3D Graphics LCD Interface With LPCXpresso 1769 Using SPI

Sashank Malladi, 010466651
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
Email: sashank.malladi@sjsu.edu

Abstract

Vector graphics is the use of geometrical primitives such as points, lines, curves, and shapes or polygons—all of which are based on mathematical expressions—to represent images in computer graphics. The purpose of this lab is to create a module with SPI enabled LCD running through a microcontroller unit using LPC1769 as microcontroller and ST7735 as LCD module. The system under consideration is the CPU module and the LCD connected through a Serial Peripheral Interface (SPI) port. Due to the embedded environment and limited port availability, serial connection is preferred over the parallel connection. The objective of this lab is achieved by successfully implementing 3D vector graphics on LCD

Keywords: LPCXpresso 1769, 1.8' LCD Module

1. Introduction

This lab is focused on implementing 3-D graphic designs on an LCD module that can collect statistical data from the CPU module. This CPU module uses LPC1769 CPU, based on the ARM cortex M0 core that is provided with a user-friendly IDE called LPCXpresso . In this embedded environment, parallel communication leads to a superfluous usage of large number of CPU pins that are required to process the signals and hence serial communication is the preferred choice of interface between the data LCD module and the CPU module. The main objective of this lab experiment is to understand designing graphics using vectors and display it on LCD module with effective SPI communication between LCD module and CPU. It is essential to have prior programming knowledge of C/C++ languages to program the microcontroller circuit.

2. Methodology

This section includes the objectives, technical details, challenges, results and design of the project explained with block diagrams, flowcharts and layouts.

2.1 Objectives and Technical Challenges

The objectives of the lab includes:

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

There were certain technical challenges faced while performing the experiment:

1. Connecting the berg strips to the LPC board.
2. Mounting and soldering the LCD display onto the breakout board.
3. Connection between LCD display and CPU module

When the berg strips were connected to the CPU module, due to the small width of the pins, it was not connected to the CPU module, so the pins were soldered.

There were loose connections between the LCD display and the CPU module pins which was resolved using jumper cables.

2.2. Problem Formulation and Design

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

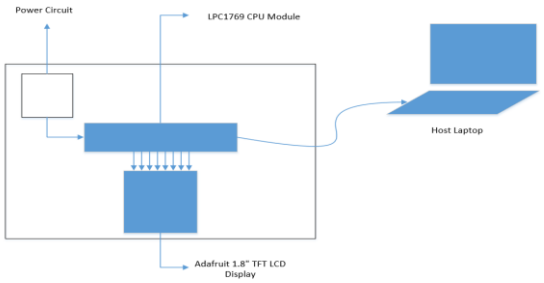

Figure 1. Wire wrap board layout

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

communication between the LCD display and the CPU module.

2.2.1 LCD Display

The CPU module used here is LPC1769 which consists of two inbuilt SPI (serial port interface) controllers for serial communication (SPI0 and SPI1). In this lab experiment, the SPI bus establish the necessary connection with the LCD module allowing transfer of required data to LCD module. The figure below provides the basic pins on the ST7735 LCD module.

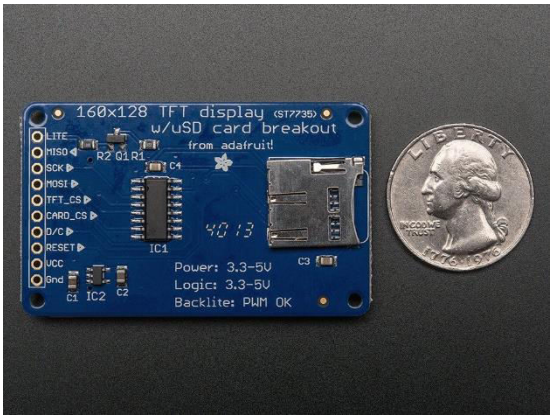

Figure 2. Adafruit 1.8" Color TFT Display pin layout

[1] Adafruit 1.8" Color TFT LCD Display, http://www.adafruit.com/products/358

The detailed pin configuration of the LCD display are as follows:

Symbol	Name
LITE	Backlight
MISO	Master In Slave Out
SCK	Serial Clock
MOSI	Master Out Slave In
TFT_CS	TFT Chip Select
CARD_CS	Card Chip Select
D/C	Data/Command selector
RESET	Reset
VCC	VCC
GND	Ground

Table 1. Pin configuration of Adafruit 1.8" TFT color LCD

[2] Adafruit 1.8" Color TFT LCD Display, http://www.adafruit.com/products/358

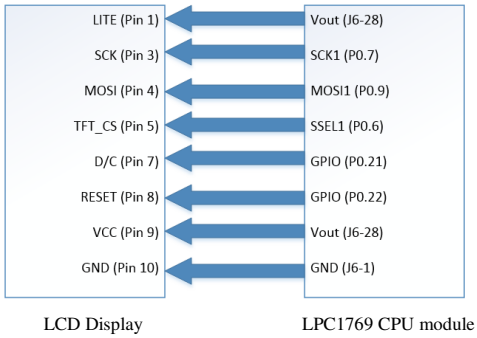

Figure 3. LCD Display interface with CPU module

[1] Adafruit 1.8" Color TFT LCD Display, http://www.adafruit.com/products/358

In our implementation, SPI port number 1 is used to connect to the LCD Display. The following table shows the required connection between the CPU module and LCD display:

1.8" Color TFT LCD	LPCXpresso 1769 CPU
LITE (Pin 1)	Vout (J6-28)
SCK (Pin 3)	SCK1 (P0.7)
MOSI (Pin 4)	MOSI1 (P0.9)
TFT_CS (Pin 5)	SSEL1 (P0.6)
D/C (Pin 7)	GPIO (P0.21)
RESET (Pin 8)	GPIO (P0.22)
VCC (Pin 9)	Vout (J6-28)
GND (Pin 10)	GND (J6-1)

Table 2. Pin connection of LCD display interface with CPU module

[3] Adafruit 1.8" Color TFT LCD Display, http://www.adafruit.com/products/358

The target board used is LPCXpresso 1769 CPU module which supports interfaces like CAN, SPI, UART, I2C, Ethernet and USB. It is used to program the flash memory for erase, read and write implementation.

The software design was implemented using the source code developed using graphics library provided by Adafruit on the LPCXpresso v6.1.0_164 IDE.

In this project, a square and a triangle is dynamically rotated and displayed on the LCD

display. Additionally, a tree pattern is dynamically drawn and displayed on the LCD display.

2.2.2 SPI Overview

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, flash memories, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.

Figure 4. SPI Interface Overview

2.2.3 Vectors Overview

Vector equation followed in implementation of rotating square screen saver is as follows

Generalized equation:

V(p)=v(p1)+ c(v(p2)-v(p1))

Based on co-ordinates

X=x1+c(x2-x1)

Y=y1+c(y2-y1)

Where X and Y are new co-ordinates reduced by factor of c(constant) which are used in the rotating square matrix.

As such new points are found on each side of the square and new square is formed with rotation and size reduction.

Equations used for world coordinates to camera coordinates

X prime[i]=-sin(phita)*x[i]+ cos(Phita)*y[i];

Yprime[i]=-cos(phita)cos(phi)x[i]-sin(phita)cos(phi)y[i]+ sin(phi)z[i];

Z prime[i]= -cos(phita)sin(phi)-cos(phita)sin(phi)y[i]-cos(phi)z[i]+rho;

Where

rho= sqrt(pow(xe,2)+ pow(ye,2)+ pow(ze,2))

3-D to 2-D coordinates

X2d = x prime[i]*D/ z prime[i];

z2d = y prime[i]*D/ z prime[i];

For shadow equation used to calculate the lamda for intersection points is as follows

Lamda=(n*a – n*pi)/ n*(ps-pi);

Where

n is perpendicular line to the XY Axis

a is the arbitrary point

ps is light source vector

pi is vertices of the cube vector

3. Implementation

The implementation consist of Adafruit 1.8” Color TFT LCD Display that uses Serial Peripheral Interface (SPI) for sequential data access. The LCD display has 4-5 wire SPI digital interface.

The LPCXpresso 1769 consist of the SSP which is a Synchronous Serial Port controller capable of operation on SPI, 4 wire SPI. It can interact with masters and slaves on the bus. Only a single master and single slave can communicate on the bus during given data transfer

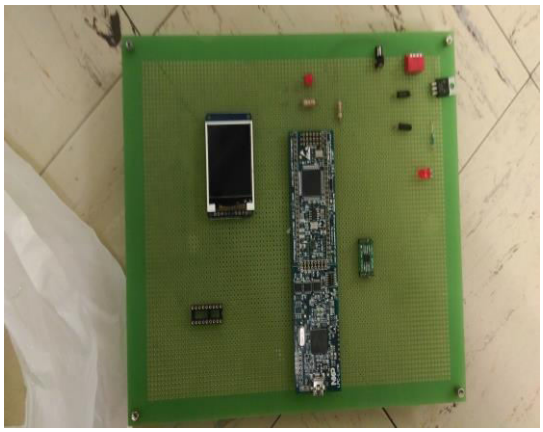

Figure 5. Embedded board design

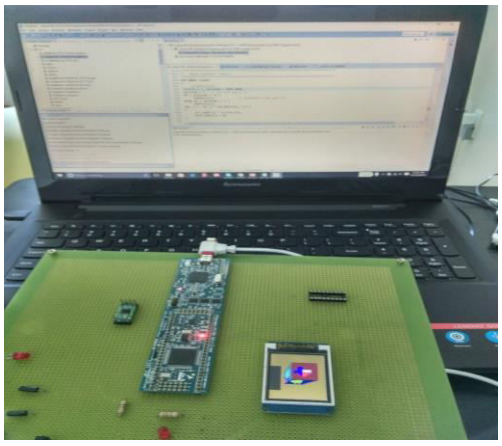

Figure 6. Project Overview

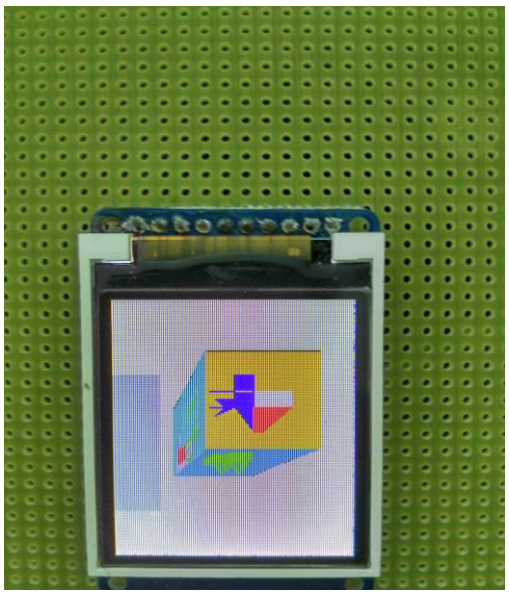

Figure 7. 3D solid cube

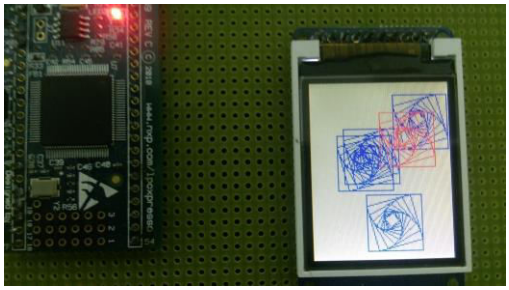

Figure 8. Rotating squares

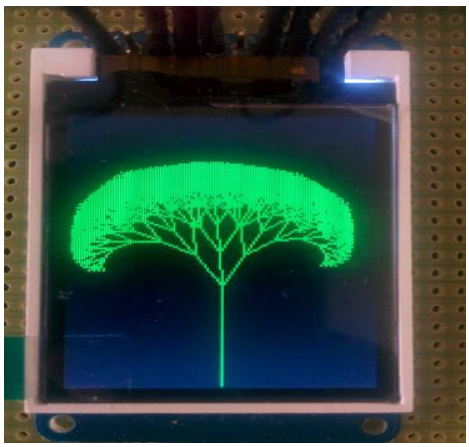

Figure 9. Tree pattern

3.1 Algorithm

Algorithm to display 3D cube:

1. Cube is plotted on world co-ordinate system.
2. Coordinates X,Y,Z in world space are converted into camera space using the transformation matrix.
3. The coordinates are then converted into 2 D coordinates
4. Using the coordinates in 2D, the 3D cube is plotted using drawline() function.
5. Use flood fill algorithm to paint the surfaces of the cube.

Algorithm to display rotating squares:

1. Initially rotating squares are drawn using drawline() function provided by the graphic library of Adafruit LCD display.
2. The coordinates of the square are then converted into 3D coordinates by adding extra dimention of Z to existing co-ordinates.
3. Transfer the pattern on to the required surface using transformation based on independent and function variables.

Algorithm to display tree patterns:

1. Initially trees are drawn using drawline() function provided by the graphic library of Adafruit LCD display.
2. The coordinates of the trees are then converted into 3D coordinates by adding extra dimention of Z to existing co-ordinates.
3. Transfer the pattern on to the required surface using transformation based on independent and function variables.

Algorithm to display shadow cube:

1. An arbitrary point is taken assuming it to be a light source.
2. From the light source using the vector equation for line we calculate the intersecting points of light source and cube vertices.
3. 3D intersecting points are converted into 2d and plotted on to LCD display.

4. Use flood fill algorithm to paint the shadow.

3.2 Flow Charts

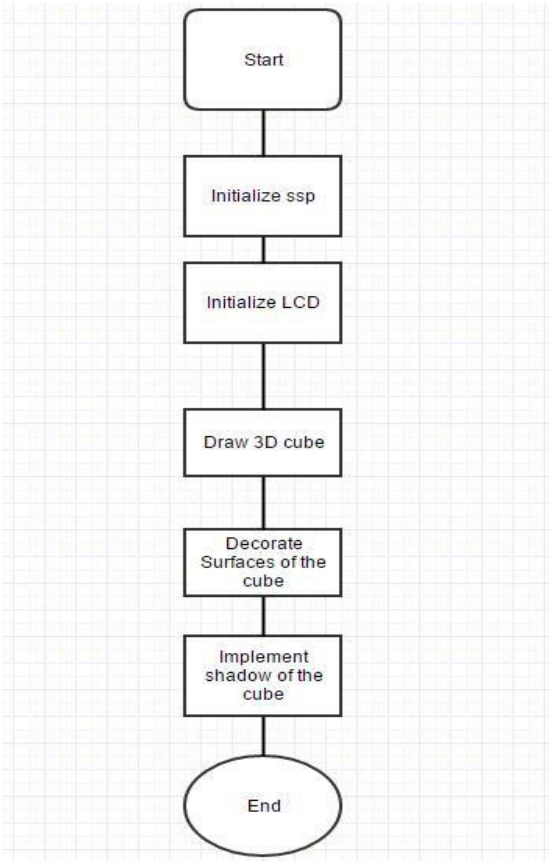

Figure 10. Implementation flowchart

3.3 Pseudo Code

```
// Function to initialize LCD
void lcd_init(){
initialize portnumber = 1;
set output as SSP0 P0.6;
set D/C as P0.21;
set RESET as P0.22;
for ( i = 0; i < SSP_BUFSIZE; i++ ) //Initialize RD and WR
buffer      {       src_addr[i] = 0;
                    dest_addr[i] = 0;
            }

          /* Sleep out */
          SSP_SSELToggle( portnum, 0 );
          src_addr[0] = 0x11;       /* Sleep out */
          SSPSend( portnum, (uint8_t *)src_addr, 1 );
          SSP_SSELToggle( portnum, 1 );
          lcddelay(200);
          /* delay 200 ms */
          /* Display on */
          SSP_SSELToggle( portnum, 0 );
          src_addr[0] = 0x29;       /* Disp On */
          SSPSend( portnum, (uint8_t *)src_addr,
1 );
          SSP_SSELToggle( portnum, 1 );
          /* delay 200 ms */
          lcddelay(200);
}

//Main Function
int main(int){
SSP1Init(); //SPI initialize
lcd_init(); //LCD initialize
//Draw 3d Cube
int
cubebase[][3]={{50,50,0},{110,50,0},{110,110,0}
,{50,110,0}};
 int
cube[][3]={{50,50,60},{110,50,60},{110,110,60},
{50,110,60}};
 int cubeitr=0;
 for(cubeitr=0;cubeitr<4;cubeitr++){
          if(cubeitr<3){
                    struct Vector3i P;
                    P.x=cube[cubeitr][0];
                    P.y=cube[cubeitr][1];
                    P.z=cube[cubeitr][2];
                    struct Vector3i P1;

          P1.x=cube[cubeitr+1][0];

          P1.y=cube[cubeitr+1][1];

          P1.z=cube[cubeitr+1][2];
                    struct      Vector2i
p2d=Point(P);
                    struct      Vector2i
p12d=Point(P1);

          drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLA
CK);

           }
           else{

                    struct Vector3i P;
                    P.x=cube[cubeitr][0];
                    P.y=cube[cubeitr][1];
                    P.z=cube[cubeitr][2];
                    struct Vector3i P1;
                    P1.x=cube[0][0];
                    P1.y=cube[0][1];
```

```
                        P1.z=cube[0][2];
                        struct Vector2i p2d=Point(P);
                        struct Vector2i p12d=Point(P1);

        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
         }
}
{
        struct Vector3i P;
        P.x=cubebase[0][0];
        P.y=cubebase[0][1];
        P.z=cubebase[0][2];
        struct Vector3i P1;
        P1.x=cubebase[1][0];
        P1.y=cubebase[1][1];
        P1.z=cubebase[1][2];
        struct Vector2i p2d=Point(P);
        struct Vector2i p12d=Point(P1);
        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
}

{
        struct Vector3i P;
        P.x=cubebase[0][0];
        P.y=cubebase[0][1];
        P.z=cubebase[0][2];
        struct Vector3i P1;
        P1.x=cubebase[3][0];
        P1.y=cubebase[3][1];
        P1.z=cubebase[3][2];
        struct Vector2i p2d=Point(P);
        struct Vector2i p12d=Point(P1);
        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
}

 cubeitr=0;
 for(cubeitr=0;cubeitr<4;cubeitr++){
          if(cubeitr!=2){
          struct Vector3i P;
                  P.x=cube[cubeitr][0];
                  P.y=cube[cubeitr][1];
                  P.z=cube[cubeitr][2];
                  struct Vector3i P1;
                  P1.x=cubebase[cubeitr][0];
                  P1.y=cubebase[cubeitr][1];
                  P1.z=cubebase[cubeitr][2];
                  struct Vector2i p2d=Point(P);
                  struct Vector2i p12d=Point(P1);

        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
          }
 }

DrawBackgroundorForest();// fill the surfaces of the cube with
color
DrawScreenSaver();// fill one of the surface with
the rotating squares
DrawForest();// fill one of the surfaces with trees
FloodFill();// fill one of the surfaces with the map
of texas

// drawing  the shadow of the cube

int shadeitr=0;
 for(shadeitr=0;shadeitr<4;shadeitr++){
          struct Vector3i p;
          p.x=cube[shadeitr][0];
          p.y=cube[shadeitr][1];
          p.z=cube[shadeitr][2];
          lamda[shadeitr]=Shade(p);
 }

float intersectionPoint[4][3]={};
shadeitr=0;
for(shadeitr=0;shadeitr<4;shadeitr++){
          struct Vector3i p;
          p.x=cube[shadeitr][0];
          p.y=cube[shadeitr][1];
          p.z=cube[shadeitr][2];
          float lam=lamda[shadeitr];
          intersectionPoint[shadeitr][0]=p.x-
lam*(90-p.x);
          intersectionPoint[shadeitr][1]=p.y-
lam*(60-p.y);
          intersectionPoint[shadeitr][2]=p.z-
lam*(10-p.z);
}

          int
intersectionPoint2d[][2]={{0,0},{0,0},{0,0},{0,0}
};
          {
          for(cubeitr=0;cubeitr<4;cubeitr++){
                                    struct
Vector3i P;

          P.x=intersectionPoint[cubeitr][0];

          P.y=intersectionPoint[cubeitr][1];

          P.z=intersectionPoint[cubeitr][2];
                                    struct
Vector2i p2d=Point(P);

          intersectionPoint2d[cubeitr][0]=p2d.x;

          intersectionPoint2d[cubeitr][1]=p2d.y;
           }
```

```
        int n = sizeof(intersectionPoint2d)/sizeof(intersectionPoint2d[0]);
        int base[]={0,0};
          int i, j;
          for(i = 0; i<160; i++) {
            for(j = 0; j <= 130; j++) {
                        struct Point p = {i, j};
                         if(isInside(intersectionPoint2d,  n, p)){
                         drawPixel(i,j,0x4c4c33);
                    }
              }
            }
}


}
```

4. Testing and Verification

This section details the testing and verification procedures used.

4.1 Testing Procedure

Following are the steps to follow for debugging purposes in case of errors.

1. The connections are checked for any shorting or loose connections.
2. The wire wrapping and the soldering techniques are checked. The board is connected to the IDE and the program is built and debugged.
3. The VCC and Ground connections are mainly verified thoroughly.
4. The schematics are checked.
5. The code is checked.
6. No pins of the SPI flash should be left without connections. All the 8 pins have to be securely wire wrapped or soldered.

4.2 Verification Procedure

If the steps given in the testing procedure are followed properly, the screenshot given below would be the correct output of the experiment:

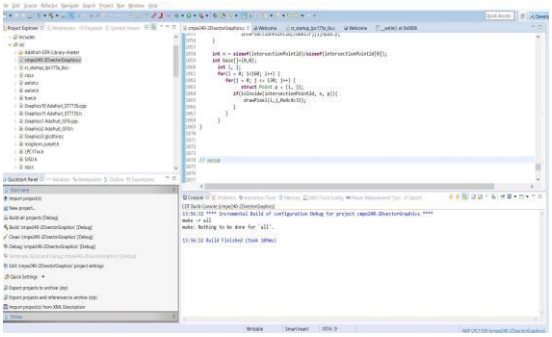

Figure 11. Program output on IDE console

5. Conclusion

The objective of this lab was to interface 1.8" Color TFT LCD Display to LPC1769 module to demonstrate Graphics Engine logic and Graphics User Interface. The rotating square, triangle and tree pattern is successfully displayed on the LCD Display.

6. Acknowledgement

I would like to thank Dr Harry Li for instructing the various steps in the project in detail. Also, I would like to thank my classmates for the meaningful discussions about the project.

7. References

[1] H.Li, Guidelines for CMPE240 project and report, Computer Engineering Department, San Jose State University, San Jose 95112.
[2] H.Li, CMPE240 Lecture Notes, Computer Engineering Department, San Jose State University, San Jose 95112.
[3] NXP LPCXpresso1769 Discussion forums at www.lpcware.com/forum.
[4] LPC1769 User manual.
[5] Adafruit 1.8" Color TFT LCD Display datasheet.
[6] ST7735R Controller datasheet.

8. Appendix

Appendix A: Bill of Materials

Number	Description
1	Wire wrapping board
2	Adafruit 1.8" Color TFT LCD Display
3	LPCXpresso 1769 CPU Module
4	Jumper cables x 6
5	Burg strips
6	Solder iron

Table 3: Bill of materials

Appendix B. Source Code

```
#include <cr_section_macros.h>
#include <NXP/crp.h>


// Variable to store CRP value in. Will be placed automatically
```

```
// by the linker when "Enable Code Read Protect" selected.
// See crp.h header for more information
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;

#include "LPC17xx.h"                    /* LPC17xx definitions */
#include "ssp.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "font.h"
#include "SJSU.h"


/* Be careful with the port number and location number, because
some of the location may not exist in that port. */
#define PORT_NUM          1
#define LOCATION_NUM      0

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

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

// Define Infinite (Using INT_MAX caused overflow problems)
#define INF 10000
//LCD
#define ST7735_TFTWIDTH  127 //LCD width
#define ST7735_TFTHEIGHT 159 //LCD height
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define swap(x, y) { x = x + y; y = x - y; x = x - y; }
#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))
//color code
#define GREEN 0x00FF00
#define BLACK  0x000000
#define RED  0xFF0000
#define BLUE  0x0000FF
#define WHITE  0xFFFFFF
#define CYAN    0x00FFFF
#define PURPLE 0x8E388E
#define YELLOW  0xFFD700
#define ORANGE 0xFF8000

int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;
int cursor_x = 0, cursor_y = 0;
//uint16_t textcolor = RED, textbgcolor= GREEN;
float textsize = 2;
int wrap = 1;

//for writing data into the SPI
void spiwrite(uint8_t c)
{
   int portnum = 1;
   src_addr[0] = c;
   SSP_SSELToggle( portnum, 0 );
   SSPSend( portnum, (uint8_t *)src_addr, 1 );
   SSP_SSELToggle( portnum, 1 );
}
//writing commands into SPI
void writecommand(uint8_t c) {
   LPC_GPIO0->FIOCLR |= (0x1<<21);
   spiwrite(c);
}
//making LCD ready to write data
void writedata(uint8_t c) {

   LPC_GPIO0->FIOSET |= (0x1<<21);
   spiwrite(c);
}
//writing data to the LCD
void writeword(uint16_t c) {

   uint8_t d;
   d = c >> 8;
   writedata(d);
   d = c & 0xFF;
   writedata(d);
}
//write colour
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 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);
}

//drawLetters()

void HLine(int16_t x0,int16_t x1,int16_t y,uint16_t color){
   width = x1-x0+1;
   setAddrWindow(x0,y,x1,y);
   writecommand(ST7735_RAMWR);
   write888(color,width);
}
void VLine(int16_t x,int16_t y0,int16_t y1,uint16_t color){
   width = y1-y0+1;
   setAddrWindow(x,y0,x,y1);
   writecommand(ST7735_RAMWR);
   write888(color,width);
}

void drawCircle(int16_t x0, int16_t y0, int16_t r, uint32_t color)
{
	int16_t f = 1 - r;
	int16_t ddF_x = 1;
	int16_t ddF_y = -2 * r;
	int16_t x = 0;
	int16_t y = r;
	drawPixel(x0 , y0+r, color);
	drawPixel(x0 , y0-r, color);
	drawPixel(x0+r, y0 , color);
	drawPixel(x0-r, y0 , color);
	while (x<y)
	{
		if (f >= 0)
		{
			y--;
			ddF_y += 2;
			f += ddF_y;
		}
		x++;
		ddF_x += 2;
		f += ddF_x;
		drawPixel(x0 + x, y0 + y, color);
		drawPixel(x0 - x, y0 + y, color);
		drawPixel(x0 + x, y0 - y, color);
		drawPixel(x0 - x, y0 - y, color);
		drawPixel(x0 + y, y0 + x, color);
		drawPixel(x0 - y, y0 + x, color);
		drawPixel(x0 + y, y0 - x, color);
		drawPixel(x0 - y, y0 - x, color);
	}
}

//Initialize LCD

void lcd_init()
{
/*
 * portnum     = 0 ;
 * cs          = p0.16 / p0.6 ?
 * rs         = p0.21
 * rst         = p0.22
 */
   uint32_t portnum = 1;
   int i;
   printf("LCD initialized\n");
   /* Notice the hack, for portnum 0 p0.16 is used
*/
   if ( portnum == 0 )
     {
      LPC_GPIO0->FIODIR |= (0x1<<16);         /*
SSP1, P0.16 defined as Outputs */
     }
     else
     {
      LPC_GPIO0->FIODIR |= (0x1<<6);          /*
SSP0 P0.6 defined as Outputs */
     }
   /* Set rs(dc) and rst as outputs */
   LPC_GPIO0->FIODIR |= (0x1<<21);           /*
rs/dc P0.21 defined as Outputs */
   LPC_GPIO0->FIODIR |= (0x1<<22);        /* rst
P0.22 defined as Outputs */



   /* Reset sequence */
   LPC_GPIO0->FIOSET |= (0x1<<22);

   for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */
   LPC_GPIO0->FIOCLR |= (0x1<<22);
   for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */
   LPC_GPIO0->FIOSET |= (0x1<<22);
   for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */

   //ask?
    for ( i = 0; i < SSP_BUFSIZE; i++ )     /* Init
RD and WR buffer */
      {
         src_addr[i] = 0;
         dest_addr[i] = 0;
      }

    /* do we need Sw reset (cmd 0x01) ? */

    /* Sleep out */
    SSP_SSELToggle( portnum, 0 );
    src_addr[0] = 0x11;    /* Sleep out */
    SSPSend( portnum, (uint8_t *)src_addr, 1 );
```

```
    SSP_SSELToggle( portnum, 1 );

    for(i = 0; i < 10000000; i++);//lcddelay(500);
/*delay 500 ms */

   /* delay 200 ms */
   /* Disp on */
    SSP_SSELToggle( portnum, 0 );
    src_addr[0] = 0x29;    /* Disp On */
    SSPSend( portnum, (uint8_t *)src_addr, 1 );
    SSP_SSELToggle( portnum, 1 );
   /* delay 200 ms */
    for(i = 0; i < 1000000; i++);//lcddelay(500);
/*delay 500 ms */
}

//Draw line function
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;
        }
        }
}

void makeBackGroundBlack()
{
        int i, j;
        for(i = 0; i < ST7735_TFTWIDTH; i++)
                for(j = 0; j <
ST7735_TFTHEIGHT; j++)
                        drawPixel(i, j, 0x00);
}

void drawBitmap(int16_t x, int16_t y, uint8_t
*bitmap, int16_t w, int16_t h, uint16_t color)
{
        int16_t i, j, byteWidth = (w + 7) / 8;
        for(j=0; j<h; j++)
        {
                for(i=0; i<w; i++ )
                {

        if(pgm_read_byte(bitmap + j * byteWidth
+ i / 8) & (128 >> (i & 7)))
                        {
                                drawPixel(x+i,
y+j, color);
                        }
                }
        }
}

void write(uint8_t c, uint8_t textSize, uint16_t
textbgcolor, uint16_t textcolor)
{
        uint8_t wrap = 1;
        if (c == '\n')
        {
                cursor_y += textSize*8;
                cursor_x = 0;
        }
        else if (c == '\r')
        {
                // skip em
        }
        else
        {
                drawChar(cursor_x, cursor_y, c,
textcolor, textbgcolor, textSize);
                cursor_x += textSize*6;
                if (wrap && (cursor_x > (100 -
textSize*6)))
                {
                        cursor_y += textSize*8;
                        cursor_x = 0;
                }
        }

}
```

```
// Draw a character
void drawChar(int16_t x, int16_t y, unsigned char c, uint16_t color, uint16_t bg, uint8_t size)
{
	int8_t i, j;
	if((x >= _width) || // Clip right
	(y >= _height) || // Clip bottom
	((x + 6 * size - 1) < 0) || // Clip left
	((y + 8 * size - 1) < 0)) // Clip top
	return;
	for (i=0; i<6; i++ ) {
	uint8_t line;
	if (i == 5)
	line = 0x0;
	else
	line = pgm_read_byte(font+(c*5)+i);
	for (j = 0; j<8; j++) {
	if (line & 0x1) {
	if (size == 1) // default size
	drawPixel(x+i, y+j, color);
	else { // big size
	fillRect(x+(i*size), y+(j*size), size, size, color);
	}
	} else if (bg != color) {
	if (size == 1) // default size
	drawPixel(x+i, y+j, bg);
	else { // big size
	fillRect(x+i*size, y+j*size, size, size, bg);
	}
	}
	line >>= 1;
	}
	}
}

void fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint32_t color)
{
	int16_t i;
	// Update in subclasses if desired!
	for (i=x; i<x+w; i++) {
		drawFastVLine(i, y, h, color);
	}
}



void drawFastVLine(int16_t x, int16_t y, int16_t h, uint32_t color)
{
	// Update in subclasses if desired!
	drawline(x, y, x, y+h-1, color);
}
```

```
//////////to detect poin is inside polygon


struct Point
{
  int x;
  int y;
};

// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
int onSegment(struct Point p, struct Point q, struct Point r)
{
  if (q.x <= MAX(p.x, r.x) && q.x >= MIN(p.x, r.x) && q.y <= MAX(p.y, r.y) && q.y >= MIN(p.y, r.y))
    return 1;
  return 0;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(struct Point p,struct Point q,struct Point r)
{
  int val = (q.y - p.y) * (r.x - q.x) -
        (q.x - p.x) * (r.y - q.y);

  if (val == 0) return 0;  // colinear
  return (val > 0)? 1: 2; // clock or counterclock wise
}

// The function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
int doIntersect(struct Point p1,struct Point q1,struct Point p2,struct Point q2)
{
  // Find the four orientations needed for general and
  // special cases
  int o1 = orientation(p1, q1, p2);
  int o2 = orientation(p1, q1, q2);
  int o3 = orientation(p2, q2, p1);
  int o4 = orientation(p2, q2, q1);

  // General case
  if (o1 != o2 && o3 != o4)
    return 1;
```

```
    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return 1;

    // p1, q1 and p2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return 1;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return 1;

     // p2, q2 and q1 are colinear and q1 lies on segment p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return 1;

    return 0; // Doesn't fall in any of the above cases
}

// Returns true if the point p lies inside the polygon[] with n
vertices
int isInside(struct Point polygon[], int n, struct Point p)
{
    // There must be at least 3 vertices in polygon[]
    if (n < 3)  return 0;

    // Create a point for line segment from p to infinite
    struct Point extreme = {INF, p.y};

    // Count intersections of the above line with sides of polygon
    int count = 0, i = 0;
    do
    {
        int next = (i+1)%n;

        // Check if the line segment from 'p' to 'extreme' intersects
        // with the line segment from 'polygon[i]' to 'polygon[next]'
        if (doIntersect(polygon[i], polygon[next], p, extreme))
        {
            // If the point 'p' is colinear with line segment 'i-next',
            // then check if it lies on segment. If it lies, return true,
            // otherwise false
            if (orientation(polygon[i], p, polygon[next]) == 0)
               return onSegment(polygon[i], p, polygon[next]);

            count++;
        }
        i = next;
    } while (i != 0);

    // Return true if count is odd, false otherwise
    return count&1;  // Same as (count%2 == 1)
}
////////////////////////////////////



struct Vector2i {
    int x, y;
};

struct Vector3i {
    int x, y, z;
};

struct Vector2i Point(struct Vector3i P ){
            int viewPoint[]={100,100,110};
            int                                          pita=
acosf(viewPoint[0]/(sqrt(pow(viewPoint[0],2)+po
w(viewPoint[1],2))));
            int
phi=acosf(viewPoint[2]/(sqrt(pow(viewPoint[0],2)
+pow(viewPoint[1],2)+pow(viewPoint[2],2))));
            struct Vector3i CameraPoint;
            CameraPoint.x=-sin(pita)*P.x+
cos(pita)*P.y;
            CameraPoint.y=-cos(pita)*cos(phi)*P.x-
sin(pita)*cos(phi)*P.y+sin(phi)*P.z;
            CameraPoint.z=-cos(pita)*sin(phi)*P.x-
cos(pita)*sin(phi)*P.y-
cos(phi)*P.z+(sqrt(pow(viewPoint[0],2)+pow(vie
wPoint[1],2)+pow(viewPoint[2],2)));

            //sqrt(pow(P.x,2)+pow(P.y,2)+pow(P.z,2)
);
            struct Vector2i derivedPoint;
            derivedPoint.x=((CameraPoint.x*120  ) /
CameraPoint.z);
            derivedPoint.y=((CameraPoint.y  *120  )/
CameraPoint.z)+140;
             return derivedPoint;
}


float Shade(struct Vector3i P ){
            int viewPoint[]={90,60,10};
            float lamda=-P.z/viewPoint[2]-P.z;
            return lamda;
}



void DrawScreenSaver(){
            int
baseCoOrdinates[4][3]={{0,0,0},{0,0,0},{0,0,0},{
0,0,0}};
            int
newCoOrdinates[4][3]={{0,0},{0,0},{0,0},{0,0}};
            uint32_t                              *color[9]
={0xFF0000,0x003399,0x00CC00,0x9933FF,0x0
0FFFF,0xFF6600,0xFF3399,0x006666,0x80000};
            int numSavers=0;
            while(numSavers<10){
            int intSquareSize=rand()%(20)+10;
            int x=rand()%(20)+50;
```

```
	int y=50;
	int z=rand()%(25)+0;
	baseCoOrdinates[0][0]=x;
	baseCoOrdinates[0][1]=y;
	baseCoOrdinates[0][2]=z;

	baseCoOrdinates[1][0]=x;
	baseCoOrdinates[1][1]=y;
	baseCoOrdinates[1][2]=z+intSquareSize;

	baseCoOrdinates[2][0]=x+intSquareSize;
	baseCoOrdinates[2][1]=y;
	baseCoOrdinates[2][2]=z+intSquareSize;

	baseCoOrdinates[3][0]=x+intSquareSize;
	baseCoOrdinates[3][1]=y;
	baseCoOrdinates[3][2]=z;

	int intColor=rand()%(8)+0;
	int saverItrations=0;
	while(saverItrations<10){
	int i=0;

	while(i<4){

		if(i==3){

			struct Vector3i P;
			P.x=baseCoOrdinates[i][0];
			P.y=baseCoOrdinates[i][1];
			P.z=baseCoOrdinates[i][2];
			struct Vector3i P1;
			P1.x=baseCoOrdinates[0][0];
			P1.y=baseCoOrdinates[0][1];
			P1.z=baseCoOrdinates[0][2];
			struct Vector2i p2d=Point(P);
			struct Vector2i p12d=Point(P1);

	drawline(p2d.x,p2d.y,p12d.x,p12d.y,color[intColor]);

	newCoOrdinates[i][0]=baseCoOrdinates[i][0]+0.2*(ba
seCoOrdinates[0][0]-baseCoOrdinates[i][0]);

	newCoOrdinates[i][1]=baseCoOrdinates[i][1]+0.2*(ba
seCoOrdinates[0][1]-baseCoOrdinates[i][1]);

	newCoOrdinates[i][2]=baseCoOrdinates[i][2]+0.2*(ba
seCoOrdinates[0][2]-baseCoOrdinates[i][2]);
		}
		else{
			struct Vector3i P;
			P.x=baseCoOrdinates[i][0];
			P.y=baseCoOrdinates[i][1];
			P.z=baseCoOrdinates[i][2];
			struct Vector3i P1;
			P1.x=baseCoOrdinates[i+1][0];
			P1.y=baseCoOrdinates[i+1][1];

			P1.z=baseCoOrdinates[i+1][2];
				struct	Vector2i
p2d=Point(P);
				struct	Vector2i
p12d=Point(P1);

			drawline(p2d.x,p2d.y,p12d.x,p12d.y,color
[intColor]);

			newCoOrdinates[i][0]=baseCoOrdinates[i
][0]+0.2*(baseCoOrdinates[i+1][0]-
baseCoOrdinates[i][0]);

			newCoOrdinates[i][1]=baseCoOrdinates[i
][1]+0.2*(baseCoOrdinates[i+1][1]-
baseCoOrdinates[i][1]);

			newCoOrdinates[i][2]=baseCoOrdinates[i
][2]+0.2*(baseCoOrdinates[i+1][2]-
baseCoOrdinates[i][2]);
			}
			i++;
		}

		memcpy(baseCoOrdinates,
newCoOrdinates, sizeof(baseCoOrdinates));
		saverItrations++;

		}
		numSavers++;
		}

}



//drawing background for forest
void DrawBackgroundorForest(){
//sec2
		int itr=0;
		for(itr=0;itr<61;itr++){
		struct Vector3i P;
		P.x=50;
		P.y=50;
		P.z=itr;
		struct Vector3i P1;
		P1.x=50;
		P1.y=110;
		P1.z=itr;
		struct Vector2i p2d=Point(P);
		struct Vector2i p12d=Point(P1);
		drawline(p2d.x,p2d.y,p12d.x,p12d.y,0x00
3d99);
```

```
        }


//sec1
        itr=0;
        for(itr=0;itr<61;itr++){
        struct Vector3i P;
        P.x=50;
        P.y=50;
        P.z=itr;
        struct Vector3i P1;
        P1.x=110;
        P1.y=50;
        P1.z=itr;
        struct Vector2i p2d=Point(P);
        struct Vector2i p12d=Point(P1);
        drawline(p2d.x,p2d.y,p12d.x,p12d.y,0x00cc99);
        }

// secup

        itr=50;
        for(itr=50;itr<110;itr++){
        struct Vector3i P;
        P.x=itr;
        P.y=110;
        P.z=60;
        struct Vector3i P1;
        P1.x=itr;
        P1.y=50;
        P1.z=60;
        struct Vector2i p2d=Point(P);
        struct Vector2i p12d=Point(P1);
        drawline(p2d.x,p2d.y,p12d.x,p12d.y,0xcc3300);
        }

}

void grow_mytree(int x0, int y0, float angle, int length, int level,
int color){
//      printf("inside mytree\n");

        int x1, y1, length1;
        float angle1;

        if(level>0){
                //x-y coordinates of branch
                x1 = x0+length*cos(angle);
//              printf("%f\n",x1);
          y1 = y0+length*sin(angle);
//          printf("%f\n",y1);

          drawline(x0,y0,x1,y1,color); //tree branch

          angle1 = angle + 0.52; //deviate right->0.52 rad/30
deg
                length1 = 0.8 * length; //reduction of
length by 20% of previous length

grow_mytree(x1,y1,angle1,length1,level-1,color);

                angle1 = angle - 0.52; //deviate left-
>0.52 rad/30 deg
                length1 = 0.8 * length;

grow_mytree(x1,y1,angle1,length1,level-1,color);

                angle1 = angle; //center->0 deg
                length1 = 0.8 * length;

grow_mytree(x1,y1,angle1,length1,level-1,color);
        }
        //      printf("exiting mytree\n");
}

void DrawForest(){
        {
                int baseX=50;
                int baseY=rand()%(30)+70;
                int baseZ=rand()%(20)+0;
                struct Vector3i P;
                P.x=baseX;
                P.y=baseY;
                P.z=baseZ;
                struct Vector2i p2d=Point(P);


grow_mytree(p2d.x,p2d.y,5.23,4,7,GREEN);
//right branch (angle = 5.23 rad/300 deg)
        //
grow_mytree(32,150,5.23,30,4,GREEN);

grow_mytree(p2d.x,p2d.y,4.18,4,7,GREEN);  //left
branch (angle = 4.18 rad/240 deg)
        //
grow_mytree(32,150,4.18,30,4,GREEN);

grow_mytree(p2d.x,p2d.y,4.71,4,7,GREEN);
//center branch (angle = 4.71 rad/0 deg)
        //
grow_mytree(32,150,4.71,30,4,GREEN);
        }

        {
                int baseX=50;
                int baseY=rand()%(30)+70;
                int baseZ=rand()%(20)+0;
                struct Vector3i P;
                P.x=baseX;
                P.y=baseY;
                P.z=baseZ;
                struct Vector2i p2d=Point(P);
```

```
        grow_mytree(p2d.x,p2d.y,5.23,4,7,GREEN);     //right
branch (angle = 5.23 rad/300 deg)
        //  grow_mytree(32,150,5.23,30,4,GREEN);
        grow_mytree(p2d.x,p2d.y,4.18,4,7,GREEN);     //left
branch (angle = 4.18 rad/240 deg)
        //  grow_mytree(32,150,4.18,30,4,GREEN);
        grow_mytree(p2d.x,p2d.y,4.71,4,7,GREEN);   //center
branch (angle = 4.71 rad/0 deg)
        //  grow_mytree(32,150,4.71,30,4,GREEN);
        }

        {
                int baseX=50;
                int baseY=rand()%(30)+70;
                int baseZ=rand()%(20)+0;
                struct Vector3i P;
                P.x=baseX;
                P.y=baseY;
                P.z=baseZ;
                struct Vector2i p2d=Point(P);

        grow_mytree(p2d.x,p2d.y,5.23,4,7,GREEN);     //right
branch (angle = 5.23 rad/300 deg)
        //  grow_mytree(32,150,5.23,30,4,GREEN);
        grow_mytree(p2d.x,p2d.y,4.18,4,7,GREEN);     //left
branch (angle = 4.18 rad/240 deg)
        //  grow_mytree(32,150,4.18,30,4,GREEN);
        grow_mytree(p2d.x,p2d.y,4.71,4,7,GREEN);   //center
branch (angle = 4.71 rad/0 deg)
        //  grow_mytree(32,150,4.71,30,4,GREEN);
        }

        {
                int baseX=50;
                int baseY=rand()%(30)+70;
                int baseZ=rand()%(20)+0;
                struct Vector3i P;
                P.x=baseX;
                P.y=baseY;
                P.z=baseZ;
                struct Vector2i p2d=Point(P);

        grow_mytree(p2d.x,p2d.y,5.23,4,7,GREEN);     //right
branch (angle = 5.23 rad/300 deg)
        //  grow_mytree(32,150,5.23,30,4,GREEN);
        grow_mytree(p2d.x,p2d.y,4.18,4,7,GREEN);     //left
branch (angle = 4.18 rad/240 deg)
        //  grow_mytree(32,150,4.18,30,4,GREEN);
        grow_mytree(p2d.x,p2d.y,4.71,4,7,GREEN);   //center
branch (angle = 4.71 rad/0 deg)
        //  grow_mytree(32,150,4.71,30,4,GREEN);
        }

}
```

```
//int
texas[][3]={{95,65,60},{80,65,60},{80,55,60},{7
5,60,60},{70,55,60},{75,65,60},{60,80,60},{60,9
5,60},{85,95,60},{85,75,60},{95,75,60}};


void FloodFill(){

int cubeitr;

//red
        //int
texas[][3]={{95,65,60},{80,65,60},{80,55,60},{7
5,60,60},{70,55,60},{75,65,60},{60,80,60},{60,9
5,60},{85,95,60},{85,75,60},{95,75,60}};
                int
texas3[][3]={{75,75,60},{75,95,60},{85,95,60},{
85,75,60},{80,75,60}};
                int
texas32d[][2]={{0,0},{0,0},{0,0},{0,0},{0,0}};
                {

        for(cubeitr=0;cubeitr<5;cubeitr++){
                                                struct
Vector3i P;

        P.x=texas3[cubeitr][0];

        P.y=texas3[cubeitr][1];

        P.z=texas3[cubeitr][2];
                                                struct
Vector2i p2d=Point(P);

        texas32d[cubeitr][0]=p2d.x;

        texas32d[cubeitr][1]=p2d.y;
                }

                struct Point p ;
                 int                n                =
sizeof(texas32d)/sizeof(texas32d[0]);
                 int base[]={0,0};
                  int i, j;
                  for(i = 0; i<160; i++) {
                    for(j = 0; j <= 130; j++) {
                                        struct  Point  p
= {i, j};

if(isInside(texas32d, n, p)){

if(i>60){

drawPixel(i,j,WHITE);
                                                }
                                        }
```

```
                    }
                }
        }



int
texas1[][3]={{95,65,60},{80,65,60},{80,55,60},{75,60,60},{70
,55,60},{75,65,60},{60,75,60},{85,75,60},{95,75,60}};
        int
texas12d[][2]={{95,65},{80,65},{80,55},{75,60},{70,55},{75,
65},{60,75},{85,75},{95,75}};
        {
        for(cubeitr=0;cubeitr<9;cubeitr++){
                                struct Vector3i P;
                                P.x=texas1[cubeitr][0];
                                P.y=texas1[cubeitr][1];
                                P.z=texas1[cubeitr][2];
                                struct Vector2i
p2d=Point(P);
                                texas12d[cubeitr][0]=p2d.x;
                                texas12d[cubeitr][1]=p2d.y;
         }

         int n = sizeof(texas12d)/sizeof(texas12d[0]);
         int base[]={0,0};
          int i, j;
          for(i = 0; i<160; i++) {
            for(j = 0; j <= 130; j++) {
                        struct Point p = {i, j};
                         if(isInside(texas12d, n, p)){
                                if(i>50){
                                 drawPixel(i,j,BLUE);
                                }
                    }
            }
          }
}
          //white
        //int
texas[][3]={{95,65,60},{80,65,60},{80,55,60},{75,60,60},{70,
55,60},{75,65,60},{60,80,60},{60,95,60},{85,95,60},{85,75,60
},{95,75,60}};

                int
texas2[][3]={{75,75,60},{60,75,60},{60,80,60},{75,95,60}};
                int texas22d[][2]={{0,0},{0,0},{0,0},{0,0}};
                {
                for(cubeitr=0;cubeitr<4;cubeitr++){
                                        struct Vector3i P;

        P.x=texas2[cubeitr][0];

        P.y=texas2[cubeitr][1];

        P.z=texas2[cubeitr][2];
                                        struct
Vector2i p2d=Point(P);

        texas22d[cubeitr][0]=p2d.x;

        texas22d[cubeitr][1]=p2d.y;
                }

                int n =
sizeof(texas22d)/sizeof(texas22d[0]);
                int base[]={0,0};
                  int i, j;
                  for(i = 0; i<160; i++) {
                    for(j = 0; j <= 130; j++) {
                                struct Point p
= {i, j};

if(isInside(texas22d, n, p)){

drawPixel(i,j,RED);
                                }
                    }
                  }
        }
}
void DrawMap(){
         int
texas[][3]={{95,65,60},{80,65,60},{80,55,60},{7
5,60,60},{70,55,60},{75,65,60},{60,80,60},{60,9
5,60},{85,95,60},{85,75,60},{95,75,60}};

         int TexItr=0;
         for(TexItr=0;TexItr<11;TexItr++){
                if(TexItr<10){
                                struct Vector3i
P;

        P.x=texas[TexItr][0];

        P.y=texas[TexItr][1];

        P.z=texas[TexItr][2];
                                struct Vector3i
P1;

        P1.x=texas[TexItr+1][0];

        P1.y=texas[TexItr+1][1];

        P1.z=texas[TexItr+1][2];
                                struct Vector2i
p2d=Point(P);
                                struct Vector2i
p12d=Point(P1);
```

```
                                struct Point temp = {p2d.x,
p2d.y};

        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);

                }
                else{

                                struct Vector3i P;
                                P.x=texas[TexItr][0];
                                P.y=texas[TexItr][1];
                                P.z=texas[TexItr][2];
                                struct Vector3i P1;
                                P1.x=texas[0][0];
                                P1.y=texas[0][1];
                                P1.z=texas[0][2];
                                struct          Vector2i
p2d=Point(P);
                                struct          Vector2i
p12d=Point(P1);

        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
                }
         }



}



/*************************************************
**************************
**   Main Function  main()
*************************************************
**************************/
int main (void)
{
          //EINTInit();
 uint32_t i, portnum = PORT_NUM;
 portnum = 1 ; /* For LCD use 1 */
 if ( portnum == 0 )
           SSP0Init();          /* initialize SSP port */
 else if ( portnum == 1 )
           SSP1Init();
 for ( i = 0; i < SSP_BUFSIZE; i++ )
 {
           src_addr[i] = (uint8_t)i;
           dest_addr[i] = 0;
 }

           //initialize LCD

 lcd_init();
 fillRect(0,0,ST7735_TFTWIDTH,ST7735_TFTH
 EIGHT,0xDBB84D);
  //cube
 //int
 cube[][3]={{65,80,0},{115,80,0},{115,130,0},{65
 ,130,0}};
  //int
 cubebase[][3]={{65,80,50},{115,80,50},{115,130,
 50},{65,130,50}};
  //int
 cube[][3]={{54,62,0},{64,52.9,0},{64,62,0},{64,6
 5.9,0}};
  //int
 cubebase[][3]={{54,61,14},{64,61.1,14},{54,51,1
 4},{64,40.9,14}};

 int
 cubebase[][3]={{50,50,0},{110,50,0},{110,110,0}
 ,{50,110,0}};
  int
 cube[][3]={{50,50,60},{110,50,60},{110,110,60},
 {50,110,60}};
  int cubeitr=0;
  for(cubeitr=0;cubeitr<4;cubeitr++){
            if(cubeitr<3){
                                struct Vector3i P;
                                P.x=cube[cubeitr][0];
                                P.y=cube[cubeitr][1];
                                P.z=cube[cubeitr][2];
                                struct Vector3i P1;

           P1.x=cube[cubeitr+1][0];

           P1.y=cube[cubeitr+1][1];

           P1.z=cube[cubeitr+1][2];
                                struct          Vector2i
 p2d=Point(P);
                                struct          Vector2i
 p12d=Point(P1);

           drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLA
 CK);

            }
            else{

                                struct Vector3i P;
                                P.x=cube[cubeitr][0];
                                P.y=cube[cubeitr][1];
                                P.z=cube[cubeitr][2];
                                struct Vector3i P1;
                                P1.x=cube[0][0];
                                P1.y=cube[0][1];
                                P1.z=cube[0][2];
```

```
                        struct Vector2i p2d=Point(P);
                        struct Vector2i p12d=Point(P1);

        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
         }
}
{
        struct Vector3i P;
        P.x=cubebase[0][0];
        P.y=cubebase[0][1];
        P.z=cubebase[0][2];
        struct Vector3i P1;
        P1.x=cubebase[1][0];
        P1.y=cubebase[1][1];
        P1.z=cubebase[1][2];
        struct Vector2i p2d=Point(P);
        struct Vector2i p12d=Point(P1);
        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
}

{
        struct Vector3i P;
        P.x=cubebase[0][0];
        P.y=cubebase[0][1];
        P.z=cubebase[0][2];
        struct Vector3i P1;
        P1.x=cubebase[3][0];
        P1.y=cubebase[3][1];
        P1.z=cubebase[3][2];
        struct Vector2i p2d=Point(P);
        struct Vector2i p12d=Point(P1);
        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
}

cubeitr=0;
 for(cubeitr=0;cubeitr<4;cubeitr++){
          if(cubeitr!=2){
          struct Vector3i P;
                P.x=cube[cubeitr][0];
                P.y=cube[cubeitr][1];
                P.z=cube[cubeitr][2];
                struct Vector3i P1;
                P1.x=cubebase[cubeitr][0];
                P1.y=cubebase[cubeitr][1];
                P1.z=cubebase[cubeitr][2];
                struct Vector2i p2d=Point(P);
                struct Vector2i p12d=Point(P1);

        drawline(p2d.x,p2d.y,p12d.x,p12d.y,BLACK);
          }
 }


DrawBackgroundorForest();
DrawScreenSaver();
DrawForest();
FloodFill();;
// draw shadow
double lamda[4];
 int shadeitr=0;
 for(shadeitr=0;shadeitr<4;shadeitr++){
          struct Vector3i p;
          p.x=cube[shadeitr][0];
          p.y=cube[shadeitr][1];
          p.z=cube[shadeitr][2];
          lamda[shadeitr]=Shade(p);
 }

float intersectionPoint[4][3]={};
shadeitr=0;
for(shadeitr=0;shadeitr<4;shadeitr++){
          struct Vector3i p;
          p.x=cube[shadeitr][0];
          p.y=cube[shadeitr][1];
          p.z=cube[shadeitr][2];
          float lam=lamda[shadeitr];
          intersectionPoint[shadeitr][0]=p.x-
lam*(90-p.x);
          intersectionPoint[shadeitr][1]=p.y-
lam*(60-p.y);
          intersectionPoint[shadeitr][2]=p.z-
lam*(10-p.z);
}

          int
intersectionPoint2d[][2]={{0,0},{0,0},{0,0},{0,0}
};
          {
          for(cubeitr=0;cubeitr<4;cubeitr++){
                                struct Vector3i
P;

          P.x=intersectionPoint[cubeitr][0];

          P.y=intersectionPoint[cubeitr][1];

          P.z=intersectionPoint[cubeitr][2];
                                struct Vector2i
p2d=Point(P);

          intersectionPoint2d[cubeitr][0]=p2d.x;

          intersectionPoint2d[cubeitr][1]=p2d.y;
           }

          int                   n                   =
sizeof(intersectionPoint2d)/sizeof(intersectionPoin
t2d[0]);
                                int base[]={0,0};
            int i, j;
            for(i = 0; i<160; i++) {
              for(j = 0; j <= 130; j++) {
```

```
                                struct Point p = {i, j};
                                 if(isInside(intersectionPoint2d,       n,
p)){
                                 drawPixel(i,j,0x4c4c33);
                            }
                      }
                   }
}

  return 0;
}


/*************************************************
**************************
**                      End Of File
*************************************************
**************************/



/*
```