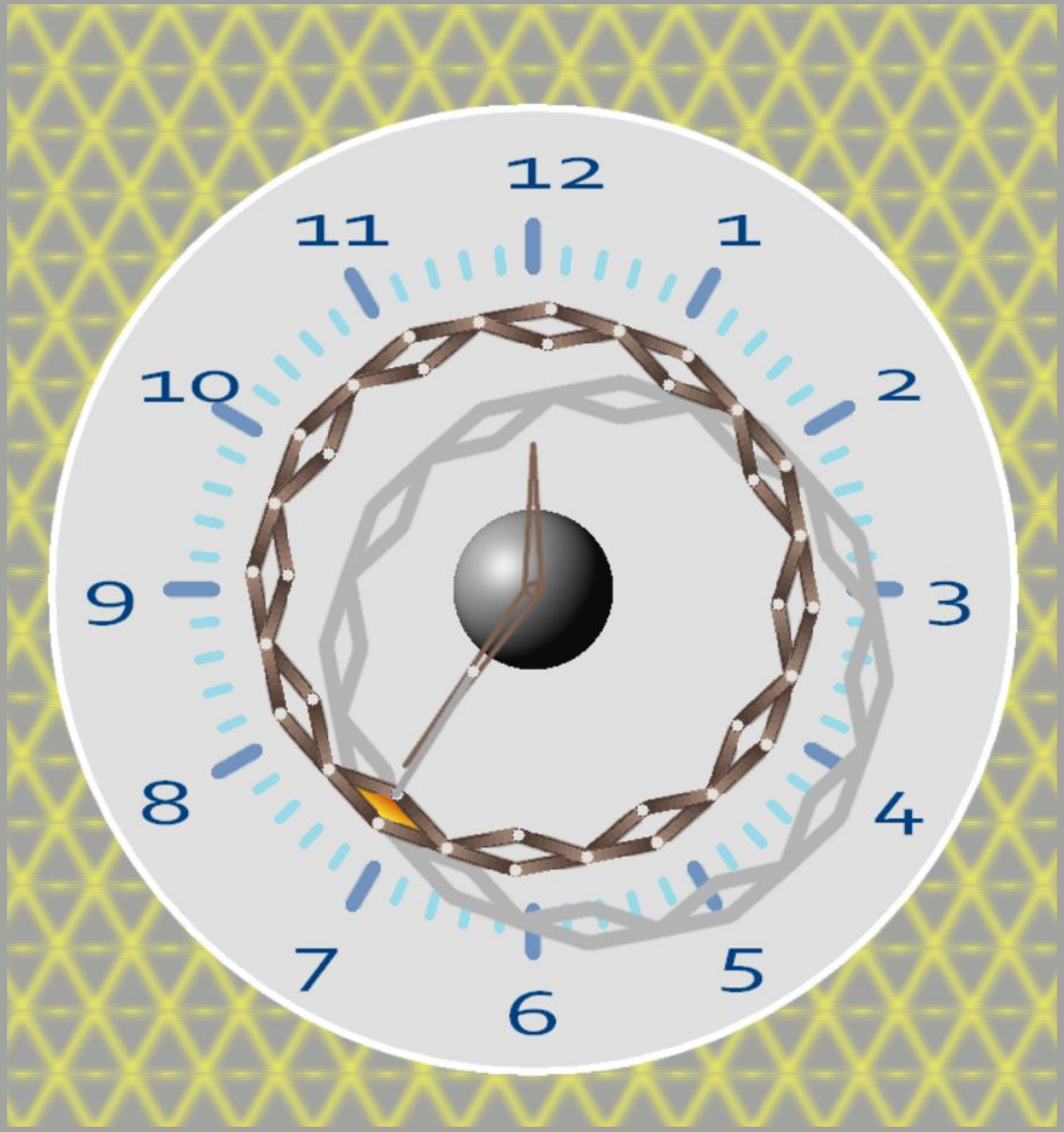
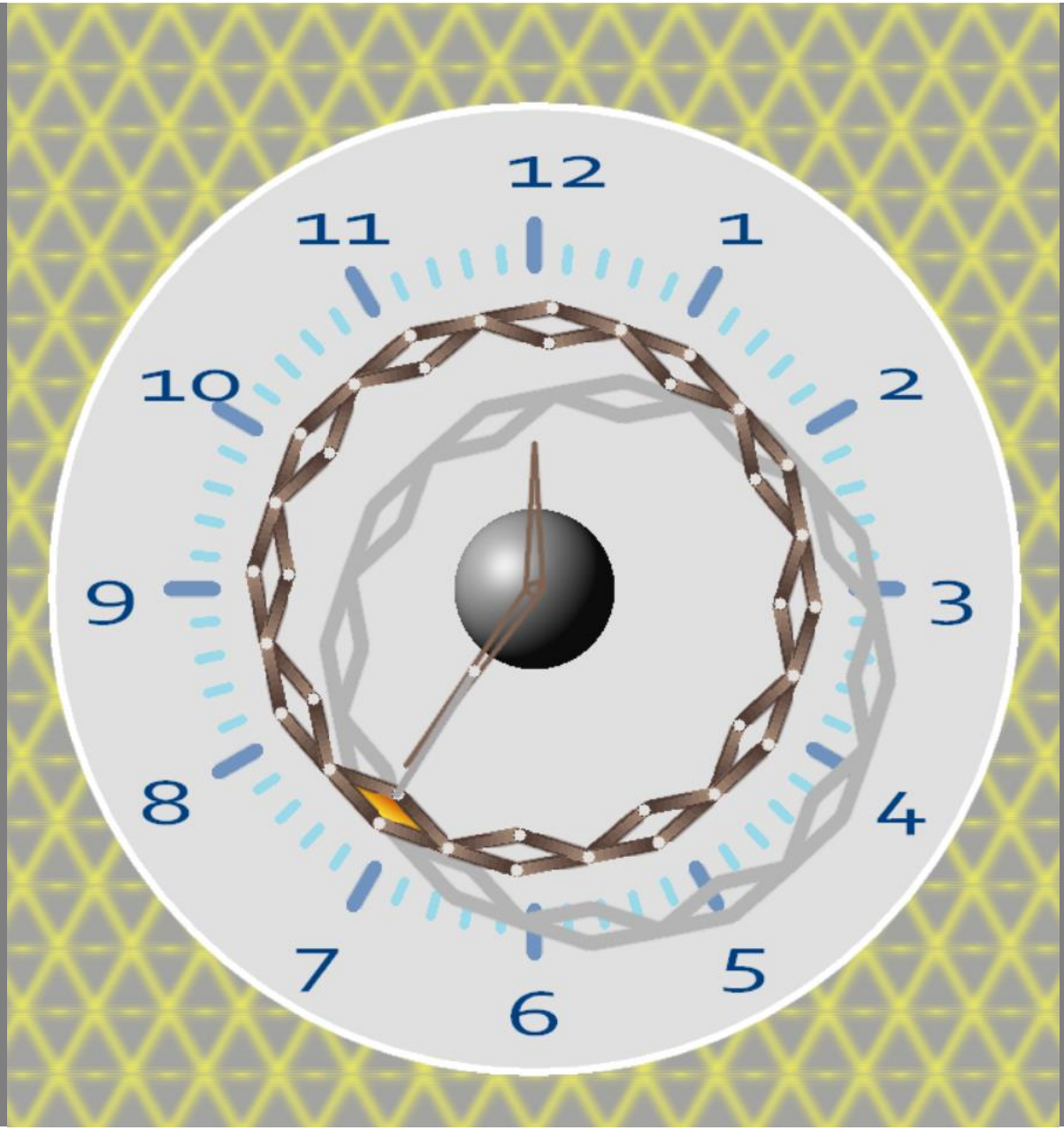
Design a Clock

2050289 朱昀玮



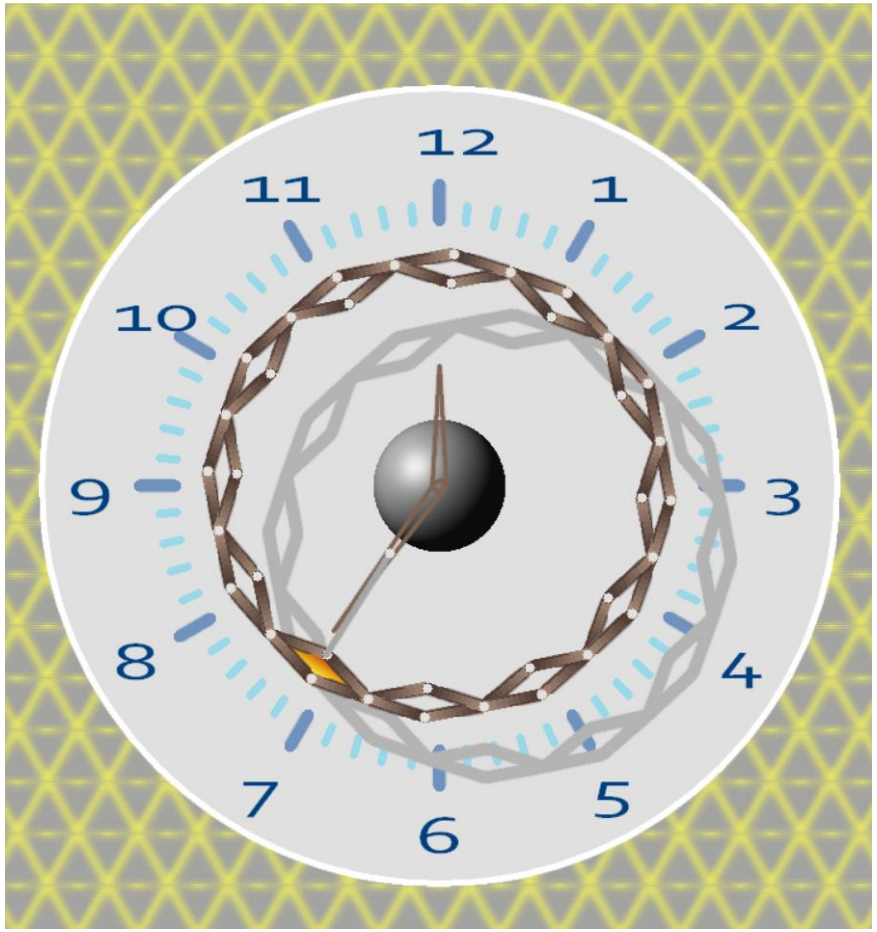
- 设计思路
- 技术原理
- 性能优化*



设计思路

设计思路

- 机械结构 → 机械运动的木框连杆结构
- 立体感 → 阴影、3 维球
- 好看 → 渐变色、一些颜色搭配
- 是个时钟 → 表盘、指针



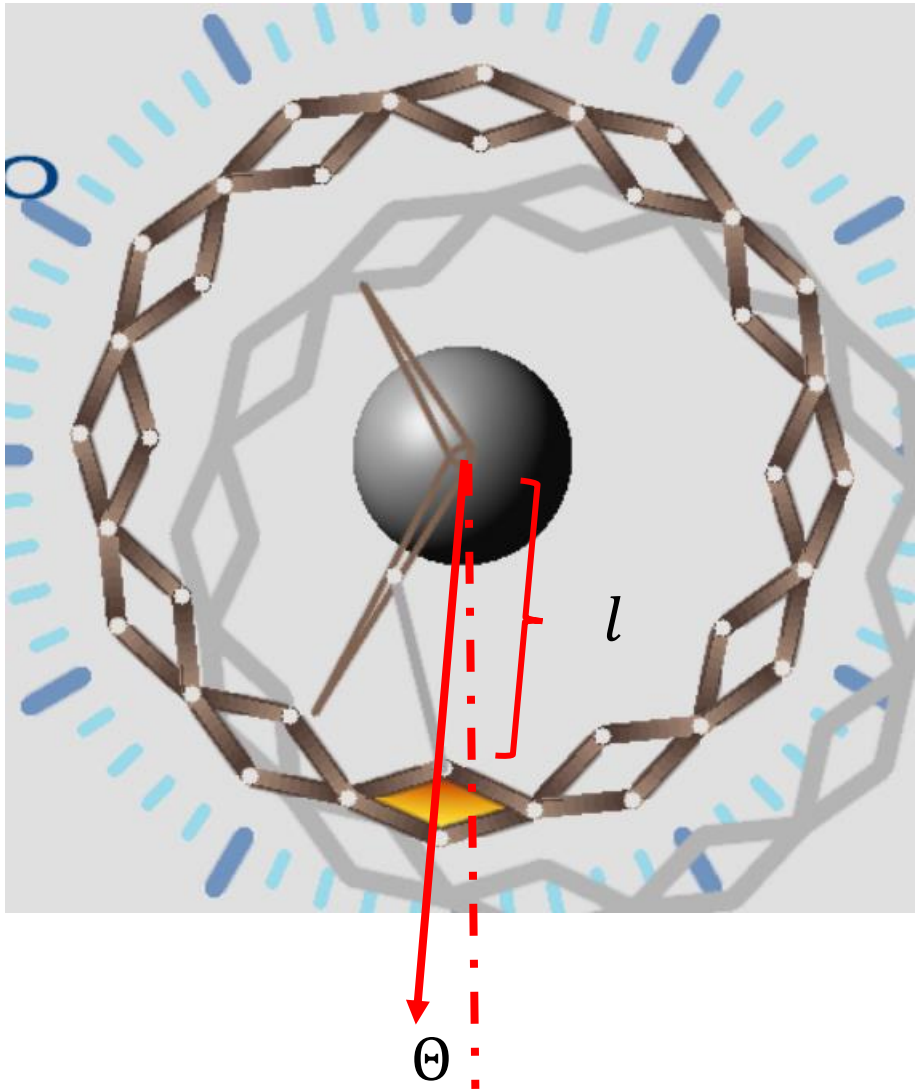
表盘

- 几何约束

$\theta(t)$: 指针的角度随着时间的变化

$l(t)$: 表盘上某个固定点到表盘中心的位置

通过这两个约束可以确定四边形的形状



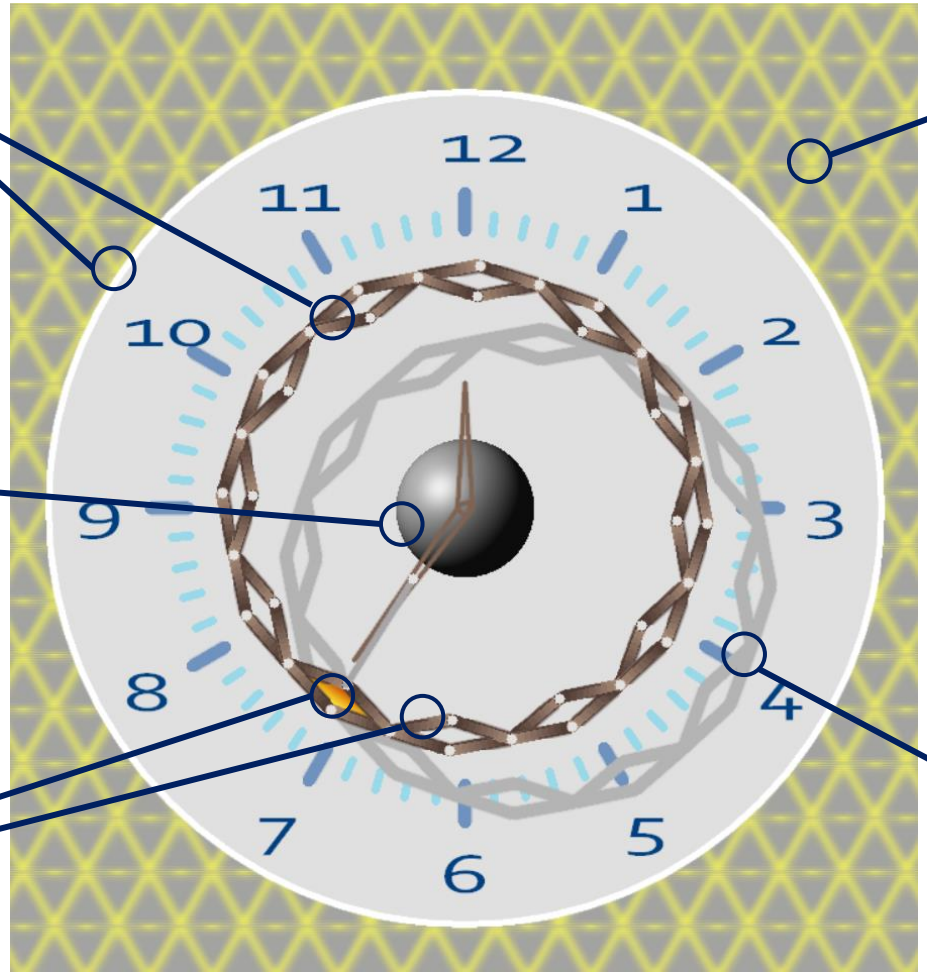
主要技术

主要技术

抗锯齿

Blinn-Phong光照模型的球

渐变色

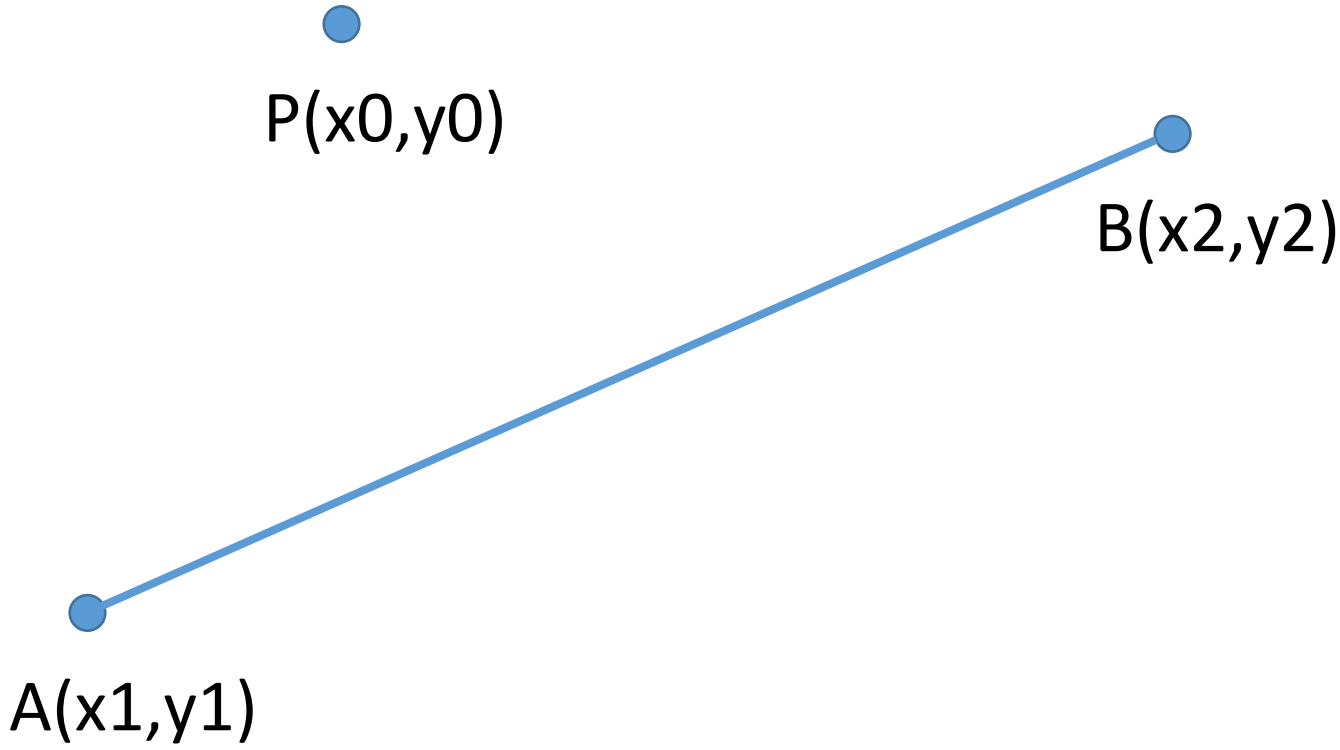


高斯模糊

阴影

抗锯齿

问题：此时给出像素点p(x0,y0)，求出该像素点的颜色

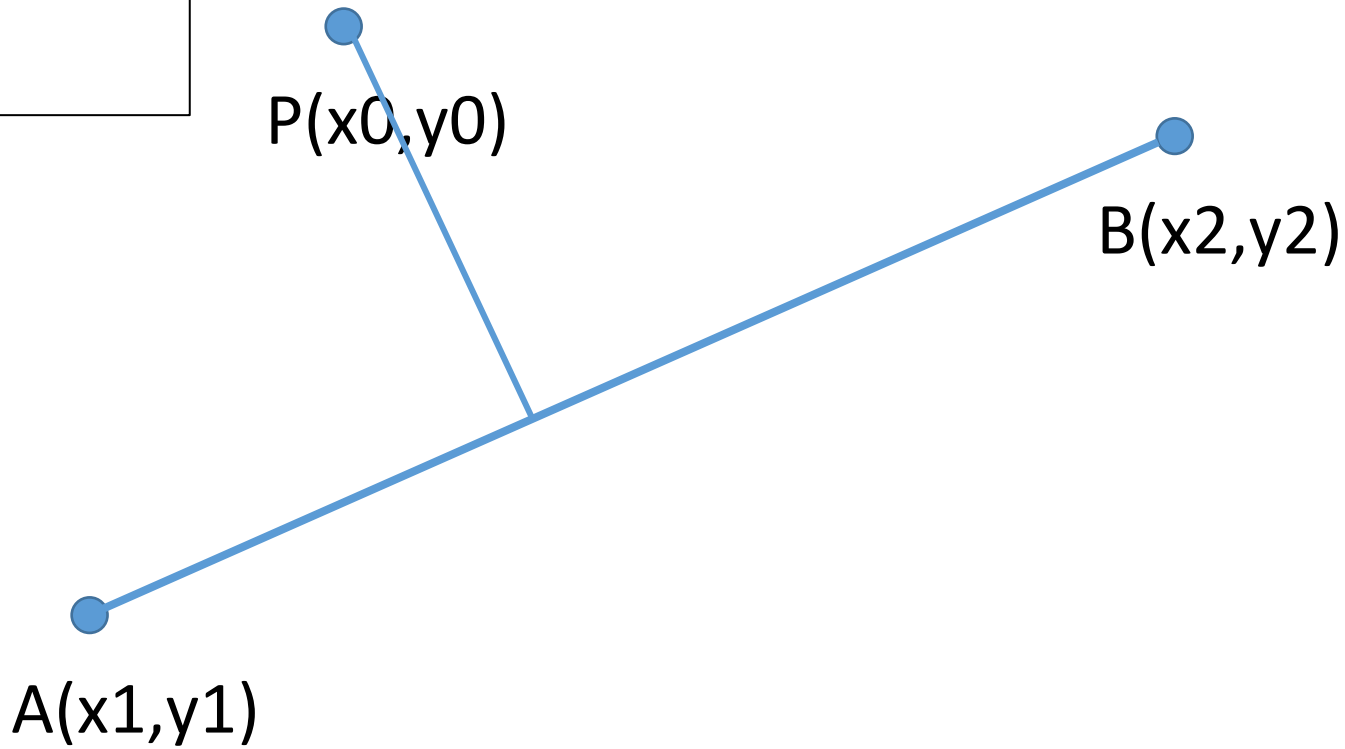


图源：wws分享ppt

抗锯齿

考虑：

- 我们需要让他越靠近边缘时颜色越淡
- 离开直线区域则颜色为空
- 在直线内部则颜色为纯色



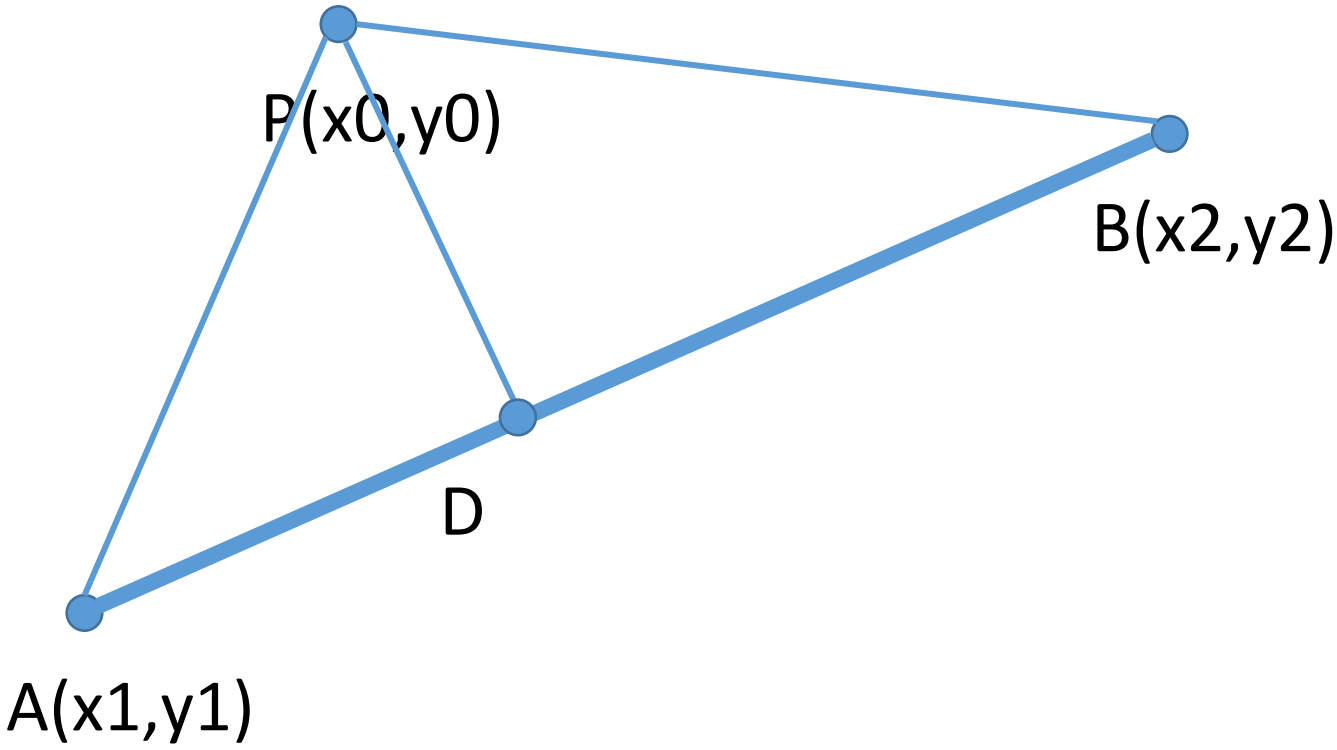
图源：wws分享ppt

抗锯齿

直线的半宽度 r，点到直线距离 d，AD长度为 h

$$h = \frac{\overrightarrow{PA} \cdot \overrightarrow{BA}}{\overrightarrow{BA}^2} \quad (0 \le h \le 1)$$

$$d = \sqrt{\overrightarrow{PA}^2 - (\overrightarrow{BA} \cdot h)^2} - r$$



抗锯齿

距离 ⟶ 颜色

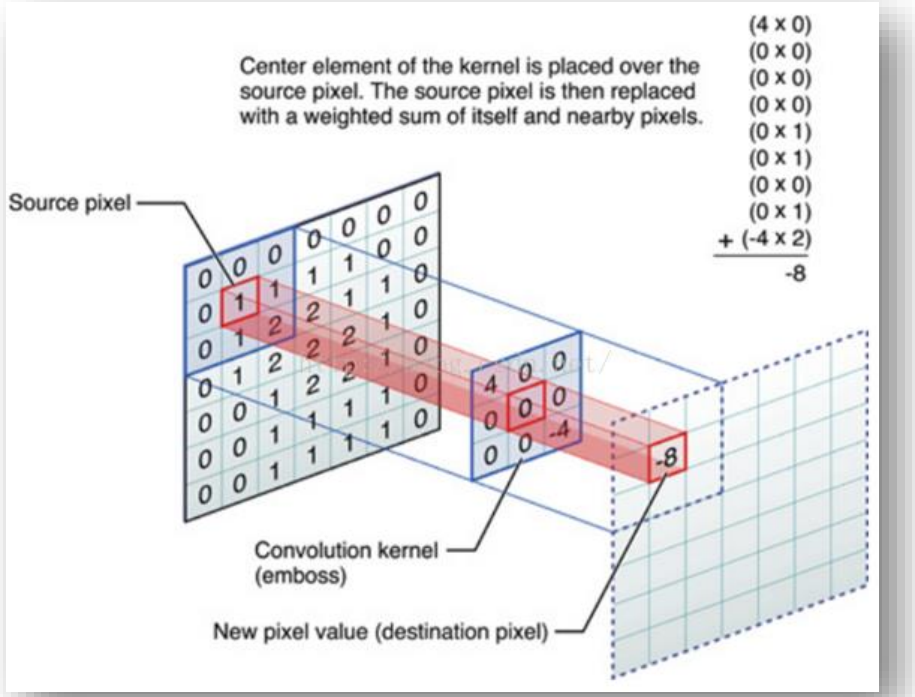
前景色 + 后景色 => 最终颜色

$$alpha = 0.5 - d \ (0 \leq alpha \leq 1)$$

$$final_{color} = origin_{color} * (1 - alpha) + target_{color} * alpha$$

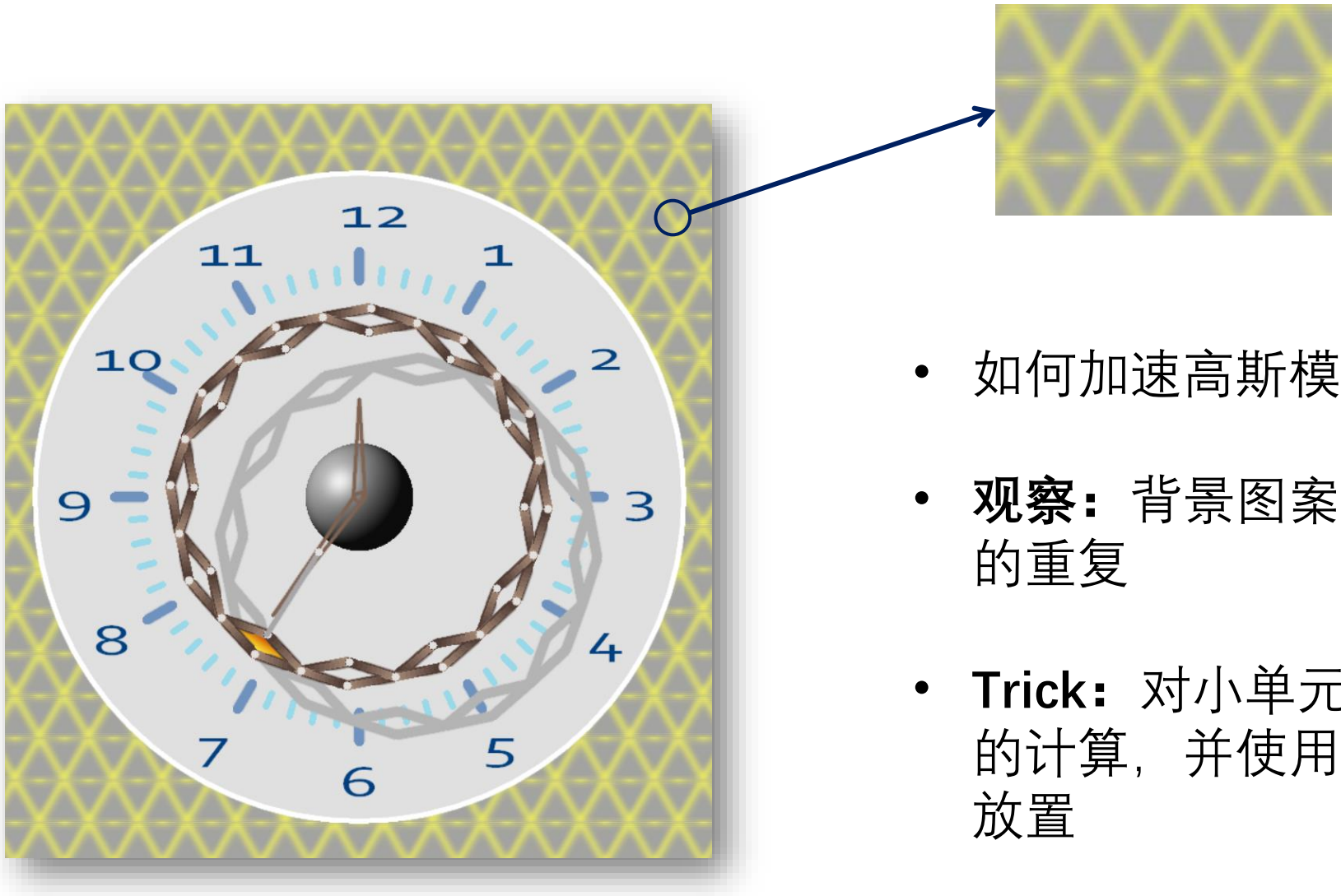
高斯模糊

- 后处理



0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

高斯模糊

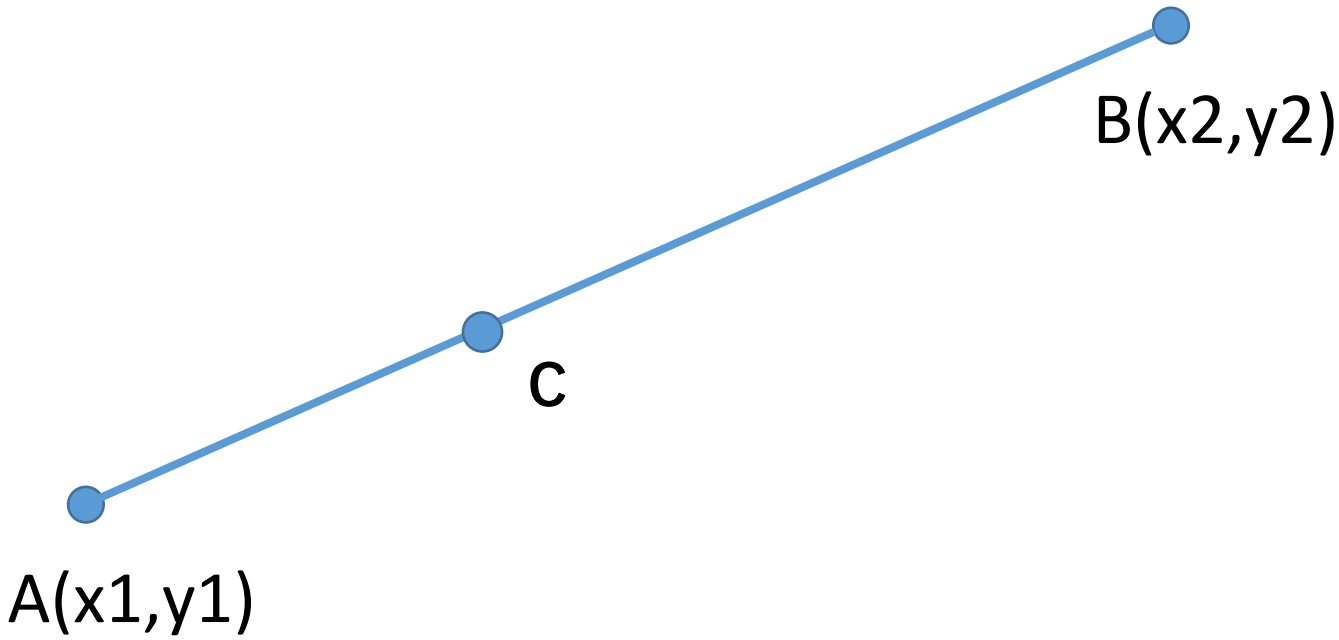


- 如何加速高斯模糊计算？
- **观察：**背景图案是两种小单元的重复
- **Trick：**对小单元进行高斯模糊的计算，并使用 putimage 重复放置

渐变色

“插值”

$$Color_C = \frac{CB}{AB} \cdot Color_A + \frac{AC}{CB} \cdot Color_B$$



性 能 优 化

如何优化?

Premature Optimization is the root of all evil!

把精力放在优化**费时**的代码上

怎么知道是否费时？测!

SDF 加速

Brute-force : 先画线，再抗锯齿

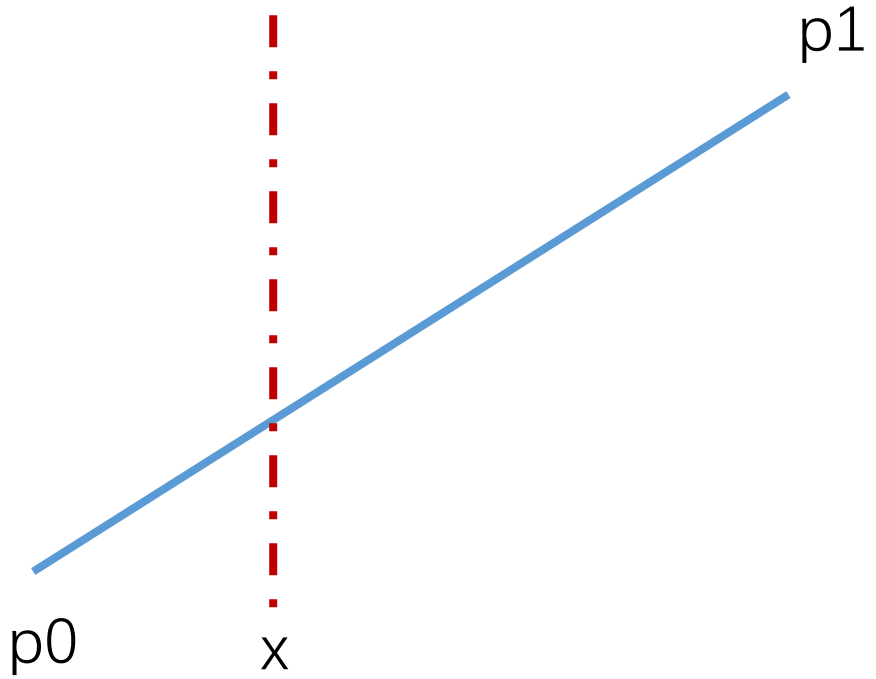
为什么不在绘制时就实现抗锯齿?

- Problem1：怎么画直线?
- Problem2：怎么抗锯齿?

- Ans1:

根据src和dst得到直线斜率k
则，如果当前绘制的点为(x,y)
那么下一个点一定为(x+1,y+k)
那么对于宽度为 w 的直线怎么办?

1. 先画线
2. 找出直线的包围框
3. 对包围框中的**每个点**进行test



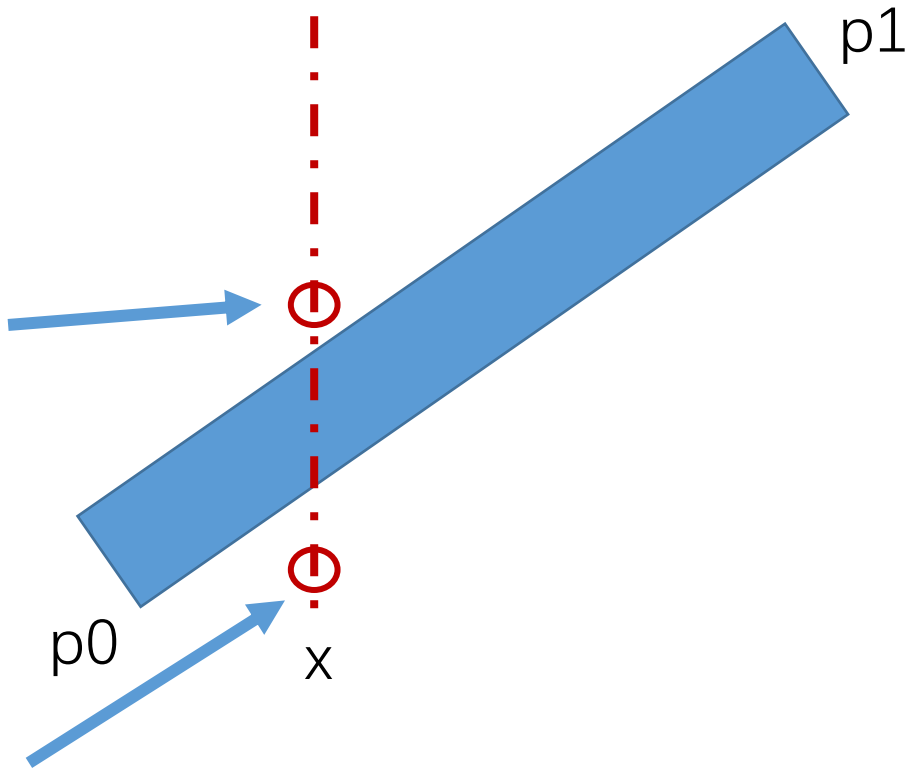
为什么不在绘制时就实现抗锯齿？

- Problem1：怎么画直线？（√）
- Problem2：怎么抗锯齿？

- Ans2:

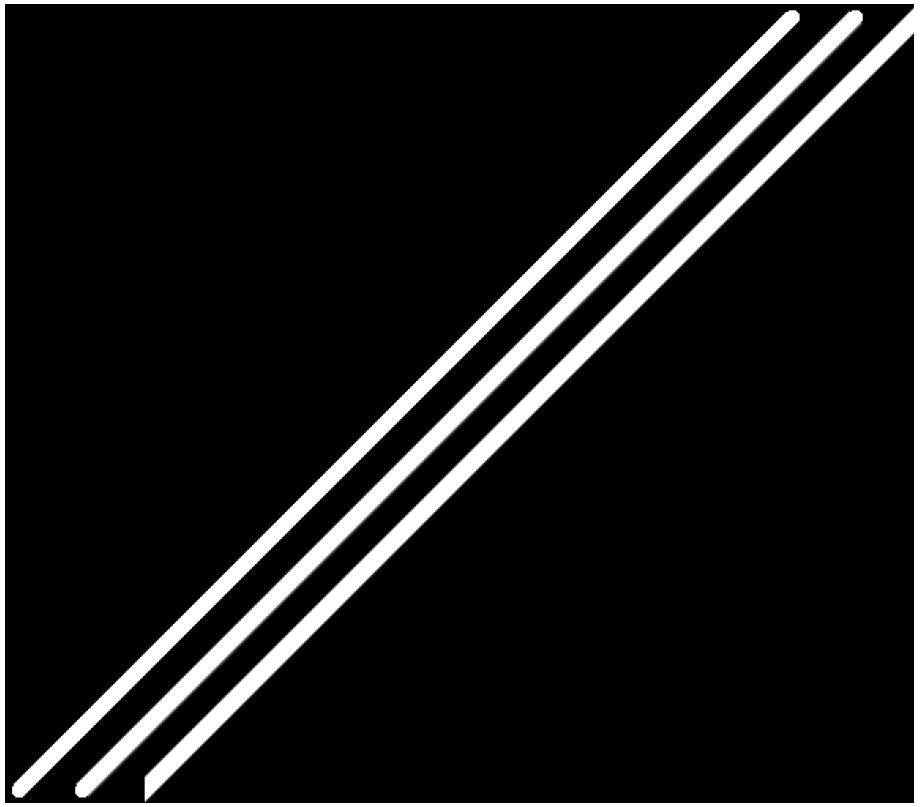
一个简单的思路

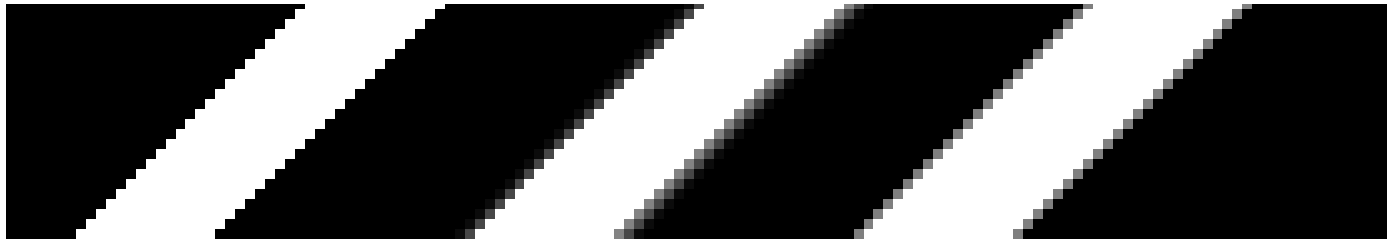
假设在 x = x0 时，绘制的点位于 y0-y1 之间，那么只要 test y0 和 y1 附近的点即可



SDF 加速

Talk is cheap, show me the code!





```
1th Line 0.086ms
2th DDA Line 15.711ms
3th SDF line 113.505ms
请按任意键继续. . .
```

Line：line 函数绘制
DDA：dda算法 + 抗锯齿
SDF line： brute force

SDF 加速

```
1th Line 0.086ms
2th DDA Line 15.711ms
3th SDF line 113.505ms
请按任意键继续. . .
```

- 为什么调用 Line 函数只要0.086 ms?
- 当前算法中似乎已经处理了最少量的像素。

- 破案了，是调用了很多次 putpixel

- Why?
- RefreshWindow 慢!

```
void line( int x1, int y1, int x2, int y2
{
    ...
    // Draw the line
    LineTo( hDC, x2, y2 );
    ...
    // rect : line 占用的区域
    RefreshWindow( &rect );
}
```

```
void putpixel( int x, int y, int color
{
    ...
    SetPixelV( hDC, x, y, color );
    ...
    RECT rect = { x, y, x+1, y+1 };
    RefreshWindow( &rect );
}
```

SDF 加速

- 如何避免使用 putpixel？
- 使用一个缓冲区，定位完成所有的像素，再统一放到屏幕上，只要一次Refresh
- 如何获得缓冲区？

```
1th Line 0.085ms
2th DDA Line 16.706ms
3th DDA without putpixel 1.075ms
请按任意键继续. . .
```

```
// 创建一个img指针
IMAGE* img = new IMAGE(x_len,y_len);
// 获得缓冲区，存在img中
getimage(img, upper_left.x, upper_left.y, x_len, y_len);
// 从img中获得buffer
DWORD* buffer = GetImageBuffer(img);
```

SDF 加速

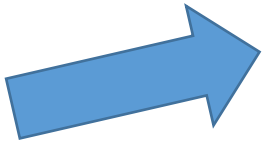
How far do we go?

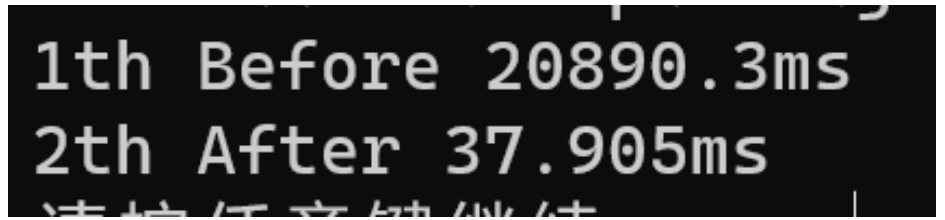
113 ms -> 1ms

高斯模糊加速

Brute-Force: 对每个点进行卷积，9次运算

- 第一步：摆脱 putpixel 和 getpixel





- 第二步：观察，高斯卷积核具有对称性

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

高斯模糊加速

- 第二步：观察，高斯卷积核具有对称性

- 思路：先做 行卷积 1x3，再做列卷积 1x3
  - before: 每个像素 8 次浮点加法，9次浮点乘法
  - After：每个像素 4 次浮点加法，6 次浮点乘法

```
1th Old algorithm 26.362ms
2th New Algorithm 20.699ms
```

21% boost

- 为什么没有预想的（~30%）那么多？Cache miss

高斯模糊加速

What's next?

- 开发并行性
  - SIMD：SSE指令集，AVX指令集
  - 多线程/多进程并行
  - GPU并行

- 减少 Cache Miss，让尽可能多的数据能够直接从缓存中拿取

1 帧不卡，2帧流畅，3帧电竞，为什么要追求速度？

FOR FUN!