Y[10],RED);

// SHADOW DRAWLINES

drawLine(P.X[12],P.Y[12],P.X[13],P.Y[13],WHITE);
drawLine(P.X[13],P.Y[13],P.X[15],P.Y[15],WHITE);
drawLine(P.X[14],P.Y[14],P.X[15],P.Y[15],WHITE);
drawLine(P.X[14],P.Y[14],P.X[12],P.Y[12],WHITE);

// LIGHT RAYS DRAWLINES

drawLine(P.X[12],P.Y[12],P.X[11],P.Y[11],YELLOW);
drawLine(P.X[13],P.Y[13],P.X[11],P.Y[11],YELLOW);
drawLine(P.X[14],P.Y[14],P.X[11],P.Y[11],YELLOW);
drawLine(P.X[15],P.Y[15],P.X[11],P.Y[11],YELLOW);

// DISPLAY CONSOLE RAY EQUATION
//for(int i=0;i<=10;i++)
```

```
//printf("X[%d]=%f,Y[%d]=%f\n",i,perspective.X[i],i,perspective.Y[i]);
//printf("1:%f,2:%f,3:%f,4:%f \n",lambda1,lambda2,lambda3,lambda4);

}



/*

 Main Function main()

*/



int main (void)

{

        uint32_t pnum = PORT_NUM;

        pnum = 1 ;

        if ( pnum == 1 )
                SSP1Init();

        else
                puts("Port number is not correct");

        lcd_init();

        fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, BLACK);
        int choice;
        printf("Enter 1 for 2D Tree and 2 for 3D Shadow");
        scanf("%d",&choice);
        if(choice == 1)
        {
        drawTree();
        }
        else if(choice == 2)
        {
        drawShadow();
        }
        else{
                printf("Wrong Choice");
        }


         return 0;

}
```