3D Cube Shading Model using LPC1769

AKHIL CHERUKURI
Computer Engineering Department
San Jose State University
San Jose, California
akhil.cherukuri@sjsu.edu

***Abstract*—The primary objective of the lab is to perform interfacing of the peripheral LCD Display module to the NXP LPC1769 LPCXpresso Board utilizing the Serial Peripheral Interface. The Program is designed in such way that the LPC1769 controls the pixel content displayed on the LCD. The program is implemented in 'C' programming language which displays 3D cube with a light source and a shadow on the LCD with Diffuse Reflection Shading Model. Successful validation of 3D graphics processing engine design is shown on the LPC1769 LCD Module.**

I. Introduction

The main objective of this lab is to design and build a microprocessor unit that and control the LCD display using SPI Interface. The program code is written in 'C' language on an IDE and compiled. The compiled code is loaded on to the LPC1769 memory and is executed. 3D Graphics Processing Engine with shading model and diffuse reflection computation will be implemented and the 3D objects defined in the code is seen as the output on the LCD node.

This paper concentrates on providing detailed description of the software and hardware methodology implemented in the lab project. Along with this, it also includes details on the testing and verification. Diagrams and Images are also provided when needed to show a clear perspective or output.

The LPC module is connected to the computers USB port to get the required power supply. The LCD module draws in 3.3v from the LPC in built power supply. In LPC1769, pin numbers 5, 7, 8 are used for communication as we are using SPI port number 1 for our project code. The LCD peripheral is connected via appropriate pins with the LPC module's SPI port for communication.

II. Methodology

In this section, the system layout with hardware and software design is explained along with any technical difficulties and solutions encountered while developing the circuit or implementation of the program.

A. Objectives

The primary objective of the lab is to display the 2D and 3D images on the LCD with the help form LPC1769. It includes:

- Getting familiarized with the LPCXpresso IDE to test the LCD program that yields the result as expected.
- Getting familiarized with the datasheets of NXP LPC1769 and Adafruit ST7735R LCD Module.
- LCD module to SPI interface on the LPC1769.
- Connect the pins between the LPC1769 and LCD module and
- Understanding the pin functionalities of LPC1769 and LCD module, especially SPI pins that enable to display images on the LCD.
- Understanding LCD hardware functionality of displaying images.
- Thorough understanding of 2D and 3D graphics processing engine design.
- 'C' language is required for programming the LPC1769 which controls the LCD module.
- Synchronization establishment between the LCD and LPCXpresso 1769 board.
- Debugging capability.

Technical challenges faced while building the project. They are as follows:

- Displaying the images on the LCD Screen by computing the equations needed.
- Debugging Hard and Soft faults

B. System Layout

The power is supplied from the computer to LPC Module via USB Cable. The USB Cable is also used to Flash and Debug the system. The LCD Display is connected to the LPC GPIO Ports via connecting wires using SPI Interface. The LCD Display Module is powered from the VCC and GND pins from the LPC1769 Module.

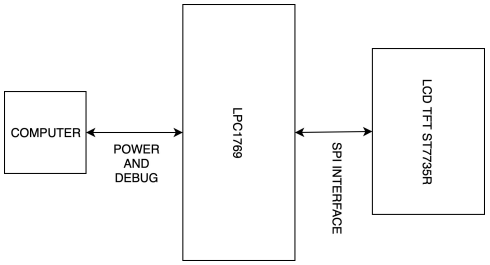

Fig. 1. System layout

To complete this lab, we needed the following components:

- NXP LPC1769
- 1.8” Color TFT LCD Display by Adafruit
- 6” x 5” Prototype Board
- PC with MCUXpresso IDE

C. Circuit Design

LPCXpresso module is connected to the 1.8” 18 bit color TFT(Thin Film Transistor) LCD display to ensure the functionality working as expected. This module is used to display images and shapes of geometric figures. It uses 4-wire SPI to communicate and has its own pixel addressable frame buffer. The resolution of the LCD display is 128 by 160. It includes a 3.3V regulator for low voltage drop out and 3/5 level shifter circuit so that we can have input power logic of 3V or 5V. The LPC module and LCD display are built on the wire wrapping board and are connected to the required pin via connecting wires. Once the required hardware is built, MCUXpresso IDE Version 11 is utilized to program the LPC module.

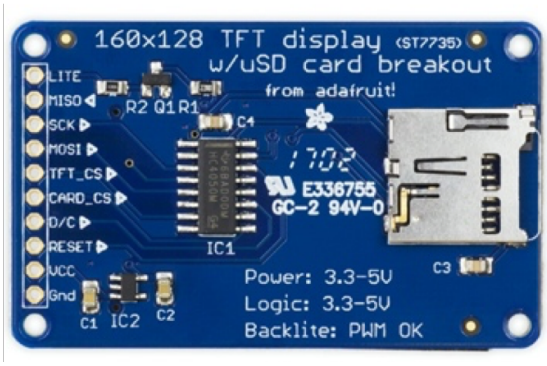

Fig. 2. 160x128 TFT Display with ST7735R

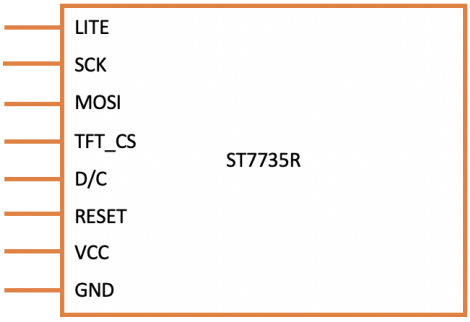

Fig. 3. LCD Pinout

1) ST7735R Driver for LCD: The ST7735R is a single-chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source line and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI), 8, 12, 16, and 18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits.

LPCXpresso		LPCXpresso	
VOUT (+3.3V out) if self powered, else +3.3V input		GND	
not used		VIN (4.5-5.5V)	
not used		VB (battery supply)	
not used		RESET_N	
RD-		P0.9	MOSI1
RD+		P0.8	MISO1
TD-		P0.7	SCK1
TD+		P0.6	SSEL1
USB-D-		P0.0	TXD3/SDA1
USB-D+		P0.1	RXD3/SCL1
P0.4	CAN_RX2	P0.18	MOSI0
P0.5	CAN_TX2	P0.17	MISO0
P0.10	TXD2/SDA2	P0.15	TXD1/SCK0
P0.11	RXD2/SCL2	P0.16	RXD1/SSEL0
P2.0	PWM1.1	P0.23	AD0.0
P2.1	PWM1.2	P0.24	AD0.1
P2.2	PWM1.3	P0.25	AD0.2
P2.3	PWM1.4	P0.26	AD0.3/AOUT
P2.4	PWM1.5	P1.30	AD0.4
P2.5	PWM1.6	P1.31	AD0.5
P2.6		P0.2	
P2.7		P0.3	
P2.8		P0.21	
P2.10		P0.22	
P2.11		P0.27	
P2.12		P0.28	
GND		P2.13	

Fig. 4. LPC1769 GPIO Pins

It can perform display data RAM read/write operation with no external operation clock to minimize power consumption.

Some of the technical specifications for the Liquid Crystal Display Module are:

- 1.8” diagonal LCD TFT display.
- 128x160 resolution.
- 18-bit (262,144) color.
- 4 or 5 wire SPI digital interface.
- 2 white LED backlight with PWM dimmer.
- 3.3V/5V compatible power logic.
- Overall dimensions: 34 mm X 56mm X 6.5mm
- Current drawn: 50mA (full backlight)
- Manufactured by Adafruit.
- ST7735R controlled with built in pixel-addressable RAM buffer video.

III. Software Requirement

Coding is done in which data is sent and processed by LCD in the form of frames. Every frame consists of a start of frame and end of the frame for the LCD to differentiate the incoming data. Usually, the sequence would start with a header byte followed by the command and then the data. At the end, a frame tail would be present. On the whole, the data in each frame is limited to 249 bytes maximum as the total including the header, command, and tail will come up to 255 bytes.

The program is built using Adafruit graphics library and NXPXpresso IDE. The software requirements involve initializing the LCD module using Adafruit library for sending data buffer to the LCD as well as polling the GPIO pins for any external interrupt and based on the external interrupt taking actions for displaying content on the LCD screen.

We need the following to full-fill software requirement for successful implementation :

1) Windows/Mac/Linux
2) NXPXpresso IDE Version 11
3) External LCD Communication programming Opcode.

IV. Circuit Implementation

The sample code is downloaded from GitHub website and the zip file is loaded into the development environment. Later the code is modified to set the pins and add functionality to the pins.

First, We set up the connections between the LCD and LPC Module.

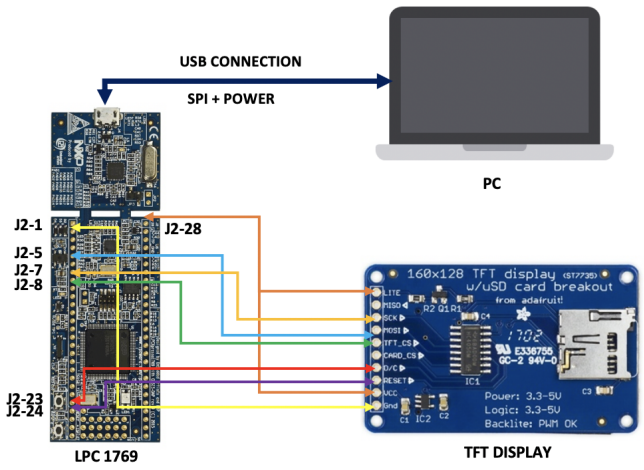

Fig. 5. Whole Interface Circuit Connections

A. LCD Interfacing Circuits

The SPI interface circuit inbuilt on the wire wrapping board of size 6x4 inches. It consists of an external LCD and LPC1769 module. The circuit is connected to the computer using the USB cable. The detailed design and connections are shown in the Figure 6.

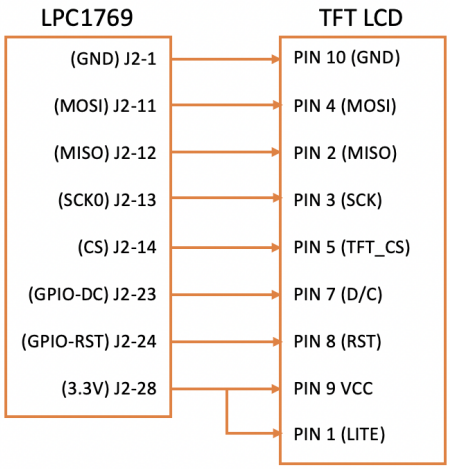

Fig. 6. Pinout between LPC and LCD

B. Algorithm

Few steps are needed to control LCD functionality, below is an algorithm for controlling the LCD:

1) Initialize the SPI and the LCD.
2) Set the bit mask for the MOSI, SCK, SEEL pin of the SPI port.
3) Define function opcode for LCD.
4) Send commands to LCD to display desired image.

Check Figure 7 for flowchart representation.

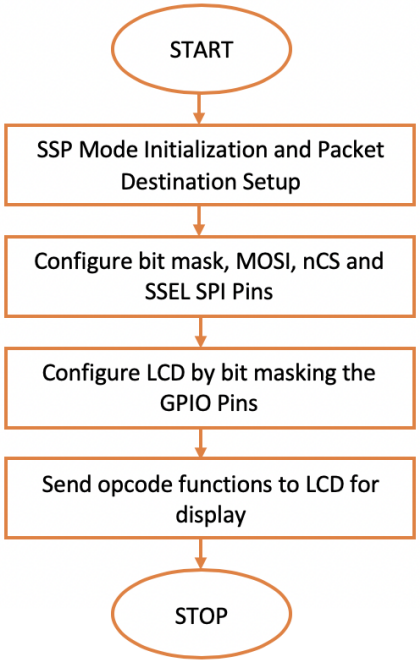

Fig. 7. Program Flowchart

C. Pseudo Code

```
float Lambda3D(float Zi,float Zs)
{
float lambda;
lambda = -Zi/(Zs-Zi);
return lambda;
}

typedef struct{
float_t x_value;
float_t y_value;
float_t z_value;
}Pts3D;

typedef struct{
float_t x;
float_t y;
}Pts2D;

Pts3D ShadowPoint3D(Pts3D Pi,
Pts3D Ps, float lambda)
{
Pts3D pt;
pt.x_value = (Pi.x_value +(lambda
*(Ps.x_value-Pi.x_value)));
pt.y_value = (Pi.y_value + (lambda
*(Ps.y_value-Pi.y_value)));
pt.z_value = (Pi.z_value +(lambda
*(Ps.z_value-Pi.z_value)));
return pt;
}

Pts2D get3DTransform(Pts3D Pi)
{
float_t Xe=130;
float_t Ye=130;
float_t Ze=130;
float_t Rho=sqrt(pow(Xe,2)
+pow(Ye,2)+pow(Xe,2));
float_t D_focal=100;

typedef struct{
float X;float Y;float Z;
}pworld;

typedef struct{
float X;float Y;float Z;
}pviewer;

typedef struct{
float X;float Y;
}pperspective;

pworld world;
pviewer viewer;
pperspective perspective;

world.X = Pi.x_value;
world.Y = Pi.y_value;
world.Z = Pi.z_value;

float sPheta =
        Ye/sqrt(pow(Xe,2)+pow(Ye,2));
float cPheta =
        Xe/sqrt(pow(Xe,2)+pow(Ye,2));
float sPhi =
        sqrt(pow(Xe,2)+pow(Ye,2))/Rho;
float cPhi = Ze/Rho;

viewer.X=-sPheta
        *world.X+cPheta*world.Y;
viewer.Y = -cPheta * cPhi *
        world.X - cPhi * sPheta *
        world.Y + sPhi * world.Z;
viewer.Z = -sPhi * cPheta *
        world.X - sPhi * cPheta *
        world.Y-cPheta * world.Z + Rho;

perspective.X=D_focal*
        viewer.X/viewer.Z;
perspective.Y=D_focal*
        viewer.Y/viewer.Z;

Pts2D pt;
        pt.x=perspective.X;
        pt.y=perspective.Y;
        return pt;

}

int getDiffuseColor(Pts3D Pi)
{
uint32_t print_diffuse_color;
Pts3D Ps = {40,60,120};
int diff_red, diff_green, diff_blue;
float new_red, new_green
, new_blue, reflect[3]
= {0.8, 0.0, 0.0}, scaling = 4000;

float_t temp = ((Ps.z_value - Pi.z_value)
/sqrt(pow((Ps.x_value - Pi.x_value),2)
+ pow((Ps.y_value - Pi.y_value),2)
+ pow((Ps.z_value - Pi.z_value),2)))
/ (pow((Ps.x_value - Pi.x_value),2)
+ pow((Ps.y_value - Pi.y_value),2)
+ pow((Ps.z_value - Pi.z_value),2));

temp *= scaling;

diff_red = reflect[0] * temp * 255;
diff_green = reflect[1] * temp;
```

```
diff_blue  = reflect[2] * temp;
new_red = (diff_red << 16);
new_green = (diff_green << 8);
new_blue = (diff_blue);
print_diffuse_color = new_red +
new_green +new_blue;
return print_diffuse_color;
}

// Cube implementation

void drawCube(){
#define UpperBD 52
#define NumOfPts 10
#define Pi 3.1415926
typedef struct{
    float X[UpperBD];
    float Y[UpperBD];
    float Z[UpperBD];
}pworld;
typedef struct{
    float X[UpperBD];
    float Y[UpperBD];
    float Z[UpperBD];
}pviewer;
typedef struct{
    float X[UpperBD];
    float Y[UpperBD];
}pperspective;

// EYE VIEW(130,130,130),

float_t Xe=130, Ye=130, Ze=130;

float_t Rho=sqrt(pow(Xe,2)
    +pow(Ye,2)+pow(Xe,2));
float_t D_focal=100;
float_t L1,L2,L3,L4;

pworld WCS;
pviewer V;
pperspective P;

// X-Y-Z World Coordinate System

// Origin
WCS.X[0]=0.0; WCS.Y[0]=0.0;
WCS.Z[0]=0.0; WCS.X[1]=50.0;
WCS.Y[1]=0.0; WCS.Z[1]=0.0;
WCS.X[2]=0.0; WCS.Y[2]=50.0;
WCS.Z[2]=0.0; WCS.X[3]=0.0;
WCS.Y[3]=0.0; WCS.Z[3]=50.0;

// Elevate Cube along Z_w axis by 10

WCS.X[4]=50.0; WCS.Y[4]=0;
WCS.Z[4]=10.0; WCS.X[5]=0.0;
WCS.Y[5]=50.0; WCS.Z[5]=10.0;
WCS.X[6]=0.0; WCS.Y[6]=0.0;
WCS.Z[6]=60.0; WCS.X[7]=50.0;
WCS.Y[7]=0.0; WCS.Z[7]=60.0;
WCS.X[8]=50.0; WCS.Y[8]=50.0;
WCS.Z[8]=10.0; WCS.X[9]=0.0;
WCS.Y[9]=50.0; WCS.Z[9]=60.0;
WCS.X[10]=50.0; WCS.Y[10]=50.0;
WCS.Z[10]=60.0;

// POINT OF LIGHT SOURCE PS(40, 60, 120)

WCS.X[11]=40.0; WCS.Y[11]=60.0;
WCS.Z[11]=120.0;

Pts3D Ps;
Ps.x_value = WCS.X[11];
Ps.y_value = WCS.Y[11];
Ps.z_value = WCS.Z[11];

Pts3D P1;
P1.x_value = WCS.X[6];
P1.y_value = WCS.Y[6];
P1.z_value = WCS.Z[6];

Pts3D P2;
P2.x_value = WCS.X[7];
P2.y_value = WCS.Y[7];
P2.z_value = WCS.Z[7];

Pts3D P3;
P3.x_value = WCS.X[9];
P3.y_value = WCS.Y[9];
P3.z_value = WCS.Z[9];

Pts3D P4;
P4.x_value = WCS.X[10];
P4.y_value = WCS.Y[10];
P4.z_value = WCS.Z[10];

// Shadow Points

Pts3D S1,S2,S3,S4;
L1 = Lambda3D(WCS.Z[6],WCS.Z[11]);
L2 = Lambda3D(WCS.Z[7],WCS.Z[11]);
L3 = Lambda3D(WCS.Z[9],WCS.Z[11]);
L4 = Lambda3D(WCS.Z[10],WCS.Z[11]);
S1 = ShadowPoint3D(P1,Ps,L1);
S2 = ShadowPoint3D(P2,Ps,L2);
S3 = ShadowPoint3D(P3,Ps,L3);
S4 = ShadowPoint3D(P4,Ps,L4);

WCS.X[12]=S1.x_value;
WCS.Y[12]=S1.y_value;
WCS.Z[12]=S1.z_value;
```

```
WCS.X[13]=S2.x_value;
WCS.Y[13]=S2.y_value;
WCS.Z[13]=S2.z_value;
WCS.X[14]=S3.x_value;
WCS.Y[14]=S3.y_value;
WCS.Z[14]=S3.z_value;
WCS.X[15]=S4.x_value;
WCS.Y[15]=S4.y_value;
WCS.Z[15]=S4.z_value;


float sPheta = Ye/
    sqrt(pow(Xe,2)+pow(Ye,2));
float cPheta = Xe/
    sqrt(pow(Xe,2)+pow(Ye,2));
float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/
    Rho;
float cPhi = Ze/Rho;

// WORLD TO VIEWER TRANSFROM

for(int i=0;i<=UpperBD;i++)
{
V.X[i]=-sPheta*WCS.X[i]+cPheta*WCS.Y[i];
V.Y[i] = -cPheta * cPhi * WCS.X[i]-
    cPhi * sPheta * WCS.Y[i]+ sPhi
    * WCS.Z[i];
V.Z[i] = -sPhi * cPheta * WCS.X[i]-
    sPhi * cPheta * WCS.Y[i]-
    cPheta * WCS.Z[i] + Rho;
}

for(int i=0;i<=UpperBD;i++)
{
P.X[i]=D_focal*V.X[i]/V.Z[i];
P.Y[i]=D_focal*V.Y[i]/V.Z[i];
}

//CUBE SHADOWS
// SHADOW DRAWLINES
drawLine
(P.X[12],P.Y[12],P.X[13],P.Y[13],GREY1);
drawLine
(P.X[13],P.Y[13],P.X[15],P.Y[15],GREY1);
drawLine
(P.X[14],P.Y[14],P.X[15],P.Y[15],GREY1);
drawLine
(P.X[14],P.Y[14],P.X[12],P.Y[12],GREY1);

//SHADOW FILL
for(int x=-40;x<=60;x++)
for(int y=-60;y<=40;y++)
{
Pts3D pt; Pts2D pt_new;
pt.x_value=x; pt.y_value=y; pt.z_value=0;
pt_new = get3DTransform(pt);
drawPixel(pt_new.x,pt_new.y,GREY1);
}

// LIGHT RAYS DRAWLINE BACK
drawLine
(P.X[12],P.Y[12],P.X[11],P.Y[11],GREY1);

// CUBE DRAWLINES
drawLine
(P.X[7],P.Y[7],P.X[4],P.Y[4],BLACK);
drawLine
(P.X[7],P.Y[7],P.X[6],P.Y[6],BLACK);
drawLine
(P.X[7],P.Y[7],P.X[10],P.Y[10],BLACK);
drawLine
(P.X[8],P.Y[8],P.X[4],P.Y[4],BLACK);
drawLine
(P.X[8],P.Y[8],P.X[5],P.Y[5],BLACK);
drawLine
(P.X[8],P.Y[8],P.X[10],P.Y[10],BLACK);
drawLine
(P.X[9],P.Y[9],P.X[6],P.Y[6],BLACK);
drawLine
(P.X[9],P.Y[9],P.X[5],P.Y[5],BLACK);
drawLine
(P.X[9],P.Y[9],P.X[10],P.Y[10],BLACK);
//RED TOP SIDE FILL
float diff_color;
for(int y=0;y<=50;y++)
for(int x=0;x<=50;x++)
{
Pts3D pt; Pts2D pt2d;
pt.x_value=x; pt.y_value=y;
pt.z_value=60;
pt2d = get3DTransform(pt);
// RED DIFFUSE
diff_color = getDiffuseColor(pt);
drawPixel(pt2d.x,pt2d.y,diff_color);
}
//GREEN FRONT SIDE FILL
for(int y=0;y<=50;y++)
for(int z=60;z>=10;z--)
{
Pts3D pt; Pts2D pt2d;
pt.x_value=50; pt.y_value=y;
pt.z_value=z;
pt2d = get3DTransform(pt);
drawPixel(pt2d.x,pt2d.y,GREEN1);
}
//BLUE RIGHT SIDE FILL
for(int z=60;z>=10;z--)
for(int x=50;x>=0;x--)
{
Pts3D pt; Pts2D pt2d;
pt.x_value=x; pt.y_value=50;
pt.z_value=z;
```

```
pt2d = get3DTransform(pt);
drawPixel(pt2d.x,pt2d.y,BLUE1);
}
// LIGHT RAYS DRAWLINES FRONT
drawLine
(P.X[13],P.Y[13],P.X[11],P.Y[11],GREY1);
drawLine
(P.X[14],P.Y[14],P.X[11],P.Y[11],GREY1);
drawLine
(P.X[15],P.Y[15],P.X[11],P.Y[11],GREY1);
}


int main (void)
{

uint32_t pnum = PORT_NUM;
pnum = 1 ;
if ( pnum == 1 )
    SSP1Init();
else
puts("INCORRECT PORT NUMBER");
lcd_init();
fillrect(0, 0, ST7735_TFTWIDTH,
    ST7735_TFTHEIGHT, BLACK);
//USER FUNCTION CALL
drawCube();

return 0;
}
```

V. Testing and Verification

Testing section mainly has to deal with testing the code and verifying the outputs. Procedures are followed to ensure working for the LCD. A multi-meter is used to ensure that no components are being shorted. This is done before debugging and actual testing is done. When the program in debugged, the LCD module should be initialized and output should displayed. IDE needs to recognize the USB link for the programmed to the debugged and to generate actual output.

3D Cube Shading Model: First, Cube was displayed in world coordinate system. Then, a point was created and was treated as a source of light and shadow has been implemented by computing shadow points from source of light point. Next, we have filled the sides displayed in perspective view with color. After that, compute diffuse reflection on the top surface of the cube using Digital Differential Algorithm formula. The four ray equations derived mathematically. So, keep track each set of 4 points and produce a dark red or such shade for the cube by plotting 4 vertices polygons. Assuming reflective nature for red is 0.8 and for blue and green are 0.0. All this is done in a 3D space, which is then projected onto a 2D space. This 2D space was displayed on the LCD display.

$$I_{diff}(x,y,z) = K \ * \frac{1}{\|r^2\|} * \frac{n*r}{\|n\|*\|r\|}(r_r, g_g, b_b) \qquad (1)$$

Equation 1: Digital Differential Algorithm

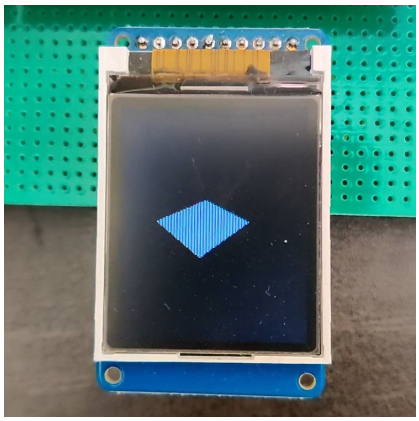

Fig. 8. Output Screenshot 1

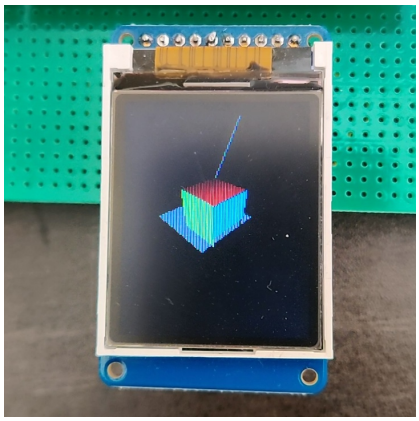

Fig. 9. Output Screenshot 2

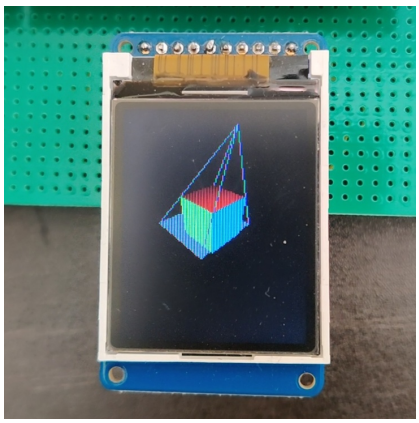

Fig. 10. Output Screenshot 3

Conclusion

We have successfully compiled the code for the 3D Cube Shading Model and debugged on MCUXpresso IDE and were able to display the 3D Cube on the LCD Display and show diffuse deflection on the top surface of the cube in world co-ordinate system.NXP's LPC1769 is a microprocessor capable of utilizing the Adafruit's LCD to create images. Creating the 3D Cube with diffuse reflections was a challenging experience and learnt a lot during this process.

Acknowledgment

The work and content described in this paper was achieved with the support of Dr. Harry Li. His informative lectures, previous lab assignments and class discussion in subject CMPE 240 - Advanced Computer Design have contributed to completion of the lab assignments. I would also thank the 4 member discussion teams created, as the other members gave valuable insight and feedback to various discussions and doubts.

References

[1] Dr. Harry Li, CMPE240 - Advanced Microcomputer Design lecture notes, Computer Engineering Department, San Jose State University.
[2] LPC1769 Datasheet PDF on www.nxp.com
[3] LCD Datasheet PDF on www.adafruit.com/product/358
[4] UM10360 LPC176x/5x User Manual on www.nxp.com
[5] Lecture Notes and Slides, Programs on github.com/hualili/CMPE240-Adv-Microprocessors

VI. Appendix

A. Components

Description	Quantity
1.8" 18-bit TFT color LCD	1
LPCXpresso 1769 CPU Module	1
Wire Wrapping Board	1
Wire Wrapping Board Kit	1
Various Color Wires for Wire Wrapping	3
¼" Stands for the Circuit Board	4
Male Header Pins	10
Female Header Pins	10
Type C to Micro USB Connector	1
MCUXpresso IDE 11	Mac/Linux/Windows

Fig. 11. All Components Needed

B. Wire Wrapping Board

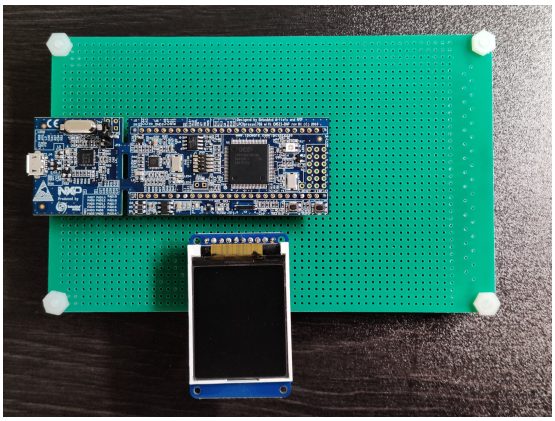

Fig. 12. Hardware Setup 1

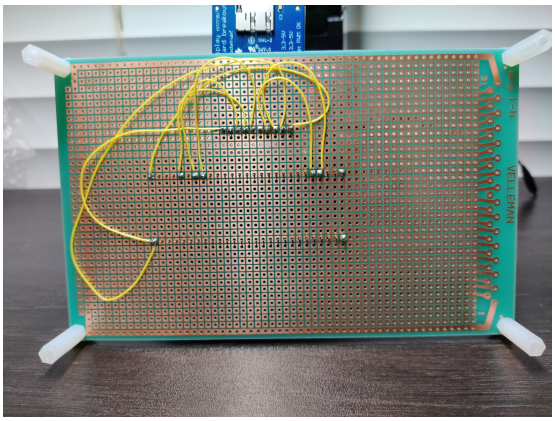

Fig. 13. Hardware Setup 2

C. Source Code

```
/*
===============================================
 Name        : 2019F_LAB3_014525420.c
 Author      : AKHIL CHERUKURI
 Version     : FINAL
 Copyright   : $(copyright)
 Description : Lab LPC-ARM Architecture
 3D Graphics Processing Engine Design
===============================================
*/
#include <cr_section_macros.h>
#include <NXP/crp.h>
#include "LPC17xx.h"
#include "ssp.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
```

```
#define PORT_NUM            0

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

#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159
#define ST7735_CASET 0x2A
#define ST7735_RASET 0x2B
#define ST7735_RAMWR 0x2C
#define ST7735_SLPOUT 0x11
#define ST7735_DISPON 0x29

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

// Defining Colors

#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF

// Self Defined Colors
#define YELLOW 0xFFFF00
#define BROWN 0xA52A2A
#define BLUE1 0x3999FF
#define ORANGE1 0xFFA845
#define GREEN1 0x93FF60
#define GREY1 0x7A7C7B


int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;


void spiwrite(uint8_t c)
{
 int pnum = 1;
 src_addr[0] = c;
 SSP_SSELToggle( pnum, 0 );
 SSPSend( pnum, (uint8_t *)src_addr, 1 );
 SSP_SSELToggle( pnum, 1 );
}

void writecommand(uint8_t c)
{
 LPC_GPIO0->FIOCLR |= (0x1<<21);
 spiwrite(c);
}

void writedata(uint8_t c)
{
 LPC_GPIO0->FIOSET |= (0x1<<21);
 spiwrite(c);
}

void writeword(uint16_t c)
{
 uint8_t d;
 d = c >> 8;
 writedata(d);
 d = c & 0xFF;
 writedata(d);
}

void write888(uint32_t color,
    uint32_t repeat)
{
 uint8_t red, green, blue;
 int i;
 red = (color >> 16);
 green = (color >> 8) & 0xFF;
 blue = color & 0xFF;
 for (i = 0; i< repeat; i++) {
  writedata(red);
  writedata(green);
  writedata(blue);
 }
}

void setAddrWindow(uint16_t x0,
    uint16_t y0, uint16_t x1,
    uint16_t y1)
{
 writecommand(ST7735_CASET);
 writeword(x0);
 writeword(x1);
 writecommand(ST7735_RASET);
 writeword(y0);
 writeword(y1);

}

void fillrect(int16_t x0, int16_t y0,
    int16_t x1, int16_t y1,
    uint32_t color)
{
 //int16_t i;
 int16_t width, height;
 width = x1-x0+1;
 height = y1-y0+1;
 setAddrWindow(x0,y0,x1,y1);
 writecommand(ST7735_RAMWR);
 write888(color,width*height);
}
```

```
void lcddelay(int ms)
{
 int count = 24000;
 int i;
 for ( i = count*ms; i--; i > 0);
}

void lcd_init()
{
 int i;
 printf("CUBE PRINTED\n");
 // Set P0.16, P0.21, P0.22 = output
 LPC_GPIO0->FIODIR |= (0x1<<16);
 LPC_GPIO0->FIODIR |= (0x1<<21);
 LPC_GPIO0->FIODIR |= (0x1<<22);

 // Hardware Reset Sequence
 LPC_GPIO0->FIOSET |= (0x1<<22);
 lcddelay(500);
 LPC_GPIO0->FIOCLR |= (0x1<<22);
 lcddelay(500);
 LPC_GPIO0->FIOSET |= (0x1<<22);
 lcddelay(500);

 // initialize buffers
 for ( i = 0; i < SSP_BUFSIZE; i++ )
 {
   src_addr[i] = 0;
   dest_addr[i] = 0;
 }

 // Take LCD display out of sleep mode
 writecommand(ST7735_SLPOUT);
 lcddelay(200);

 // Turn LCD display on
 writecommand(ST7735_DISPON);
 lcddelay(200);

}

// Coordinate to Cartesian
int16_t xCartesian(int16_t x){
        x = x + (_width>>1);
        return x;
}

int16_t yCartesian(int16_t y){
        y = (_height>>1) - y;
        return y;
}

void drawPixel(int16_t x, int16_t y,
    uint32_t color)
{
// Cartesian Implementation
x=xCartesian(x); y=yCartesian(y);

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

void drawLine(int16_t x0, int16_t y0,
    int16_t x1, int16_t y1,
    uint32_t color)
{
 int16_t slope = abs(y1 - y0)
    > abs(x1 - x0);
 if (slope) {
  swap(x0, y0);
  swap(x1, y1);
 }
 if (x0 > x1) {
  swap(x0, x1);
  swap(y0, y1);
 }
 int16_t dx, dy;
 dx = x1 - x0;
 dy = abs(y1 - y0);
 int16_t err = dx / 2;
 int16_t ystep;
 if (y0 < y1) {
  ystep = 1;
 }
 else {
  ystep = -1;
 }
 for (; x0 <= x1; x0++) {
  if (slope) {
   drawPixel(y0, x0, color);
  }
  else {
   drawPixel(x0, y0, color);
  }
  err -= dy;
  if (err < 0) {
   y0 += ystep;
   err += dx;
  }
 }
}

// 3D ENGINE

float Lambda3D(float Zi,float Zs)
{
float lambda;
lambda = -Zi/(Zs-Zi);
```

```
return lambda;
}

typedef struct{
float_t x_value; float_t y_value;
float_t z_value;
}Pts3D;

typedef struct{
float_t x; float_t y;
}Pts2D;

Pts3D ShadowPoint3D(Pts3D Pi,
    Pts3D Ps, float lambda)
{
Pts3D pt;
pt.x_value = (Pi.x_value +
    (lambda*(Ps.x_value-Pi.x_value)));
pt.y_value = (Pi.y_value
    (lambda*(Ps.y_value-Pi.y_value)));
pt.z_value = (Pi.z_value +
    (lambda*(Ps.z_value-Pi.z_value)));
return pt;
}

Pts2D get3DTransform(Pts3D Pi)
{
float_t Xe=130, Ye=130, Ze=130;
float_t Rho=sqrt(pow(Xe,2)+pow(Ye,2)
    +pow(Xe,2));
float_t D_focal=100;

typedef struct{
        float X; float Y; float Z;
}pworld;

typedef struct{
        float X; float Y; float Z;
}pviewer;

typedef struct{
        float X; float Y;
}pperspective;

pworld world;
pviewer viewer;
pperspective perspective;

world.X = Pi.x_value;
world.Y = Pi.y_value;
world.Z = Pi.z_value;

float sPheta = Ye/sqrt(pow(Xe,2)
    +pow(Ye,2));
float cPheta = Xe/sqrt(pow(Xe,2)
    +pow(Ye,2));
float sPhi = sqrt(pow(Xe,2)+
    pow(Ye,2))/Rho;
float cPhi = Ze/Rho;

viewer.X=-sPheta*world.X+cPheta*world.Y;
viewer.Y = -cPheta * cPhi * world.X
    - cPhi * sPheta * world.Y
    + sPhi * world.Z;
viewer.Z = -sPhi * cPheta * world.X
    - sPhi  * cPheta * world.Y-cPheta
    * world.Z + Rho;

perspective.X=D_focal*viewer.X/viewer.Z;
perspective.Y=D_focal*viewer.Y/viewer.Z;

Pts2D pt;
pt.x=perspective.X;
pt.y=perspective.Y;
return pt;
}

int getDiffuseColor(Pts3D Pi)
{
uint32_t print_diffuse_color;
Pts3D Ps = {40,60,120};
int diff_red, diff_green, diff_blue;
float new_red, new_green, new_blue,
    r1=0.8, r2=0.0, r3=0.0,
    scaling = 4000;

float_t temp = ((Ps.z_value - Pi.z_value)
/sqrt(pow((Ps.x_value - Pi.x_value),2)
+ pow((Ps.y_value - Pi.y_value),2)
+ pow((Ps.z_value - Pi.z_value),2)))
/ (pow((Ps.x_value - Pi.x_value),2)
+ pow((Ps.y_value - Pi.y_value),2)
+ pow((Ps.z_value - Pi.z_value),2));

temp *= scaling;

diff_red = r1 * temp * 255;
diff_green = r2 * temp;
diff_blue  = r3 * temp;
new_red = (diff_red << 16);
new_green = (diff_green << 8);
new_blue = (diff_blue);
print_diffuse_color = new_red +
    new_green +new_blue;
return print_diffuse_color;
}

void drawCube(){
#define UpperBD 52
#define NumOfPts 10
#define Pi 3.1415926
typedef struct{
```

```
    float X[UpperBD];
    float Y[UpperBD];
    float Z[UpperBD];
}pworld;
typedef struct{
    float X[UpperBD];
    float Y[UpperBD];
    float Z[UpperBD];
}pviewer;
typedef struct{
    float X[UpperBD];
    float Y[UpperBD];
}pperspective;

// EYE VIEW(130,130,130),

float_t Xe=130, Ye=130, Ze=130;

float_t Rho=sqrt(pow(Xe,2)
    +pow(Ye,2)+pow(Xe,2));
float_t D_focal=100;
float_t L1,L2,L3,L4;

pworld WCS;
pviewer V;
pperspective P;

// X-Y-Z World Coordinate System

// Origin
WCS.X[0]=0.0; WCS.Y[0]=0.0;
WCS.Z[0]=0.0; WCS.X[1]=50.0;
WCS.Y[1]=0.0; WCS.Z[1]=0.0;
WCS.X[2]=0.0; WCS.Y[2]=50.0;
WCS.Z[2]=0.0; WCS.X[3]=0.0;
WCS.Y[3]=0.0; WCS.Z[3]=50.0;

// Elevate Cube along Z_w axis by 10

WCS.X[4]=50.0; WCS.Y[4]=0;
WCS.Z[4]=10.0; WCS.X[5]=0.0;
WCS.Y[5]=50.0; WCS.Z[5]=10.0;
WCS.X[6]=0.0; WCS.Y[6]=0.0;
WCS.Z[6]=60.0; WCS.X[7]=50.0;
WCS.Y[7]=0.0; WCS.Z[7]=60.0;
WCS.X[8]=50.0; WCS.Y[8]=50.0;
WCS.Z[8]=10.0; WCS.X[9]=0.0;
WCS.Y[9]=50.0; WCS.Z[9]=60.0;
WCS.X[10]=50.0; WCS.Y[10]=50.0;
WCS.Z[10]=60.0;

// POINT OF LIGHT SOURCE PS(40, 60, 120)

WCS.X[11]=40.0; WCS.Y[11]=60.0;
WCS.Z[11]=120.0;
```

```
Pts3D Ps;
Ps.x_value = WCS.X[11];
Ps.y_value = WCS.Y[11];
Ps.z_value = WCS.Z[11];

Pts3D P1;
P1.x_value = WCS.X[6];
P1.y_value = WCS.Y[6];
P1.z_value = WCS.Z[6];

Pts3D P2;
P2.x_value = WCS.X[7];
P2.y_value = WCS.Y[7];
P2.z_value = WCS.Z[7];

Pts3D P3;
P3.x_value = WCS.X[9];
P3.y_value = WCS.Y[9];
P3.z_value = WCS.Z[9];

Pts3D P4;
P4.x_value = WCS.X[10];
P4.y_value = WCS.Y[10];
P4.z_value = WCS.Z[10];

// Shadow Points

Pts3D S1,S2,S3,S4;
L1 = Lambda3D(WCS.Z[6],WCS.Z[11]);
L2 = Lambda3D(WCS.Z[7],WCS.Z[11]);
L3 = Lambda3D(WCS.Z[9],WCS.Z[11]);
L4 = Lambda3D(WCS.Z[10],WCS.Z[11]);
S1 = ShadowPoint3D(P1,Ps,L1);
S2 = ShadowPoint3D(P2,Ps,L2);
S3 = ShadowPoint3D(P3,Ps,L3);
S4 = ShadowPoint3D(P4,Ps,L4);

WCS.X[12]=S1.x_value;
WCS.Y[12]=S1.y_value;
WCS.Z[12]=S1.z_value;
WCS.X[13]=S2.x_value;
WCS.Y[13]=S2.y_value;
WCS.Z[13]=S2.z_value;
WCS.X[14]=S3.x_value;
WCS.Y[14]=S3.y_value;
WCS.Z[14]=S3.z_value;
WCS.X[15]=S4.x_value;
WCS.Y[15]=S4.y_value;
WCS.Z[15]=S4.z_value;


float sPheta = Ye/
    sqrt(pow(Xe,2)+pow(Ye,2));
float cPheta = Xe/
    sqrt(pow(Xe,2)+pow(Ye,2));
float sPhi = sqrt(pow(Xe,2)+pow(Ye,2))/
```

```
    Rho;
float cPhi = Ze/Rho;

// WORLD TO VIEWER TRANSFROM

for(int i=0;i<=UpperBD;i++)
{
V.X[i]=-sPheta*WCS.X[i]+cPheta*WCS.Y[i];
V.Y[i] = -cPheta * cPhi * WCS.X[i]-
    cPhi * sPheta * WCS.Y[i]+ sPhi
    * WCS.Z[i];
V.Z[i] = -sPhi * cPheta * WCS.X[i]-
    sPhi * cPheta * WCS.Y[i]-
    cPheta * WCS.Z[i] + Rho;
}

for(int i=0;i<=UpperBD;i++)
{
P.X[i]=D_focal*V.X[i]/V.Z[i];
P.Y[i]=D_focal*V.Y[i]/V.Z[i];
}

//CUBE SHADOWS
// SHADOW DRAWLINES
drawLine
(P.X[12],P.Y[12],P.X[13],P.Y[13],GREY1);
drawLine
(P.X[13],P.Y[13],P.X[15],P.Y[15],GREY1);
drawLine
(P.X[14],P.Y[14],P.X[15],P.Y[15],GREY1);
drawLine
(P.X[14],P.Y[14],P.X[12],P.Y[12],GREY1);

//SHADOW FILL
for(int x=-40;x<=60;x++)
for(int y=-60;y<=40;y++)
{
Pts3D pt; Pts2D pt_new;
pt.x_value=x; pt.y_value=y; pt.z_value=0;
pt_new = get3DTransform(pt);
drawPixel(pt_new.x,pt_new.y,GREY1);
}

// LIGHT RAYS DRAWLINE BACK
drawLine
(P.X[12],P.Y[12],P.X[11],P.Y[11],GREY1);

// CUBE DRAWLINES
drawLine
(P.X[7],P.Y[7],P.X[4],P.Y[4],BLACK);
drawLine
(P.X[7],P.Y[7],P.X[6],P.Y[6],BLACK);
drawLine
(P.X[7],P.Y[7],P.X[10],P.Y[10],BLACK);
drawLine
(P.X[8],P.Y[8],P.X[4],P.Y[4],BLACK);
drawLine
(P.X[8],P.Y[8],P.X[5],P.Y[5],BLACK);
drawLine
(P.X[8],P.Y[8],P.X[10],P.Y[10],BLACK);
drawLine
(P.X[9],P.Y[9],P.X[6],P.Y[6],BLACK);
drawLine
(P.X[9],P.Y[9],P.X[5],P.Y[5],BLACK);
drawLine
(P.X[9],P.Y[9],P.X[10],P.Y[10],BLACK);
//RED TOP SIDE FILL
float diff_color;
for(int y=0;y<=50;y++)
for(int x=0;x<=50;x++)
{
Pts3D pt; Pts2D pt2d;
pt.x_value=x; pt.y_value=y;
pt.z_value=60;
pt2d = get3DTransform(pt);
// RED DIFFUSE
diff_color = getDiffuseColor(pt);
drawPixel(pt2d.x,pt2d.y,diff_color);
}
//GREEN FRONT SIDE FILL
for(int y=0;y<=50;y++)
for(int z=60;z>=10;z--)
{
Pts3D pt; Pts2D pt2d;
pt.x_value=50; pt.y_value=y;
pt.z_value=z;
pt2d = get3DTransform(pt);
drawPixel(pt2d.x,pt2d.y,GREEN1);
}
//BLUE RIGHT SIDE FILL
for(int z=60;z>=10;z--)
for(int x=50;x>=0;x--)
{
Pts3D pt; Pts2D pt2d;
pt.x_value=x; pt.y_value=50;
pt.z_value=z;
pt2d = get3DTransform(pt);
drawPixel(pt2d.x,pt2d.y,BLUE1);
}
// LIGHT RAYS DRAWLINES FRONT
drawLine
(P.X[13],P.Y[13],P.X[11],P.Y[11],GREY1);
drawLine
(P.X[14],P.Y[14],P.X[11],P.Y[11],GREY1);
drawLine
(P.X[15],P.Y[15],P.X[11],P.Y[11],GREY1);
}

int main (void)
{

uint32_t pnum = PORT_NUM;
```

```
pnum = 1 ;
if ( pnum == 1 )
    SSP1Init();
else
puts("INCORRECT PORT NUMBER");
lcd_init();
fillrect(0, 0, ST7735_TFTWIDTH,
    ST7735_TFTHEIGHT, BLACK);
//USER FUNCTION CALL
drawCube();

return 0;
}
```