3D Graphics Engine Implementing Diffuse Reflection using LPC1769

Akarsh Chandrashekar
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: akarsh.chandrashekar@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 solid cubes with decorations on three sides. The decorations are implemented as 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 data sheets, 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 the cubes with decorated patterns on three surfaces. The three visible sides of the cubes were colored and decorated. Shadows were generated for all of them and diffuse reflection was implemented on the top of one of the cubs. two side had tree forest while the top side had the alphabet 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 LPC modules along with an LCD display and a power circuit. GPIO testing circuit was also designed for testing purposes.
2. 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 randomized trees and initials on the surfaces.
6. Generating shadows of cube onto the x-y plane.
7. Implementing diffused reflection for one of the cubes.
8. 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 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 a tree-forest pattern using randomized locations was implemented. Generated solid decorated cube with it's 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

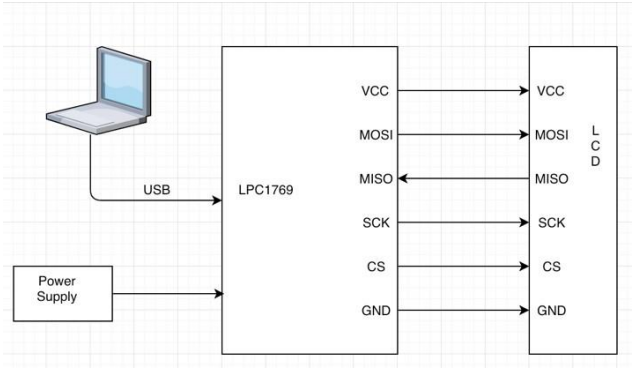

Figure 1 System Block Diagram

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.

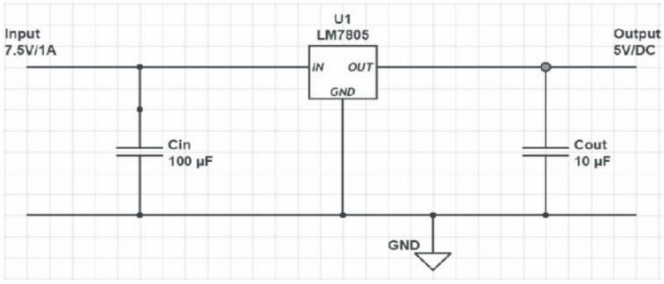

Figure 2 Power Circuit

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.

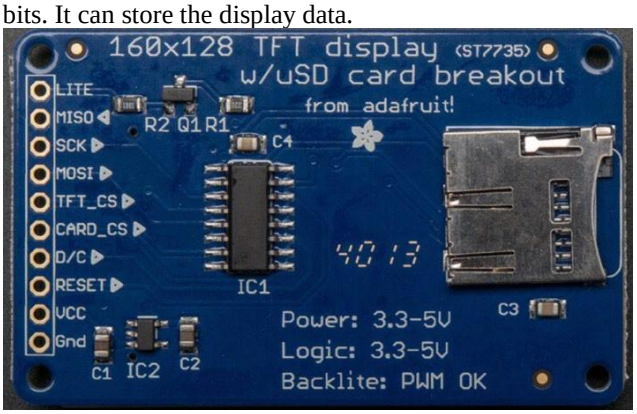

Figure 5: LCD ST7735R

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 Trees:

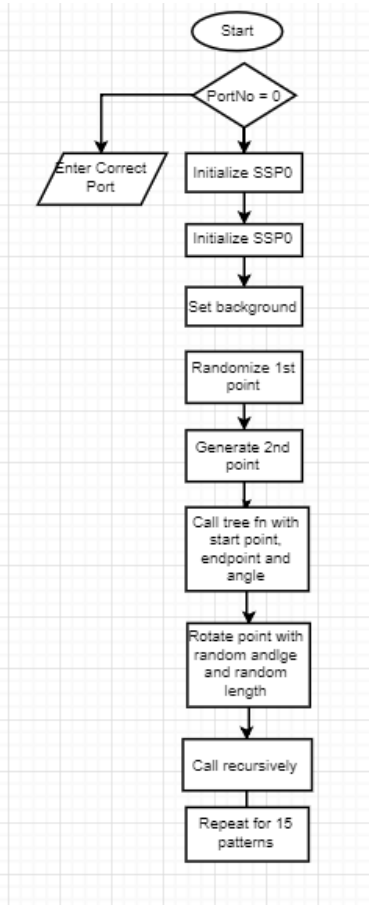

Figure 6: Flow Diagram for Trees

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.

$$\vec{P} = \vec{P}s + \lambda\left(\vec{P}s - \vec{P}i\right)$$

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.

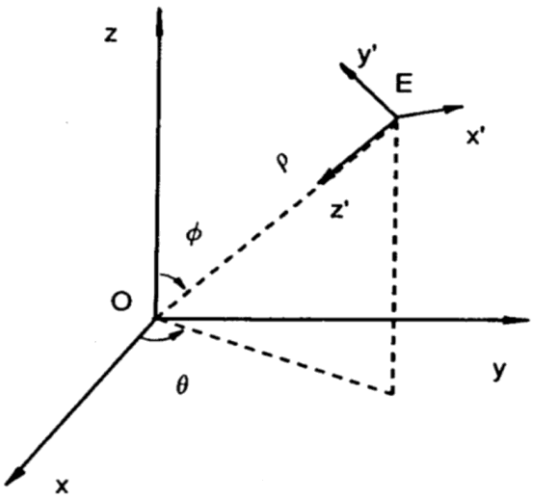

Figure 11: 3D-Coordinate system

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 3D solid cubes and produce it's shadows by using the 3D vector and plane intersection concept.
12. Draw trees and pattern 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, 0xBFEFFF);

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

```
draw_cube(start_pnt,size);
```

9. void Draw_diffusion_latest(int pt, int cube_size, int elev, struct Point_3D pl_source);

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 void draw3Dcoordinate()

12. Function to draw cube void draw_cube(int start_pnt, int size)

void draw_cube(int srt_pnt, int size)

13. Implementing Tree Pattern
void drawTrees_on_cube_surface(uint32_t color,int start_pnt, int size, char plane)

14. 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)

15. Drawing the Initial X
void draw_X (start_pnt, size)

16. Implementing diffuse reflection void Draw_diffusion_XZ(int pt, int cube_size, int elev, struct Point_3D pl_source)

17. Drawing shadow void draw_shadow(double start_pnt, double size, struct Point_3D ps)

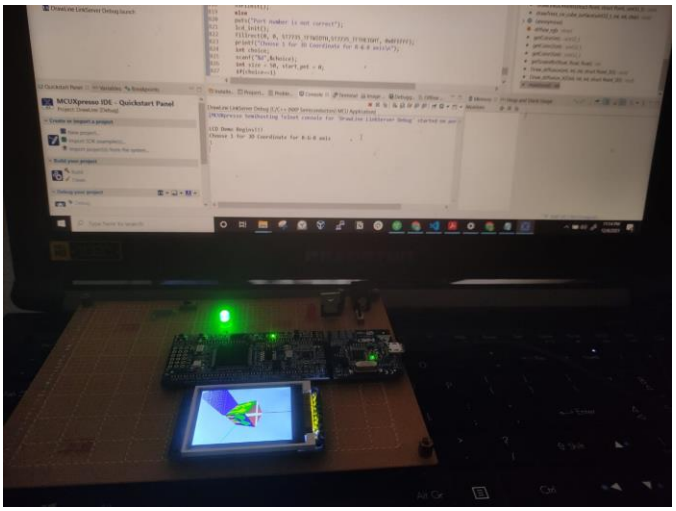

Figure 11: System setup

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

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.
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.

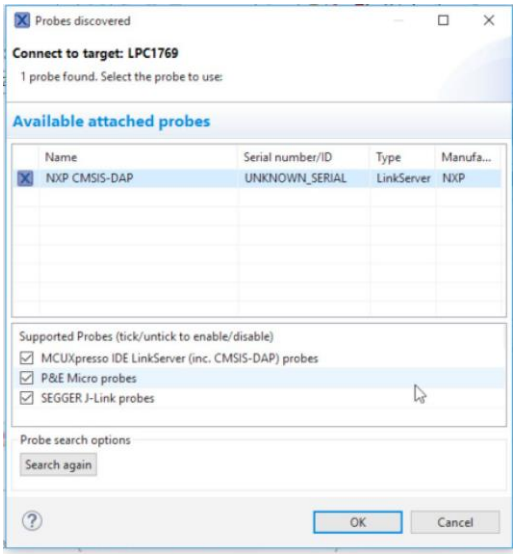

Figure 14: Probe detection

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.

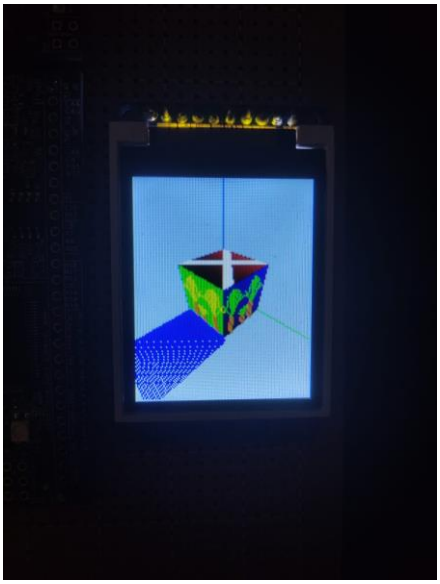

Figure 15: Diffuse Reflection and Shadow

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

```
/*
 * project_final
 *
 * Author: Akarsh Chandrashekar
 */

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



#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF
#define BROWN 0x993300
#define DARKGREEN 0x1A9900
#define GOLD 0xFFD700
#define SILVER 0xD3D3D3
#define ORANGE 0xD3D3D3
#define ORANGERED 0xFF4500
#define SEAGREEN 0x2E8B57
#define YELLOW 0xFFFF00
#define FOREST 0x228B22
#define SKYBLUE 0x87CEEB
#define MAROON 0x800000
#define GREEN1 0xDFFF00
#define GREEN2 0x7CFC00
#define GREEN3 0x4CBB17



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

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

#define PORT_NUM            0


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

```
#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159

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

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

#define pi 3.14
#define alpha pi/6

#define UpperBD 51
#define      Kdr    0.8
#define      Kdg    0.0
#define      Kdb    0.0

#define      display_scaling            150000.0
#define      display_shifting    0.45



#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF
#define BROWN 0x993300
#define DARKGREEN 0x1A9900
#define GOLD 0xFFD700
#define SILVER 0xD3D3D3
#define ORANGE 0xD3D3D3
#define ORANGERED 0xFF4500
#define SEAGREEN 0x2E8B57
#define YELLOW 0xFFFF00
#define FOREST 0x228B22
#define SKYBLUE 0x87CEEB
#define MAROON 0x800000
#define GREEN1 0xDFFF00
#define GREEN2 0x7CFC00
#define GREEN3 0x4CBB17



int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

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 drawPixel(int16_t x, int16_t y, uint32_t
color)
{
        if ((x < 0) || (x >= _width) || (y < 0)
|| (y >= _height))
        return;
        setAddrWindow(x, y, x + 1, y + 1);
        writecommand(ST7735_RAMWR);
        write888(color, 1);
}

void drawLine(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) {
			drawPixel(y0, x0,
color);
		}
		else {
			drawPixel(x0, y0,
color);
		}
		err -= dy;
		if (err < 0) {
			y0 += ystep;
			err += dx;
		}
	}
}


////////////////////////////////////////
////////////////////////////////////////
////

//Define 2d Point struct
struct Point
{
	int16_t x;
	int16_t y;
};

////////////////////////////////////////
////////////////////////////////////////
///////
///Define 3d Point structure
struct Point_3D
 {
	float x;
	float y;
	float z;
 };
```

```
int v2p_x1(int16_t x)
{
	return x+_width/2;

}
int v2p_y1(int16_t y)
{
	return -y+_height/2;

}
//define e location
 float_t xe = 100;
 float_t ye = 100;
 float_t ze = 100;


 // To convert world to viewer coordinates
 struct Point Transformation_pipeline (struct
Point_3D world)
 {
	struct Point perspective;
	struct Point_3D viewer;

	//define distance
	float_t Rho
=sqrt(pow(xe,2)+pow(ye,2)+pow(ze,2));
	float_t D_focal = 100;	//D value
is supposed to between (10-20)

	//sinThetha and cosTheta
	double sintheta = ye /
sqrt(pow(xe,2)+pow(ye,2));
	double costheta = xe /
sqrt(pow(xe,2)+pow(ye,2));

	//second angle
	double sinphi =
sqrt(pow(xe,2)+pow(ye,2))/Rho;
	double cosphi = ze / Rho;

	viewer.x = (-
sintheta*world.x)+(costheta*world.y);
	viewer.y = (-costheta*cosphi*world.x)-
(cosphi*sintheta*world.y)+(sinphi*world.z);
	viewer.z = (-sinphi*costheta*world.x)-
(sinphi*costheta*world.y)-
(costheta*world.z)+Rho;
	perspective.x =
D_focal*viewer.x/viewer.z;
	perspective.y =
D_focal*viewer.y/viewer.z;

	perspective.x = v2p_x1(perspective.x);
	perspective.y = v2p_y1(perspective.y);

	return perspective;
 }

 // To draw the world 3D coordinate system -
xw,yw,zw
 void draw3Dcoordinate()
```

```
 {
        struct Point axis;
        int x1,y1,x2,y2,x3,y3,x4,y4;
        struct Point_3D org ={0.0,0.0,0.0};
        struct Point_3D x_ax
={0.0,0.0,0.0};
        struct Point_3D y_ax
={0.0,0.0,0.0};
        struct Point_3D z_ax
={0.0,0.0,0.0};
        axis =
Transformation_pipeline(org);

        //x axis Red
        x1=axis.x;
        y1=axis.y;
        x_ax.x = 180.0;
        axis =
Transformation_pipeline(x_ax);
        x2=axis.x;
        y2=axis.y;
        drawLine(x1,y1,x2,y2,RED);

        //y axis Green
        y_ax.y = 180.0;
        axis =
Transformation_pipeline(y_ax);
        x3=axis.x;
        y3=axis.y;
        drawLine(x1,y1,x3,y3,GREEN);

         //z axis Blue
        z_ax.z =180.0;
        axis =
Transformation_pipeline(z_ax);
        x4=axis.x;
        y4=axis.y;
        drawLine(x1,y1,x4,y4,BLUE);

 }

////////////////////////////////////////////
//////////////////////////
 void draw_cube(int start_pnt, int size)
 {
       struct Point lcd;
       int x_p[8], y_p[8];
       double x[8] =
{start_pnt,(start_pnt+size),(start_pnt+size
),start_pnt,start_pnt,(start_pnt+size),(sta
rt_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)};
       for(int i=0; i<8; i++){
             struct Point_3D
a={x[i],y[i],z[i]};
             lcd = Transformation_pipeline(a);
             x_p[i] = lcd.x;
             y_p[i] = lcd.y;
       }

       drawLine(x_p[0], y_p[0], x_p[1],
y_p[1],BLACK);
       drawLine(x_p[1], y_p[1], x_p[2],
y_p[2],BLACK);
       drawLine(x_p[2], y_p[2], x_p[3],
y_p[3],BLACK);
       drawLine(x_p[3], y_p[3], x_p[0],
y_p[0],BLACK);

       drawLine(x_p[4], y_p[4], x_p[5],
y_p[5],BLACK);
       drawLine(x_p[5], y_p[5], x_p[6],
y_p[6],BLACK);
       drawLine(x_p[6], y_p[6], x_p[7],
y_p[7],BLACK);
       drawLine(x_p[7], y_p[7], x_p[4],
y_p[4],BLACK);

       drawLine(x_p[0], y_p[0], x_p[4],
y_p[4],BLACK);
       drawLine(x_p[5], y_p[5], x_p[1],
y_p[1],BLACK);
       drawLine(x_p[7], y_p[7], x_p[3],
y_p[3],BLACK);
       drawLine(x_p[6], y_p[6], x_p[2],
y_p[2],BLACK);

 }

////////////////////////////////////////////
//////////////////////////////////////////
 void fill_cube(int start_pnt,int size)
 {
       struct Point s1;

       int i,j;
       size=size+start_pnt;

       for(i=0;i<size;i++)
       {
             for(j=0;j<size;j++)
             {
                   struct Point_3D a=
{j,i,size};
                   struct Point_3D b=
{i,size,j};
                   struct Point_3D c=
{size,j,i};

       s1=Transformation_pipeline(a);   //top
fill green
                   drawPixel(s1.x,s1.y,RED);

       s1=Transformation_pipeline(b);   // right
fill yellow
```

```
	drawPixel(s1.x,s1.y,BLUE);

	s1=Transformation_pipeline(c);   //
left fill pink

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

 void draw_shadow(double start_pnt, double
size, struct Point_3D ps)
  {
	int i,j;
	int xShad[4]={0}, yShad[4]={0},
zShad[4]={0};
	struct Point s[4];
	double x[4] = {start_pnt,
start_pnt+size, start_pnt+size, start_pnt};
	double y[4] = {start_pnt, start_pnt,
start_pnt+size, start_pnt+size};
	double z[4]=
{start_pnt+size,start_pnt+size,start_pnt+si
ze,start_pnt+size};
	for(i=0; i<4; i++)
	{

		xShad[i] = ps.x + (-
ps.z/(z[i]-ps.z))*(x[i]-ps.x);
		yShad[i] = ps.y + (-
ps.z/(z[i]-ps.z))*(y[i]-ps.y);
		zShad[i] = ps.z + (-
ps.z/(z[i]-ps.z))*(z[i]-ps.z);
	}

	struct Point_3D shades[4];
	for(int i=0; i<4; i++){
		shades[i].x = xShad[i];
		shades[i].y = yShad[i];
		shades[i].z = zShad[i];
		s[i] = Transformation_pipeline
(shades[i]);
	}
	for(int i=size;i<shades[2].x;i++)
	{

		for(int j=0;j<shades[2].y;j++)
		{
			struct Point_3D
a= {i,j,0};

			struct Point
s1=Transformation_pipeline(a);   //top fill
green

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

////////////////////////////////////////////
///////////////

 void draw_X(int start_pnt, int size)
  {
	struct Point p1;
	int i,j;
	size=size+start_pnt;
	int map[size][size];

	for(i = start_pnt; i < size/2;i++)
		{
			for(j = start_pnt; j <
size;j++)
			{
			map[i][j]=0;
			}
		}

	for(i = start_pnt; i < size;i++)
		{
			for(j = start_pnt; j <
size;j++)
			{
				if(i==j ||
i==size-j-1)

	map[i][j]=1;
				else
if((j+1<size && i==j+1) || (j+1<size && i==size-
j-2))

	map[i][j]=1;
				else
if((j+2<size && i==j+2) || (j+2<size && i==size-
j-3))

	map[i][j]=1;
				else
if((j+3<size && i==j+3) || (j+3<size && i==size-
j-4))

	map[i][j]=1;
				else
if((j+4<size && i==j+4) || (j+4<size && i==size-
j-5))

	map[i][j]=1;
				else
if((j+5<size && i==j+5) || (j+5<size && i==size-
j-6))

	map[i][j]=1;
				else
if((j+6<size && i==j+6) || (j+6<size && i==size-
j-7))

	map[i][j]=1;
				else
```

```
if((j+7<size && i==j+7) || (j+7<size &&
i==size-j-8))

	map[i][j]=1;
                                    else
if((j+8<size && i==j+8) || (j+8<size &&
i==size-j-9))

	map[i][j]=1;
                                    else
if((j+9<size && i==j+9) || (j+9<size &&
i==size-j-10))

	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)
			{
				struct Point_3D
a= {j,i,size};
				p1 =
Transformation_pipeline(a);

	drawPixel(p1.x,p1.y,WHITE);
                        }
                        else if(map[i][j]==0)
                        {
                                struct Point_3D
a= {j,i,size};
                                p1 =
Transformation_pipeline(a);
                        }
                }
        }
   }



//////////////////////////////////////////
//////////////////////

struct Point drawbranch(int x1, int y1, int
size, char plane){
        if(plane == 'y') {
        struct Point_3D d={y1, size, x1};
        return Transformation_pipeline (d);
        }
        struct Point_3D d={size, y1, x1};
    return Transformation_pipeline (d);
 }

 void drawLine2DPoints(struct Point p1,
struct Point p2, uint32_t color){
        drawLine(p1.x, p1.y, p2.x, p2.y, color);
 }

 //void draw_tree_for_cube(uint32_t color,int
start_pnt, int size, char plane)
 void drawTrees_on_cube_surface(uint32_t
color,int start_pnt, int size, char plane)
 {
        int i=0;
        float sin30=0.5;
        float cos30=0.866;
        float d=0.134;
        int
tree_branch[3][3]={{start_pnt,start_pnt+10,0.5*s
ize},

        {start_pnt,start_pnt+20,0.3*size},

        {start_pnt,start_pnt+20,0.8*size}};
        while(i<3)
        {
                int x0, y0, y1, x1;
                x0=tree_branch[i][0];
                x1=tree_branch[i][1];
                y0=tree_branch[i][2];
                y1=y0;
                i++;

                struct Point arr[2];
                arr[0] = drawbranch(x0, y0, size,
plane);
                arr[1] = drawbranch(x1, y1, size,
plane);
                for (int j = -1; j < 2; j++)
                {
                        drawLine(arr[0].x+j,
arr[0].y+j, arr[1].x+j, arr[1].y+j, 0x00FF8000);
                }

                for(int it=0;it<7;it++)
                {
                        struct Point left[4];
                        struct Point middle[4];
                        struct Point right[4];
                        middle[0] = arr[1];
                        int16_t x2=(0.6*(x1-x0))+x1;
                        int16_t y2=y1;
                        middle[1] = drawbranch(x2,
y2, size, plane);
                        int16_t xr=
((d*x1)+(cos30*x2)-(sin30*y2)+(sin30*y1));
                        int16_t yr=((sin30*x2)-
(sin30*x1)+(cos30*y2)-(cos30*y1)+y1);
                        middle[2] = drawbranch(xr,
yr, size, plane);
                        int16_t
xl=((d*x1)+(cos30*x2)+(sin30*y2)-(sin30*y1));
                        int16_t yl=((sin30*x1)-
(sin30*x2)+(d*y2)+(cos30*y1));
                        middle[3] = drawbranch(xl,
```

```
yl, size, plane);
                    // Right subtree
building
                    int16_t xrLen =
sqrt(pow((xr-x1),2)+pow((yr-y1),2)) ;
                    int16_t xrImag=
(0.8*xrLen)+xr;
                    int16_t xr1 =
((d*xr)+(cos30*xrImag)-
(sin30*yr)+(sin30*yr));
                    int16_t yr1 =
((sin30*xrImag)-(sin30*xr)+(cos30*yr)-
(cos30*yr)+yr);
                    right[0] = middle[2];
                    right[1] =
drawbranch(xr1, yr1,size, plane);
                    int16_t
xrr,xrl,yrr,yrl;
                    xrr =
((d*xr)+(cos30*xr1)-
(sin30*yr1)+(sin30*yr));
                    yrr = ((sin30*xr1)-
(sin30*xr)+(cos30*yr1)-(cos30*yr)+yr);
                    right[2] =
drawbranch(xrr, yrr,size, plane);
                    xrl =
((d*xr)+(cos30*xr1)+(sin30*yr1)-
(sin30*yr));
                    yrl = ((sin30*xr)-
(sin30*xr1)+(d*yr)+(cos30*yr1));
                    right[3] =
drawbranch(xrl, yrl,size, plane);
                    // Left Subtree
building
                    left[0] = middle[3];
                    int16_t xlImag=
(0.8*xrLen)+xl;
                    int16_t xl1 =
((d*xl)+(cos30*xlImag)+(sin30*yl)-
(sin30*yl));
                    int16_t yl1 =
((sin30*xl)-
(sin30*xlImag)+(d*yl)+(cos30*yl));
                    left[1] =
drawbranch(xl1, yl1,size, plane);
                    int16_t
xlr,xll,ylr,yll;
                    xlr =
((d*xl)+(cos30*xl1)-
(sin30*yl1)+(sin30*yl));
                    ylr = ((sin30*xl1)-
(sin30*xl)+(cos30*yl1)-(cos30*yl)+yl);
                    left[2] =
drawbranch(xlr, ylr,size, plane);
                    xll =
((d*xl)+(cos30*xl1)+(sin30*yl1)-
(sin30*yl));
                    yll = ((sin30*xl)-
(sin30*xl1)+(d*yl)+(cos30*yl1));
                    left[3] =
drawbranch(xll, yll,size, plane);
                    // Drawing all the branches.
                    for(int i1=0; i1<4; i1++){

      drawLine2DPoints(middle[0], middle[i1],
color);

      drawLine2DPoints(right[0], right[i1],
color);

      drawLine2DPoints(left[0], left[i1],
color);
                    }
                    x0=x1;
                    x1=x2;
             }
      }
 }
////////////////////////////////////////////
////////////////////////////////////////////
typedef struct {
      float r,g,b;
} diffuse_rgb;

uint32_t getColor(int Kr){
      uint32_t color = 0x000000;
      Kr |= 0x000000;
      color |= Kr << 16;
      color |= (0x000000) << 8;
      color |= (0x000000);
      return color;
}

uint32_t getColor2(int Kb){
      uint32_t color = 0x000000;
      Kb |= 0x000000;
      color |= (0x000000) << 16;
      color |= (0x000000) << 8;
      color |= Kb;
      return color;
}

uint32_t getColor3(int Kg){
      uint32_t color = 0x000000;
      Kg |= 0x000000;
      color |= (0x000000) << 16;
      color |= Kg << 8;
      color |= (0x000000);
      return color;
}

int getScaledKr(float rmax, float rmin, float
kr) {

float k = (255.0-20.0)/(rmax-rmin);
float c = 20.0-(k*rmin);
return (int)(k*kr+c);
}

void Draw_diffusion(int pt, int cube_size, int
elev, struct Point_3D pl_source)
{
```

```
        struct Point_3D pt1 =
{pt,pt,(cube_size+pt+elev)};
        struct Point_3D pt2 =
{(cube_size+pt),(cube_size+pt),(cube_size+p
t+elev)};
        float diffuse_rmin, diffuse_rmax;
        float temp_distance1 = pow((pt1.x-
pl_source.x),2)+

pow((pt1.y- pl_source.y),2)+

pow((pt1.z- pl_source.z),2);
        float temp_distance2 = pow((pt2.x-
pl_source.x),2)+

pow((pt2.y- pl_source.y),2)+

pow((pt2.z- pl_source.z),2);
        diffuse_rmax =  (Kdr * abs(pt1.z-
pl_source.z)) /
(temp_distance1*sqrt(temp_distance1));
        diffuse_rmin =  (Kdr * abs(pt2.z-
pl_source.z)) /
(temp_distance2*sqrt(temp_distance2));
        diffuse_rgb  diffuse;
        for(int i=0; i<cube_size; i++) {
              for(int j=0; j<cube_size;
j++) {
                    struct Point_3D pt =
{i, j, cube_size+elev};
                    float distance =
pow((pt.x- pl_source.x),2)+

         pow((pt.y- pl_source.y),2)+

         pow((pt.z- pl_source.z),2);
                    diffuse.r = (Kdr *
abs(pt.z-pl_source.z)) /
(distance*sqrt(distance));
                    diffuse.g = (Kdg *
abs(pt.z-pl_source.z)) / distance;
                    diffuse.b = (Kdb *
abs(pt.z-pl_source.z)) / distance;
                    int scaled_kr =
getScaledKr(diffuse_rmax, diffuse_rmin,
diffuse.r);
                    struct Point lcd =
Transformation_pipeline(pt);
                    drawPixel(lcd.x,
lcd.y, getColor(scaled_kr));
              }
        }

}

void Draw_diffusion_XZ(int pt, int
cube_size, int elev, struct Point_3D
pl_source)
{
        struct Point_3D pt1 =
{pt,(cube_size+pt+elev),(cube_size+pt+elev)
};
        struct Point_3D pt2 =
{(cube_size+pt),(cube_size+pt),pt+elev};
        float diffuse_rmin, diffuse_rmax;
        float temp_distance1 = pow((pt1.x-
pl_source.x),2)+
                                    pow((pt1.y-
pl_source.y),2)+
                                    pow((pt1.z-
pl_source.z),2);
        float temp_distance2 = pow((pt2.x-
pl_source.x),2)+
                                    pow((pt2.y-
pl_source.y),2)+
                                    pow((pt2.z-
pl_source.z),2);
        float Kdb2 = 0.8;
        float Kdr2 = 0.0;
        float Kdg2 = 0.0;
        diffuse_rmax =  (Kdb2 * abs(pt1.y-
pl_source.y)) / temp_distance1;
        diffuse_rmin =  (Kdb2 * abs(pt2.y-
pl_source.y)) / temp_distance2;
        diffuse_rgb  diffuse;
        for(int i=0; i<cube_size; i++) {
              for(int j=0; j<cube_size; j++) {
                    struct Point_3D pt = {i,
cube_size+elev, j};
                    float distance = pow((pt.x-
pl_source.x),2)+

pow((pt.y- pl_source.y),2)+

pow((pt.z- pl_source.z),2);
                    diffuse.r = (Kdr2 *
abs(pt.y-pl_source.y)) / distance;
                    diffuse.g = (Kdg2 *
abs(pt.y-pl_source.y)) / distance;
                    diffuse.b = (Kdb2 *
abs(pt.y-pl_source.y)) / distance;
                    int scaled_kr =
getScaledKr(diffuse_rmax, diffuse_rmin,
diffuse.b);
                    struct Point lcd =
Transformation_pipeline(pt);
                    drawPixel(lcd.x, lcd.y,
getColor2(scaled_kr));
              }
        }

}


/*Main Function main()*/
int main (void)

{
      uint32_t pnum = PORT_NUM;
      pnum = 1 ;
      if ( pnum == 1 )
      SSP1Init();
```

```
	else
	puts("Port number is not correct");
	lcd_init();
	fillrect(0, 0,
ST7735_TFTWIDTH,ST7735_TFTHEIGHT,
0xBFEFFF);
	printf("Choose 1 for 3D Coordinate
for R-G-B axis\n");
	int choice;
	scanf("%d",&choice);
	int size = 50, start_pnt = 0;
	if(choice==1)
	 {
		draw3Dcoordinate();
		draw_cube(start_pnt,size);
		fill_cube(start_pnt,size);
		struct Point_3D pl_source = {-
400,20,400};
		int elev = 0;
		Draw_diffusion_XZ(start_pnt,
size, elev, pl_source);

	drawTrees_on_cube_surface(GREEN,star
t_pnt,size, 'y');

	drawTrees_on_cube_surface(YELLOW,sta
rt_pnt,size, 'x');
		Draw_diffusion(start_pnt,
size, elev, pl_source);
		draw_X(start_pnt, size);

	draw_shadow(start_pnt,size,pl_source
);
	}
	else
	{
		printf("Invalid choice.!!!!");
	}

return 0;

}
```