CMPE 240 - Advanced Computer Design

Assignment: 3D – Graphics Design (Cube and Half-sphere)

Name: Tirumala Saiteja Goruganthu

SJSU ID: 016707210

Team: Student_Team1 (Harish Marepalli, Debashish Panigrahi, Shahnawaz Idariya are my team members)

Professor: Harry Li

Date: 11/01/2022

Target Board: LPC1769

Purpose:

This assignment focuses on collecting data from a CPU Module to implement 3D visual designs on an LCD Display using the transformation pipeline. The CPU Module utilized in this project is the LPC 1769, which is based on the ARM Cortex M0 Core. For the implementation of our task, we use the MCUXpresso IDE. The main goal of this assignment is to learn how to create and display world coordinate system with respect to 3D objects based on the transformation pipeline. Following is the description of the tasks given to achieve the goal.

Given Tasks:

The final program should:

1. Draw x_w-y_w-z_w axis, and a floating cube. Make x_w as red, y_w as green and z_w as blue. Make the cube (wire frame) lines as white lines.
    a. Each axis length should be equal to 50.
    b. The eye coordinates (200,200,200) and focallength (D) can be anything between 20 and 50.
    c. Create a cube of size 100 for each of its sides and place the cube float with one of its vertices equal to (100,100,110).
    d. Finally, draw the world coordinate system with the floating cube.

2. Create a sphere with R = 100. You may want to create 8 to 10 levels of cross section contours, denoted as C_1, C_2, …, C_i, of the sphere as illustrated in the class. The largest contour C_1 is a circle on the x_w-y_w plane, then a set of parallel smaller contours with the same location of their center but with Z_w distance in the range of 5 to 10 of your choice from one to the other.

a. Write a program to compute equal distanced vertex on the contour and organize 4-vertices patches from every two consecutive contours, be sure to have all the vertices so the entire surface of the half sphere can be covered by the 4-vertices patches.
b. Draw the half sphere on top of of the world coordinate system and the cube.

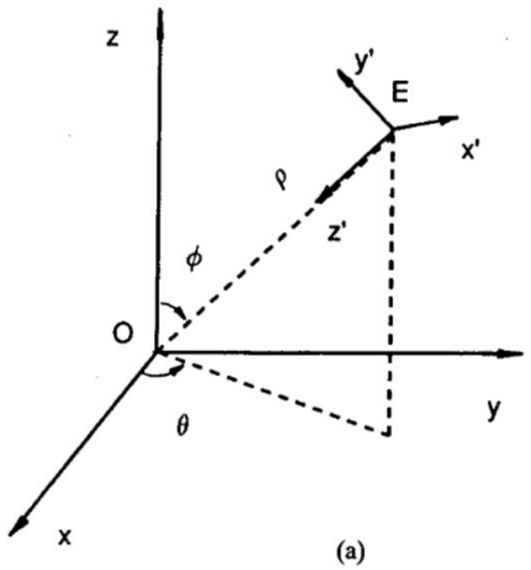
Components Used:

1. Prototype Board
2. LPC1769 inserted into female header pins and soldered to the board.
3. LCD TFT ST7735 Display Device.
4. USB probe to dump the program into on-chip flash memory.

Theory and Formulation:

The LPC1769 and 1.8" TFT LCD Display are soldered together on a prototype board and connected to the power circuit to work as a standalone module. In SPI Communication, we have a Master – Slave architecture where one-part acts as a Master of the communication and the other acts as a Slave. This type of architecture is implemented by connecting LCD module as a slave and LPC CPU Module as a master. There are four main SPI logic signals used. They are SCK (Serial Clock, output from master), MOSI (Master Output Slave Input, output from master), MISO (Master Input Slave Output, output from slave) and CS (Chip Select, active low, output from master). Data Transfer take place from host computer to CPU module through the USB link and then the CPU transfers the data to the LCD Display module.

a. Everyday things are in a system called world-coordinate system. This system is used to gauge all the 3D objects around us. Moreover, when we suppress one axis, we can gauge 2D objects as well. Now we see objects in another system called viewer-coordinate system. The relationship between the world and viewer coordinate system is shown as below (referred from Prof. Harry Li IEEE paper):



(a)

where,
E = viewer's eye coordinates

The viewer coordinate system in the above picture is depicted as (x^1-y^1-z^1) system and (x-y-z) as the world coordinate system.

The mathematical formulation to convert the world to viewer coordinate system is as follows:

$$(x_e, y_e, z_e, 1) = \mathbf{T}\,(x_w, y_w, z_w, 1)$$

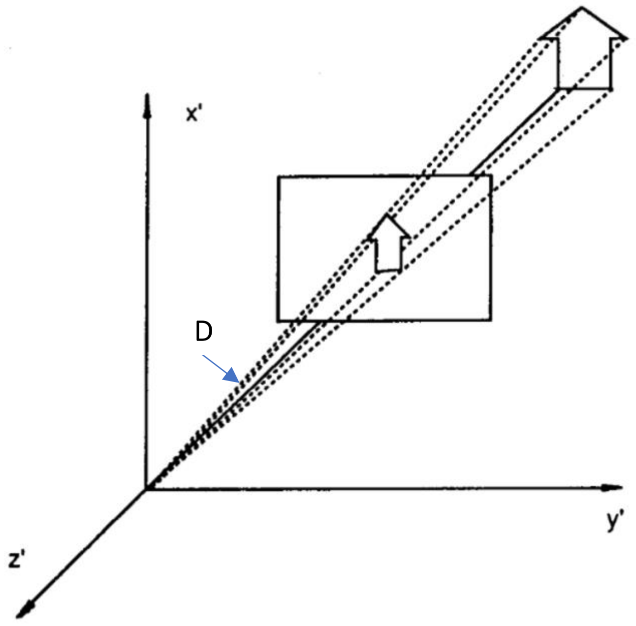
where, T = Transformation matrix which is defined as follows (referred from Prof. Harry Li IEEE paper)

$$\mathbf{T} = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\cos\phi\cos\theta & -\cos\phi\sin\theta & \sin\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\cos\theta & -\cos\phi & \rho \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b. Upon viewing the transformation, the perspective transformation is performed to produce a two-dimensional description of the three-dimensional object. The transformation is defined as (referred from Prof. Harry Li IEEE paper):

$$x_p = x_e\left(\frac{D}{z_e}\right)$$
$$y_p = y_e\left(\frac{D}{z_e}\right)$$

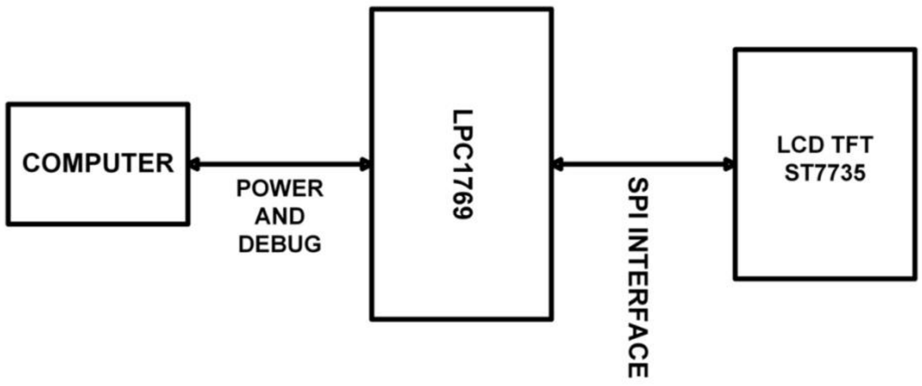
where (xp, yp) is the vertex to be plotted on the screen. The physical meaning of D explained directly from the figure below (referred from Prof. Harry Li IEEE paper)



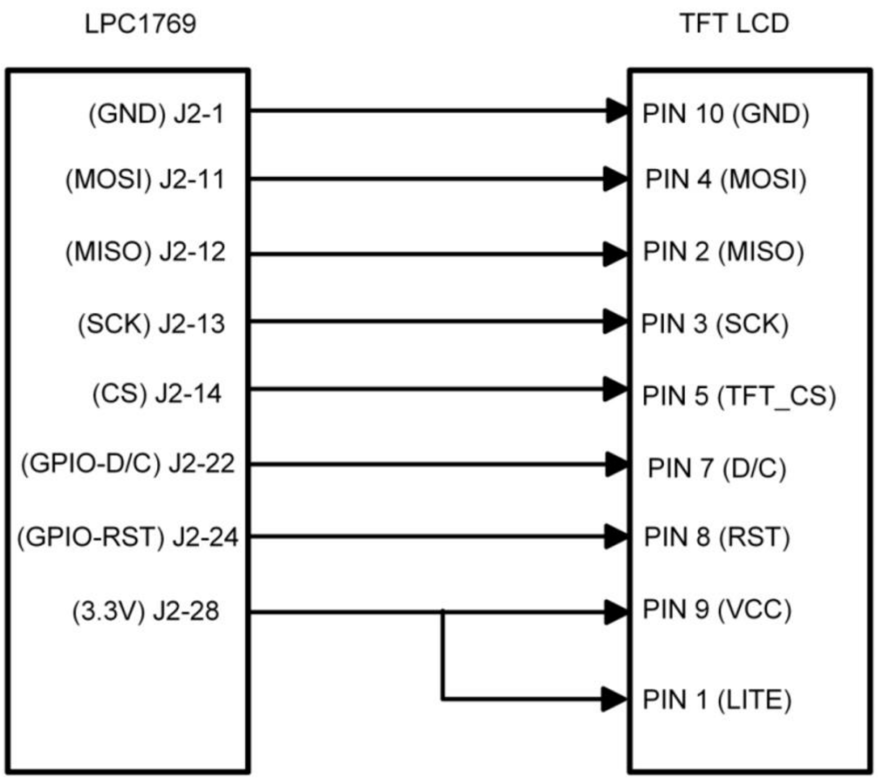
From the above picture, we can say that 'D' is the distance from the frontal plane to the origin of the viewer-coordinate system. The longer the distance from the "eye" to the frontal plane, the shorter the distance from the frontal plane to the actual three-dimensional object and hence, the bigger the picture on the display.

Rubrics - Block Diagrams:

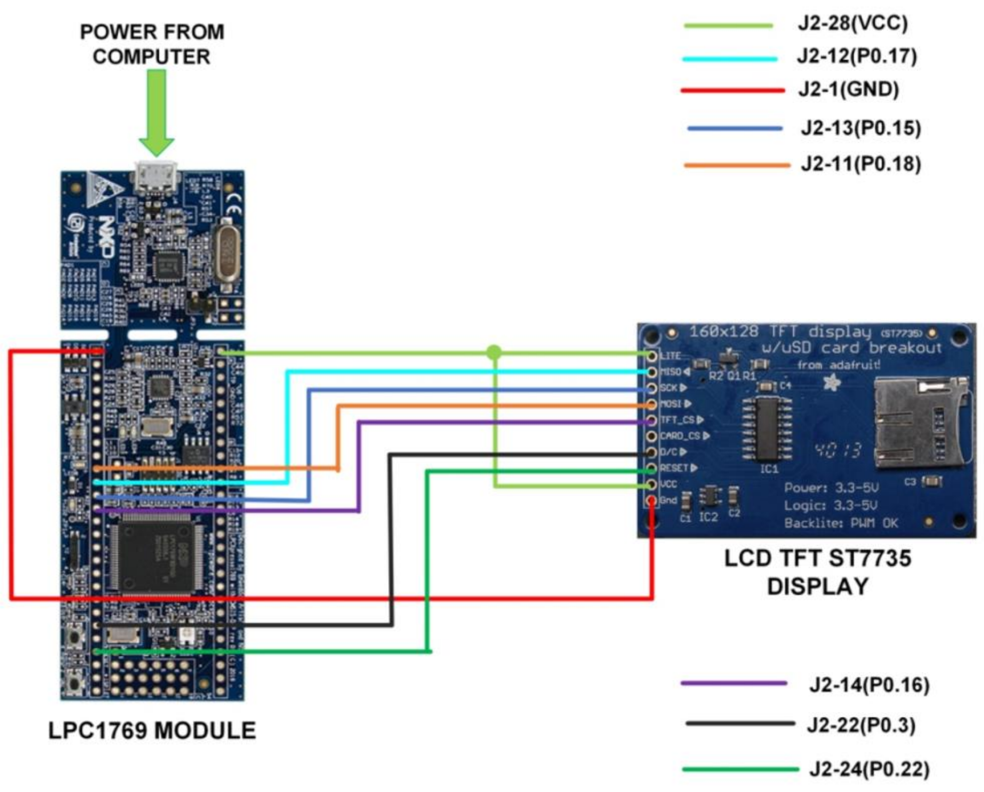
1. The block diagram of the entire system setup including the computer block (Done using Visio tool):



2. The system block diagram of the SPI color LCD interface (Done using Visio tool):



3. The schematic diagram of the interface connections between the LPC1769 interface to LCD color display panel (Done using Visio tool):

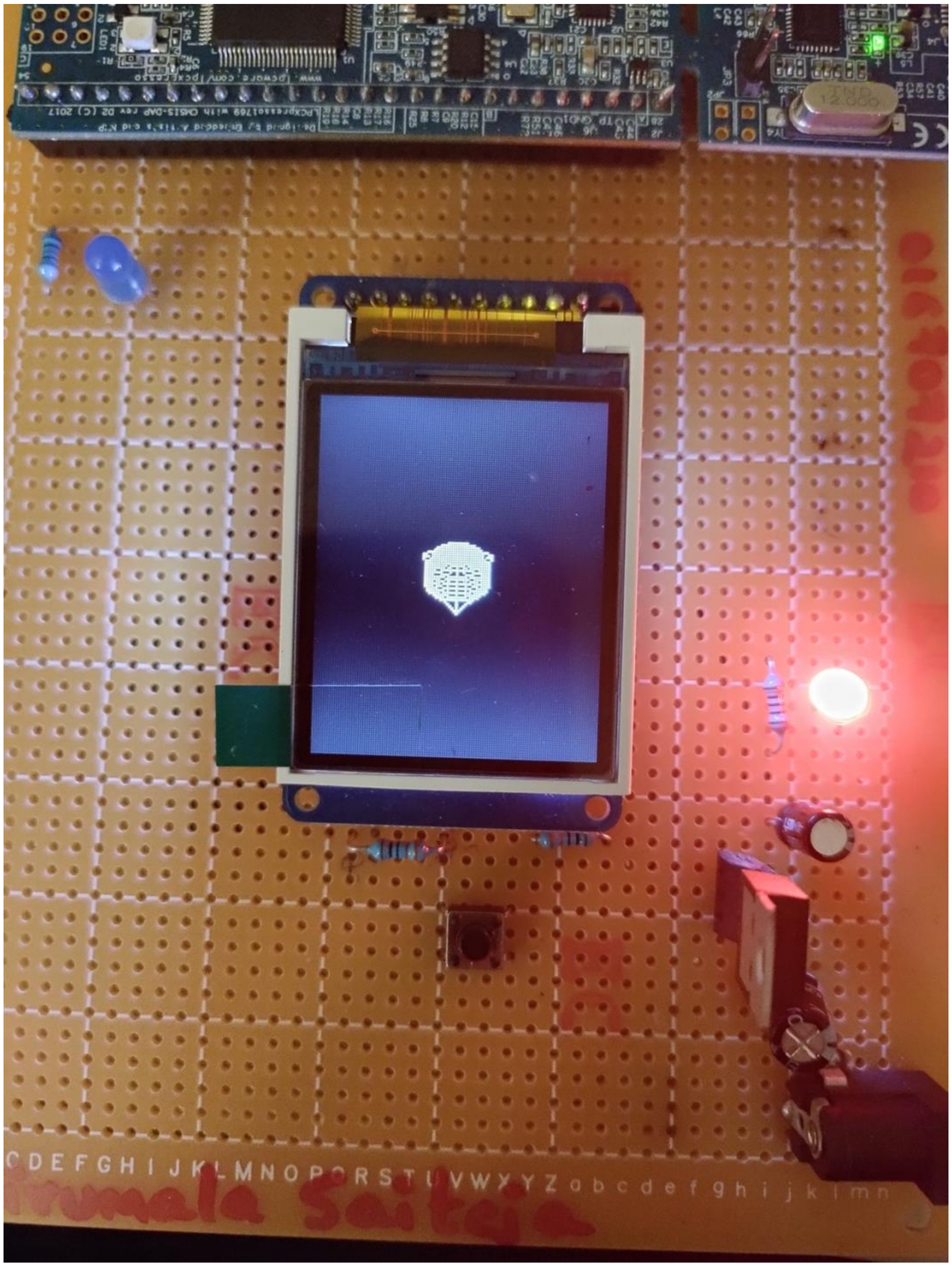


4. The pin-connectivity table for the interface is as follows:

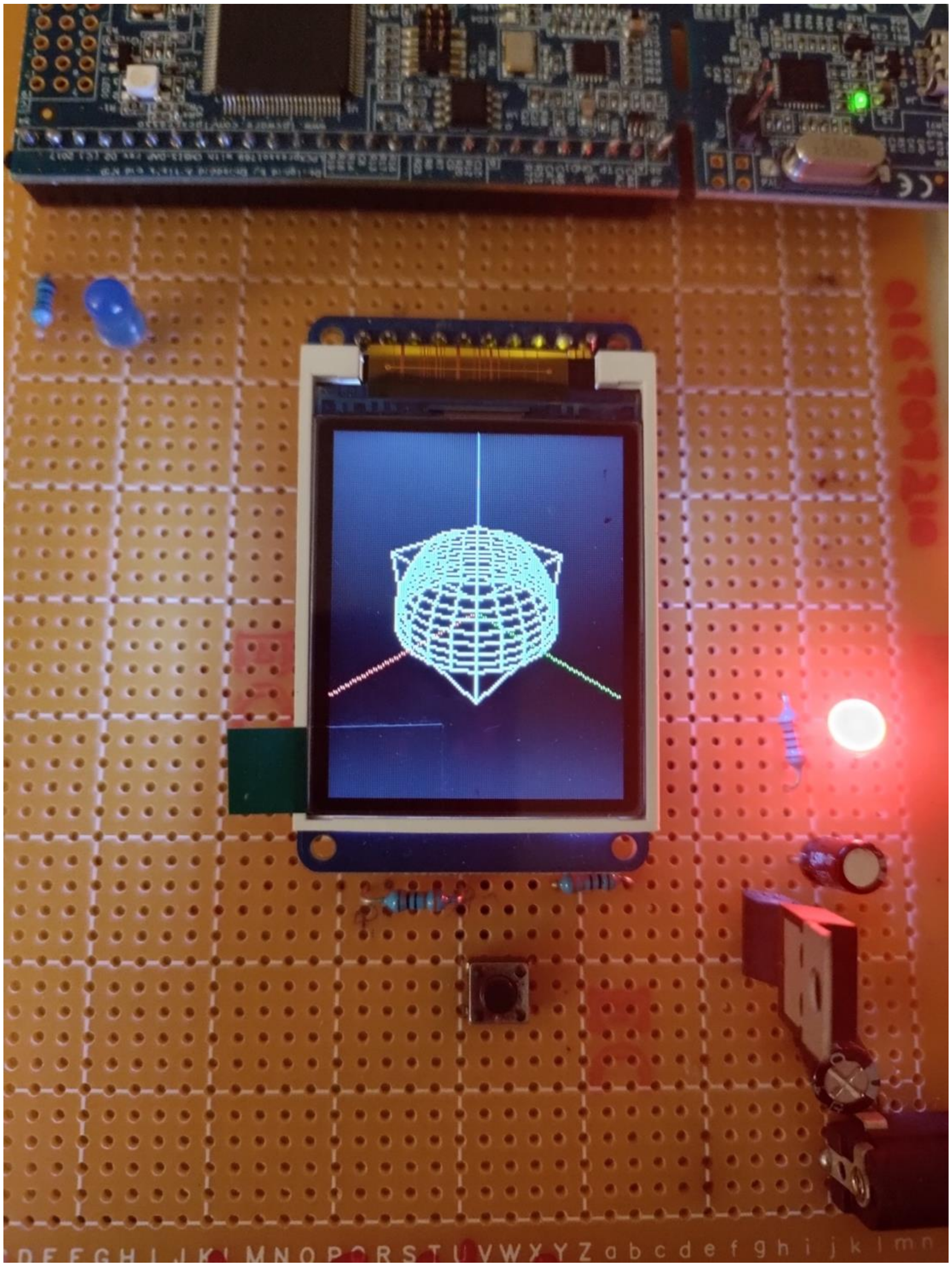
LPC1769 PINS (JTAG PINS)	**LCD TFT ST7735**
VCC (J2-28)	LITE
P0.17 (J2-12)	MISO
P0.15 (J2-13)	SCK
P0.18 (J2-11)	MOSI
P0.16 (J2-14)	TFT_CS
Not Connected	CARD_CS
P0.3 (J2-22)	D/C
P0.22 (J2-24)	RESET
VCC (J2-28)	VCC
GND (J2-1)	GND

Rubrics – Snippets:

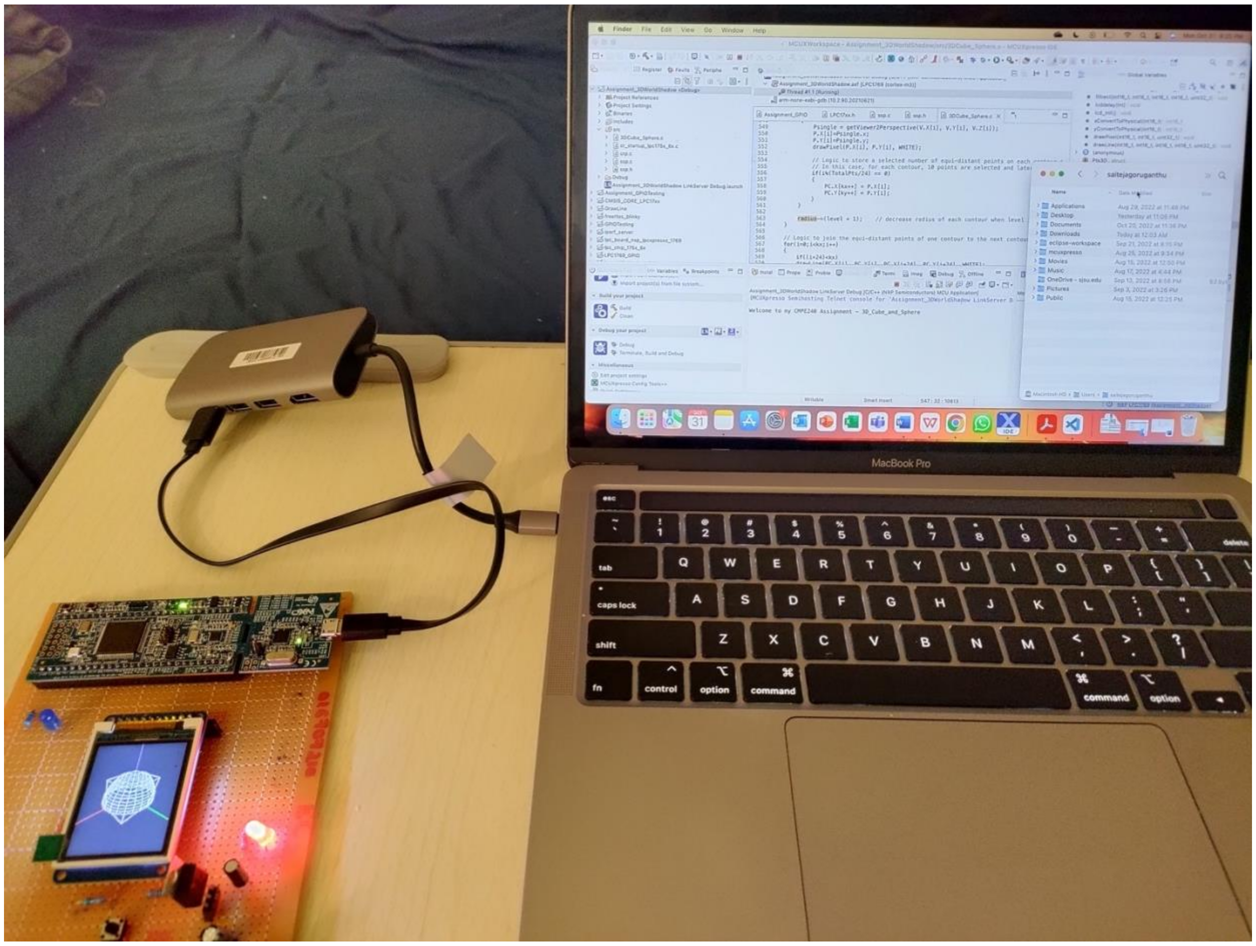
1. The snippet of the LCD display covering the output with given parameters i.e., D=50, Axes Length = 50, Eye Coordinates = (200,200,200), Sphere radius =100, Cube size = 100.



2. The snippet of the LCD display covering the output with given parameters i.e., D=120, Axes Length = 200, Eye Coordinates = (200,200,200), Sphere radius =100, Cube size = 100.



3. The snippet of the entire prototype system along with the laptop opened with Xpresso and my home folder (with my name).



CONCLUSION:

Upon completion of the experiment, we understand how to implement desired 3D graphics using transformation pipeline and communicate it with LCD for displaying purpose. The experiment related source code has been included in the appendix.

APPENDIX

Source Code:

```
/*
===============================================================================
 Name        : 3DCube_Sphere.c
 Author      : Tirumala Saiteja Goruganthu
 SJSU ID      : 016707210
 Course      : CMPE240 - Advanced Computer Design
 Professor   : Harry Li
 Description : This program is written to realize a cube and a half sphere. The
               cube is drawn using the points first plotted in world coordinate
               system and then using world-to-viewer transform, perspective
               transform, we use drawLine method to plot the cube on the LCD screen.
               Similarly, the half sphere is realized by drawing a circle in the X-Y
               plane and simultaneously drawing contours with same center, decreased
               radius and increasing Z value. For the 4-vertices patches, I have
               written a simple code to first store a selected equi-distant points on
               each contour and joining the points from one contour to upper contour
               using drawLine method.
===============================================================================
*/

#include <cr_section_macros.h>
#include <NXP/crp.h>
#include "LPC17xx.h"                        /* LPC17xx definitions */
#include "ssp.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

/* Be careful with the port number and location number, because

some of the location may not exist in that port. */
```

```
#define PORT_NUM            0

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

#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159

#define ST7735_CASET 0x2A
#define ST7735_RASET 0x2B
#define ST7735_RAMWR 0x2C
#define ST7735_SLPOUT 0x11
#define ST7735_DISPON 0x29

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

// defining color values

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

// Self Defined Colors
#define BROWN 0xA52A2A
#define YELLOW 0xFFFE00

// Self Defined GREEN Shades
#define GREEN1 0x38761D
#define GREEN2 0x002200
#define GREEN3 0x00FC7C
#define GREEN4 0x32CD32
```

```
#define GREEN5 0x228B22
#define GREEN6 0x006400

// Self Defined RED Shades
#define RED1 0xE61C1C
#define RED2 0xEE3B3B
#define RED3 0xEF4D4D
#define RED4 0xE88080

int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

// Defining the eye co-ordinates
float Xe=200, Ye=200, Ze=200;

// Defining the focal length
float D_focal=120;

void spiwrite(uint8_t c)
{

    int pnum = 0;

    src_addr[0] = c;

    SSP_SSELToggle( pnum, 0 );

    SSPSend( pnum, (uint8_t *)src_addr, 1 );

    SSP_SSELToggle( pnum, 1 );

}

void writecommand(uint8_t c)

{
    LPC_GPIO0->FIOCLR |= (0x1<<3);

    spiwrite(c);

}

void writedata(uint8_t c)

{

    LPC_GPIO0->FIOSET |= (0x1<<3);

    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 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 > 0; i--);
}

void lcd_init()

{
    int i;
    printf("Welcome to my CMPE240 Assignment - 3D_Cube_and_Sphere\n");
    // Set pins P0.16, P0.3, P0.22 as output
    LPC_GPIO0->FIODIR |= (0x1<<16);

    LPC_GPIO0->FIODIR |= (0x1<<3);

    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);

}

// Converting virtual X-Coordinate to physical X-Coordinate
int16_t xConvertToPhysical(int16_t x)
{
    x = x + (_width>>1);
    return x;
}

// Converting virtual Y-Coordinate to physical Y-Coordinate
int16_t yConvertToPhysical(int16_t y)
{
    y = (_height>>1) - y;
    return y;
}

void drawPixel(int16_t x, int16_t y, uint32_t color)

{
    // Convert Virtual to Physical
    x=xConvertToPhysical(x);
    y=yConvertToPhysical(y);

     if ((x < 0) || (x >= _width) || (y < 0) || (y >= _height))

     return;

     setAddrWindow(x, y, x + 1, y + 1);

     writecommand(ST7735_RAMWR);

     write888(color, 1);

}

/*****************************************************************************


** Descriptions:        Draw line function

**

** parameters:           Starting point (x0,y0), Ending point(x1,y1) and color

** Returned value:        None

**

*****************************************************************************/
```

```
void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint32_t color)

{

    int16_t slope = abs(y1 - y0) > abs(x1 - x0);

    if (slope) {

     swap(x0, y0);

     swap(x1, y1);

    }

    if (x0 > x1) {

     swap(x0, x1);

     swap(y0, y1);

    }

    int16_t dx, dy;

    dx = x1 - x0;

    dy = abs(y1 - y0);

    int16_t err = dx / 2;

    int16_t ystep;

    if (y0 < y1) {

     ystep = 1;
    }
    else {
     ystep = -1;
    }
    for (; x0 <= x1; x0++) {
     if (slope) {
      drawPixel(y0, x0, color);
     }
     else {
      drawPixel(x0, y0, color);
     }
     err -= dy;
     if (err < 0) {
      y0 += ystep;
      err += dx;
     }
    }
}
```

```
// Declare a structure for 3D
typedef struct
{
    float x_value; float y_value; float z_value;
}Pts3D;

// Declare a structure for 2D
typedef struct
{
    float x; float y;
}Pts2D;

// World to Viewer Transform method
Pts3D getWorld2Viewer(float WCS_X, float WCS_Y, float WCS_Z)
{
    Pts3D V;


    float Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Xe,2));

    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;

    V.x_value = -sPheta * WCS_X + cPheta * WCS_Y;
    V.y_value = -cPheta * cPhi * WCS_X - cPhi * sPheta * WCS_Y + sPhi * WCS_Z;
    V.z_value = -sPhi * cPheta * WCS_X - sPhi * cPheta * WCS_Y - cPhi * WCS_Z + Rho;

    return V;
}

// Viewer to Perspective Transform method
Pts2D getViewer2Perspective(float V_X, float V_Y, float V_Z)
{
    Pts2D P;
    P.x=V_X*(D_focal/V_Z);
    P.y=V_Y*(D_focal/V_Z);

    return P;
}

// method to draw the cube
void drawCube()
{
    #define UpperBD 52
    #define NumOfPts 10

    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;

    pworld WCS;
    pviewer V;
    pperspective P;
    Pts3D Vsingle;
    Pts2D Psingle;

    // Origin
    WCS.X[0]=0.0; WCS.Y[0]=0.0; WCS.Z[0]=0.0;

    // Axes
```

```
WCS.X[1]=200.0; WCS.Y[1]=0.0; WCS.Z[1]=0.0;
WCS.X[2]=0.0; WCS.Y[2]=200.0; WCS.Z[2]=0.0;
WCS.X[3]=0.0; WCS.Y[3]=0.0; WCS.Z[3]=200.0;

// Elevate Cube along Z_w axis by 10
WCS.X[4]=100.0; WCS.Y[4]=0; WCS.Z[4]=10.0;
WCS.X[5]=0.0; WCS.Y[5]=100.0; WCS.Z[5]=10.0;
WCS.X[6]=0.0; WCS.Y[6]=0.0; WCS.Z[6]=110.0;
WCS.X[7]=100.0; WCS.Y[7]=0.0; WCS.Z[7]=110.0;
WCS.X[8]=100.0; WCS.Y[8]=100.0; WCS.Z[8]=10.0;
WCS.X[9]=0.0; WCS.Y[9]=100.0; WCS.Z[9]=110.0;
WCS.X[10]=100.0; WCS.Y[10]=100.0; WCS.Z[10]=110.0;

// World to Viewer Transform for each point
for(int i=0;i<=NumOfPts;i++)
{
    Vsingle = getWorld2Viewer(WCS.X[i], WCS.Y[i], WCS.Z[i]);
    V.X[i] = Vsingle.x_value;
    V.Y[i] = Vsingle.y_value;
    V.Z[i] = Vsingle.z_value;
}

// Viewer to perspective transform for each point
for(int i=0;i<=NumOfPts;i++)
{
    Psingle = getViewer2Perspective(V.X[i], V.Y[i], V.Z[i]);
    P.X[i]=Psingle.x;
    P.Y[i]=Psingle.y;
}

// Draw Lines for all the edges of the cube
drawLine(P.X[0],P.Y[0],P.X[1],P.Y[1],RED);
drawLine(P.X[0],P.Y[0],P.X[2],P.Y[2],0x00FF00);
drawLine(P.X[0],P.Y[0],P.X[3],P.Y[3],0x0000FF);
drawLine(P.X[7],P.Y[7],P.X[4],P.Y[4],WHITE);
drawLine(P.X[7],P.Y[7],P.X[6],P.Y[6],WHITE);
drawLine(P.X[7],P.Y[7],P.X[10],P.Y[10],WHITE);
```

```
    drawLine(P.X[8],P.Y[8],P.X[4],P.Y[4],WHITE);
    drawLine(P.X[8],P.Y[8],P.X[5],P.Y[5],WHITE);
    drawLine(P.X[8],P.Y[8],P.X[10],P.Y[10],WHITE);
    drawLine(P.X[9],P.Y[9],P.X[6],P.Y[6],WHITE);
    drawLine(P.X[9],P.Y[9],P.X[5],P.Y[5],WHITE);
    drawLine(P.X[9],P.Y[9],P.X[10],P.Y[10],WHITE);
}

// Method to draw the half sphere using contours
void drawSphere()
{
    #define UpperBD 360
    #define UpperPCBD 240
    #define TotalPts 360
    #define NumOfLevels 10

    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;

    // Structure to hold the equi-distant points on each contour
    typedef struct
    {
        float X[UpperPCBD]; float Y[UpperPCBD]; float Z[UpperPCBD];
    }pcontour;
```

```
pworld WCS;
pviewer V;
pperspective P;
pcontour PC;
Pts3D Vsingle;
Pts2D Psingle;

float angle_theta;
int radius = 100;
int kx=0, ky=0, i, level;

for(level=0;level<=NumOfLevels-1;level++)
{
    for(i=0;i<TotalPts;i++)
    {
        // Logic to draw a circle using angles
        angle_theta = i*3.142 /180;
        WCS.X[i] = 0 + radius*cos(angle_theta);
        WCS.Y[i] =  0 + radius*sin(angle_theta);
        WCS.Z[i] = 10*level;    // Elevate the contour using Z_w each level
    }

    // World to Viewer Transform for each point
    for(i=0;i<TotalPts;i++)
    {
        Vsingle = getWorld2Viewer(WCS.X[i], WCS.Y[i], WCS.Z[i]);
        V.X[i] = Vsingle.x_value;
        V.Y[i] = Vsingle.y_value;
        V.Z[i] = Vsingle.z_value;
    }

    // Viewer to perspective transform for each point
    for(i=0;i<TotalPts;i++)
    {
        Psingle = getViewer2Perspective(V.X[i], V.Y[i], V.Z[i]);
        P.X[i]=Psingle.x;
        P.Y[i]=Psingle.y;
```

```
            drawPixel(P.X[i], P.Y[i], WHITE);

            // Logic to store a selected number of equi-distant points on each contour into PC.
            // In this case, for each contour, 24 points are selected and later joined.
            if(i%(TotalPts/24) == 0)
            {
                PC.X[kx++] = P.X[i];
                PC.Y[ky++] = P.Y[i];
            }
        }

        radius-=(level + 1);    // decrease radius of each contour when level increases and Z_w increases
    }

    // Logic to join the equi-distant points of one contour to the next contour
    for(i=0;i<kx;i++)
    {
        if((i+24)<kx)
        drawLine(PC.X[i], PC.Y[i], PC.X[i+24], PC.Y[i+24], WHITE);
    }
}

int main (void)
{
    uint32_t pnum = 0 ;

    if ( pnum == 0 )
        SSP0Init();
    else
         puts("Port number is not correct");

     lcd_init();

     fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, BLACK);

     drawCube();
    drawSphere();


    return 0;
}
```