3D Graphics Engine Implementing Diffuse Reflection using LPC1769

Neel Parag Patel
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: neel.p.patel@sjsu.edu

Abstract

This paper describes the interfacing, design, development and implementation of a 3D graphics using LPC1769 module and TFT color LCD Display. The report illustrates detailed design and development stages of the 2D and 3D graphic engine in both software and hardware implementation. The main goal of this lab was to generate three solid cubes with decorations on three sides. The decorations are implemented as a screensaver of a set of randomized rotating squares, a patch of forest which were produced by randomized trees and an alphabet on the top side. The main challenge in this project was understanding interfacing of the LCD with the LPC, generating randomized pattern for the trees, converting values from world to viewer coordinate systems with various transformation matrices and implementing diffused reflection. This was overcome by understanding the datasheets, learning how transformations work and optimizing their implementation in the code.

Keywords: - LPC1769, 3D-Graphics Engine, Shading Model, world to viewer transformation, diffuse reflection.

1. Introduction

The objective of this lab is to understand the concept of 3D graphic engine and the interfacing of the LCD and LPC1769 Xpresso module. In this lab, a 1.8 inch TFT LCD module with resolution of 160*128 with SPI digital interface was used. The LPC module was a master and was responsible for generating the code and driving the LCD module. A standalone power supply circuit was also implemented on the board to supply a constant 3.3V power to the LPC. The LPC in turn drives the LCD module with a SPI interface.

The code produced three cubes with decorated patterns on three surfaces. 3 cubes were made each of different size and orientation angle. The three visible sides of the cubes were colored and decorated. Shadows were generated for all of them and diffuse reflation was implemented on the top of one of the cubs. One side had randomized square patterns the other had a tree forest while the top side had the alphabet initial 'H' displayed.

The report is divided into multiple sections. Section 2 describes the methodology which focuses on the design phase of the system. Section 3 includes the implementation which discusses both the hardware interfacing as well as the software code for generating the desired set of patterns.

LCD was interfaced with the microcontroller using MISO, MOSI and CS pins. One of the main challenges was to make physical connections from the LCD to the Board. Soldering and De-soldering the pins on the wire wrapping board was difficult. Also, the shading model and diffuse reflection algorithms were difficult to implement in code. With help of datasheets and the knowledge on 3D graphics engine we were able to overcome these problems.

2. Methodology

This section deals with the main objective of this project, the hardware and software goals that were carried out in development and the technical challenges that were faced during the implementation. The design, system layout, implementation and software details which were used to help generate tree-forest pattern and rotating squares are also discussed. The steps for designing a power supply to power up the prototype wire-wrapping board is also discussed. The 3D to 2D matrix conversion along with linear decoration, shading and diffuse reflection are explained in detail.

C/C++ programming language was used for implementing the code and LPCxpresso/MCUxpresso IDE was used for interfacing the LPC to our system.

2.1. Objectives and Technical Challenges

This section deals with interfacing the TFT LCD display module to the LCP1769. Also, design steps for for a power supply circuit which provide require power for these modules has been discussed. Studying graphic engine, understanding the SPI interface between modules on the board also was a key learning from this project.

The project can be divided into many small stages which can be clubbed together to formulate the final system. The various steps can be depicted as follows:

1. Setup a wire wrapping board to accommodate an two LPC modules along with an LCD display and a power circuit. GPIO testing circuit was also designed for testing purposes.
2. One LPC module was interfaced with the color LCD.
3. Generate a stable input current and voltage with the help of the power circuit to drive the LCD with the help of the LPC module.
4. Implement the code for transforming a cube in 3D coordinated into 2D coordinates.

5. Generating 3 different cubes, including a shifted and a tilted/rotated cube along with colors on the cube's surfaces.
6. Generating randomized squares, trees and user's initials on the surfaces.
7. Generating shadows of each cube onto the x-y plane.
8. Implementing diffused reflection for one of the cubes.
9. The implementation is done using C/C++ and driving the LCD display successfully from the LPC.

There were a few challenges faced duing the implementation phase. They are listed below.

1. Interfacing and powering the LCD from the LPC module.
2. Soldering the LPC and LCD pins accurately.
3. Setting up development environment on Ubuntu and Windows system for MCUXpresso.
4. Generating direction for rotation logic for randomized square pattern.
5. Generating an algorithm for randomizing trees.
6. Conversion from 3D world to viewer coordinate system.
7. Implementing shadows using vector mathematics.
8. Implementing diffuse reflection on one of the cube's surface.

2.2. Problem Formulation and Design

This project is divided to two main sections: hardware design and software implementation. Hardware design contains of implementing hardware components which are LCD display device, power supply circuit, LPC1769 on the prototype wire wrapping board. LPC1769 and LCD display are connected via SPI (Serial Peripheral Interface). LPC1769 is connected directly to a laptop with Micro USB port. Power supply circuit was also designed to generate power for LCD and LPC modules, so they can operate as a standalone board.

The software implementation consists of a C/C++ programming for 2D and 3D graphic engine to generate screensaver. A randomized rotating square pattern based on user input and a tree-forest pattern using randomized locations was implemented. Generated 3 solid decorated cube with their shadows using vector mathematics and implementing diffuse reflection. SPI interfacing was used for communication between the LPC which was the master and the LCD module which was its slave.

3. Implementation

This section discussed the hardware and software implementation of the project. The power circuit, testing circuit, interfacing details and the process to dump the C/C++ code onto the LPC module using MCUXpresso development environment is also discussed. Finally, the algorithm to drive the LCD display with the randomized patterns is explained. LPC modules were tested by putting simple on-board LED by running the blinking code.

This Section divided into two sections:

3.1. Hardware Design

System Block Diagram

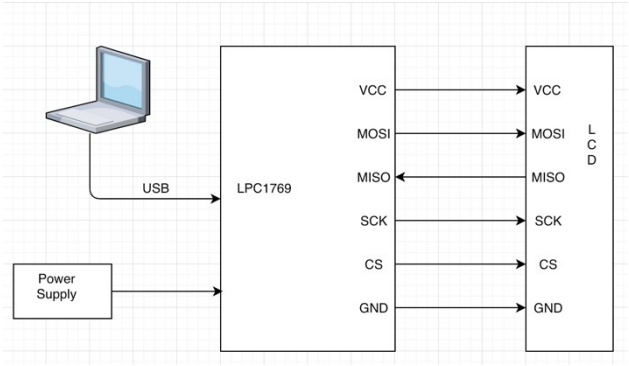

Figure 1 System Block Diagram

Hardware design involves designing a power regulation circuit to convert 7.5V from the wall adapter to 5V that is accepted by the LPC. Next the SPI pins of the CPU module were connected to the respective pins of the LCD. 26-gauge wire was used to make the connections between pins.

3.1.1. Circuit Design

The power regulation circuit was needed for this lab and it was built on the wire-wrapping board using capacitors and a LM7805. This provides a constant 5V output which is needed for our LPC module.

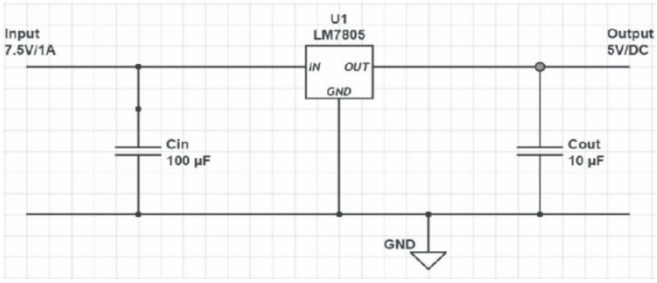

Figure 2 Power Circuit

Bill of material (list of components).

Description	Quantity
Wire Wrapping Board	1
LCP1769Xpresso module	1
TFT LCD Display (ST7735)	1
OSEPP Female header	10 pieces
Wires (26-gauge)	4

LM7805 Regulator	1
Wire wrapping tool kit	1
100 µF Capacitor	1
10 µF Capacitor	1
Power Supply adaptor (7.5V, 1A)	1
LED	1
Stand for prototype board	4
Soldering Kit	1

Figure 3: Bill of materials

The next step was to connect the LCD to the Board. The connection table is given below.

LCD Module	LPC1769 Module	Description
LITE	J2-28 (or power supply output)	LITE Should be connected to power on the backlight in LCD(PP)
MISO	J2-12 (MISO0)	Master In Slave Out
SCK	J2-13 (TXD1, SCK0)	This is the SPI clock input pin. Serial Clock.
MOSI	J2-11 (MOSI0)	Master Out Slave In
TFT_CS	J2-14 (RDX1/SSEL0)	Chip Select, Slave select
CARD_CS	-	Not connected
D/C	J2-21 (GPIO)	GPIO (General Purpose Input/Output)
RESET	J2-22 (GPIO)	GPIO (General

Figure: 4 Connection Table

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), 8/9/16/18-bit parallel interface. It has an on-chip RAM of 132x162x18 bits. It can store the display data.

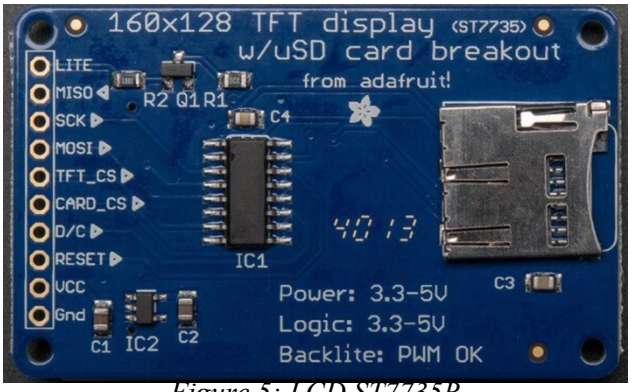

Figure 5: LCD ST7735R

3.2. Software Design

Software implementation has been done on MCUXpresso with C programming language, Using C language provided required open source library. The flow charts and pseudo code for every single implementation is discussed here. The entire code implementation is provided in the appendix.

3.2.1 Flowchart for Squares:

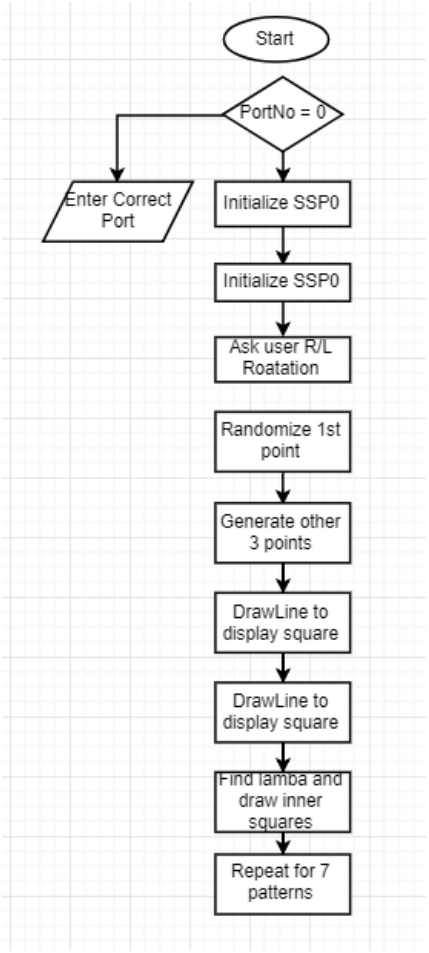

Figure 6: Flow Diagram for squares

3.2.2 Flowchart for Trees:

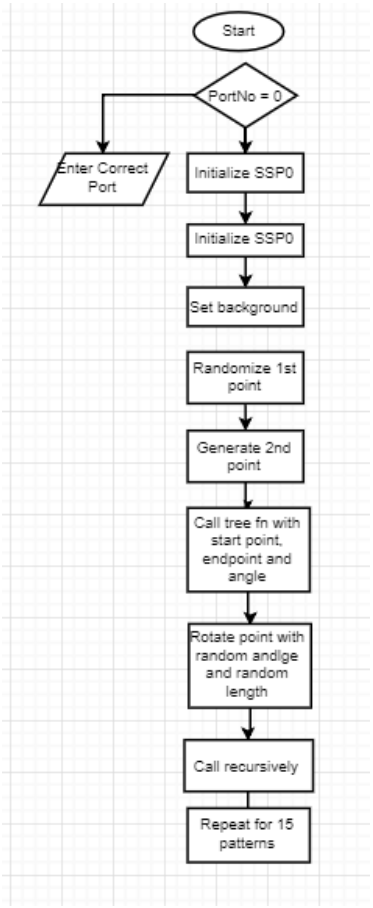

Figure 7: Flow Diagram for trees

3.2.1 Vector Mathematics

There were several vector concepts that were used in this lab. All the formulas that are used are given in this section.

1. **2D- Rotation Matrix**
   The 2D rotation matrix is given as follows:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Figure 8: 2D Rotation Matrix

2. **3D- Rotation Matrix**
   The 3D rotation matrices are used for rotating 3D objects along respective axes. This gives new rotated 3D co-ordinates.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figure 8: 3D Rotation Matrix for x-axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\alpha & 0 & sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -sin\alpha & 0 & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figure 9: 3D Rotation Matrix for y-axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figure 9: 3D Rotation Matrix for z-axis

3. **World to Viewer Transform**
   The 3D world coordinates needed to be translated to Viewer Co-ordinates, so that we can decided where to view the object from.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -sin\Theta & cos\Theta & 0 & 0 \\ -cos\phi sin\Theta & -cos\phi sin\Theta & sin\phi & 0 \\ -sin\phi cos\Theta & -sin\phi cos\Theta & -cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figure 10: World – Viewer Transformation

4. **Perspective Projection**
   These co-ordinates are the 2D co-ordinates that are going to be plotted on the LCD display. The can be given as :

$$x'' = \left(\frac{D}{z'}\right) x'$$

$$y'' = \left(\frac{D}{z'}\right) y'$$

Figure 11: Perspective Projection

5. **Vector Equation for Shadow Co-ordinates** This equation is used to find the intersection point on the plane so that the shadow can be plotted. Here, Ps is the source vector, Pi is the

cube vertex vector and P is the vector that is or the intersection point on the plane.

$$\vec{P} = \vec{Ps} + \lambda\left(\vec{Ps} - \vec{Pi}\right)$$

Figure 12: Perspective Projection

The world co-ordinate system can be pictured as show in the picture below. As shown E is the virtual cam location and this is the viewing position, our Ps vector is based on this point.

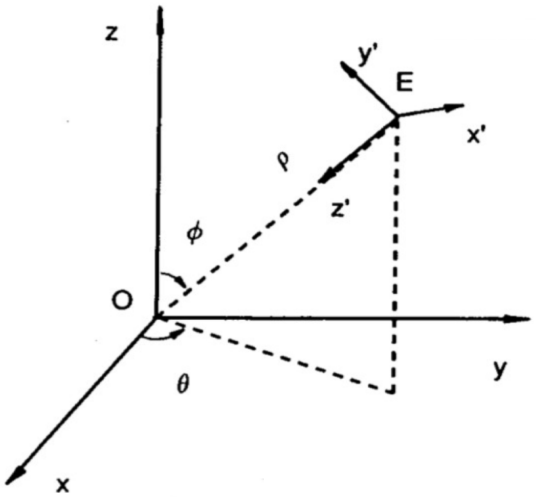

Figure 11: 3D-Coordinate system

3.2.2 Algorithm

1. First SSP has to be initialized.
2. Active low chip select is sent to make the connection from LPC to LCD.
3. Fill the LCD display with white color by sending data(fillrect()) on the MOSI line.
4. SSP clock(SSP_CLK) is selected as pclk/8 by setting the 10th and 11th bits in PCLKSEL1 register.
5. P0.6 to P0.9 pins are set to be functional as SSP1.
6. P0.16 is defined as GPIO and output to select the slave.
7. Pre-scale CLK Value by 2 and initialize the LCD Module.
8. Define and initialize camera and light source location.
9. Draw the world Coordinate System.
10. Get and plot the viewer coordinate system points after translation of world coordinates to viewer coordinate system and prospective projection.
11. Draw three 3D solid cubes and produce their shadows by using the 3D vector and plane intersection concept.
12. Draw screen saver, trees and initial on each cube on the sides.
13. Find the intensity of each pixel on the side of the square and draw the pixel with that intensity.

3.2.3 Pseudocode

On the basis of the algorithm, below are a few code snippets to understand the pseudo code.

1. SSP0Init()

Select Port0 and initialize using FIODIR, FIOCLR and FIOSET

2. fillrect(0,0,ST7735_TFTWIDTH,ST7735_TFTHEIGHT, WHITE)

3. //Function for SPI write
   void spiwrite(uint8_t c)

4. //Function to draw pixel on the screen
   void drawPixel(int16_t x, int16_t y, uint32_t color)

5. Function to perform LCD delay in milliseconds
   void lcddelay(int ms){

6. Initialize LCD
   void lcd_init()

7. Function to fill rectangle
   void fillrect(int16_t x0, int16_t y0, int16_t x1, Int16_t y1, uint32_t color)

8. Function to draw Cube and its shadow using the Ray Equation

void draw_shadow (double x[], double y[], double z[], int size, double xShade, double yShade, double zShade)

9. void fillScreen(uint16_t color) {
   fillRect(0, 0, _width, _height, color);

10. Draw line function
    void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,uint32_t color)

11. Function 3D world coordinate system to viewer coordinate system and the prospective projection
    struct cordinate viewer_cordinate (int xw, int yw, int zw)

12. Function to draw cube

void draw_cube(int srt_pnt, int size)

13. Implementing Tree Pattern
    void draw_tree(uint32_t color,int srt_pnt, int size)

14. Implementing Square Pattern
    void draw_square(int ang)

15. Implementing rotated square pattern on tilted cube
    void draw_square(int start_x, int start_y ,int start_z, int len, float ang)

16. filling the sides of rotated cube with RGB colors
    void fill_rotated_cube (int start_x, int start_y, int start_z, int size , float ang)

17. Drawing the Initial H
void draw_H (start_pnt, size)
18. Drawing the initial H on rotated cube
void initial_tilted_H(int start_x, int start_y, int start_z, int size, int ang)

19. Implementing diffuse reflection on the sides of the cube
void diffused_reflection(int size)

4. Testing and Verification

This section demonstrates the process that was carried out in order to test and debug the code. Verification of the system was done in order to make sure that the output was correct and as desired.

4.1. Testing

The testing requirements are given as:

1. Make sure that all connections are proper and there is no short circuit.
2. After the hardware is built, connect the board to the development environment.
3. Build the code and debug and load it onto the flash memory.
4. Output is observed on the LCD display

Figure 11: System setup

4.2. Verification

1. After the LPC module is connected and LCD have been connected, the first step to verify the power circit and see if the LED glows on.

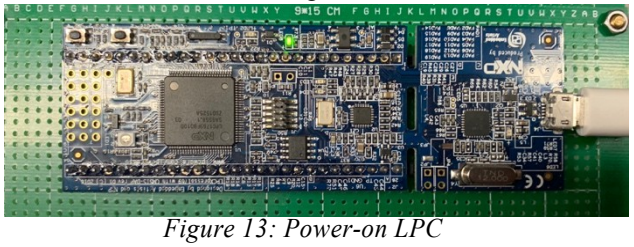

Figure 13: Power-on LPC

2. When the module has been powered up it has to be detected by the Xpresso IDE as a probe. We have used the board as a probe to debug the code. This allows us to monitor real time running of the code on the board.

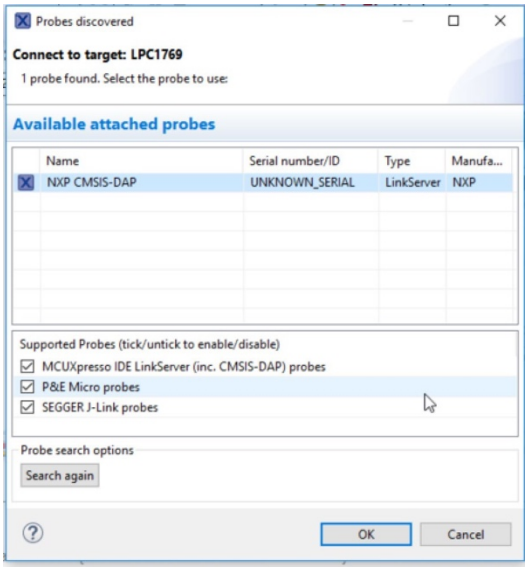

Figure 14: Probe detection

3. After this the GUI flash tool was used to load the code onto the LPC module. If the .axf file loaded properly an "operation complete" message is shown.

4. Once the flashing operation is complete, the probe needs to debug. For that we do Debug->debug as a probe.

5. "LCD demo begins" is shown on the terminal window and the screen saver appears on the boards.

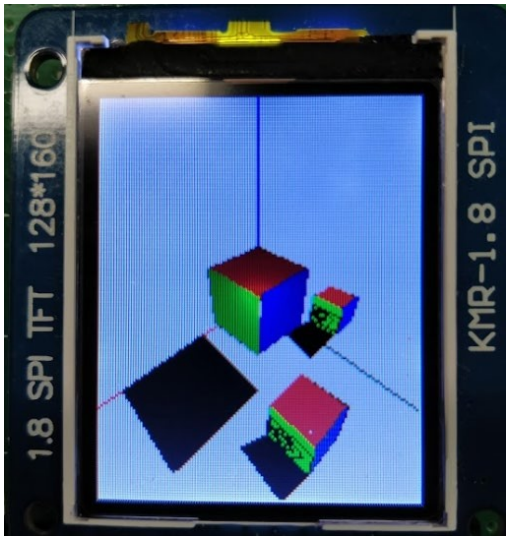

Figure 15: Diffuse Reflection and Shadow

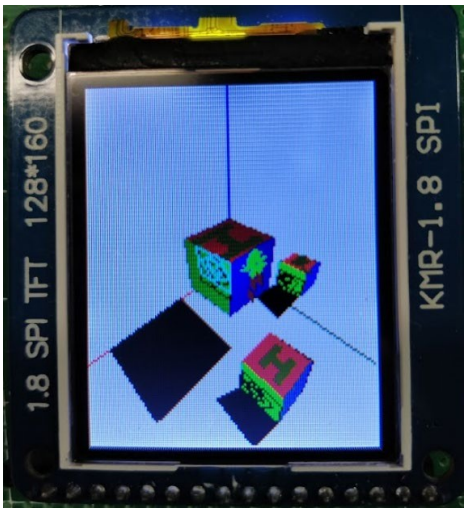

Figure 16: Final display with sqares, shadows, trees, initials and diffuse reflection

5. Conclusion

This lab helped us in understanding the SPI communications protocol and LPC module. A small application was developed to understand 2-dimensional rotation and 3-dimensional transformation and displayed on the LCD using SPI. The data sheet and and user manual was read extensively to further appreciate and understand the features of LPC1769. The conversion of objects from world coordinate to viewer coordinate was implemented successfully with proper scaling. Shadows were implemented and diffuse reflection was successfully achieved with proper scaling and gradient.

6. Acknowledgement

I would like to thank Dr. Hua Harry Li for his Guidance throughout this project. I would also like to thank my colleagues and team members who helped me with making the hardware circuit and design the software algorithms.

7. References

[1]www.github.com/hualili

[2] UM10360 Datasheet

[3]ST7735R TFT-LCD
https://learn.adafruit.com/assets/19554

9. Appendix

```
/*
===============================================
=====================================
 Name        : DrawCube.c
 Author      : Himanshu Gunjal(Adding on
Prof. Harry Hua Li's code template)
===============================================
=====================================
*/
#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>

/* 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_int16_t(a, b) { int16_t t =
a; a = b; b = t; }

struct coordinate_pnt {
      int x;
      int y;
};


struct coordinates{
int x;
int y;
};

int cam_x = 90;
int cam_y = 90;
int cam_z = 125;
int light_x = 50;
int light_y = 50;
int light_z = 60;

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

// defining color values


#define B_BLACK 0x000000
#define LT_GRAY 0xF6F3F3
#define GREEN 0x00FF00
#define LIGHTGREEN 0x80FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define YELLOW 0x666600
#define BLUE 0x0007FF
#define RED 0xFF0000
#define LIGHTRED 0xFF3333
#define BROWN 0xA52A2A
#define LIGHTBLUE 0x0080FF
#define DARK_GREEN 0x006400
#define AZURE 0xF0FFFF

#define random(x) (rand()%x)

int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;
double mySin(double a);
double myCos(double a);

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 fillrect(int16_t x0, int16_t y0,
int16_t x1, int16_t y1, uint32_t color)
{
	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 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 coordinate_pnt project_coordinates
(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 coordinate_pnt screen;
      theta =

acos(cam_x/sqrt(pow(cam_x,2)+pow(cam_y,2))
);
      phi =
acos(cam_z/sqrt(pow(cam_x,2)+pow(cam_y,2)+
pow(cam_z,2)));

      rho=
sqrt((pow(cam_x,2))+(pow(cam_y,2))+(pow(ca
m_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 coordinate_pnt lcd;
      int x1,y1,x2,y2, x3,y3,x4,y4;
      lcd = project_coordinates (0,0,0);
      x1=lcd.x;
      y1=lcd.y;
      lcd = project_coordinates (180,0,0);
      x2=lcd.x;
      y2=lcd.y;
      lcd = project_coordinates (0,180,0);
```

```
	x3=lcd.x;
	y3=lcd.y;
	lcd = project_coordinates (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,DARK_GREEN);
	//y axis  green
	draw_line(x1, y1, x4, y4,BLUE);
	//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 coordinate_pnt lcd;
	int
x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,x7,y7;
	cam_x = 120;
	cam_y = 120;
	cam_z = 120;

	lcd = project_coordinates
(start_pnt,start_pnt,(size+start_pnt));
	x1=lcd.x;
	y1=lcd.y;
	lcd = project_coordinates
((size+start_pnt),start_pnt,(size+start_pn
t));
	x2=lcd.x;
	y2=lcd.y;
	lcd = project_coordinates
((size+start_pnt),(size+start_pnt),(size+s
tart_pnt));
	x3=lcd.x;
	y3=lcd.y;
	lcd = project_coordinates
(start_pnt,(size+start_pnt),(size+start_pn
t));
	x4=lcd.x;
	y4=lcd.y;
	lcd = project_coordinates
((size+start_pnt),start_pnt,start_pnt);
	x5=lcd.x;
	y5=lcd.y;
	lcd = project_coordinates
((size+start_pnt),(size+start_pnt),start_p
nt);
	x6=lcd.x;
	y6=lcd.y;
	lcd = project_coordinates
(start_pnt,(size+start_pnt),start_pnt);
	x7=lcd.x;
	y7=lcd.y;
	draw_line(x1, y1, x2, y2,B_BLACK);
	draw_line(x2, y2, x3, y3,B_BLACK);
	draw_line(x3, y3, x4, y4,B_BLACK);
	draw_line(x4, y4, x1, y1,B_BLACK);
	draw_line(x2, y2, x5, y5,B_BLACK);
	draw_line(x5, y5, x6, y6,B_BLACK);
	draw_line(x6, y6, x3, y3,B_BLACK);
	draw_line(x6, y6, x7, y7,B_BLACK);
	draw_line(x7, y7, x4, y4,B_BLACK);
}

void draw_rotated_cube(int start_x, int
start_y ,int start_z, int size,float
angle)
{
	struct coordinate_pnt 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 = project_coordinates
(x[0],y[0],z[0]);
	x1=lcd.x;
	y1=lcd.y;
	lcd = project_coordinates
(x[1],y[1],z[1]);
	x2=lcd.x;
	y2=lcd.y;
	lcd = project_coordinates
(x[2],y[2],z[2]);
	x3=lcd.x;
	y3=lcd.y;
	lcd = project_coordinates
(x[3],y[3],z[3]);
	x4=lcd.x;
	y4=lcd.y;
	lcd = project_coordinates
(x[4],y[4],z[4]);
	x5=lcd.x;
	y5=lcd.y;
	lcd = project_coordinates
(x[5],y[5],z[5]);
	x6=lcd.x;
	y6=lcd.y;
	lcd = project_coordinates
(x[6],y[6],z[6]);
	x7=lcd.x;
	y7=lcd.y;
	lcd = project_coordinates
(x[7],y[7],z[7]);
	x8=lcd.x;
	y8=lcd.y;
	draw_line(x1, y1, x2, y2,B_BLACK);
	draw_line(x1, y1, x4, y4,B_BLACK);
	draw_line(x1, y1, x8, y8,B_BLACK);
	draw_line(x2, y2, x3, y3,B_BLACK);
	draw_line(x2, y2, x7, y7,B_BLACK);
	draw_line(x6, y6, x3, y3,B_BLACK);
	draw_line(x5, y5, x6, y6,B_BLACK);
	draw_line(x4, y4, x5, y5,B_BLACK);
	draw_line(x5, y5, x8, y8,B_BLACK);
	draw_line(x3, y3, x4, y4,B_BLACK);
	draw_line(x6, y6, x7, y7,B_BLACK);
	draw_line(x8, y8, x7, y7,B_BLACK);

	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,intColor=0,i=
0;
	struct coordinate_pnt lcd;
	uint32_t color, colorArray
[12]={B_BLACK,0x0000FFFF,0x00FF007F,0x00FF
8000,0x0000FF80,0x000000FF,0x00FFFF00,0x00
330066,0x0000FF80,0x00FF00FF,0x0000FF00,0x
000080FF};
	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 = project_coordinates
(angle,x0,y0);
		x0=lcd.x;
		y0=lcd.y;
		lcd = project_coordinates
(angle,x1,y1);
		x1=lcd.x;
		y1=lcd.y;
		lcd = project_coordinates
(angle,x2,y2);
		x2=lcd.x;
		y2=lcd.y;
		lcd = project_coordinates
(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,intColor=0,i=
0;
	struct coordinate_pnt lcd;
	int x,y;
	uint32_t color, colorArray
[12]={B_BLACK,0x0000FFFF,0x00FF007F,0x00FF
8000,0x0000FF80,0x000000FF,0x00FFFF00,0x00
330066,0x0000FF80,0x00FF00FF,0x0000FF00,0x
000080FF};
	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 = project_coordinates
(x,y,y0);
		x0=lcd.x;
		y0=lcd.y;

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


		x = x2;
		y = start_y;
		rotate_point(&x, &y, angle);
```

```
		lcd = project_coordinates(x,
y,y2);
		x2=lcd.x;
		y2=lcd.y;

		x = x3;
		y = start_y;
		rotate_point(&x, &y, angle);
		lcd = project_coordinates(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_tree(uint32_t color,int
start_pnt, int size)
{
	int i=0, angle;

	struct coordinate_pnt lcd;
	int
tree_branch[3][3]={{start_pnt,start_pnt+20
,0.5*size},
	{start_pnt+10,start_pnt+20,0.3*size}
,
	{start_pnt+15,start_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 = project_coordinates
(y0,angle,x0);
		xp0=lcd.x;
		yp0=lcd.y;
		lcd = project_coordinates
(y1,angle,x1);
		xp1=lcd.x;
		yp1=lcd.y;
		draw_line(xp0, yp0, xp1,
yp1,BROWN);  //level 0 straight line
		draw_line((xp0+1), (yp0+1),
(xp1+1), (yp1+1),BROWN);  //level 0
straight line
		draw_line((xp0-1), (yp0-1),
(xp1-1), (yp1-1),BROWN);  //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 =
project_coordinates  (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 =
project_coordinates (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 =
project_coordinates  (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 =
project_coordinates (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 =
project_coordinates (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 =

project_coordinates (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 =
project_coordinates (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 =
project_coordinates (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 =
project_coordinates (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 coordinate_pnt 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=project_coordinates(x,y, zsize);
      //top fill green

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

            }
      }

      for(i = start_y; i < ysize; i++)
      {

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

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

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

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

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

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 coordinate_pnt 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 = project_coordinates
(xs[4],ys[4],zs[4]);
	s6 = project_coordinates
(xs[5],ys[5],zs[5]);
	s7 = project_coordinates
(xs[6],ys[6],zs[6]);
	s8 = project_coordinates
(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);
}
void draw_rotated_A(int start_x , int
start_y , int start_z, int size , int
rotated_angle)
{
	int  x0,y0,x1,y1,x2,y2;
	int  x3,y3,x4,y4,x5,y5;
	int x,y;
	struct coordinate_pnt p1;
	x0 = start_x + size/2;
	y0 = start_y + size/10;
	x1 = start_x + size*0.8;
	y1 = start_y + size*0.8;
	x2 = start_x + size/10;
	y2 = start_y + size*0.8;


	x = x0 ; y = y0 ;
	rotate_point(&x, &y, rotated_angle);
	p1 = project_coordinates(x, y,
start_z+size);
	x3 = p1.x;
	y3 = p1.y;
	x = x1 ; y = y1 ;
	rotate_point(&x, &y, rotated_angle);
	p1 = project_coordinates(x, y,
start_z+size);
	x4 = p1.x;
	y4 = p1.y;
	x = x2 ; y = y2 ;
	rotate_point(&x, &y, rotated_angle);
	p1 = project_coordinates(x, y,
start_z+size);
	x5 = p1.x;
	y5 = p1.y;

	printf("Rotated values\n x3 %d y3 %d
x4 %d y4 %d x5 %d y3
%d",x3,y3,x4,y4,x5,y5);
	draw_line(x3, y3, x4, y4, BLACK);
	draw_line(x3, y3, x5, y5, BLACK);
	//draw_line(x0, y0, x1, y1, BLACK);
}
```

```
void draw_H(int start_pnt, int size)
{
	struct coordinate_pnt p1;
	int i,j;
	size=size+start_pnt;
	int map[size][size];


	for(i = start_pnt; i < size;i++)
	{
		for(j = start_pnt; j <
size;j++)
		{
			if(i>=6 && i<=13 &&
j>=6 && j<=44)

	map[i][j]=1;
			else if(i>=17 && i<=34
&& j>=20 && j<=30)

	map[i][j]=1;
			else if(i>=34 && i<=41
&& j>=6 && j<=44)

	map[i][j]=1;

			else

	map[i][j]=0;
		}
	}

	for(i=0;i<size;i++)
	{
		for(j=0;j<size;j++)
		{
			if(map[i][j]==1)
			{
				p1 =
project_coordinates(j,i,size);

	draw_pixel(p1.x,p1.y,YELLOW);
			}
			else if(map[i][j]==0)
			{
				p1 =
project_coordinates(j,i,size);
			}
		}
	}
}

void initial_tilted_H(int x_dist, int
y_dist,int z_dist,int size)
{
	struct coordinate_pnt p1;
	int i,j;
	int map[size][size];
	double gamma = 3.14/9;

	for(i=0;i<size;i++)
	{

		for(j=0;j<size;j++)
		{
			if(i>=4 && i<=9 &&
j>=5 && j<=25)
				map[i][j]=1;
			else if( i>=9 && i<=17
&& j>=13 && j<=17)   //for H
				map[i][j]=1;
			else if(i>=17 && i<=22
&& j>=5 && j<=25)
				map[i][j]=1;
			else
				map[i][j]=0;
		}
	}

for(i=0;i<size;i++)
{

  for(j=0;j<size;j++)
  {
	if(map[i][j]==1)
	{
		p1=
project_coordinates(j+x_dist,i*cos(gamma)+
size*sin(gamma)+y_dist,i*(-
sin(gamma))+size*cos(gamma)+z_dist);
		draw_pixel(p1.x,p1.y,YELLOW);
	}
  }
 }
}

int calIDiff(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 coordinate_pnt lcd;
```

```
	struct coordinate_pnt tmp;
	int x1, y1, x2, y2, x3, y3, x4, y4,
x5, y5, x6, y6, x7, y7, x8, y8;
	lcd = project_coordinates(0, 0, 0);
	x1 = lcd.x;
	y1 = lcd.y;
	lcd = project_coordinates(size, 0,

0);
	x2 = lcd.x;
	y2 = lcd.y;
	lcd = project_coordinates(0, size,
0);
	x3 = lcd.x;
	y3 = lcd.y;
	lcd = project_coordinates(0, 0,
size);
	x4 = lcd.x;
	y4 = lcd.y;
	lcd = project_coordinates(size, 0,
size);
	x5 = lcd.x;
	y5 = lcd.y;
	lcd = project_coordinates(size,

size, 0);
	x6 = lcd.x;
	y6 = lcd.y;
	lcd = project_coordinates(size,
size, size);
	x7 = lcd.x;
	y7 = lcd.y;

	lcd = project_coordinates(0, size,
size);
	x8 = lcd.x;
	y8 = lcd.y;


	for (int i = 0; i <= size; i++) {
		for (int j = 0; j <= size;

j++)
		{
			tmp =

project_coordinates(i, j, size);
			int kR =
calIDiff(light_x, light_y, light_z, i, j,
size, 255);


			if (kR + 170 > 255)

				kR = 255;
			else
				kR += 170;

			 0x000000) <<
0x000000;	 16);
```

```
			color |= (0x000000 <<
8);
			color |= 0x000000;
			draw_pixel(tmp.x,
tmp.y, color);
		}
	}

    for (int i = 0; i <= size; i++)
    {
      for (int j = 0; j <= size; j++)
      {
      tmp = project_coordinates
(size,i,j);
      int kR = calIDiff(light_x, light_y,
light_z, size, i, j, 255);
      if (kR + 170 > 255)
      {
          kR = 255;
          }
      else
          {
             kR += 170;
          }

      uint32_t color = 0x000000;
      color |= (0x000000 << 16);
      color |= ((kR |= 0x000000) << 8);
      color |= 0x000000;
      draw_pixel(tmp.x,tmp.y,color);
      }
    }

    for (int i = 0; i <= size; i++)
    {
		for (int j = 0; j <= size;
j++) {

		tmp = project_coordinates
(i,size,j);
		int kR = calIDiff(light_x,

light_y, light_z, size, i, j, 255);
		if (kR + 170 > 255)
		{

			kR = 255;
			}
		else
			{

				kR += 170;
			}

		uint32_t color = 0x000000;
		color |= (0x000000 << 16);
		color |= (0x000000 << 8);
		color |= (kR |= 0x000000);
uint32_t color =

color |= ((kR |=
```

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

```c
int main(void)
{
	uint32_t pnum = PORT_NUM;

	int size = 45, start_pnt = 0;
	double x[8] =
{start_pnt,(start_pnt+size),(start_pnt+siz
e),start_pnt,start_pnt,(start_pnt+size),(s
tart_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");


	//To initialize LCD
	lcd_init();
	fillrect(0, 0, ST7735_TFTWIDTH,
ST7735_TFTHEIGHT, LT_GRAY);

	//Draw XYZ axes
	draw_coordinates();
	//Draw shadow for first cube
	draw_shadow(x, y, z, size, -
500,0,500);
	draw_cube(start_pnt,size);
	//For shifted and rotated cube
	draw_rotated_cube(20, 60, -50, 30,
90);
	fill_rotated_cube(20, 60, -50, 30,
90);
	draw_rotated_xz_square(20, 60, -50,
30, 90);
	//For shifted and tilted cube
	draw_rotated_cube(70, -60, -40, 30,
90);
	fill_rotated_cube(70, -60, -40, 30,
90);
	draw_rotated_xz_square(70, -60, -40,
30, 90);

	//Diffuse reflection on the big cube
	diffused_reflection(size);
	draw_H(start_pnt, size);
	initial_tilted_H(20, 60, -50, 30);
	draw_square(size + start_pnt);
	draw_tree(GREEN,start_pnt,size);
	initial_tilted_H(20, 70, 45, 15);

	return 0;
}
```