3D Vector Graphics Display Interface with LPC 1769

Karan Daryani *(*013722748*)*
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 95112
E-mail: karan.daryani@sjsu.edu

Abstract

LPC176x/5x does not consist a built-in Graphics Engine. This lab contains design, and development and implementation of graphics engine using Liquid Color Display. The work concentrates on implementation, design and testing of hardware and software components required for interfacing an LCD module to LPCXpresso 1769 thereby establishing a serial communication between two. The LCD module connected to the LPC module displays the 3D image of a cube with initial of our names on the top surface, and two surfaces. A 3D cube with different 2Dgraphics on each of its face has been implemented along with shadow and diffused reflection. .

Keywords*: LPC1769, SPI, LCD, 3D Graphics, GPIO.*

1. Introduction

The main function of the graphics processing engine is to render the 2D or 3D graphics using software coding methods. Due to growth of technology, many graphics applications like gaming has gained lot of popularity. In this lab we are interfacing LPC 1769 module with a 2.4'' 240x320 TFT LCD. The LCD interfaces with the LPC using the Serial Peripheral Interface protocol. The LPC1769 is an ARM Cortex-M3 based microcontroller for embedded applications requiring a high level of integration and low power dissipation. The SPI protocol establishes connection and performs the transmission and reception of data from external devices. The main objective of this lab is to design and implement a power circuit suitable for the CPU module, interface LCD and display a 3D cube letter on each visible surfaces of the cube.

2. Objectives and Technical Challenges

The objectives of the lab include:
1. Interface the LCD display with the LPC1769 CPU module using SPI protocol.
2. To build a power unit for the CPU module using voltage regulator.
3. Develop a program that sends the data buffer to the module, initialize the external LCD and perform. different operation on it which is connected with the module.
4. Design graphic engine logic to display 3D cube on the LCD.

There were certain technical challenges faced while performing the experiment:
1. Connecting the berg strips to the LPC board.
2. Mounting and soldering the LCD display onto the breakout board.
3. Connection between LCD display and CPU module.

When the berg strips were connected to the CPU module, due to the small width of the pins, it was not connected to the CPU module, so the pins were soldered. There were loose connections between the LCD display and the CPU module pins which was resolved using jumper cables.

2.1. Problem Formulation and Design

This section will provide the detailed design. It includes the block diagram and the schematics and pin connection between the components used for this lab assignment. The hardware used for this lab is connected to the wire wrapping board using soldering technique.

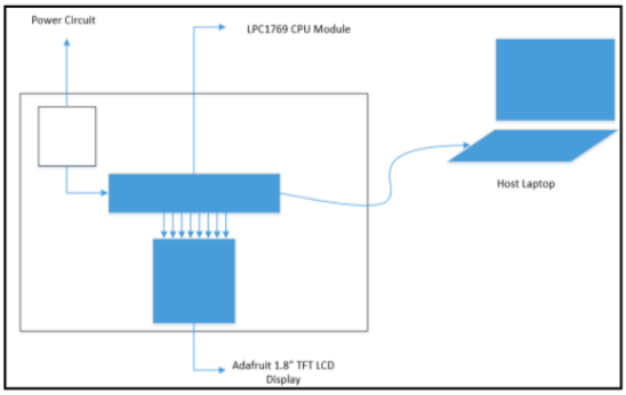

Figure 1. Overview of the Project

The hardware design includes Adafruit 1.8" Color TFT LCD Display and LPCXpresso 1769 CPU module. The critical part of this lab is the communication between the LCD display and the CPU module. The hardware design includes Adafruit 1.8" Color TFT LCD Display and LPCXpresso 1769 CPU module. The critical part of this lab is the communication between the LCD display and the CPU module.

3. Implementation

The overall layout of the board along with the computer is as follows

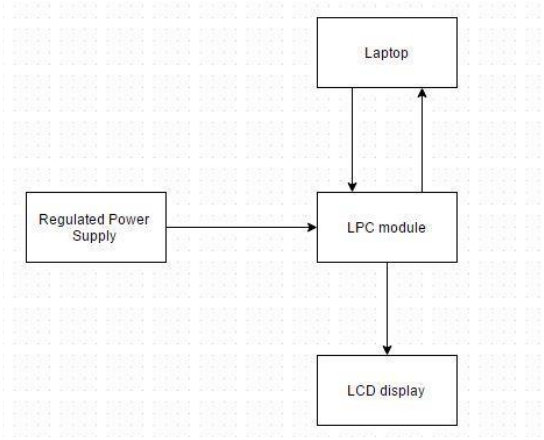

Figure 2. Block Diagram

The implementation of this lab in done in two parts hardware and software. The details of each part is described in this section.

3.1. Hardware Design

The hardware components required to implement this lab is as follows:

No.	Item and Description	Notes
1.	Wire Wrapping board	8.5x11 inches
2.	NXP's LPC1769 module (ARM Cortex M3)	
3.	SPI based color LCD display	Driver ST7735R
4.	Colored wires	For connections
5.	Laptop	With MCU Xpresso IDE
6.	Power Regulator IC 7805	Output 3.3V
7.	Wall mount adaptor	Voltage > 7.5V DC

Table 1. Bill of materials

While making a prototype board the most important part is a power regulation circuit to give 5 V output to power up LPC1769. After that interface of LCD and the CPU Module. The CPU Module (Microcontroller) is operated as the Master. Its MISO, MOSI and SCK pins are used to interface with the SPI LCD Display which works as slave. The following pictures shows the hardware developed for of the project.

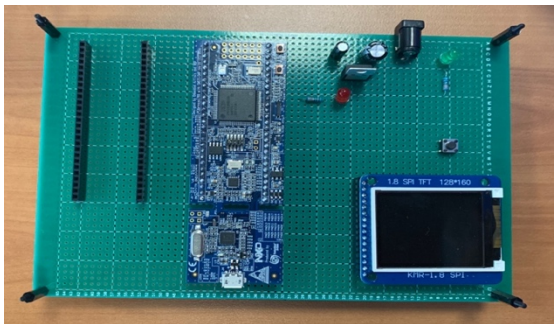

Figure 3. Hardware model of the system

The ST7735R is a single-chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source lines and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI), 8-bit/9-bit/16-bit/18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits. It can perform display data RAM read/write operation with no external operation clock to minimize power consumption. In addition, because of the integrated power supply circuits necessary to drive liquid crystal, it is possible to make a display system with fewer components.

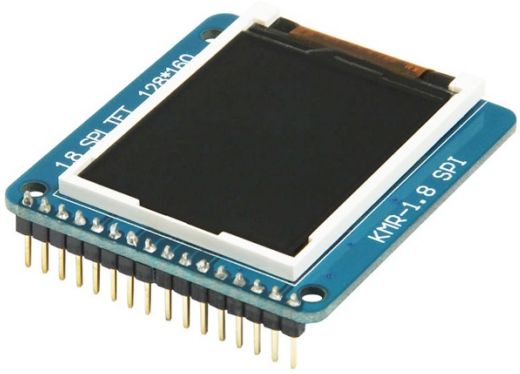

Figure 4. LCD Display Module

3.1.1 Circuit Design

The first thing when making a prototype board is a power regulator circuit. The CPU Module needs 5V to function properly so a voltage regulator is used to reduce any input voltage to 5V. Using a LM 7805 Voltage Regulator IC, capacitor, switch and LEDs a power circuit is made. The circuit diagram for the same is shown.

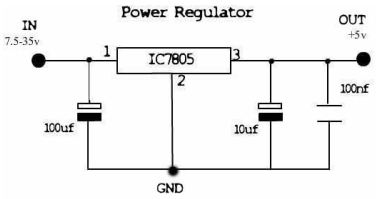

Figure 5. Circuit Diagram for Power Sensor

The following image shows the pin description of LCD Module

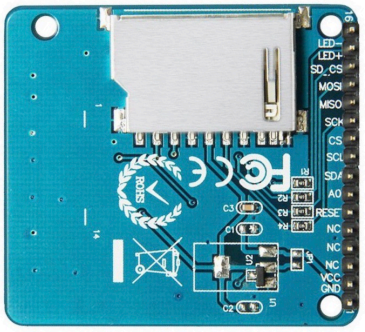

Figure 6. LCD Pin Description

The following table gives the detailed description of interface between the LPC and the LCD.

Pins Connection for LPC and LCD Interface	
LPC Pins	***LCD Pins***
Vcc (J6-28)	LED+ (Pin 2)
GND	LED- (Pin 1)
MISO0 (P0.17) (J6-12)	MISO (Pin 5)
SCK0 (P0.15) (J6-13)	SCK (Pin 6)
MOSI0 (P0.18) (J6-11)	SDA (Pin 9)
SSEL0 (P0.16) (J6-14)	TFT_CS (Pin 7)
GPIO (P0.21) (J2-23)	AO (Pin 10)
GPIO (P0.22) (J2-24)	RESET (Pin 11)
VCC 3.3V	VCC (Pin 15)
GND	GND (Pin 16)

Table 2. Pin connection between LCD and LPC 1769

In the figure 6, the LCD has 10 pins which connect to the CPU Module. The module is connected to the Host Computer via USB cable through which it is powered up. Pin J6-11, J6-13 and J6-14 of the LPC 1769 is used for serial communication with SPI port number 0 for our algorithm. When the CS Pin of the LCD Module is logic 1 the device is deselected, and data output pins are at high impedance. The output of the LCD Display that is the second pin is connected to the MISO (Master In Slave Out) of the CPU module. The SPI instruction uses MOSI and serially writes instructions on the rising edge of the clock. The LED+ pin is connected to the Vcc and the LED- is connected to the GND pin, this lights up the backlight and the display becomes more clear. The Chip Select signal for the TFT Display should be kept low all the time for the duration of the RESET operation to avoid resetting the internal logic state of the device. Instructions vary in length. For some we send only the opcode, for some we send dummy bytes also so that there is some time for the CPU module to recognize the instruction.

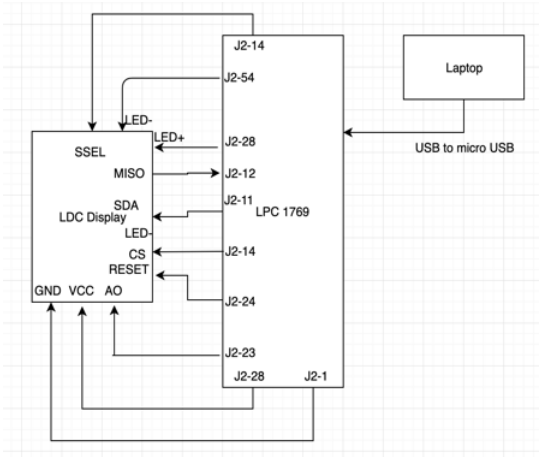

Figure 7. SPI Connection Between LCD and LPC

3.2. Software Design

The implementation of entire system is done in C++ language. This help us to use the open source library provided by the NXP for LPC 1769 Board support package.

A detailed description of the algorithm is provided below to help in better understanding the architecture. Entire source code with the detailed flow chart is provided in the appendix.

3.2.1 Algorithm

The algorithm specifies the basic steps required to establish the software part in the Integrated Development Environment and making the Liquid Crystal Display work as instructed.

1. Make all necessary connections between the Master, Slave and the LCD.
2. Send the data on the MOSI of the LPC, for the device to test.
3.Print the data on the LCD.
4. Select and send the next data/command, and disable the chip select.
5.Send the data which is supposed to be written.
6.Set the chip select.
7. Clear the chip select and display 3D axis on the LCD.
8.Call the function of cube to display on the screen
9. Draw the cube in XYZ plane of LCD.
10.Draw the shadow of the cube.
11.Decorate the surfaces of cube with Initials.
12. Diffuse Reflection of the top surface of the cube with decoration is computed.

3.2.2 Flow Chart

The below flow chart accurately depicts the entire process structure of this lab experiment.

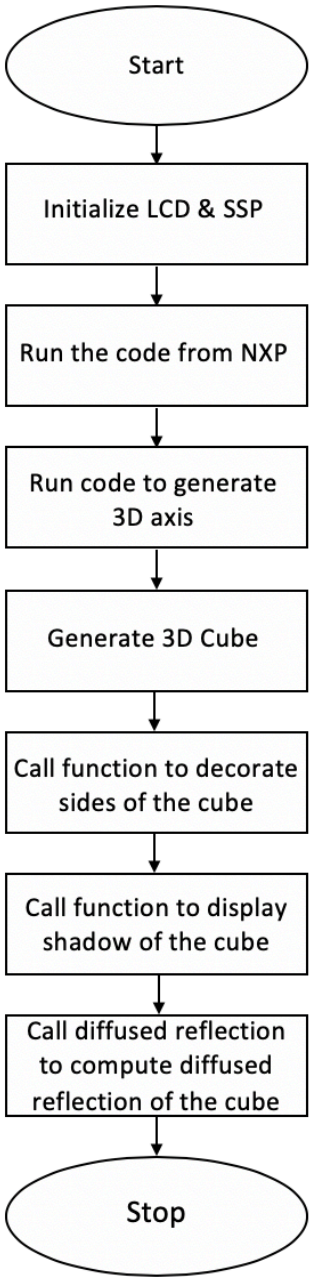

Figure 8. Flowchart

3.2.3 Pseudo Code

The pseudo code for the LCD is implemented and tested. This code will draw the forest of trees at the initial stage. Further a 3D cube with shadow and each side compromising of different shapes are depicted on the LCD. The first step in the procedure is to build the code. Then we should debug the code. Immediately, ”Run“ button is executed. This makes the code to run in a loop. This data is printed onto the terminal.

/ Pseudo Code #include "LPC17xx.h" /* LPC17xx definitions

main()
void spiwrite() // get the input c
void writecommand () // get the input c
void writedata() and void writeword()
// get the input c and write data or word according to it.
void write888() // get the color and repeat parameter to display
void setAddrWindow()
// get four parameters
void drawPixel()
// Input: x, y, color get two coordinates and display pixel according to it.
void drawLine()
//get four coordinates and display line according to it.
void fillrect ()
// get four coordinates and color to display the area for display
void lcddelay()
// delay the display time in ms
void lcd_init () // intialise the lcd.
void axis_generation()
void cube_generation()
void draw_Shadow(double inti_pos, double size, doubleshadow_Xposition,double shadow_Yposition,double shadow_Zposition) // draws the shadow of the cube void
void diffused_reflection()

4. Testing and Verification

This section provides a testing and verification procedure for power circuit module and the data transfer between LCD and CPU module.
The Test procedure is as follows

1.Verify that the CPU is connected and LPC link is established.
2. Verify the LCD is properly connected to LPC.
3. Check to see that LCD is given VCC and GND from LPC

4. Run the code and verify the result on the LCD.
5. The LCD glows when the CPU module receives the power. Import the project LCD_Test into the development tool.
6. The project if having no errors would build and debug successfully with successful link connection to the CPU module.
7.The data would be transferred from main program in IDE to the CPU via USB cable.
8. This will initiate the display of the cube with 3d axis along with a shadow and diffused reflection is presented.
9. After filling the entire screen, a delay is introduced so

that a better view of the 3D Display can be obtained. Thus, interfacing of Liquid Crystal Display with LPC 1769 using Serial Peripheral Interface protocol can be tested and verified successfully.

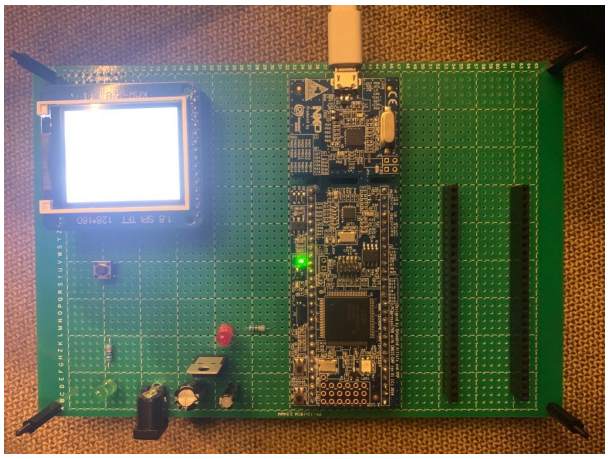

Fig. 9. Prototype board with Working LCD through SPI

The program ran successfully producing the following result.

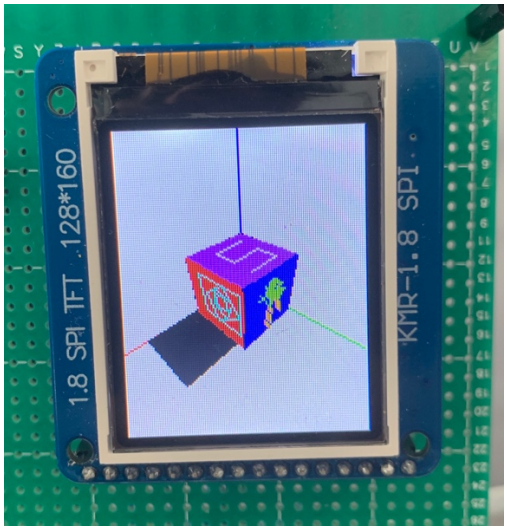

Figure 10. 3D Cube with Trees, Square and Letter S

Therefore, from the above screenshots, we can verify that the interface between the LCD and the LPC module was designed and implemented. The objective of the lab of displaying the three-dimensional images was satisfactorily completed.

5. Conclusion

The graphics engine design was tested successfully by displaying tree and 3D cube on the hardware connecting the LCD module with the LPCXpresso 1769. The work provides an opportunity to study 7805 voltage regulators, SPI LCD displays, LPCXpresso CPU module, LPCXpresso and soldering techniques and most importantly it provides an insight to Graphic Display Drivers and their functionalities.

6. Acknowledgement

We would like to express our special gratitude to Dr. Harry Li for providing valuable information on LPC 1769, MCU Xpresso IDE and LCD. We have put in special effort to complete this project.

7. References

[1] Dr. H. Li, Guidelines for CMPE240 project and report, Computer Engineering Department, San Jose State University, San Jose 95112

[2] Dr. H. Li, CMPE240 Lecture Notes, Computer Engineering Department, San Jose State University, San Jose 95112

[3] NXP LPCXpresso1769 Discussion forums at www.lpcware.com/forum

[4] LPC1769 User manual

[5] http://www.adafruit.com/datasheets/W25Q80BV.pdf

[6] LCD Screen Datasheet – ST7735R http://www.adafruit.com/datasheets/ST7735R_V0.2.pdf

8. Appendix

[A] Source code

```
#include "board.h"
#include "FreeRTOS.h"
#include "task.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ssp.h>

#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 color values

#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0000FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF



int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

int cursor_x = 0, cursor_y = 0;
int rotation = 0;
int textsize  = 1;
int x_diff = 64;
int y_diff = 80;

struct point_coordinate{
     int x;
     int y;
};

int cam_x = 120;
int cam_y = 120;
int cam_z = 120;

int light_x = 22;
int light_y = 22;
int light_z = 60;

#define LOCATION_NUM        0
```

```
uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];
int colstart = 0;
int rowstart = 0;


typedef struct Point3D
{
     float x;
     float y;
     float z;
} Point3D;

typedef struct Point
{
     float x;
     float y;
} Point;

void spiwrite(uint8_t c)

{

 int pnum = 0;

 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 draw_rect(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("LCD Demo Begins!!!\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 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 draw_pixel(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);

}



/**********************************
**********************************
********

** Descriptions:        Draw line
function

**

** parameters:           Starting
point (x0,y0), Ending point(x1,y1)
and color

** Returned value:        None

**

**********************************
**********************************
*******/


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

{

 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) {

  draw_pixel(y0, x0, color);

 }

 else {

  draw_pixel(x0, y0, color);

 }

 err -= dy;

 if (err < 0) {

  y0 += ystep;

  err += dx;

 }

}
```

```
}


struct point_coordinate
world_to_viewer_coord (int x_w, int
y_w, int z_w)
{
      int scrn_x, scrn_y, Dist=100,
x_diff=ST7735_TFTWIDTH/2,
y_diff=ST7735_TFTHEIGHT/2;
      double x_p, y_p, z_p, theta,
phi, rho;
      struct point_coordinate
screen;
      theta =
acos(cam_x/sqrt(pow(cam_x,2)+pow(ca
m_y,2)));
      phi =
acos(cam_z/sqrt(pow(cam_x,2)+pow(ca
m_y,2)+pow(cam_z,2)));
      //theta = 0.785;
      //phi = 0.785;
      rho=
sqrt((pow(cam_x,2))+(pow(cam_y,2))+
(pow(cam_z,2)));
      x_p = (y_w*cos(theta))-
(x_w*sin(theta));
      y_p = (z_w*sin(phi))-
(x_w*cos(theta)*cos(phi))-
(y_w*cos(phi)*sin(theta));
      z_p = rho-
(y_w*sin(phi)*cos(theta))-
(x_w*sin(phi)*cos(theta))-
(z_w*cos(phi));
            scrn_x = x_p*Dist/z_p;
      scrn_y = y_p*Dist/z_p;
      scrn_x = x_diff+scrn_x;
      scrn_y = y_diff-scrn_y;
      screen.x = scrn_x;
      screen.y = scrn_y;
      return screen;
}
```

```
void draw_coordinates ()
{
      struct point_coordinate lcd;
      int x1,y1,x2,y2, x3,y3,x4,y4;
            lcd =
world_to_viewer_coord (0,0,0);
            x1=lcd.x;
            y1=lcd.y;
            lcd =
world_to_viewer_coord (180,0,0);
            x2=lcd.x;
            y2=lcd.y;
            lcd =
world_to_viewer_coord (0,180,0);
            x3=lcd.x;
            y3=lcd.y;
            lcd =
world_to_viewer_coord (0,0,180);
            x4=lcd.x;
            y4=lcd.y;


      draw_line(x1,y1,x2,y2,RED);
      //x axis  red

      draw_line(x1,y1,x3,y3,0x0000F
F00); //y axis  green
            draw_line(x1, y1, x4,
y4,0x000000FF);       //z axis
blue
}

void rotate_point(int *x, int *y,
float angle)
{
      int temp_x = *x , temp_y =
*y;
      angle = angle*(3.14285/180);
      float cos_rotation =
cos(angle);
      float sin_rotation =
sin(angle);
      *x = temp_x*cos_rotation -
temp_y*sin_rotation;
      *y = temp_x*sin_rotation +
temp_y*cos_rotation;
}

void draw_cube(int start_pnt, int
size)
{
      struct point_coordinate lcd;
      int
x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
,x7,y7;
      cam_x = 100;
      cam_y = 100;
      cam_z = 110;

      lcd = world_to_viewer_coord
(start_pnt,start_pnt,(size+start_pn
t));
      x1=lcd.x;
      y1=lcd.y;
      lcd = world_to_viewer_coord
((size+start_pnt),start_pnt,(size+s
tart_pnt));
      x2=lcd.x;
      y2=lcd.y;
      lcd = world_to_viewer_coord
((size+start_pnt),(size+start_pnt),
(size+start_pnt));
      x3=lcd.x;
      y3=lcd.y;
      lcd = world_to_viewer_coord
(start_pnt,(size+start_pnt),(size+s
tart_pnt));
      x4=lcd.x;
      y4=lcd.y;
      lcd = world_to_viewer_coord
((size+start_pnt),start_pnt,start_p
nt);
      x5=lcd.x;
      y5=lcd.y;
```

```
      lcd = world_to_viewer_coord
((size+start_pnt),(size+start_pnt),
start_pnt);
      x6=lcd.x;
      y6=lcd.y;
      lcd = world_to_viewer_coord
(start_pnt,(size+start_pnt),start_p
nt);
      x7=lcd.x;
      y7=lcd.y;
      draw_line(x1, y1, x2,
y2,0x00000000);
      draw_line(x2, y2, x3,
y3,0x00000000);
      draw_line(x3, y3, x4,
y4,0x00000000);
      draw_line(x4, y4, x1,
y1,0x00000000);
      draw_line(x2, y2, x5,
y5,0x00000000);
      draw_line(x5, y5, x6,
y6,0x00000000);
      draw_line(x6, y6, x3,
y3,0x00000000);
      draw_line(x6, y6, x7,
y7,0x00000000);
      draw_line(x7, y7, x4,
y4,0x00000000);
}

void draw_rotated_cube(int start_x,
int start_y ,int start_z, int
size,float angle)
{
      struct point_coordinate lcd;
      int
x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6
,x7,y7,x8,y8;
      int x[8],y[8],z[8];
      double xs[8] = {0}, ys[8] =
{0}, zs[8] = {0};
      cam_x = 120;
      cam_y = 120;
      cam_z = 120;
      x[0] = start_x;
      y[0] = start_y;
      z[0] = start_z;
      x[1] = start_x;
      y[1] = start_y;
      z[1] = size+start_z;
      x[2] = size+start_x;
      y[2] = start_y;
      z[2] = size+start_z;
      x[3] = start_x+size;
      y[3] = start_y;
      z[3] = start_z;
      x[4] = size+start_x;
      y[4] = start_y+size;
      z[4] = start_z;
      x[5] = size+start_x;
      y[5] = size+start_y;
      z[5] = start_z+size;
      x[6] = start_x;
      y[6] = start_y+size;
      z[6] = start_z+size;
      x[7] = start_x;
      y[7] = start_y+size;
      z[7] = start_z;

      rotate_point(&x[0], &y[0],
angle);
      rotate_point(&x[1], &y[1],
angle);
      rotate_point(&x[2], &y[2],
angle);
      rotate_point(&x[3], &y[3],
angle);
      rotate_point(&x[4], &y[4],
angle);
      rotate_point(&x[5], &y[5],
angle);
      rotate_point(&x[6], &y[6],
angle);
      rotate_point(&x[7], &y[7],
angle);
```

```
      lcd = world_to_viewer_coord
(x[0],y[0],z[0]);
      x1=lcd.x;
      y1=lcd.y;
      lcd = world_to_viewer_coord
(x[1],y[1],z[1]);
      x2=lcd.x;
      y2=lcd.y;
      lcd = world_to_viewer_coord
(x[2],y[2],z[2]);
      x3=lcd.x;
      y3=lcd.y;
      lcd = world_to_viewer_coord
(x[3],y[3],z[3]);
      x4=lcd.x;
      y4=lcd.y;
      lcd = world_to_viewer_coord
(x[4],y[4],z[4]);
      x5=lcd.x;
      y5=lcd.y;
      lcd = world_to_viewer_coord
(x[5],y[5],z[5]);
      x6=lcd.x;
      y6=lcd.y;
      lcd = world_to_viewer_coord
(x[6],y[6],z[6]);
      x7=lcd.x;
      y7=lcd.y;
      lcd = world_to_viewer_coord
(x[7],y[7],z[7]);
      x8=lcd.x;
      y8=lcd.y;
      draw_line(x1, y1, x2,
y2,0x00FF0000);
      draw_line(x1, y1, x4,
y4,0x00FF0000);
      draw_line(x1, y1, x8,
y8,0x00FF0000);
      draw_line(x2, y2, x3,
y3,0x00FF0000);
      draw_line(x2, y2, x7,
y7,0x00FF0000);
      draw_line(x6, y6, x3,
y3,0x00FF0000);
      draw_line(x5, y5, x6,
y6,0x00FF0000);
      draw_line(x4, y4, x5,
y5,0x00FF0000);
      draw_line(x5, y5, x8,
y8,0x00FF0000);
      draw_line(x3, y3, x4,
y4,0x00FF0000);
      draw_line(x6, y6, x7,
y7,0x00FF0000);
      draw_line(x8, y8, x7,
y7,0x00FF0000);

      fill_Triangle(x1, y1, x1-10,
y1+5, x4-10, y4+10, BLACK);
      fill_Triangle(x1, y1, x4, y4,
x4-10, y4+10, BLACK);
      draw_shadow(xs, ys, zs, size,
-1000, 0, 1000);

}

void draw_square(int angle)
{
      int
x0,y0,y1,x1,x2,y2,x3,y3,size,intCol
or=0,i=0;
      struct point_coordinate lcd;
      uint32_t color, colorArray
[12]={0x00FF0000,0x0000FFFF,0x00FF0
07F,0x00FF8000,0x0000FF80,0x000000F
F,0x00FFFF00,0x00330066,0x0000FF80,
0x00FF00FF,0x0000FF00,0x000080FF};
      while(i<2)
      {
            i++;
            x0 = 1+ rand() %
(angle/2);
            y0=1+ rand() %
(angle/2);
            size = 30 + rand() %
(angle/4);
            if(intColor>12)
                  intColor =0;
```

```
            color =
colorArray[intColor];
            intColor++;
            x1=size+x0;

            if(x1>angle)
                  x1=angle-1;

            x2=x1;
            x3=x0;
            y1=y0;
            y2=size+y1;
            if(y2>angle)
                  y2=angle-1;

            y3=y2;

            lcd =
world_to_viewer_coord
(angle,x0,y0);
            x0=lcd.x;
            y0=lcd.y;
            lcd =
world_to_viewer_coord
(angle,x1,y1);
            x1=lcd.x;
            y1=lcd.y;
            lcd =
world_to_viewer_coord
(angle,x2,y2);
            x2=lcd.x;
            y2=lcd.y;
            lcd =
world_to_viewer_coord
(angle,x3,y3);
            x3=lcd.x;
            y3=lcd.y;

            draw_line(x0, y0, x1,
y1,color);
            draw_line(x1, y1, x2,
y2,color);
            draw_line(x2, y2, x3,
y3,color);
            draw_line(x3, y3, x0,
y0,color);
            //for rotation
            int it;
            for(it=0;it<3;it++)
            {
                  x0=(x0+(0.4*(x1-
x0)));
                  y0=(y0+(0.4*(y1-
y0)));
                  x1=(x1+(0.4*(x2-
x1)));
                  y1=(y1+(0.4*(y2-
y1)));
                  x2=(x2+(0.4*(x3-
x2)));
                  y2=(y2+(0.4*(y3-
y2)));
                  x3=(x3+(0.4*(x0-
x3)));
                  y3=(y3+(0.4*(y0-
y3)));

                  draw_line(x0, y0,
x1, y1,color);
                  draw_line(x1, y1,
x2, y2,color);
                  draw_line(x2, y2,
x3, y3,color);
                  draw_line(x3, y3,
x0, y0,color);
            }
      }
}

void draw_rotated_xz_square(int
start_x, int start_y ,int start_z,
int len, float angle)
{
      int
x0,y0,y1,x1,x2,y2,x3,y3,size,intCol
or=0,i=0;
```

```
	struct point_coordinate lcd;
	int x,y;
	uint32_t color, colorArray
[12]={0x00FF0000,0x0000FFFF,0x00FF0
07F,0x00FF8000,0x0000FF80,0x000000F
F,0x00FFFF00,0x00330066,0x0000FF80,
0x00FF00FF,0x0000FF00,0x000080FF};
	while(i<1)
	{
		i++;
		x0 = start_x + len;
		y0 = start_z;
		size = 20;
		if(intColor>12)
			intColor =0;

		color =
colorArray[intColor];
		intColor++;
		x1=size+x0;

		if(x1> start_x)
			x1=start_x;

		x2=x1;
		x3=x0;
		y1=y0;
		y2=size+y1;
		if(y2> start_z + len)
			y2= start_z + len-
1;


		y3=y2;

		x = x0;
		y = start_y;
		rotate_point(&x, &y,
angle);
		lcd =
world_to_viewer_coord (x,y,y0);
		x0=lcd.x;
		y0=lcd.y;

		x = x1;
		y = start_y;
		rotate_point(&x, &y,
angle);
		lcd =
world_to_viewer_coord (x, y,y1);
		x1=lcd.x;
		y1=lcd.y;

		x = x2;
		y = start_y;
		rotate_point(&x, &y,
angle);
		lcd =
world_to_viewer_coord(x, y,y2);
		x2=lcd.x;
		y2=lcd.y;

		x = x3;
		y = start_y;
		rotate_point(&x, &y,
angle);
		lcd =
world_to_viewer_coord(x, y, y3);
		x3=lcd.x;
		y3=lcd.y;

		draw_line(x0, y0, x1,
y1,color);
		draw_line(x1, y1, x2,
y2,color);
		draw_line(x2, y2, x3,
y3,color);
		draw_line(x3, y3, x0,
y0,color);
		//for rotation
		int it;
		for(it=0;it<3;it++)
		{
			x0=(x0+(0.4*(x1-
x0)));
			y0=(y0+(0.4*(y1-
y0)));
			x1=(x1+(0.4*(x2-
x1)));
```

```
                y1=(y1+(0.4*(y2-
y1)));
                x2=(x2+(0.4*(x3-
x2)));
                y2=(y2+(0.4*(y3-
y2)));
                x3=(x3+(0.4*(x0-
x3)));
                y3=(y3+(0.4*(y0-
y3)));

                draw_line(x0, y0,
x1, y1,color);
                draw_line(x1, y1,
x2, y2,color);
                draw_line(x2, y2,
x3, y3,color);
                draw_line(x3, y3,
x0, y0,color);
           }
     }
}

void draw_rotated_tree(uint32_t
color,int start_x, int start_y, int
start_z, int size , int
rotate_angle)
{
     int i=0, angle;
     int x,y;
     struct point_coordinate lcd;

     while(i<2)
     {
          int x0, y0, y1,
x1,xp0,xp1,yp0,yp1;
          angle = start_x+size;
          if(i == 0)

     x0=start_y+(size/3);//start_y
          else
               x0 = start_y +
size/3 + size/3;
               y0=start_z;//start_z
               y1=start_z+20;
               x1=x0;

               i++;
               x = angle ; y = x0;
               rotate_point(&x, &y,
rotate_angle);
               lcd =
world_to_viewer_coord (x,y,y0);
               xp0=lcd.x;
               yp0=lcd.y;
               x = angle ; y = x1;
               rotate_point(&x, &y,
rotate_angle);
               lcd =
world_to_viewer_coord (x,y,y1);
               xp1=lcd.x;
               yp1=lcd.y;
               draw_line(xp0, yp0, xp1,
yp1, color);    //level 0 straight
line
               draw_line((xp0+1),
(yp0+1), (xp1+1), (yp1+1),color);
     //level 0 straight line
               draw_line((xp0-1), (yp0-
1), (xp1-1), (yp1-1), color);
     //level 0 straight line

               int it=0;
               for(it=0;it<4;it++){
                    int16_t
x2=(0.6*(x1-x0))+x1;  // length of
level 1 = 0.8 of previous level
                    int16_t y2=y1;
                    x = angle ; y = x1;
                    rotate_point(&x,
&y, rotate_angle);
                    lcd =
world_to_viewer_coord (x,y,y2);
                    int xp2=lcd.x;
                    int yp2=lcd.y;
```

```
                draw_line(xp1, yp1,
xp2, yp2,color);//level 1 straight
line

                //for right rotated
angle 30 degree
                int16_t xr=
((0.134*x1)+(0.866*x2)-
(0.5*y2)+(0.5*y1));
                int16_t
yr=((0.5*x2)-(0.5*x1)+(0.866*y2)-
(0.866*y1)+y1);
                x = angle ; y = xr;
                rotate_point(&x,
&y, rotate_angle);
                lcd =
world_to_viewer_coord (x,y,yr);
                int xpr=lcd.x;
                int ypr=lcd.y;

                //for left rotated
angle 30 degree
                int16_t
xl=((0.134*x1)+(0.866*x2)+(0.5*y2)-
(0.5*y1));
                int16_t
yl=((0.5*x1)-
(0.5*x2)+(0.134*y2)+(0.866*y1));
                x = angle ; y = xl;
                rotate_point(&x,
&y, rotate_angle);
                lcd =
world_to_viewer_coord (x,y,yl);
                int xpl=lcd.x;
                int ypl=lcd.y;

                draw_line(xp1, yp1,
xpr, ypr,color);
                draw_line(xp1, yp1,
xpl, ypl,color);

                x0=x1;
                x1=x2;
          }
     }
}
void draw_tree(uint32_t color,int
start_pnt, int size)
{
     int i=0, angle;
     struct point_coordinate lcd;
     int
tree_branch[3][3]={{start_pnt,start
_pnt+20,0.5*size},{start_pnt+10,sta
rt_pnt+20,0.3*size},{start_pnt+15,s
tart_pnt+37,0.8*size}};
     while(i<2)
     {
          int x0, y0, y1,
x1,xp0,xp1,yp0,yp1;
          angle = start_pnt+size;
          x0=tree_branch[i][0];
          x1=tree_branch[i][1];
          y0=tree_branch[i][2];
          y1=y0;
          i++;
          lcd =
world_to_viewer_coord
(y0,angle,x0);
          xp0=lcd.x;
          yp0=lcd.y;
          lcd =
world_to_viewer_coord
(y1,angle,x1);
          xp1=lcd.x;
          yp1=lcd.y;
          draw_line(xp0, yp0, xp1,
yp1,0x00FF8000);//level 0 straight
line
          draw_line((xp0+1),
(yp0+1), (xp1+1),
(yp1+1),0x00FF8000);  //level 0
straight line
          draw_line((xp0-1), (yp0-
1), (xp1-1), (yp1-1),0x00FF8000);
     //level 0 straight line

          int it=0;
```

```
          for(it=0;it<4;it++){
                int16_t
x2=(0.6*(x1-x0))+x1;  // length of
level 1 = 0.8 of previous level
                int16_t y2=y1;
                lcd =
world_to_viewer_coord
(y2,angle,x2);
                int xp2=lcd.x;
                int yp2=lcd.y;
                draw_line(xp1, yp1,
xp2, yp2,color);//level 1 straight
line


                //for right rotated
angle 30 degree
                int16_t xr=
((0.134*x1)+(0.866*x2)-
(0.5*y2)+(0.5*y1));
                int16_t
yr=((0.5*x2)-(0.5*x1)+(0.866*y2)-
(0.866*y1)+y1);
                lcd =
world_to_viewer_coord
(yr,angle,xr);
                int xpr=lcd.x;
                int ypr=lcd.y;

                //for left rotated
angle 30 degree
                int16_t
xl=((0.134*x1)+(0.866*x2)+(0.5*y2)-
(0.5*y1));
                int16_t
yl=((0.5*x1)-
(0.5*x2)+(0.134*y2)+(0.866*y1));
                lcd =
world_to_viewer_coord
(yl,angle,xl);
                int xpl=lcd.x;
                int ypl=lcd.y;

                draw_line(xp1, yp1,
xpr, ypr,color);
                draw_line(xp1, yp1,
xpl, ypl,color);

                //for branches on
right rotated branch angle 30
degree
                int16_t xrLen =
sqrt(pow((xr-x1),2)+pow((yr-y1),2))
;    //length of right branch
                int16_t xrImag=
(0.8*xrLen)+xr; //imaginary
vertical line x coordinate, y= yr
                int16_t xr1 =
((0.134*xr)+(0.866*xrImag)-
(0.5*yr)+(0.5*yr));
                int16_t yr1 =
((0.5*xrImag)-(0.5*xr)+(0.866*yr)-
(0.866*yr)+yr);
                lcd =
world_to_viewer_coord
(yr1,angle,xr1);
                int xpr1=lcd.x;
                int ypr1=lcd.y;

                //for right branch
                int16_t
xrr,xrl,yrr,yrl;
                xrr =
((0.134*xr)+(0.866*xr1)-
(0.5*yr1)+(0.5*yr));
                yrr = ((0.5*xr1)-
(0.5*xr)+(0.866*yr1)-
(0.866*yr)+yr);
                lcd =
world_to_viewer_coord
(yrr,angle,xrr);
                int xprr=lcd.x;
                int yprr=lcd.y;

                //for left branch
                xrl =
((0.134*xr)+(0.866*xr1)+(0.5*yr1)-
(0.5*yr));
```

```
                yrl = ((0.5*xr)-
(0.5*xr1)+(0.134*yr)+(0.866*yr1));
                lcd =
world_to_viewer_coord
(yrl,angle,xrl);
                int xprl=lcd.x;
                int yprl=lcd.y;
                //for branches on
left rotated branch angle 30 degree
                int16_t xlImag=
(0.8*xrLen)+xl; //imaginary
vertical line x coordinate, y= yr
                int16_t xl1 =
((0.134*xl)+(0.866*xlImag)+(0.5*yl)
-(0.5*yl));
                int16_t yl1 =
((0.5*xl)-
(0.5*xlImag)+(0.134*yl)+(0.866*yl))
;
                lcd =
world_to_viewer_coord
(yl1,angle,xl1);
                int xpl1=lcd.x;
                int ypl1=lcd.y;
                //for right branch
                int16_t
xlr,xll,ylr,yll;
                xlr =
((0.134*xl)+(0.866*xl1)-
(0.5*yl1)+(0.5*yl));
                ylr = ((0.5*xl1)-
(0.5*xl)+(0.866*yl1)-
(0.866*yl)+yl);
                lcd =
world_to_viewer_coord
(ylr,angle,xlr);
                int xplr=lcd.x;
                int yplr=lcd.y;
                //for left branch
                xll =
((0.134*xl)+(0.866*xl1)+(0.5*yl1)-
(0.5*yl));
                yll = ((0.5*xl)-
(0.5*xl1)+(0.134*yl)+(0.866*yl1));
                lcd =
world_to_viewer_coord
(yll,angle,xll);
                int xpll=lcd.x;
                int ypll=lcd.y;
                draw_line(xpr, ypr,
xpr1, ypr1,color);
                draw_line(xpr, ypr,
xprr, yprr,color);
                draw_line(xpr, ypr,
xprl, yprl,color);
                draw_line(xpl, ypl,
xpl1, ypl1,color);
                draw_line(xpl, ypl,
xplr, yplr,color);
                draw_line(xpl, ypl,
xpll, ypll,color);

                x0=x1;
                x1=x2;
            }
        }
}

void fill_rotated_cube(int start_x,
int start_y, int start_z, int size
, float angle)
{
    struct point_coordinate s1;
    int xsize = start_x + size,
ysize = start_y + size, zsize =
start_z + size;
    int i,j;
    int x,y;

    for(i = start_x; i < xsize;
i++)
    {
        for(j = start_y; j <
ysize; j++)
        {
            x = i;
            y = j;
```

```
                rotate_point(&x,
&y, angle);

     s1=world_to_viewer_coord(x,y,
zsize);    //top fill green

     draw_pixel(s1.x,s1.y, RED);

          }
     }

     for(i = start_y; i < ysize;
i++)
     {
          for(j = start_z; j <
zsize; j++)
          {
               x = xsize; y = i;
               rotate_point(&x,
&y, angle);

     s1=world_to_viewer_coord(x,
y, j);     // left fill pink

     draw_pixel(s1.x,s1.y, BLUE);
          }
     }

     for(i = start_x; i < xsize;
i++)
     {
          for(j = start_z; j <
zsize; j++)
          {
               x = i;
               y = start_y;
               rotate_point(&x,
&y, angle);

     s1=world_to_viewer_coord(x,
y, j);     // right fill yellow

     draw_pixel(s1.x,s1.y,0x00FFFF
00);
          }
     }
}

void draw_HorizontalLine(int16_t x,
int16_t y, int16_t width, uint32_t
color)
{
     draw_line(x, y, x+width-1, y,
color);
}

void fill_Triangle(int16_t x0,
int16_t y0,int16_t x1, int16_t y1,
int16_t x2, int16_t y2, uint32_t
color) {
     int16_t x, y, j, l;
     if (y0 > y1) {
          swap(y0, y1);
          swap(x0, x1);
     }
     if (y1 > y2) {
          swap(y2, y1);
          swap(x2, x1);
     }
     if (y0 > y1) {
          swap(y0, y1);
          swap(x0, x1);
     }
     if(y0 == y2) {
          x = y = x0;
          if(x1 < x) x = x1;
          else if(x1 > y) y = x1;

          if(x2 < x) x = x2;
          else if(x2 > y) y = x2;
          draw_HorizontalLine(x,
y0, y-x+1, color);
          return;
     }

     int16_t dx01 = x1 - x0, dy01
= y1 - y0, dx02 = x2 - x0, dy02 =
```

```
y2 - y0, dx12 = x2 - x1, dy12 = y2
- y1;
      int32_t sa = 0, sb = 0;

      if(y1 == y2) l = y1;
      else l = y1-1;

      for(j=y0; j<=l; j++) {
            x = x0 + sa / dy01;
            y = x0 + sb / dy02;
            sa += dx01;
            sb += dx02;
            if(x > y) swap(x,y);
            draw_HorizontalLine(x,
j, y-x+1, color);
      }
      sa = dx12 * (j - y1);
      sb = dx02 * (j - y0);
      for(; j<=y2; j++) {
            x = x1 + sa / dy12;
            y = x0 + sb / dy02;
            sa += dx12;
            sb += dx02;
            if(x > y) swap(x,y);
            draw_HorizontalLine(x,
j, y-x+1, color);
      }
}

void draw_shadow(double x[], double
y[], double z[], int size, double
xShad, double yShad, double zShad)
{
      int xs[8]={0}, ys[8]={0},
zs[8]={0};
      struct point_coordinate
s5,s6,s7,s8;
      int i;
      for(i=4; i<8; i++){
            xs[i]=x[i]-
((z[i]/(zShad-z[i]))*(xShad-x[i]));
            ys[i]=y[i]-
((z[i]/(zShad-z[i]))*(yShad-y[i]));
            zs[i]=z[i]-
((z[i]/(zShad-z[i]))*(zShad-z[i]));
      }
      s5 = world_to_viewer_coord
(xs[4],ys[4],zs[4]);
      s6 = world_to_viewer_coord
(xs[5],ys[5],zs[5]);
      s7 = world_to_viewer_coord
(xs[6],ys[6],zs[6]);
      s8 = world_to_viewer_coord
(xs[7],ys[7],zs[7]);

      draw_line(s5.x, s5.y, s6.x,
s6.y, BLACK);
      draw_line(s6.x, s6.y, s7.x,
s7.y, BLACK);
      draw_line(s7.x, s7.y, s8.x,
s8.y, BLACK);
      draw_line(s8.x, s8.y, s5.x,
s5.y, BLACK);

      fill_Triangle(s5.x, s5.y,
s6.x, s6.y, s7.x, s7.y,BLACK);
      fill_Triangle(s5.x, s5.y,
s7.x, s7.y, s8.x, s8.y,BLACK);
}

int calculate_intensity(int16_t
xPs, int16_t yPs, int16_t zPs,
int16_t xPi, int16_t yPi,
      int16_t zPi, int16_t k) {
      double cosVal;
      double r = sqrt(
      pow((zPs - zPi), 2) +
pow((yPs - yPi), 2) + pow((xPs -
xPi), 2));
      double rcos = sqrt(pow((zPs -
zPi), 2));
      cosVal = rcos / r;
      return (255 * k * cosVal) /
pow(r, 2);
}
```

```
void diffused_reflection(int size)
{
      struct point_coordinate lcd;
      struct point_coordinate tmp;
      int x1, y1, x2, y2, x3, y3,
x4, y4, x5, y5, x6, y6, x7, y7, x8,
y8;
      lcd =
world_to_viewer_coord(0, 0, 0);
      x1 = lcd.x;
      y1 = lcd.y;
      lcd =
world_to_viewer_coord(size, 0, 0);
      x2 = lcd.x;
      y2 = lcd.y;
      lcd =
world_to_viewer_coord(0, size, 0);
      x3 = lcd.x;
      y3 = lcd.y;
      lcd =
world_to_viewer_coord(0, 0, size);
      x4 = lcd.x;
      y4 = lcd.y;
      lcd =
world_to_viewer_coord(size, 0,
size);
      x5 = lcd.x;
      y5 = lcd.y;
      lcd =
world_to_viewer_coord(size, size,
0);
      x6 = lcd.x;
      y6 = lcd.y;
      lcd =
world_to_viewer_coord(size, size,
size);
      x7 = lcd.x;
      y7 = lcd.y;
      lcd =
world_to_viewer_coord(0, size,
size);
      x8 = lcd.x;
      y8 = lcd.y;
      for (int i = 0; i <= size;
i++) {
            for (int j = 0; j <=
size; j++)
            {
                  tmp =
world_to_viewer_coord(i, j, size);
                  int kR =
calculate_intensity(light_x,
light_y, light_z, i, j, size, 255);

                  if (kR + 170 > 255)
                        kR = 255;
                  else
                        kR += 170;

                  uint32_t color =
PURPLE;
                  draw_pixel(tmp.x,
tmp.y, color);
            }
      }
    for (int i = 0; i <= size; i++)
{
      for (int j = 0; j <= size;
j++) {
      tmp = world_to_viewer_coord
(size,i,j);
      int kR =
calculate_intensity(light_x,
light_y, light_z, size, i, j, 255);
      if (kR + 170 > 255) {
          kR = 255;
          } else {
            kR += 170;
          }
      uint32_t color = RED;

      draw_pixel(tmp.x,tmp.y,color)
;
      }
    }
    for (int i = 0; i <= size; i++)
{
```

```
            for (int j = 0; j <=
size; j++) {
            tmp =
world_to_viewer_coord (i,size,j);
            int kR =
calculate_intensity(light_x,
light_y, light_z, size, i, j, 255);
            if (kR + 170 > 255) {
                kR = 255;
                } else {
                  kR += 170;
                }
            uint32_t color = BLUE;

      draw_pixel(tmp.x,tmp.y,color)
;
            }
          }

}

void draw_rotated_S(int start_x,
int start_y, int start_z, int size,
int angle)
{
      int xs[6], ys[6], z = start_z
+ size;
      struct point_coordinate p0,
p1, p2, p3, p4, p5;

      xs[0] = start_x + 10;
      ys[0] = start_y + 10;

      xs[1] = start_x + size - size
/ 3;
      ys[1] = ys[0];

      xs[2] = xs[1];
      ys[2] = ys[1] + size/3;

      xs[3] = xs[0];
      ys[3] = ys[2];

      xs[4] = xs[0];
      ys[4] = ys[3] + size/3;

      xs[5] = xs[1];
      ys[5] = ys[4];

      rotate_point(&xs[0], &ys[0],
angle);
      rotate_point(&xs[1], &ys[1],
angle);
      rotate_point(&xs[2], &ys[2],
angle);
      rotate_point(&xs[3], &ys[3],
angle);
      rotate_point(&xs[4], &ys[4],
angle);
      rotate_point(&xs[5], &ys[5],
angle);

      p0 =
world_to_viewer_coord(xs[0], ys[0],
z);
      p1 =
world_to_viewer_coord(xs[1], ys[1],
z);
      p2 =
world_to_viewer_coord(xs[2], ys[2],
z);
      p3 =
world_to_viewer_coord(xs[3], ys[3],
z);
      p4 =
world_to_viewer_coord(xs[4], ys[4],
z);
      p5 =
world_to_viewer_coord(xs[5], ys[5],
z);

      draw_line(p0.x, p0.y, p1.x,
p1.y, WHITE);
      draw_line(p2.x, p2.y, p1.x,
p1.y, WHITE);
      draw_line(p2.x, p2.y, p3.x,
p3.y, WHITE);
```

```
	draw_line(p4.x, p4.y, p3.x,
p3.y, WHITE);
	draw_line(p4.x, p4.y, p5.x,
p5.y, WHITE);
}

void draw_S(int start_x, int
start_y, int start_z, int size)
{
	int xs[6], ys[6], z = start_z
+ size;
	int xss[6], yss[6];
	int x, y;
	struct point_coordinate p0,
p1, p2, p3, p4, p5;

	xs[0] = start_x + 10;
	ys[0] = start_y + 10;

	xs[1] = start_x + size - size
/ 3;
	ys[1] = ys[0];

	xs[2] = xs[1];
	ys[2] = ys[1] + size/3;

	xs[3] = xs[0];
	ys[3] = ys[2];

	xs[4] = xs[0];
	ys[4] = ys[3] + size/3;

	xs[5] = xs[1];
	ys[5] = ys[4];

	p0 =
world_to_viewer_coord(xs[0], ys[0],
z);
	p1 =
world_to_viewer_coord(xs[1], ys[1],
z);
	p2 =
world_to_viewer_coord(xs[2], ys[2],
z);
	p3 =
world_to_viewer_coord(xs[3], ys[3],
z);
	p4 =
world_to_viewer_coord(xs[4], ys[4],
z);
	p5 =
world_to_viewer_coord(xs[5], ys[5],
z);

	draw_line(p0.x, p0.y, p1.x,
p1.y, WHITE);
	draw_line(p2.x, p2.y, p1.x,
p1.y, WHITE);
	draw_line(p2.x, p2.y, p3.x,
p3.y, WHITE);
	draw_line(p4.x, p4.y, p3.x,
p3.y, WHITE);
	draw_line(p4.x, p4.y, p5.x,
p5.y, WHITE);
}


void funct_main (void)
{

	uint32_t pnum = PORT_NUM;
	 int size = 50, start_pnt =
0;
	double x[8] =
{start_pnt,(start_pnt+size),(start_
pnt+size),start_pnt,start_pnt,(star
t_pnt+size),(start_pnt+size),start_
pnt};
	 double y[8] = {start_pnt,
start_pnt, start_pnt+size,
start_pnt+size, start_pnt,
start_pnt, (start_pnt+size),
(start_pnt+size) };
	 double z[8] = {start_pnt,
start_pnt, start_pnt, start_pnt,
(start_pnt+size), (start_pnt+size),
(start_pnt+size),
(start_pnt+size)};
```

```
	pnum = 0 ;

	if ( pnum == 0)

	SSP0Init();

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

	for (int i = 0; i <
SSP_BUFSIZE; i++ )
	  {
	    src_addr[i] =
(uint8_t)i;
	    dest_addr[i] = 0;
	  }

	  //To initialize LCD
	    lcd_init();
	    fillrect(0, 0,
ST7735_TFTWIDTH, ST7735_TFTHEIGHT,
WHITE);
	    draw_coordinates();
	    draw_shadow(x, y, z,
size, -1000,0,1000);

      draw_cube(start_pnt,size);

      diffused_reflection(size);
	    draw_square(size +
start_pnt);

      draw_tree(0x0066CC00,start_pn
t,size);
	    draw_S(0 ,0 ,0, size);

	    //return 0;
	    while (1) {
	    }
}

/* Sets up system hardware */
static void prvSetupHardware(void)
{
	SystemCoreClockUpdate();
	Board_Init();

	/* Initial LED0 state is off
*/
	Board_LED_Set(0, false);
}


int main(void)
{
	prvSetupHardware();

	/* LED1 toggle thread */
	xTaskCreate(funct_main,
(signed char *) "funct_main",

	configMINIMAL_STACK_SIZE,
NULL, (tskIDLE_PRIORITY + 1UL),
			(xTaskHandle
*) NULL);

	/* Start the scheduler */
	vTaskStartScheduler();

	/* Should never arrive here
*/
	return 1;
}
```