2D Graphics Implementation using LPC1769

Divya Khandelwal, MS.
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: divya.khandelwal@sjsu.edu

Abstract

In this project, a graphic display prototype is developed using LPC 1769 interfaced with a 1.8" ST7735R TFT LCD. The aim is to display two screen savers namely rotating square patterns and randomized tree generation. The key in this project is to correctly understand and execute the concept of 2D pattern generation.

1. Introduction

The LPC1769 is a Cortex-M3 microcontroller for embedded applications featuring a high level of integration and low power consumption at frequencies of 120MHz [1]. Along with numerous other features, this module has 70 General Purpose I/O (GPIO) pins with configurable pull up/down resistors and 3 SSP/SPI pins. For this project, these two functionalities of LPC1769 are the key.

For this project, a prototype circuit is created using LPC1769 module and 1.8" ST7735R TFT LCD. The ST7735R is a single-chip controller/driver for 262K-color, graphic type TFT-LCD [2]. It accepts Serial Peripheral Interface (SPI) when connected to an external microcontroller. The display data is stored in the on-chip display data RAM of 132 x 162 x 18 bits.

MCUXpresso from NXP has been used as the IDE to work with this module using C programming language.

2. Methodology

In this section, system layout and its hardware and software configurations are discussed. In addition, the objectives for this project have been listed with the challenges faced during implementation.

2.1. Objectives and Technical Challenges

The objectives of this project are as follows:

1. To be able to create a prototype circuit with LPC1769 module, power-supply and LCD.
2. To gain hands on experience with MCUXpresso environment, its debugging tools and C programming language.
3. To understand and implement SSP and SPI interface.

The technical challenges faced are as follows:

1. Soldering the components on the wire-wrapping board with proper pin connections.
2. Understanding the 2D vector graphics mathematics and implementing it in the C code.
3. Displaying the graphic patterns over the physical display using coordinate system mapping.

2.2. Problem Formulation and Design

In this section, we discuss the system design for the project. This project focuses on generating two screensavers using 2D vector graphics. The first pattern is of rotating squares and the second is of generating randomized trees. Here, Fig.1 shows the system block diagram.

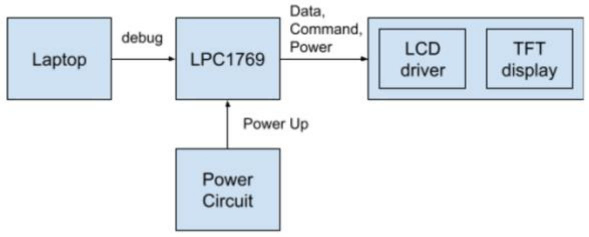

Fig.1 System Block Diagram

3. Implementation

In this section the hardware and software design of the project is described involving LPC1769 interfaced with TFT display.

3.1. Hardware Design

This section describes the list of components, system pin connectivity information between subsystems, schematic of the system and photos of the implemented project board.

3.1.1 List of Components

Table I. lists the hardware components required to build the prototype.

S.No	Item Name
1.	Wire Wrapping Board
2.	LPC1769 CPU module
	Power/Debug Cable
3.	160 x 128 TFT display (ST7735)
	Pin Headers
4.	9V wall mount Power Adaptor
	J1 right angle connector
	Capacitors (100uF/50V, 10uF/50V)
	LM7805 Voltage regulator
	LED (Red)
	SPDT Switch
	Resistor (330 ohms)
5.	Stand offs

Table I. List of Hardware Components

3.1.2 Serial Peripheral Interface

Serial Peripheral Interface is a full duplex serial interface. It can handle multiple masters and slaves being connected to a given bus. [3] During a data transfer the master always sends 8 to 16 bits of data to the slave, and the slave always sends a byte of data to the master. Fig. 2 shows the communication of master with its slaves over SPI [4]. Master uses slave select signal (SS) to select which slave it wants to communicate to.

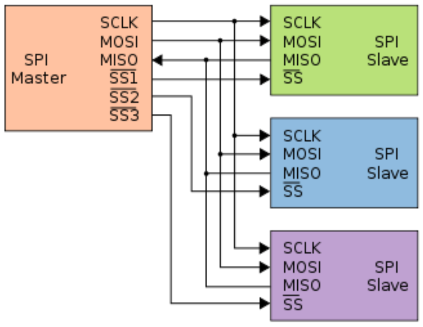

Fig. 2 Serial Peripheral Interface

3.1.3 System Schematic

The system works in Master Slave configuration. The LPC1769 acts as master which communicates with the slave i.e LCD display via a Serial Peripheral Interface (SPI). Fig.3 shows the pin connectivity of the two modules through a schematic diagram.

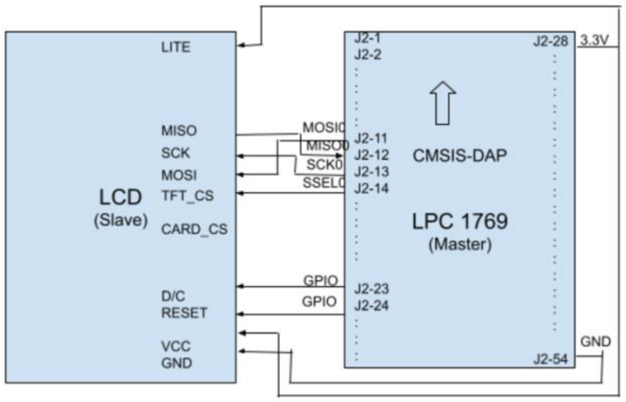

Fig.3 System Schematic

LCD Pins	LPC1769 ports	LPC1769 pins
LITE	VCC (3.3V)	J2-28
MISO	P0.17 (MISO0)	J2-12
SCK	P0.15 (SCK0)	J2-13
MOSI	P0.18 (MOSI0)	J2-11
TFT_CS	P0.16 (SSEL0)	J2-14
CARD_CS	-	-
D/C	P0.21(GPIO)	J2-23
RESET	P0.22 (GPIO)	J2-24
VCC	VCC (3.3V)	J2-28
GND	GND	J2-1/J2-54

Table II. Pin Connectivity Table

3.2. Software Design

This section focuses on the development environment (IDE) used, algorithm design and Pseudo code.

3.2.1 Integrated Development Environment (IDE)

In this project, MCUExpresso by NXP has been used as the IDE. MCUXpresso is a GNU and Eclipse-based IDE that provides an easy-to-use development environment for general purpose, crossover and Bluetooth Arm Cortex-M-based MCUs from NXP, here LPC1769.
For this project, C is used as the programming language and the code is operated in debug mode on the CPU module.

3.2.2 Algorithm Design

This project follows a basic system algorithm based on SPI interface between LPC1769 and the LCD display. The MCUXpresso also plays an important role since it allows the user to render an input to modify the graphic display, through the debug mode. Fig.4 shows the flow chart of the working system.

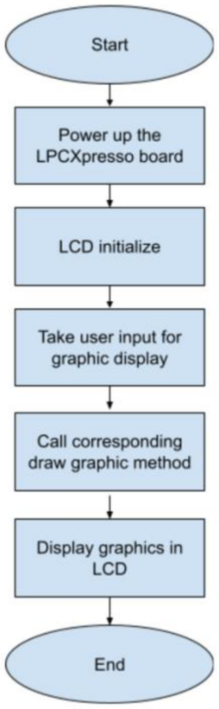
Start
Power up the LPCXpresso board
LCD initialize
Take user input for graphic display
Call corresponding draw graphic method
Display graphics in LCD
End

Fig.4 System Flow chart

In this project, it is required to read and write to various registers of the LPC1769 for interfacing with LCD display and successful data/command exchange. Table III shows the list of those 32-bit registers.

Used to make LPC1769 - GPIO pins	
LPC_GPOI -> FIODIR	Direction of the I/O (Input pin =0 / Output pin = 1)
LPC_GPIO -> FIOPIN	It reads value at the input pin
Used to select the slave – sends output to TFT_CS	
LPC_GPIO ->FIOSET	It sets the output pin

LPC_GPIO -> FIOCLR	It clears the output
Used to make P0.15-P0.18 of LPC1769 as SSP0	
LPC_PINCON->PINSEL0	Pin[31:30]=10; select SCK0
LPC_PINCON->PINSEL1	Pin[1:0]= 10; select SSEL0 Pin[3:2]= 10; select MISO0 Pin[5:4]= 10; select MOSI0
Used to control basic operations of SSP controller (here used to fetch SPI frame format)	
LPC_SSP0->CR0	Pin[3:0]=0111; select 8-bit transfer Pin[5:4]=00; SPI frame format Pin[6]=0/1; Clock out polarity Pin[7]=0/1; Clock out phase Pin[15:8]= Serial clock Rate Pin[31:15]= 0x00 ; Reserved

Table III. Registers modified for LPC1769 and LCD interface

3.2.3 Implementation – Graphic Design

The following are the two graphic patterns that we need to display as screensavers.

1. Rotating square pattern:
   For this design, 2D rotating patterns are created inside their corresponding parent square. There is an entire data set of parent squares. Location and reduction of parent squares are randomized. Each set of rotating patterns has a different color and new pattern is displayed without erasing the previous ones.
   The vector graphics formula used to generate square in spiral manner is:

   $$P(x,y) = P1(x1,y1) + \lambda * (P2(x2,y2) - P1(x1,y1)) \qquad (1)$$

   where, $\lambda = 0.8$ by default
   and $\lambda = 0.2$ when prompted for user selected input.

2. Tree Generation:
   For this design, a patch of forest is created by modifying one parent tree. Location of new trees, size of tree trunks and branches are randomly generated using rand() function. Moreover, the branch angle are randomized. The program needs to display trees without erasing till an user input is detected. The formula for generating 2D trees is as follows:

   $$P(x,y) = P2(x1,y1) + \lambda * (P2(x2,y2) - P1(x1,y1)) \qquad (2)$$

   where, $\lambda = 0.8$ by default

   Here, vector P2(x2,y2) - P1(x1,y1) depicts the trunk of the tree. To find a branch at an angle of the trunk, we use the concept of translation and rotation from 2D vector graphics,

   $$P'(x,y) = T^{-1}RT\ P(x,y) \qquad (3)$$

   Here, T = 3x3 Translation Matrix for pre-processing
   R = 3x3 Rotation Matrix
   T^{-1} = 3x3 Translation Matrix for post processing
   α = branch angle with respect to the trunk of the tree.

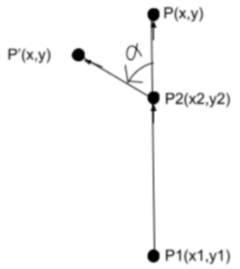

Fig.5 Vector representation of the tree trunk and one branch

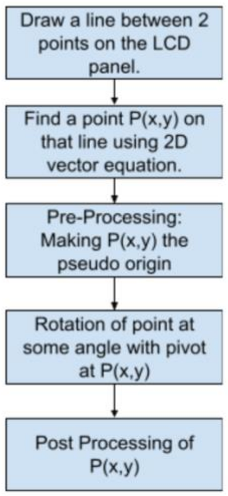
Draw a line between 2 points on the LCD panel.
Find a point P(x,y) on that line using 2D vector equation.
Pre-Processing: Making P(x,y) the pseudo origin
Rotation of point at some angle with pivot at P(x,y)
Post Processing of P(x,y)

Fig. 6 Process steps to generate trees using 2D vectors

4. Testing and Verification

This section focuses on testing and verification of the prototype hardware and software so as to confirm that obtained results matches the desired ones.

4.1 Testing

The following are the test cases:

1. Make sure the Power plug is connected to the J1 power connector port.
2. Check for the power circuit switch to be in ON state and power circuit LED is glowing.
3. Check if all the soldered connection are intact and correct.
4. Make sure the program is run in debug mode on LPC1769 from MCUXpresso IDE.
5. Check if the desired graphics are rendered on the display.

4.2 Verification

Below are the results of the implementation of the project:

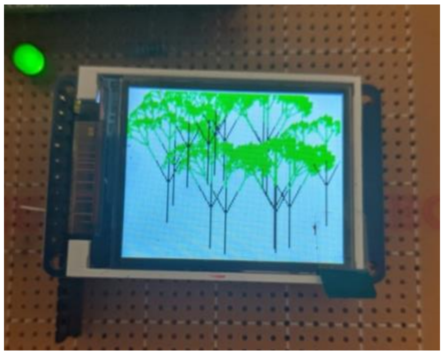
Fig.7 Random Tree generation up to 10 levels of branches

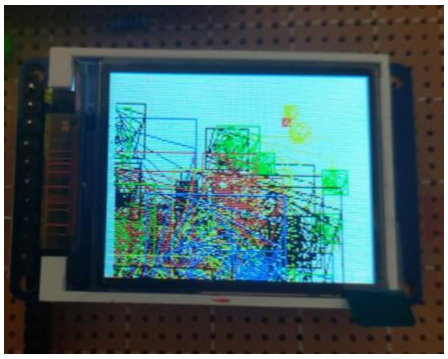
Fig.7 Rotating square patterns

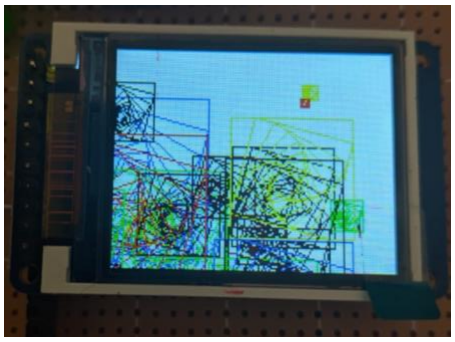
Fig.8 Rotating square generation growing

5. Conclusion

The 2D graphic patterns of rotating squares and trees were successfully rendered on the LCD display interfaced with LPC1769. In Section 8, the source code is attached (Appendix).

6. Acknowledgement

This project was successfully implemented under the mentorship of Dr. Harry Li, Department of Computer Engineering, San Jose State University, CA.

7. References

[1] H. Li, "2018F-107-lecGPP-2018-9-10", *Lecture Notes of CMPE 242*, Computer Engineering Dept., College of Engineering, San Jose State University, September 10, 2018 pp. 1.
[2] Sitronix Technology Corp., "ST7735R 262K Color Single-Chip TFT Controller/Driver", V0.2, May 8, 2009.
[3] NXP Inc., "LPC 1769/68/67/66/65/64/63 data sheet", Revision 9.10, September 8, 2020
[4] en.wikipedia.org/wiki/Serial_Peripheral_Interface

8. Appendix

Below is the source code of the C program for 2D vector graphics.

```
/*
============================================
===================================
 Name        : Divya_Lab2D_CMPE240.c
 Author      : $(author)
 Version     :
 Copyright   : $(copyright)
 Description : main definition
============================================
===================================
*/

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



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

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

#define PORT_NUM           0


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


#define ST7735_TFTWIDTH 127
#define ST7735_TFTHEIGHT 159

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



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

// defining color values

#define LIGHTBLUE 0x00FFE0
#define GREEN 0x00FF00
#define DARKBLUE 0x000033
#define BLACK 0x000000
#define BLUE 0x0007FF
#define RED 0xFF0000
#define MAGENTA 0x00F81F
#define WHITE 0xFFFFFF
#define PURPLE 0xCC33FF
#define YELLOW      0xFFFF00
#define BROWN 0x480000
```

```
int _height = ST7735_TFTHEIGHT;
int _width = ST7735_TFTWIDTH;

#define LAMBDA 0.6
//  30 * 3.14/180 ;
#define CLK 0.52
//-----------------------------------------
-----------------
// Code for drawing a line - Given By Prof.
Harry Li
//-----------------------------------------
-----------------



void spiwrite(uint8_t c);

void writecommand(uint8_t c);

void writedata(uint8_t c);

void writeword(uint16_t c);

void   write888(uint32_t   color,   uint32_t
repeat);

void setAddrWindow(uint16_t x0, uint16_t y0,
uint16_t x1, uint16_t y1);

void fillrect(int16_t x0, int16_t y0, int16_t
x1, int16_t y1, uint32_t color);

void lcddelay(int ms);

void lcd_init();

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

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



//-----------------------------------------
-----------------
// Code for tree generation
//-----------------------------------------
-----------------
void draw_complete_tree();
void  branchReduction(float  x0,  float  y0,
float  x1,  float  y1,  float  lambda,  float
arr[]);
void preProcessing(float x, float y, float
delta_x, float delta_y, float arr_T[]);
void Rotation(float x, float y, float alpha,
float arr_R[]);
void postProcessing(float x, float y, float
delta_x, float delta_y, float arr_P[]);
struct point{
       int x;
       int y;
};

struct tree{
  struct  point  p0,  p1,  right_pt,  left_pt,
reduced_pt;

};

void  drawSquares(struct  point  p0,  struct
point p1, struct point p2, struct point p3,
uint32_t color, uint8_t level, float lambda);

void calculateBranchEnds(const struct point*
p1,const    struct    point*    p2,    float
branch_angle,  float  lambda,  struct  point*
reduced_point, struct point* rotated_point);

//-----------------------------------------
-----------------
// Main
//-----------------------------------------
-----------------
int main (void)

{

        uint32_t pnum = PORT_NUM;

        pnum = 0 ;

        if ( pnum == 0 )
               SSP0Init();

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

        lcd_init();

        while(1)
        {
               fillrect(0,                0,
ST7735_TFTWIDTH, ST7735_TFTHEIGHT, WHITE);

 int optionSel=1;
printf("Please choose from following options
to implement graphics on the LCD module.\n");
printf("select  1  for  drawing  a  rotating
squares.\n");
printf("select 2 for drawing trees.\n");
scanf("%d",& optionSel);

if(optionSel==1)
 {

       float lambda= 0.8;
       printf("Enter lambda value:\n");
       scanf("%f",&lambda);

       int colour;
       int  colorArray[]  =  {BLACK  ,  BROWN,
BLUE, GREEN, RED, MAGENTA, YELLOW};

       struct point p0, p1, p2, p3;

for(int i = 0; i < 50; i++)
       {
       p0.x = rand() % 110;
       p0.y = rand() % 150;
       int len = rand() % 100;
       colour = rand() % 7;

       p1.x = p0.x;
       p1.y = p0.y - len;
       p2.x = p0.x - len;
       p2.y = p1.y;
       p3.x = p2.x;
       p3.y = p0.y;


drawSquares(p0,       p1,       p2,       p3,
colorArray[colour], 10,lambda);

}}
       else if(optionSel==2)
{
       for (int i=0;i<15;i++){
```

```
draw_complete_tree();
                    }
                    lcddelay(5000);
            }

        }
        return 0;
}



float  distance_between_point(struct  point*
p1,struct point* p2){
      return sqrt(pow(p1->x - p2->x, 2) +
pow(p1->y - p2->y, 2));
}

void drawHands(struct point *p0,struct point
*p1,  float  lambda,  float  angle,  uint32_t
color,  struct  point   *reduced_point,struct
point  *p1_right,struct point  *p1_left);

void    tree_maker(float,    struct    tree
*base_tree, struct tree * right_tree, struct
tree * centre_tree, struct tree * left_tree);


void draw_complete_tree()
{

       struct point p0, p1;

       float min_x = 40;
       float height = 50;
       p0.y = rand()% 120 + 30;
       p0.x = rand()% 64 ;
       p1.y = p0.y;
       p1.x = p0.x+ height;

       int variance = 10*3.14/180;
       float    rand_angle    =    CLK;//    +
rand()%(variance*2) - variance;

       struct   point   p1_left,   p1_right,
reduced_point, temp1,  temp2,  temp3,  temp4,
temp5,  temp6;


       drawLine(p0.x,p0.y,p1.x,p1.y,BROWN);
       drawHands(&p0,&p1,LAMBDA,  rand_angle,
BROWN, &p1_left, &reduced_point,&p1_right);

       struct tree base_tree;
    base_tree.p0= p0;
    base_tree.p1= p1;
    base_tree.right_pt = p1_right;
       base_tree.left_pt = p1_left;
       base_tree.reduced_pt= reduced_point;


       int max = 600;
       struct tree tree_array[max]; // 3^10
       tree_array[0]=base_tree;
       for(int i =0, j = 1; i<max && j <max;
i++){
       tree_maker(rand_angle, tree_array + i,
tree_array+j,               tree_array+j+1,
tree_array+j+2);
       j += 3;
       }


}
```

```
struct  point  branchExtension(struct  point*
p0, struct point* p1, float lambda);

void  tree_maker(float  angle,  struct  tree
*base_tree, struct tree * right_tree, struct
tree * centre_tree, struct tree * left_tree){

       left_tree->p0 = base_tree->left_pt;
       left_tree->p1                        =
branchExtension(&(base_tree->reduced_pt),
&(base_tree->left_pt), LAMBDA);
       drawLine(left_tree->p0.x,left_tree-
>p0.y,left_tree->p1.x,left_tree->p1.y,
GREEN);
       drawHands(&(left_tree     ->     p0),
&(left_tree  ->  p1),LAMBDA,  angle,  GREEN,
&(left_tree->left_pt),&(left_tree-
>reduced_pt),&(left_tree->right_pt));


       centre_tree->p0 = base_tree->p1;
       centre_tree->p1                      =
branchExtension(&(base_tree->reduced_pt),
&(base_tree->p1), LAMBDA);
       drawLine(centre_tree-
>p0.x,centre_tree->p0.y,centre_tree-
>p1.x,centre_tree->p1.y, GREEN);
       drawHands(&(centre_tree    ->    p0),
&(centre_tree  ->  p1),LAMBDA,  angle,  GREEN,
&(centre_tree->left_pt),&(centre_tree-
>reduced_pt),&(centre_tree->right_pt));


       right_tree->p0 = base_tree->right_pt;
       right_tree->p1                       =
branchExtension(&(base_tree->reduced_pt),
&(base_tree->right_pt), LAMBDA);
       drawLine(right_tree->p0.x,right_tree-
>p0.y,right_tree->p1.x,right_tree->p1.y,
GREEN);
       drawHands(&(right_tree     ->     p0),
&(right_tree  ->  p1),LAMBDA,  angle,  GREEN,
&(right_tree->left_pt),&(right_tree-
>reduced_pt),&(right_tree->right_pt));

}


void drawHands(struct point *p0,struct point
*p1,  float  lambda,  float  angle,  uint32_t
color,  struct  point   *p1_left,  struct point
*reduced_point,struct point  *p1_right){
       // draw right branch
       calculateBranchEnds(p0,   p1   ,angle,
lambda, reduced_point, p1_right);
       drawLine(reduced_point-
>x,reduced_point->y,p1_right->x,   p1_right-
>y, color);

       // draw left branch
       calculateBranchEnds(p0,  p1  ,-angle,
lambda, reduced_point, p1_left);
       drawLine(reduced_point-
>x,reduced_point->y,p1_left->x,  p1_left->y,
color);
}

//-----------------------------------------
----------------
// Code for drawing rotating square pattern
//-----------------------------------------
----------------
void  drawSquares(struct  point  p0,  struct
point p1, struct point p2, struct point p3,
uint32_t color, uint8_t level, float lambda)
             {
                    struct   point   new_p0,
new_p1, new_p2, new_p3;
                    while(level!=0)
                    {
                           drawLine(p0.x,
p0.y, p1.x, p1.y, color);
                           drawLine(p1.x,
p1.y, p2.x, p2.y, color);
                           drawLine(p2.x,
p2.y, p3.x, p3.y, color);
                           drawLine(p3.x,
p3.y, p0.x, p0.y, color);

                           new_p0.x = p0.x +
lambda * (p1.x - p0.x);
                           new_p0.y = p0.y +
lambda * (p1.y - p0.y);
                           new_p1.x = p1.x +
lambda * (p2.x - p1.x);
                           new_p1.y = p1.y +
lambda * (p2.y - p1.y);
                           new_p2.x = p2.x +
lambda * (p3.x - p2.x);
                           new_p2.y = p2.y +
lambda * (p3.y - p2.y);
                           new_p3.x = p3.x +
lambda * (p0.x - p3.x);
                           new_p3.y = p3.y +
lambda * (p0.y - p3.y);

                           p0 = new_p0;
                           p1 = new_p1;
                           p2 = new_p2;
                           p3 = new_p3;

                           level--;

                    }

             }



void  branchReduction(float  x0,  float  y0,
float x1, float y1, float lambda, float arr[])
{
       arr[0] = x0 + lambda * (x1 - x0);
       arr[1] = y0 + lambda * (y1 - y0);
}

struct  point  branchExtension(struct  point*
p0, struct point* p1, float lambda)
{
       struct point output;
       lambda = 1;
       output.x = p1->x + (lambda) * (p1->x -
p0->x);
       output.y = p1->y + (lambda) * (p1->y -
p0->y);
       return output;
}

void  preProcessing(float  x,  float  y,  float
delta_x, float delta_y, float arr_T[])
{
       arr_T[0] = x - delta_x;
       arr_T[1] = y - delta_y;

}

void Rotation(float x, float y, float alpha,
float arr_R[])
{
       arr_R[0] = cos(alpha) * x - sin(alpha)
* y;
       arr_R[1] = sin(alpha) * x + cos(alpha)
* y;

}

void postProcessing(float x, float y, float
delta_x, float delta_y, float arr_P[])
{
       arr_P[0] = x + delta_x;
       arr_P[1] = y + delta_y;

}



void calculateBranchEnds(const struct point*
p1,const    struct    point*    p2,    float
branch_angle,  float  lambda,  struct  point*
reduced_point, struct point* rotated_point)
{

       float pseudo_origin[2] = {};
       float pt1_wrt_pseudo_origin[2] = {};
       float  rotated_pt_wrt_pseudo_origin[2]
= {};
       float  rotated_pt_wrt_normal_origin[2]
= {};

       //Create Branch
       branchReduction(p1->x,  p1->y,  p2->x,
p2->y, lambda, pseudo_origin);
       preProcessing(p2->x,            p2->y,
*pseudo_origin     ,     *(pseudo_origin+1),
pt1_wrt_pseudo_origin);
       Rotation(*pt1_wrt_pseudo_origin,
*(pt1_wrt_pseudo_origin+1),    branch_angle,
rotated_pt_wrt_pseudo_origin);
       postProcessing(*rotated_pt_wrt_pseudo
_origin,  *(rotated_pt_wrt_pseudo_origin+1),
*pseudo_origin     ,     *(pseudo_origin+1),
rotated_pt_wrt_normal_origin);

       //child branch origin = parent branch
head
       reduced_point->x = *pseudo_origin;
       reduced_point->y = *(pseudo_origin+1);
       rotated_point->x                     =
*(rotated_pt_wrt_normal_origin);
       rotated_point->y                     =
*(rotated_pt_wrt_normal_origin+1);
}

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<<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;

  }

 }

}
```