同济大学计算机系

OOP 钟表大作业实验报告

学　　号 **2152118**

姓　　名 **史君宝**

专　　业 **计算机科学与技术（计科 1 班）**

完成时间 **2023.10.7**

一、设计思路与功能描述

1. 得分点

（1）独立设计
（2）完成抗锯齿
（3）图形优化算法

2. 设计思路

（1）与日常结合。

当我们谈到时间的时候，我们自然而然与日常结合。随着时间的流逝，一天的光阴也在不断的变化，今天的结束是明天的开始，一天中的光线和色彩的变化从明到暗，再从暗到明。
因此设计时希望结合这种首尾相接的潮汐变化感，结合这种光线色彩的变化，所以会想到用渐变色，随着时间的推移，从蓝到红，再从红到蓝，对应这一天的明暗变化。

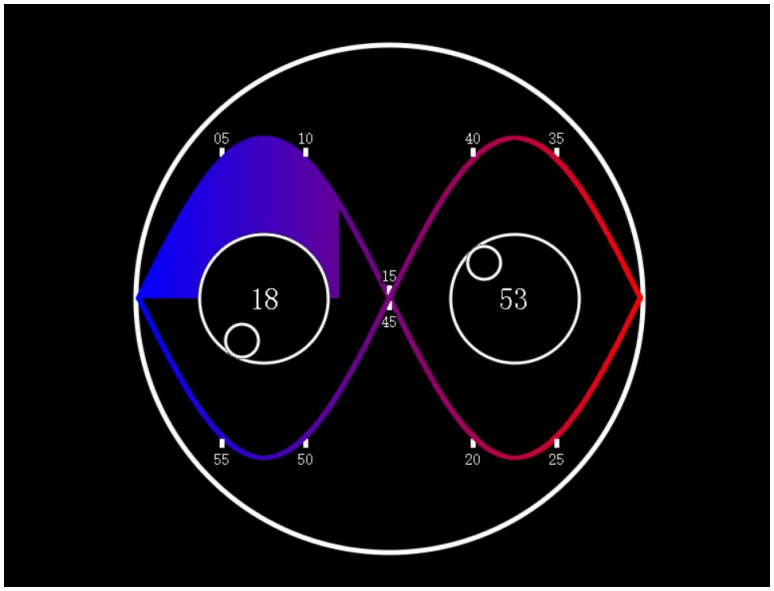
（2）新的计时工具。

时间的概念自从出现，人们就像抓住它、珍惜它，从最初的水滴、日晷等等计时工具不断演化。如今我们已经知道的较为精确的，用于校时的工具大多都用到了量子力学中正弦性的概念，比如铯原子钟。因此正弦函数在本次设计中也加以考虑。

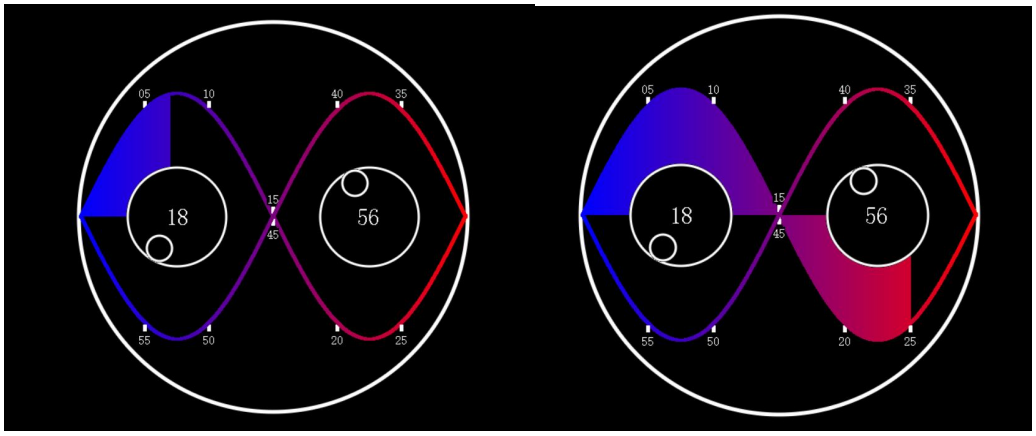
（3）简约形象。

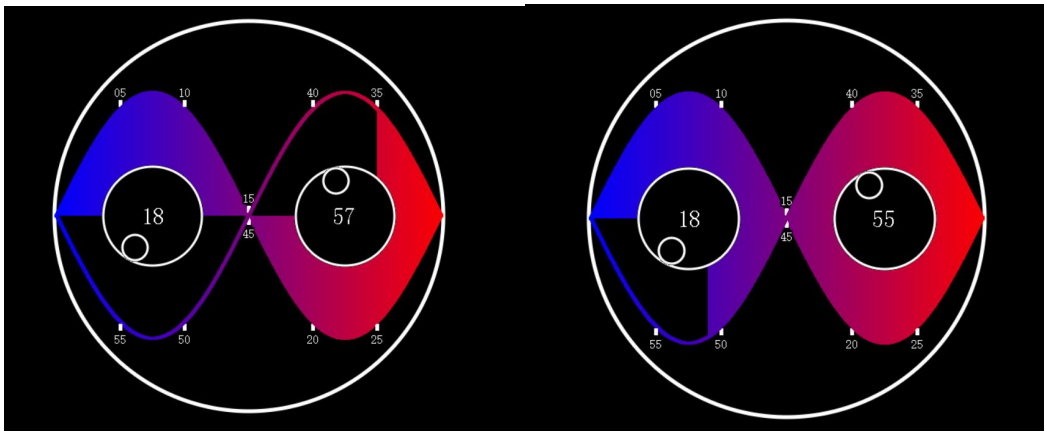
看到了很多学长的作品，都很不错，有的样式精美，但是却忽略了简约的风格，有的甚至不能一眼看出当前时间。因此本次设计决定采用简单的色彩，尽量时他人能够在知晓方法后瞬间知道当前时间。同时为了形象，也借助了中国太极的设计思想。

（4）设计成品：



其中左右两个小半圆分别对应的是小时和分钟，如当前时间就是 18:53。而中间渐变的正在渲染正弦函数的就是秒针的变化，它的涂色顺序如下：





3. 功能描述

（1）函数声明：

```
void init();           //初始化
void init_sec();       //初始化秒针
void pic_again(int hour, int min, int sec);     //再次绘画时间
void new_hour();       //绘制新的时针
void new_min();        //绘制新的分针
void new_sec();        //绘制新的秒针
void sin_xy();         //因为需要计算正弦函数，所以封装了一个计算过程
void buffer_sin();

//封装一个混合颜色的函数，利用ALpha实现抗锯齿
COLORREF mix_color(COLORREF bg, COLORREF color, double alpha);
//画一个没有锯齿的圆
void SDF_circle(Point p, COLORREF color, int r, double line_width);
```

（2）主函数：

功能:初始化屏幕，并获取当前时间，并将时间与已经存储的时间相对比，以此来检验是否时间发变化，并重新绘制图片。

```
int main()
{
    time(&now);
    localtime_s(&t, &now);  // 获取当地时间


    initgraph(800, 600);    // 图形方式初始化
    sin_xy();
    init();         // 自定义图形初始化函数，用于绘制时钟界面

    int cur_hour = t.tm_hour;
    int cur_min = t.tm_min;
    int cur_sec = t.tm_sec;

    while (!_kbhit())   // 无键盘操作时进入循环
    {
        time(&now);
        localtime_s(&t, &now);  // 获取当地时间

        if (cur_sec != t.tm_sec)
        {
            pic_again(t.tm_hour, t.tm_min, t.tm_sec);

            cur_hour = t.tm_hour;
            cur_min = t.tm_min;
            cur_sec = t.tm_sec;
        }
    }
    _getch();       // 按任意键准备退出时钟程序
    closegraph();       // 退出图形界面
    return 0;
}
```

（3）init 函数

功能:初始化屏幕，并将刚开始的时针，分针，秒针画出来。

```
47  void init()
48  {
49
50      cleardevice();
51
52      setbkcolor(COLORREF(BLACK));
53
54      init_sec();
55
56      new_sec();
57      new_hour();
58      new_min();
59
60  }
```

（5）init_sec 函数

功能:调用 buffer_sin 函数将刚开始的秒针的正弦函数画出来。

```
void init_sec()
{
    double line_width = 5;

    Point p;
    p.x = circle_x;
    p.y = circle_y;
    SDF_circle(p, WHITE, circle_radis, line_width / 2);

    buffer_sin();
}
```

（6）pic_again 函数

功能:利用双缓冲区将新的时针、分针、秒针画出来。

```
void pic_again(int hour, int min, int sec)
{
    BeginBatchDraw();

    new_sec();
    new_hour();
    new_min();

    FlushBatchDraw();

    EndBatchDraw();
}
```

（7）New_hour 函数

功能:具体地将时针画出来。

```
void new_hour()
{
    clearcircle(circle_x - circle_radis / 2, circle_y, circle_radis / 4);

    settextstyle(30, 0, _T("SimSun"));

    TCHAR s[20];
    _stprintf_s(s, _T("%02d"), t.tm_hour);
    outtextxy(275-15, 300-15, s);

    double line_width = 3;

    Point p1;
    p1.x = circle_x - circle_radis / 2;
    p1.y = circle_y;

    SDF_circle(p1, WHITE, circle_radis / 4, line_width / 2);

    int hour_pie_x;
    int hour_pie_y;
    double hour_pie_radis = t.tm_hour * 1.0 / 6 * PI + t.tm_min * 1.0 / 360 * PI;

    hour_pie_x = int(circle_x - circle_radis / 2 + circle_radis / 4 * 0.75 * sin(hour_pie_radis));
    hour_pie_y = int(circle_y - circle_radis / 4 * 0.75 * cos(hour_pie_radis));


    Point p2;
    p2.x = hour_pie_x;
    p2.y = hour_pie_y;

    SDF_circle(p2, WHITE, circle_radis / 16, line_width / 2);

}
```

（8）New_min 函数

功能:具体地将分针画出来。

```
void new_min()
{
    clearcircle(circle_x + circle_radis / 2, circle_y, circle_radis / 4);

    settextstyle(30, 0, _T("SimSun"));

    TCHAR str[20];
    _stprintf_s(str, _T("%02d"), t.tm_min);
    outtextxy(525-15, 300-15, str);

    double line_width = 3;
    Point p1;
    p1.x = circle_x + circle_radis / 2;
    p1.y = circle_y;

    setlinestyle(PS_SOLID | PS_ENDCAP_FLAT, 3);

    int min_pie_x;
    int min_pie_y;

    double min_pie_radis = t.tm_min * 1.0 / 30 * PI + t.tm_sec * 1.0 / 1800 * PI;
    min_pie_x = int(circle_x + circle_radis / 2 + circle_radis / 4 * 0.75 * sin(min_pie_radis));
    min_pie_y = int(circle_y - circle_radis / 4 * 0.75 * cos(min_pie_radis));

    Point p2;
    p2.x = min_pie_x;
    p2.y = min_pie_y;
    SDF_circle(p1, WHITE, circle_radis / 4, line_width / 2);
    SDF_circle(p2, WHITE, circle_radis / 16, line_width / 2);

}
```

（9）New_sec 函数

功能:具体地将秒针画出来。

```
void new_sec()
{
    setlinestyle(PS_SOLID | PS_ENDCAP_FLAT, 5);
    clearcircle(circle_x, circle_y, circle_radis);

    init_sec();

    double hour_bar_one = startX;
    double hour_bar_two = endX;

    setlinestyle(PS_SOLID | PS_ENDCAP_FLAT, 1);

    if (t.tm_sec < 30) {
        hour_bar_one = startX + (circle_radis * 1.0 / 15) * t.tm_sec;
    }
    else {
        hour_bar_one = endX;
        hour_bar_two = endX - (circle_radis * 1.0 / 15) * (t.tm_sec - 30);
    }

    for (int x = startX; x <= int(hour_bar_one); ++x)
    {
        setcolor(RGB(r[x - startX], 0, b[x - startX]));
        line(x, sin_x[x - startX] + 2, x, circle_y);
    }

    for (int x = endX; x >= int(hour_bar_two); --x)
    {
        setcolor(RGB(r[x - startX], 0, b[x - startX]));
        line(x, sin_y[x - startX] - 2, x, circle_y);
    }

}
```

二、问题与解决方法

1. 遇到的问题

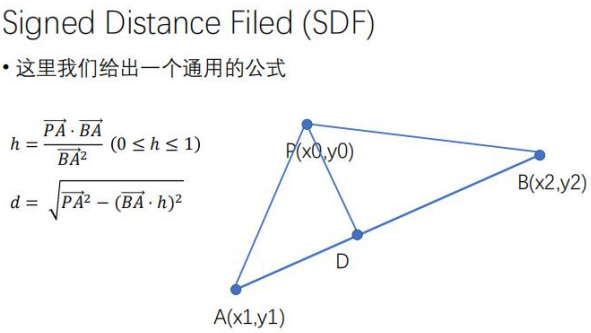
（1）抗锯齿的问题。
（2）正弦函数比较难画，过程中计算量比较大。
（3）出现了图像闪烁的问题，需要减少画图时间。

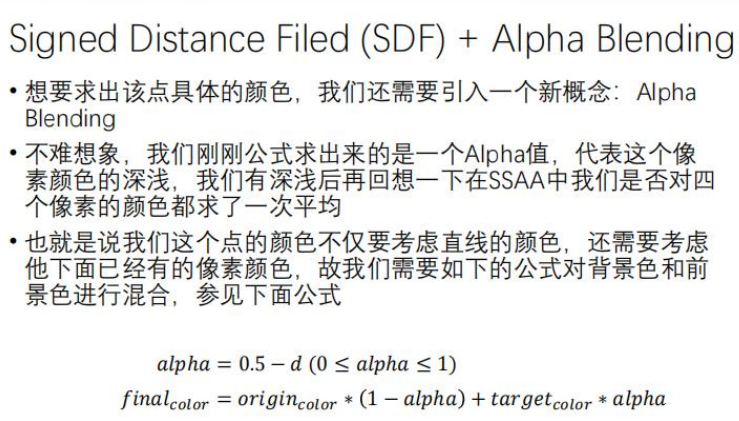
2. 解决的方法

（1）抗锯齿的问题。

在画图过程中由于要花圆，其中的锯齿现象十分严重，就需要进行一定的优化。在老师所发的参考资料中其实有很多方法，比如 SSAA、SDF、SDF+Alpha 方法。由于在上面的时钟中要画的比较过，所以我选择了 SDF+Alpha 的方法，并构建了两个函数用来实现它。

SDF+Alpha 的方法原理：





具体解决：

设计了两个函数，分别是 SDF_circle 和 mix_color 一同解决问题。
SDF_circle 函数，首先确定要画图的具体位置，即两个循环，然后根据具体的坐标算出与圆形的距离，并减去半径。由于要画的圆线条有一定的宽度，所以采用了绝对值进行了处理。最后根据具体的距离，算出对应的 Alpha 值，并对颜色进行一定的混合即调用 mix_color 函数，最后实现填涂就可以了。

```
void SDF_circle(Point p, COLORREF color, int r, double line_width)
{
    for (int i = p.x - r - 10; i < p.x + r + 10; i++) {      //所画的x的变化区域
        for (int j = p.y - r - 10; j < p.y + r + 10; j++) {  //所画的y的变化区域
            Point p1 = { i,j };             //其中任一点的坐标
            double d;                       //与圆形的距离

            d = abs(sqrt((pow(p1.x - p.x, 2) + pow(p1.y - p.y, 2))) - r - line_width);
            //由于圆边有宽度，所以取绝对值进行了一定的处理

            double alpha = 0.5 - (d - line_width);  //计算Alpha值
            if (d <= line_width) {                  //圆宽度内涂白
                putpixel(i, j, WHITE);
            }
            if (alpha >= 0 && alpha <= 1) {         //边缘化进行混合颜色处理
                COLORREF bg = getbkcolor();
                COLORREF result = mix_color(bg, color, alpha);
                putpixel(i, j, result);
            }
        }
    }
}
```

```
COLORREF mix_color(COLORREF bg, COLORREF color, double alpha)
{
    COLORREF result;
    int r = int(GetRValue(bg) * (1 - alpha) + GetRValue(color) * alpha);
    int g = int(GetGValue(bg) * (1 - alpha) + GetGValue(color) * alpha);
    int b = int(GetBValue(bg) * (1 - alpha) + GetBValue(color) * alpha);
    result = RGB(r, g, b);
    return result;
}
```

（2）正弦函数难画的问题。

首先我们根据正弦函数的计算公式，将对应的 x 的坐标与对应的 y 的坐标全部都算出来了，然后我们将其储存在自定义的全局变量的数组中。同时由于正弦函数的绘制过程中需要对其边的颜色进行一定的变化，所以就自行设计构建了对应的渐变颜色与坐标之间的关系，在之后通过绘制就玩车鞯了正弦函数的变化。

之后需要对秒针变化，正弦函数内的填涂进行一定的处理，同样，也是构建了随时间变化与坐标之间的关系，通过时间的变化能够将其对应起来，就完成了绘制了。

全局变量：

```
int startX = circle_x - circle_radis;        //正弦函数起始坐标
int endX = circle_x + circle_radis;          //正弦函数终止坐标

double amplitude = (2*circle_radis) / PI;  // 正弦函数的振幅

int sin_x[2 * circle_radis + 1];           //储存上面的正弦函数与x坐标之间的关系
int sin_y[2 * circle_radis + 1];           //储存下面的正弦函数与x坐标之间的关系
int r[2 * circle_radis + 1];               //实现渐变色
int b[2 * circle_radis + 1];
```

正弦函数的绘制：

```
for (int x = startX; x <= endX; ++x)
{
    setcolor(RGB(r[x - startX], 0, b[x - startX]));
    line(x, sin_x[x - startX] - 2, x, sin_x[x - startX] + 2);

    setcolor(WHITE);
    settextstyle(16, 0, _T("SimSun"));
    if ((x - startX) % (circle_radis / 3) == 0 && x != startX && x != endX)
    {
        if (x <= circle_x)
        {
            line(x, sin_x[x - startX] - 3, x, sin_x[x - startX] - 12);
            _stprintf_s(abc, _T("%02d"), hour_num);     // 高版本 VC 推荐使用 _stprintf_s 函数
            outtextxy(x - 8, sin_x[x - startX] - 30, abc);
            hour_num += 5;
        }
        else
        {
            line(x, sin_x[x - startX] + 3, x, sin_x[x - startX] + 12);
            _stprintf_s(abc, _T("%02d"), hour_num);     // 高版本 VC 推荐使用 _stprintf_s 函数
            outtextxy(x - 8, sin_x[x - startX] + 15, abc);
            hour_num += 5;
        }

    }
}
```

（3）图形闪烁问题：

在上面的过程中由于有很多的计算，同时伴随着清屏和重新绘制的情况，所以就会出现闪烁的问题，即图像并不稳定。

经过查阅参考资料之后，比较推荐的一个方法就是创建缓冲区，即将原来由 CPU 画显示器读的单缓冲中再申请一段缓冲区域，这样能够加快绘制的速度，同时解决闪烁的问题。网上有很多解决的具体方法，这里我们直接用 EasyX 自带的函数就能是实现解决。

在每次重新绘制的时候调用 BeginBatchDraw();函数在缓冲区中进行绘制，直到全部都绘制完毕之后才调用 FlushBatchDraw();函数将缓冲区的图像显示出来，最后结束缓冲区绘制就可以了。

```
void pic_again(int hour, int min, int sec)
{
    BeginBatchDraw();

    new_sec();
    new_hour();
    new_min();

    FlushBatchDraw();

    EndBatchDraw();
}
```

三、心得体会

这次大作业总体来说并不难，而 EasyX 库也是一种比较简单容易上手的 C++库。其中已经实现了很多有用的函数，我们具体的官方所给的文档，就能够找到对应的函数。

同时老师的准备也很充分，陈宇飞老师与沈坚老师也有所不同。这次实验本就趣味性十足，并不拘泥与一点。甚至我觉得里面中最让我觉得自豪的不是我的而具体的图形优化算法、比如 SDF 算法，最让我喜欢的是钟表样式的设计。感觉到了自己的艺术细胞，虽然可能旁人看起来十分的丑。不过这种趣味性的大作业才能激发学习兴趣。过程中我也在网上看到了一些用 EasyX 库进行游戏设计的项目，最让我深刻的是一款拳王的仿真游戏，有街机 98 拳王的味道了，深刻感受到了创造力十足。

同时另一个值得感慨的是资料的丰富，不仅有往届学长的作品集锦作为艺术源泉，还有具体的图形优化等等算法，可谓是一条龙服务。自己在做的过程中也十分顺利，甚至自信到未写完就国庆跑出去溜达了。非常喜欢这样的大作业。

四、源代码

```
#include <iostream>
#include <iomanip>
#include <graphics.h>
#include <math.h>
#include <conio.h>
#include <time.h>
```

```
using namespace std;
#define PI 3.1415926

struct tm t;
time_t now;

const int circle_x = 400;
const int circle_y = 300;
const int circle_radis = 252;

int startX = circle_x - circle_radis;        //正弦函数起始坐标
int endX = circle_x + circle_radis;          //正弦函数终止坐标

double amplitude = (2*circle_radis) / PI;  // 正弦函数的振幅

int sin_x[2 * circle_radis + 1];           //储存上面的正弦函数与 x 坐标之间的关系
int sin_y[2 * circle_radis + 1];           //储存下面的正弦函数与 x 坐标之间的关系
int r[2 * circle_radis + 1];                //实现渐变色
int b[2 * circle_radis + 1];

void init();           //初始化
void init_sec();       //初始化秒针
void pic_again(int hour, int min, int sec);    //再次绘画时间
void new_hour();       //绘制新的时针
void new_min();        //绘制新的分针
void new_sec();        //绘制新的秒针
void sin_xy();         //因为需要计算正弦函数，所以封装了一个计算过程
void buffer_sin();

//封装一个混合颜色的函数，利用 ALpha 实现抗锯齿
COLORREF mix_color(COLORREF bg, COLORREF color, double alpha);
//画一个没有锯齿的圆
void SDF_circle(Point p, COLORREF color, int r, double line_width);


struct Point {
    int x;
    int y;
};

void init()
{
```

```
    cleardevice();

    setbkcolor(COLORREF(BLACK));

    init_sec();

    new_sec();
    new_hour();
    new_min();

}

void init_sec()
{
    double line_width = 5;

    Point p;
    p.x = circle_x;
    p.y = circle_y;
    SDF_circle(p, WHITE, circle_radis, line_width / 2);

    buffer_sin();
}

void pic_again(int hour, int min, int sec)
{
    BeginBatchDraw();

    new_sec();
    new_hour();
    new_min();

    FlushBatchDraw();

    EndBatchDraw();
}

void sin_xy() {
    for (int x = startX; x <= endX; ++x)
    {
        double radian = (x - startX) * 1.0 / (circle_radis)*PI;  // 角
度转弧度
        double y = circle_y - sin(radian) * amplitude;  // 计算正弦函
```

```
数值的 y 坐标

        sin_x[x - startX] = int(circle_y - sin(radian) * amplitude);
        sin_y[x - startX] = int(circle_y + sin(radian) * amplitude);
        r[x - startX] = (x - startX) / 2;
        b[x - startX] = 255 - (x - startX) / 2;
    }
}

void buffer_sin()
{

    int hour_num = 5;
    TCHAR abc[20];

    for (int x = startX; x <= endX; ++x)
    {
        setcolor(RGB(r[x - startX], 0, b[x - startX]));
        line(x, sin_x[x - startX] - 2, x, sin_x[x - startX] + 2);

        setcolor(WHITE);
        settextstyle(16, 0, _T("SimSun"));
        if ((x - startX) % (circle_radis / 3) == 0 && x != startX && x !=
endX)
        {
            if (x <= circle_x)
            {
                line(x, sin_x[x - startX] - 3, x, sin_x[x - startX] - 12);
                _stprintf_s(abc, _T("%02d"), hour_num);     // 高版本
VC 推荐使用 _stprintf_s 函数
                outtextxy(x - 8, sin_x[x - startX] - 30, abc);
                hour_num += 5;
            }
            else
            {
                line(x, sin_x[x - startX] + 3, x, sin_x[x - startX] + 12);
                _stprintf_s(abc, _T("%02d"), hour_num);     // 高版本
VC 推荐使用 _stprintf_s 函数
                outtextxy(x - 8, sin_x[x - startX] + 15, abc);
                hour_num += 5;
            }

        }
    }
```

```
    hour_num = 35;

    for (int x = endX; x >= startX; --x)
    {

        setcolor(RGB(r[x - startX], 0, b[x - startX]));
        line(x, sin_y[x - startX] - 2, x, sin_y[x - startX] + 2);

        setcolor(WHITE);
        settextstyle(16, 0, _T("SimSun"));
        if ((x - startX) % (circle_radis / 3) == 0 && x != startX && x !=
endX)
        {
            if (x > circle_x)
            {
                line(x, sin_y[x - startX] - 3, x, sin_y[x - startX] - 12);
                _stprintf_s(abc, _T("%02d"), hour_num);     // 高版本
VC 推荐使用 _stprintf_s 函数
                outtextxy(x - 8, sin_y[x - startX] - 30, abc);
                hour_num += 5;
            }
            else
            {
                line(x, sin_y[x - startX] + 3, x, sin_y[x - startX] + 12);
                _stprintf_s(abc, _T("%02d"), hour_num);     // 高版本
VC 推荐使用 _stprintf_s 函数
                outtextxy(x - 8, sin_y[x - startX] + 15, abc);
                hour_num += 5;
            }

        }

    }


}

COLORREF mix_color(COLORREF bg, COLORREF color, double alpha)
{
    COLORREF result;
    int r = int(GetRValue(bg) * (1 - alpha) + GetRValue(color) * alpha);
    int g = int(GetGValue(bg) * (1 - alpha) + GetGValue(color) * alpha);
    int b = int(GetBValue(bg) * (1 - alpha) + GetBValue(color) * alpha);
```

```
    result = RGB(r, g, b);
    return result;
}

void SDF_circle(Point p, COLORREF color, int r, double line_width)
{
    for (int i = p.x - r - 10; i < p.x + r + 10; i++) {      //所画的
x 的变化区域
        for (int j = p.y - r - 10; j < p.y + r + 10; j++) {  //所画的
y 的变化区域
            Point p1 = { i,j };              //其中任一点的坐标
            double d;                        //与圆形的距离

            d = abs(sqrt((pow(p1.x - p.x, 2) + pow(p1.y - p.y, 2))) - r
- line_width);
            //由于圆边有宽度，所以取绝对值进行了一定的处理

            double alpha = 0.5 - (d - line_width);  //计算 Alpha 值
            if (d <= line_width) {                  //圆宽度内涂白
                putpixel(i, j, WHITE);
            }
            if (alpha >= 0 && alpha <= 1) {         //边缘化进行混合颜
色处理
                COLORREF bg = getbkcolor();
                COLORREF result = mix_color(bg, color, alpha);
                putpixel(i, j, result);
            }
        }
    }
}

void new_hour()
{
    clearcircle(circle_x - circle_radis / 2, circle_y, circle_radis / 4);

    settextstyle(30, 0, _T("SimSun"));

    TCHAR s[20];
    _stprintf_s(s, _T("%02d"), t.tm_hour);
    outtextxy(275-15, 300-15, s);

    double line_width = 3;

    Point p1;
```

```
    p1.x = circle_x - circle_radis / 2;
    p1.y = circle_y;

    SDF_circle(p1, WHITE, circle_radis / 4, line_width / 2);

    int hour_pie_x;
    int hour_pie_y;
    double hour_pie_radis = t.tm_hour * 1.0 / 6 * PI + t.tm_min * 1.0 /
360 * PI;

    hour_pie_x = int(circle_x - circle_radis / 2 + circle_radis / 4 * 0.75
* sin(hour_pie_radis));
    hour_pie_y = int(circle_y - circle_radis / 4 * 0.75 *
cos(hour_pie_radis));


    Point p2;
    p2.x = hour_pie_x;
    p2.y = hour_pie_y;

    SDF_circle(p2, WHITE, circle_radis / 16, line_width / 2);

}

void new_min()
{
    clearcircle(circle_x + circle_radis / 2, circle_y, circle_radis / 4);

    settextstyle(30, 0, _T("SimSun"));

    TCHAR str[20];
    _stprintf_s(str, _T("%02d"), t.tm_min);
    outtextxy(525-15, 300-15, str);

    double line_width = 3;
    Point p1;
    p1.x = circle_x + circle_radis / 2;
    p1.y = circle_y;

    setlinestyle(PS_SOLID | PS_ENDCAP_FLAT, 3);

    int min_pie_x;
    int min_pie_y;
```

```
    double min_pie_radis = t.tm_min * 1.0 / 30 * PI + t.tm_sec * 1.0 /
1800 * PI;
    min_pie_x = int(circle_x + circle_radis / 2 + circle_radis / 4 * 0.75
* sin(min_pie_radis));
    min_pie_y = int(circle_y - circle_radis / 4 * 0.75 *
cos(min_pie_radis));

    Point p2;
    p2.x = min_pie_x;
    p2.y = min_pie_y;
    SDF_circle(p1, WHITE, circle_radis / 4, line_width / 2);
    SDF_circle(p2, WHITE, circle_radis / 16, line_width / 2);

}

void new_sec()
{
    setlinestyle(PS_SOLID | PS_ENDCAP_FLAT, 5);
    clearcircle(circle_x, circle_y, circle_radis);

    init_sec();

    double hour_bar_one = startX;
    double hour_bar_two = endX;

    setlinestyle(PS_SOLID | PS_ENDCAP_FLAT, 1);

    if (t.tm_sec < 30) {
        hour_bar_one = startX + (circle_radis * 1.0 / 15) * t.tm_sec;
    }
    else {
        hour_bar_one = endX;
        hour_bar_two = endX - (circle_radis * 1.0 / 15) * (t.tm_sec - 30);
    }

    for (int x = startX; x <= int(hour_bar_one); ++x)
    {
        setcolor(RGB(r[x - startX], 0, b[x - startX]));
        line(x, sin_x[x - startX] + 2, x, circle_y);
    }

    for (int x = endX; x >= int(hour_bar_two); --x)
    {
        setcolor(RGB(r[x - startX], 0, b[x - startX]));
```

```
            line(x, sin_y[x - startX] - 2, x, circle_y);
        }

    }

    int main()
    {
        time(&now);
        localtime_s(&t, &now);  // 获取当地时间


        initgraph(800, 600);// 图形方式初始化
        sin_xy();
        init();           // 自定义图形初始化函数，用于绘制时钟界面

        int cur_hour = t.tm_hour;
        int cur_min = t.tm_min;
        int cur_sec = t.tm_sec;

        while (!_kbhit())    // 无键盘操作时进入循环
        {
            time(&now);
            localtime_s(&t, &now); // 获取当地时间

            if (cur_sec != t.tm_sec)
            {
                pic_again(t.tm_hour, t.tm_min, t.tm_sec);

                cur_hour = t.tm_hour;
                cur_min = t.tm_min;
                cur_sec = t.tm_sec;
            }
        }
        _getch();     // 按任意键准备退出时钟程序
        closegraph();    // 退出图形界面
        return 0;
    }
```