Shading and Diffuse Reflection for Advanced Microprocessor 3D GE Design

Rutvik Girishkumar Patel, ID - 014557530

*Computer Engineering Department, College of Engineering San Jose State University, San Jose, CA 95192*

*E-mail:* [rutvik.g.patel@sjsu.edu](mailto:rutvik.g.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 floating 3D cube with 2D graphics on two of its frontal faces and a halfsphere with a solid “S” on it with diffuse reflection computation and shading* *has been implemented by communicating with SPI bus of LPC1769 module.*

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

# Introduction

The LPC1769 microcontroller is a 32-bit microcontroller based on the ARM cortex M3 processor. It is commonly utilized in embedded inclined applications that require great integration while dissipating little power. It has a few characteristics, including a 120MHZ CPU clock, 512 kB flash memory, and 64 kB data memory. Some of the peripherals supported by the CPU are 3 SSP/SPI, 2 CAN channels, 4 UARTS, an 8 channel 12-bit ADC, and a 10-bit DAC. It also has up to 70 general purpose I/O pins.

To meet the above-mentioned purpose, we constructed a prototype board for this project using the mentioned components, including an LPC 1769 CPU module and a 1.8" ST7735R TFT LCD. The CPU module is powered by a USB cable, and the above-mentioned display is programmed in C++ using the MCUxpresso IDE. The purpose of this project is to produce a revolving square pattern and a 2D tree pattern with a maximum level of 10 using the vector graphics formula.

# Methodology

The following section goes over the system layout, flow chart, implementation, and hardware and software setup.

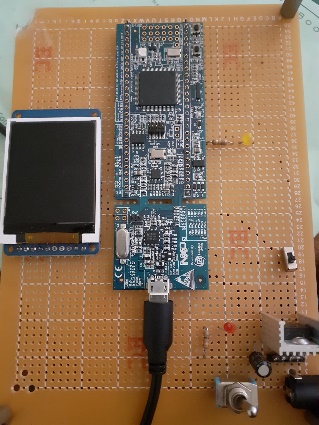
# objectives and technical challenges

The goals of this project are as follows:

1. Include the CPU module and LCD display in order to produce a prototype circuit board.
2. Understand the notion of producing 3-D vector graphics using the CPU module.
3. Understand and realize the Special Purpose Registers used in the communication protocol.
4. Successful debugging and algorithm execution on the IDE.

The problems occurred to us during implementation

* 1. Making proper connections by soldiering took time.



**Figure 1**: Prototype board

# Problem Formulation and Design

Diagram

Description automatically generated

**Figure 2**: System Design

The above diagram refers to the system design.

This project aims to implement project 3D shading model and diffuse reflection computation.

* Generating a solid cube
* Generating a half sphere
* Generating trees on two frontal faces of the cube.
* Computing diffuse reflection on the top surface of the cube.
* Producing a shade of cube based on the light source.
* Decorating half sphere by placing letter S

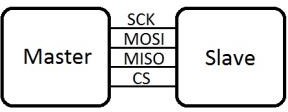
# Implementation

This section describes how to design and build an SPI interface hardware component. The overall design can be broken down into pieces, which are discussed further in the Hardware Design section.

# Hardware Design

* + 1. **System Block Diagram**

The diagram represents the block diagram of the system:



**Figure 3**: Block Diagram

# Schematic design

When working with the LPC1769, the first thing we need to do is build a power regulator circuit. The LPC requires 5V to work, thus we need a voltage regulator, an LM 7805 voltage regulator IC, a capacitor, and a switch.The table below provides a summary of the link between the LPC and the LCD.

Diagram, schematic

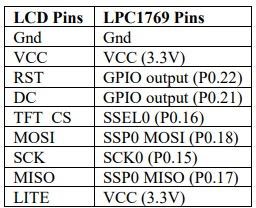
Description automatically generated

**Figure 4**: Schematic design of the prototype of the board

A close-up of a computer chip

Description automatically generated with medium confidence

**Figure 5** : LCD pin description



**Table 1** : Pin Description

**Table

Description automatically generated**

**Table 2**. Circuit level connections between LPC and LCD

Diagram, schematic

Description automatically generated

**Figure 6**: TFT LCD Display and LPC 1769 Connections

|  |  |
| --- | --- |
| 1 | Wire Wrapping Board |
| 2 | LPC 1769 |
| 3 | LPC TFT ST7735R |
| 4 | USB cable for power supply |
| 5 | Laptop |
| 6 | Resistors |
| 7 | Wires for connections |
| 8 | Power Regulator IC 708 |

**Table 3**. Circuit level connections between LPC and LCD

# Bill of materials

The resources used to build the project are mentioned below:

The CPU display serves as the primary component in this case. The circuit should supply 5V to the LPC 1769. The LPC display functions as a slave and is connected to MISO, MOSI, and SCK pins. The ST7735R, a single chip driver for a 262K-color graphic type TFT-LCD, is employed. The chip takes SPI and is directly linked to an external CPU. The display resolution is 160\*128 pixels, and three primary colors are employed for representation: red, green, and blue. The implementation is divided into two components:

Software and Hardware.

# Software Design

We have used open-source software MCUXpresso IDE for the purpose. The code is written in C++.

# Flow Chart

# 

# Figure 7: Flow chart of system

# Algorithm Description

A top-level view of the approach for implementation of

3D graphics, shading and diffuse reflection are

mentioned below.

* + - 1. Initialize SPI communication between LPC1769 and LCD
      2. Draw the world coordinate system.
      3. Find 2D points for floating 3D cube using 3D-2D transformations and perspective projection. The points must be chosen so that the cube rotated by 10 degree clockwise. The rotation of cube is achieved with respect to vector parallel to the Y axis and going through the center of the cube.
      4. Set a point for source of light.
      5. Find the intersection points of the light source and cube surface and XY plane.
      6. Draw the shadow of the cube.
      7. Calculate points of half sphere surface.
      8. Find 2D points for the half sphere using 3D-2D transformations around the origin of xw-yw-zw plane.
      9. Decorating the half sphere by placing letter S
      10. Draw the cube in XYZ plane of LCD.
      11. Perform compute diffuse reflection on the top surface of the cube.
      12. On the face of the cube on the XZ plane implement the 2D growing trees screensaver.

# Implementation

* + - 1. Tree generation: In this step, we generate trees at random locations to create a forest. Using the rand() function, the branches and trees are generated in random fashion. P(x,y) = P\_1(x\_1,y\_1) + lambda\*(P\_2(x\_2,y\_2) – P\_1(x\_1,y\_1)) is the main formula utilized here.

# Pseudo code

3D transformation matrix

Ɵ = Cos-1 (Xeye / √(Xeye2+Yeye2))

ɸ = Cos-1 (Zeye / √(Xeye2+Yeye2+Zeye2))

ρ = √(Xeye2+Yeye2+Zeye2)

Xprime = YWorld\*Cos Ɵ – Xworld\*Sin Ɵ

Yprime = ZWorld\*Sin ɸ - Cos Ɵ Cos ɸ \*(XWorld-YWorld)

Zprime = ρ - Cos Ɵ Sin ɸ (YWorld-XWorld) – Zworld\* Cos ɸ

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

# int round(float number){}

# void drawVLine(int16\_t x, int16\_t y, int16\_t h, uint32\_t color){}

# void drawRect(int16\_t x, int16\_t y,int16\_t w, int16\_t h,uint32\_t color) {}

# void draw\_fast\_h\_line(int16\_t x, int16\_t y,int16\_t w, uint32\_t color){}

# void draw\_fast\_v\_line(int16\_t x, int16\_t y,int16\_t h, uint32\_t color){}

# void drawBackground(int16\_t x, int16\_t y, int16\_t w, int16\_t h, uint32\_t color){}

# struct coordinates transfPipeline(float xw, float yw, float zw){}

# struct coordinates transfPipelineRotate(float xw1, float yw1, float zw1){}

# struct coordinate rotatePoints(struct coordinate pt){}

# float lambda\_calc(float zi, float zs){}

# void drawCoords(){}

# struct coordinates Fx(float u, float r, float z){}

# void drawSphere(uint32\_t color)

# {// Function to draw sphere}

# void drawShadowSphere(int size,struct coordinates arr[size])

# {// Function to draw shadow for sphere}

# void draw\_S(int sizeM, int sizeN, struct coordinates xPos[sizeM][sizeN], struct coordinates yPos[sizeM][sizeN], uint32\_t color)

# {// Function to draw letter S}

# float diffusedReflection(struct coordinate Pi)

# {// Function to perform diffused reflection}

# uint32\_t findColor(float x, float y, float Idiff1, float Idiff2, struct coordinates p1, struct coordinates p2){}

# void DDA\_line(struct coordinates p1, struct coordinates p2, float Idiff1, float Idiff2){}

# void fillside(struct coordinate startPt, struct coordinate endPt, int size, int side, struct coordinate defaultPt)

# {// Function to fill side}

# void drawTree(uint32\_t color,int start\_pnt, int size)

# {// Function to draw tree}

# void draw3DCube(struct coordinate cornerPt, int cube\_size, uint32\_t color)

# {// Function to draw 3D Cube}

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

# {// Function to fill shadow}

# void drawShadow(double cube\_size, struct coordinate ps,struct coordinate cornerPt)

# {//Function to draw shadow}

# void draw\_HorizontalLine(int16\_t x, int16\_t y, int16\_t width, uint32\_t color){}

# int main(void)

# {

# 

# drawShadow(size,Ps,cornerPt);

# drawTree(0x0066CC00, 50, size);

# drawSphere(BLUE);

# draw3DCube(cornerPt,size,BLUE);

# fillShadow()

# diffusedReflection()

# 

# }

# Testing and Verification

Testing and verification are critical for comparing actual outcomes to projected results.

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

* + - 1. Open 3D\_Engine project in the IDE.
      2. Open 3D-Engine.c file.
      3. Connect the LPC 1769 board to your computer with a micro-USB cable.
      4. Build the project by clicking the hammer button. Observe the console’s output. Make sure there are no errors.
      5. Click the debug button. It’s the blue bug button. A window will pop up asking you to select your LPC-Link.
      6. Once selected, code is loaded on the target and Eclipse will switch to a debug perspective.
      7. Click the resume (F8) button to execute the code.

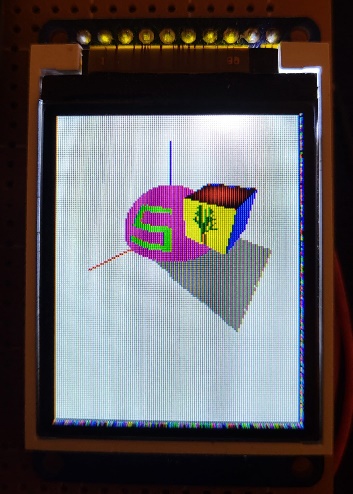
# Testing

The following test scenarios were utilized in the lab:

1. Ensure that the LCD in this project is operational.
2. Ensuring that the soldering is done appropriately.
3. Verifying that the needed 3D images are properly rendered on the LCD pane
4. The project was built and debugged successfully without any errors with successful link connection to the CPU module.
5. A floating 3D cube of size 100 rotated 5 degrees clockwise was drawn on XYZ plane.
6. Ensure a half sphere is visualized around the origin.
7. Verify that the sides of cube and half sphere are filled resulting in solid figures.
8. Verify that the screen savers can be seen on frontal sides of the cube.
9. Ensure deflection is performed on the top of the sphere.
10. Ensure the shadow of cube is drawn filled with solid gray color.
11. Ensure an initial S is decorated on the half sphere.

# Verification

The LCD display's output was tested with a variety of inputs, and the findings were rectified.

****

**Figure 8** : 3D shading model and diffuse reflection

From the figure ,we can verify that a solid cube is drawn with its front side a tree screen saver is visualized, and on top a deflection is performed. Around the origin we can see that a solid half sphere is drawn with initials “S” written over it. We can see that a shadow is generated based on the light source and is filled with solid gray color.

# Conclusion

The creation and execution of 3D graphics for LCD displays are successful. We may infer from this project that utilizing a set of matrices and a fundamental line-point formula, we can develop an advanced feature of a 3D graphics engine. It is crucial to read the user manual and datasheets in their entirety. Implementing the interface requires a thorough understanding of the 3D transformation matrix, perspective projection, and flood fill method.

# Acknowledgement

Prof. Dr. Harry Li's leadership and consistent efforts ensured the success of this project. I'd also like to thank the other members of my group for their help and support.

# References

1. H. Li, *Lecture Notes of CMPE 240*, Computer Engineering Department, College of Engineering, San Jose State University.
2. NXP, “UM10360 user manu.pdf”, UM10360LPC176x/5x User manual.

# Appendix

// Course: CMPE - 240

// Submitted by: Rutvik Patel

#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 0xFFFFFF

#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

#define BEBLUE 0XD30C7B

float rad = 150;

struct coordinates

{

float x;

float y;

};

struct coordinate

{

float x;

float y;

float z;

};

int xcamera = 250, ycamera = 250, zcamera = 200;

int elev = 50;

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;

double rotateAngle = -0.174533;

struct coordinates cubeCenterPoint = {0,100};

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 drawRect(int16\_t x, int16\_t y,

int16\_t w, int16\_t h,

uint32\_t color) {

draw\_fast\_h\_line(x, y, w, color);

draw\_fast\_h\_line(x, y+h-1, w, color);

draw\_fast\_v\_line(x, y, h, color);

draw\_fast\_v\_line(x+w-1, y, h, color);

}

void draw\_fast\_h\_line(int16\_t x, int16\_t y,

int16\_t w, uint32\_t color) {

drawline(x, y, x+w-1, y, color);

}

void draw\_fast\_v\_line(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);

}

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(float xw, float yw, float zw)

{

int xDPrime, yDPrime, D = 60, l1 = 60, l2 = 60;

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;

}

struct coordinates transfPipelineRotate(float xw1, float yw1, float zw1)

{

float xw = ((xw1-cubeCenterPoint.x) \* cos(rotateAngle)) - ((zw1-cubeCenterPoint.y) \* sin(rotateAngle)) + cubeCenterPoint.x;

float zw = ((xw1-cubeCenterPoint.x) \* sin(rotateAngle)) + ((zw1-cubeCenterPoint.y) \* cos(rotateAngle)) + cubeCenterPoint.y;

float yw = yw1;

int xDPrime, yDPrime, D = 60, l1 = 60, l2 = 60;

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;

}

struct coordinate rotatePoints(struct coordinate pt){

float xw = ((pt.x-cubeCenterPoint.x) \* cos(rotateAngle)) - ((pt.z-cubeCenterPoint.y) \* sin(rotateAngle)) + cubeCenterPoint.x;

float zw = ((pt.x-cubeCenterPoint.x) \* sin(rotateAngle)) + ((pt.z-cubeCenterPoint.y) \* cos(rotateAngle)) + cubeCenterPoint.y;

float yw = pt.y;

struct coordinate res = {xw,yw,zw};

return res;

}

float lambda\_calc(float zi, float zs)

{

return (-zi / (zs - zi));

}

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, GREEN);

drawline(x1, y1, x4, y4, BLUE);

}

struct coordinate Fx(float u, float r, float z){

float xw = cos(u)\*r;

float yw = sin(u)\*r;

float zw = z;

struct coordinate res = {xw,yw,zw};

return res;

}

void drawSphere(uint32\_t color){

float PI = 3.141;

float endU=PI\*2;

float inc = 0.02; //PI / 20;

int sizeM = 12;

int sizeN = 40;

int shadowArrSize = 100;

struct coordinate shadowArr[shadowArrSize];

struct coordinates xPos[sizeM][sizeN];

struct coordinates yPos[sizeM][sizeN];

for(int i =0;i<sizeM;i++){

for(int j = 0;j<sizeN;j++){

struct coordinates temp = {0,0};

xPos[i][j] = temp;

yPos[i][j] = temp;

}

}

int counter = 0;

int counterY1 = 0;

int counterY2 = 0;

float angle\_1 = 0, angle\_2 = 0;

float angle\_3 = 0, angle\_4 = 0;

int shadowCounter = 0;

for(float alpha = inc; alpha < ((PI/2)); alpha += inc){

struct coordinates arr1[40];

struct coordinates arr2[40];

float r = sin(alpha) \* rad;

float z = cos(alpha) \* rad;

float r1 = sin(alpha+inc) \* rad;

float z1 = cos(alpha+inc) \* rad;

int a = 0;

for(float i=0;i<endU;i+=inc){

struct coordinate temp\_3d\_pt = Fx(i,r,z);

struct coordinates temp1 = transfPipeline(temp\_3d\_pt.x,temp\_3d\_pt.y,temp\_3d\_pt.z);

struct coordinates currPt;

drawPixel(temp1.x, temp1.y, color);

if( counter > 0 && counter < 5 && a < 40){

if(a == 34) angle\_1 = i;

if(a==39) angle\_2 = i;

xPos[counter-1][a] = temp1;

a++;

}

if( counter > 10 && counter < 15 && a < 40){

if(a == 34) angle\_3 = i;

else if(a==39) angle\_4 = i;

xPos[counter-7][a] = temp1;

a++;

}

if( counter > 20 && counter < 25 && a < 40){

xPos[counter-13][a] = temp1;

a++;

}

if(counter == 5 && i >= 0 && i <= 0.1){

int yc= (i - angle\_1) \* 100;

yPos[counterY1][counterY2] = temp1;

counterY2++;

if(i == 0.1) counterY1++;

}

if (counter == 15 && i >= angle\_3 && i <= angle\_4){

int yc= (i - angle\_3) \* 100;

yPos[counterY1][counterY2] = temp1;

counterY2++;

if(i == angle\_4) counterY1++;

}

}

if(alpha == 0.66 ) counter = 1;

else if(alpha > 0.66 && alpha < 0.76 && counter < 10) counter++;

else if(alpha >= 0.96 && counter < 11) counter = 11;

else if(alpha > 0.96 && alpha < 1.06 && counter < 20) counter++;

else if(alpha >= 1.26 && counter < 21 ) counter = 21;

else if(alpha > 1.26 && alpha < 1.36) counter++;

}

if(counter > 0) draw\_S(sizeM,sizeN,xPos,yPos,GREEN);

}

void drawShadowSphere(int size,struct coordinate arr[size]){

struct coordinate Ps = {0, 150, 250};

for(int i =0;i<size;i++){

if(arr[i].x==0 && arr[i].y==0 && arr[i].z==0) break;

float lt = lambda\_calc(arr[i].z,Ps.z);

struct coordinate maxShadowPt;

maxShadowPt.x = arr[i].x + lt \* (Ps.x - arr[i].x);

maxShadowPt.y = arr[i].y + lt \* (Ps.y - arr[i].y);

maxShadowPt.z = arr[i].z + lt \* (Ps.z - arr[i].z);

struct coordinates temp1 = transfPipeline(maxShadowPt.x,maxShadowPt.y,maxShadowPt.z);

struct coordinates temp2 = transfPipeline(maxShadowPt.x,0,0);

drawline(temp1.x,temp1.y,temp2.x,temp2.y,SHADOW);

}

}

void draw\_S(int sizeM, int sizeN, struct coordinates xPos[sizeM][sizeN], struct coordinates yPos[sizeM][sizeN], uint32\_t color){

for(int i =0;i<sizeM;i++){

for(int j = 0;j<sizeN;j++){

struct coordinates temp = {0,0};

drawPixel(xPos[i][j].x,xPos[i][j].y,color);

drawPixel(yPos[i][j].x,yPos[i][j].y,color);

}

}

}

float diffusedReflection(struct coordinate Pi)

{

struct coordinate Ps = {0, 150, 250};

float red;

float scaling = 20000, 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 \*0.9 \* temp);

return red;

}

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 fillside(struct coordinate startPt, struct coordinate endPt, int size, int side, struct coordinate defaultPt)

{

struct coordinate temp;

struct coordinates currPt;

if (side == 1)

{

for (float i = startPt.z; i < endPt.z; i+=3)

{

for (float j = startPt.y; j < endPt.y; j++)

{

temp.x = defaultPt.x;

temp.y = j;

temp.z = i;

currPt = transfPipelineRotate(temp.x, temp.y, temp.z);

drawPixel(currPt.x, currPt.y, YELLOW);

}

}

}

else if (side == 2)

{

for (float i = startPt.z; i < endPt.z; i+=3)

{

for (float j = startPt.x; j < endPt.x; j++)

{

temp.x = j;

temp.y = defaultPt.y;

temp.z = i;

currPt = transfPipelineRotate(temp.x, temp.y, temp.z);

drawPixel(currPt.x, currPt.y, BLUE);

}

}

}

else if (side == 3)

{

uint32\_t color;

float r, g = 0, b = 0;

for (float i = startPt.y; i < endPt.y; i+=3)

{

for (float j = startPt.x; j < endPt.x; j++)

{

temp.x = j;

temp.y = i;

temp.z = defaultPt.z;

r = diffusedReflection(temp);

g = 0;

b = 0;

color = (((uint32\_t)round(r)) << 16) + (((uint32\_t)g) << 8) + ((uint32\_t)b);

currPt = transfPipelineRotate(temp.x, temp.y, temp.z);

drawPixel(currPt.x, currPt.y, color);

}

}

}

}

void drawTree(uint32\_t color,int start\_pnt, int size)

{

int i=0, angle;

struct coordinates lcd;

int tree\_branch[3][3]={{start\_pnt,start\_pnt+30,0.5\*size+100},{start\_pnt+10,start\_pnt+40,0.3\*size+100},{start\_pnt+10,start\_pnt+40,0.8\*size+100}};

while(i<1)

{

int x0, y0, y1, x1,xp0,xp1,yp0,yp1;

angle = start\_pnt;

x0=tree\_branch[i][0];

x1=tree\_branch[i][1];

y0=tree\_branch[i][2];

y1=y0;

i++;

lcd = transfPipelineRotate(angle,y0,x0);//lcd = transfPipeline(y0,angle,x0);

xp0=lcd.x;

yp0=lcd.y;

lcd = transfPipelineRotate(angle,y1,x1);

xp1=lcd.x;

yp1=lcd.y;

drawline(xp0, yp0, xp1, yp1,0x00FF8000); //level 0 straight line

drawline((xp0+1), (yp0+1), (xp1+1), (yp1+1),0x6A4B19); //level 0 straight line

drawline((xp0-1), (yp0-1), (xp1-1), (yp1-1),0x6A4B19); //level 0 straight line

int it=0;

for(it=0;it<3;it++){

int16\_t x2=(0.8\*(x1-x0))+x1; // length of level 1 = 0.8 of previous level

int16\_t y2=y1;

lcd = transfPipelineRotate(angle,y2,x2);

int xp2=lcd.x;

int yp2=lcd.y;

drawline(xp1, yp1, xp2, yp2,color); //level 1 straight line

drawRect(0,0,127,159,0x00000000);

//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 = transfPipelineRotate(angle,yr,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 = transfPipelineRotate(angle,yl,xl);

int xpl=lcd.x;

int ypl=lcd.y;

drawline(xp1, yp1, xpr, ypr,color);

drawline(xp1, yp1, xpl, ypl,color);

drawRect(0,0,127,159,0x00000000);

//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 = transfPipelineRotate(angle,yr1,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 = transfPipelineRotate(angle,yrr,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 = transfPipelineRotate(angle,yrl,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 = transfPipelineRotate(angle,yl1,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 = transfPipelineRotate(angle,ylr,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 = transfPipelineRotate(angle,yll,xll);

int xpll=lcd.x;

int ypll=lcd.y;

drawline(xpr, ypr, xpr1, ypr1,color);

drawline(xpr, ypr, xprr, yprr,color);

drawline(xpr, ypr, xprl, yprl,color);

drawline(xpl, ypl, xpl1, ypl1,color);

drawline(xpl, ypl, xplr, yplr,color);

drawline(xpl, ypl, xpll, ypll,color);

x0=x1;

x1=x2;

}

drawRect(0,0,127,159,0x00000000);

}

}

void draw3DCube(struct coordinate cornerPt, int cube\_size, uint32\_t color)

{

struct coordinates c1, c2, c3, c4, c5, c6, c7;

struct coordinate cubeDimension = cornerPt; //(0,50,50)

float Idiff5, Idiff6, Idiff3, Idiff4;

struct coordinate p5,p6,p3,p4;

struct coordinate FillCord = {cubeDimension.x,cubeDimension.y+cube\_size,cubeDimension.z+cube\_size};

c1 = transfPipelineRotate(cubeDimension.x, cubeDimension.y, cubeDimension.z);

struct coordinate t1 = {cubeDimension.x, cubeDimension.y, cubeDimension.z};

c2 = transfPipelineRotate(cubeDimension.x, cubeDimension.y + cube\_size, cubeDimension.z);

c3 = transfPipelineRotate(cubeDimension.x, cubeDimension.y + cube\_size, cubeDimension.z + cube\_size);

struct coordinate t3 = {cubeDimension.x, cubeDimension.y + cube\_size, cubeDimension.z + cube\_size};

p3 = rotatePoints(t3);

Idiff3 = diffusedReflection(p3);

c4 = transfPipelineRotate(cubeDimension.x, cubeDimension.y, cubeDimension.z + cube\_size);

struct coordinate t4 = {cubeDimension.x, cubeDimension.y, cubeDimension.z + cube\_size};

p4 = rotatePoints(t4);

Idiff4 = diffusedReflection(p4);

c5 = transfPipelineRotate(cubeDimension.x - cube\_size, cubeDimension.y, cubeDimension.z + cube\_size);

struct coordinate t5 = {cubeDimension.x - cube\_size, cubeDimension.y, cubeDimension.z + cube\_size};

p5 = rotatePoints(t5);

Idiff5 = diffusedReflection(p5);

c6 = transfPipelineRotate(cubeDimension.x - cube\_size, cubeDimension.y + cube\_size, cubeDimension.z + cube\_size);

struct coordinate t6 = {cubeDimension.x - cube\_size, cubeDimension.y + cube\_size, cubeDimension.z + cube\_size};

p6 = rotatePoints(t6);

Idiff6 = diffusedReflection(p6);

c7 = transfPipelineRotate(cubeDimension.x - cube\_size, cubeDimension.y + cube\_size, cubeDimension.z);

struct coordinate t7 = {cubeDimension.x - cube\_size, cubeDimension.y + cube\_size, cubeDimension.z};

DDA\_line(c5, c4, Idiff5, Idiff4);

DDA\_line(c4, c3, Idiff4, Idiff3);

DDA\_line(c3, c6, Idiff3, Idiff6);

DDA\_line(c6, c5, Idiff6, Idiff5);

fillside(t5, t3, cube\_size, 3, FillCord);

fillside(t7, t3, cube\_size, 2, FillCord);

fillside(t1, t3, cube\_size, 1, FillCord);

}

void fillShadow(struct coordinate startPt, struct coordinate endPt)

{

struct coordinate temp;

struct coordinates currPt;

for (float i = startPt.y; i < endPt.y; i+=2)

{

for (float j = startPt.x; j < endPt.x; j+=2)

{

temp.x = j;

temp.y = i;

temp.z = 0;

currPt = transfPipeline(temp.x, temp.y, temp.z);

drawPixel(currPt.x, currPt.y, SHADOW);

}

}

}

void drawShadow(double cube\_size, struct coordinate ps,struct coordinate cornerPt)

{

float l1, l2, l3, l4;

struct coordinates s1, s2, s3, s4;

struct coordinate xp1, xp2, xp3, xp4;

int pt = 0;

struct coordinate pp1 = {cornerPt.x-cube\_size,cornerPt.y,cornerPt.z+cube\_size};

struct coordinate p1 = rotatePoints(pp1);//p1

struct coordinate pp3 = {cornerPt.x , cornerPt.y+cube\_size , cornerPt.z+cube\_size}; //p4

struct coordinate p3 = rotatePoints(pp3); //p4

struct coordinates psp = transfPipeline(ps.x, ps.y, ps.z);

l1 = lambda\_calc(p1.z, ps.z);

l3 = lambda\_calc(p3.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);

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);

fillShadow(xp1, xp3);

}

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 size = 100;

struct coordinate cornerPt = {50, 100, 50};

struct coordinate Ps = {0, 150, 250};

drawShadow(size,Ps,cornerPt);

drawSphere(BEBLUE);

draw3DCube(cornerPt,size,BLUE);

drawTree(DARKGREEN, 50, size);

return 0;

}