Two-Dimensional Computer Graphics Using LPC1769 Cortex-M3 Microcontroller

Tse-Jen Lu
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail: tse-jen.lu@sjsu.com

Abstract

Computer graphic plays an important role for interaction with the machine and the users. To get the better understanding of how a microprocessor draws and displays graphs, we use a LPC1769 microcontroller module, which is ARM Cortex-M3 based, connect with a 160x128 resolution LCD Display panel to show the pattern of the rotating rectangles and the trees. Because of the limited computing resource, we rotate a line (vector) at arbitrary position by implement the transform algorithm.

1. Introduction

To get better understanding advance computer design, we build a module with LPC1769 and 1.8" LCD display panel. We trying to draw rotated squares pattern and the tree structures. However, the LPC1769 has limited computing power and limited memory address. Therefore, we avoid using recursion function and need to implement transform function. In this project, we connect LPC1769 with LCD display panel, control it by GPIO and SSP. And discuss how to implement the rotated squares and tree-branches-tree structure base on a drawLine function.

2. Methodology

The flowing section will provides mathematical theories for the translate function, rotate function and combine into a transform function.

2.1. Objectives and Technical Challenges

Since LPC1769 has limited CPU speed, we cannot directly use trigonometric functions like sine or cosine functions for an arbitrary point in a coordinate system. sine and cosine function can be described as an Taylor series expansion. For example, the expansion of sine function at x = 0 shows as below:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Therefore, when the origin point moves to x1, the variable x will become (x - x1). It will cause huge amount of calculation. As result, we cannot calculate a line rotation from arbitrary position expect origin point.

The second challenges are that LPC 1769 has limited memory spaces. We cannot use recursion method for the draw the periodic pattern. For example, tree structure can be described as a trunk periodically to grow up with three branches. Although recursively to call the drawing function is an elegant coding style, the called function will fill too many stacks in the limited memory space to cause memory overflow. This article will provide a method by using two pointers to implement the growing behavior of a tree structure.

2.2. Problem Formulation and Design

I. Drawing a rotating square

The first part is drawing the rotating squares. Since rotate a vector at arbitrary position by calculate Sine and Cosine function spend a huge amount of computation, we need another method to draw the inner square. Therefore, we find the middle point by using the formula as below:

$$P(x', y') = P_1(x_1, y_1) + \lambda \times \left(P_2(x_2, y_2) - P_1(x_1, y_1)\right)$$
$$, where\ \lambda = 0.8$$

The parameter (x1, y1) and (x2, y2) are the start point and end point of a vector, λ determine the position of new point (x', y'), when λ between 0 and1, the new point will be located between (x1, y1) and (x2, y2), In this project, we choose λ = 0.8.

Next, we need four vectors to draw a square. Assume the four points of each corner define as (x1, y1), (x2, y2), (x3, y3) and (x4, y4). The new four vectors can be calculated as below:

$$P(x_1', y_1') = P_1(x_1, y_1) + \lambda \times \left(P_2(x_2, y_2) - P_1(x_1, y_1)\right)$$
$$P(x_2', y_2') = P_2(x_2, y_2) + \lambda \times \left(P_3(x_3, y_3) - P_2(x_2, y_2)\right)$$
$$P(x_3', y_3') = P_3(x_3, y_3) + \lambda \times \left(P_4(x_4, y_4) - P_3(x_3, y_3)\right)$$
$$P(x_4', y_4') = P_4(x_4, y_4) + \lambda \times \left(P_1(x_1, y_1) - P_4(x_4, y_4)\right)$$

Finally, to repeatedly draw the inner squares, set a for-loop to repeatedly count and draw the four inner vectors. Therefore, the LCD panel will display the rotated square with several inner squares.

II. Draw a Three-Branches-Tree

The second part is drawing a three-branches-tree. It contains several steps:

1. we need to use a rotate function to rotate a vector from origin point. The rotation function shows as below:

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

Where α is the angle that rotate from x-axis to y-axis on xy-plane. Or we can rewrite as:

$$x_i' = x_i\, cos(\alpha) - y_i\, sin(\alpha)$$
$$y_i' = x_i\, sin(\alpha) + y_i\, cos(\alpha)$$

2. For rotate the vectors from an arbitrary position, the second step is to implement the translation matrix into transform function. The purpose of translation matrix is to shift the pivot point to the origin point and then shift back. The translate matrix can be described as:

$$T = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

And inversed translated matrix can be calculated and described as:

$$T^{-1} = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore, the overall transform function is combine translate matrix times rotation function, it can be described as:

$$p' = T^{-1} * R * T * p$$

Or,

$$\begin{pmatrix} x_i' \\ y_i' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & -\Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$$

Where Δx and Δy are the distance difference between origin point and pivot point. By using the transform function that we show above, we can now draw the branches now at any position in the display panel.

To repeatedly draw three branches from the previous branch, we first created an array with two times power of ten base two. And create two variables "fast" and "slow", slow variable point to the array variable which is the top of the root branch, use this point calculate the points of branches by using transform function, the result will draw the line and put into the next array address which is pointed by fast variable. The program will be stopped when fast variable equals the length of the array.

3. Implementation

The microcontroller can be separated to two parts. One is hardware, another one is software, and integrated each part of design.

3.1. Hardware Design

There are two parts of hardware design. One is LPC1769 microcontroller module and another one is Adafruit 1.8" SPI display panel. Since the 1.8" LCD panel has 128 * 160 color pixels, three sub-pixels per pixel, each sub-pixel has 8-bit, the frame rate set as 30. The bandwidth will be chosen. We therefore select SPI as our transmission.

For the power system, we choose power regular 7805 to provide 5-volt input for the LPC1769 module. Besides, the micro USB can also provide 5-volt to the LPC1769.

For the SPI connection between LPC1769 and LCD display panel, the mapping table can be shown as Table 1. And the diagram of our system shows in the Fig. 1., the implement result shows as Fig. 2.

Table. 1. LCD and LPC1769 mapping table

LCD Pins	LPC1769 Pins
GND	GND
VCC	VCC (3.3)
RST	GPIO Output (P0.22)
RS/DC	RS/DC / GPIO output (P0.21)
TFT_CS	SSEL0 (P0.16)
MOSI	SSP0 MOSI (P0.18)
SCK	SCK0 (P0.15)
MISO	SSP) MISO (P0.17)
LITE	VCC (3.3)

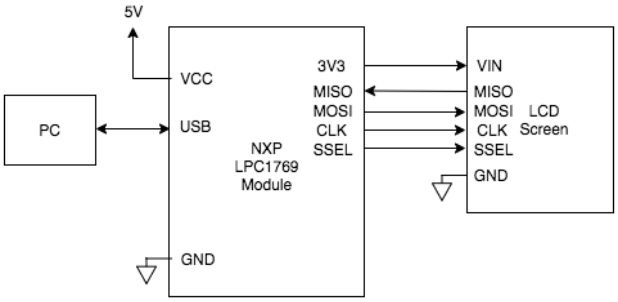


Fig.1 System Block Diagram

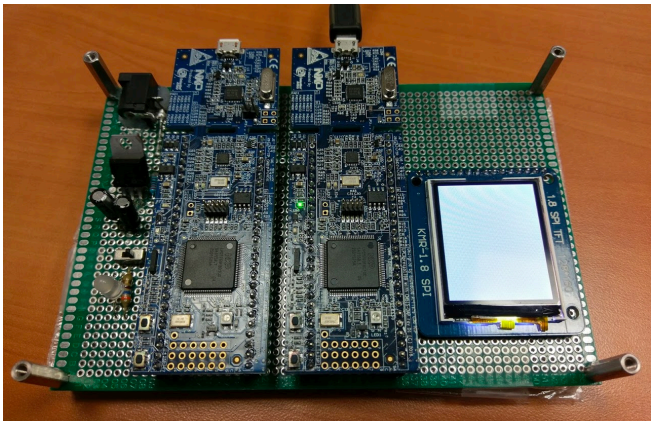


Fig. 2 System Implementation

3.2. Software Design

a. Software Development Environment:

The software development environment is based on the MCUXpresso IDE which is Eclipse-based development

environment, which is design for NXP MCUs based ARM Cortex-M microcontrollers. The program including three patches; GPIO, SSP and DrawLine which is including the driver ST7735 for the Adafruit 1.8" SPI display panel. The DrawLine function is filling the color in each pixel between two points. In the following lab, we draw the patterns by calculated the points as input variables of the DrawLine function.

b. Algorithm Description:

- Pseudo code for the Rotation Squares:

drawRect(coordinate of two, number of layers) {
calculate the position of each corners;
Set variable lambda equal zero for the first round.
for (number of layers) {
find tour new positions of the corner by the formula:

$$P(x', y') = P_1(x_1, y_1) + \lambda \times \left(P_2(x_2, y_2) - P_1(x_1, y_1)\right)$$

drawLine by the new positions;
Set lambda equal 0.8;
}
}

- Pseudo code for Three-Branches-Tree

1. Define struct Point with variable x and y.
2. Define "Rotate" Function to calculate rotated point:

Point Rotate (point 1 (the bottom of a branch) , point 2 (the top of a branch), rotate angle) {
Define translation matrix, rotation matrix, inversed translation matrix;
Multiply rotation matrix and inverse translation matrix into a temporary matrix;
Multiply inversed translation matrix and the temporary matrix;
Get new point by formula:

$$x_i' = x_i\, cos(\alpha) - y_i\, sin(\alpha)$$
$$y_i' = x_i\, sin(\alpha) + y_i\, cos(\alpha)$$

Return new point;
}

3. Since we want every new branch is 0.8 longer than previous branch and every rotation has random angel varies. Define a function "branchPoint" as below:

Point Rotate (point 1 (the bottom of a branch) , point 2 (the top of a branch), rotate angle) {
Rotation angel add various angel by calling a random function;
Define a new point with 0.8 longer than previous branch by point 1 and point 2,
Now we get a new branch extend from previous branch;
Get new point by calling "Rotate" function to rotate the branch;
Return new point;
}

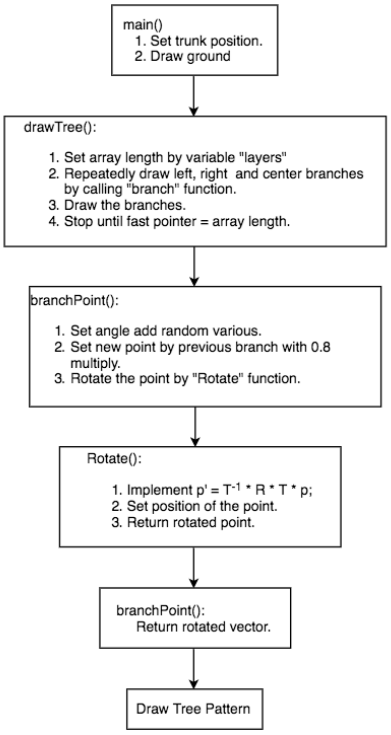
4. Define a "drawTree" function. Implement the algorithm that repeatedly draw the new branches from the previous branch.

Void drawTree (bottom point of a trunk, top point of a trunk, number of layers) {
*Set the size of a 2D array "arr" with 2*pow(2, layers) for the raw and 2 for the column.*
Initialize the array arr[0][0] with the bottom point of a trunk;
Initialize the array arr[0][1] with the top point of a trunk;
Set two pointers "fast" and "slow", fast pointer point to 1 and slow pointer point to 0 which is the trunk;

While (fast pointer < the length of the array) {
Find branch by calling "branchPoint" function by input slow pointer which is point to the root branch itself;
Put the new point into the array which is indicate by fast pointer;
Draw the new line by the new point;
Move the faster pointer to the next array address.

Repeat the action above, but input the angel with +30 and -30;
After draw the three branches, move the slow pointers to the next;
}
}

The flow chart is as follows:



5. Plus, to draw leafs, set the color change to green when fast pointer more than a certain number. Here we set the number to 64.
6. At the main function, setup the trunk position as the input of "drawTree" function.

4. Testing and Verification

To test the drawing function, we first setup the software environment and test the GPIO functions. By using MCUXpresso IDE as our platform, import "2018S-11-GPIO-2015-1-30.zip", "2018S-12-SSP_Test.zip" and "2018S-13-2017F-112-LCD-DrawLine.zip". [2] We can see a single line appear on the screen. Shows as Fig. 3. After that, we can make sure that our environment is ready.

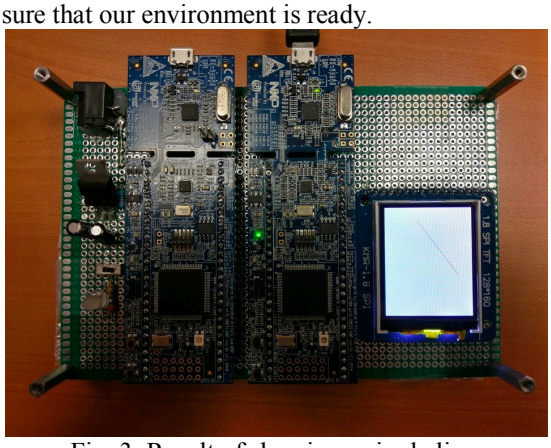


Fig. 3. Result of showing a single line.

For the first part of this project, drawing the rotating squares. We first draw a single rotating square in the whole frame. Shows as Fig. 4:

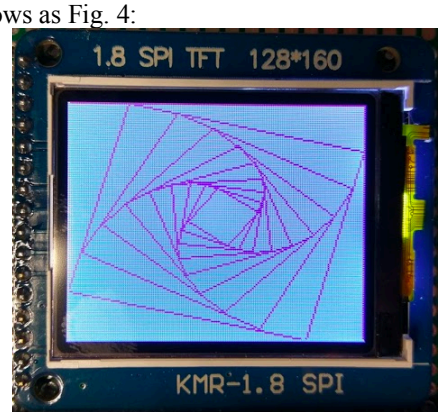


Fig. 4. Single rotating square.

Next, implement random function let the rotating squares has random position, size and color. Shows as Fig. 5:

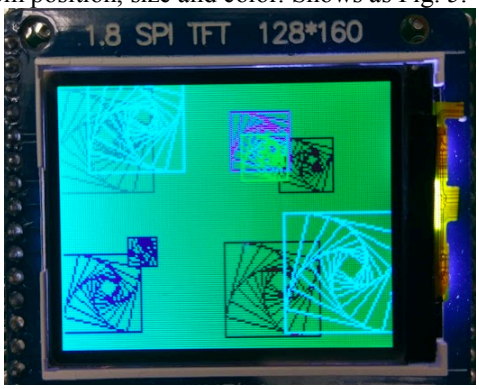


Fig. 5. Multiple rotating square with random position, size and color.

The second part is drawing three-branches-tree. We first implement the rotation function; the vectors can rotate any angle from the origin point. Show as Fig. 5.

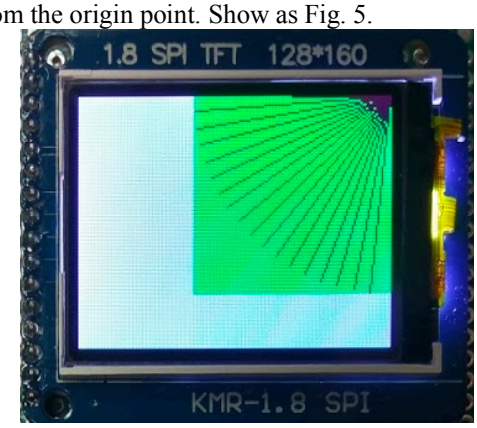


Fig. 5. Implement the rotation function.

Next, Implement the transform function to let the vectors can rotate at arbitrary position. Here, we set the vector rotate from the position (30, 30). The result show as Fig. 6.

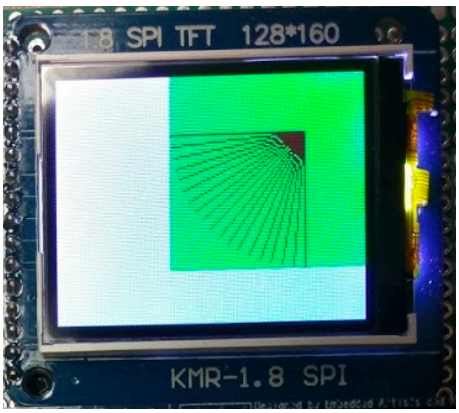


Fig. 6. Implement the transform function

Now, we implement the drawTree function to draw a simple tree with three branches. The result shows as Fig. 7. and Fig. 8.

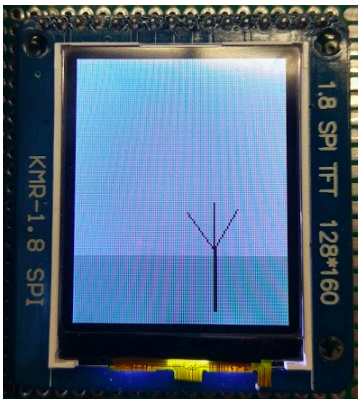


Fig. 7. Implement the drawTree function

Fig. 7. Implement the function with more loops.

Finally, implement the autoDrawTree function, set the number of tree and number of tree layers. Also let the tree growing at random position and fill the color. The result shows as Fig. 8.

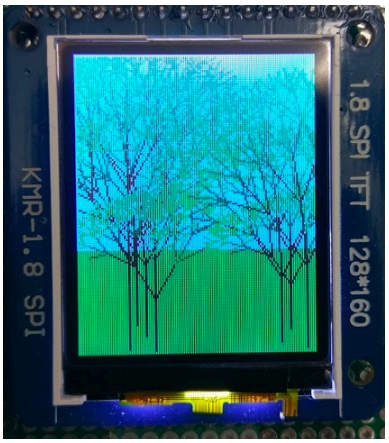


Fig. 8. Final result of Three-Branches-Tree.

5. Conclusion

In this project, we build a microcontroller by LPC1769 and try to draw two patterns on the LCD display panel. Since limited computational power, we implement the transform function to draw the vectors at arbitrary positions. Another problem is that we cannot use recursion method to call the draw tree function, we therefore implement the two pointer method to repeatedly draw three branches from the previous branch. Finally we complete the function of "rotating squares" and "Three-Branches-Tree".

6. References

[1] Li, Hua. "Three-dimensional computer graphics using EGA or VGA card." IEEE Transactions on Education 35.1 (1992): 44-49.

[2] Li, Hua. "2018S-13-2017F-112-LCD-DrawLine.zip" github. https://github.com/hualili/CMPE240-Adv-Microprocessors/blob/master/2018S-13-2017F-112-LCD-DrawLine.zip

[3] Taylor series expansion of sine function, https://en.wikipedia.org/wiki/Sine#cite_note-3

7. Appendix

Source code is listed at next pages.

Appendix.

1- Source Code – Rotate Squares (Function part only.)

```
/*
============================================================================
 Name        : RotateSquare.c
 Author      : $RJ
 Version     :
 Copyright   : Tse-Jen Lu, 011490492
 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>
#include<time.h>

// 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 BROWN 0x783F04
#define GRASS 0XB6D7A8
#define SKY 0XCFE2F3
#define LEAF 0X4BCC14

/*
 Main drawRect()
 */
void drawRect(int p0_x, int p0_y, int p2_x, int p2_y, int16_t layers, int
COLOR) {
   //set 4 points
   int16_t p1_x = p0_x;
   int16_t p1_y = p2_y;
   int16_t p3_x = p2_x;
   int16_t p3_y = p0_y;

   //For loop
   int16_t pTemp_x = p0_x;
   int16_t pTemp_y = p0_y;
   float lambda = 0;
   for (int i = 0; i < layers; i++) {
      pTemp_x = p0_x;
      pTemp_y = p0_y;

      p0_x = p0_x + lambda * (p1_x - p0_x);
      p0_y = p0_y + lambda * (p1_y - p0_y);

      p1_x = p1_x + lambda * (p2_x - p1_x);
      p1_y = p1_y + lambda * (p2_y - p1_y);

      p2_x = p2_x + lambda * (p3_x - p2_x);
      p2_y = p2_y + lambda * (p3_y - p2_y);

      p3_x = p3_x + lambda * (pTemp_x - p3_x);
      p3_y = p3_y + lambda * (pTemp_y - p3_y);

      //       drawLine(x0,y0,x1,y1,PURPLE);
      //p0 - p1
      drawLine(p0_x, p0_y, p1_x, p1_y, COLOR);
      //p1 - p2
      drawLine(p1_x, p1_y, p2_x, p2_y, COLOR);
      //p2 - p3
      drawLine(p2_x, p2_y, p3_x, p3_y, COLOR);
      //p3 - p0
      drawLine(p3_x, p3_y, p0_x, p0_y, COLOR);

      lambda = 0.8;
   }
}

/*
 * AutoDraw
 */
void autoDraw(int num) {
   srand(time(0));

   int color[9] = {LIGHTBLUE, LEAF, DARKBLUE, WHITE, BLUE,
RED, MAGENTA, PURPLE, BROWN};
   for (int i = 0; i < num; i++) {
      int gap = 15;
      int px = (rand()%(ST7735_TFTWIDTH - 2 * gap)) + gap;
      int py = (rand()%(ST7735_TFTHEIGHT - 2* gap)) + gap;
      int size = (rand()%(4 * gap)) / 2;

      drawRect(px - size, py - size, px + size, py + size, 10,
color[rand()%9]);
   }
}

/*

 Main Function main()

 */
int main (void)

{

   uint32_t pnum = PORT_NUM;

   pnum = 0 ;

   if ( pnum == 0 )
      SSP0Init();

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

   lcd_init();

   fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, GRASS);
   printf("ST7735_TFTWIDTH: %d \n", ST7735_TFTWIDTH);
   printf("ST7735_TFTHEIGHT: %d \n", ST7735_TFTHEIGHT);

   char name[100];
   while (name[0] != '0' ) {
      // Print function here.
      autoDraw(9);
      scanf("%s", name);
      printf("Your Name is: %s \n", name);
      fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT,
GRASS);
```

```
    }

    return 0;
}
```

2- Source Code – Three-Branches-Tree (Function part only.)

```
/*

============================================================================
 Name        : Tree.c
 Author      : $RJ
 Version     :
 Copyright   : Tse-Jen Lu / 011490492
 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 <stdbool.h>
#include <math.h>
#include<time.h>

// 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 BROWN 0x783F04
#define GRASS 0XB6D7A8
#define SKY 0XCFE2F3
#define LEAF 0X4BCC14

/*
 * Rotation, Rotate from Zero.
 */
//https://stackoverflow.com/questions/22957175/how-does-function-actually-return-struct-variable-in-c
struct Point RotateFromZero(int degree, struct Point p) {
    int alpha = degree * 3.1415926 / 180;
    int temp = p.x;
    p.x = p.x * cos( alpha ) - p.y * sin( alpha );
    p.y = temp * sin( alpha ) + p.y * cos( alpha );
    return p;
}

/*
 * Rotation, Rotate from Any Point.
 */
struct Point Rotate(struct Point p1, struct Point p2, int degree) {
    float alpha = degree * 3.1415926 / 180;
    //    int dx = p2.x - p1.x;
    //    int dy = p2.y - p1.y;
    float dx = -1 * p1.x;
    float dy = -1 * p1.y;

    float T[3][3] = {{1, 0 , dx}, {0, 1, dy}, {0, 0, 1}};
    float R[3][3] = {{cos(alpha), -sin(alpha), 0}, {sin(alpha), cos(alpha), 0}, {0, 0, 1}};
    float IT[3][3] = {{1, 0, -dx}, {0, 1, -dy}, {0, 0, 1}};
    float Pre_TSigma[3][3] = {{0, 0 , 0}, {0, 0, 0}, {0, 0, 0}};
    // Matrix R * Matrix T
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++){
                Pre_TSigma[i][j] += R[i][k] * T[k][j];
            }
        }
    }
    float TSigma[3][3] = {{0, 0 , 0}, {0, 0, 0}, {0, 0, 0}};
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++){
                TSigma[i][j] += IT[i][k] * Pre_TSigma[k][j];
            }
        }
    }

    struct Point p_prime = {0, 0};
    p_prime.x = TSigma[0][0] * p2.x + TSigma[0][1] * p2.y + TSigma[0][2];
    p_prime.y = TSigma[1][0] * p2.x + TSigma[1][1] * p2.y + TSigma[1][2];
    return p_prime;
}

/*
 * find branchPoint
 */
struct Point branchPoint(struct Point p1, struct Point p2, int degree) {
    srand(time(0));
    degree = degree - rand()%11;

    struct Point p_temp = {0,0};
    p_temp.x = p2.x + lambda * (p2.x - p1.x);
    p_temp.y = p2.y + lambda * (p2.y - p1.y);
    struct Point p_prime = {0,0};
    p_prime = Rotate(p2, p_temp, degree);
    return p_prime;
}

void drawTree (struct Point TrunkDown, struct Point TrunkUp, int layers) {
    int arrLength = 2 * pow(2 , layers);
    struct Point arr[arrLength][2]; //arr[][] = {TrunkDown, TrunkUp}, {pRight, TrunkDown}, {pCenter, TrunkDown}...
    arr[0][0] = TrunkDown;
    arr[0][1] = TrunkUp;
    int fast = 1;
    int slow = 0;

    int COLOR;
    while (fast < arrLength) {
        if (fast < 8) {
            COLOR = BROWN;
        } else if (fast < 32){
            COLOR = LEAF;
        } else {
            COLOR = GREEN;
        }
        if ( fast < arrLength) {
            arr[fast][0] = arr[slow][1];
            arr[fast][1] = branchPoint(arr[slow][0], arr[slow][1], 0);
```

```
            drawLine(arr[fast][0].x, arr[fast][0].y, arr[fast][1].x,
arr[fast][1].y, COLOR);
            fast++;
        }
        if ( fast < arrLength) {
            arr[fast][0] = arr[slow][1];
            arr[fast][1] = branchPoint(arr[slow][0], arr[slow][1], -30);
            drawLine(arr[fast][0].x, arr[fast][0].y, arr[fast][1].x,
arr[fast][1].y, COLOR);
            fast++;
        }
        if ( fast < arrLength) {
            arr[fast][0] = arr[slow][1];
            arr[fast][1] = branchPoint(arr[slow][0], arr[slow][1], 30);
            drawLine(arr[fast][0].x, arr[fast][0].y, arr[fast][1].x,
arr[fast][1].y, COLOR);
            fast++;
        }
        slow++;
    }
}
/*
 * Auto Draw Tree
 */
void autoDrawTree(int num, int layers) {

    srand(time(0));
    for (int i = 0; i < num; i++) {
        int px = rand()%ST7735_TFTWIDTH;
        float py = rand()%(50);
        int tall = (rand()%30) + 20;
        struct Point TrunkDown = {px, py};
        struct Point TrunkUp = {px, py + tall};

        drawLine(TrunkDown.x, TrunkDown.y, TrunkUp.x, TrunkUp.y,
BROWN);
        drawTree (TrunkDown, TrunkUp, layers);
    }
}

/*
 Main Function main()
 */
int main (void)
{
    uint32_t pnum = PORT_NUM;
    pnum = 0 ;
    if ( pnum == 0 )
        SSP0Init();
    else
        puts("Port number is not correct");
    lcd_init();
    //Sky and GRASS
    fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, SKY);
    fillrect(0, 0, ST7735_TFTWIDTH, 0.4 * ST7735_TFTHEIGHT,
GRASS);

    char key[100];
    while (key[0] != '0' ) {
        // Print function here.
        autoDrawTree(5, 6); //layers = 6 for demo.
        scanf("%s", key);
        printf("Your key is: %s \n", key);
        fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, SKY);
        fillrect(0, 0, ST7735_TFTWIDTH, 0.4 * ST7735_TFTHEIGHT,
GRASS);
    }
    fillrect(0, 0, ST7735_TFTWIDTH, ST7735_TFTHEIGHT, SKY);
    return 0;
}
```