钟表设计实验报告

国豪 06 班 2251079 隋建政

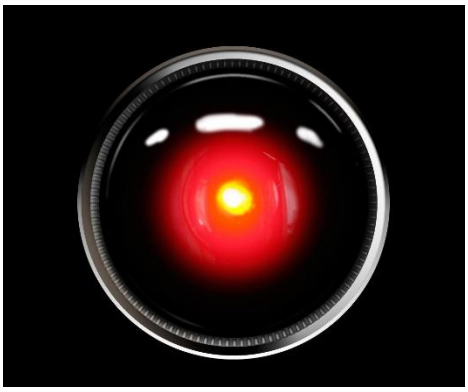
2022.3.22

1. 设计思路与功能描述

1.1 表盘设计

1.1.1 整体思路

表盘的设计思路来源于科幻电影中独眼机器人的造型，比如《2001 太空漫游》中的 HAL9000、《机器人总动员》中的 AUTO 自动驾驶再到如今《流浪地球》中的 MOSS。随着时代变迁，独眼人工智能这一经典形象带给人冷峻与绝对理性的感受并未发生过变化，这与我心目中的时间带给人的感受是一致的，且人工智能圆形的外形也很容易修改成表盘的模样。



《2001 太空漫游》中的 HAL9000（图片来源：Hal 9000 1920x1080 Wallpapers - Wallpaper Cave）

我的表盘设计

1.1.2 背景设计

背景希望做出金属表面的质感，所以采用一个由深灰色到浅灰色的渐变营造金属反光的质感，该渐变背景的实现主要是通过很多条不同颜色的斜向线段构成，实现起来并不复杂。

1.1.3 盘面设计

盘面设计非常简单，主要是通过大小不同的圆形做出不同圈层的效果。比较麻烦的是一蓝一红两个不规则圈层的实现，主要是在内部规则圆环的基础上添加两个半径较大的扇形，再一起利用内部黑色圆形进行遮挡就形成了这种形状，同时设置了 Sleep()函数，并按周期改变相应扇形的起始角度和终止角度以达到不断旋转的效果，该动画被设置为 30FPS 而不与指针一样一秒一帧是由于下述我希望达成的效果。

两列以相反方向运行的不规则圆圈让我联想到科幻电影中经常出现的太空列车在太空

中安静驶过的情形，也像是表示我的人工智能钟表正在不断运行的反馈效果，这一动画效果很符合我希望的成品效果。

最中心盘面采取外圈暗中心亮的颜色构成，为的是在二维平面构图中营造立体的感觉，最外圈添加蔓延至更外面的渐变，希望营造光晕的效果，实现与背景的渐变效果大同小异，只不过对象由直线变为圆圈。在中心的外部圈层我添加了刻度盘，目的是方便确认时间，也让整体更有钟表的感觉。

1.2 指针设计

1.2.1 时针

由于时针普遍不需要精准的读取而只需要知道大概出于哪个时刻即可，所以我在外圈并没有添加小刻度而只有 12 个大刻度用以划分区间。时针的造型也是用的一个色彩鲜艳的黄色球形。时针以一小时为周期做圆运动，缓慢移动的圆形时针让我联想到绕恒星做周期性运动的行星，被两列“太空列车”夹在中间缓慢移动的“行星”也很符合整体的意境。主要是通过周期性改变圆心位置，并用黑色背景不断覆盖原先位置进行实现的。

1.2.2 分针

我希望分针可以准确的指示时间，所以利用了直线进行绘制，但又不希望太复杂的指针破坏整体简约画面的美感，所以仅由一条黑色线段代替。效果还是令人满意的，红色表盘内部的黑色细线让我联想到蛇的眼睛，这一效果似乎也与我希望营造的冷峻感不谋而合。

1.2.3 秒针

秒针我依然是利用球型指针，但鉴于秒针移动速度比较快，所以我在小球后部添加了甩尾造型，以营造一种速度感。同时加上尾流的小球让人联想到快速移动的彗星，不同层次不同速度的指针与装饰让画面看起来不显得单调。小球本体的实现与时针一致，尾流是通过与小球相同周期运动的渐变效果扇形实现的，再在中间添加遮挡用的圆形达到尾流的效果。

去掉中间圆形遮挡后的效果

1.3 抗锯齿的实现

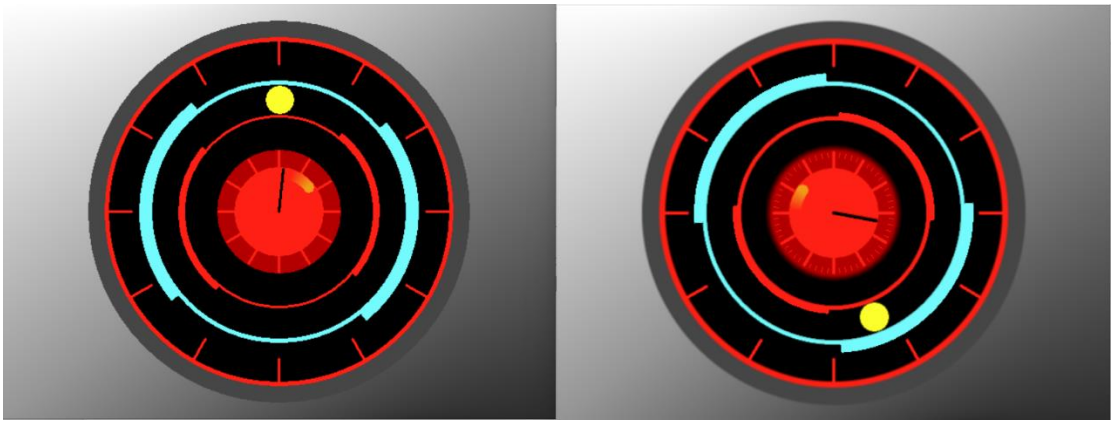
1.3.1 圆形的抗锯齿

利用 SDF 与 Alpha Blending 算法相结合实现圆形的抗锯齿。圆形的 SDF 计算比较简单，直接利用点到圆心的距离再减去半径即可得到。为减小运算时长，只遍历包围整个圆形且边长为直径加 10 的正方形所包围的像素点。通过 Alpha Blending 算法计算该像素点的颜色并利用 putpixel()函数为其上色。为优化程序，圆上的像素点不进行 Alpha Blending 的计算而是直接通过 fillcircle()函数处理。

圆形抗锯齿的效果（左图为抗锯齿前，右图为抗锯齿后）

1.3.1 直线的抗锯齿

Alpha Blending 的部分与圆形抗锯齿几乎无差异，主要的差异在于 SDF 的计算，可将空间相对于线段的部分分为三块，在两端超出线段两头垂线所夹部分的像素点，其距离为该点到端点的距离，而中间部分的距离为垂线长度。通过这样的计算使直线的 SDF 类似于胶囊的形状。



直线抗锯齿的效果（注意中心部分黑色指针的抗锯齿效果，左图为抗锯齿前，右图为抗锯齿后）

1.4 延时效果的实现

我希望可以实现两种不同的延时效果，其中指针的运动以一秒为一帧即可，而外部不规则圈层我则希望可以达到 30 帧以一个较为流畅的动画进行演示。因此在整体的

While 循环中，我使用了 Sleep 函数使其每 1/30s 就会进行一次循环，但仅使用 Sleep 函数进行的时间截断并不准确，由于图像绘制所需要的时间不断叠加，图像延迟时间会不断加长。在 While 循环中我加入条件判断语句，调用 clock 函数，每当显示时间推进一秒时对应指针的角度便会相应的增加，以达到指针一秒一帧的效果。

2. 遇到的问题及解决方案

2.1 屏幕闪烁的问题

在初期进行试验时，进行简单的图形动画就会出现屏幕闪烁的问题，该问题应该是由于图像绘制的时间延迟造成的。解决方案是利用 BeginBatchDraw()函数和 EndBatchDraw()函数，利用批量绘图功能可以完美的解决该问题。

2.2 动画延迟的问题

仅利用 Sleep 函数不能很精准的控制时间间隔，我想该问题是由于图像处理的时长不断叠加，表现在动画中就是延迟时长不断扩大。因此我引入了一些条件判断语句，利用 clock（）函数精准的获取当前程序运行的时长，同时也解决了背景动画与指针动画帧数不统一的问题。

2.3 抗锯齿的问题

在写直线抗锯齿的代码时，初期试验一直不成功，表现在图像的问题是仅有端点两头有模糊处理而线段中间部分没有任何效果，因此我猜想为中间部分的距离计算出现问题。最后经过很多次的修改与核实，问题出现在其中一个计算向量乘积的函数返回类型为 int，这样会导致之后的整除运算将数据截断造成很大误差，将该函数的返回值改为 double 型就解决了问题。

3. 心得体会

本次实验给我留下印象最深的便是写直线抗锯齿的过程，在好几个夜晚我苦思冥想，对着写好的函数埋头苦算，始终想不通究竟是哪里出了问题导致最后效果不成功。尽管早早猜出是距离计算出现错误但在反复验算之后也不能发现问题的根本，最后竟是这小小的函数返回类型让我吃尽苦头。在编写函数时，我认为通过 int 型坐标相加减得到的 int 型向量，其做相乘运算的返回值也肯定是 int 型并无问题，但却忽略了在计算完向量乘积后还要将其进行除法运算，如果是 int 型的整除那么问题就太大了。在检查过程中，我一步步的检查变量的取值情况才发现这个致命的问题。不要想当然的写下函数返回类型，我想是我在这次实验中得到的最大教训。

同时在使用 EasyX 进行图像绘制过程中，我也颇有体会，一是要学会使用工具，在阅读说明的过程中就能发现很多内容就是专门用来解决自己正面临难题的。二是多多利用函数，在不断更替颜色的过程中如果在主函数中频繁切换就显得太冗杂太繁琐，不如自己另写函数将图像颜色也作为参量，在调用时就更得心应手。三是多利用宏定义来定义颜色，在颜色库中寻得心仪的颜色后最好将其 RGB 值宏定义为相应的容易理解的模样，这样方便修改也更方便调用多种不同的颜色进行绘图。

最后是本次实验的遗憾的也就是图中扇形的抗锯齿处理没有完成，在网络上寻找的资料中仅能找到关于坐标轴对称扇形的 SDF 计算方法，我自己的思路是按照与圆形相同的思路进行 SDF 计算，但仅遍历扇形两边延长所组成矩形内的像素点，但由于我的扇形是不断旋转，我并没有想到什么好的办法进行选取，而且鉴于我发现似乎逐帧图形的抗锯齿效果明显不如静止图形（不知道是什么原因），于是最终放弃了这个比较浪费时间但可能收益较少的想法。

4. 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
#define MY_RED RGB(255,32,21)
#define MY_ORANGE RGB(240, 157, 27)
#define MY_BLUE RGB(115, 251, 253)
#define MY_YELLOW RGB(253, 255, 45)
#define MY_GREY RGB(70,70,70)
#define MY_DARKRED RGB(179,0,0)

const int height = 480;
const int width = 640;
const int FPS = 30;
struct Point
{
    int x;
    int y;
};

double length(Point a, Point b)/*计算两点之间的长度*/
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

double multiplication(Point a, Point b)/*计算两向量的乘积*/
{
    return a.x * b.x + a.y * b.y;
}

int sdf_color(int bg, int color, double alpha)/*根据alpha blending算法计算像素点的颜色*/
```

```
{
    return RGB(GetRValue(bg) * (1 - alpha) + GetRValue(color) * alpha, GetGValue(bg) * (1 - alpha) +
GetGValue(color) * alpha, GetBValue(bg) * (1 - alpha) + GetBValue(color) * alpha);
}

double sdf_of_circle(Point o, Point p, int r)/*计算圆形的SDF*/
{
    return length(o, p) - r;
}

void SDF_Circle(Point o, int color, int r)/*圆形的SDF+alpha blending算法抗锯齿*/
{
    Point p;
    double d, alpha;
    int X, Y;
    for (X = o.x - r - 5; X < o.x + r + 5; X++) {
        for (Y = o.y - r - 5; Y < o.y + r + 5; Y++) {
            p = { X,Y };
            d = sdf_of_circle(o, p, r);
            alpha = 1 - d / 3;/*d/3时视觉效果更棱角分明但旋转的圆圈抗锯齿效果不佳，d/4时旋转圆圈抗锯齿稍好但依
然不佳，且整体画面显得很朦胧*/
            if (alpha >= 0 && alpha <= 1)
                putpixel(X, Y, sdf_color(getpixel(X, Y), color, alpha));
            else if (d < 0)
                continue;/*内部直接通过fillcircle()函数绘制以减少计算时间*/
        }
    }
    setfillcolor(color);
    setlinecolor(color);
    fillcircle(o.x, o.y, r);
}

double sdf_of_line(Point p, Point a, Point b)/*计算直线的SDF*/
{
    Point ap = { p.x - a.x,p.y - a.y };
    Point ab = { b.x - a.x,b.y - a.y };
    double h = (multiplication(ap, ab) / multiplication(ab, ab));
    if (h < 0)/*在两端点之外的区域采取直接使用端点与采样点之间的距离作为SDF*/
        h = 0;
    else if (h > 1)
        h = 1;
    double X = a.x + h * (b.x - a.x), Y = a.y + h * (b.y - a.y);
    return length({ int(X),int(Y) }, p);
}
```

```
void SDF_Line(Point a, Point b, int color)/*圆形的SDF+alpha blending算法抗锯齿*/
{
    Point p;
    double d, alpha;
    int X, Y;
    for (X = min(a.x, b.x) - 10; X <= max(a.x, b.x) + 10; X++) {
        for (Y = min(a.y, b.y) - 10; Y <= max(a.y, b.y) + 10; Y++) {
            p = { X,Y };
            d = sdf_of_line(p, a, b);
            alpha = 1 - d / 3;
            if (alpha >= 0 && alpha <= 1)
                putpixel(X, Y, sdf_color(getpixel(X, Y), color, alpha));
        }
    }
    setlinecolor(color);
    line(a.x, a.y, b.x, b.y);
}

void background()/*渐变效果的背景*/
{
    double x, y;

    for (x = width / 2, y = -height / 2; x >= -width / 2 && y <= height / 2; x -= (width / height), y += 1) {
        setlinecolor(RGB(int(150 - (-height / 2 - y) * 105 / height),
            int(150 - (-height / 2 - y) * 105 / height), int(150 - (-height / 2 - y) * 105 / height)));
        line(-width / 2, int(y), int(x), height / 2);
    }
    for (x = -width / 2, y = height / 2; x <= width / 2 && y >= -height; x += (width / height), y -= 1) {
        setlinecolor(RGB(int(150 - (height / 2 - y) * 105 / height),
            int(150 - (height / 2 - y) * 105 / height), int(150 - (height / 2 - y) * 105 / height)));
        line(int(x), -height / 2, width / 2, int(y));
    }
    /*深灰色到白色的渐变*/
}

void DrawCircle(int x, int y, int r, int color)/*画圆*/
{
    SDF_Circle({ x,y }, color, r);
}

void DrawPointer(int x1, int x2, int color)/*绘制指针*/
{
    for (double theta = 0; theta < 2 * PI; theta += (PI / 6)) {
```

```
        SDF_Line({ int(x1 * sin(theta)), int(x1 * cos(theta)) }, { int(x2 * sin(theta)), int(x2 *
cos(theta)) }, color);
    }/*每PI/6一个指针*/
}

void DrawLittlePointer(int x1, int x2, int color)
{
    setlinestyle(PS_SOLID | PS_ENDCAP_ROUND, 1);
    setlinecolor(color);
    for (double theta = 0; theta < 2 * PI; theta += (PI / 30)) {
        line(int(x1 * sin(theta)), int(x1 * cos(theta)), int(x2 * sin(theta)), int(x2 * cos(theta)));
    }
    setlinestyle(PS_SOLID | PS_ENDCAP_ROUND, 3);
}

void Rotating_Ring(int x, int y, int r1, int r2, int color1, double angle)/*绘制旋转的异形圆环*/
{
    DrawCircle(x, y, r1, color1);
    fillpie(r2, r2, -r2, -r2, angle, angle + PI / 2);
    fillpie(r2, r2, -r2, -r2, angle + PI, angle + PI * 3 / 2);
}

void Ball_Tail(double theta)/*带有渐变色的小球尾流*/
{
    double angle = theta - PI / 6;
    for (; angle <= theta; angle += PI / 60000) {
        setlinecolor(RGB(int(255 - 15 * (angle - theta + PI / 6) / (PI / 6)), int(32 + 125 * (angle - theta +
PI / 6) / (PI / 6)),
            int(21 + 6 * (angle - theta + PI / 6) / (PI / 6))));
        line(0, 0, int(50 * sin(angle)), int(50 * cos(angle)));
    }
}

void Minute_Pointer(int x, int y, int r, double angle, int color)
{
    SDF_Line({ 0, 0 }, { int(r * sin(angle)), int(r * cos(angle)) }, color);
}

// 精确延时函数(可以精确到 1ms，精度 ±1ms)
void HpSleep(int ms)
{
    static clock_t oldclock = clock();   // 静态变量，记录上一次 tick
    oldclock += ms * CLOCKS_PER_SEC / 1000;  // 更新 tick
    if (clock() > oldclock)      // 如果已经超时，无需延时
```

```
        oldclock = clock();
    else
        while (clock() < oldclock)     // 延时
            Sleep(1);        // 释放 CPU 控制权，降低 CPU 占用率
}
int getTime()
{
    return clock() / CLOCKS_PER_SEC;
}
int main()
{
    struct tm t;
    time_t now;
    time(&now);
    localtime_s(&t, &now);
    int lastTime = 0;
    float x = 0, y = 0, * p_x = &x, * p_y = &y;
    double angle_blue = -PI / 2, angle_red = 0;
    double angle_yellow = ((t.tm_sec + t.tm_min * 60 + (t.tm_hour % 12 ) * 3600) * 2 * PI / (12 * 3600)),
        angle_orange = (t.tm_sec ) * 2 * PI / 60, angle_pointer = (t.tm_sec + t.tm_min * 60 ) * 2 * PI / 3600;
    initgraph(width, height);
    setorigin(width / 2, height / 2);
    getaspectratio(p_x, p_y);
    setaspectratio(*p_x, -(*p_y));
    BeginBatchDraw();
    background();
    DrawCircle(0, 0, 220, MY_GREY);
    DrawCircle(0, 0, 200, MY_RED);
    DrawCircle(0, 0, 193, BLACK);
    setlinestyle(PS_SOLID | PS_ENDCAP_ROUND, 3);
    DrawPointer(170, 197, MY_RED);
    EndBatchDraw();
    while (!_kbhit())
    {
        int now = getTime();
        BeginBatchDraw();
        DrawCircle(0, 0, 160, BLACK);
        Rotating_Ring(0, 0, 150, 160, MY_BLUE, angle_blue);
        angle_blue += (PI / 300);
        DrawCircle(0, 0, 145, BLACK);
        DrawCircle(int(130 * sin(angle_yellow)), int(130 * cos(angle_yellow)), 15, MY_YELLOW);
        if (now - lastTime > 0) {
            angle_yellow += 2 * PI / (60 * 60 * 12);
        }
```

```
    Rotating_Ring(0, 0, 110, 115, MY_RED, angle_red);
    angle_red -= (PI / 300);
    DrawCircle(0, 0, 105, BLACK);
    for (int r = 78; r >= 70; r--) {
        DrawCircle(0, 0, r, (78 - r) * 179 / 8);
    }
    DrawCircle(0, 0, 70, MY_DARKRED);
    DrawCircle(0, 0, 50, MY_RED);
    DrawPointer(50, 70, MY_RED);
    DrawLittlePointer(65, 70, MY_RED);
    DrawCircle(int(45 * sin(angle_orange)), int(45 * cos(angle_orange)), 5, MY_ORANGE);
    Ball_Tail(angle_orange);
    if (now - lastTime > 0) {
        angle_orange += 2 * PI / (60);
    }
    DrawCircle(0, 0, 36, MY_RED);
    Minute_Pointer(0, 0, 50, angle_pointer, BLACK);
    if (now - lastTime > 0) {
        angle_pointer += 2 * PI / (60 * 60);
        lastTime = now;
    }
    EndBatchDraw();
    HpSleep(int(1000 / 30));
}
_getch();
closegraph();
return 0;
}
```