CMPE 240 - Advanced Computer Design

Project: Shading and Diffuse Reflection

Name: Tirumala Saiteja Goruganthu

SJSU ID: 016707210

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

Professor: Harry Li

Date: 12/04/2022

Target Board: LPC1769

Purpose:

This project focuses on collecting data from a CPU Module to implement 3D visual designs on an LCD Display using the transformation pipeline. It mainly focuses on the 3D shading model and diffuse reflection computation. 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 compute and display shadows and diffuse reflection of a 3D world cube and a half-sphere. Following is the description of the tasks given to achieve the goal.

Given Tasks:

The final program should:

1. Generate a solid cube and a top half of the sphere.
2. The size of the cube with side length of 50 to begin with, and the virtual camera location E = (ex, ey, ez) = (150, 150, 100) for example. You can make some changes of these parameters to optimize visual display; The size for the sphere is defined with a radius R = 100. You can make some changes of their sizes to optimize visual display; Some note points are below
    a) Place the top half sphere on the origin of Xw-Yw-Zw;
    b) The orientation of the cube is defined with one of its sides in parallel with Zw-axis. The cube floats from Xw-Yw plane by 10 unit. and the center of the cube is at (80,80,35) to make room for the half sphere in the back.
    c) Define an arbitrary vector with P_i(0,0,35) and P_{i+1}=(200,220,40), and rotate the cube by 5 degrees clockwise.
3. Compute diffuse reflection on the top surface of the cube, use one primitive color, for example red.

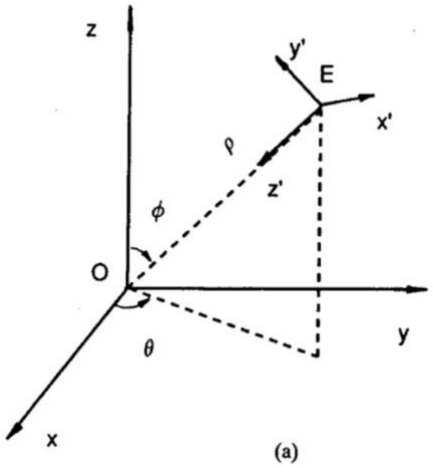
4. Place a point light source P_s(Xs, Ys, Zs) in the world coordinate system, you design its location, as a reference, you may consider P_s(Xs, Ys, Zs) = (-20, -20, 220) as a testing location and you can make adjustment of its location later accordingly.
5. Compute the ray equation and its intersection with the Xw-Yw plane. For the top surface of the cube there are 4 ray equations, each of the ray equations forms intersection point on the Xw-Yw plane. Keep track of this set of this points and produce a shade of dark blue by plotting a polygon.
6. Compute diffuse reflection on the top surface of the cube. Assuming reflectivity for red is 0.8 and for blue and green are 0.0.
7. Use scaling linear equation to scale up the diffuse reflection color from 20 to 255 for example with an offset = 20 or higher to make diffuse reflection with the best dynamic range of the color.
8. Use Linear decoration algorithm to place a tree on one of the two frontal surfaces of the cube per your choice.
9. Option Bonus Points to compute diffuse reflection on visible part of the half sphere (we done this part); or to decorate the half sphere by placing letter S (for SJSU) with your created font.

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:

1. 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:

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

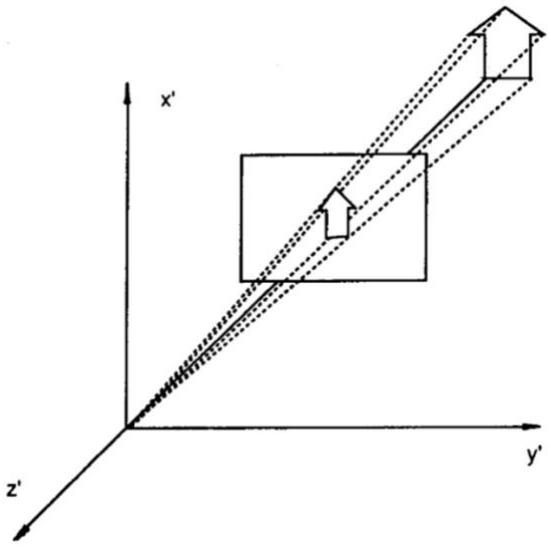
where, T = Transformation matrix which is defined as follows (referred from Prof. Harry Li IEEE paper)

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

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.

2. To compute the rotation of point P(x, y, z) with respect to arbitrary vector $P_i(x_i, y_i, z_i)$ and $P_{i+1}(x_{i+1}, y_{i+1}, z_{i+1})$, we must perform following 7 steps:
   a. Step-1 is to do the translation to move the arbitrary vector point $P_i(x_i, y_i, z_i)$ to the origin using the translation matrix. This translation changes the point $P_{i+1}(x_{i+1}, y_{i+1}, z_{i+1})$ which we must calculate because we use this new P_{i+1_T} point to calculate the cosine and sine values of the rotation matrix. We also must translate the given 3D point using this matrix to get new 3D point P_{pt_T}.
   b. Step-2 is to do the rotation of point P_{i+1_T} with respect to Zw-axis until the arbitrary vector is on Zw-Xw plane. This rotation changes the point to a new point P_{i+1_TRz} which we must calculate because we use this new P_{i+1_TRz} point to calculate the cosine and sine values of the rotation matrix. We also must rotate the P_{pt_T} using this matrix to get new 3D point P_{pt_TRz}.
   c. Step-3 is to do the rotation with respect to Yw-axis of the point P_{pt_TRz} using this matrix to get new 3D point P_{pt_TRzRy}.
   d. Step-4 is to do the rotation of alpha degrees with respect to Zw-axis of the point P_{pt_TRzRy} using this matrix to get new 3D point $P_{pt\_TRzRyRz}$.
   e. Step-5 is just reverting step-3 using the inverse of the matrix used in step-3.
   f. Step-6 is just reverting step-2 using the inverse of the matrix used in step-2.
   g. Step-7 is just reverting step-1 using the inverse of the matrix used in step-1.

3. To compute the diffuse reflection of a point P(x, y, z), we need to calculate the distance between that point and the point light source (P_s) and the angle made by the directional vector of the P and P_s and the normal vector n. Finally, the intensity value at that point is directly proportional to the angle and inversely proportional to the distance. Here, the proportionality constant is known as the reflectivity of the point P.
4. To compute the shade points, we first need to calculate the lambda value. This lambda value is therefore used in the calculation of ray equation for all 4 top surface points of the cube. These points are then sent through the transformation pipeline and plotted on the LCD display device.
5. To draw a tree on one of the visible surfaces of the cube, the linear decoration algorithm is used. This algorithm states that we first draw a tree by making an axis invisible and then after the tree points are calculated, just increase the coordinate value of the previously invisible axis to draw the tree on the cube.
6. As a bonus option, we also were able to successfully implement the computation of diffuse reflection on top half of the sphere.

Algorithm:

The Algorithm used in this project is as follows:

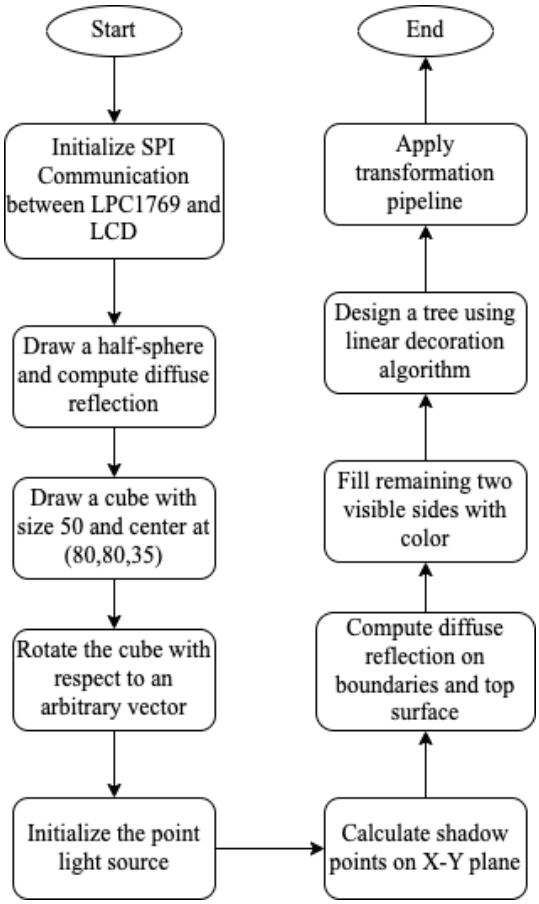
1. Initialize the SPI Communication between LPC1769 and the LCD display device. In this project, we used SSP0.
2. Since one of the tasks is to put the sphere behind the cube, we first draw the top half of the sphere using its own method. This method uses linear contour radius reducing algorithm to first draw a set of contours and then saved few points on each contour to connect them separately using drawLine method. Apart from this, as put forward earlier, we also implemented the diffuse reflection computation to compute the intensity on every point of the half sphere. We used reflectivity coefficients as (k_r, k_g, k_b) = (0, 1, 0).
3. Next step is to draw a cube. First, we initialized few data sets for the cube to centered at (80,80,35) as mentioned in the project task and then using these points, the following tasks are done.
4. After getting the data sets, we rotated the cube with a given arbitrary vector. The code to do this rotation has been written in a separate method to give the reader flexibility and understandability. Note that we are still in the world coordinate system.
5. Initialize a point to be the point-light source.
6. Next step is to compute the shade points of the top 4 points of the cube. This is done by first calculating lambda values for each point and then substituting these values in their corresponding ray equation formulae. This calculation of lambda and ray equations is done by two separate methods.
7. After completing the shadow computation, we then proceed to diffuse reflection computation where we use a separate method to calculate the intensity value at that world

point. Note that the diffuse reflection calculation is done on both the boundaries and the top surface fill of the cube.

8. Apart from that, we also took the liberty to fill remaining two visible sides of the cube with a color of our choice since the next step, which is drawing a tree, will not look good without a background.
9. Next step is to design a tree on one of the visible surfaces of the cube using the linear decoration algorithm. Also, I took the liberty to include an option for the user to set to control the projection of tree on either of the two visible surfaces. In this project, we have written three methods to implement the draw tree task.
10. The final step is to use the transformation pipeline concept to display the world coordinates onto an LCD display device. This is generally a two-step process where first world to viewer transformation is applied and then viewer to perspective projection is applied. In this project, we implemented this using two methods to compute the above-mentioned tasks.

Flowchart:

The Flowchart used in this project is as follows (drawn using online tool):



Pseudo Code:

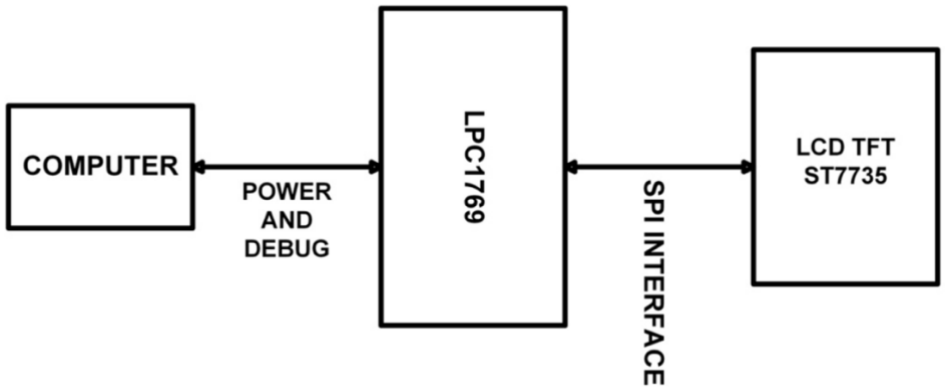
The Pseudo code used in this project is as follows:

1. Initialize eye coordinates, focal length and the point light source coordinate.
2. spiwrite() to initialize the SPI protocol through SSP.
3. fillrect() to fill the whole rectangle of the given coordinates.
4. lcd_init() to initialize the LCD display device by performing a set of hardware reset sequence and also to initialize the buffers.
5. xConvertToPhysical() is used to convert virtual X-coordinate to physical X-coordinate.
6. yConvertToPhysical() is used to convert virtual Y-coordinate to physical Y-coordinate.
7. drawPixel() method to plot a given pixel on the display device.
8. drawLine() method is used to draw a line by taking starting and ending points as its inputs.
9. Lambda3D() method is used to calculate the Lambda value in Ray equation calculation.
10. shadowPoint3D() method is used to compute the ray equation for shadow point calculation.
11. getDiffuseColor() method is used to is used to compute the diffuse reflection color with given reflectivity coefficients.
12. getDiffuseColorGreen() method is used to is used to compute the diffuse reflection color with given reflectivity coefficients.
13. getWorld2Viewer() is used to convert the points from world coordinate system to viewer coordinate system.
14. getViewer2perspective() is used to convert the points from viewer coordinate system to perspective projection.
15. get3DTransform() method is used to convert the points from world coordinate system to perspective projection.
16. rotate_pointIn3D() method is used to rotate a given point using 3 step algorithm.
17. drawTree() method is used to design a tree where its trunk and its branches are designed and implemented in two different methods.

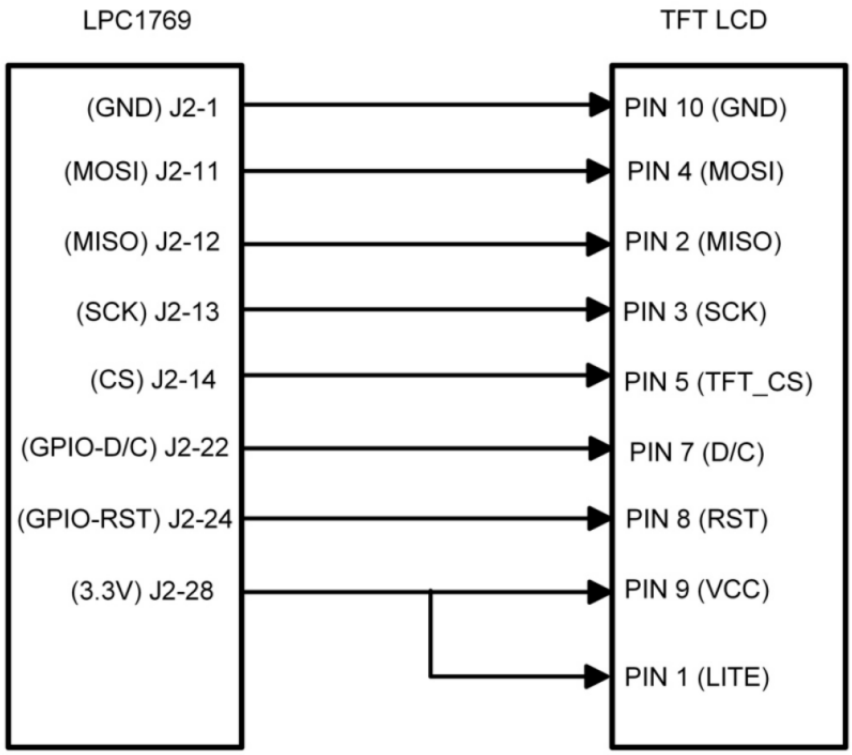
18. designTreeTrunkIn3D() method is used to design a Tree using drawline method and rotatepoint method. Here, we use recursion to implement multiple levels of branch generation for a tree.
19. rotate3DZwAxis() method is used to rotate the cube with respect to Zw axis alone. Note that, this method is written to visualize the arbitrary vector rotation written below.
20. rotateCoord3D() method is used to rotate the cube with respect to an arbitrary vector.
21. drawCube() method is used to draw a cube by implementing all the tasks given in the project tasks list.
22. drawSphere() method is used to draw the half sphere using contours. 40 points on each contour are saved in an array and later joined with another 40 points of the corresponding upper contour to get a dome-like structure.
23. Finally, the main() method to start the whole program. The drawSphere and drawCube methods are initialized here in that respective order.

Rubrics - Block Diagrams:

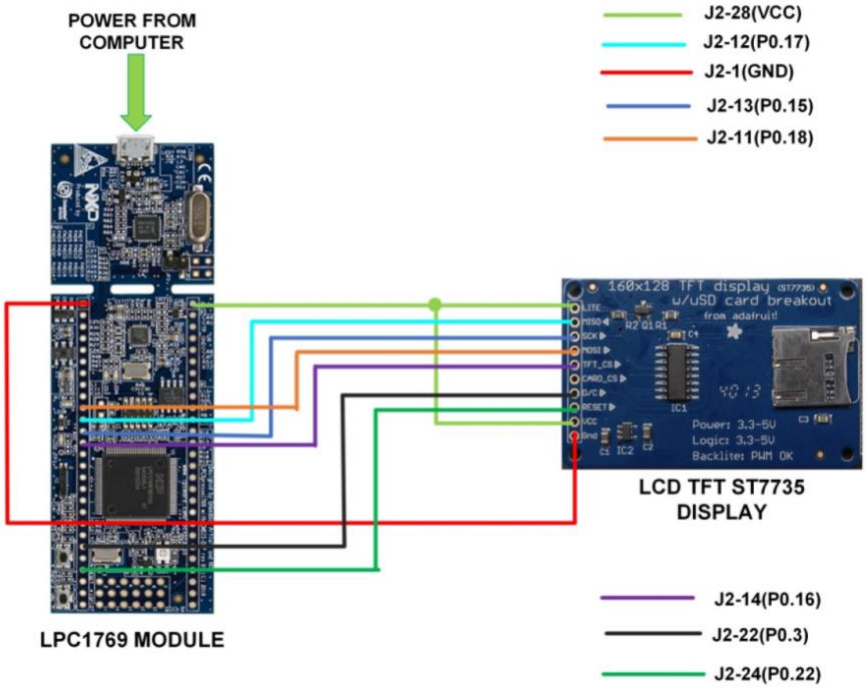
1. The block diagram of the entire system setup including the computer block (Done using Visio tool):



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:

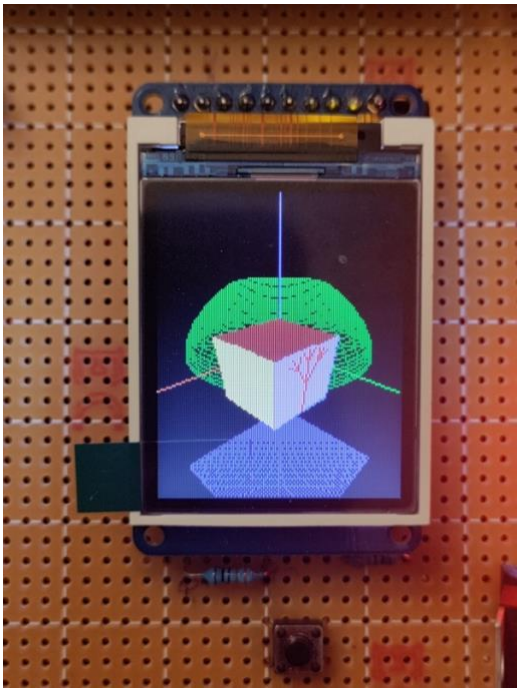
LPC1769 PINS (JTAG PINS)	LCD TFT ST7735
VCC (J2-28)	LITE
P0.17 (J2-12)	MISO
P0.15 (J2-13)	SCK
P0.18 (J2-11)	MOSI
P0.16 (J2-14)	TFT_CS
Not Connected	CARD_CS
P0.3 (J2-22)	D/C
P0.22 (J2-24)	RESET
VCC (J2-28)	VCC
GND (J2-1)	GND

Testing and Verification:

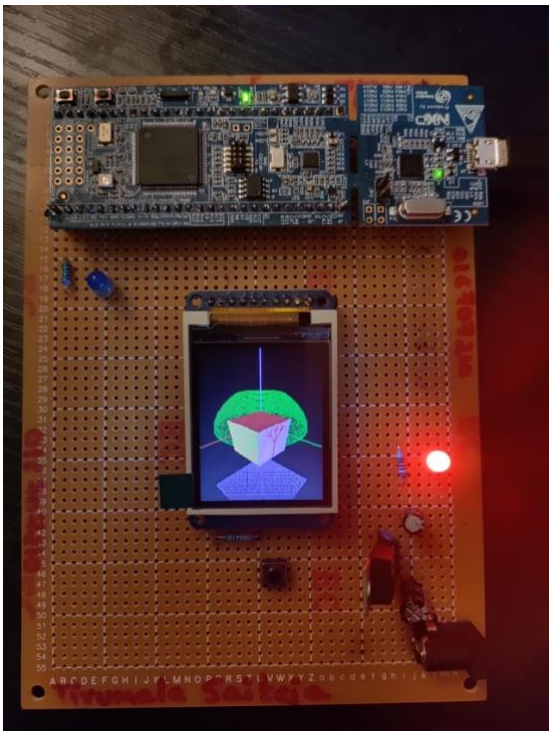
1. This section provides testing and verification procedure for 3D shadow and diffuse reflection computation on both cube and half-sphere. The LPC1769 module was connected to the laptop via a USB connector and Vcc and corresponding current started flowing to power up an LED (soldered for testing and debugging purposes) and the LCD display device. The files *ssp.c* and *ssp.h* are seen as helper or header files which instantiates the SPI protocol before the communication.
2. The project was built and debugged successfully without any errors with successful link connection to the LPC1769 module. After running the code, we first see a half-sphere being drawn with diffuse reflection intensity value applied. It shows up as green since the reflectivity coefficient for green is set as 1 and remaining are set as 0.
3. Next, an arbitrary vector rotated cube was drawn with top surface having diffuse reflection intensity color as red with reflectivity coefficient as 0.8.
4. After the cube, the shadow boundaries appeared and right after that, the shadow fill code worked its charm to fill in dark blue inside the shadow boundaries.
5. Just to make sure the tree looks better, the fill for other two visible surfaces of the cube will be filled with a color of own choice.
6. Finally, the tree is drawn on any one of the visible surfaces of the cube. Note that in my code, user can provide which face he wants to fill the tree with.

Output Snippets:

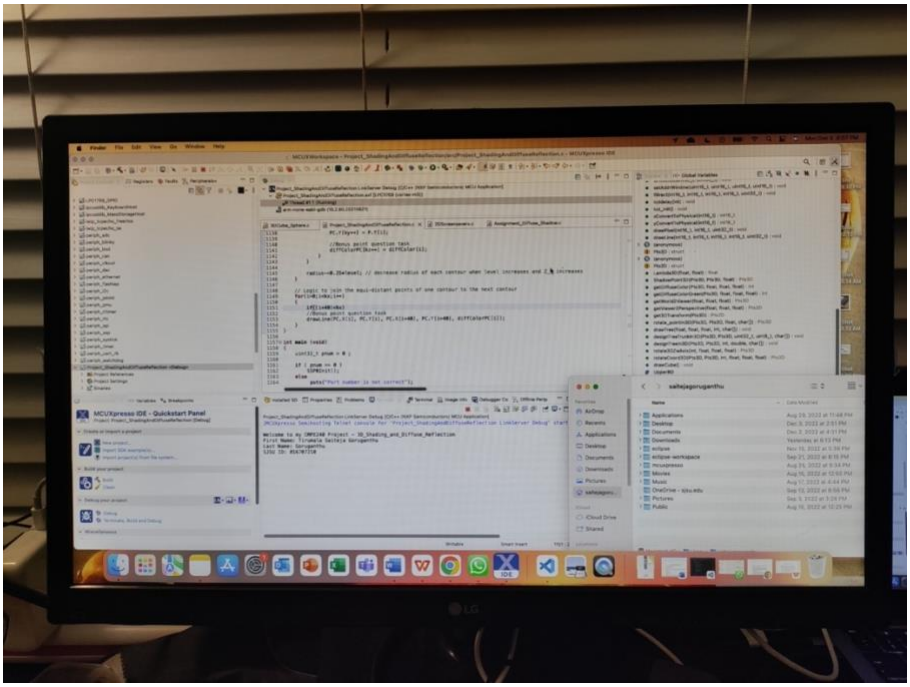
1. Picture of the LCD device with the final output.



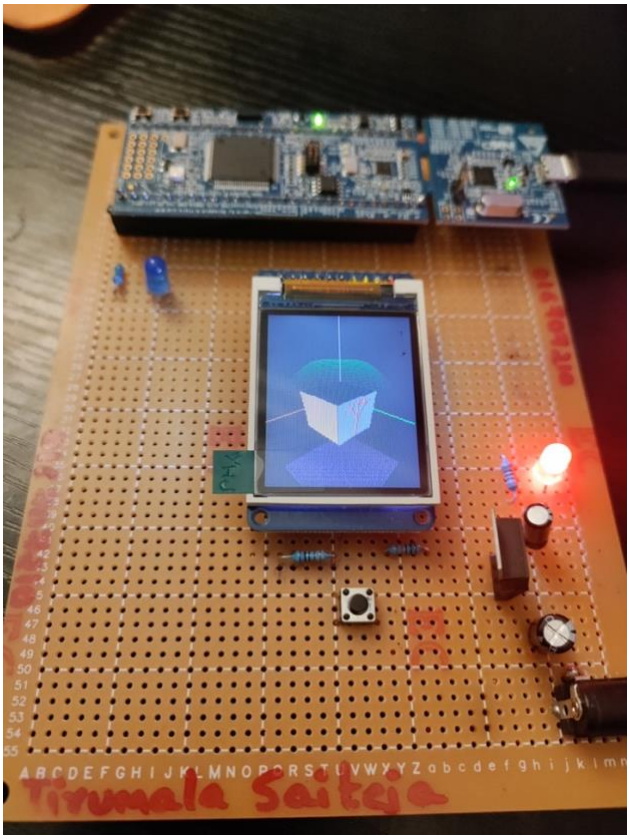
2. Picture of the prototype system with the final output



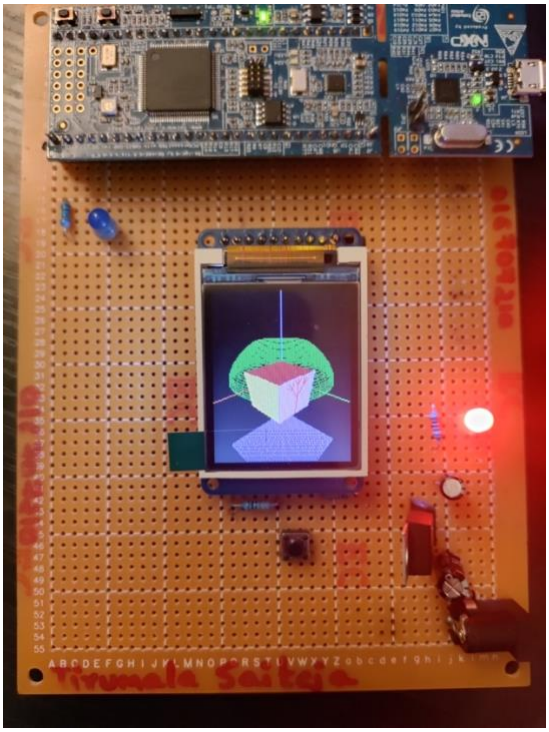
3. Picture of the entire system with my folder opened



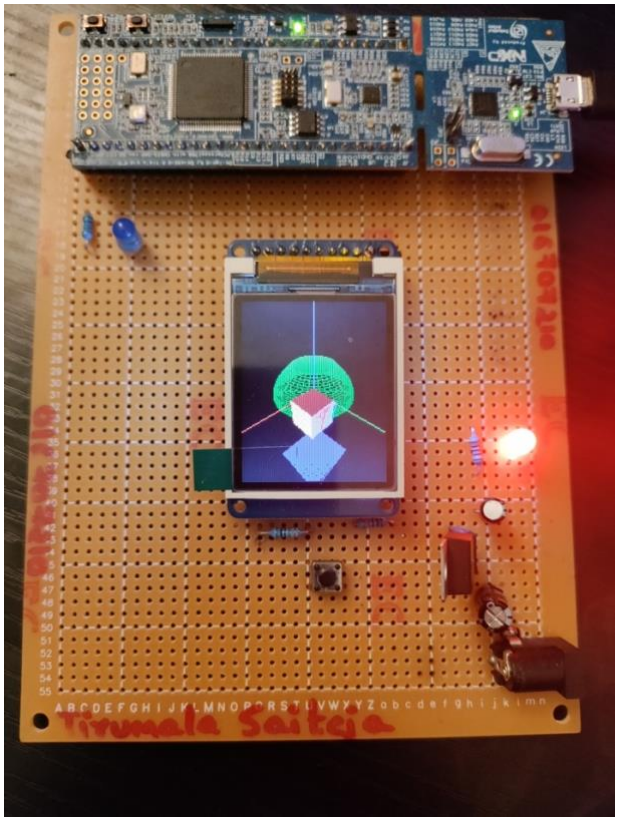
4. Picture of the Final output angled enough to show the diffuse reflection dimmed



5. Picture of the Final output angled 90^{o} (normal) to show the diffuse reflection



6. Picture of the final output with different eye coordinates i.e., (200, 200, 200)



Conclusion:

The design and implementation of the 3D shadow and diffuse reflection computations on a cube and a half-sphere is completed successfully. The data from the computer was transmitted to LCD through LPC1769 using the SPI protocol. The LCD displays the data correctly. Therefore, upon completion of the experiment, we understand how to implement or calculate shadow points and diffuse reflection points through code and understood few concepts of important algorithms like linear decoration algorithm. Apart from that, we were able to implement desired 3D graphics using transformation pipeline and communicate it with LCD device for displaying purposes. The experiment related source code has been included in the appendix.

APPENDIX

Source Code:

```
/*
============================================================================
 Name        : Project_ShadingAndDiffuseReflection.c
 Author      : Tirumala Saiteja Goruganthu
 Version     :
 Copyright   : $(copyright)
 Description : This project is for 3D shading model and diffuse reflection computation on
 LPC1769 micro processor platform for the purpose of design 3D Graphics Processing Engine.

 This project consists of the following tasks:

 1. Generate a Solid cube and a top half of a sphere.

 2. The cube must be of size 50 with virtual camera location coordinates being (200,200,200).

 3. The size of the sphere is defined with radius R = 100. The top half of the sphere must be
    placed on the origin. The orientation of the cube is defined with one of its side in
    parallel with zw-axis. The cube floats from xw-yw plane by 10 unit, and the center of
    the cube is at (80,80,35) to make room for the half sphere in the back.

 4. After setting the above parameters, rotate the cube by 5 degree clockwise with respect to
     the arbitrary vector P_i(0,0,35) and P_i+1(200,220,40). This is achieved in the
     method 'rotateCoord3D.'

 5. After rotating the cube, compute the diffuse reflection on the top surface of the cube using
    one primitive color (Example: red). This is achieved in the method 'getDiffuseColor.'

 6. Now by placing the point light source coordinates at (-20,-20, 220), compute 4 ray equations
    on top 4 points of the cube and its intersection of xw-yw plane. keep track this set of
    4 points and produce a shade of dark blue by plotting a polygon. This is achieved in the
    method 'ShadowPoint3D.'

 7. Compute diffuse reflection on the top surface of the cube. Assuming reflectivity for red
```

```
    is 0.8 and for blue and green are 0.0. This is achieved in the method 'getDiffuseColor.'

 8. Use scaling linear equation to scale up the diffuse reflection color from 20 to 255
    for example with an offset = 20 or higher to make diffuse reflection with the best
    dynamic range of the color. This is achieved in the method 'getDiffuseColor.'

 9. Using Linear Decoration Algorithm, place a tree on one of the 2 frontal surfaces of the cube.
    This is achieved in the method 'drawTree.'

 10. Optional Bonus point Question is also done:
     compute diffuse reflection on visible part of the half sphere. This is achieved in the
     method 'getDiffuseColor.'
================================================================================
*/

#include <LPC17xx.h>                    /* LPC17xx definitions */
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "ssp.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
#define GREY1 0x7A7C7B
#define BLUE1 0x3999FF

// 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
```

```
#define pi 3.1416

int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

// Defining the eye co-ordinates
//float Xe=250, Ye=100, Ze=60;
//float Xe=100, Ye=250, Ze=60; //for a better tree view
float Xe=150, Ye=150, Ze=100;

// Defining the focal length
float D_focal=120;

// Defining the point light source coordinates
float Psx=-20, Psy=-20, Psz=220;

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 Project - 3D_Shading_and_Diffuse_Reflection\n");
    printf("First Name: Tirumala Saiteja Goruganthu\n");
    printf("Last Name: Goruganthu\n");
    printf("SJSU ID: 016707210\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;

// This method is used to calculate the Lambda value in Ray equation calculation
float Lambda3D(float Zi,float Zs)
{
    float lambda;
    lambda = -Zi/(Zs-Zi);
    return lambda;
}

// This method is used to compute the ray equation for shadow point calculation
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;
}

// This method is used to compute the diffuse reflection color with given reflectivity coefficients
int getDiffuseColor(Pts3D Pi, float reflectivity_r, float reflectivity_g, float reflectivity_b)
{
    uint32_t print_diffuse_color;
    Pts3D Ps;
```

```
    Ps.x_value = Psx;
    Ps.y_value = Psy;
    Ps.z_value = Psz;
    int diff_red, diff_green, diff_blue;
    float new_red, new_green, new_blue, scaling = 16000;

    //Calculate the diffuse reflection for point Pi
    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));

    //Scale the above result
    temp *= scaling;
    diff_red = reflectivity_r * temp * 255;
    diff_green = reflectivity_g * temp * 255;
    diff_blue  = reflectivity_b * temp * 255;

    //Shift the above values into their spectrum. Eg: shift 16 bits to put value in red spectrum.
    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;
}

// This method is used to compute the diffuse reflection color with given reflectivity coefficients
int getDiffuseColorGreen(Pts3D Pi, float reflectivity_r, float reflectivity_g, float reflectivity_b)
{
    uint32_t print_diffuse_color;
    Pts3D Ps;
    Ps.x_value = Psx;
    Ps.y_value = Psy;
    Ps.z_value = Psz;
    int diff_red, diff_green, diff_blue;
    float new_red, new_green, new_blue, scaling = 16000;
```

```
    //Calculate the diffuse reflection for point Pi
    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));

    //Scale the above result
    temp *= scaling;
    diff_red = reflectivity_r * temp * 255;
    diff_green = reflectivity_g * temp * 255;
    diff_blue  = reflectivity_b * temp * 255;

    //Shift the above values into their spectrum. Eg: shift 16 bits to put value in red spectrum.
    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;
}

// 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;
}

// World to Viewer to Perspective Transform method
Pts2D get3DTransform(Pts3D Pi)
{
    float Rho=sqrt(pow(Xe,2)+pow(Ye,2)+pow(Xe,2));

    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;

}

/* Rotate point p with respect to o and angle <angle> */
Pts3D rotate_pointIn3D(Pts3D p, Pts3D o, float angle, char showOn[])
{
    Pts3D rt, t, new;

    char front[] = "Front";
    char right[] = "Right";

    float s = sin(angle);
    float c = cos(angle);

    /*
     * The below commented part is used to project the tree onto front side of the cube
     */
    if(strcmp(showOn, front) == 0)
    {
```

```
    //translate point to origin
    t.x_value = o.x_value;
    t.y_value = p.y_value - o.y_value;
    t.z_value = p.z_value - o.z_value;

    rt.x_value = o.x_value;
    rt.y_value = t.y_value * c - t.z_value * s;
    rt.z_value = t.y_value * s + t.z_value * c;

    //translate point back
    new.x_value = o.x_value;
    new.y_value = rt.y_value + o.y_value;
    new.z_value = rt.z_value + o.z_value;
}


/*
 * The below commented part is used to project the tree onto right side of the cube
 */
if(strcmp(showOn, right) == 0)
{
    //translate point to origin
    t.x_value = p.x_value - o.x_value;
    t.y_value = o.y_value;
    t.z_value = p.z_value - o.z_value;

    rt.x_value = t.x_value * c - t.z_value * s;
    rt.y_value = o.y_value;
    rt.z_value = t.x_value * s + t.z_value * c;

    //translate point back
    new.x_value = rt.x_value + o.x_value;
    new.y_value = o.y_value;
    new.z_value = rt.z_value + o.z_value;
}

return new;
```

```
}

// This method is used to draw a tree onto a side of the cube
void drawTree(float xstart, float ystart, float zstart, int cube_side, char showOn[])
{
    Pts3D start3D, end3D;

    double lambda = 0.6;
    char front[] = "Front";
    char right[] = "Right";

    /*
     * The below commented part is used to project the tree onto front side of the cube
     */
    if(strcmp(showOn, front) == 0)
    {
        start3D.x_value = xstart + cube_side;
        start3D.y_value = ystart + (cube_side/2);
        start3D.z_value = zstart;

        end3D.x_value = xstart + cube_side;
        end3D.y_value = ystart + (cube_side/2);
        end3D.z_value = zstart + (cube_side/2);
    }

    /*
     * The below commented part is used to project the tree onto right side of the cube
     */
    if(strcmp(showOn, right) == 0)
    {
        start3D.x_value = xstart + (cube_side/2);
        start3D.y_value = ystart + cube_side;
        start3D.z_value = zstart;

        end3D.x_value = xstart + (cube_side/2);
        end3D.y_value = ystart + cube_side;
        end3D.z_value = zstart + (cube_side/2);
```

```
    }

    designTreeTrunkIn3D(start3D, end3D, RED, 1, showOn);
    designTreeIn3D(start3D, end3D, 2, lambda, showOn);
}

/* Design the branch of a tree */
void designTreeTrunkIn3D(Pts3D start3D, Pts3D end3D, uint32_t color, uint8_t thickness, char showOn[])
{
    Pts2D lcd, start, end;

    lcd = get3DTransform(start3D);

    start.x = lcd.x;
    start.y = lcd.y;

    lcd = get3DTransform(end3D);

    end.x = lcd.x;
    end.y = lcd.y;

    int i;
    for(i = 0; i < thickness; i++)
        drawLine(start.x + i, start.y, end.x + i, end.y, color);
}

/* Design a Tree using drawline method and rotatepoint method */
void designTreeIn3D(Pts3D start3D, Pts3D end3D, int level, double lambda, char showOn[])
{
    Pts2D lcd, c, start, end;
    Pts3D c3D, rtl3D, rtr3D;

    char front[] = "Front";
    char right[] = "Right";

    if(level == 0)
        return;
```

```
lcd = get3DTransform(start3D);

start.x = lcd.x;
start.y = lcd.y;

lcd = get3DTransform(end3D);

end.x = lcd.x;
end.y = lcd.y;

/*
 * The below commented part is used to project the tree onto front side of the cube
 */
if(strcmp(showOn, front) == 0)
{
    c3D.x_value = start3D.x_value;
    c3D.y_value = end3D.y_value + (lambda*(end3D.y_value - start3D.y_value));
    c3D.z_value = end3D.z_value + (lambda*(end3D.z_value - start3D.z_value));
}

/*
 * The below commented part is used to project the tree onto right side of the cube
 */
if(strcmp(showOn, right) == 0)
{
    c3D.x_value = end3D.x_value + (lambda*(end3D.x_value - start3D.x_value));
    c3D.y_value = start3D.y_value;
    c3D.z_value = end3D.z_value + (lambda*(end3D.z_value - start3D.z_value));
}

lcd = get3DTransform(c3D);

c.x = lcd.x;
c.y = lcd.y;

drawLine(c.x, c.y, end.x, end.y, RED);
```

```
    designTreeIn3D(end3D, c3D, level - 1, lambda, showOn);

    rtl3D = rotate_pointIn3D(c3D, end3D, pi/6, showOn);

    lcd = get3DTransform(rtl3D);

    drawLine(lcd.x, lcd.y, end.x, end.y, RED);
    designTreeIn3D(end3D, rtl3D, level - 1, lambda, showOn);

    rtr3D = rotate_pointIn3D(c3D, end3D, -pi/6, showOn);

    lcd = get3DTransform(rtr3D);

    drawLine(lcd.x, lcd.y, end.x, end.y, RED);
    designTreeIn3D(end3D, rtr3D, level - 1, lambda, showOn);
}

// This method is used to rotate the cube with respect to Zw axis alone
// Note that, this method is written to visualize the arbitrary vector rotation written below
Pts3D rotate3DZwAxis(int angle, float cube_x, float cube_y, float cube_z)
{
    Pts3D final;
    float angle_rad;

    angle_rad = (pi * angle) / 180;

    final.x_value = cube_x*cos(angle_rad) - cube_y*sin(angle_rad);
    final.y_value = cube_y*cos(angle_rad) + cube_x*sin(angle_rad);
    final.z_value = cube_z;

    return final;
}

//  This method is used to rotate the cube with respect to an arbitrary vector
Pts3D rotateCoord3D(Pts3D ARBPi, Pts3D ARBPi1, int angle, float cube_x, float cube_y, float cube_z)
{
    Pts3D ARBPi1_T, ARBPi1_R;
```

```
  Pts3D cube_T, cube_Rzw, cube_Ryw, cube_Rzw_main, cube_Ryw_reverse, cube_Rzw_reverse, cube_Treverse;

  float cos_alpha_dash, sin_alpha_dash, cos_alpha_doubledash, sin_alpha_doubledash;

  float denom1, denom2;

  float angle_rad;

  //Step1: Translation
    //Find new Pi+1 point after Translation for calculating cos(alpha_dash) and sin(alpha_dash) values
    ARBPi1_T.x_value = ARBPi1.x_value - ARBPi.x_value;
    ARBPi1_T.y_value = ARBPi1.y_value - ARBPi.y_value;
    ARBPi1_T.z_value = ARBPi1.z_value - ARBPi.z_value;

    //Multiply Translation matrix with the given cube point
    cube_T.x_value = cube_x - ARBPi.x_value;
    cube_T.y_value = cube_y - ARBPi.y_value;
    cube_T.z_value = cube_z - ARBPi.z_value;

  //Step2: Rotation with respect to Zw axis (Note that here alpha_dash will be negative since rotation is clockwise =>
so this changes the rotation_Zw matrix)
    //Calculate cos(alpha_dash) and sin(alpha_dash) values
    denom1 = sqrt(pow(ARBPi1_T.x_value,2) + pow(ARBPi1_T.y_value,2));
    cos_alpha_dash = ARBPi1_T.x_value / denom1;
    sin_alpha_dash = ARBPi1_T.y_value / denom1;

    //Find new Pi+1 point after Rotation. This new point is used in calculating the cos(alpha_doubledash) and
sin(alpha_doubledash) values
    ARBPi1_R.x_value = ARBPi1_T.x_value*cos_alpha_dash + ARBPi1_T.y_value*sin_alpha_dash;
    ARBPi1_R.y_value = ARBPi1_T.y_value*cos_alpha_dash - ARBPi1_T.x_value*sin_alpha_dash;
    ARBPi1_R.z_value = ARBPi1_T.z_value;

    //Multiply Rotation_Zw matrix with the translated cube point
    cube_Rzw.x_value = cube_T.x_value*cos_alpha_dash + cube_T.y_value*sin_alpha_dash;
    cube_Rzw.y_value = cube_T.y_value*cos_alpha_dash - cube_T.x_value*sin_alpha_dash;
    cube_Rzw.z_value = cube_T.z_value;
```

```
   //Step3: Rotation with respect to Yw axis (Note that here alpha_doubledash will be negative since rotation is clockwise => so this changes the rotation_Yw matrix)
       //Calculate cos(alpha_dash) and sin(alpha_dash) values
       denom2 = sqrt(pow(ARBPi1_R.x_value,2) + pow(ARBPi1_R.z_value,2));
       cos_alpha_doubledash = ARBPi1_R.z_value / denom2;
       sin_alpha_doubledash = ARBPi1_R.x_value / denom2;

       //Multiply Rotation_Yw matrix with the rotated_Zw cube point
       cube_Ryw.x_value = cube_Rzw.x_value*cos_alpha_doubledash - cube_Rzw.z_value*sin_alpha_doubledash;
       cube_Ryw.y_value = cube_Rzw.y_value;
       cube_Ryw.z_value = cube_Rzw.z_value*cos_alpha_doubledash + cube_Rzw.x_value*sin_alpha_doubledash;

   //Step4: Rotation with respect to Zw axis
       //change given angle from degrees to radians
       angle_rad = (pi * angle) / 180;

       //Multiply Rotation_Yw matrix with the rotated_Yw cube point
       cube_Rzw_main.x_value = cube_Ryw.x_value*cos(angle_rad) - cube_Ryw.y_value*sin(angle_rad);
       cube_Rzw_main.y_value = cube_Ryw.y_value*cos(angle_rad) + cube_Ryw.x_value*sin(angle_rad);
       cube_Rzw_main.z_value = cube_Ryw.z_value;

   //Step5: Revert back the Rotation with respect to Yw axis
       //Multiply Rotation_Yw matrix with the rotated_Zw_main cube point
       cube_Ryw_reverse.x_value = cube_Rzw_main.x_value*cos_alpha_doubledash + cube_Rzw_main.z_value*sin_alpha_doubledash;
       cube_Ryw_reverse.y_value = cube_Rzw_main.y_value;
       cube_Ryw_reverse.z_value = cube_Rzw_main.z_value*cos_alpha_doubledash - cube_Rzw_main.x_value*sin_alpha_doubledash;

   //Step6: Revert back the Rotation with respect to Zw axis
       //Multiply Rotation_Yw matrix with the rotated_Yw_reverse cube point
       cube_Rzw_reverse.x_value = cube_Ryw_reverse.x_value*cos_alpha_dash - cube_Ryw_reverse.y_value*sin_alpha_dash;
       cube_Rzw_reverse.y_value = cube_Ryw_reverse.y_value*cos_alpha_dash + cube_Ryw_reverse.x_value*sin_alpha_dash;
       cube_Rzw_reverse.z_value = cube_Ryw_reverse.z_value;
```

```
    //Step7: Revert back the Translation
        //Multiply Translation matrix with the given cube point
        cube_Treverse.x_value = cube_Rzw_reverse.x_value + ARBPi.x_value;
        cube_Treverse.y_value = cube_Rzw_reverse.y_value + ARBPi.y_value;
        cube_Treverse.z_value = cube_Rzw_reverse.z_value + ARBPi.z_value;

    return cube_Treverse;
}

// method to draw the cube
void drawCube()
{
    #define UpperBD 52
    #define NumOfPts 16

    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, RWCS, ARBPi, ARBPi1;
    Pts2D Psingle;
    float_t L1,L2,L3,L4;
    int angle, i;
```

```
angle = -5; //minus for clockwise

// 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;

// New points to define the cube center as (80,80,35)
WCS.X[4]=55.0; WCS.Y[4]=55.0; WCS.Z[4]=10.0;
WCS.X[5]=55.0; WCS.Y[5]=55.0; WCS.Z[5]=60.0;
WCS.X[6]=55.0; WCS.Y[6]=105.0; WCS.Z[6]=10.0;
WCS.X[7]=55.0; WCS.Y[7]=105.0; WCS.Z[7]=60.0;
WCS.X[8]=105.0; WCS.Y[8]=55.0; WCS.Z[8]=10.0;
WCS.X[9]=105.0; WCS.Y[9]=55.0; WCS.Z[9]=60.0;
WCS.X[10]=105.0; WCS.Y[10]=105.0; WCS.Z[10]=10.0;
WCS.X[11]=105.0; WCS.Y[11]=105.0; WCS.Z[11]=60.0;

//Define given Arbitrary vectors
ARBPi.x_value=0.0; ARBPi.y_value=0.0; ARBPi.z_value=35.0;
ARBPi1.x_value=200.0; ARBPi1.y_value=220.0; ARBPi1.z_value=40.0;

//For loop to rotate each vertex of the cube and get new coordinates
for(i=4;i<12;i++)
{
   RWCS = rotateCoord3D(ARBPi, ARBPi1, angle, WCS.X[i], WCS.Y[i], WCS.Z[i]);
   //RWCS = rotate3DZwAxis(angle, WCS.X[i], WCS.Y[i], WCS.Z[i]);
   WCS.X[i] = RWCS.x_value;
   WCS.Y[i] = RWCS.y_value;
   WCS.Z[i] = RWCS.z_value;
}

//Shadow Calculation
```

```
// Point of Light Source PS(-20, -20, 220)
WCS.X[12]=Psx; WCS.Y[12]=Psy; WCS.Z[12]=Psz;

Pts3D Ps;
Ps.x_value = WCS.X[12]; Ps.y_value = WCS.Y[12]; Ps.z_value = WCS.Z[12];

Pts3D P1;
P1.x_value = WCS.X[5]; P1.y_value = WCS.Y[5]; P1.z_value = WCS.Z[5];

Pts3D P2;
P2.x_value = WCS.X[9]; P2.y_value = WCS.Y[9]; P2.z_value = WCS.Z[9];

Pts3D P3;
P3.x_value = WCS.X[11]; P3.y_value = WCS.Y[11]; P3.z_value = WCS.Z[11];

Pts3D P4;
P4.x_value = WCS.X[7]; P4.y_value = WCS.Y[7]; P4.z_value = WCS.Z[7];

// Shadow Points

Pts3D S1,S2,S3,S4;
L1 = Lambda3D(WCS.Z[5],WCS.Z[12]);
L2 = Lambda3D(WCS.Z[9],WCS.Z[12]);
L3 = Lambda3D(WCS.Z[11],WCS.Z[12]);
L4 = Lambda3D(WCS.Z[7],WCS.Z[12]);
S1 = ShadowPoint3D(P1,Ps,L1);
S2 = ShadowPoint3D(P2,Ps,L2);
S3 = ShadowPoint3D(P3,Ps,L3);
S4 = ShadowPoint3D(P4,Ps,L4);

WCS.X[13]=S1.x_value;     WCS.Y[13]=S1.y_value;     WCS.Z[13]=S1.z_value; //S1
WCS.X[14]=S2.x_value;     WCS.Y[14]=S2.y_value;     WCS.Z[14]=S2.z_value; //S2
WCS.X[15]=S3.x_value;     WCS.Y[15]=S3.y_value;     WCS.Z[15]=S3.z_value; //S3
WCS.X[16]=S4.x_value;     WCS.Y[16]=S4.y_value;     WCS.Z[16]=S4.z_value; //S4

// 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;
}

Pts3D temp_pt;

temp_pt.x_value = WCS.X[5]; temp_pt.y_value = WCS.Y[5]; temp_pt.z_value = WCS.Z[5];

float temp_color;

temp_color = getDiffuseColor(temp_pt, 0.8, 0.0, 0.0);

// 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);

//New Centered Cube DrawLines
drawLine(P.X[6],P.Y[6],P.X[4],P.Y[4],WHITE);
drawLine(P.X[10],P.Y[10],P.X[8],P.Y[8],WHITE);
drawLine(P.X[6],P.Y[6],P.X[10],P.Y[10],BLUE);
drawLine(P.X[8],P.Y[8],P.X[4],P.Y[4],WHITE);

drawLine(P.X[7],P.Y[7],P.X[5],P.Y[5],temp_color);
drawLine(P.X[7],P.Y[7],P.X[11],P.Y[11],WHITE);
drawLine(P.X[9],P.Y[9],P.X[11],P.Y[11],WHITE);
```

```
drawLine(P.X[9],P.Y[9],P.X[5],P.Y[5],temp_color);

drawLine(P.X[9],P.Y[9],P.X[8],P.Y[8],temp_color);
drawLine(P.X[11],P.Y[11],P.X[10],P.Y[10],WHITE);
drawLine(P.X[5],P.Y[5],P.X[4],P.Y[4],temp_color);
drawLine(P.X[7],P.Y[7],P.X[6],P.Y[6],BLUE);

// Shadow Drawlines
drawLine(P.X[13],P.Y[13],P.X[14],P.Y[14],DARKBLUE);
drawLine(P.X[14],P.Y[14],P.X[15],P.Y[15],DARKBLUE);
drawLine(P.X[16],P.Y[16],P.X[15],P.Y[15],DARKBLUE);
drawLine(P.X[16],P.Y[16],P.X[13],P.Y[13],DARKBLUE);

double xvaluestart = S1.x_value;
double xvalueend = S2.x_value;
double yvaluestart = S1.y_value;
double yvalueend = S4.y_value;

//Shadow fill
for(double x=xvaluestart;x<=xvalueend;x+=1.234)
   for(double y=yvaluestart;y<=yvalueend;y+=1.234)
   {
      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,DARKBLUE);
   }

//Red - top side diffuse reflection fill
float diff_color;
for(float y=WCS.Y[5];y<=WCS.Y[6];y+=0.9888)
   for(float x=WCS.X[4];x<=WCS.X[8];x+=0.9888)
   {
      Pts3D pt; Pts2D pt2d;
      pt.x_value=x; pt.y_value=y; pt.z_value=WCS.Z[5];
      pt2d = get3DTransform(pt);
      // RED DIFFUSE
```

```
            diff_color = getDiffuseColor(pt, 0.8, 0.0, 0.0);
            drawPixel(pt2d.x,pt2d.y,diff_color);
        }

    //Front side Fill
    for(float y=WCS.Y[5];y<=WCS.Y[6];y+=0.988)
        for(float z=WCS.Z[5];z>=WCS.Z[4];z-=0.988)
        {
            Pts3D pt; Pts2D pt2d;
            pt.x_value=WCS.X[8]; pt.y_value=y; pt.z_value=z;
            pt2d = get3DTransform(pt);
            drawPixel(pt2d.x,pt2d.y,0xf59105);
        }

    //Right side Fill
    for(float z=WCS.Z[5];z>=WCS.Z[4];z-=0.988)
        for(float x=WCS.X[8];x>=WCS.X[4];x-=0.988)
        {
            Pts3D pt; Pts2D pt2d;
            pt.x_value=x; pt.y_value=WCS.Y[6]; pt.z_value=z;
            pt2d = get3DTransform(pt);
            drawPixel(pt2d.x,pt2d.y,0x5905f5);
        }

    //Draw Tree on the given visible side
    int cube_side = 50;
    char showOn[] = "Right";

    drawTree(WCS.X[4], WCS.Y[4], WCS.Z[4], cube_side, showOn);
}

// Method to draw the half sphere using contours
void drawSphere()
{
    #define UpperBD 360
    #define UpperPCBD 800
    #define TotalPts 360
```

```
#define NumOfLevels 20

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, temp3D;
Pts2D Psingle;

int diffColor[UpperBD];
int diffColorPC[UpperPCBD];

float angle_theta;
int radius = 100;
int kx=0, ky=0, kz=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] = 4*level; // Elevate the contour using Z_w each level

        temp3D.x_value = WCS.X[i];
        temp3D.y_value = WCS.Y[i];
        temp3D.z_value = WCS.Z[i];

        //Bonus point question task
        diffColor[i] = getDiffuseColor(temp3D, 0.0, 1.0, 0.0);
    }

    // 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;

        //Bonus point question task
        drawPixel(P.X[i], P.Y[i], diffColor[i]);
```

```
            // Logic to store a selected number of equi-distant points on each contour into PC.
            // In this case, for each contour, 40 points are selected and later joined.
            if(i%(TotalPts/40) == 0)
            {
                PC.X[kx++] = P.X[i];
                PC.Y[ky++] = P.Y[i];

                //Bonus point question task
                diffColorPC[kz++] = diffColor[i];
            }
        }

        radius-=0.25*level; // 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+40)<kx)
        //Bonus point question task
        drawLine(PC.X[i], PC.Y[i], PC.X[i+40], PC.Y[i+40], diffColorPC[i]);
    }
}

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);
```

```
    drawSphere();

    drawCube();

    return 0;
}
```