2023

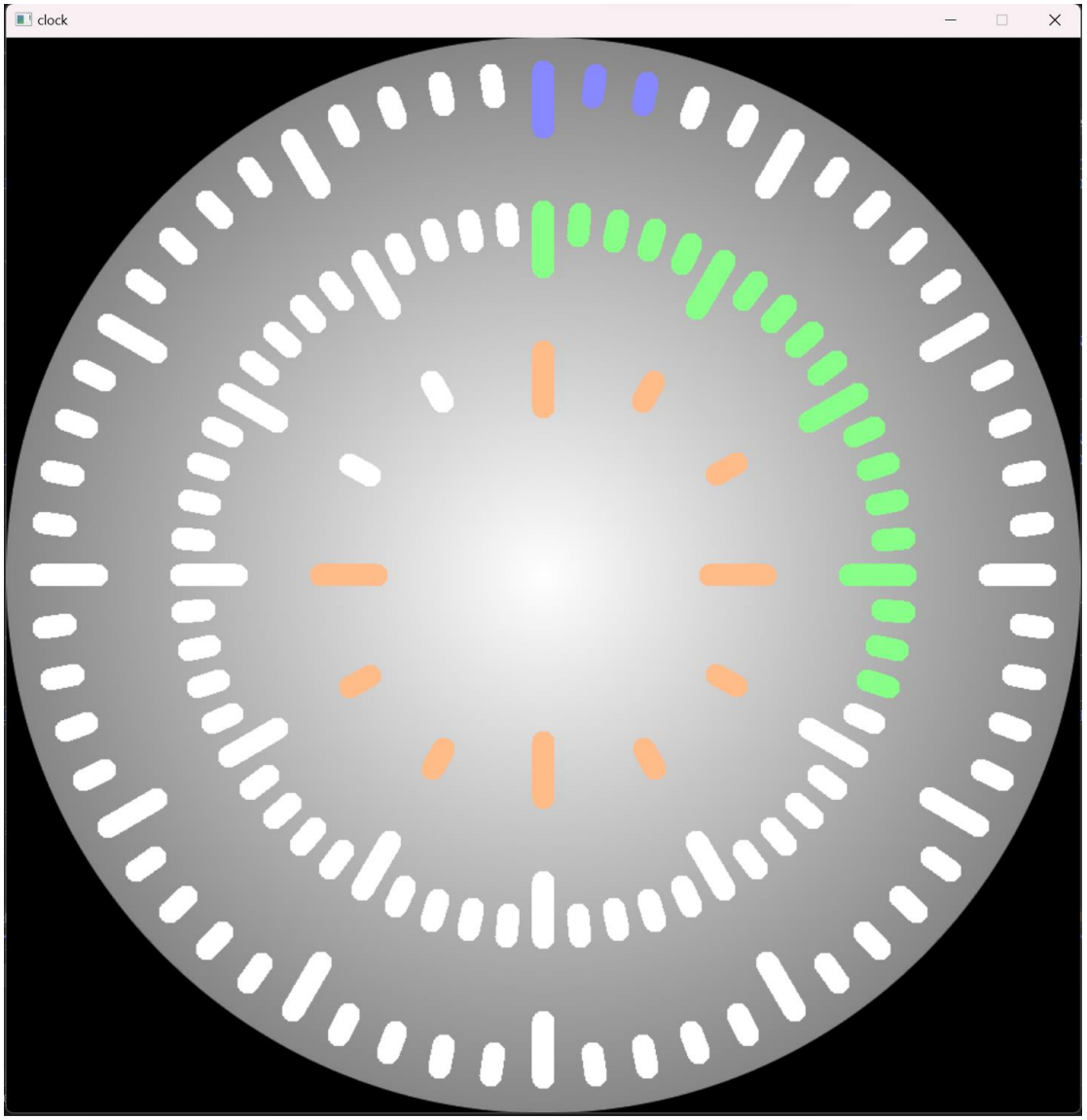
钟表设计实验报告

国 01 2253372 康嘉玮 于 2023.3.23

1. 功能描述

本程序是一款很普通的钟表程序，能够实时显示当前时间的时、分、秒。

本钟表程序利用周围的光标数量显示时间，其中最外层蓝色光标指示秒数，中层绿色光标指示分数，最内层橙色或者红色光标用 12 小时制指示时数，红色为上午（AM），橙色为下午（PM）。例如，下图指示时间用 24 小时制表示为 21:18:02。



2. 设计思路

2.1 时间读取

本程序使用 struct tm 结构体类型数据存放当前时间数据。连续得到两次当前时间数据之后，如果两个秒数不同（也就是说不在同一秒），则调用一次表盘改变函数，详见后续。

2.2 外围表盘

2.2.1 表盘基本设计

表盘设计为圆形，中心点为白色，从中心向两边颜色逐渐加深。实现方式是先设定初始颜色和初始半径，画出最外围的圆，然后使 R、G、B 值逐渐减小，同时逐渐减小半径，画出一个略小的圆，然后再减小 R、G、B 值和半径并画圆，如此循环，直到圆的半径等于 0 为止。必须是先画大圆再逐渐减小圆的半径，否则后画的大圆会覆盖掉先画的小圆，最后只剩最大的一个圆，没有渐变的视觉效果。

2.2.2 表盘抗锯齿

在 2.2.1 中可以看出程序实际上画了远不止一个圆，理论上应该是所有圆都要抗锯齿处理。但实际上对内层圆颜色是渐变的而非突变的，没有明显的锯齿，执行抗锯齿算法效果并不明显而且明显加长表盘绘制所需时间，所以只需要对最外层圆使用抗锯齿算法。

抗锯齿算法采用的是 SDF 和 Alpha Blending。相对直线而言，圆的“有向距离场”简单很多，距离 d 的计算非常简单。然后令 alpha=0.5-d，如果 alpha>1 则置 1，alpha<0 则置 0，最后根据 alpha 计算对应点处的颜色并输出对应颜色的像素点。

在画出第一个圆之前先执行一遍抗锯齿算法，画出为了抗锯齿而在画面上增加的一些像素点，再按照 2.2.1 所述画渐变圆。之所以不在画完最大圆之后画抗锯齿是因为实际成品中最大圆两侧颜色并不相同，且内侧颜色与内层圆相近，因此实际上需要让抗锯齿只对大圆外侧生效，故先画抗锯齿部分再画本体，否则抗锯齿部分会与整体有严重的割裂感。

2.3 指针

2.3.1 初始化

画完表盘之后，在指定位置画一条短粗直线作为光标，除最内圈外每旋转 6° 角画一个光标，内外共画 2 层，内侧作为分针使用，外侧作为秒针使用。最内圈则是每旋转 30° 画一个光标，当作时针使用。为了增强直观性，每隔若干个光标，加长一个，大致类似于传统表盘上 1，2，……，12 这些数字的位置。

理论上直线光标周围最好也要做抗锯齿处理，但我实际上并没有如此处理。原因后续会提到。

2.3.2 指示时间

要指示时间就要先实时读取时间。读取完后，利用一个函数改变当前表盘光标的状态。

先判断是否为初始状态，如果是，就要从头开始为光标涂色，否则只涂色当前秒数对应的光标。如果分钟数改变（此时秒数为 0），就把最外层所有光标涂白，然后涂一格分钟光标，最后再给最外层最顶上一个光标（代表 0 秒）涂色。小时数改变或者上下午切换时同理。

3. 遇到的问题与解决方法

3.1 设计与实际的冲突

最早的设计是渐变色表盘与渐变色传统指针。但实际写出来之后发现，每次指针移动的运算次数太多，导致秒针移动时会出现闪动。原本指针的位置被灰色覆盖，然后指针在下一个位置出现。这个过程正常来说应该肉眼不可见，但由于运算次数太多，过程不能瞬间完成，上述闪烁非常明显，最后被迫放弃这一设计。后来还企图使用过渐变扇形指针等设计，但因相同的原因放弃，最终只能彻底抛弃渐变色指针，改用现在使用的指针。

3.2 抗锯齿算法的时间问题

前面已经提到我本来打算对光标使用抗锯齿的，但发现了两个问题，导致我放弃了实现。

第一个问题是，短粗直线光标周围的颜色并不是纯色，而是渐变色。其实这个问题解决不难，因为渐变色是可以近似看成纯色的。

真正让我放弃做直线抗锯齿的是第二个问题：SDF 算法的速度问题。尽管 SDF 算法远快于 SSAA 算法，但是想让它瞬间解决一百多条直线的锯齿还是不容易的。我从网上得知可以用叫做 AABB 的方式优化 SDF，但以我的水平理解 SDF 就已经比较困难，AABB 更是根本没看懂，导致最终不得不放弃这些直线的抗锯齿。

4. 心得体会

这次绘制钟表的程序设计让我理解了解析几何的重要性。计算机画图，很多时候都要靠解析几何的方式确定某一个物体绘制时的旋转角、长度、位置等信息，因此解析几何在计算机画图当中算是比较重要的知识。比如绘制钟表时，如果不懂解析几何，就不知道如何画出指向正确的表盘指针。

本次大作业也是我第一次接触抗锯齿算法，也让我看到了我的一些不足之处。我觉得应该多看一些书或者资料，尽量学习一些比较常见的图形处理算法。

5. 源代码

5.1 header.h:

```
#pragma once

void init();

void ticktock(int new_hour, int new_minute, int new_second, int old_hour = -1, int old_minute = -1, int old_second = -1);

void AntiAliasingForCircle(int x0, int y0, int radius, int thickness, int fg_color, int bk_color);
```

5.2 main.cpp

```
#include <iostream>
#include <iomanip>
#include <graphics.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#include "header.h"
using namespace std;
#define PI 3.1415927

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

    init();
// 自定义图形初始化函数，用于绘制时钟界面
    ticktock(t.tm_hour, t.tm_min, t.tm_sec);

    while                                                          (!_kbhit())
// 无键盘操作时进入循环
    {
        struct tm t1;
        time_t now1;

        time(&now1);
        localtime_s(&t1,                                               &now1);
// 获取当地时间
```

```
            if (t.tm_sec != t1.tm_sec)
                ticktock(t1.tm_hour, t1.tm_min, t1.tm_sec, t.tm_hour, t.tm_min,
t.tm_sec);
            t = t1;
            now = now1;
        }
        char ch = _getch();
// 按任意键准备退出时钟程序
        closegraph();
// 退出图形界面
        return 0;
    }
```

5.3 init.cpp

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

    using namespace std;

    #define PI 3.1415927
    #define MY_ORANGE 0x88BBFF
    #define MY_BLUE 0xFF8888
    #define MY_GREEN 0x88FF88
    #define MY_RED 0x8888FF

    #define RADIUS 480
```

```
#define RADIUS_OF_HR 200
#define RADIUS_OF_MIN 325
#define RADIUS_OF_SEC 450
#define LENGTH_OF_POINTER 20
#define LENGTH_OF_SPECIAL 50
#define THICKNESS 20



void init()
{
    initgraph(2 * RADIUS + 1, 2 * RADIUS + 1);

    AntiAliasingForCircle(RADIUS, RADIUS, RADIUS, 1, (0xFF - RADIUS / 4) *
(256 * 256 + 256 + 1), 0);                    //外层圆抗锯齿

    int radius;
    for (radius = RADIUS; radius >= 0; radius -= 4)
                                                //画渐变圆
    {
        int rgb = 0xFF - radius / 4;
        setfillcolor(rgb * 256 * 256 + rgb * 256 + rgb);
        setlinecolor(rgb * 256 * 256 + rgb * 256 + rgb);
        solidcircle(RADIUS, RADIUS, radius);
    }

    setlinecolor(WHITE);

    int i;
    for (i = 0; i < 60; i++)
                                                //和下一个 for 循环都是画空白光
标
```

```
        {
            setlinestyle(0, THICKNESS);
            line(RADIUS + (int)(RADIUS_OF_SEC * sin(i * PI / 30)), RADIUS - (int)(RADIUS_OF_SEC * cos(i * PI / 30)), RADIUS + (int)((RADIUS_OF_SEC - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 30)), RADIUS - (int)((RADIUS_OF_SEC - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i * PI / 30)));

            setlinestyle(0, THICKNESS);
            line(RADIUS + (int)(RADIUS_OF_MIN * sin(i * PI / 30)), RADIUS - (int)(RADIUS_OF_MIN * cos(i * PI / 30)), RADIUS + (int)((RADIUS_OF_MIN - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 30)), RADIUS - (int)((RADIUS_OF_MIN - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i * PI / 30)));
        }
        for (i = 0; i < 12; i++)
        {
            setlinestyle(0, THICKNESS);
            line(RADIUS + (int)(RADIUS_OF_HR * sin(i * PI / 6)), RADIUS - (int)(RADIUS_OF_HR * cos(i * PI / 6)), RADIUS + (int)((RADIUS_OF_HR - (i % 3 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 6)), RADIUS - (int)((RADIUS_OF_HR - (i % 3 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i * PI / 6)));
        }
    }

    void ticktock(int new_hour, int new_minute, int new_second, int old_hour, int old_minute, int old_second)
    {
        int i;

        setlinestyle(0, THICKNESS);
        if (!new_second)
                                                    //新的分钟，此时秒数显示清零
```

```
        {
            setlinecolor(WHITE);
            for (i = 0; i < 60; i++)
                line(RADIUS + (int)(RADIUS_OF_SEC * sin(i * PI / 30)), RADIUS - (int)(RADIUS_OF_SEC * cos(i * PI / 30)), RADIUS + (int)((RADIUS_OF_SEC - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 30)), RADIUS - (int)((RADIUS_OF_SEC - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i * PI / 30)));
        }
        setlinecolor(MY_BLUE);
        if (old_second == -1)
                                                    //if 语句初始化秒针，下面一行是画新的一秒
            for (i = 0; i < new_second; i++)
                line(RADIUS + (int)(RADIUS_OF_SEC * sin(i * PI / 30)), RADIUS - (int)(RADIUS_OF_SEC * cos(i * PI / 30)), RADIUS + (int)((RADIUS_OF_SEC - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 30)), RADIUS - (int)((RADIUS_OF_SEC - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i * PI / 30)));
        line(RADIUS + (int)(RADIUS_OF_SEC * sin(new_second * PI / 30)), RADIUS - (int)(RADIUS_OF_SEC * cos(new_second * PI / 30)), RADIUS + (int)((RADIUS_OF_SEC - (new_second % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(new_second * PI / 30)), RADIUS - (int)((RADIUS_OF_SEC - (new_second % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(new_second * PI / 30)));



        setlinestyle(0, THICKNESS);
                                                    //新的小时，分数清零
        if (!new_minute)
        {
            setlinecolor(WHITE);
            for (i = 0; i < 60; i++)
                line(RADIUS + (int)(RADIUS_OF_MIN * sin(i * PI / 30)), RADIUS - (int)(RADIUS_OF_MIN * cos(i * PI / 30)), RADIUS + (int)((RADIUS_OF_MIN - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 30)), RADIUS - (int)((RADIUS_OF_MIN - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i
```

```
* PI / 30)));
        }
        setlinecolor(MY_GREEN);
        if (old_minute == -1)
                                                    //if 语句初始化分针，下面一行是
画新的一分钟
            for (i = 0; i < new_minute; i++)
                line(RADIUS + (int)(RADIUS_OF_MIN * sin(i * PI / 30)), RADIUS -
(int)(RADIUS_OF_MIN * cos(i * PI / 30)), RADIUS + (int)((RADIUS_OF_MIN - (i %
5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 30)), RADIUS -
(int)((RADIUS_OF_MIN - (i % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i
* PI / 30)));
        line(RADIUS + (int)(RADIUS_OF_MIN * sin(new_minute * PI / 30)), RADIUS
- (int)(RADIUS_OF_MIN * cos(new_minute * PI / 30)), RADIUS + (int)((RADIUS_OF_MIN
- (new_minute % 5 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(new_minute *
PI / 30)), RADIUS - (int)((RADIUS_OF_MIN - (new_minute % 5 ? LENGTH_OF_POINTER :
LENGTH_OF_SPECIAL)) * cos(new_minute * PI / 30)));


        setlinestyle(0, THICKNESS);
        if (!(new_hour % 12))
                                                    //AM/PM 切换，时数清零
        {
            setlinecolor(WHITE);
            for (i = 0; i < 12; i++)
                line(RADIUS + (int)(RADIUS_OF_HR * sin(i * PI / 6)), RADIUS -
(int)(RADIUS_OF_HR * cos(i * PI / 6)), RADIUS + (int)((RADIUS_OF_HR - (i % 3 ?
LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 6)), RADIUS -
(int)((RADIUS_OF_HR - (i % 3 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i
* PI / 6)));
        }
        setlinecolor(new_hour >= 12 ? MY_ORANGE : MY_RED);
        if (old_hour == -1)
                                                    //if 语句初始化时针，下面一行是
画新的一小时
            for (i = 0; i < new_hour % 12; i++)
```

```
            line(RADIUS + (int)(RADIUS_OF_HR * sin(i * PI / 6)), RADIUS -
(int)(RADIUS_OF_HR * cos(i * PI / 6)), RADIUS + (int)((RADIUS_OF_HR - (i % 3 ?
LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(i * PI / 6)), RADIUS -
(int)((RADIUS_OF_HR - (i % 3 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * cos(i
* PI / 6)));

        line(RADIUS + (int)(RADIUS_OF_HR * sin(new_hour * PI / 6)), RADIUS -
(int)(RADIUS_OF_HR * cos(new_hour * PI / 6)), RADIUS + (int)((RADIUS_OF_HR -
(new_hour % 3 ? LENGTH_OF_POINTER : LENGTH_OF_SPECIAL)) * sin(new_hour * PI /
6)), RADIUS - (int)((RADIUS_OF_HR - (new_hour % 3 ? LENGTH_OF_POINTER :
LENGTH_OF_SPECIAL)) * cos(new_hour * PI / 6)));

    }


    void AntiAliasingForCircle(int x0, int y0, int radius, int thickness, int
fg_color, int bk_color)

    {

        int x, y;

        for (y = 0; y <= RADIUS * 2 + 1; y++)

            for (x = 0; x <= RADIUS * 2 + 1; x++)

            {

                double distance = fabs(sqrt((x0 - x) * (x0 - x) + (y0 - y) * (y0
- y)) - radius) - thickness / 2.0;


                double alpha = 0.5 - distance;

                if (alpha > 1)

                    alpha = 1;

                else if (alpha < 0)

                    alpha = 0;


                if (alpha)

                    putpixel(x, y, RGB(alpha * GetRValue(fg_color) + (1 - alpha)
* GetRValue(bk_color), alpha * GetGValue(fg_color) + (1 - alpha) *
GetGValue(bk_color), alpha * GetBValue(fg_color) + (1 - alpha) *
GetBValue(bk_color)));

            }
```

```
}
```