Implementation of 3D Graphics

Jay Jatinkumar Patel
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 95112
E-mail: jay.j.patel@sjsu.edu

Abstract

This paper describes the design and development details of hardware and software components required for interfacing LPCXpresso 1769 with SPI enabled TFT LCD. The graphical LCD is an 18-bit color TFT display. This paper mainly presents implementation, design and testing of 3D graphics implementation on LPCXpresso CPU module and LCD as output terminal. A 3D cube with different 2D graphics on each of its face has been implemented by communicating with SPI bus of LPC1769 module.

Keywords-LPCXpresso 1769, SPI LCD, 3D Graphics.

1. Introduction

The main objective of the project is to display a cube on LCD through 3D Graphics implementation on LPCXpresso 1769. The goal of this project is to learn how to interface display device with LPCXpresso 1769, to use LPCXpresso IDE and to program the CPU module to implement 3D graphics.

The LPCxpresso LPC1769 module is a development module which has ARM Cortex-M3 Microcontroller in it. The module has UART, SPI and I2C port which help us to communicate with the module. To communicate with the external LCD, we are using SPI port of the module. A 1.8" LCD is interfaced with the LPCXpresso 1769. The LCD used in this lab is a 1.8" diagonal LCD TFT display with 128x160 resolution, 18-bit (262,144) color and has 4 or 5 wire SPI digital interface. This LCD is then initialized by sending the required opcode with the help of SPI. The function to display line on the LCD is called with the help of SPI.

2. Methodology

This section of the design methodology describes the overall system layout as well as the hardware and software design methodology. Basic components involved in the design are computer, LPCXpresso CPU module, SPI LCD, prototype board and power adapter. The host laptop is connected to CPU module using micro USB connector. The CPU module is connected to LCD.

2.1. Objectives and Technical Challenges

The following are the objective of this project:

1. Provide 3.3V DC supply to LCD to power up the module. This 3.3V DC is given from 28th pin of CPU module.
2. To build interface circuit on wire-wrapping board using external LCD.
3. Connect the pins and check the connectivity using a multimeter.
4. Write a SPI interface program for LCD.
5. Writing program for displaying a cube.
6. Test and debug the system if necessary.

The technical challenges came on both the hardware and the software side. The LCD must be connected properly to the CPU module to complete the communication. The SPI port of module and LCD must be defined and initialized properly.

2.2. System Design

The power supply is designed to power on the LCD and the CPU module. The CPU Module is also powered by connecting the USB cable. Upon powering up by connecting a power adapter, the Master Out Slave In (MOSI) will transmit the op-code from the computer to the LCD, and this will convert the op-code to the particular operation and will try to retrieve the bytes. The data can be accessed on the rising edge of the serial clock. A standard SPI uses MISO to read data or status from the device on the falling edge of the clock. In this way, the CPU module transmits the data onto SPI LCD and the perspective projection of a 3D cube is implemented and displayed on LCD.

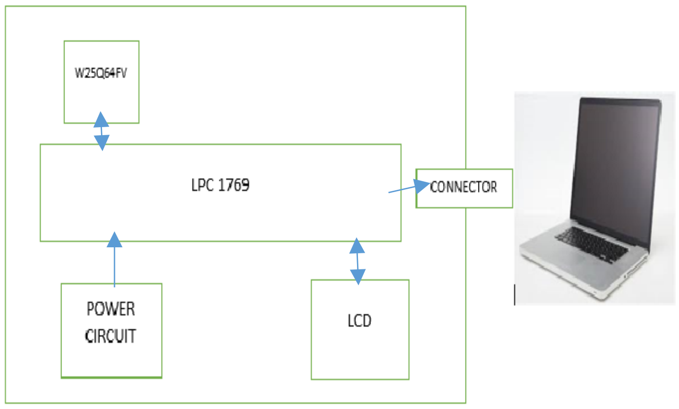

Fig-1. System level block diagram.

3.Implementation

This section describes the implementation and organization of the process and hardware and software description of the module. The power circuit module uses 7805 IC as a voltage regulator for CPU module. To test the input power to CPU and LED light on the CPU Module will glow. The circuit is connected between the LCD and CPU module as shown in the below figure.

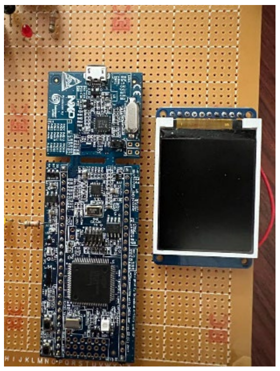
Fig-2. Prototype Board

3.1. Hardware Design

The hardware setup includes 5V regulated power supply circuit, LPC1769 CPU module and AT45 SPI Flash Memory.

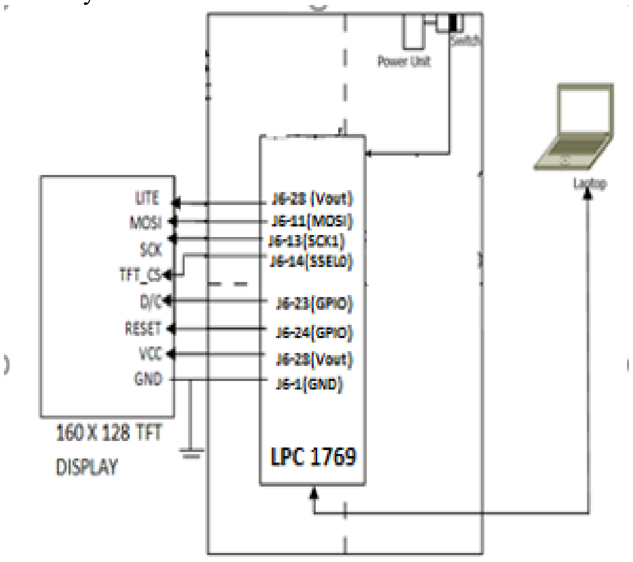

Fig-3. Schematic design of prototype board

The Power rating of the CPU module and the LCD are taken care so that the device is not burnt out. The CPU module is powered by 5V. The LCD Module is powered by connecting to the CPU module. This is from the Vout of the CPU module. The control registers of the SSP ports are to be configured. They should be configured such that, it is able to accept only eight bits of data. And also, the transmit pin cannot be set until it is filled up. The CPU module requires the control register to be set to its particular LCD opcode, i.e. SSP0. The clock is also selected for the SSP and the clock has to be scaled down for the master.

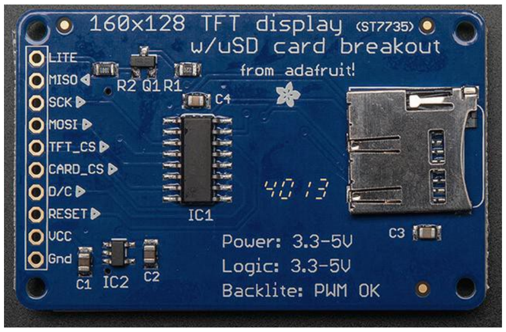

Fig-4. LCD Pin Description

CPU MODULE	TFT LCD DISPLAY
Vout (J6-28)	LITE (Pin 1)
MISO0(J6-12)	MISO (Pin 2)
SCK1 (J6-13)	SCK (Pin 3)
MOSI0 (J6-11)	MOSI (Pin 4)
SSEL0 (J6-14)	TFT_CS (Pin 5)
Not Used	CARD_CS (Pin 6)
GPIO(P0.21)(J6-23)	D/C (Pin 7)
GPIO(P0.22)(J6-24)	RESET (Pin 8)
Vout(J6-28)	VCC (Pin 9)
GND(J6-1)	GND (Pin 10)

Table 1. Circuit level connections between LPC and LCD

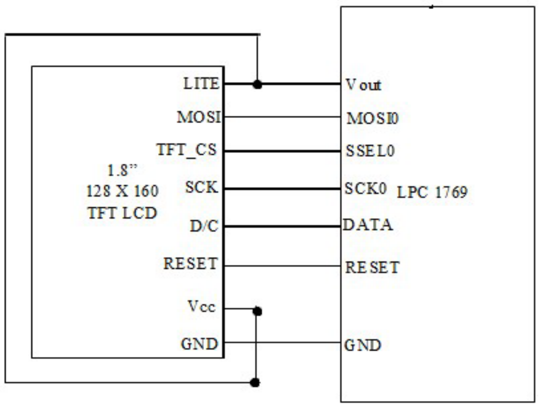

Fig-5. TFT LCD Display and LPC 1769 Connections

As seen from the figure above, the LCD has 10 pins. The module is connected to the CPU via USB cable through which it is power up. The LPC1769 J6-11, J6-13 and J6-14 pin is used for communication as we are using SPI port number 0 for our algorithm. The external LCD is connected to this port by connecting the appropriate pin of our external LCD with the module SPI port which we are using for communication. The reset and Data command pin of the LCD is connected to the pin number J6-23, J6-24 of the LPC module. Input is the fourth pin that is connected to MOSI (J6-11) of the LPC module. The chip select is the fifth pin and this is used to enable or disable the LCD operations.

When the CS is high the device is deselected and data output pins are at high impedance. The output of the flash that is the second pin is connected to the MISO of the CPU module. The SPI instruction use MOSI and serially writes instructions on the rising edge of the clock. Once LITE pin is connected which is the backlight pin, the LCD glows. This is connected to the Vcc (J6-28) so that the pin is forced high. The TFT_CS signal should be kept low for the duration of the RESET operation to avoid resetting the internal logic state of the device. The CARD_CS is not used in our scenario. Instructions vary in length. For some we send only the opcode, for some we send dummy bytes also so that there is some time for the CPU module to recognize the instruction.

3.2 Software Design

LPCXpresso v8.1.2 603 IDE is used as the development tool. Microsoft Windows 10 is the operating systems platform. The LPCXpresso IDE project consists of three main files.

1. ssp.c: This file contains the initialization code for SSP i.e. SPI of LPC1769.
2. ssp.h: It is the header file and contains all the definitions of SPI registers and function prototypes.
3. 3D-Cube.c: It is the main code. Functions to display 3D cube and display of 2D objects like square screensaver and a map or last name initials and growing tree screensaver on different faces are implemented in this code.

The LPCXpresso IDE is used as our development tool and Console is used to print the data that was received at the terminal. The next step is to debug the program step by step and try to figure out the errors in the software part. If there are, we rectify it and try to implement the same procedure. If everything is correct, then the data transmitted will be shown in the terminal. One more feature is to add the clock rate which should match on par with the LCD. Only if this is correct, the correct data will be displayed else errors may occur.

3.2.1 Flowchart

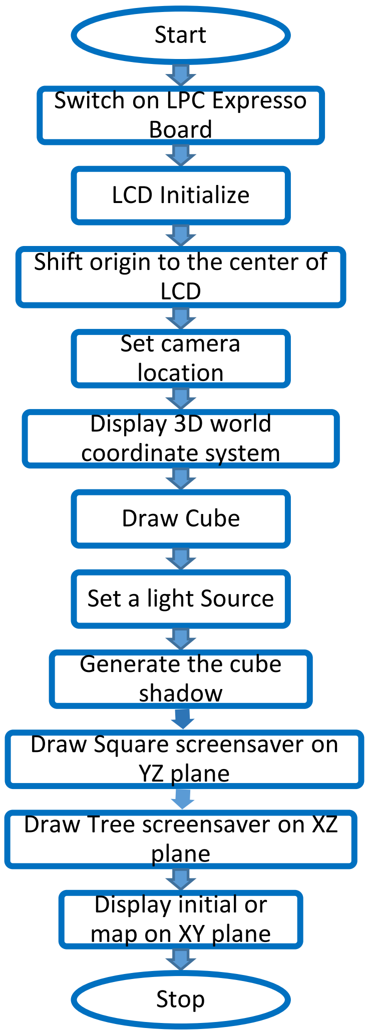
Start
Switch on LPC Expresso Board
LCD Initialize
Shift origin to the center of LCD
Set camera location
Display 3D world coordinate system
Draw Cube
Set a light Source
Generate the cube shadow
Draw Square screensaver on YZ plane
Draw Tree screensaver on XZ plane
Display initial or map on XY plane
Stop

Fig-6. Flowchart showing the software implementation.

3.2.2 Algorithm

The algorithm is as discussed below:

1. Initialize SPI communication between LPC1769 and LCD
2. Draw the world coordinate system.
3. Find 2D points for 3D cube using 3D-2D transformations and perspective projection.
4. Draw the cube in XYZ plane of LCD.
5. Set a point for source of light.
6. Find the intersection points of the light source and cube surface and XY plane.
7. Draw the shadow of the cube.
8. On the face of the cube on the YZ plane implement the 2D rotating squares screensaver.
9. On the face of the cube on the XZ plane implement the 2D growing trees screensaver.

10. On the face of the cube on the XY plane draw two initials of the name.
11. Fill all the shapes using flood fill algorithm.

To build and run program on IDE, below steps are followed:

12. Open 3D-Cube project in the IDE.
13. Open 3D-Cube.c file.
14. Connect the LPC 1769 board to your computer with a micro USB cable.
15. Build the project by clicking the hammer button. Observe the console's output. Make sure there are no errors.
16. Click the debug button. It's the green bug button. A window will pop up asking you to select your LPC-Link.
17. Once selected, code is loaded on the target and Eclipse will switch to a debug perspective.
18. Click the resume (F8) button to execute the code.

3.2.3 Pseudo Code

```
3D transformation matrix
    $\Theta = \mathrm{Cos}\text{-}1\ (Xeye / \sqrt{(Xeye^2+Yeye^2)})$
    $\phi = \mathrm{Cos}\text{-}1\ (Zeye / \sqrt{(Xeye^2+Yeye^2+Zeye^2)})$
    $\rho = \sqrt{(Xeye^2+Yeye^2+Zeye^2)}$
    $Xprime = YWorld*\mathrm{Cos}\ \Theta - Xworld*\mathrm{Sin}\ \Theta$
    $Yprime = ZWorld*\mathrm{Sin}\ \phi - \mathrm{Cos}\ \Theta\ \mathrm{Cos}\ \phi *(XWorld-YWorld)$
    $Zprime = \rho - \mathrm{Cos}\ \Theta\ \mathrm{Sin}\ \phi\ (YWorld-XWorld) - Zworld* \mathrm{Cos}\ \phi$

Perspective Projection
    $XScreen = Xprime*D/Zprime$
    $YScreen = Yprime*D/Zprime$

Shadow Computation
$XShadow = X - ((Z / (ZSource - Z))*(XSource-X))$
$YShadow = Y - ((Z / (ZSource - Z))*(YSource-Y))$
```

```
LCD_Initialization
void draw_line() {
    Function to draw a line on LCD display
  }
 void draw3DCube() {
   Function to draw a cube on LCD display
  }
void diffusedReflection() {
   Function to calculate diffusive reflection at a point
  }
 void drawShadow() {
   Function to implement shadow of the cube.
  }
void drawTree(){
    Function to implement growing trees screensaver on surface of the cube.
  }

void draw_J(){
    Function to draw initial (J) on the surface of the cube.
  }
void draw_P(){
    Function to draw initial (P) on the surface of the cube.
  }

 int main() {
    SSP0_init();
    lcd_init();
    drawCoords();
    drawShadow();
    draw3DCube();
    diffusedReflection();
    drawTree();
    draw_J();
    draw_P();
    return 0;
  }
```

4. Testing and Verification

This section provides testing and verification procedure for 3D graphics cube on LCD. The LPC CPU module was connected to the laptop via a USB connector and the power circuit was turned on. The project was imported into development tool including "ssp.c", "ssp.h" and "LCD_Test.c" files.

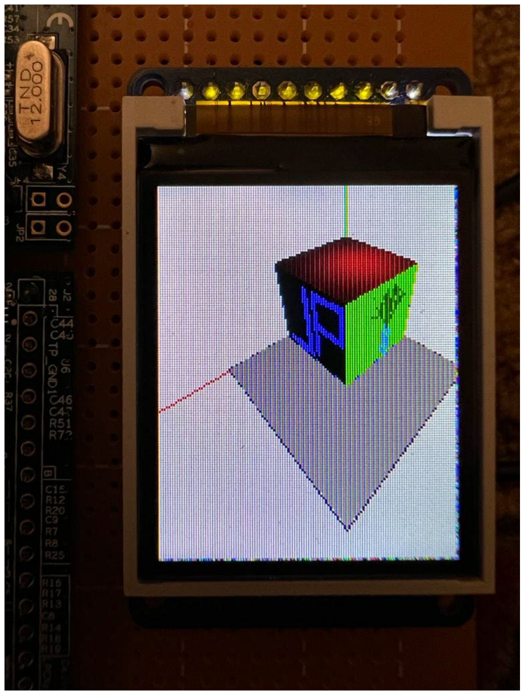

Fig-7. 3D cube with all its faces and shadow

The project was build and debugged successfully without any errors with successful link connection to the CPU module. A 3D cube of size 100 was drawn on XYZ plane. Random sized colorful squares were displayed on one surface of cube on the LCD. The squares screensaver on the YZ plane face of the cube rotated as well shrink simultaneously. After a square becomes a single pixel, new random color square pops up.

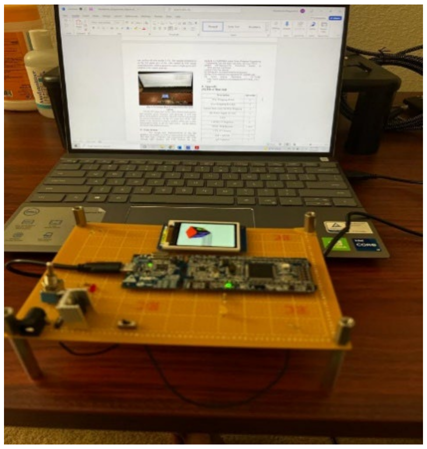

Fig-7. Prototype Board connected to the host laptop

On the XZ plane face of the cube a brown trunk grows and multiple green branches start growing. A new tree grows from random point after the previous tree has around six levels of branches. On the XY plane face of the cube initial of the last name is drawn. Shadow of the cube is also drawn and it varies as per the light source. All the shapes are filled with different colors.

5. Conclusion

The design and implementation of the 3D graphics and display on LCD is implemented successfully. The data from Computer was transmitted to LCD through CPU Module using SPI protocol. The LCD displays the data correctly. Thus from the project we learned about 7805 voltage regulator, SPI LCD Interface and LPCXpresso CPU module and 3D to 2D transformations. Thorough reading of datasheets and user manual is essential. Thorough understanding of 3D transformation matrix, perspective projection and flood fill algorithm is essential for implementation of the interface.

6. Acknowledgement

Thanks to Professor Li for his devoted guidance and thorough discussions on basic information needed to work on the project as well as the interface between SPI LCD and LPC 1769 CPU module and the understanding of the 3D transformation matrix, perspective projection and flood fill algorithm.

7. References

[1] Dr.H. Li, Guidelines for CMPE240 project and report, Computer Engineering Department, San Jose State University, San Jose 95112
[2] Dr.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] W25Q64FV W25Q64FVDB081D datasheet
[6] http://www.adafruit.com/datasheets/W25Q80BV.pdf
[7] LCD Screen Datasheet – ST7735R http://www.adafruit.com/datasheets/ST7735R_V0.2.pdf

8. Appendix

[A] Bill of Materials

Description	Quantity
Wire Wrapping Board	1
Wire Wrapping Tool Kit	1
Various Wire Color for Wire Wrapping	3
DC Power Supply 6V 10A	1
LED	1
LM7805 5V Regulator	1
680 Ω, 1K Ω Resistor	1 each
LCD TFT Display	1
10µF Capacitor	4
.1µF Capacitor	2
16Hole IC Sockets for Wire Wrapping	2 each
Soldering Rod	1
SOIC 8-pin Flash Memory	1
1µF Polarized Capacitors	2

Table.2. Bill of Material

[B] Source Code

```
// Course: CMPE - 240
// Submitted by: Jay Jatinkumar Patel
// SID: 015216357
```

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

// Define Global variables
#define PORT_NUM 0
#define LOCATION_NUM 0
uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];
int colstart = 0;
int rowstart = 0;

// Define colors
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
//#define BLUE 0x0000FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF
#define YELLOW 0xFFFF00
#define SILVER 0xC0C0C0
#define LIME 0x00FF00
#define ORANGE 0xFFA500
#define MAROON 0x800000
#define FOREST 0x228B22
#define DARKGREEN 0x006400
#define LIGHTGREEN 0X90EE90
#define SEAGREEN 0X2E8B57
#define ORANGERED 0XFF4500
#define GOLD 0XFFD700
#define GRAY 0X808080
#define SHADOW 0X8A795D

struct coordinates
{
   int x;
   int y;
};

int cam_x = 200;
int cam_y = 200;
int cam_z = 200;

struct coordinate
{
   int x;
   int y;
   int z;
};

int xcamera = 200, ycamera = 200, zcamera = 200;
int elev = 0;
int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;
int cursor_x = 0, cursor_y = 0;

uint32_t colortext = GREEN, colorbg = YELLOW;
float textsize = 3;
int wrap = 1;
int round(float number)
{
   return (number >= 0) ? (int)(number + 0.5) :
(int)(number - 0.5);
}

void drawVLine(int16_t x, int16_t y, int16_t h, uint32_t
color)
{
   drawline(x, y, x, y + h, color);
}

void drawBackground(int16_t x, int16_t y, int16_t w,
int16_t h, uint32_t color)
{
   int16_t i;
   for (i = x; i < x + w; i++)
   {
      drawVLine(i, y, h, color);
   }
}

struct coordinates transfPipeline(int xw, int yw, int zw)
{
   int xDPrime, yDPrime, D = 100, l1 = 80, l2 = 50;
   double xPrime, yPrime, zPrime, theta, phi, rho;
   struct coordinates proj;

   theta = acos(xcamera / sqrt(pow(xcamera, 2) +
pow(ycamera, 2)));
   phi = acos(zcamera / sqrt(pow(xcamera, 2) +
pow(ycamera, 2) + pow(zcamera, 2)));
   rho = sqrt((pow(xcamera, 2)) + (pow(ycamera, 2)) +
(pow(zcamera, 2)));
   xPrime = (yw * cos(theta)) - (xw * sin(theta));
   yPrime = (zw * sin(phi)) - (xw * cos(theta) * cos(phi))
- (yw * cos(phi) * sin(theta));
   zPrime = rho - (yw * sin(phi) * cos(theta)) - (xw *
sin(phi) * cos(theta)) - (zw * cos(phi));
   xDPrime = xPrime * D / zPrime;
   yDPrime = yPrime * D / zPrime;
   xDPrime = l1 + xDPrime;
   yDPrime = l2 - yDPrime;
```

```
    proj.x = xDPrime;
    proj.y = yDPrime;
    return proj;
}

void drawCoords()
{
    struct coordinates lcd;
    int x1, y1, x2, y2, x3, y3, x4, y4;
    lcd = transfPipeline(0, 0, 0);
    x1 = lcd.x;
    y1 = lcd.y;
    lcd = transfPipeline(250, 0, 0);
    x2 = lcd.x;
    y2 = lcd.y;
    lcd = transfPipeline(0, 250, 0);
    x3 = lcd.x;
    y3 = lcd.y;
    lcd = transfPipeline(0, 0, 250);
    x4 = lcd.x;
    y4 = lcd.y;
    drawline(x1, y1, x2, y2, RED);
    drawline(x1, y1, x3, y3, BLUE);
    drawline(x1, y1, x4, y4, GREEN);
}

float diffusedReflection(struct coordinate Pi)
{
    struct coordinate Ps = {40, 60, 160};
    float red;
    float scaling = 4000, shift = 0.2;
    float distanceSqr = (pow((Ps.x - Pi.x), 2) + pow((Ps.y -
Pi.y), 2) + pow((Ps.z - Pi.z), 2));
    float cosine = ((Ps.z - Pi.z) / sqrt(distanceSqr));
    float temp = cosine / distanceSqr;
    temp = (scaling * temp);
    temp = (temp < shift) ? shift : temp;
    red = (255 * 0.8 * temp);
    return red;
}

void fillShadow(struct coordinate startPt, struct
coordinate endPt)
{

    struct coordinate temp;
    struct coordinates currPt;
    for (float i = startPt.y; i < endPt.y; i++)
    {
        for (float j = startPt.x; j < endPt.x; j++)
        {
            temp.x = j;
            temp.y = i;
            temp.z = 0;
            currPt = transfPipeline(temp.x, temp.y, temp.z);
            drawPixel(currPt.x, currPt.y, SHADOW);
        }
    }
}

void fillside(struct coordinate startPt, struct coordinate
endPt, int size, int side)
{
    struct coordinate temp;
    struct coordinates currPt;
    if (side == 1)
    {
        for (float i = startPt.z; i < endPt.z; i++)
        {
            for (float j = startPt.y; j < endPt.y; j++)
            {
                temp.x = size;
                temp.y = j;
                temp.z = i;
                currPt = transfPipeline(temp.x, temp.y, temp.z);
                drawPixel(currPt.x, currPt.y, BLACK);
            }
        }
    }
    else if (side == 2)
    {
        for (float i = startPt.z; i < endPt.z; i++)
        {
            for (float j = startPt.x; j < endPt.x; j++)
            {
                temp.x = j;
                temp.y = size;
                temp.z = i;
                currPt = transfPipeline(temp.x, temp.y, temp.z);
                drawPixel(currPt.x, currPt.y, GREEN);
            }
        }
    }

    else if (side == 3)
    {
        uint32_t color;
        float r, g = 0, b = 0;
        for (float i = startPt.y; i < endPt.y; i++)
        {
            for (float j = startPt.x; j < endPt.x; j++)
            {
                temp.x = j;
                temp.y = i;
                temp.z = size + elev;
                r = diffusedReflection(temp);
                g = 0;
                b = 0;
            color = (((uint32_t)round(r)) << 16) +
(((uint32_t)g) << 8) + ((uint32_t)b);
            currPt = transfPipeline(temp.x, temp.y, temp.z);
            drawPixel(currPt.x, currPt.y, color);
         }
      }
   }
}

uint32_t findColor(float x, float y, float Idiff1, float
Idiff2, struct coordinates p1, struct coordinates p2)
{
   float newx, newy, ncolor;
   uint32_t color;
   float green, blue;

   newx = Idiff1 + (Idiff2 - Idiff1) * (x - p1.x) / (p2.x -
p1.x);
   newy = Idiff1 + (Idiff2 - Idiff1) * (y - p1.y) / (p2.y -
p1.y);
   ncolor = (newx + newy) / 2.0;
   green = 0;
   blue = 0;
   color = (((uint32_t)round(ncolor)) << 16) +
(((uint32_t)green) << 8) + ((uint32_t)blue);

   return color;
}

void DDA_line(struct coordinates p1, struct coordinates
p2, float Idiff1, float Idiff2)
{
   uint32_t color;

   int dx = p2.x - p1.x;
   int dy = p2.y - p1.y;

   int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

   float xIncrement = dx / (float)steps;
   float yIncrement = dy / (float)steps;

   float X = p1.x;
   float Y = p1.y;
   for (int i = 0; i < steps; i++)
   {
      color = findColor(X, Y, Idiff1, Idiff2, p1, p2);
      drawPixel(X, Y, color);
      X += xIncrement;
      Y += yIncrement;
   }
}

void draw_J(struct coordinate startPt, struct coordinate
endPt, int size)
{
   struct coordinate temp;
   struct coordinates currPt;
   int start_pnt = 0;

         for (float i = startPt.z; i < endPt.z; i++) {
                for (float j = startPt.y; j < endPt.y; j++)
{
                       temp.x = size;
                       temp.y = j;
                       temp.z = i;
                       currPt = transfPipeline(temp.x,
temp.y, temp.z);

                       if (j >= start_pnt + 6 && j <=
(size / 2) - 5
                                     && i >=
start_pnt + 2 && i <= start_pnt + 6)
                              drawPixel(currPt.x,
currPt.y, BLUE);
                       else if (j >= (size / 2) - 10 && j
<= (size / 2) - 5
                                     && i >=
start_pnt + 10 && i <= size - 10)
                              drawPixel(currPt.x,
currPt.y, BLUE);
                }
         }

}

void draw_P(struct coordinate startPt, struct coordinate
endPt, int size)
{
         struct coordinate temp;
         struct coordinates currPt;
         int start_pnt = 0;

         for (float i = startPt.z; i < endPt.z; i++) {
                for (float j = startPt.y; j < endPt.y; j++)
{
                       temp.x = size;
                       temp.y = j;
                       temp.z = i;
                       currPt = transfPipeline(temp.x,
temp.y, temp.z);

                       if (j >= (size / 2) + 4 && i >=
start_pnt + 10 && i <= size - 10
                                     && j <=
(size / 2) + 7)
                              drawPixel(currPt.x,
currPt.y, BLUE);
                       else if (j >= (size / 2) + 7 && j
<= (size) - 8 && i >= size - 15
```

```
                                        && i <= size
- 10)
                                drawPixel(currPt.x,
currPt.y, BLUE);
                        else if (j >= (size / 2) + 7 && j
<= (size) - 8
                                        && i >=
(size / 2) - 2 && i <= (size / 2) + 2)
                                drawPixel(currPt.x,
currPt.y, BLUE);
                        else if (j >= (size) - 10 && j
<= (size) - 6 && i >= (size / 2) - 2
                                        && i <=
(size) - 10)
                                drawPixel(currPt.x,
currPt.y, BLUE);
                }
        }
}

void draw3DCube(int cube_size)
{
   struct coordinates c1, c2, c3, c4, c5, c6, c7;
   int pt = 0;
   struct coordinate cubeDimension = {pt, pt, (cube_size
+ pt + elev)}; //(0,0,100)
   float Idiff1, Idiff2, Idiff3, Idiff4;
   struct coordinate t1, t2, t3, t4, t5, t6, t7;
   t1 = cubeDimension;
   Idiff1 = diffusedReflection(t1);
   c1 = transfPipeline(cubeDimension.x,
cubeDimension.y, cubeDimension.z);
   cubeDimension.x = (cube_size + pt); //(100,0,100)
   t2 = cubeDimension;
   Idiff2 = diffusedReflection(t2);
   c2 = transfPipeline(cubeDimension.x,
cubeDimension.y, cubeDimension.z);

   cubeDimension.y = (cube_size + pt); //(100,100,110)
   t3 = cubeDimension;
   Idiff3 = diffusedReflection(t3);
   c3 = transfPipeline(cubeDimension.x,
cubeDimension.y, cubeDimension.z + elev);

   cubeDimension.x = pt; //(0,100,100)
   t4 = cubeDimension;
   Idiff4 = diffusedReflection(t4);
   c4 = transfPipeline(cubeDimension.x,
cubeDimension.y, cubeDimension.z);

   cubeDimension.x = (cube_size + pt); //(100,0,0)
   cubeDimension.y = pt;
   cubeDimension.z = pt + elev;
   t5 = cubeDimension;
   c5 = transfPipeline(cubeDimension.x,
cubeDimension.y, cubeDimension.z);

   cubeDimension.y = (cube_size + pt); //(100,100,0)
   t6 = cubeDimension;
   c6 = transfPipeline(cubeDimension.x,
cubeDimension.y, cubeDimension.z);

   cubeDimension.x = pt; //(0,100,0)
   t7 = cubeDimension;
   c7 = transfPipeline(cubeDimension.x,
cubeDimension.y, cubeDimension.z);

   DDA_line(c1, c2, Idiff1, Idiff2);
   DDA_line(c2, c3, Idiff2, Idiff3);
   DDA_line(c3, c4, Idiff3, Idiff4);
   DDA_line(c4, c1, Idiff4, Idiff1);

   drawline(c2.x, c2.y, c5.x, c5.y, BLUE);
   drawline(c5.x, c5.y, c6.x, c6.y, BLUE);
   drawline(c6.x, c6.y, c3.x, c3.y, BLUE);
   drawline(c6.x, c6.y, c7.x, c7.y, BLUE);
   drawline(c7.x, c7.y, c4.x, c4.y, BLUE);

   fillside(t5, t3, cube_size, 1); //Left fill
   fillside(t7, t3, cube_size, 2); //right fill
   fillside(t1, t3, cube_size, 3); //top surface
}

float lambda_calc(float zi, float zs)
{
   return -zi / (zs - zi);
}

void drawShadow(double cube_size, struct coordinate ps)
{
   float l1, l2, l3, l4;
   struct coordinates s1, s2, s3, s4;
   struct coordinate xp1, xp2, xp3, xp4;
   int pt = 0;
   struct coordinate p1 = {pt, pt, (cube_size + pt + elev)};
//p1
   struct coordinate p2 = {(cube_size + pt), pt, (cube_size
+ pt + elev)};          //p2
   struct coordinate p3 = {(cube_size + pt), (cube_size +
pt), (cube_size + pt + elev)}; //p4
   struct coordinate p4 = {pt, (cube_size + pt), (cube_size
+ pt + elev)};          //p3
   struct coordinates psp = transfPipeline(ps.x, ps.y, ps.z);
   l1 = lambda_calc(p1.z, ps.z);
   l2 = lambda_calc(p2.z, ps.z);
   l3 = lambda_calc(p3.z, ps.z);
   l4 = lambda_calc(p4.z, ps.z);
   xp1.x = p1.x + l1 * (ps.x - p1.x);
   xp1.y = p1.y + l1 * (ps.y - p1.y);
```

```
    xp1.z = p1.z + l1 * (ps.z - p1.z);

    xp2.x = p2.x + l2 * (ps.x - p2.x);
    xp2.y = p2.y + l2 * (ps.y - p2.y);
    xp2.z = p2.z + l2 * (ps.z - p2.z);

    xp3.x = p3.x + l3 * (ps.x - p3.x);
    xp3.y = p3.y + l3 * (ps.y - p3.y);
    xp3.z = p3.z + l3 * (ps.z - p3.z);

    xp4.x = p4.x + l4 * (ps.x - p4.x);
    xp4.y = p4.y + l4 * (ps.y - p4.y);
    xp4.z = p4.z + l4 * (ps.z - p4.z);

    s1 = transfPipeline(xp1.x, xp1.y, xp1.z);
    s2 = transfPipeline(xp2.x, xp2.y, xp2.z);
    s3 = transfPipeline(xp3.x, xp3.y, xp3.z);
    s4 = transfPipeline(xp4.x, xp4.y, xp4.z);

    drawline(s1.x, s1.y, s2.x, s2.y, DARKBLUE);
    drawline(s2.x, s2.y, s3.x, s3.y, DARKBLUE);
    drawline(s3.x, s3.y, s4.x, s4.y, DARKBLUE);
    drawline(s4.x, s4.y, s1.x, s1.y, DARKBLUE);

    fillShadow(xp1, xp3);
}

void drawTree(uint32_t color, int startPoint, int size)
{
    int i = 0, angle;
    struct coordinates lcd;
    int tree_branch[3][3] = {{startPoint, startPoint + 40, 0.5
* size + 10}, {startPoint, startPoint + 40, 0.3 * size + 10},
{startPoint, startPoint + 40, 0.8 * size + 10}};
    while (i < 1)
    {
        int x0, y0, y1, x1, xp0, xp1, yp0, yp1;
        angle = startPoint + size;
        x0 = tree_branch[i][0];
        x1 = tree_branch[i][1];
        y0 = tree_branch[i][2] - 20;
        y1 = y0;
        i++;
        lcd = transfPipeline(y0, angle, x0);
        xp0 = lcd.x;
        yp0 = lcd.y;
        lcd = transfPipeline(y1, angle, x1);
        xp1 = lcd.x;
        yp1 = lcd.y;
        drawline(xp0, yp0, xp1, yp1, 0x69b4FF);
//level 0 straight line
        drawline((xp0 + 1), (yp0 + 1), (xp1 + 1), (yp1 + 1),
0x69b4FF); //level 0 straight line
        drawline((xp0 - 1), (yp0 - 1), (xp1 - 1), (yp1 - 1),
0x69b4FF); //level 0 straight line

        int it = 0;
        for (it = 0; it < 3; it++)
        {
            int16_t x2 = (0.6 * (x1 - x0)) + x1; // length of
level 1 = 0.8 of previous level
            int16_t y2 = y1;
            lcd = transfPipeline(y2, angle, x2);
            int xp2 = lcd.x;
            int yp2 = lcd.y;
            //                  drawline(xp1, yp1, xp2,
yp2,color);         //level 1 straight line

            //for right rotated angle 30 degree
            int16_t xr = ((0.134 * x1) + (0.866 * x2) - (0.5 *
y2) + (0.5 * y1));
            int16_t yr = ((0.5 * x2) - (0.5 * x1) + (0.866 * y2)
- (0.866 * y1) + y1);
            lcd = transfPipeline(yr, angle, xr);
            int xpr = lcd.x;
            int ypr = lcd.y;

            //for left rotated angle 30 degree
            int16_t xl = ((0.134 * x1) + (0.866 * x2) + (0.5 *
y2) - (0.5 * y1));
            int16_t yl = ((0.5 * x1) - (0.5 * x2) + (0.134 * y2)
+ (0.866 * y1));
            lcd = transfPipeline(yl, angle, xl);
            int xpl = lcd.x;
            int ypl = lcd.y;

            drawline(xp1, yp1, xpr, ypr, DARKGREEN);
            drawline(xp1, yp1, xpl, ypl, DARKGREEN);

            //for branches on right rotated branch angle 30
degree
            int16_t xrLen = sqrt(pow((xr - x1), 2) + pow((yr -
y1), 2)); //length of right branch
            int16_t xrImag = (0.8 * xrLen) + xr;
//imaginary vertical line x coordinate, y= yr
            int16_t xr1 = ((0.134 * xr) + (0.866 * xrImag) -
(0.5 * yr) + (0.5 * yr));
            int16_t yr1 = ((0.5 * xrImag) - (0.5 * xr) + (0.866
* yr) - (0.866 * yr) + yr);
            lcd = transfPipeline(yr1, angle, xr1);
            int xpr1 = lcd.x;
            int ypr1 = lcd.y;

            //for right branch
            int16_t xrr, xrl, yrr, yrl;
            xrr = ((0.134 * xr) + (0.866 * xr1) - (0.5 * yr1) +
(0.5 * yr));
            yrr = ((0.5 * xr1) - (0.5 * xr) + (0.866 * yr1) -
(0.866 * yr) + yr);
            lcd = transfPipeline(yrr, angle, xrr);
            int xprr = lcd.x;
            int yprr = lcd.y;

            //for left branch
            xrl = ((0.134 * xr) + (0.866 * xr1) + (0.5 * yr1) -
(0.5 * yr));
            yrl = ((0.5 * xr) - (0.5 * xr1) + (0.134 * yr) +
(0.866 * yr1));
            lcd = transfPipeline(yrl, angle, xrl);
            int xprl = lcd.x;
            int yprl = lcd.y;
            //for branches on left rotated branch angle 30
degree
            int16_t xlImag = (0.8 * xrLen) + xl; //imaginary
vertical line x coordinate, y= yr
            int16_t xl1 = ((0.134 * xl) + (0.866 * xlImag) +
(0.5 * yl) - (0.5 * yl));
            int16_t yl1 = ((0.5 * xl) - (0.5 * xlImag) + (0.134
* yl) + (0.866 * yl));
            lcd = transfPipeline(yl1, angle, xl1);
            int xpl1 = lcd.x;
            int ypl1 = lcd.y;
            //for right branch
            int16_t xlr, xll, ylr, yll;
            xlr = ((0.134 * xl) + (0.866 * xl1) - (0.5 * yl1) +
(0.5 * yl));
            ylr = ((0.5 * xl1) - (0.5 * xl) + (0.866 * yl1) -
(0.866 * yl) + yl);
            lcd = transfPipeline(ylr, angle, xlr);
            int xplr = lcd.x;
            int yplr = lcd.y;
            //for left branch
            xll = ((0.134 * xl) + (0.866 * xl1) + (0.5 * yl1) -
(0.5 * yl));
            yll = ((0.5 * xl) - (0.5 * xl1) + (0.134 * yl) +
(0.866 * yl1));
            lcd = transfPipeline(yll, angle, xll);
            int xpll = lcd.x;
            int ypll = lcd.y;
            drawline(xpr, ypr, xpr1, ypr1, DARKGREEN);
            drawline(xpr, ypr, xprr, yprr, DARKGREEN);
            drawline(xpr, ypr, xprl, yprl, DARKGREEN);
            drawline(xpl, ypl, xpl1, ypl1, DARKGREEN);
            drawline(xpl, ypl, xplr, yplr, DARKGREEN);
            drawline(xpl, ypl, xpll, ypll, DARKGREEN);

            x0 = x1;
            x1 = x2;
        }
    }
}

struct coordinates project_coordinates (int x_w, int y_w,
int z_w)
{
            int scrn_x, scrn_y, Dist=100, x_diff=74,
y_diff=50;
            double x_p, y_p, z_p, theta, phi, rho;
            struct coordinates 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_HorizontalLine(int16_t x, int16_t y, int16_t
width, uint32_t color)
{
    drawline(x, y, x + width - 1, y, color);
}

int main(void)
{
    uint32_t portnum = PORT_NUM;
    if (portnum == 0)
        SSP0Init();
    else if (portnum == 1)
        SSP1Init();

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

    lcd_init();
    drawBackground(0, 0, _width, _height, WHITE);
    drawCoords();
    int startPoint = 0;
    int size = 100;
    double x[8] = {startPoint, (startPoint + size), (startPoint
+ size), startPoint, startPoint, (startPoint + size),
(startPoint + size), startPoint};
```

```
    double y[8] = {startPoint, startPoint, startPoint + size,
startPoint + size, startPoint, startPoint, (startPoint + size),
(startPoint + size)};
    double z[8] = {startPoint, startPoint, startPoint,
startPoint, (startPoint + size), (startPoint + size),
(startPoint + size), (startPoint + size)};

    struct coordinate pointLightSource = {0, 0, 235};
    drawCoords();
    drawShadow(size, pointLightSource);
    draw3DCube(size);

    struct coordinate t8, t9;
    t8.x = 100;
    t8.y = 0;
    t8.z = 0;

    t9.x = 100;
    t9.y = 100;
    t9.z = 100;

    lcddelay(10);
    diffusedReflection(pointLightSource);
    drawTree(0x0066CC00, startPoint, size);
    draw_J(t8, t9, size);
    draw_P(t8, t9, size);

    printf("\nName: Jay Jatinkumar Patel");
    printf("\nStudent ID: 015216357");
    printf("\nClass: CMPE 240 3D Graphics");

    return 0;
}
```