3D Graphics LCD Interface With LPCXpresso 1769 Using SPI

Wen Yuen Chao
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
*E-mail:*wenyuen.chao@sjsu.edu

Abstract

Our project can be divided into two parts. One part is build SPI handshake between two NXP LPC1769 to communicate and with one as master and the other as slave to act as 3D GE. Another part is transfer a 3D cubes into a 2D graph and draw the color and decoration for it using Mathematical methods.

1. Introduction

The project consists of software part and hardware part. For hardware part, we should build correct hardware connection between NXP LPC1769. Then we set one board as a master mode, another one is slave mode. The key issue is we should keep synchronization between two board, otherwise the slave board can not receive the command from mater side correctly. For software part, we should send command from mater side to slave side. Then the slave will start draw graph according the data point designed by us. In our project, we plan to project a 3D cube into 2D using transformation pipeline technique. Then we do a diffuse reflection computation to draw the color for each data point. Then we need to do bi-linear interpolation to find boundary color. Then we will fill the inside color of cube and make the cubes with 3 surfaces S1, S2 and S3 decorated on each surface with our designed letter. In the last, we draw a shade of cube at the LCD screen.

2. Methodology

First, we will introduce the problem of this project and what challenges we met. Second, we will talk about the hardware design and software design. Third, providing testing and verification to make sure our work is totally qualified. Last, making a conclusion for this project.

2.1. Objectives and Technical Challenges

For the hardware part, we are trying to build the SPI connection between two board from which we can learn the mechanism of SPI and how to initialize SPI related controller.

For the software part, we are trying to design program to set one NXP LPC1769 board as slave, another as a master. The slave will play a role at 3D GE, which will convert a 3D cube into 2D. This process can be divided into four parts. First, we should transfer the 3D cube into 2D graph using transformation pipeline technique. Secondly, we need to do bi-linear interpolation to find boundary color. thirdly, we need to draw color for the cube's three surface, defined as S1, S2, S3. Finally, we need to fill every surface with the calculated color.

2.2. Problem Formulation and Design

Give the detailed, one-to-one correspondence description of your design to attach/solve the problem and to achieve the objectives and the goal of the project:

To project a 3D cube into 2D graph, we should use transformation pipeline technique. It can be divided into two step. First step is World to viewer transform. We can use the data points designed by us multiply with below matrix.

$$T = \begin{bmatrix} -sin\theta & cos\theta & 0 & 0 \\ -cos\emptyset * cos\theta & -cos\emptyset * sin\theta & sin\emptyset & 0 \\ -sin\emptyset * cos\theta & -sin\emptyset * cos\theta & -cos\emptyset & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix} (1)$$

The second step is to perform perspective projection.

$$\begin{cases} x_i'' = \frac{D}{z_i'} * x_i' \\ y_i'' = \frac{D}{z_i'} * y_i' \end{cases} (2)$$

After this step, we will eliminate z, and get the projection of cube into 2D.

Next task is to draw the color of point. We use below formula to calculate the color of points.

$$I_{diff}(x,y,z) = k\frac{1}{\|r\|^2}\frac{\vec{n}\cdot\vec{r}}{\|\vec{n}\|\cdot\|\vec{r}\|}(r_r, r_g, r_b) \quad (3)$$

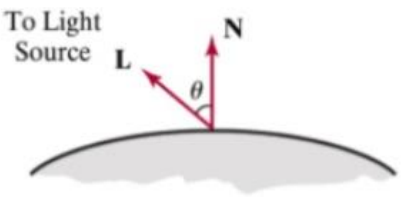

Fig. 1 Diffuse Reflection

The light source point designed by us. In our project, we assume it is white color, thus, it's color is 0xffffff. The $\| r^2 \|$ is the distance between light point and the point at 3D cube.

$$\|\vec{r}\|^2 = [x_s + \lambda(x_s - x_i)]^2 + [y_s + \lambda(y_s - y_i)]^2 + [z_s + \lambda(z_s + z_i)]^2 \quad (4)$$

cosθ is the angle between light source and the normal vector. K is diffuse coefficients, which is defined by us.

$$cos\theta = \frac{\vec{n}\cdot\vec{r}}{\|\vec{r}\|\cdot\|\vec{n}\|} \quad (5)$$

3. Implementation

In this section, we will discuss the hardware design and software design separately. In section 3.1, how each component communicate will be illustrated and we also will discuss the detail of SPI protocol. In section 3.2, we will discuss what kind of algorithm we use to make the trees and square. Moreover, pseudo code will be provided to explain how we accomplish this task.

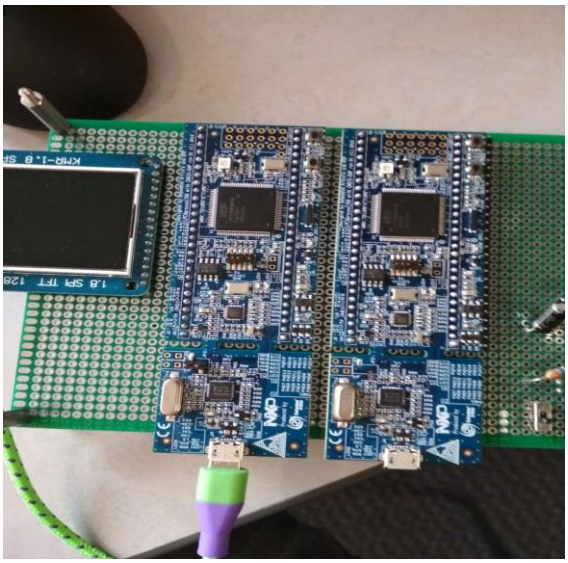

Fig. 2 NXP LPC1769 board design

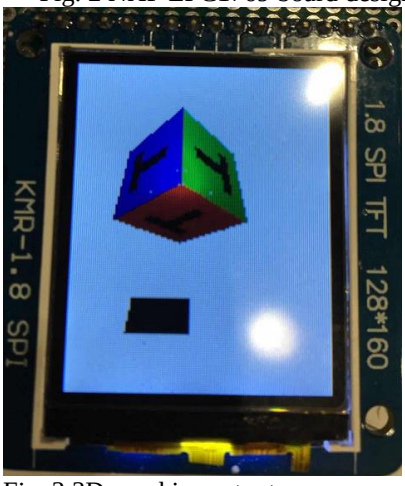

Fig. 3 3D graphics output

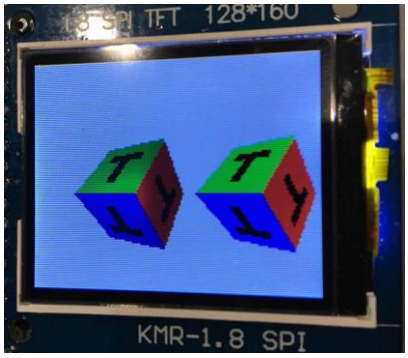

Fig. 4 output with and without light

3.1. Hardware Design

In the hardware design, we sperate this hardware system into two parts. First part is using SPI protocol between two microcontrollers, called master and slave. Second part is also using SPI protocol between slave and LCD displayer, as illustrated in Fig.1.

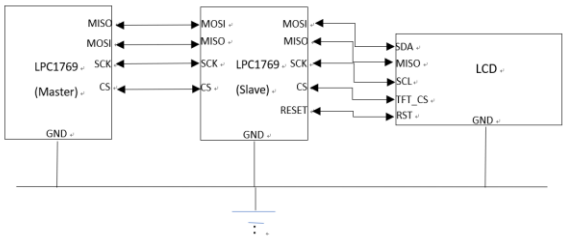

Fig. 5 Hardware Architecture

3.1.1 SPI Protocol

In Fig.3, master sent a command by using SPI to slave for asking it to draw a cube.

Next, I will explain how master communicate with slave by using SPI protocol. First step, master can choose slave by setting CS pin to low voltage level. Second step, master sent one-byte command to slave by using serial communication and each bit is sent in each clock cycle. Third step, the slave read each bit as they are received, shown in Fig. 2. After slave received these command, it can start to draw the 3D GE.

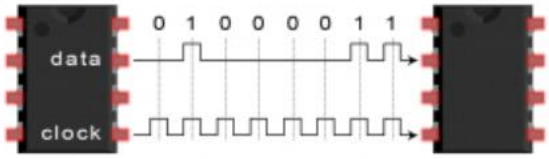

Fig. 6 serial communication

3.1.2 LCD Displayer

In this lab, we use STM7735 LCD module and the resolution is 168x128. This module has two inbuild SPI controllers and we can use it with 3.3V and 5V.

Fig. 7 Back side of LCD

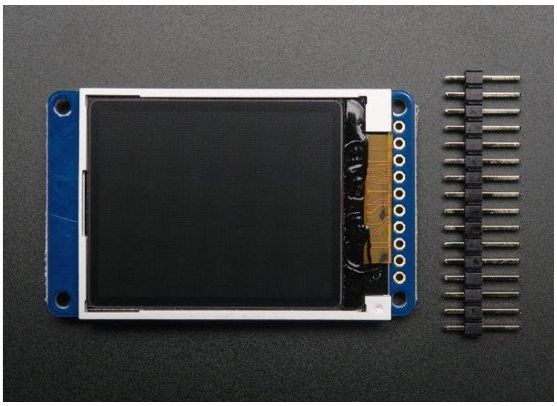

Fig. 8 Front side of LCD

Last, I attach all hardware material we used in this project, shown in Table 1.

Number	Quantity and Description	Note
1.	Wire Wrapping Board	8.5*11 inches
2.	LPC XPresso Module (ARM Cortex-M3)	2 pieces
3.	Wire Wrapping tool kit	1. Adjustable Temperature Soldering Iron 2. Solder Wire 3. Extra Lead-free Soldering Iron Tips 4. Desoldering Pump 5. Anti-static Tweezers 6. Stand with Cleaning Sponge 7. Solder Wick 8. Tool Box
4.	DC power supply (5VDC, 1500mA)	
5.	LCD color displayer module	
6.	USB to serial cable	

Table 1. bill of material

3.2. Software Design

In this session, we use a flow chart to illustrate the whole architecture of our software design as following

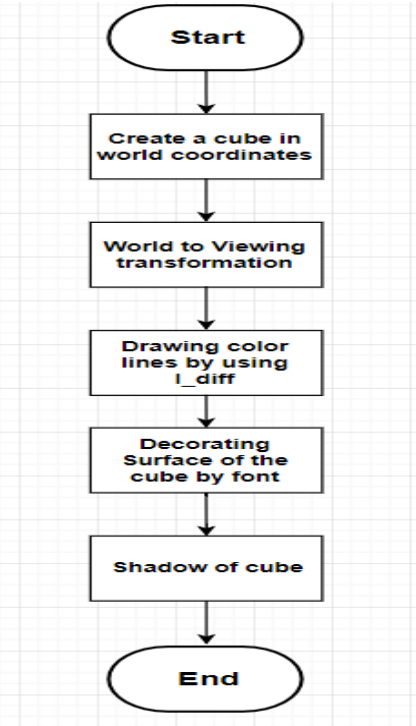

Fig. 9 serial communication

Algorithm to SPI handshake:

1. Set master board to SPI master mode, slave board to SPI slave mode.
2. Initialize system clock and SSP related register.
3. Send data from master board to slave board.

Algorithm to display 3D cube:

1. project the points received from slave into 2D using transformation pipeline
2. connect the points using drawline() function.
3. Use bilinear interpolation to find boundary color.
4. Fill the inside color of each surface S1, S2 and S3.
5. Draw the decoration for each surface with designed letter.
6. Draw the shade of cube.

3.3. Pseudo Code

```
// Define coordinate struct
struct coordinate_pnt {};
// Define camera coordinate
int cam_x, cam_y, cam_z;
// Define light coordinate
int light_x, int light_y, int light_;
// Idiff calculate function
int calIDiff(int16_t xPs, int16_t yPs, int16_t zPs, int16_t
xPi, int16_t yPi, int16_t zPi, int16_t k) {
        // calculate ||r||
        // calculate cos()
```

```
        // define K value
        // return Idiff
};
// Transform 3D coordinate to 2D coordinate
struct coordinate_pnt project_coordinates(int x_w, int y_w, int z_w) {
        // calculate theta value
        // calculate phi value
        // calculate rho value
        // using transformation matrix to calculate new coordinate (x, y, z)
// return 2D coordinate
};
// Draw original cube
// call struct coordinate_pnt project_coordinates() to get 2D coordinate
for (int i = 0; i <= 45; i++) {
for (int j = 0; j <= 45; j++) {
//get the after transformation pipeline points
//call function calIDiff, to calculate each point on the surface
//add extra source light
//get rgb color
//draw the points and color on screen
}
}
// draw these 3 surface whit these 2D coordinates
// draw color
// draw "F" on each surface
for (x axis) {
for (y axis) {
                // calculate tmp coordinates (i, j, fixed value);
                // print every pixels with #000000 color drawPixel(tmp1.x, tmp1.y, BLACK);
        }
}
// Draw original cube first
// call calIDiff() to calculate every pixel on 3D cube
callIDiff();
```

4. Testing and Verification

We should take following step to ensure we can get correct result

i. The LPC CPU module is connected to the laptop via a USB connector and the power circuit is turned on to output the voltage to the CPU module.
ii. The LCD glows when the CPU module receives the power.
iii. Import the project LCD_Test into the development tool.
iv. The project if having no errors would build and debug successfully with successful link connection to the CPU module.
v. The data would be transferred from main program in IDE to the CPU via USB cable.
vi. This will initiate the display of the screen saver by drawing squares one each at a time at random locations.
vii. After filling the entire screen, a delay is introduced so that a better view of the screen saver can be obtained.

5. Conclusion

The design and implementation of interface of LPC module with SPI LCD color display was performed successfully. The data from laptop is transmitted to CPU module and then to SPI LCD. The result was displayed on LCD screen. The work provides an opportunity to study 7805 voltage regulator, SPI LCD displays, LPCXpresso CPU module, LPCXpresso and soldering techniques and most importantly it provides an insight to Graphic Display Drivers and their functionalities.

6. Acknowledgement

By this lab, I understand how master communicate with slave by SPI. Also, knowing how to draw the 3D GE by using the formulas provided by professor Harry Li.

7. References

[1] LM7805 5V Regulator Datasheet https://www.sparkfun.com/datasheets/Components/L M7805.pdf
[2] LPCXpresso 1769 Datasheet http://www.nxp.com/documents/data_sheet/LPC1769 _68_67_66_65_64_63.pdf
[3] TJCTM24024 Pin Diagram https://123d.circuits.io/components/1100976
[4] LPCXpresso Pin Diagram https://openwsn.atlassian.net/wiki/display/OW/Xpres soHack
[5] ILI9341 Data sheet https://www.adafruit.com/datasheets/ILI9341.pdf

8. Appendix

Diagrams, source code listing, circuit schematics, relevant datasheets etc. go here.

```
/*
===============================================================
        Name        : DrawLine.c
        Author      : $RJ
        Version     :
        Copyright   : $(copyright)
        Description : main definition
```

```
==========================================
==========================================
 */
#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
struct coordinate_pnt {
        int x;
        int y;
};
int cam_x = 90;
int cam_y = 90;
int cam_z = 90;
int light_x = 50;
int light_y = 50;
int light_z = 50;
#define swap(x, y) {x = x + y; y = x - y; x = x - y ;}
// defining color values
#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF
#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 i;
        int16_t width, height;
        width = x1 - x0 + 1;
        height = y1 - y0 + 1;
```

```
		setAddrWindow(x0, y0, x1, y1);
		writecommand(ST7735_RAMWR);
		write888(color, width * height);
	}
	void lcddelay(int ms)
	{
		int count = 24000;
		int i;
		for (i = count * ms; i--; i > 0)
			;
	}
	void lcd_init()
	{
		int i;
		printf("LCD Demo Begins!!!\n");
		// Set pins P0.16, P0.21, P0.22 as output
		LPC_GPIO0->FIODIR |= (0x1 << 16);
		LPC_GPIO0->FIODIR |= (0x1 << 21);
		LPC_GPIO0->FIODIR |= (0x1 << 22);
		// Hardware Reset Sequence
		LPC_GPIO0->FIOSET |= (0x1 << 22);
		lcddelay(500);
		LPC_GPIO0->FIOCLR |= (0x1 << 22);
		lcddelay(500);
		LPC_GPIO0->FIOSET |= (0x1 << 22);
		lcddelay(500);
		// initialize buffers
		for (i = 0; i < SSP_BUFSIZE; i++) {
			src_addr[i] = 0;
			dest_addr[i] = 0;
		}
		// Take LCD display out of sleep mode
		writecommand(ST7735_SLPOUT);
		lcddelay(200);
		// Turn LCD display on
		writecommand(ST7735_DISPON);
		lcddelay(200);
	}
	void 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);
	}
	/**************************************
*********************************
	 ** Descriptions:	Draw line function
	 **
	 ** parameters:		Starting point (x0,y0),
Ending point(x1,y1) and color
	 ** Returned value:	None
	 **

**************************************************
*****************************/
	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;
			}
		}
	}
	/*
	 draw square function
	 */
	void drawSqure(int16_t xa, int16_t ya, int16_t xb,
int16_t yb, int16_t xc,
			int16_t yc, int16_t xd, int16_t
yd, uint32_t color) {
		drawLine(xa, ya, xb, yb, color);
		drawLine(xb, yb, xc, yc, color);
		drawLine(xc, yc, xd, yd, color);
		drawLine(xd, yd, xa, ya, color);
	}
```

```
	/*
	 drawMulSquare()
	 */
	void drawMulSquare(int16_t xa, int16_t ya, int16_t xb, int16_t yb, int16_t xc,
			int16_t yc, int16_t xd, int16_t yd, int16_t ct, uint32_t color) {
		while (ct < 10) {
			xa += 0.2 * (xd - xa);
			ya += 0.2 * (yd - ya);
			xb += 0.2 * (xa - xb);
			yb -= 0.2 * (yb - ya);
			xc -= 0.2 * (xc - xb);
			yc -= 0.2 * (yc - yb);
			xd -= 0.2 * (xd - xc);
			yd += 0.2 * (yc - yd);
			drawSqure(xa, ya, xb, yb, xc, yc, xd, yd, GREEN);
			ct++;
		}
	}
	/*
	 draw single tree
	 */
	void drawTree(int16_t xa, int16_t ya, int16_t xb, int16_t yb, uint32_t color) {
		drawLine(xa, ya, xb, yb, color);
		drawLine(xa, ya,
				myCos(30.0) * xb - mySin(30.0) * yb + (1 - myCos(30.0)) * (xa)
						+ mySin(30.0) * (ya),
				mySin(30.0) * xb + myCos(30.0) * yb + (1 - myCos(30.0)) * (ya)
						- mySin(30.0) * (xa), color);
		drawLine(xa, ya,
				myCos(-30.0) * xb - mySin(-30.0) * yb + (1 - myCos(-30.0)) * (xa)
						+ mySin(-30.0) * (ya),
				mySin(-30.0) * xb + myCos(-30.0) * yb + (1 - myCos(-30.0)) * (ya)
						- mySin(-30.0) * (xa), color);
	}
	/*
	 calculate destination point
	 */
	int calDesXR(int16_t xa, int16_t ya, int16_t xb, int16_t yb) {
	//	return 0.707107 * (xb + ya - yb) + 0.292893 * xa;
		return myCos(30.0) * xb - mySin(30.0) * yb + (1 - myCos(30.0)) * (xa)
				+ mySin(30.0) * (ya);
	}
	int calDesYR(int16_t xa, int16_t ya, int16_t xb, int16_t yb) {
		return mySin(30.0) * xb + myCos(30.0) * yb + (1 - myCos(30.0)) * (ya)
				- mySin(30.0) * (xa);
	}
	int calDesXL(int16_t xa, int16_t ya, int16_t xb, int16_t yb) {
		return myCos(-30.0) * xb - mySin(-30.0) * yb + (1 - myCos(-30.0)) * (xa)
				+ mySin(-30.0) * (ya);
	}
	int calDesYL(int16_t xa, int16_t ya, int16_t xb, int16_t yb) {
		return mySin(-30.0) * xb + myCos(-30.0) * yb + (1 - myCos(-30.0)) * (ya)
				- mySin(-30.0) * (xa);
	}
	/*
	 Tree Layer
	 */
	void drawTreeLayer(int16_t xs, int16_t ys, int16_t xe, int16_t ye,
			uint32_t color, int16_t ct) {
		int xr, yr, txr, tyr, txl, tyl, xl, yl;
		if (ct > 15) {
			return;
		}
		xr = calDesXR(xs, ys, xe, ye);
		yr = calDesYR(xs, ys, xe, ye);
		txr = calDesXR(xr, yr, xr, yr + (ye - ys) * 0.6);
		tyr = calDesYR(xr, yr, xr, yr + (ye - ys) * 0.6);
		xl = calDesXL(xs, ys, xe, ye);
		yl = calDesYL(xs, ys, xe, ye);
		txl = calDesXL(xl, yl, xl, yl + (ye - ys) * 0.6);
		tyl = calDesYL(xl, yl, xl, yl + (ye - ys) * 0.6);
		drawTree(xe, ye, xe + (xe - xs) * 0.6, ye + (ye - ys) * 0.6, color);
		drawTree(xr, yr, txr, tyr, color);
		drawTree(xl, yl, txl, tyl, color);
		ct = ct + 3;
		drawTreeLayer(xe, ye, xe + (xe - xs) * 0.6, ye + (ye - ys) * 0.6, color,
				ct);
		drawTreeLayer(xr, yr, txr, tyr, color, ct);
		drawTreeLayer(xl, yl, txl, tyl, color, ct);
	}
	/*
	 sin() & cos();
```

```
 */
double mySin(double a) {
	double result;
	const double pi = acos(-1.0);
	result = a * pi / 180;
	return sin(result);
}
double myCos(double a) {
	double result;
	const double pi = acos(-1.0);
	result = a * pi / 180;
	return cos(result);
}
/*
 Idiff;
 */
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);
}
struct coordinate_pnt project_coordinates(int x_w, int y_w, int z_w) {
//	int scrn_x, scrn_y, Dist = 100, x_diff = 74, y_diff = 50;
	int scrn_x, scrn_y, Dist = 100, x_diff = 74, y_diff = 100;
	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)));
	//theta = 0.785;
	//phi = 0.785;
	rho = sqrt((pow(cam_x, 2)) + (pow(cam_y, 2)) + (pow(cam_z, 2)));
	x_p = (y_w * cos(theta)) - (x_w * sin(theta));
	y_p = (z_w * sin(phi)) - (x_w * cos(theta) * cos(phi))
			- (y_w * cos(phi) * sin(theta));
	z_p = rho - (y_w * sin(phi) * cos(theta)) - (x_w * sin(phi) * cos(theta))
			- (z_w * cos(phi));
	scrn_x = x_p * Dist / z_p;
	scrn_y = y_p * Dist / z_p;
	scrn_x = x_diff + scrn_x;
	scrn_y = y_diff - scrn_y;
	screen.x = scrn_x;
	screen.y = scrn_y;
	return screen;
}
struct coordinate_pnt newProject_coordinates(int x_w, int y_w, int z_w) {
	int scrn_x, scrn_y, Dist = 100, x_diff = 74, y_diff = 30;
//	int scrn_x, scrn_y, Dist = 100, x_diff = 74, y_diff = 70;
	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)));
	//theta = 0.785;
	//phi = 0.785;
	rho = sqrt((pow(cam_x, 2)) + (pow(cam_y, 2)) + (pow(cam_z, 2)));
	x_p = (y_w * cos(theta)) - (x_w * sin(theta));
	y_p = (z_w * sin(phi)) - (x_w * cos(theta) * cos(phi))
			- (y_w * cos(phi) * sin(theta));
	z_p = rho - (y_w * sin(phi) * cos(theta)) - (x_w * sin(phi) * cos(theta))
			- (z_w * cos(phi));
	scrn_x = x_p * Dist / z_p;
	scrn_y = y_p * Dist / z_p;
	scrn_x = x_diff + scrn_x;
	scrn_y = y_diff - scrn_y;
	screen.x = scrn_x;
	screen.y = scrn_y;
	return screen;
}
void draw_coordinates() {
	struct 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(45, 0, 0);
	x2 = lcd.x;
	y2 = lcd.y;
	lcd = project_coordinates(0, 45, 0);
	x3 = lcd.x;
	y3 = lcd.y;
	lcd = project_coordinates(0, 0, 45);
	x4 = lcd.x;
	y4 = lcd.y;
```

```
		lcd = project_coordinates(45, 0, 45);
		x5 = lcd.x;
		y5 = lcd.y;
		lcd = project_coordinates(45, 45, 0);
		x6 = lcd.x;
		y6 = lcd.y;
		lcd = project_coordinates(45, 45, 45);
		x7 = lcd.x;
		y7 = lcd.y;
		lcd = project_coordinates(0, 45, 45);
		x8 = lcd.x;
		y8 = lcd.y;
		for (int i = 0; i <= 45; i++) {
			for (int j = 0; j <= 45; j++) {
				tmp = project_coordinates(i, j, 45);
				int kR = calIDiff(light_x, light_y, light_z, i, j, 45, 255);
				if (kR + 150 > 255) {
					kR = 255;
				} else {
					kR += 150;
				}
				uint32_t color = 0x000000;
				color |= ((kR |= 0x000000) << 16);
				color |= (0x000000 << 8);
				color |= 0x000000;
				drawPixel(tmp.x, tmp.y, color);
			}
		}
		for (int i = 0; i <= 45; i++) {
			for (int j = 0; j <= 45; j++) {
				tmp = project_coordinates(45, i, j);
				int kR = calIDiff(light_x, light_y, light_z, 45, i, j, 255);
				if (kR + 150 > 255) {
					kR = 255;
				} else {
					kR += 150;
				}
				uint32_t color = 0x000000;
				color |= (0x000000 << 16);
				color |= ((kR |= 0x000000) << 8);
				color |= 0x000000;
				drawPixel(tmp.x, tmp.y, color);
			}
		}
		for (int i = 0; i <= 45; i++) {
			for (int j = 0; j <= 45; j++) {
				tmp = project_coordinates(i, 45, j);
				int kR = calIDiff(light_x, light_y, light_z, 45, i, j, 255);
				if (kR + 150 > 255) {
					kR = 255;
				} else {
					kR += 150;
				}
				uint32_t color = 0x000000;
				color |= (0x000000 << 16);
				color |= (0x000000 << 8);
				color |= (kR |= 0x000000);
				drawPixel(tmp.x, tmp.y, color);
			}
		}
		struct coordinate_pnt tmp1;
	// struct coordinate_pnt tmp2;
		for (int i = 10; i <= 15; i++) {
			for (int j = 10; j <= 35; j++) {
				tmp1 = project_coordinates(i, j, 45);
				drawPixel(tmp1.x, tmp1.y, BLACK);
			}
		}
		for (int i = 15; i <= 35; i++) {
			for (int j = 20; j <= 25; j++) {
				tmp1 = project_coordinates(i, j, 45);
				drawPixel(tmp1.x, tmp1.y, BLACK);
			}
		}
		for (int i = 10; i <= 15; i++) {
			for (int j = 10; j <= 35; j++) {
				tmp1 = project_coordinates(i, 45, j);
				drawPixel(tmp1.x, tmp1.y, BLACK);
			}
		}
		for (int i = 15; i <= 35; i++) {
			for (int j = 20; j <= 25; j++) {
				tmp1 = project_coordinates(i, 45, j);
```

```
                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                    }
                }
                for (int i = 10; i <= 15; i++) {
                    for (int j = 10; j <= 35; j++) {
                        tmp1 =
project_coordinates(45, i, j);
                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                    }
                }
                for (int i = 15; i <= 35; i++) {
                    for (int j = 20; j <= 25; j++) {
                        tmp1 =
project_coordinates(45, i, j);
                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                    }
                }
        }
        void newDraw_coordinates() {
                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 = newProject_coordinates(0, 0, 0);
                x1 = lcd.x;
                y1 = lcd.y;
                lcd = newProject_coordinates(45, 0, 0);
                x2 = lcd.x;
                y2 = lcd.y;
                lcd = newProject_coordinates(0, 45, 0);
                x3 = lcd.x;
                y3 = lcd.y;
                lcd = newProject_coordinates(0, 0, 45);
                x4 = lcd.x;
                y4 = lcd.y;
                lcd = newProject_coordinates(45, 0, 45);
                x5 = lcd.x;
                y5 = lcd.y;
                lcd = newProject_coordinates(45, 45, 0);
                x6 = lcd.x;
                y6 = lcd.y;
                lcd = newProject_coordinates(45, 45,
45);
                x7 = lcd.x;
                y7 = lcd.y;
                lcd = newProject_coordinates(0, 45, 45);
                x8 = lcd.x;
                y8 = lcd.y;
                for (int i = 0; i <= 45; i++) {
                    for (int j = 0; j <= 45; j++) {
                        tmp =
newProject_coordinates(i, j, 45);
                        drawPixel(tmp.x,
tmp.y, 0xFF0000);
                    }
                }
                for (int i = 0; i <= 45; i++) {
                    for (int j = 0; j <= 45; j++) {
                        tmp =
newProject_coordinates(45, i, j);
                        drawPixel(tmp.x,
tmp.y, 0x00FF00);
                    }
                }
                for (int i = 0; i <= 45; i++) {
                    for (int j = 0; j <= 45; j++) {
                        tmp =
newProject_coordinates(i, 45, j);
                        drawPixel(tmp.x,
tmp.y, 0x0000FF);
                    }
                }
                struct coordinate_pnt tmp1;
                for (int i = 10; i <= 15; i++) {
                    for (int j = 10; j <= 35; j++) {
                        tmp1 =
newProject_coordinates(i, j, 45);
                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                    }
                }
                for (int i = 15; i <= 35; i++) {
                    for (int j = 20; j <= 25; j++) {
                        tmp1 =
newProject_coordinates(i, j, 45);
                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                    }
                }
                for (int i = 10; i <= 15; i++) {
                    for (int j = 10; j <= 35; j++) {
                        tmp1 =
newProject_coordinates(i, 45, j);
                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                    }
                }
                for (int i = 15; i <= 35; i++) {
                    for (int j = 20; j <= 25; j++) {
                        tmp1 =
newProject_coordinates(i, 45, j);
                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                    }
                }
                for (int i = 10; i <= 15; i++) {
                    for (int j = 10; j <= 35; j++) {
```

```
                                        tmp1                    =
newProject_coordinates(45, i, j);
                                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                                }
                        }
                        for (int i = 15; i <= 35; i++) {
                                for (int j = 20; j <= 25; j++) {
                                        tmp1                    =
newProject_coordinates(45, i, j);
                                        drawPixel(tmp1.x,
tmp1.y, BLACK);
                                }
                        }
        }
        int main(void)
        {
                uint32_t pnum = PORT_NUM;
                pnum = 0;
                if (pnum == 0)
                        SSP0Init();
                else
                        puts("Port    number    is    not
correct");
                lcd_init();
                fillrect(0,     0,     ST7735_TFTWIDTH,
ST7735_TFTHEIGHT, WHITE);
                draw_coordinates();
                newDraw_coordinates();
                return 0;
        }
```