2D and 3D Graphic Designs on LPC1769

Akansha Jajodia (014489215)
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 95192 E-mail: akansha.jajodia@sjsu.edu

Abstract

*This paper describes the design, implementation and testing for the graphics engine prototype for LPC1769 using 128*160 TFT Display. There are three objectives for this lab. One is to display a 3D cube with a size 80. The next goal is to draw and fill the shadow points from the point light source. Finally, the top surface of the cube is filled using calculated diffuse reflection values and boundaries of top surface is computed using DDA Algorithm. The description given here focuses on the hardware and software components required to interface LPC 1769 with the LCD through serial communication and also the test procedure and results are attached. MCUXpresso IDE is the development platform used here.* ***Keywords*** *- LPC1769, SPI LCD, 3D Graphics, MCUXpresso ID, DDA Algorithm, Diffuse Reflection.*

1. Introduction

This lab is focused on implementation of a 2D and 3D graphic designs on LCD Display by collecting data from a CPU Module. The CPU Module used here is LPC 1769 which is based on ARM Cortex M0 Core.
MCUXpresso IDE is used for implementing the design. The program is written in C language and compiled. The compiled code is then loaded on LPC1769 memory and is executed.

The important objective of this lab is to understand the SPI communication protocol between a LCD and CPU module and use it do display 2D and 3D screen saver. In these screen savers, we have to first implement simple graphics drawing capability by plotting a single line with any user defined color and width.

This paper provides detailed description of the software and the hardware methodology implemented in the lab. Along with this, this paper also includes details on the testing and verification. Diagram and images are also added.

The LPC module is connected to the computers USB port to get the required power supply of 3.3 volts. In LPC1769, pin numbers 5, 7 and 8 are used for communication as we are using SPI port1 for coding.

2. Methodology

The 1.8" TFT display has 128x160 color pixels. MCUEXpresso IDE is software tool used for implementation of 3D graphics cube and diffuse reflection. The drawLine function is used here is the base for generating a 3D cube and shadow lines along with other helper functions such as virtual to physical display conversion, Transformation pipeline for 3D graphics, diffuse reflection computation and DDA algorithm for finding boundary location and so on. The algorithm implementation is explained in the next section.

2.1. Objectives and Technical Challenges

Following are the objectives of this lab:

1. Creating a prototype board including the Microcontroller, LCD and a Power Circuit.
2. Getting familiarized with LPCXpresso IDE.
3. Getting familiarized with datasheets of NXP LPC1769 and Adafruit ST7735R LCD module.
4. Getting familiar with 2D and 3D Vector Graphics and implementation of C code for the same.
5. Implementing, Testing and Debugging of CPU and LCD Modules and accomplishing data transfer among them.

While developing this module we came across a few technical challenged. These technical challenges are listed below.

1. Understanding the Pin Structure of the LCD Module and bringing it up with SPI Communication.
2. Displaying the image on the LCD screen.
3. Compute the diffuse reflection values for the top surface.
4. Finding locations based on DDA and bilinear interpolation
5. Linear decoration of letter A

2.2. Problem Formulation and Design

This section will provide the detailed design. It includes the block diagram and the schematics and pin connection between the components used for this lab assignment. The hardware used for this lab is connected to the wire wrapping board using soldering technique.

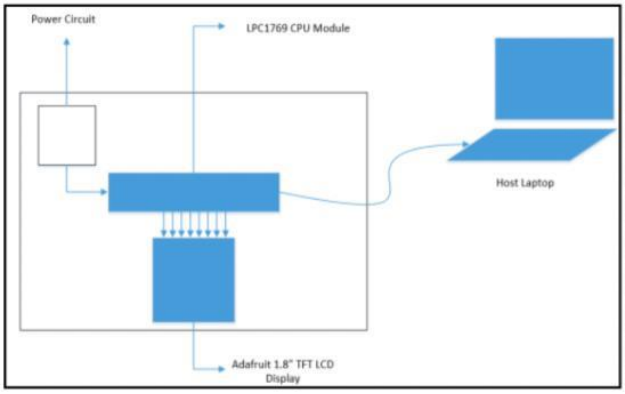


Figure 1. Overview of the Project

The hardware design includes Adafruit 1.8" Color TFT LCD Display and LPCXpresso 1769 CPU module. The critical part of this lab is the communication between the LCD display and the CPU module. The hardware design includes Adafruit 1.8" Color TFT LCD Display and LPCXpresso 1769 CPU module. The critical part of this lab is the communication between the LCD display and the CPU module.

Data is transferred from the host to the LPC module, which in turn transfers data to the LCD module. The SPI communication follows a master-slave model of control.
On the software design side, the requirement is to generate 3D cube and shadow and filling the shadow and 3 visible surfaces of the cube. The requirement also states that the top surfaces of the cube should be filled using diffuse reflection values.

The first objective is to draw a 3D cube and it's shadow point from the light source.

1. First, below steps are followed to convert 3D pointsfrom World to Viewer coordinate system.

- For conversion of 3D points to viewer system, Pipeline Transformation is used.
- An Eye position is defined in the 3D space vector E (x_e, y_e, z_e).
- The perspective of Eye is used to compute points on viewer coordinate system.
- Using the following formula the Viewer coordinates x_v, y_v, z_v coordinates are computed,

$x_v = -(x_w * \sin\theta) + (y) + (y_w * \cos\theta) + (y)$

$y_v = -(x_w * \cos * \cos\theta) + (y) - (y\varphi * \cos\theta) - (y_{w*} \cos * \sin\theta) + (y)\ \varphi * \cos\theta) - (y + (z_w * \sin)\ \varphi * \cos\theta) - (y$

$z_v = -(x_w * \sin * \cos\theta) + (y) - (y\varphi * \cos\theta) - (y_{w*} \sin * \sin\theta) + (y) - (z\varphi * \cos\theta) - (y_w * \cos) + \rho\ \varphi * \cos\theta) - (y$

where, ρ = distance of the camera from the origin (0,0,0), cos theta, sin theta, cos phi, sin phi are intersection angles.

2. Perspective Projection using the pinhole model,converting 3D points to 2D.

- In this model, a focal point D is defined at a certain distance from the eye position towards the object.
- The following formula is used to create virtual image on focal plane, $x'' = x_v * (D/z_v)$ $y'' = y_v * (D/z_v)$

where , D = distance of the focal plane from the camera/Eye position.

3. Ray equation and Shadow Computation

- A specific point is defined as light source Ps in 3D space.
- The shadow is projected in XY plane.
- Compute the equations for rays passing through light source and cube.
- The ray equations are computed using,

$$\vec{Pi'}(x,y) = \vec{P}_s\ (x,y) + \lambda(\vec{P}_s - \vec{P}_i)$$

where , $\vec{P}_s$ = Point of Light Source

$\vec{P}_i$ = Vertex of the cube

$$\lambda = \frac{(n_x * (x_a - x_s)) + (n_y * (y_a - y_s)) + (n_z * (z_a z_s))}{(n_x * (x_s - x_i)) + (n_y * (y_s - y_i)) + (n_z * (z_s - z_i))}\ n$$

(n_x, n_y, n_z) = normal vector = (0, 0, 1)

$a(a_x, a_y, a_z)$ = arbitrary point on XY plane = (0,0,0)

4. Diffusion Reflection is calculated for the 4 intersection points of the top surface.

• The following formula is used to compute the diffuse reflection,

$I_{r,g,b} = K_{dr,dg,db} * (\cos\alpha) * (1/||r||_2)) * (1/||r||_2)||_2)$

Here, $||r||2)||$ = distance of the point from origin = sqrt ((xs - xi)2) +(ys - yi)2 + (zs - zi)2)

5. DDA Algorithm is used to find boundary lines of thetop surface. It uses incremental scan to find the next location in the line based on the previous steps. Once the locations are found, the diffuse reflection value for each location is calculated using bi-linear interpolation. And the points are drawn using drawPixel function.

6. The interior surface is filled by scanning x,y fromstarting to ending point with color value calculated for each point using diffuse reflection.

7. The right,left surfaces of the cube and the shadowwere filled by scanning from starting point to end point.

3. Implementation

The overall layout of the board along with the computer is as follows:

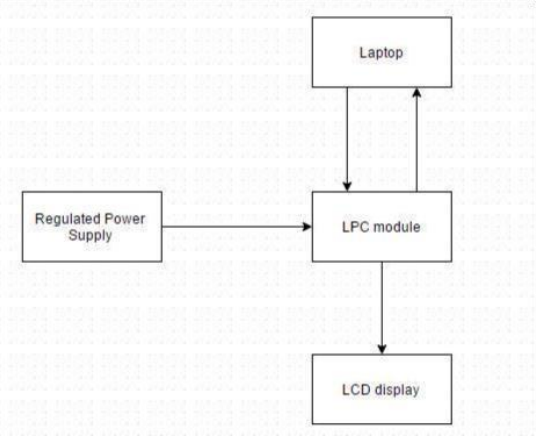


Figure 2. Block Diagram

The implementation of this lab in done in two parts hardware and software. The details of each part is described in this section.

3.1. Hardware Design

The hardware components required to implement this lab is as follows:

No.	Item and Description	Notes
1	Wire wrapping board	10 * 10 inches
2	NXP's LPC1769 module	
3	SPI based color display	Driver ST7735R
4	Colored wires	For connections
5	Laptop	With MCU xpresso IDE
6	Power regulator IC 7805	Output 3.3 V
7	LED	

Table 1. Bill of materials

While making a prototype board the most important part is a power regulation circuit to give 5 V output to power up LPC1769. After that interface of LCD and the CPU Module. The CPU Module (Microcontroller) is operated as the Master. Its MISO, MOSI and SCK pins are used to interface with the SPI LCD Display which works as slave. The following pictures shows the hardware developed for of the project.

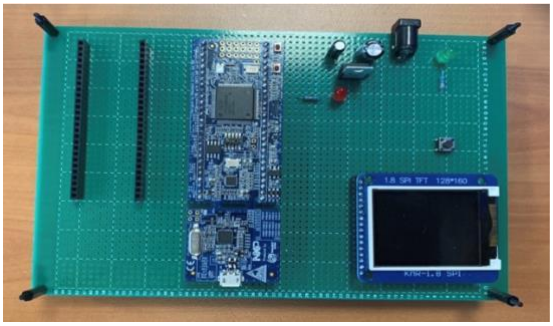


Figure 3. Hardware model of the system

The ST7735R is a single-chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source lines and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI), 8-bit/9-bit/16- bit/18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits. It can perform display data RAM read/write operation with no external operation clock to minimize power consumption. In addition, because of the integrated power supply circuits necessary to drive liquid crystal, it is possible to make a display system with fewer components.

Figure 4. LCD Display Module

3.1.1 Circuit Design

The first thing when making a prototype board is a power regulator circuit. The CPU Module needs 5V to function properly so a voltage regulator is used to reduce any input voltage to 5V. Using a LM 7805 Voltage Regulator IC, capacitor, switch and LEDs a power circuit is made. The circuit diagram for the same is shown.

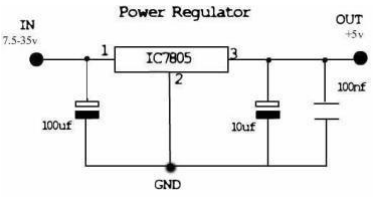


Figure 5. Circuit Diagram for Power Sensor

The following image shows the pin description of LCD Module

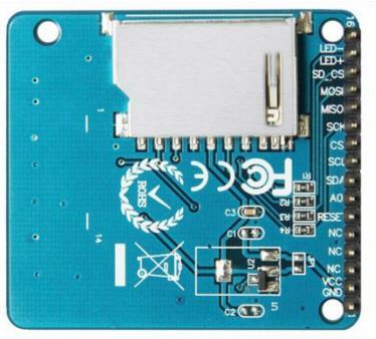


Figure 6. LCD Pin Description

The following table gives the detailed description of interface between the LPC and the LCD.

Pins Connection for LPC and LCD Interface	
LPC Pins	***LCD Pins***
Vcc (J6-28)	LED+ (Pin 2)
GND	LED- (Pin 1)
MISO0 (P0.17) (J6-12)	MISO (Pin 5)
SCK0 (P0.15) (J6-13)	SCK (Pin 6)
MOSI0 (P0.18) (J6-11)	SDA (Pin 9)
SSEL0 (P0.16) (J6-14)	TFT_CS (Pin 7)
GPIO (P0.21) (J2-23)	AO (Pin 10)
GPIO (P0.22) (J2-24)	RESET (Pin 11)
VCC 3.3V	VCC (Pin 15)
GND	GND (Pin 16)

Table 2. Pin connection between LCD and LPC 1769

In the figure 6, the LCD has 10 pins which connect to the CPU Module. The module is connected to the Host Computer via USB cable through which it is powered up. Pin J6-11, J6-13 and J6-14 of the LPC 1769 is used for serial communication with SPI port number 0 for our algorithm. When the CS Pin of the LCD Module is logic 1 the device is deselected, and data output pins are at high impedance. The output of the LCD Display that is the second pin is connected to the MISO (Master In Slave Out) of the CPU module. The SPI instruction uses MOSI and serially writes instructions on the rising edge of the clock. The LED+ pin is connected to the Vcc and the LED- is connected to the GND pin, this lights up the backlight and the display becomes more clear. The Chip Select signal for the TFT Display should be kept low all the time for the duration of the RESET operation to avoid resetting the internal logic state of the device. Instructions vary in length. For some we send only the opcode, for some we send dummy bytes also so that there is some time for the CPU module to recognize the instruction

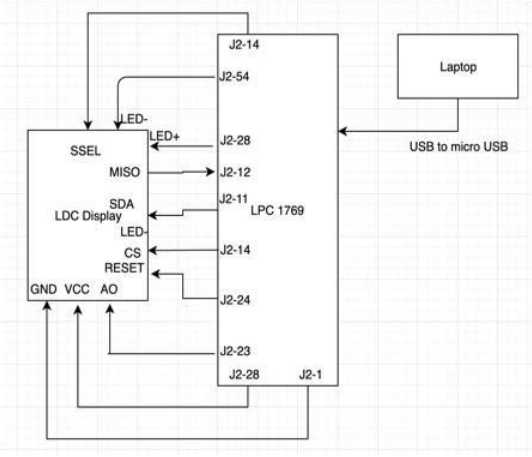


Figure 7. SPI Connection Between LCD and LPC

3.2. Software Design

This section describes the implementation process of generating squares, tree pattern and cube through flow chart, algorithm and pseudo code.

3.2.1 Algorithm

The following algorithm explains the basic steps followed to setup and run the software components.

Step 1: Start

Step 2: Initialize SPI

- Set PCONP's 21 st bit to enable SSP0
- SSP_CLK selected as PCLK/4, by writing PCLKSEL1 as 0
- Set J2 5-8 pins functionality as SSP 1
-
- Set SSEL0 as GPIO out
- SSP1 data width is set to 8 bit
- SCR register value to 15
- Pre scale CLK value set to 2

Step 3: Initialize LCD

- Set SSEL0 as 0, to make LCD slave
- D/C connected to J2-23
- RESET connected to J2-24
- Set both pins as output
- P0.22 pin value is set as logic 1
- Provide delay of 500 ms
- P0.22 pin value is set as logic 0
- P0.22 pin value is set as logic 1

Step 4: Draw shadow

- Draw shadow by connecting the intersection points from the light source.

Step 5: Fill shadow

- Fill shadow points by scanning from the starting to end point

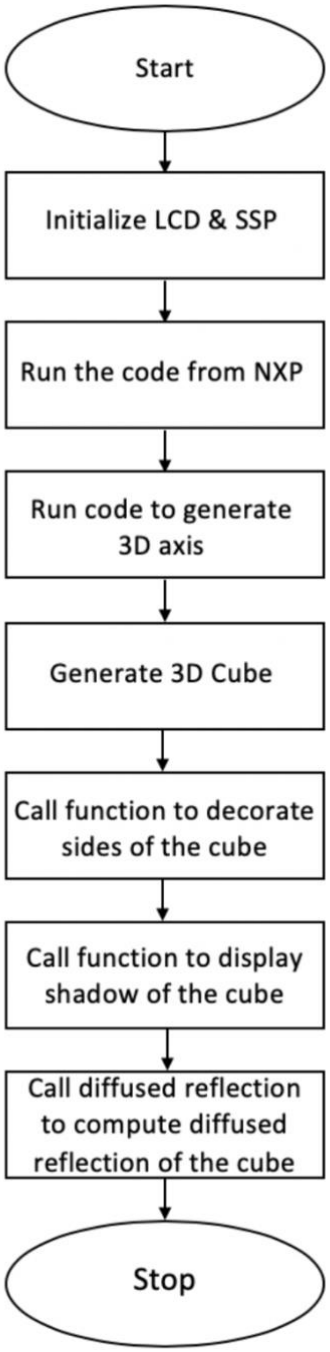
Step 6: Draw 3D Cube and fill the left and right surfaces.

Step 7: Draw boundary lines of top surface using the points calculated from DDA Algorithm and diffuse reflection values.

Step 8: Fill the interior of the top surface by scanning from starting to end point.

Step 9: Stop.

3.2.2 Flowchart



3.2.3. Pseudo code:

```
void fillshadow(struct Point_3D startPt,struct Point_3D
endPt)
{

        struct Point_3D temp;
        struct Point currPt;
        for(float i=startPt.y; i<endPt.y; i++)
        {
        for(float j=startPt.x; j<endPt.x; j++)
        {
        temp.x = j;
        temp.y = i;
        temp.z = 0;

        currPt = Transformation_pipeline(temp);
        drawPixel(currPt.x, currPt.y,DARKBLUE);
        }
        }
}

//DDA Function for line generation
void DDA_line(struct Point p1, struct Point p2, float
Idiff1,float Idiff2)
{
        uint32_t color;
   // calculate dx & dy
   int dx = p2.x - p1.x;
   int dy = p2.y - p1.y;

   // calculate steps required for generating pixels
   int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

   // calculate increment in x & y for each steps
   float Xinc = dx / (float) steps;
   float Yinc = dy / (float) steps;

   // Put pixel for each step
   float X = p1.x;
   float Y = p1.y;
   for (int i = 0; i < steps; i++)
   {
        color = findcolor(X,Y,Idiff1,Idiff2,p1,p2);
      drawPixel(X,Y,color);
      X += Xinc;         // increment in x at each step
      Y += Yinc;         // increment in y at each step

   }
}
// To draw shadow for cube
void DrawShadow(double cube_size, struct Point_3D
ps)
{
        float l1,l2,l3,l4;
        struct Point s1,s2,s3,s4,ps_p;
        struct Point_3D xp1,xp2,xp3,xp4;
        int pt = 0;
        struct Point_3D p1 =
{pt,pt,(cube_size+pt+elev)};//p1
        struct Point_3D p2 =
{(cube_size+pt),pt,(cube_size+pt+elev)};//p2
        struct Point_3D p3 =
{(cube_size+pt),(cube_size+pt),(cube_size+pt+elev)};//p4
        struct Point_3D p4 =
{pt,(cube_size+pt),(cube_size+pt+elev)};//p3
        ps_p = Transformation_pipeline(ps);
        l1 = lambda_calc(p1.z,ps.z);
        l2 = lambda_calc(p2.z,ps.z);
        l3 = lambda_calc(p3.z,ps.z);
        l4 = lambda_calc(p4.z,ps.z);

        /*printf("\nLambda 1 %0.2f:",l1);
        printf("\nLambda 2 %0.2f:",l2);
        printf("\nLambda 3 %0.2f:",l3);
        printf("\nLambda 4 %0.2f:",l4);*/

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

//      printf("\nshadow point 1 :
%0.2f,%0.2f,%0.2f",xp1.x,xp1.y,xp1.z);

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

        //printf("\nshadow point 2 :
%0.2f,%0.2f,%0.2f",xp2.x,xp2.y,xp2.z);

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


        //printf("\nshadow point 3 :
%0.2f,%0.2f,%0.2f",xp3.x,xp3.y,xp3.z);
```

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

        //printf("\nshadow point 4 :
%0.2f,%0.2f,%0.2f",xp4.x,xp4.y,xp4.z);

        s1 = Transformation_pipeline (xp1);
        s2 = Transformation_pipeline (xp2);
        s3 = Transformation_pipeline (xp3);
        s4 = Transformation_pipeline (xp4);

        drawLine(s1.x,s1.y,s2.x,s2.y,GOLD);
        drawLine(s2.x,s2.y,s3.x,s3.y,GOLD);
        drawLine(s3.x,s3.y,s4.x,s4.y,GOLD);
        drawLine(s4.x,s4.y,s1.x,s1.y,GOLD);

        /*drawLine(s1.x,s1.y,ps_p.x,ps_p.y,YELLOW);
        drawLine(s2.x,s2.y,ps_p.x,ps_p.y,YELLOW);
        drawLine(s3.x,s3.y,ps_p.x,ps_p.y,YELLOW);
        drawLine(s4.x,s4.y,ps_p.x,ps_p.y,YELLOW);*/

        fillshadow(xp1,xp3);
}
void draw_A(int start_x , int start_y , int start_z, int size
){
        struct Point_3D temp;
   struct Point p1;
   int i,j;
   size=size+start_pnt;
   int map[80][80];

   for(i = 0; i < 80;i++)
   {
      for(j = 0; j < 80;j++)
      {
         if(i>=10 && i<=12 && j>=7 && j<=60){
                  map[i][j]=1;
         }else if(i>=10 && i<=42 && j>=29 && j<=32){
                  map[i][j]=1;
         }else if(i>=10 && i<=42 && j>=7 && j<=10){
                  map[i][j]=1;
         }else if(i>=38 && i<=40 && j>=7 && j<=60){
                  map[i][j]=1;
         //}else if(i>=17 && i<=22 && j>=10 &&
j<=48){

         }else{
                  map[i][j]=0;
         }
      }
   }
for(i=0;i<80;i++)
   {
      for(j=0;j<80;j++)
      {
         if(map[i][j]==1)
         {
            temp.x = i;
            temp.y = j;
            temp.z = size;
            p1 = Transformation_pipeline(temp);
            drawPixel(p1.x,p1.y,SILVER);
         }
      }
   }
}
```

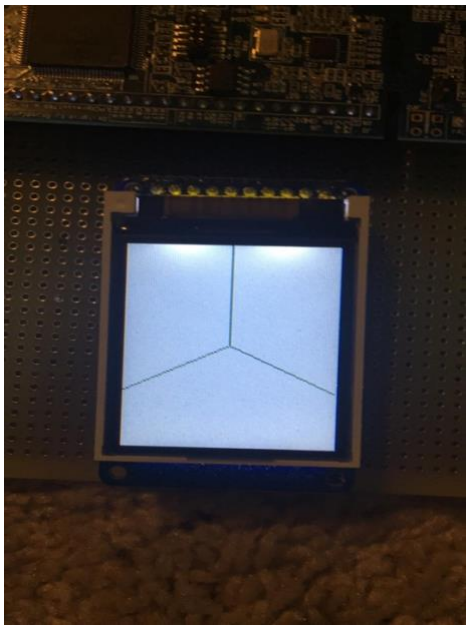
4. Testing and Verification

This section provides the testing and verification for the GPIO and the communication among the LCD display and LPCXpresso. The following are the steps to follow for debugging purposes in case of errors:

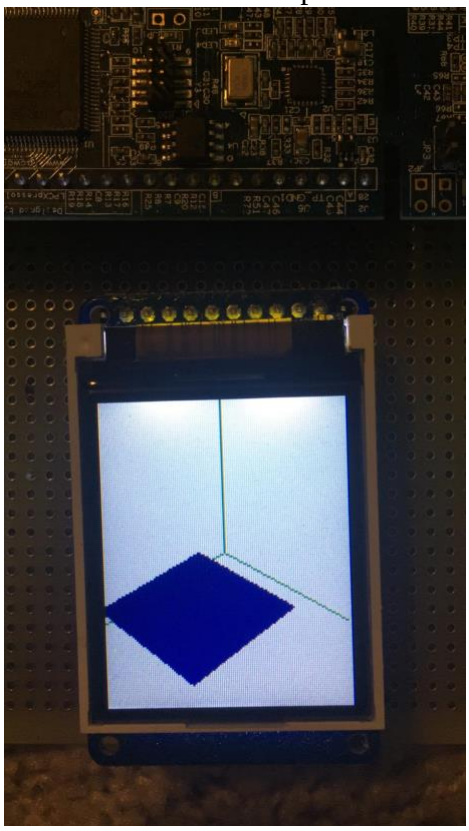
i. The connections are checked for any short or loose circuit.
ii. The schematics and the code is checked.
iii. All the 8 pins of SPI flash have to be securely wired and soldered; no pin should be left without connection.
iv. Verify that CPU is connected and LPC link is established.
v. The ground connections and VCC are verified and checked whether 5V was received at each point.
vi. Run the code and verify the result on screen.
vii. The LCD glows when the CPU module receives the power.
viii. The project if having no errors would build and debug successfully with successful link connection to CPU module.
ix. The data is transferred from MCUXpresso IDE to the target board
x. This will create 3D cube with shadow and diffuse reflection on top surface of the cube on the LCD screen. Thus, communication between the LPC and LCD via SPI is tested and verified successfully.
xi. After filling the screen, a delay is introduced.
xii. And finally, the initial letter is displayed.
xiii. The top surface of the cube has brighter color values for the nearest point from the light source. Thus the diffuse reflection has been verified.

4.1 Test Results

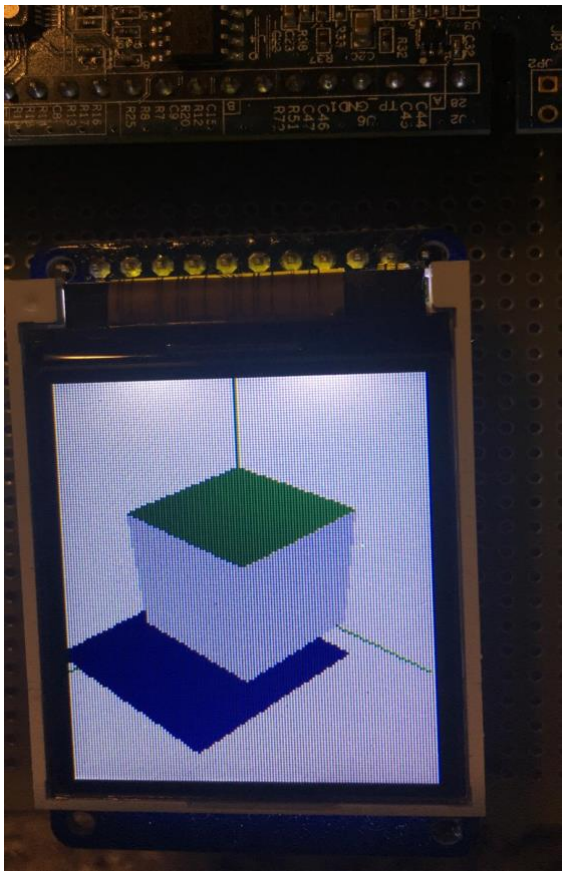
Result1 : Cooridnates



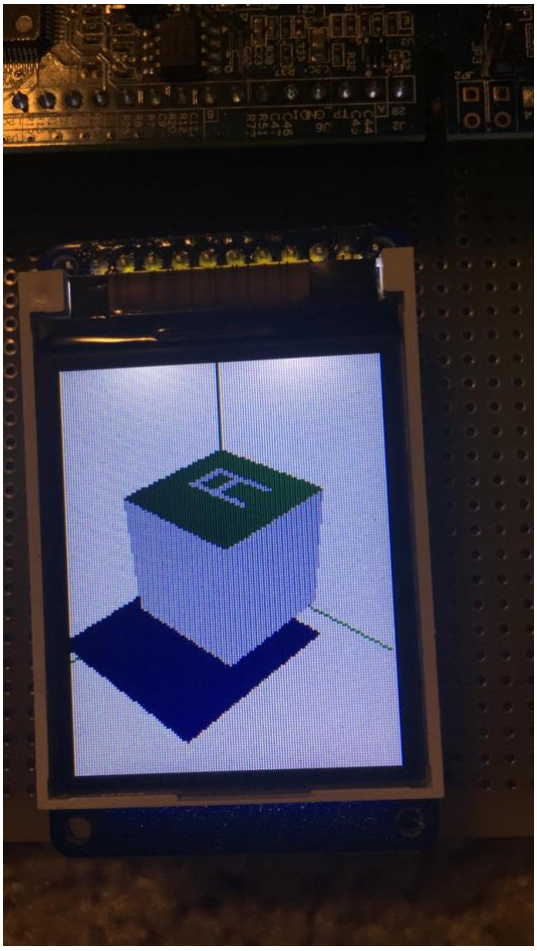
Result2 : Shadow Computation



Result3 : Cube With Shadow using DDA



Result4 : Initial of Name



5. Conclusion

The design and implementation of interface between CPU module and LCD display was implemented successfully and Graphics engine logic and GUI demonstrated. The project gave a good understanding of how 3D, 2D vector graphics engine techniques. The work provides an opportunity to study LCD display, LPCXpresso CPU module, LPCXpresso IDE and soldering techniques. The conversion of objects from world coordinate to viewer coordinate was implemented successfully with proper scaling. Shadows were implemented and diffused reflection was achieved, initials of the first name was displayed.

6. Acknowledgment

I would like to express my gratitude to Dr. Harry Li for teaching the concepts of 2D and 3D vector graphics and providing guidance for the project and support to carry out the Lab activities and giving us this opportunity to develop the project and explore new areas of Microprocessor.

7. References

[1] Dr.H.Li, Guidelines for CMPE240 project and report, Computer Engineering Department, San Jose State University, San Jose 95112

[2]Dr.H.Li, CMPE240 Lecture Notes, Computer Engineering Department, San Jose State University, San Jose 95112

[3] NXP LPCXpresso1769 Discussion forums at www.lpcware.com/forum [4] NXP, “UM10360 user manu.pdf”, UM10360 LPC176x/5x User manual.

[5] LPCXpresso 1769 Datasheet http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf

8. Appendix

Source Code

```
/*
===============================================
=======================================
 Name        : DrawLine.c
 Author      : Akansha Jajodia
 Version     : 1.0
 Copyright   : $(copyright)
 Description : Display 3Dcube with diffuse reflection and initial
letter on top surface
===============================================
=======================================
*/

#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>
#include <time.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 pi 3.14

#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
#define PINK 0xFFC0CB
#define YELLOW 0xFFFF00
#define SILVER 0xC0C0C0
#define LIME 0x00FF00
#define ORANGE 0xFFA500
#define MAROON 0x800000
#define FOREST 0x228B22
#define DARKGREEN 0x006400
#define LIGHTGREEN 0X90EE90
#define SEAGREEN 0X2E8B57
#define ORANGERED 0XFF4500
#define GOLD 0XFFD700
#define GRAY 0X808080
#define SHADOW 0X8A795D

#define pi 3.14
#define alpha pi/6

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("LCD Demo Begins!!!\n");
 // Set pins P0.16, P0.21, P0.22 as 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);

}



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

{

 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;

  }

 }

}

//Define 2d Point struct
//Define 2d Point struct
struct Point
{
    int x;
    int y;

};

//Physical to virtual
int v2p_x(int x)
{
return x+_width/2;
}
int v2p_y(int y)
{
return -y+_height/2;
}

//find branch point

//Define 3d Point structure
struct Point_3D
{
    float x;
    float y;
    float z;
};

//define eye location
    float_t xe = 350;
    float_t ye = 350;
    float_t ze = 350;

     int elev = 10;

int round(float number)
{
    return (number >= 0) ? (int)(number + 0.5) :
(int)(number - 0.5);
}

// To convert world to viewer coordinates
struct Point Transformation_pipeline (struct
Point_3D world)
{
    struct Point perspective;
    struct Point_3D viewer;
    //define distance
    float_t Rho =
sqrt(pow(xe,2)+pow(ye,2)+pow(ze,2));
    float_t D_focal = 260;

    //sin and cos pheta
    float spheta = ye /
sqrt(pow(xe,2)+pow(ye,2));
    float cpheta = xe /
sqrt(pow(xe,2)+pow(ye,2));
    //second angle
    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 = round(D_focal*viewer.x/viewer.z);
   perspective.y = round(D_focal*viewer.y/viewer.z);

   perspective.x = v2p_x(perspective.x);
   perspective.y = v2p_y(perspective.y);

   return perspective;
}

float diff_reflection(struct Point_3D Pi)
{
   //--------compute distance------------------*
      struct Point_3D Ps = {-50, 50, 360};

      float red;
      float scaling = 30000, shift = 0.2;
      float distanceSqr = (pow((Ps.x - Pi.x), 2)+ pow((Ps.y -
Pi.y), 2) + pow((Ps.z - Pi.z), 2));
      float cosine  = ((Ps.z -Pi.z)/sqrt(distanceSqr));
      float temp  =  cosine /   distanceSqr;
      temp =  (scaling * temp);
      temp = (temp < shift) ? shift : temp;
      red = (255 * 0.8 * temp);
      return red;

}

void fillshadow(struct Point_3D startPt,struct Point_3D endPt)
{

      struct Point_3D temp;
      struct Point currPt;
      for(float i=startPt.y; i<endPt.y; i++)
      {
      for(float j=startPt.x; j<endPt.x; j++)
      {
      temp.x = j;
      temp.y = i;
      temp.z = 0;

      currPt = Transformation_pipeline(temp);
      drawPixel(currPt.x, currPt.y,DARKBLUE); //SHADOW);
      }
      }
}

void fillside(struct Point_3D startPt,struct Point_3D endPt,int
size, int side)
{

   struct Point_3D temp;
   struct Point currPt;
   if(side==1)
   {
      for(float i=startPt.z; i<endPt.z; i++)
      {
         for(float j=startPt.y; j<endPt.y; j++)
         {
            temp.x = size;
            temp.y = j;
            temp.z = i;
            currPt =
Transformation_pipeline(temp);
            drawPixel(currPt.x,
currPt.y,SILVER);
         }
      }
   }
   else if(side == 2)
   {
      for(float i=startPt.z; i<endPt.z; i++)
      {
         for(float j=startPt.x; j<endPt.x; j++)
         {
            temp.x = j;
            temp.y = size;
            temp.z = i;
            currPt =
Transformation_pipeline(temp);
            drawPixel(currPt.x,
currPt.y,SILVER);
         }
      }
   }

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

```
                currPt = Transformation_pipeline(temp);
                drawPixel(currPt.x, currPt.y,color);
              }
            }
          }

}

uint32_t findcolor(float x, float y,float Idiff1,float Idiff2,struct
Point p1,struct Point p2)
{
   float newx,newy,ncolor;
   uint32_t color;
   float red,blue;

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

   return color;

}

//DDA Function for line generation
void DDA_line(struct Point p1, struct Point p2, float Idiff1,float
Idiff2)
{
   uint32_t color;
   // calculate dx & dy
   int dx = p2.x - p1.x;
   int dy = p2.y - p1.y;

   // calculate steps required for generating pixels
   int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

   // calculate increment in x & y for each steps
   float Xinc = dx / (float) steps;
   float Yinc = dy / (float) steps;

   // Put pixel for each step
   float X = p1.x;
   float Y = p1.y;
   for (int i = 0; i < steps; i++)
   {
      color = findcolor(X,Y,Idiff1,Idiff2,p1,p2);
      drawPixel(X,Y,color);
      X += Xinc;          // increment in x at each step
      Y += Yinc;          // increment in y at each step
   }

}

// To draw a 3D cube
void draw3dcube(int cube_size)
{
   struct Point c1,c2,c3,c4,c5,c6,c7;
   int pt = 0;
   struct Point_3D cu_dim =
{pt,pt,(cube_size+pt+elev)};//(0,0,100)
   float Idiff1,Idiff2,Idiff3,Idiff4;
   struct Point_3D t1,t2,t3,t4,t5,t6,t7;
   t1 = cu_dim;
   Idiff1 = diff_reflection(t1);
   c1 = Transformation_pipeline(cu_dim);

   cu_dim.x = (cube_size+pt);//(100,0,100)
   t2 = cu_dim;
   Idiff2 = diff_reflection(t2);
   c2 = Transformation_pipeline(cu_dim);

   cu_dim.y = (cube_size+pt);//(100,100,100)
   t3 = cu_dim;
   Idiff3 = diff_reflection(t3);
   c3 = Transformation_pipeline(cu_dim);

   cu_dim.x = pt;//(0,100,100)
   t4 = cu_dim;
   Idiff4 = diff_reflection(t4);
   c4 = Transformation_pipeline(cu_dim);

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

   cu_dim.y = (cube_size+pt);//(100,100,0)
   t6 = cu_dim;
   c6 = Transformation_pipeline (cu_dim);

   cu_dim.x = pt;//(0,100,0)
   t7 = cu_dim;
   c7 = Transformation_pipeline (cu_dim);

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

   lcddelay(300);
```

```
    drawLine(c2.x, c2.y, c5.x, c5.y,SILVER);
    drawLine(c5.x, c5.y, c6.x, c6.y,SILVER);
    drawLine(c6.x, c6.y, c3.x, c3.y,SILVER);
    drawLine(c6.x, c6.y, c7.x, c7.y,SILVER);
    drawLine(c7.x, c7.y, c4.x, c4.y,SILVER);

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

}

//Calculate lambda values for cube
float lambda_calc(float zi,float zs)
{
    return -zi/(zs-zi);
}

void draw3DAxes()
{
    int width = ST7735_TFTWIDTH/2;
    int height = ST7735_TFTHEIGHT/2;

    struct Point_3D world[4],viewer[4];
    //pointViewer viewer;
    struct Point perspective[4];

    int Xe = 90.00;
    int Ye = 90.00;
    int Ze = 90.00;

    double Rho = sqrt(pow(Xe,2) + pow(Ye,2) + pow(Ze,2));
    int dFocal = 40.0;

    world[0].x = 0.0;
    world[0].y = 0.0;
    world[0].z = 0.0;

    world[1].x = 200.0;
    world[1].y = 0.0;
    world[1].z = 0.0;

    world[2].x = 0.0;
    world[2].y = 200.0;
    world[2].z = 0.0;

    world[3].x = 0.0;
    world[3].y = 0.0;
    world[3].z = 200.0;

    double cTheta = acos(Xe/sqrt(pow(Xe,2)+pow(Ye,2)));
    double cPhi = acos(Ze/Rho);

    for(int i =0;i <= 3; i++)
    {
        viewer[i].x = (world[i].y * cos(cTheta))-
(world[i].x * sin(cTheta));
        viewer[i].y = (world[i].z*sin(cPhi))-
(world[i].x*cos(cTheta)*cos(cPhi))-
(world[i].y*cos(cPhi)*sin(cTheta));
        viewer[i].z = Rho-
(world[i].y*sin(cPhi)*cos(cTheta))-
(world[i].x*sin(cPhi)*cos(cTheta))-
(world[i].z*cos(cPhi));
    }

    for(int i=0; i<=3;i++)
    {
        perspective[i].x = dFocal * viewer[i].x /
viewer[i].z;
        perspective[i].y = dFocal * viewer[i].y /
viewer[i].z;
        perspective[i].x = width +
perspective[i].x;
        perspective[i].y = height -
perspective[i].y;
    }

drawLine(perspective[0].x,perspective[0].y,per
spective[1].x,perspective[1].y,DARKGREEN);

drawLine(perspective[0].x,perspective[0].y,per
spective[2].x,perspective[2].y,DARKGREEN);

drawLine(perspective[0].x,perspective[0].y,per
spective[3].x,perspective[3].y,DARKGREEN);
}

// To draw shadow for cube
void DrawShadow(double cube_size, struct
Point_3D ps)
{
    float l1,l2,l3,l4;
    struct Point s1,s2,s3,s4,ps_p;
    struct Point_3D xp1,xp2,xp3,xp4;
    int pt = 0;
    struct Point_3D p1 =
{pt,pt,(cube_size+pt+elev)};//p1
    struct Point_3D p2 =
{(cube_size+pt),pt,(cube_size+pt+elev)};//p2
    struct Point_3D p3 =
{(cube_size+pt),(cube_size+pt),(cube_size+pt+
elev)};//p4
    struct Point_3D p4 =
{pt,(cube_size+pt),(cube_size+pt+elev)};//p3
    ps_p = Transformation_pipeline(ps);
```

```
    l1 = lambda_calc(p1.z,ps.z);
    l2 = lambda_calc(p2.z,ps.z);
    l3 = lambda_calc(p3.z,ps.z);
    l4 = lambda_calc(p4.z,ps.z);

    /*printf("\nLambda 1 %0.2f:",l1);
    printf("\nLambda 2 %0.2f:",l2);
    printf("\nLambda 3 %0.2f:",l3);
    printf("\nLambda 4 %0.2f:",l4);*/

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

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

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

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


    s1 = Transformation_pipeline (xp1);
    s2 = Transformation_pipeline (xp2);
    s3 = Transformation_pipeline (xp3);
    s4 = Transformation_pipeline (xp4);

    drawLine(s1.x,s1.y,s2.x,s2.y,GRAY);
    drawLine(s2.x,s2.y,s3.x,s3.y,GRAY);
    drawLine(s3.x,s3.y,s4.x,s4.y,GRAY);
    drawLine(s4.x,s4.y,s1.x,s1.y,GRAY);

    fillshadow(xp1,xp3);
}

void draw_A(int start_pnt, int size)
{
    struct Point_3D temp;
    struct Point p1;
    int i,j;
    size=size+start_pnt;
    int map[80][80];

    for(i = 0; i < 80;i++)
    {
        for(j = 0; j < 80;j++)
        {
            if(i>=10 && i<=12 && j>=7 && j<=60){
                map[i][j]=1;
            }else if(i>=10 && i<=42 && j>=29
&& j<=32){
                map[i][j]=1;
            }else if(i>=10 && i<=42 && j>=7 &&
j<=10){
                map[i][j]=1;
            }else if(i>=38 && i<=40 && j>=7 &&
j<=60){
                map[i][j]=1;
            //}else if(i>=17 && i<=22 && j>=10
&& j<=48){

            }else{
                map[i][j]=0;
            }
        }
    }

    for(i=0;i<80;i++)
    {
        for(j=0;j<80;j++)
        {
            if(map[i][j]==1)
            {
                temp.x = i;
                temp.y = j;
                temp.z = size;
                p1 = Transformation_pipeline(temp);
                drawPixel(p1.x,p1.y,SILVER);
            }
        }
    }
}



//Main function
int main (void)
{
    uint32_t pnum = PORT_NUM;
    int start_pnt = 0;
    int size = 100;
    double x[8] =
{start_pnt,(start_pnt+size),(start_pnt+size),start
_pnt,start_pnt,(start_pnt+size),(start_pnt+size),
start_pnt};
    double y[8] = {start_pnt, start_pnt,
start_pnt+size, start_pnt+size, start_pnt,
start_pnt, (start_pnt+size), (start_pnt+size) };
```

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

    pnum = 1 ;

    if ( pnum == 1 )
        SSP1Init();

    else
        puts("Port number is not correct");

    lcd_init();
    fillrect(0, 0, ST7735_TFTWIDTH,
ST7735_TFTHEIGHT, WHITE);

    struct Point_3D pl_source = {-50, 50, 360};

    draw3DAxes();
    DrawShadow(size,pl_source);
    draw3dcube(size);
    draw_A(start_pnt, size);

    return 0;

}
```