3D Graphic Engine 3D cube rotation shadow and diffusion reflection

Prashant Gandhi - 013712361
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
prashantshushilbhai.gandhi@sjsu.edu

Abstract

LPC 176x/5x does not have in-built graphic engine feature. This document gives detail explanation for designing graphic engine using LPC1769 development board. Design uses color LCD to display 2D and 3D graphics. Color LCD uses SPI communication protocol for interfacing, due that connection between color LCD and LPC1769 development kit is done using SPI protocol. To program LPC1769 development board MCUExpresso IDE is used. Objective of this project is to display 3D coordinates, then draw 3D cube, then rotate that cube and then decorate all visible sides of cube with name initials, forest trees and rotating squares which is accomplished and tested on implemented circuit.

1. Introduction

This project uses LPCXpresso board for LPC1769. This board is low-cost development board provided by NXP. This board uses ARM based Cortex-M3 microcontroller. Cortex-M3 is 32-bit byte addressable microcontroller, it has up to 64 KB data memory and provides very effective debug options. For better understanding of LPC1769 we can divide its architecture in 3 parts which is shown in figure 1.1. Figure shows that it has some spaces for adding some extra features in it. We can use that section to create 2D and 3D graphic engine. To display 2D and 3D graphics, design uses 1.8" TFT color LCD which has 128x160 color pixels and comes with TFT ST7735R driver. LPC1769 development board and color LCD are connected through SPI (Serial Peripheral Interface) protocol. SPI protocol has higher transmission rate than the other communication protocol and we need approximately 4 Mbps transmission speed.

From block diagram we can see that we can add new features to LPC1769. So, we are adding graphic engine to LPC1769. This paper goal is to implement 2D and 3D graphics and display it on color LCD. This paper gives detail of how to implement and design trees and routing square screen saver on color LCDWe will first draw 3D coordinate axis on LCD for that we will use virtual transformation and move origin to center of the LCD. After that we will draw 3D cube using that axis, then using light source we will implement shadow of the cube on x-y plane, after that we will rotate the cube. Finally, we will decorate cube's sides and do diffusion reflection.

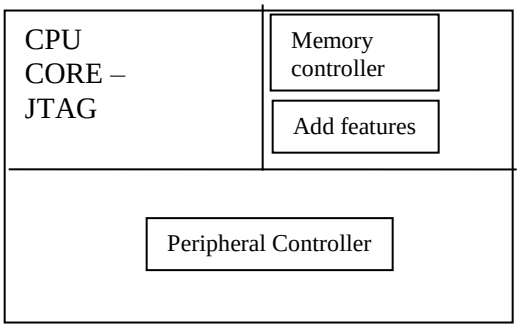


Figure-1.1: Architecture block diagram of LPC1769

2. Methodology

This section of paper provides detailed information of the project's objectives and what challenges were encountered while designing hardware and software part of the projects.

2.1. Objectives and Technical Challenges

Objective of this project is to display 3D coordinates, 3D cube with decoration, rotate the cube and diffusion reflection on cube's surface on color LCD using LPC1769 development kit. Detailed objective of this project is given below. Also, while implementing this project, encountered challenges are described below.

The objective of the lab is as follows:

1. First, deign power circuit which take input from AC power supply and using power adaptor which gives 12V DC and 1.25 A current as output. This output is given to circuit and converting 12V dc to 5V DC using LM7805 and this output is given to LPC1769 development kit.
2. Interface LPC1769 to color LCD using SPI.
3. Write a program that display 3D coordinates on color LCD.
4. Write a program that display 3D cube.
5. Write a program that show cube's shadow, decorate cube, rotate it and do diffusion reflection from cube's surface.

The challenges encountered were as follows:

1. Understand microcontroller architecture and pin diagram.

2. Understand SPI driver and design algorithm for rotating square and tree.
3. Clean shouldering.

2.2. System Layout and Design

Block diagram of this project is given in figure 2.2.1. design includes power circuit, LPC1769 development kit, color LCD and laptop. Power circuit is used to give power to LPC1769 which needs 5V as input. Using laptop, we program LPC1769 using MCUXpresso IDE. LPC1769 is connected to color LCD using SPI. Power adaptor which covert AC to 12V DC is used to supply power to whole circuit.

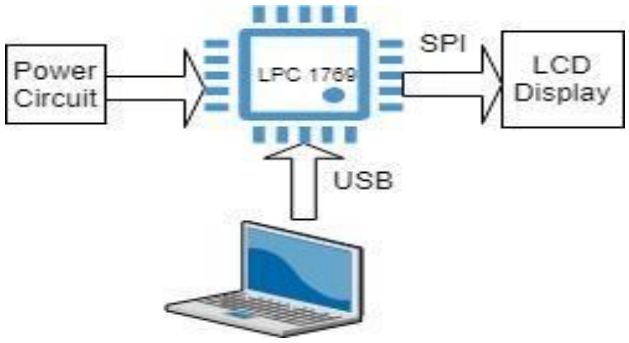


Figure-2.2.1 System Block Diagram

To display 3D coordinates and 3D cube the first process will be to do virtual transformation and move LCD origin pixel to center. Then we will do 3D transformation pipeline Technique in which first step will be world to viewer transformation whose formula is as below.

$$\mathbf{T} = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

……. equation-1

Second step will be to do perspective projection whose formula is as below.

$$x_p = x_e\left(\frac{D}{z_e}\right)$$

$$y_p = y_e\left(\frac{D}{z_e}\right)$$

……. equation-2

After these two steps we 3D points are converted to 2D and we can apply line (equation-3) formula to join this points to draw shapes.

P(x,y) = P_1(x_1,y_1) + lambda * (P_2(x_2,y_2) - P_1(x_1,y_1)) ……. equation-3

3. Implementation

Implementation is divided in 2 parts.

1.Hardware Design

2.Software Design

3.1. Hardware Design

The hardware aspect involves the designing of the power circuit, testing circuit and the SPI interface between the LPC and LCD.

The table for the hardware components is as shown below:

SL.NO	Component	Description	Notes
1	Wire wrapping Board	Wire wrapping Board	Qty-2
2	Power Adapter	12V DC	~1.25A
3	LM7805	Power Regulator	Output 5V
3	Resistor	1k	
4	Capacitors	0.1microF	Quantity-2
5	LPC Module	LPC1769	For Computing
6	LCD Display	SPI Interface LCD	ST7735R SPI 126*160 TFT LCD Display
7	Switch	Toggle and Push button Switch	

Table 1 Bill of Material

The power circuit is supplied with a power of 12V by an external adapter which is scaled down using the voltage regulator LM7805. The regulator is connected to appropriate resistor and capacitors so that there is a continuous supply of power to the LPC module. This is tested using an LED which is connected to the power circuit and is used to test if the circuit is up and running. It can be designs as shown in Figure 3.1.1.

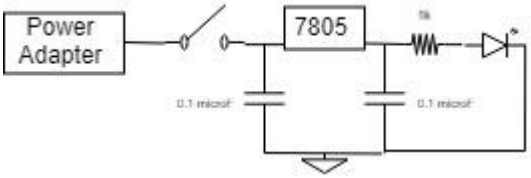


Figure-3.1.1. Power circuit

The power circuit is connected to the voltage regulator and the LPC module as shown above. The testing circuit is also connected to the LPC module to ensure that the LPC is up and running. It is also connected via

appropriate resistors so that the LED glows continuously ensuring that the LPC has a Vout of 3.3 V.

S.NO	CPU Pin	Description	Connector Pin
1	P0.2	General Purrpose O/P Pin	J2-21
2	P0.3	General Purpose Input Pin	J2-22
3	Power	5V DC	J2-2
4	Connector	PWR Connector	J2-1

Table 2 Connection of Input Output

The LPC and LCD are interfaced using the SPI that follows a serial communication protocol. It is shown in the below figure 3.1.2:

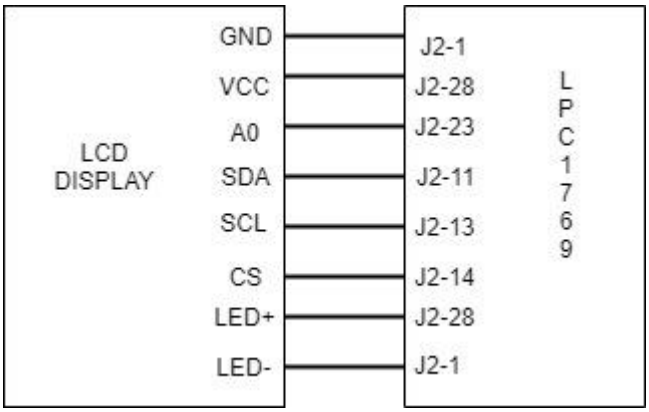


Figure 3.1.2. PIN Connection between LCD and LPC

Figure 3.1.2. PIN Connection between LCD and LPC

The serial communication takes place using 4 signals which are MOSI, MISO, SCLK and CS. The LPC communicates with the LCD via MOSI signal which means that the LPC is the master and the LCD is the slave. The CS has only one option of using the salve which is the LCD. The SCLK is used to synchronize the data signals

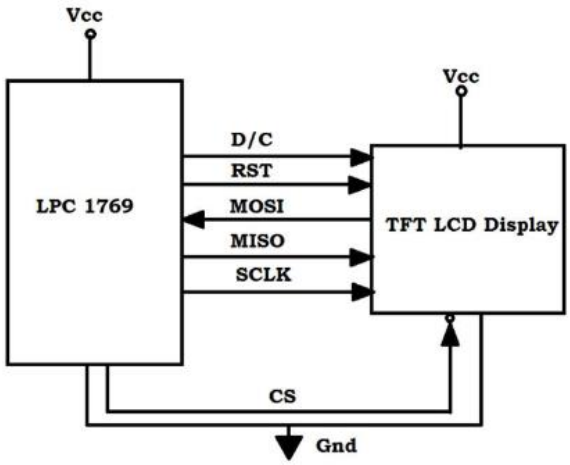


Figure 3.1.3. LPC to LCD signal

The LCD has a resolution of 128x160 and the back light indicates that it is running. The LPC module is where the code is dumped by the IDE and the LPC communicates with the LCD with the appropriate signals so that the code can be implemented and verified on the LCD. The interface between the two is shown in table-3.

Table 3. Interface between LCD and LPC and the slave sends the master a MISO signal

LCD BOARD	LCD Board Pin	LPC PIN	LPC Board Pin
LED-	16	Gnd	J2-1/J2-54
LED+	15	VIO_3V3X	J2_28
CS	10	SSEL/SSE0	P0.16/J2-14
SCL	9	SCK.SCK0	P0.15/J2-13
SDA	8	MOSI/MOSI0	P0.9/J2-11
AO	7	DATA & CONTROL	P0.21/J2-23
RST	6	RESET	P0.22/J2-24
VCC	2	VIO_3V3X	J2-28
GND	1	Gnd	J2-1/J2-54
MISO	12	MISO/MISO0	Not Used

Table 3 Pin connection between LPC and LCD

Real hardware is shown in figure 3.1.3.

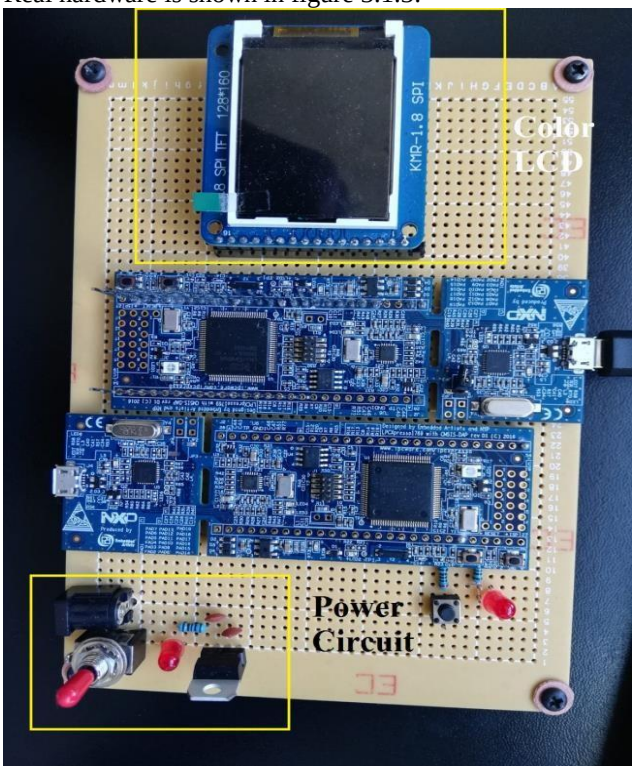


Figure 3.1.3 Hardware

3.2 Vector Calculation

We will do 3D to 2D conversion in order to display 3D images and coordinates on LCD.

1. 2D rotation matrix:

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

2. 3D tranformation matrices for X, Y and Z axes:

For X axis:

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

For Y axis:

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

For Z aixs:

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

3. World to viewer tranformation:

$$\mathbf{T} = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. Perspective Projection:

$$x_p = x_e\left(\frac{D}{z_e}\right)$$

$$y_p = y_e\left(\frac{D}{z_e}\right)$$

3.3 Software Design

3.3.1 Algorithm

Algorithm for 3D coordinate, 3D cube, Cube decoration, Shadow of cube, cube rotation and diffusion reflection is as below.

Step 1: Start

Step 2: Initialize SPI

- Set PCONP's 21st bit to enable SSP0
- SSP_CLK selected as PCLK/4, by writing PCLKSEL1 as 0
- Set J6 11-14 pins functionality as SSP 0
- Set SSEL0 as GPIO out
- SSP0 data width is set to 8 bit
- SCR register value to 7
- Pre-scale CLK value set to 2

Step 3: Initialize LCD

- Set SSEL0 as 0, to make LCD slave
- D/C connected to J6-23
- RESET connected to J6-24
- Set both pins as output
- P0.24 pin value is set as logic 1
- Provide delay of 500 ms
- P0.24 pin value is set as logic 0
- P0.24 pin value is set as logic 1
- Provide delay of 500 ms
- P0.24 pin value is set as logic 0
- Initialize SSP buffer to 0
- Send 0x11 to SSP 0 to wake LCD from sleep
- Give a delay to LCD
- Send 0x29 to SSP0 to display
- Give a delay to LCD

Step 4: Draw the 3D world coordinates

- Fill the entire LCD display with black using fill rectangle method
- Do virtual transformation and set center of LCD as origin
- Perform transformation pipeline technique using equation (1) and equation (2) and covert points in 3D coordinates to 2D coordinates.
- Draw the x,y,z axes in red, green and blue respectively

Step 5: Draw the 3D cube

- Draw the 3D using transformation pipeline technique and apply line equation-3 to draw edges of cube.
- Fill the 3 visible surfaces with different Colors

Step 6: Draw shadow of the cube using below equation on the x-y plane.

P = Ps + Lambda*(Ps – Pi) ……..Equation (3)

N*(a-v) = 0 ……Equation (4)

Here, P,Ps,Pi, n, a, and v are vectors

Step 7: Decorate Surface S1 with square pattern screensaver

Step 8: Decorate Surface S2 with Trees

Step 9: Draw the initial font ('P' in this case) on surface S3

Step 10: Rotate second cube using below steps.

- 3D rotation with respect to Z_w axis is implemented
- 3D rotation with respect to X_w is implemented
- 3D rotation with respect to Y_w is implemented
- Actual rotation w.r.t Z_w by θ is done which is $R_{zw}(\theta)$
- Undo the third step using $R^{-1}_{yw}=R_{yw}(\theta)$
- Undo the second step using $R^{-1}_{zw}=R_{zw}(\theta)$

- Undo step one using $T^{-1}_{(x)}=T_{(-x)}$

Step-11: Do diffusion reflection on this cube surface using below formula. For that we will calculate intensity of each and every pixel of top surface and as from the below formula we can see that it is a function of distance, so we can see its intensity will change as its distance from light source change.

$$Ir = \frac{Kdr * n.r}{|n| * |r| * |r|^2}$$

Here, Kdr is intensity coefficient, n and r are vectors. n is normal vector and r is a distance vector from light source.

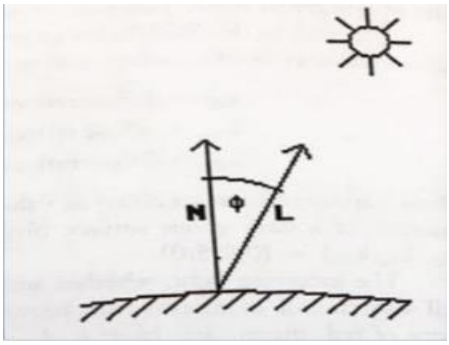
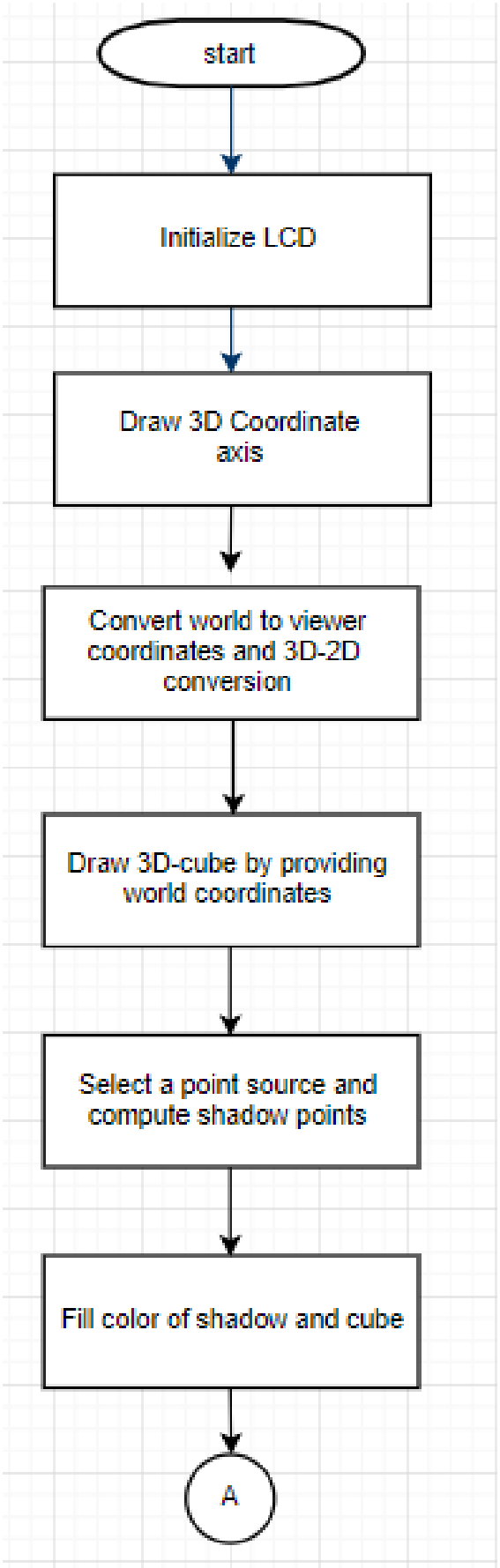
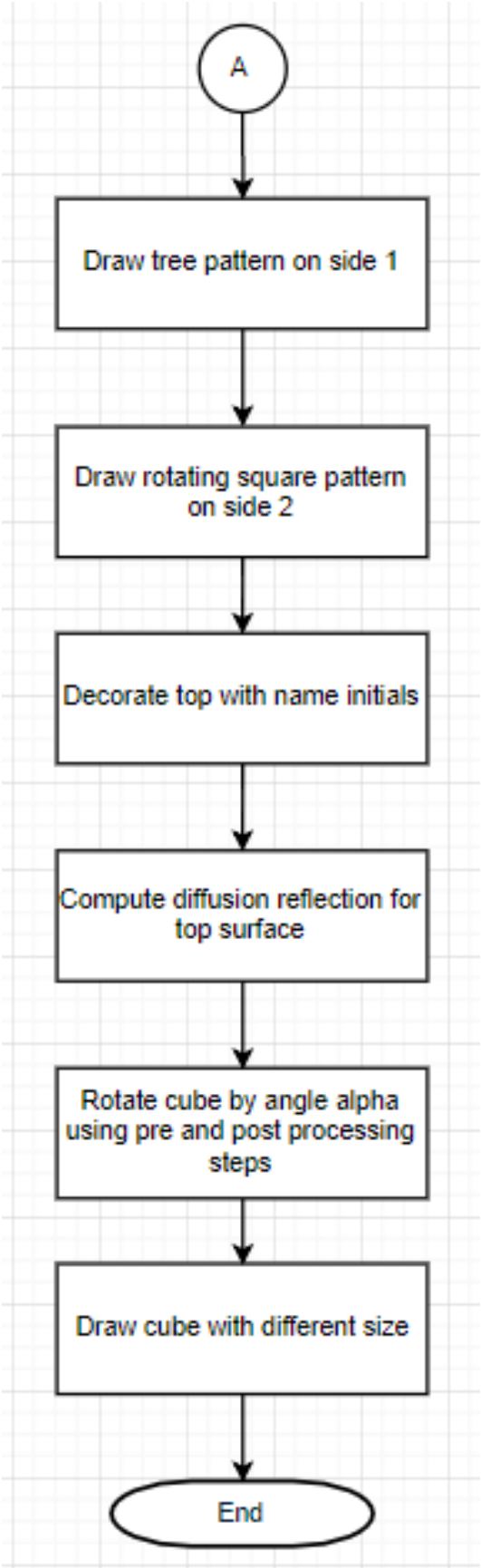


Figure 3.2.1 Diffusion Reflection.

Step 10: Stop

3.3.2 Flowchart





3.3.3 Pseudo code

1. Initialize LCD
- SSP0Init(): Initialize and construct SSP0 registers and parameters
- Select Port0 and initialize fillrect(0,0,ST7735_TFTWIDTH,ST7735_TFTHEIGHT, BLACK)
2. Do transformation pipelining
- World to viewer conversion
- Perspective projection and convert 3D to 2D.
3. Draw 3D Cube
- Select base points and size of the cube.
- Convert 8 vertices from 3D to 2D
- Join vertices to form a 3D-cube
4. Compute Shadow points
- Select a light source point and draw lines intersecting top 4 vertices of the cube.
- Compute lambda to find the intersection of shadow points with a plane
- Join shadow points and fill color on the shadow points
5. Rotate Cube
- Rotate cube using pre-processing and post-processing with respect to angle theta using matrix multiplication
- Generate 3 cubes with varying size and locations with rotation by angle theta
6. Shade cube using fill_color function API and fill_pixel function API
7. Add Tree pattern to 1 side of the cube
8. Add Rotating square pattern to another side of the cube.
9. Add initial on top surface of cube. Fill the initial with color.
10. Compute diffuse reflection for to surface of the cube
- Use the formula for calculation of intensity in computation of diffuse reflection.
- Select Kdr = 0.8 to compute diffuse reflection for predominantly red color.

4. Testing and Verification

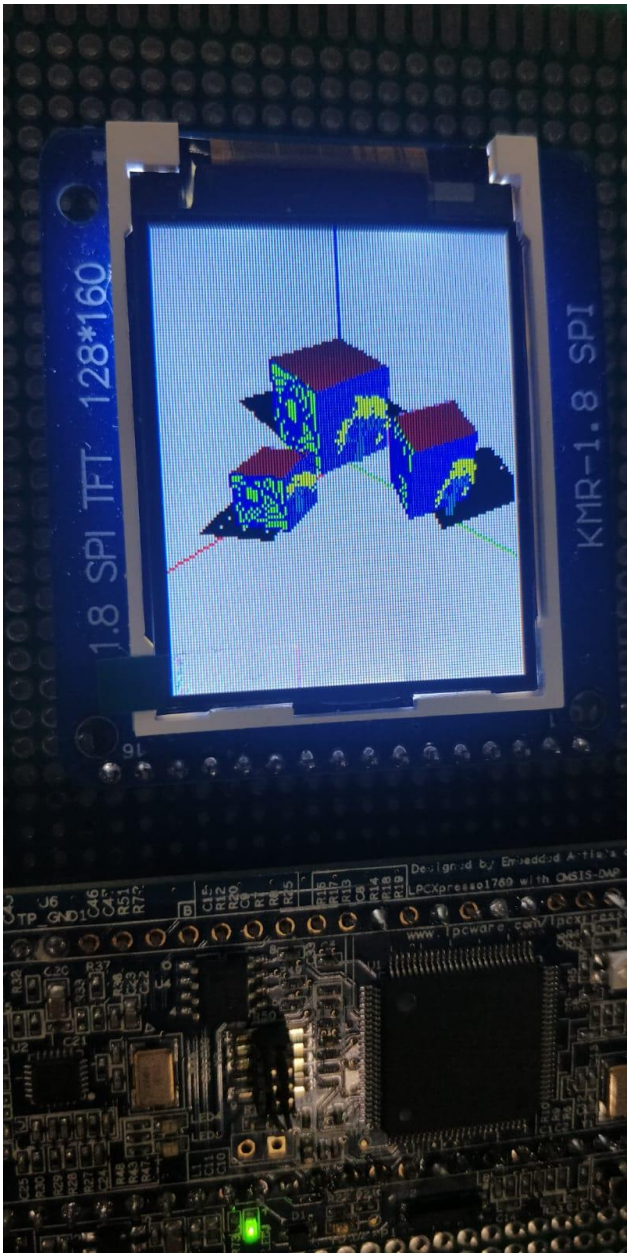


Figure 4.1 Final Output

5. Conclusion

The first step of this project was to understand LPC1769 thoroughly and based on that knowledge we implemented 3D graphic engine. Power circuit and testing circuit for LPC1769 was basic and first step of this project which is working well. After that we understand SPI protocol which helped us to interface LPC1769 with color LCD. In software part we started with simple code to draw line on color LCD. Based on that knowledge we implemented 3D coordinate axis and 3D cube using formulas given by Professor Harry Li in class. After that we programmed LPC1769 to draw cube, its shadow, diffusion reflection of it top surface and test on the LCD and it is working well. To sum up all, in this project we learnt about to design 3D graphic design.

6. Acknowledgement

I would like to express my gratitude and appreciation to our Professor Harry Li for his encouragement, valuable guidance and patient review. His constant suggestion on hardware design helped us to implement good hardware. Also, I would like to thank him for giving us such a valuable knowledge of how to program LPC1769 development kit. At last, I would also like to thank my project partners for their valuable inputs.

7. References

[1]https://github.com/hualili/CMPE240-Adv- Microprocessors.
[2] https://www.nxp.com/docs/en/user-uide/UM10360.pdf

8. Appendix

LAB_2_3D_GE.h

```
#ifndef Three_D_cube_h_
#define Three_D_cube_h_

#include <stdint.h>
#define PI 3.142
#define D 300

typedef struct coordinates_2D
{
        int16_t x;
        int16_t y;
}_2d_coordi;
typedef struct coordinates_3D
{
        float x;
        float y;
        float z;
}_3d_coordi;

typedef struct __cube
{
        _2d_coordi P1, P2, P3, P4, P5, P6, P7, P8;
        _3d_coordi P1_3D, P2_3D, P3_3D, P4_3D, P5_3D, P6_3D, P7_3D, P8_3D;

        _2d_coordi Sh1, Sh2, Sh3, Sh4;
        _3d_coordi Sh1_3D, Sh2_3D, Sh3_3D, Sh4_3D;
}_3d_cube;

typedef struct coordinates_for_initials
{
        _3d_coordi A1, A2, A3, A4, A5, A6, A7;
        _2d_coordi A1_2D, A2_2D, A3_2D, A4_2D, A5_2D, A6_2D, A7_2D;
}__initials;
```

```
#define pgm_read_byte(addr) (*(const unsigned char *)(addr))

// Display 3D cube on LCD (Draw cube)
void _3D_CUBES(_3d_cube *cube, float size_of_cube, float x_base, float y_base, float z_base);

// Draw 3D coordinates
void _3D_AXIS();


void background(int16_t x, int16_t y, int16_t w, int16_t h, uint32_t color);

void _CUBE_ROTATION(_3d_cube *cube1, _3d_cube *cube2, float , float );

void LIGHT_SOURCE_POINT(_3d_cube *cube, float , float , float );

void SHADOW_CUBE(_3d_cube *cube, int );

void PARA_CALCULATION();

void squarePattern(int x1, int y1, int x2,int y2,int x3,int y3,int x4,int y4);

void treePattern(int x, int y, float angle, int length, int level, int color);

void NAME_INITIALS(_3d_cube* cube, float );

#endif
```

LAB_2_3D_GE.c

```
#include <LAB_2_3D_GE.h>
#include "LPC17xx.h"
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "DrawLine.h"

// To draw Vertical line
void drawFastVLine(int16_t x, int16_t y, int16_t h, uint32_t color)
{
	drawline(x, y, x, y + h, color);
}

// To fill the background with given color and dimensions
void background(int16_t x, int16_t y, int16_t w, int16_t h, uint32_t color)
{
	int16_t i;
	for (i = x; i < x + w; i++)
	{
		drawFastVLine(i, y, h, color);
	}
}

static float sin_theta, cos_theta, sin_phi, cos_phi, rho;
const float xcamera = 300.00, ycamera = 300.00, zcamera = 300.00;

void PARA_CALCULATION()
{
	rho = sqrt((pow(xcamera, 2)) + (pow(ycamera, 2)) + (pow(zcamera, 2)));
	sin_theta = (ycamera / sqrt(pow(xcamera, 2) + pow(ycamera, 2)));
	cos_theta = (xcamera / sqrt(pow(xcamera, 2) + pow(ycamera, 2)));
	sin_phi = sqrt(pow(xcamera, 2) + pow(ycamera, 2)) / rho;
	cos_phi = (zcamera / rho);
}

void physical_coordinate(int16_t *x, int16_t *y)
{

	//Convert virtual imagined coordinate to physical coordinate
	*x =  (ST7735_TFTWIDTH/2) + *x;
	*y = (ST7735_TFTHEIGHT/2) - *y;

}

// To convert world to viewer coordinates - Viewer to 2D Coordinate
//virtual coordinate to physical/native coordinate
_2d_coordi TRANFER_PIPELINE_3D_TO_2D(float xw, float yw, float zw)
{
	int16_t xdoubleprime = 0, ydoubleprime = 0; /*D = 100.00*/
	float xPrime = 0.00, yPrime = 0.00, zPrime = 0.00;
	_2d_coordi projection = {0, 0};

	/************************** To convert world to viewer coordinates *************/

	xPrime =   ((-sin_theta) * xw) + (yw * cos_theta);
	yPrime = (-(xw) * cos_theta * cos_phi) + (-(yw) * cos_phi * sin_theta) + (zw * sin_phi);
	zPrime = (- (xw) * sin_phi * cos_theta) + (- (yw) * sin_phi * cos_theta)
				+ (- (zw) * cos_phi) + rho;
	//printf("(xPrime, yPrime, zPrime): (%f, %f, %f)\n", xPrime, yPrime, zPrime);
```

```
	/**************************************
**************************************/

	//Convert Viewer to 2D Coordinate
	xdoubleprime = (xPrime * D) / zPrime;
	ydoubleprime = (yPrime * D) / zPrime;
	//printf("(xdoubleprime, ydoubleprime): (%f,
%f)\n", xdoubleprime, ydoubleprime);

	//Convert virtual imagined coordinate to
physical/native coordinate
	physical_coordinate(&xdoubleprime,
&ydoubleprime);

	projection.x = xdoubleprime;
	projection.y = ydoubleprime;
	return projection;
}

// To draw the world 3D coordinate system - x,y,z
void _3D_AXIS()
{
	_2d_coordi axis;
	int16_t x1, y1, x2, y2, x3, y3, x4, y4;

	//Origin
	axis = TRANFER_PIPELINE_3D_TO_2D(0.00,
0.00, 0.00);
	x1 = axis.x;
	y1 = axis.y;

	//X-Axis
	axis =
TRANFER_PIPELINE_3D_TO_2D(180.00, 0.00, 0.00);
	x2 = axis.x;
	y2 = axis.y;

	//Y-Axis
	axis = TRANFER_PIPELINE_3D_TO_2D(0.00,
180.00, 0.00);
	x3 = axis.x;
	y3 = axis.y;

	//Z-Axis
	axis = TRANFER_PIPELINE_3D_TO_2D(0.00,
0.00, 180.00);
	x4 = axis.x;
	y4 = axis.y;

	drawline(x1, y1, x2, y2, RED);
	  //x axis  Red
	drawline(x1, y1, x3, y3, GREEN);		//y
axis  Green
	drawline(x1, y1, x4, y4, BLUE);		//z
axis  Blue
}


void fillcube(_3d_cube *cube)
{
	float i,j;
	_3d_coordi t1, t2;
	_2d_coordi m1, m2;
	/*************************************SIDE
SURFACE*****************************************
************/
	for(i = cube->P3_3D.x, j = cube->P6_3D.x; i >
cube->P4_3D.x && j > cube->P7_3D.x; i--, j--)
	{
		t1.x = i;
		t1.y = (((cube->P4_3D.y - cube-
>P3_3D.y) / (cube->P4_3D.x - cube->P3_3D.x)) * (t1.x -
cube->P3_3D.x)) + cube->P3_3D.y;
		t1.z = (((cube->P4_3D.z - cube-
>P3_3D.z) / (cube->P4_3D.x - cube->P3_3D.x)) * (t1.x -
cube->P3_3D.x)) + cube->P3_3D.z;
		//printf("Value of P1: (%f, %f, %f)\n",
t1.x, t1.y, t1.z);

		t2.x = j;
		t2.y = t1.y;
		t2.z = (((cube->P7_3D.z - cube-
>P6_3D.z) / (cube->P7_3D.x - cube->P6_3D.x)) * (t1.x -
cube->P6_3D.x)) + cube->P6_3D.z;
		//printf("Value of P2: (%f, %f, %f)\n\n",
t2.x, t2.y, t2.z);

		m1 =
TRANFER_PIPELINE_3D_TO_2D(t1.x, t1.y, t1.z);
		m2 =
TRANFER_PIPELINE_3D_TO_2D(t2.x, t2.y, t2.z);
		drawline(m1.x, m1.y, m2.x, m2.y,
BLUE);
	}

	/*************************************TOP
SURFACE*****************************************
************/
	for (i = cube->P2_3D.x, j = cube->P3_3D.x;
	i > cube->P1_3D.x && j > cube->P4_3D.x; i--,
j--)
	{
		t1.x = i;
		t1.y = (((cube->P1_3D.y - cube-
>P2_3D.y)/ (cube->P1_3D.x - cube->P2_3D.x)) * (t1.x -
cube->P2_3D.x)) + cube->P2_3D.y;
		t1.z = cube->P2_3D.z;
		//printf("Value of P1: (%f, %f, %f)\n",
t1.x, t1.y, t1.z);

		t2.x = j;
```

```
                t2.y = (((cube->P4_3D.y - cube->P3_3D.y)/ (cube->P4_3D.x - cube->P3_3D.x)) * (t1.x - cube->P3_3D.x)) + cube->P3_3D.y;
                t2.z = cube->P2_3D.z;
                //printf("Value of P2: (%f, %f, %f)\n\n", t2.x, t2.y, t2.z);

                m1 = TRANFER_PIPELINE_3D_TO_2D(t1.x, t1.y, t1.z);
                m2 = TRANFER_PIPELINE_3D_TO_2D(t2.x, t2.y, t2.z);
                drawline(m1.x, m1.y, m2.x, m2.y, WHITE);
        }
        /*************************************FRONT SURFACE*****************************************/
                for(i = cube->P2_3D.y, j = cube->P5_3D.y; i < cube->P3_3D.y && j < cube->P6_3D.y; i++, j++)
                {
                        t1.y = i;
                        //For x-y slope
                        float slope1 = ((cube->P3_3D.y - cube->P2_3D.y) / (cube->P3_3D.x - cube->P2_3D.x));
                        t1.x = ((t1.y - cube->P2_3D.y) / slope1) + cube->P2_3D.x;
                        //t1.x = cube->P2_3D.x;
                        t1.z = (((cube->P3_3D.z - cube->P2_3D.z) / (cube->P3_3D.y - cube->P2_3D.y)) * (t1.y - cube->P2_3D.y)) + cube->P2_3D.z;
                        //printf("Value of P1: (%f, %f, %f)\n", t1.x, t1.y, t1.z);

                        t2.y = j;
                        t2.x = t1.x;
                        t2.z = (((cube->P6_3D.z - cube->P5_3D.z) / (cube->P6_3D.y - cube->P5_3D.y)) * (t2.y - cube->P5_3D.y)) + cube->P5_3D.z;
                        //printf("Value of P2: (%f, %f, %f)\n\n", t2.x, t2.y, t2.z);

                        m1 = TRANFER_PIPELINE_3D_TO_2D(t1.x, t1.y, t1.z);
                        m2 = TRANFER_PIPELINE_3D_TO_2D(t2.x, t2.y, t2.z);
                        drawline(m1.x, m1.y, m2.x, m2.y, MAGENTA);
                }
}


// To draw a 3D cube
void _3D_CUBES(_3d_cube *cube, float size_of_cube, float x_base, float y_base, float z_base)
{
        //Compute eight vertices for the cube

                //(0, 0, 60)
        cube->P1 = TRANFER_PIPELINE_3D_TO_2D(cube->P1_3D.x = x_base, cube->P1_3D.y = y_base, cube->P1_3D.z = (size_of_cube + z_base));
        //printf("Native values of P1: (%d, %d)\n", cube->P1.x, cube->P1.y);


                //(60, 0, 60)
        cube->P2 = TRANFER_PIPELINE_3D_TO_2D(cube->P2_3D.x = (x_base + size_of_cube), cube->P2_3D.y = y_base, cube->P2_3D.z = (z_base + size_of_cube));
        //printf("Native values of P2: (%d, %d)\n", cube->P2.x, cube->P2.y);


                //(60, 60, 60)
        cube->P3 = TRANFER_PIPELINE_3D_TO_2D(cube->P3_3D.x = (x_base + size_of_cube), cube->P3_3D.y = (y_base + size_of_cube), cube->P3_3D.z = (z_base + size_of_cube));
        //printf("Native values of P3: (%d, %d)\n", cube->P3.x, cube->P3.y);


                //(0, 60, 60)
        cube->P4 = TRANFER_PIPELINE_3D_TO_2D(cube->P4_3D.x = x_base, cube->P4_3D.y = (y_base + size_of_cube), cube->P4_3D.z = (z_base + size_of_cube));
        //printf("Native values of P4: (%d, %d)\n", cube->P4.x, cube->P4.y);


                //(60, 0, 0)
        cube->P5 = TRANFER_PIPELINE_3D_TO_2D(cube->P5_3D.x = (x_base + size_of_cube), cube->P5_3D.y = y_base , cube->P5_3D.z = z_base);
        //printf("Native values of P5: (%d, %d)\n", cube->P5.x, cube->P5.y);


                //(60, 60, 0)
        cube->P6 = TRANFER_PIPELINE_3D_TO_2D(cube->P6_3D.x = (x_base + size_of_cube), cube->P6_3D.y = (y_base + size_of_cube), cube->P6_3D.z = z_base );
```

```
	//printf("Native values of P6: (%d, %d)\n", cube->P6.x, cube->P6.y);


			//(0, 60, 0)
	cube->P7 = TRANFER_PIPELINE_3D_TO_2D(cube->P7_3D.x = x_base, cube->P7_3D.y = (y_base + size_of_cube), cube->P7_3D.z = z_base);
	//printf("Native values of P7: (%d, %d)\n", cube->P7.x, cube->P7.y);

	//(0, 0, 0)
	cube->P8 = TRANFER_PIPELINE_3D_TO_2D(cube->P8_3D.x = x_base, cube->P8_3D.y = y_base, cube->P8_3D.z = z_base);
	//printf("Native values of P8: (%d, %d)\n", cube->P8.x, cube->P8.y);

	//Join edges to form a cube
	drawline(cube->P1.x, cube->P1.y, cube->P2.x, cube->P2.y, BLACK);
	drawline(cube->P2.x, cube->P2.y, cube->P3.x, cube->P3.y, BLACK);
	drawline(cube->P3.x, cube->P3.y, cube->P4.x, cube->P4.y, BLACK);
	drawline(cube->P4.x, cube->P4.y, cube->P1.x, cube->P1.y, BLACK);
	drawline(cube->P2.x, cube->P2.y, cube->P5.x, cube->P5.y, BLACK);
	drawline(cube->P5.x, cube->P5.y, cube->P6.x, cube->P6.y, BLACK);
	drawline(cube->P6.x, cube->P6.y, cube->P3.x, cube->P3.y, BLACK);
	drawline(cube->P6.x, cube->P6.y, cube->P7.x, cube->P7.y, BLACK);
	drawline(cube->P7.x, cube->P7.y, cube->P4.x, cube->P4.y, BLACK);
	drawline(cube->P7.x, cube->P7.y, cube->P8.x, cube->P8.y, BLACK);
	drawline(cube->P1.x, cube->P1.y, cube->P8.x, cube->P8.y, BLACK);
	drawline(cube->P5.x, cube->P5.y, cube->P8.x, cube->P8.y, BLACK);

}

void drawcube2(_3d_cube *cube, float size_of_cube, float x_base, float y_base, float z_base)
{
	//Compute eight vertices for the cube

			//(0, 0, 60)
	cube->P1 = TRANFER_PIPELINE_3D_TO_2D(cube->P1_3D.x , cube->P1_3D.y , cube->P1_3D.z + (size_of_cube ));
	//printf("Native values of P1: (%d, %d)\n", cube->P1.x, cube->P1.y);


			//(60, 0, 60)
	cube->P2 = TRANFER_PIPELINE_3D_TO_2D(cube->P2_3D.x +( size_of_cube), cube->P2_3D.y, cube->P2_3D.z + (size_of_cube));
	//printf("Native values of P2: (%d, %d)\n", cube->P2.x, cube->P2.y);


			//(60, 60, 60)
	cube->P3 = TRANFER_PIPELINE_3D_TO_2D(cube->P3_3D.x + (size_of_cube), cube->P3_3D.y + (size_of_cube), cube->P3_3D.z + (size_of_cube));
	//printf("Native values of P3: (%d, %d)\n", cube->P3.x, cube->P3.y);


			//(0, 60, 60)
	cube->P4 = TRANFER_PIPELINE_3D_TO_2D(cube->P4_3D.x, cube->P4_3D.y + (size_of_cube), cube->P4_3D.z + (size_of_cube));
	//printf("Native values of P4: (%d, %d)\n", cube->P4.x, cube->P4.y);


			//(60, 0, 0)
	cube->P5 = TRANFER_PIPELINE_3D_TO_2D(cube->P5_3D.x + (size_of_cube), cube->P5_3D.y  , cube->P5_3D.z );
	//printf("Native values of P5: (%d, %d)\n", cube->P5.x, cube->P5.y);


			//(60, 60, 0)
	cube->P6 = TRANFER_PIPELINE_3D_TO_2D(cube->P6_3D.x + (size_of_cube), cube->P6_3D.y + ( size_of_cube), cube->P6_3D.z );
	//printf("Native values of P6: (%d, %d)\n", cube->P6.x, cube->P6.y);


			//(0, 60, 0)
	cube->P7 = TRANFER_PIPELINE_3D_TO_2D(cube->P7_3D.x, cube->P7_3D.y + ( size_of_cube), cube->P7_3D.z);
```

```
	//printf("Native values of P7: (%d, %d)\n", cube->P7.x, cube->P7.y);

	//(0, 0, 0)
	cube->P8 = TRANFER_PIPELINE_3D_TO_2D(cube->P8_3D.x , cube->P8_3D.y , cube->P8_3D.z );
	//printf("Native values of P8: (%d, %d)\n", cube->P8.x, cube->P8.y);

	//Join edges to form a cube
	drawline(cube->P1.x, cube->P1.y, cube->P2.x, cube->P2.y, BLACK);
	drawline(cube->P2.x, cube->P2.y, cube->P3.x, cube->P3.y, BLACK);
	drawline(cube->P3.x, cube->P3.y, cube->P4.x, cube->P4.y, BLACK);
	drawline(cube->P4.x, cube->P4.y, cube->P1.x, cube->P1.y, BLACK);
	drawline(cube->P2.x, cube->P2.y, cube->P5.x, cube->P5.y, BLACK);
	drawline(cube->P5.x, cube->P5.y, cube->P6.x, cube->P6.y, BLACK);
	drawline(cube->P6.x, cube->P6.y, cube->P3.x, cube->P3.y, BLACK);
	drawline(cube->P6.x, cube->P6.y, cube->P7.x, cube->P7.y, BLACK);
	drawline(cube->P7.x, cube->P7.y, cube->P4.x, cube->P4.y, BLACK);
	drawline(cube->P7.x, cube->P7.y, cube->P8.x, cube->P8.y, BLACK);
	drawline(cube->P1.x, cube->P1.y, cube->P8.x, cube->P8.y, BLACK);
	drawline(cube->P5.x, cube->P5.y, cube->P8.x, cube->P8.y, BLACK);

}

void LIGHT_SOURCE_POINT(_3d_cube *cube, float xs, float xy, float xz)
{
	_2d_coordi point_source; float lambda = 2.2;

	//find lamda
	//n = (0, 0, 1), a = (0, 0, 0) ---> lamda = n* (a-Ps)/ n * (Ps - Pi)
	float numer = (0 + 0 + 1*(0 - xz)), denom = (0 + 0 + 1*(xz - cube->P2_3D.z));
	lambda = numer / denom;


	point_source = TRANFER_PIPELINE_3D_TO_2D(xs, xy, xz);

	//Line equation for vertices P2, Point source and shadow point
	cube->Sh1_3D.x = xs + (lambda * (xs - cube->P2_3D.x));
	cube->Sh1_3D.y = xy + (lambda * (xy - cube->P2_3D.y));
	cube->Sh1_3D.z = xz + (lambda * (xz - cube->P2_3D.z));
	cube->Sh1 = TRANFER_PIPELINE_3D_TO_2D(cube->Sh1_3D.x, cube->Sh1_3D.y, cube->Sh1_3D.z);


	//Line equation for vertices P3, Point source and shadow point
	cube->Sh2_3D.x = xs + (lambda * (xs - cube->P3_3D.x ));
	cube->Sh2_3D.y = xy + (lambda * (xy - cube->P3_3D.y));
	cube->Sh2_3D.z = xz + (lambda * (xz - cube->P3_3D.z));
	cube->Sh2 = TRANFER_PIPELINE_3D_TO_2D(cube->Sh2_3D.x, cube->Sh2_3D.y, cube->Sh2_3D.z);

	//Line equation for vertices P4, Point source and shadow point
	cube->Sh3_3D.x = xs + (lambda * (xs - cube->P4_3D.x ));
	cube->Sh3_3D.y = xy + (lambda * (xy - cube->P4_3D.y));
	cube->Sh3_3D.z = xz + (lambda * (xz - cube->P4_3D.z));
	cube->Sh3 = TRANFER_PIPELINE_3D_TO_2D(cube->Sh3_3D.x, cube->Sh3_3D.y, cube->Sh3_3D.z);


	//Line equation for vertices P1, Point source and shadow point
	cube->Sh4_3D.x = xs + (lambda * (xs - cube->P1_3D.x));
	cube->Sh4_3D.y = xy + (lambda * (xy - cube->P1_3D.y));
	cube->Sh4_3D.z = xz + (lambda * (xz - cube->P1_3D.z));
	cube->Sh4 = TRANFER_PIPELINE_3D_TO_2D(cube->Sh4_3D.x, cube->Sh4_3D.y, cube->Sh4_3D.z);

}

void SHADOW_CUBE(_3d_cube *cube, int size_of_cube)
{
	int x, y, x2, y2;
	_2d_coordi temp1, temp2;
	float i,j;
```

```
	_3d_coordi t1, t2;
	_2d_coordi m1, m2;

	//Join shadow coordinates
	drawline(cube->Sh1.x, cube->Sh1.y, cube->Sh2.x, cube->Sh2.y, BLACK);
	drawline(cube->Sh2.x, cube->Sh2.y, cube->Sh3.x, cube->Sh3.y, BLACK);
	drawline(cube->Sh3.x, cube->Sh3.y, cube->Sh4.x, cube->Sh4.y, BLACK);
	drawline(cube->Sh4.x, cube->Sh4.y, cube->Sh1.x, cube->Sh1.y, BLACK);

	//Color the shadow
	for (i = cube->Sh1_3D.x, j = cube->Sh2_3D.x; i > cube->Sh4_3D.x && j > cube->Sh3_3D.x; i--, j--)
	{
		t1.x = i;
		t1.y = (((cube->Sh4_3D.y - cube->Sh1_3D.y)/ (cube->Sh4_3D.x - cube->Sh1_3D.x)) * (t1.x - cube->Sh1_3D.x)) + cube->Sh1_3D.y;
		t1.z = cube->Sh1_3D.z;

		t2.x = j;
		t2.y = (((cube->Sh3_3D.y - cube->Sh2_3D.y)/ (cube->Sh3_3D.x - cube->Sh2_3D.x)) * (t1.x - cube->Sh2_3D.x)) + cube->Sh2_3D.y;
		t2.z = t1.z;

		m1 = TRANFER_PIPELINE_3D_TO_2D(t1.x, t1.y, t1.z);
		m2 = TRANFER_PIPELINE_3D_TO_2D(t2.x, t2.y, t2.z);
		drawline(m1.x, m1.y, m2.x, m2.y, BLACK);
	}

	fillcube(cube);

	/*********************Draw Square and Tree Pattern*********************************/

	squarePattern(cube->P2.x+2, cube->P2.y+5, cube->P3.x-5, cube->P3.y-5, cube->P6.x-2, cube->P6.y-2, cube->P5.x+2, cube->P5.y+5);

	/*****************************Draw Square and Tree Pattern**********************/
	treePattern(cube->P3.x+10,cube->P3.y+10,5.23,4,4,GREEN);
	treePattern(cube->P3.x+10,cube->P3.y+10,4.18,4,4,GREEN);
	treePattern(cube->P3.x+10,cube->P3.y+10,4.71,4,4,GREEN);

	/*****************************Draw Initial*****************************************/

	NAME_INITIALS(cube, size_of_cube);
}


void _CUBE_ROTATION(_3d_cube *cube1, _3d_cube *cube2, float alpha, float theta_deg)
{
	float theta = (theta_deg * PI)/180.00;
	float add = 5;
	float cos_theta = cos(theta);
	float sin_theta = sin(theta);

	//Compute new rotated coordiantes for P1
		cube2->P1_3D.x = alpha * ((cube1->P1_3D.x * cos_theta) - (cube1->P1_3D.y * sin_theta) - (200* (cos_theta - sin_theta - 1))) + add;
		cube2->P1_3D.y = alpha * ((cube1->P1_3D.x * sin_theta) + (cube1->P1_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
		cube2->P1_3D.z = alpha * cube1->P1_3D.z;

		//Compute new rotated coordiantes for P2
		cube2->P2_3D.x = alpha * ((cube1->P2_3D.x * cos_theta) - (cube1->P2_3D.y * sin_theta) - (200 * (cos_theta - sin_theta - 1))) + add;
		cube2->P2_3D.y = alpha * ((cube1->P2_3D.x * sin_theta) + (cube1->P2_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
		cube2->P2_3D.z = alpha * cube1->P2_3D.z;

		//Compute new rotated coordiantes for P3
		cube2->P3_3D.x = alpha * ((cube1->P3_3D.x * cos_theta) - (cube1->P3_3D.y * sin_theta) - (200 * (cos_theta - sin_theta - 1))) + add;
		cube2->P3_3D.y = alpha * ((cube1->P3_3D.x * sin_theta) + (cube1->P3_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
		cube2->P3_3D.z = alpha * cube1->P3_3D.z;

		//Compute new rotated coordiantes for P4
		cube2->P4_3D.x = alpha * ((cube1->P4_3D.x * cos_theta) - (cube1->P4_3D.y * sin_theta) - (200 * (cos_theta - sin_theta - 1))) + add;
		cube2->P4_3D.y = alpha * ((cube1->P4_3D.x * sin_theta) + (cube1->P4_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
```

```
		cube2->P4_3D.z = alpha * cube1->P4_3D.z;

		//Compute new rotated coordiantes for P5
		cube2->P5_3D.x = alpha * ((cube1->P5_3D.x * cos_theta) - (cube1->P5_3D.y * sin_theta) - (200 * (cos_theta - sin_theta - 1))) + add;
		cube2->P5_3D.y = alpha * ((cube1->P5_3D.x * sin_theta) + (cube1->P5_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
		cube2->P5_3D.z = alpha * cube1->P5_3D.z;

		//Compute new rotated coordiantes for P6
		cube2->P6_3D.x = alpha * ((cube1->P6_3D.x * cos_theta) - (cube1->P6_3D.y * sin_theta) - (200 * (cos_theta - sin_theta - 1))) + add;
		cube2->P6_3D.y = alpha * ((cube1->P6_3D.x * sin_theta) + (cube1->P6_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
		cube2->P6_3D.z = alpha * cube1->P6_3D.z;

		//Compute new rotated coordiantes for P7
		cube2->P7_3D.x = alpha * ((cube1->P7_3D.x * cos_theta) - (cube1->P7_3D.y * sin_theta) - (200 * (cos_theta - sin_theta - 1))) + add;
		cube2->P7_3D.y = alpha * ((cube1->P7_3D.x * sin_theta) + (cube1->P7_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
		cube2->P7_3D.z = alpha * cube1->P7_3D.z;

		//Compute new rotated coordiantes for P8
		cube2->P8_3D.x = alpha * ((cube1->P8_3D.x * cos_theta) - (cube1->P8_3D.y * sin_theta) - (200 * (cos_theta - sin_theta - 1))) + add;
		cube2->P8_3D.y = alpha * ((cube1->P8_3D.x * sin_theta) + (cube1->P8_3D.y * cos_theta) - (200 * (sin_theta + cos_theta - 1))) + add;
		cube2->P8_3D.z = alpha * cube1->P8_3D.z;

}

void squarePattern(int x1, int y1, int x2,int y2,int x3,int y3,int x4,int y4)
{
	int p1_1=0, p1_2=0, p1_3=0, p1_4=0, p2_1=0,p2_2=0, p2_3=0, p2_4=0;
	float lambda=0.8;

	// Draw 10 levels of squares in each pattern
	for(int i=1;i<=5;i++){
		lcddelay(100);
		// Use the equation given in class to calculate the 4 vertices' coordinates of the recursive squares
		p1_1 = (x2+(lambda*(x1-x2)));
		p2_1 = (y2+(lambda*(y1-y2)));
		p1_2 = (x3+(lambda*(x2-x3)));
		p2_2 = (y3+(lambda*(y2-y3)));
		p1_3 = (x4+(lambda*(x3-x4)));
		p2_3 = (y4+(lambda*(y3-y4)));
		p1_4 = (x1+(lambda*(x4-x1)));
		p2_4 = (y1+(lambda*(y4-y1)));

		// Draw a square (4 drawline() for 4 sides)
		drawline(p1_1,p2_1,p1_2,p2_2,RED);
		drawline(p1_2,p2_2,p1_3,p2_3,RED);
		drawline(p1_4,p2_4,p1_3,p2_3,RED);
		drawline(p1_1,p2_1,p1_4,p2_4,RED);

		// Initiate the original vertices' values with the new calculated vertices' values
		x1 = p1_1;
		x2 = p1_2;
		x3 = p1_3;
		x4 = p1_4;

		y1 = p2_1;
		y2 = p2_2;
		y3 = p2_3;
		y4 = p2_4;
	}
}

void treePattern(int x, int y, float angle, int length, int level, int color)
{

	int x1,y1,len;
	float ang;

	if(level>0){
		// To calculate the x,y vertices for the branch after rotation
		x1 = x+length*cos(angle);
	   y1 = y+length*sin(angle);

	   // To draw the tree branch
	   drawline(x,y,x1,y1,color);

	   // Add 30 degree to angle to rotate it to right
	   ang = angle + 0.52;
	   // To calculate 80% of the line length
	   len = 0.8 * length;
```

```
            // Call drawTree2d function recursively to
draw tree pattern
            treePattern(x1,y1,ang,len,level-1,color);

            // Subtract 30 degree from the angle to rotate it
to right
            ang = angle - 0.52;
            len = 0.8 * length;

           treePattern(x1,y1,ang,len,level-1,color);

            // Draw the next level
            ang = angle;
            len = 0.8 * length;
            treePattern(x1,y1,ang,len,level-1,color);
          }
}

void NAME_INITIALS(_3d_cube *cube, float
size_of_cube)
{
          __initials name_initials_coordi;
          int x, y, x2, y2;
          _2d_coordi temp1, temp2;

          name_initials_coordi.A1_2D =
TRANFER_PIPELINE_3D_TO_2D(name_initials_coord
i.A1.x = cube->P1_3D.x + 10,name_initials_coordi.A1.y
= cube->P1_3D.y + 10, name_initials_coordi.A1.z =
size_of_cube);
          name_initials_coordi.A2_2D =
TRANFER_PIPELINE_3D_TO_2D(name_initials_coord
i.A2.x = cube->P1_3D.x + 10,name_initials_coordi.A2.y
= cube->P1_3D.y + 30 , name_initials_coordi.A2.z =
size_of_cube);
          name_initials_coordi.A3_2D =
TRANFER_PIPELINE_3D_TO_2D(name_initials_coord
i.A3.x = cube->P1_3D.x + 15,name_initials_coordi.A3.y
= cube->P1_3D.y + 30, name_initials_coordi.A3.z =
size_of_cube);
          name_initials_coordi.A4_2D =
TRANFER_PIPELINE_3D_TO_2D(name_initials_coord
i.A4.x = cube->P1_3D.x + 15,name_initials_coordi.A4.y
= cube->P1_3D.y + 12, name_initials_coordi.A4.z =
size_of_cube);
          name_initials_coordi.A5_2D =
TRANFER_PIPELINE_3D_TO_2D(name_initials_coord
i.A5.x = cube->P1_3D.x + 30,name_initials_coordi.A5.y
= cube->P1_3D.y + 12, name_initials_coordi.A5.z =
size_of_cube);
          name_initials_coordi.A6_2D =
TRANFER_PIPELINE_3D_TO_2D(name_initials_coord
i.A6.x = cube->P1_3D.x + 30,name_initials_coordi.A6.y
= cube->P1_3D.y + 10 , name_initials_coordi.A6.z =
size_of_cube);

          drawline(name_initials_coordi.A1_2D.x,
name_initials_coordi.A1_2D.y,
name_initials_coordi.A2_2D.x,
name_initials_coordi.A2_2D.y, RED);
          drawline(name_initials_coordi.A2_2D.x,
name_initials_coordi.A2_2D.y,
name_initials_coordi.A3_2D.x,
name_initials_coordi.A3_2D.y, RED);
          drawline(name_initials_coordi.A3_2D.x,
name_initials_coordi.A3_2D.y,
name_initials_coordi.A4_2D.x,
name_initials_coordi.A4_2D.y, RED);
          drawline(name_initials_coordi.A4_2D.x,
name_initials_coordi.A4_2D.y,
name_initials_coordi.A5_2D.x,
name_initials_coordi.A5_2D.y, RED);
          drawline(name_initials_coordi.A5_2D.x,
name_initials_coordi.A5_2D.y,
name_initials_coordi.A6_2D.x,
name_initials_coordi.A6_2D.y, RED);
          drawline(name_initials_coordi.A1_2D.x,
name_initials_coordi.A1_2D.y,
name_initials_coordi.A6_2D.x,
name_initials_coordi.A6_2D.y, RED);
}
```

Main.c

```
#include "LPC17xx.h"
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "DrawLine.h"
#include <LAB_2_3D_GE.h>

#define PORT_NUM          0
#define LOCATION_NUM      0

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


/*********************************************
*
 Main Function  main()
 *********************************************/
int main(void)
{
          // Axes
          int16_t _height = ST7735_TFTHEIGHT;
          int16_t _width = ST7735_TFTWIDTH;

          uint32_t i, portnum;
          portnum = PORT_NUM;
```

```
	/****************** initialize SSP port and LCD *************/
	if (portnum == 0)
		SSP0Init();
	else if (portnum == 1)
		SSP1Init();

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

	// To initialize the LCD
	lcd_init();
	//Refresh background with white color
	background(0, 0, _width, _height, WHITE);
	/****************** *******
*************************************/

	PARA_CALCULATION();

	// To draw the 3D world coordinate system
	_3D_AXIS();

	/**********************To draw a 3D cubes **********************/
	float cube1size = 40.00, x_base_1 = 0.00, y_base_1 = 0.00, z_base_1 = 00.00;
	float cube2size = 25.00, x_base_2 = 50.00, y_base_2 = 00.00, z_base_2 = 0.00;
	float cube3size = 30.00, x_base_3 = 00.00, y_base_3 = 50.00, z_base_3 = 0.00;

	_3d_cube cube1, cube2, cube3, temp1, temp2;


	_3D_CUBES(&cube1, cube1size, x_base_1, y_base_1, z_base_1);
	lcddelay(500);


	_3D_CUBES(&cube2, cube2size, x_base_2, y_base_2, z_base_2);
	lcddelay(500);


	_3D_CUBES(&cube3, cube3size, x_base_3, y_base_3, z_base_3);
	lcddelay(500);

	_CUBE_ROTATION(&cube2, &temp1, 0.7, 45.00);
	_CUBE_ROTATION(&cube3, &temp2, 0.9, -45.00);

	drawcube2(&temp1, cube2size, x_base_3, x_base_3, x_base_3);
	drawcube2(&temp2, cube3size, x_base_3, x_base_3, x_base_3);

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

	/**********************Light point source & Shadow **********************/

	float xs = 00.00, ys = 130.00, zs = 130.00;
	LIGHT_SOURCE_POINT(&cube1, xs, ys, zs);
	LIGHT_SOURCE_POINT(&temp1, xs, ys, zs);
	LIGHT_SOURCE_POINT(&temp2, xs, ys, zs);

	//Draw shadow
	SHADOW_CUBE(&cube1, cube1size);
	SHADOW_CUBE(&cube2, cube2size);
	SHADOW_CUBE(&cube3, cube3size);

	return 0;
}
```