逻辑教育
Logic education

Hello CC

OpenGL ES 主题[8]

视觉班—OpenGL ES 实现滤镜处理[1]

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

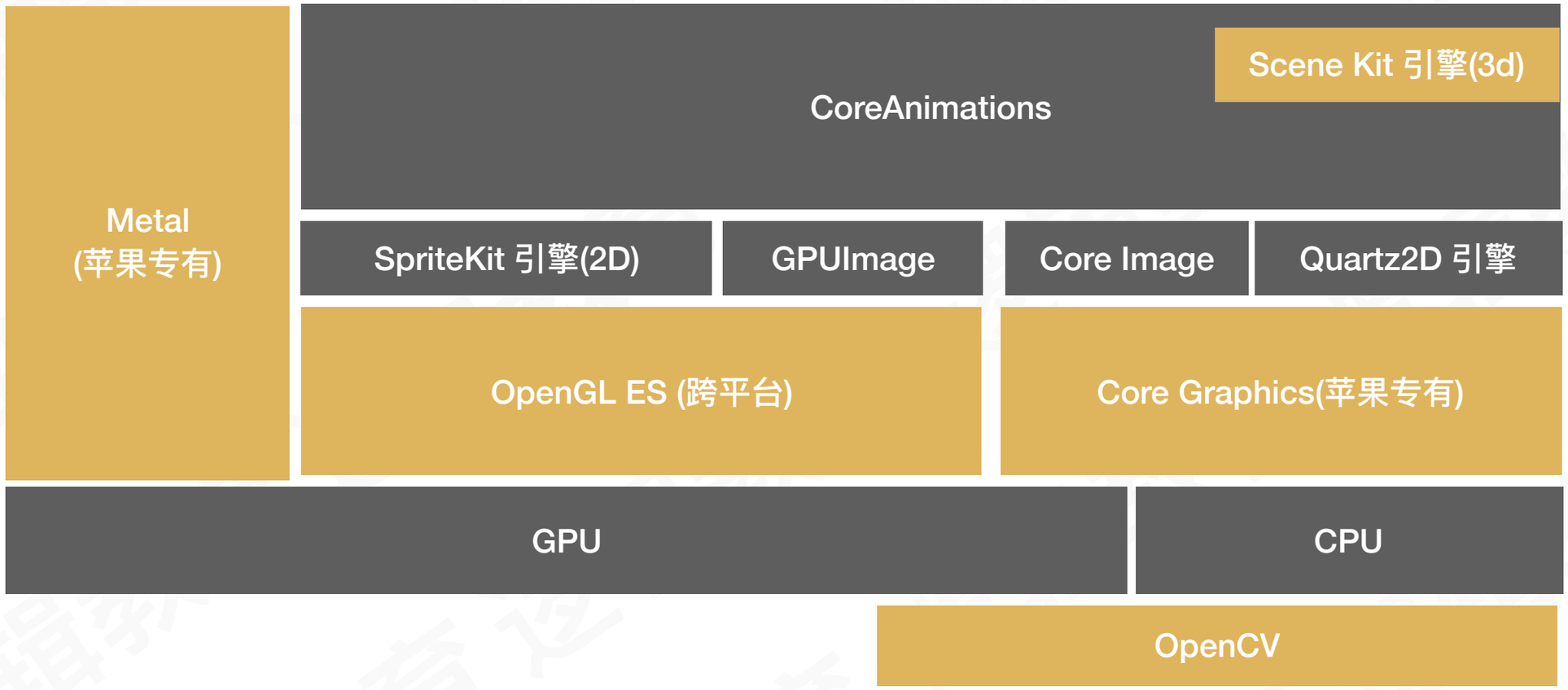
课程目标

1. iOS 图形处理框架快速了解
2. 图片显示原理
3. 滤镜实现原理
4. 实现图片加载
5. 实现图片滤镜处理

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

iOS 与图形图像处理框架



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

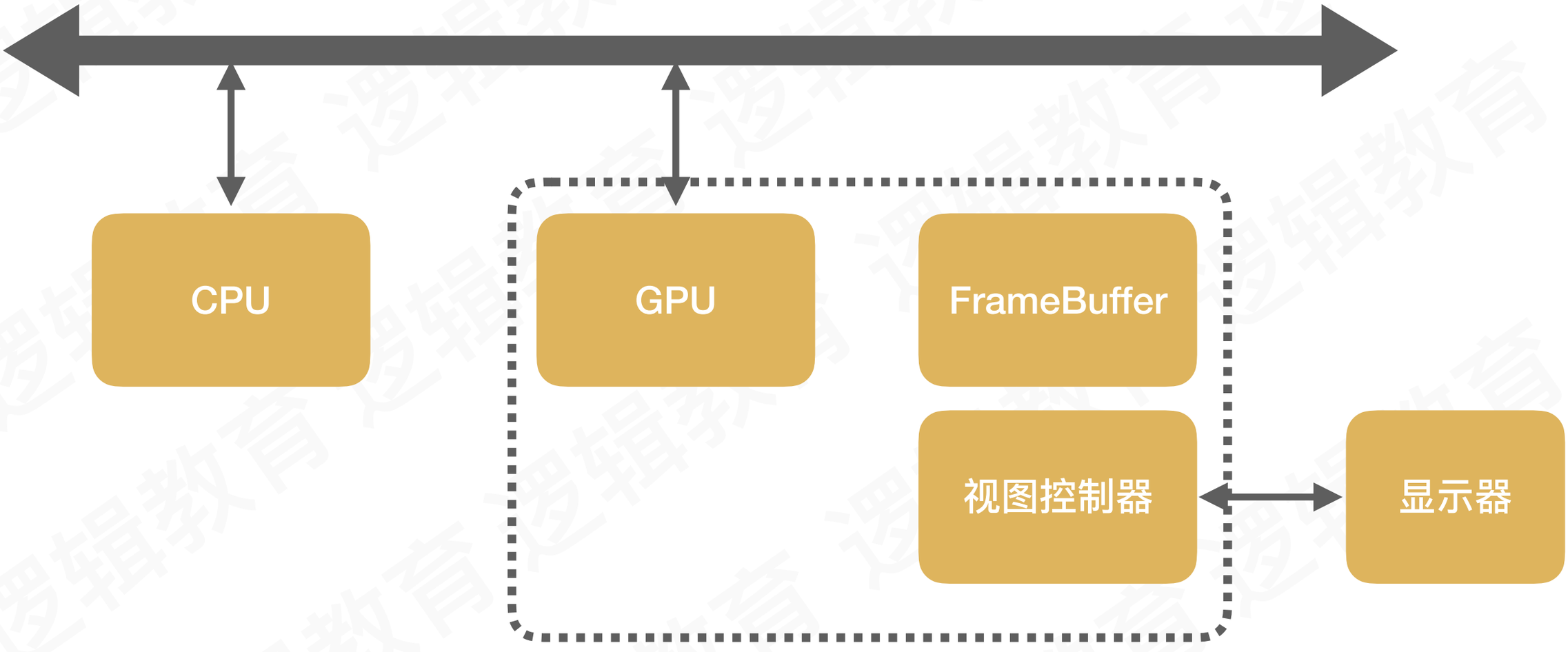
图片显示原理

图片显示的原理

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

图片显示原理



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

图片显示原理

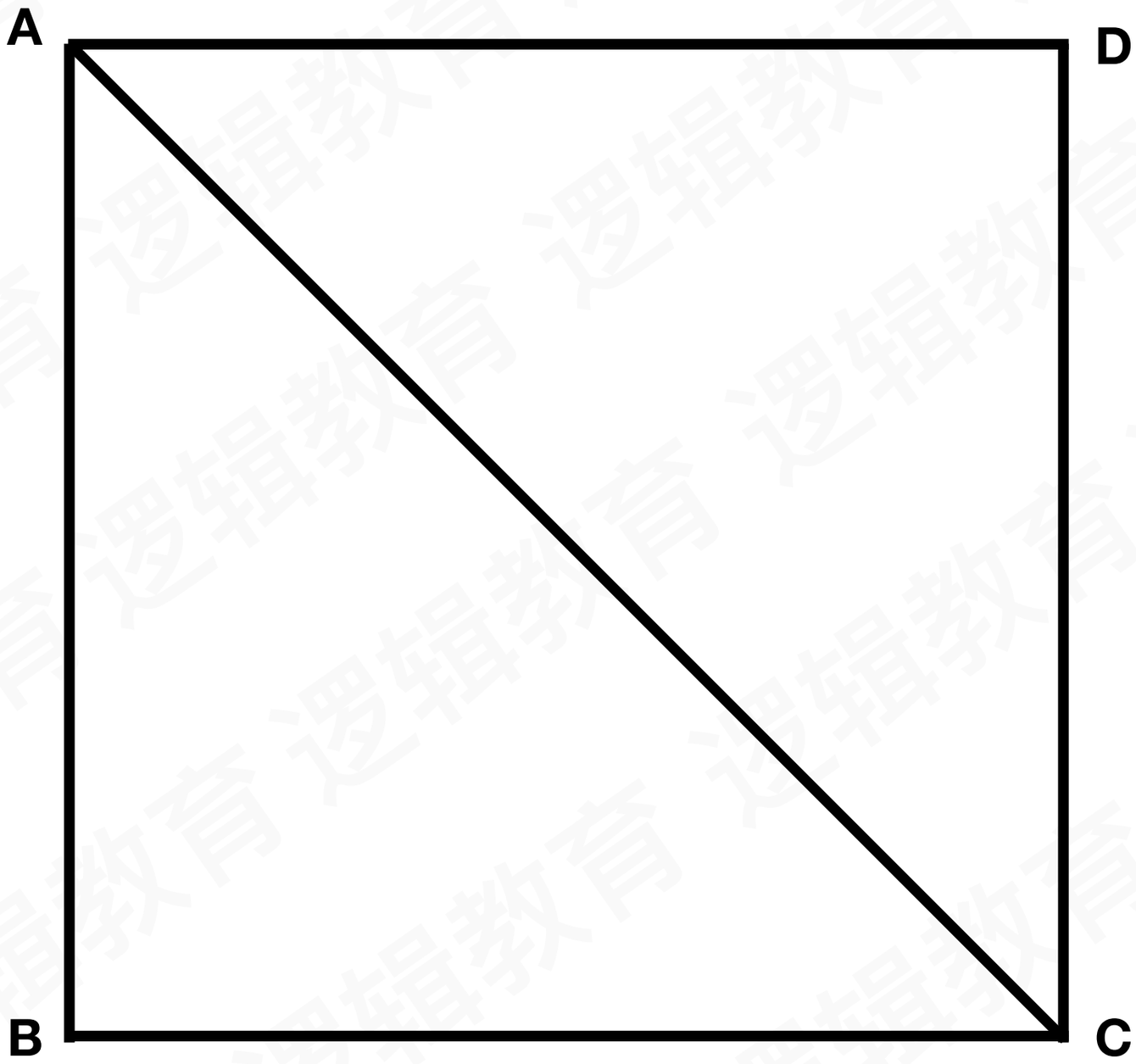
- CPU：计算视图frame，图片解码，需要绘制纹理图片通过数据总线交给GPU
- GPU：纹理混合，顶点变换与计算,像素点的填充计算，渲染到帧缓冲区
- 时钟信号：垂直同步信号V-Sync / 水平同步信号H-Sync
- iOS设备双缓冲机制：显示系统通常会引入两个帧缓冲区，双缓冲机制

图片显示到屏幕上是CPU与GPU的协作完成

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

图片滤镜原理



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

图片滤镜原理

A
(-1,1,0)

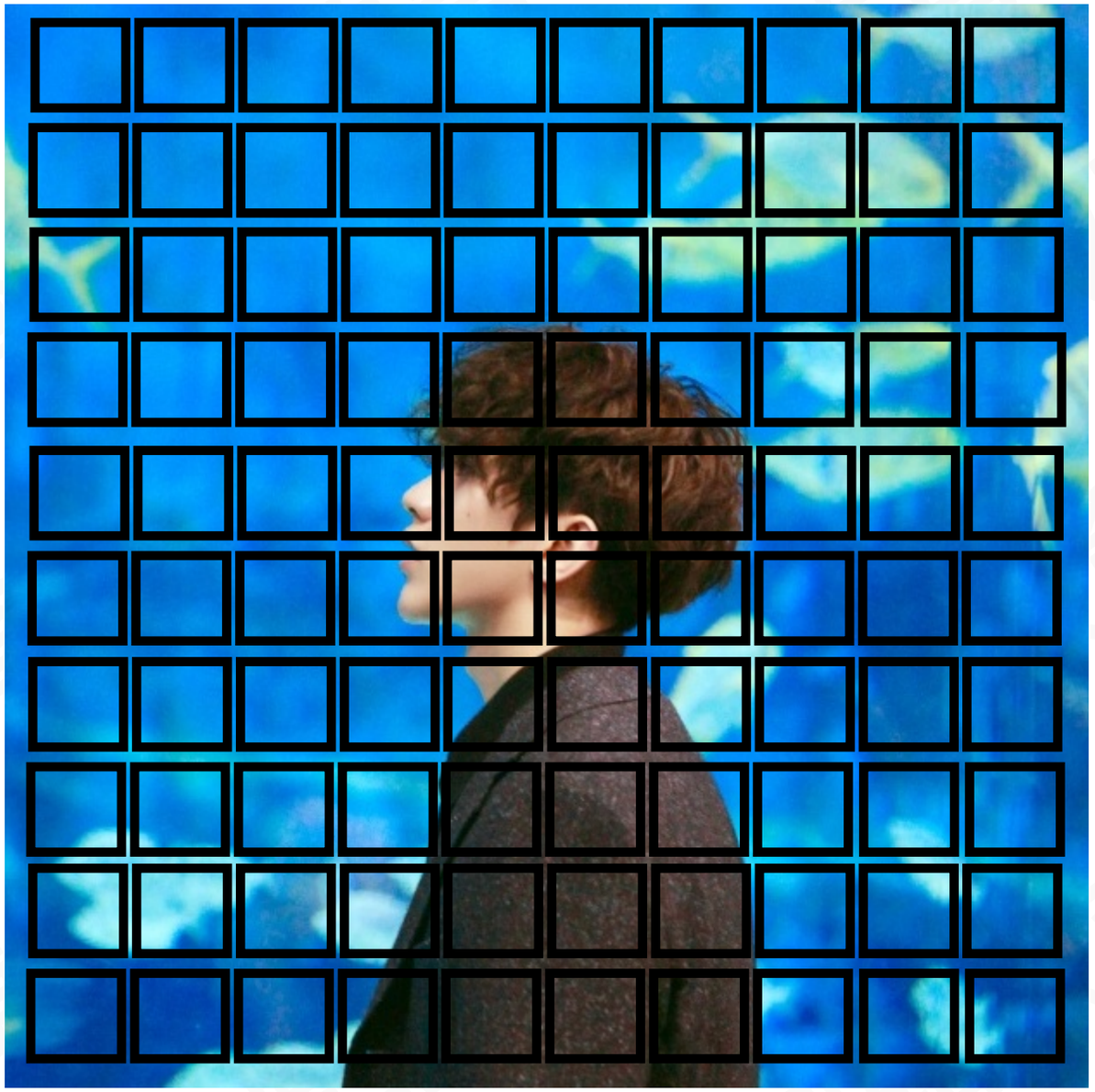
D
(1,1,0)

B
(-1,-1,0)

C
(1,-1,0)

逻辑教育
Logic education

图片滤镜原理



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

图片滤镜实现思路

- 前提条件: 能够用GLSL 显示普通图片

- 思路
  - 初始化(上下文,顶点数组,顶点数据,顶点缓存区, CAEAGLayer ， 绑定渲染缓存区/帧缓存区， 获取图片路径并将图片->纹理， 设置视口,link默认着色器)
  - 创建CADisplayLink 刷新图片

课程研发:CC老师
课程授课:CC老师

灰度滤镜原理

1.浮点算法：Gray=R*0.3+G*0.59+B*0.11
2.整数方法：Gray=(R*30+G*59+B*11)/100
3.移位方法：Gray =(R*76+G*151+B*28)>>8;
4.平均值法：Gray=(R+G+B)/3;
5.仅取绿色：Gray=G;

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

颠倒滤镜原理

纹理坐标：翻转即可

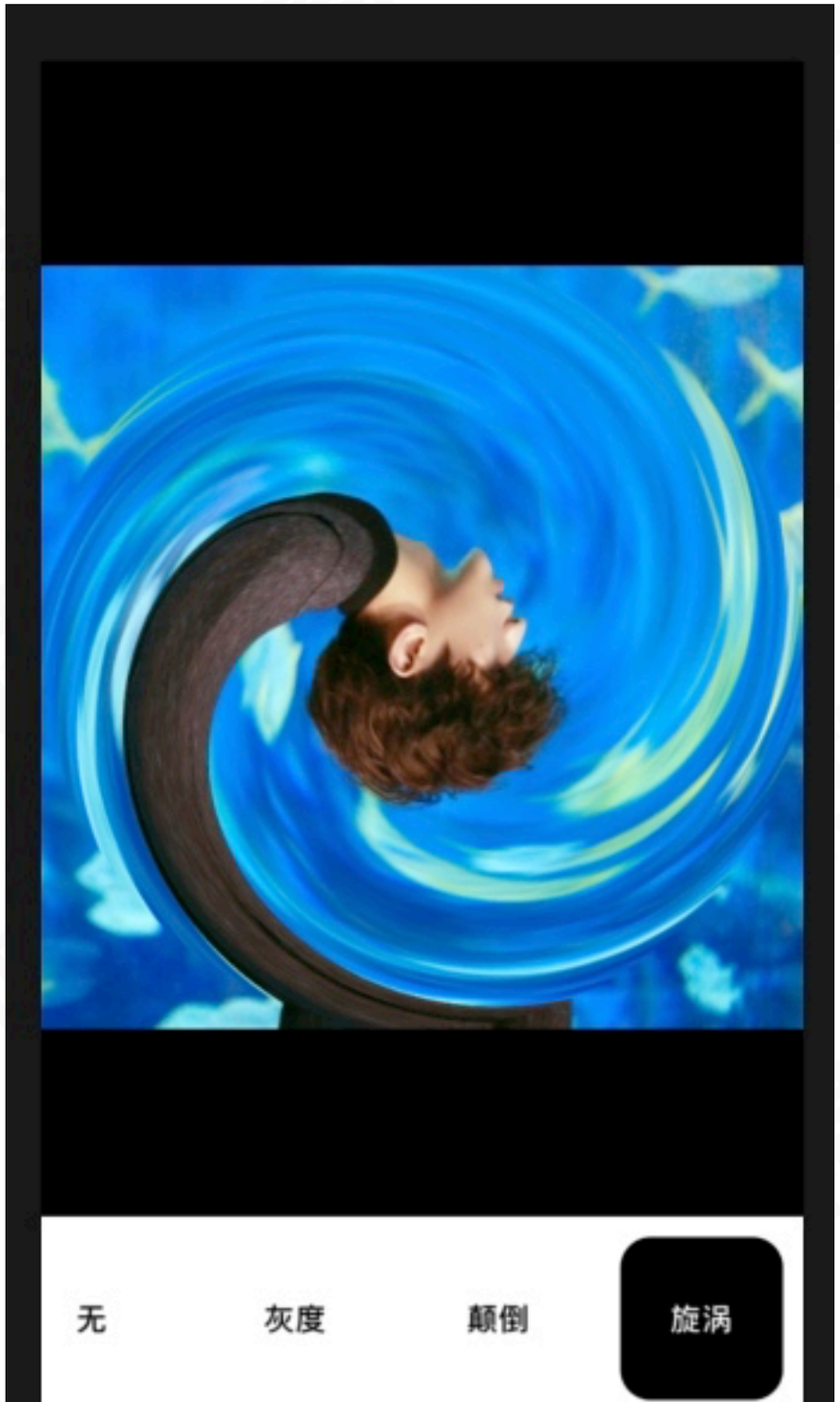
课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

旋涡滤镜原理

图像漩涡主要是在某个半径范围里，把当前采样点旋转一定角度，旋转以后当前点的颜色就被旋转后的点的颜色代替，因此整个半径范围里会有旋转的效果。如果旋转的时候旋转角度随着当前点离半径的距离递减，整个图像就会出现漩涡效果。这里使用的了抛物线递减因子：`(1.0-(r/Radius)*(r/Radius))`。

课程研发:CC老师
课程授课:CC老师



当前角度:

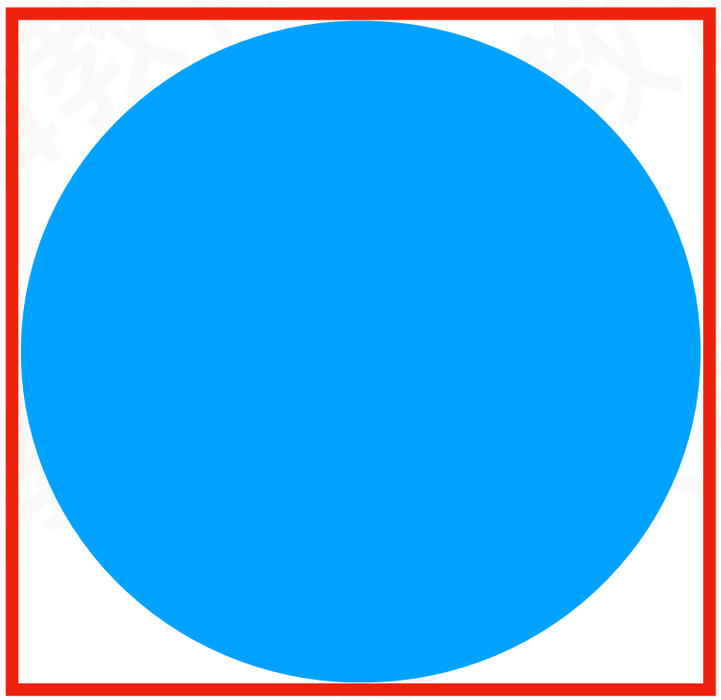
```
atan(dxy.y, dxy.x)
```

加剧旋涡角度:

```
atan(dxy.y ,dxy.x ) + radians(uD) * 2.0
```

加剧旋涡衰减角度:

```
atan(dxy.y ,dxy.x ) + radians(uD) * 2.0 * disValue
```



```
precision mediump float;
//PI
const float PI = 3.14159265;
//纹理采样器
uniform sampler2D Texture;
//旋转角度
const float uD = 80.0;
//旋涡半径
const float uR = 0.5;
//纹理坐标
varying vec2 TextureCoordsVarying;

void main()
{
    //旋转正方形范围:[512,512]
    ivec2 ires = ivec2(512, 512);
    //获取旋转的直径
    float Res = float(ires.s);
    //纹理坐标[0,0],[1,0],[0,1],[1,1]...
    vec2 st = TextureCoordsVarying;
    //半径 = 直径 * 0.5;
    float Radius = Res * uR;

    //准备旋转处理的纹理坐标 =  纹理坐标 * 直径
    vec2 xy = Res * st;

    //纹理坐标的一半
    vec2 dxy = xy - vec2(Res/2., Res/2.);
    //r
    float r = length(dxy);

    //抛物线递减因子：(1.0-(r/Radius)*(r/Radius) )
    float beta = atan(dxy.y, dxy.x) + radians(uD) * 2.0 * (1.0-(r/Radius)*(r/Radius));

    if(r<=Radius)
    {
        //获取的纹理坐标旋转beta度.
        xy = Res/2.0 + r*vec2(cos(beta), sin(beta));
    }

    //st = 旋转后的纹理坐标/旋转范围
    st = xy/Res;

    //将旋转的纹理坐标替换原始纹理坐标TextureCoordsVarying 获取对应像素点的颜色.
    vec3 irgb = texture2D(Texture, st).rgb;

    //将计算后的颜色填充到像素点中 gl_FragColor
    gl_FragColor = vec4( irgb, 1.0 );
}
```

逻辑教育
Logic education

马赛克滤镜原理

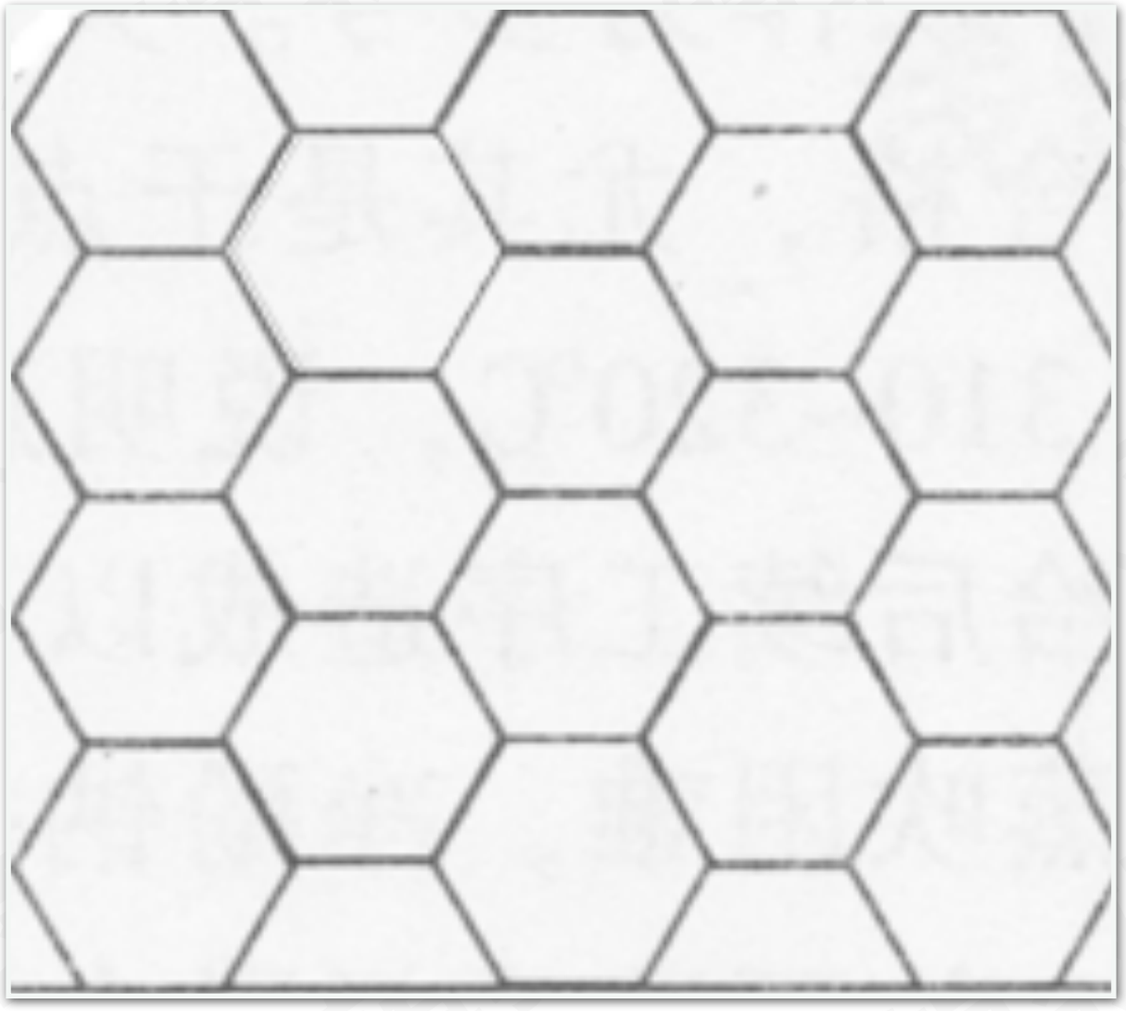
马赛克效果就是把图片的一个相当大小的区域用同一个点的颜色来表示.可以认为是大规模的降低图像的分辨率,而让图像的一些细节隐藏起来。

课程研发:CC老师
课程授课:CC老师

```
precision mediump float;
//纹理坐标
varying vec2 TextureCoordsVarying;
//纹理采样器
uniform sampler2D Texture;
//纹理图片size
const vec2 TexSize = vec2(400.0, 400.0);
//马赛克Size
const vec2 mosaicSize = vec2(16.0, 16.0);

void main()
{
    //计算实际图像位置
    vec2 intXY = vec2(TextureCoordsVarying.x*TexSize.x, TextureCoordsVarying.y*TexSize.y);

    // floor (x) 内建函数,返回小于/等于x的最大整数值.
    // floor (intXY.x / mosaicSize.x) * mosaicSize.x 计算出一个小马赛克的坐标.
    vec2 XYMosaic = vec2(floor(intXY.x/mosaicSize.x)*mosaicSize.x, floor(intXY.y/
mosaicSize.y)*mosaicSize.y);

    //换算回纹理坐标
    vec2 UVMosaic = vec2(XYMosaic.x/TexSize.x, XYMosaic.y/TexSize.y);

    //获取到马赛克后的纹理坐标的颜色值
    vec4 color = texture2D(Texture, UVMosaic);

    //将马赛克颜色值赋值给gl_FragColor.
    gl_FragColor = color;
}
```

逻辑教育
Logic education

六边形马赛克滤镜原理

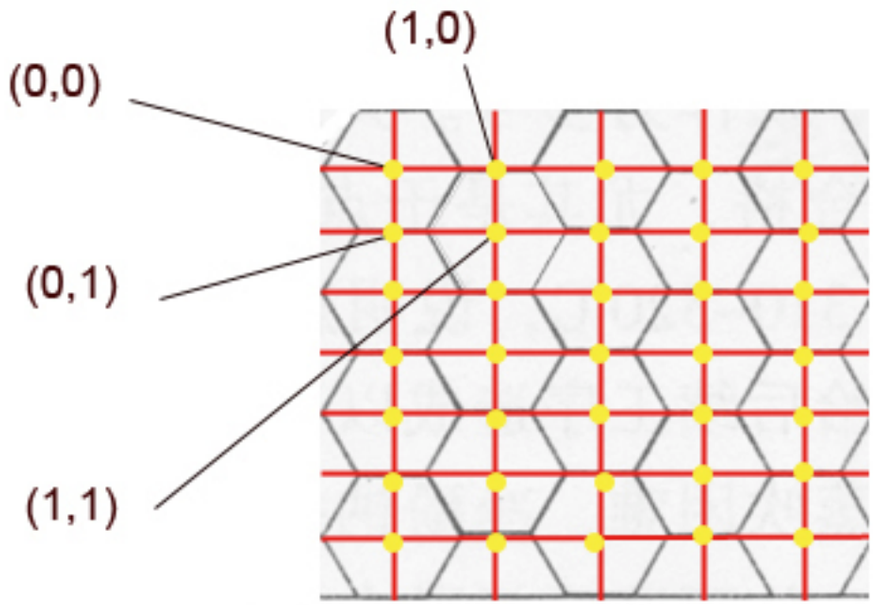


课程研发:CC老师
课程授课:CC老师

六边形马赛克滤镜原理

思路： 我们要做的效果就是让一张图片，分割成由六边形组成，让每个六边形中的颜色相同（直接取六边形中心点像素RGB较方便，我们这里采用的就是这种方法）

将它进行分割，取每个六边形的中心点画出一个矩阵，如下：



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

六边形马赛克滤镜原理

如上图，画出很多长和宽比例为 2：√3 的的矩形阵。然后我们可以对每个点进行编号，如上图中，采用坐标系标记。

假如我们的屏幕的左上点为上图的（0,0）点，则屏幕上的任一点我们找到它所对应的那个矩形了。

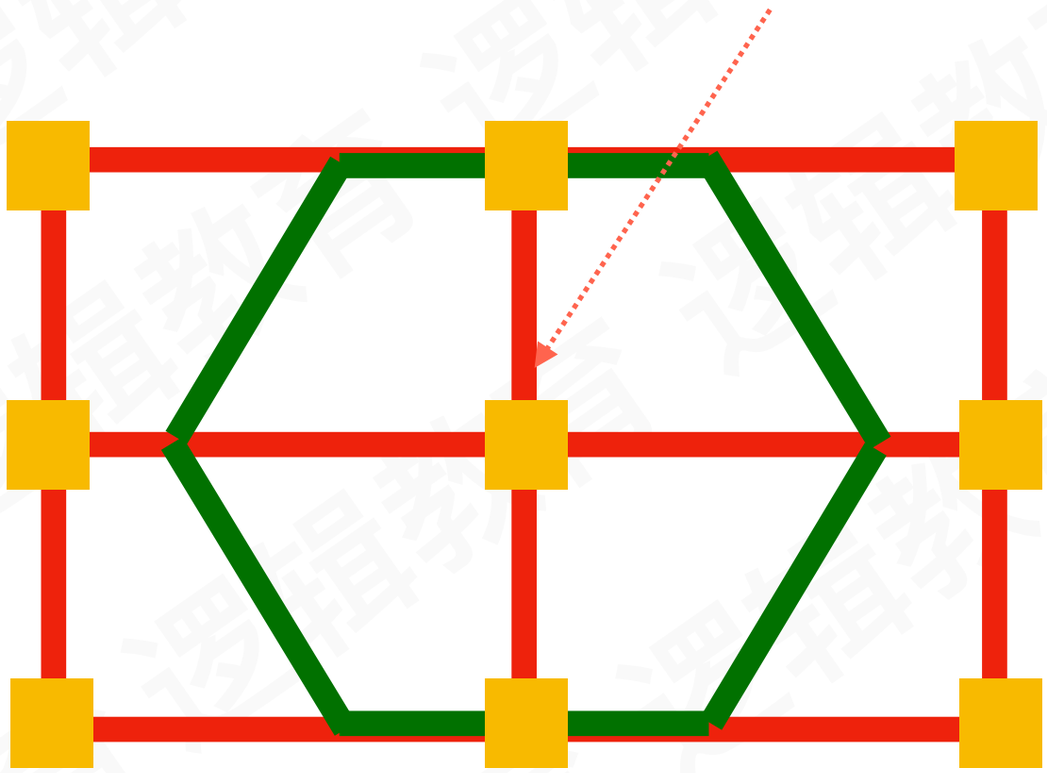
假定我们设定的矩阵比例为 2*LEN ：√3*LEN ，那么屏幕上的任意点（x, y）所对应的矩阵坐标为 (int(x/(2*LEN)), int(y/(√3*LEN)))。

```
//wx,wy -> 表示纹理坐标在所对应的矩阵坐标为
 int wx = int(x /( 1.5 * length));
 int wy = int(y /(TR * length));
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

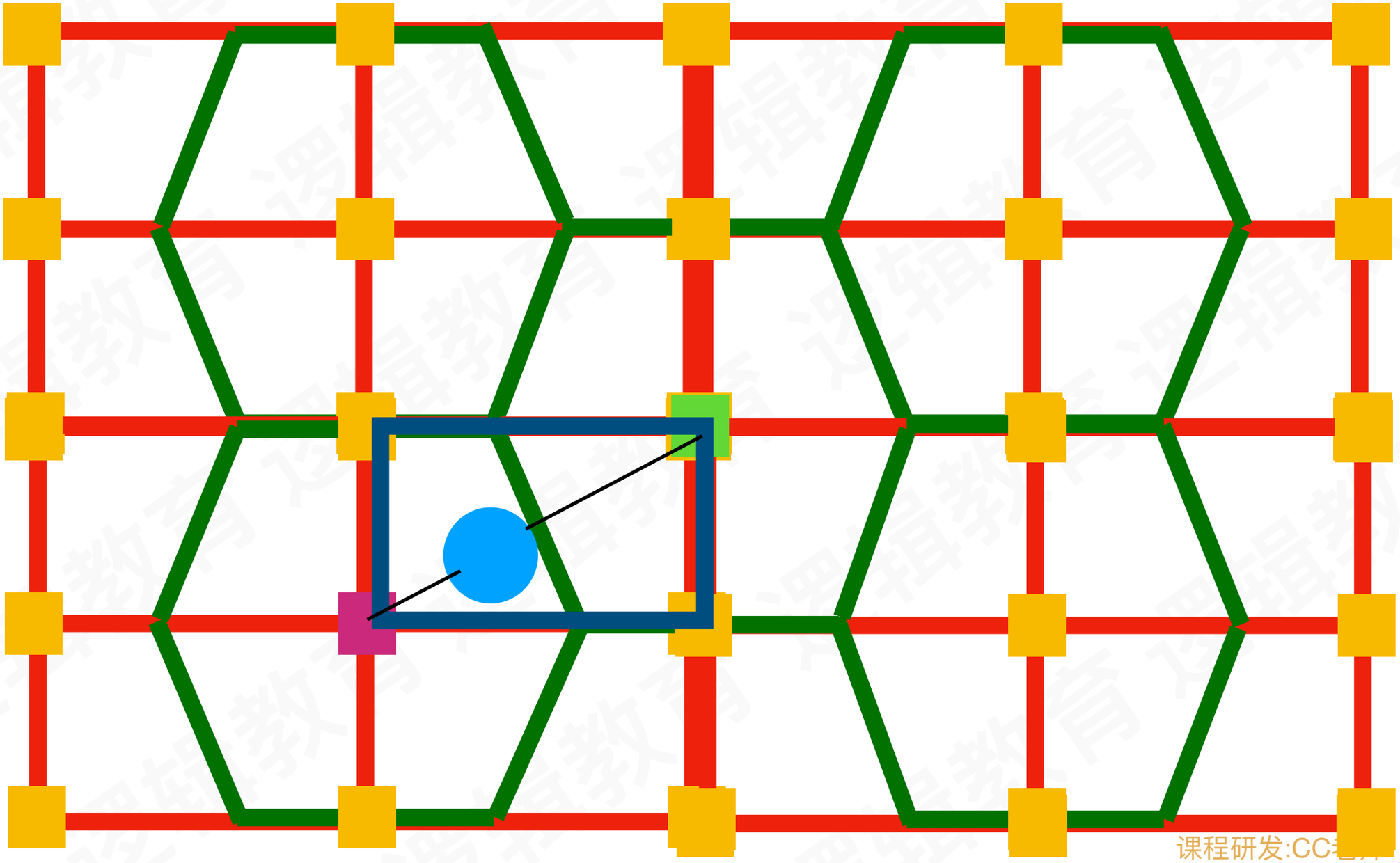
六边形马赛克滤镜原理



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

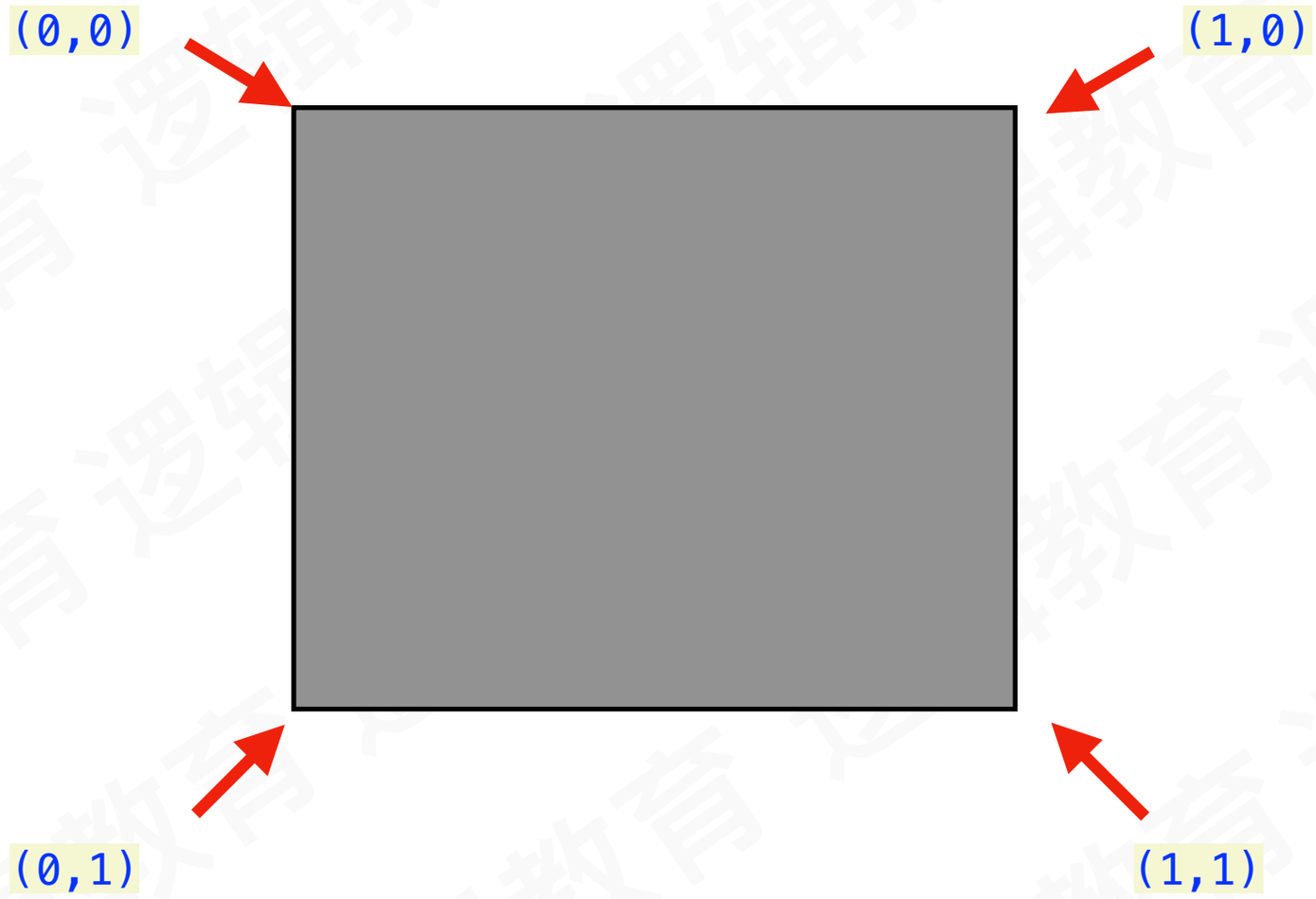
六边形马赛克滤镜原理



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

纹理坐标计算



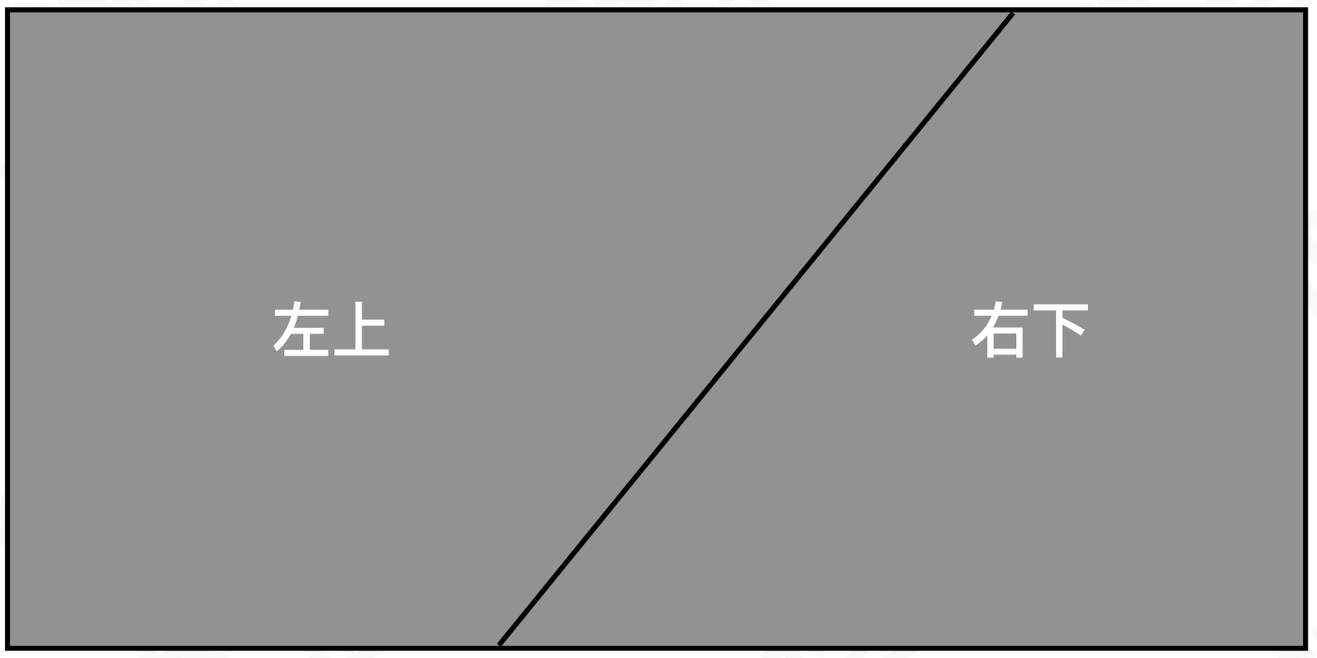
课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

坐标计算

```
vec2(length * 1.5 * float(wx), length * TR * float(wy));
```

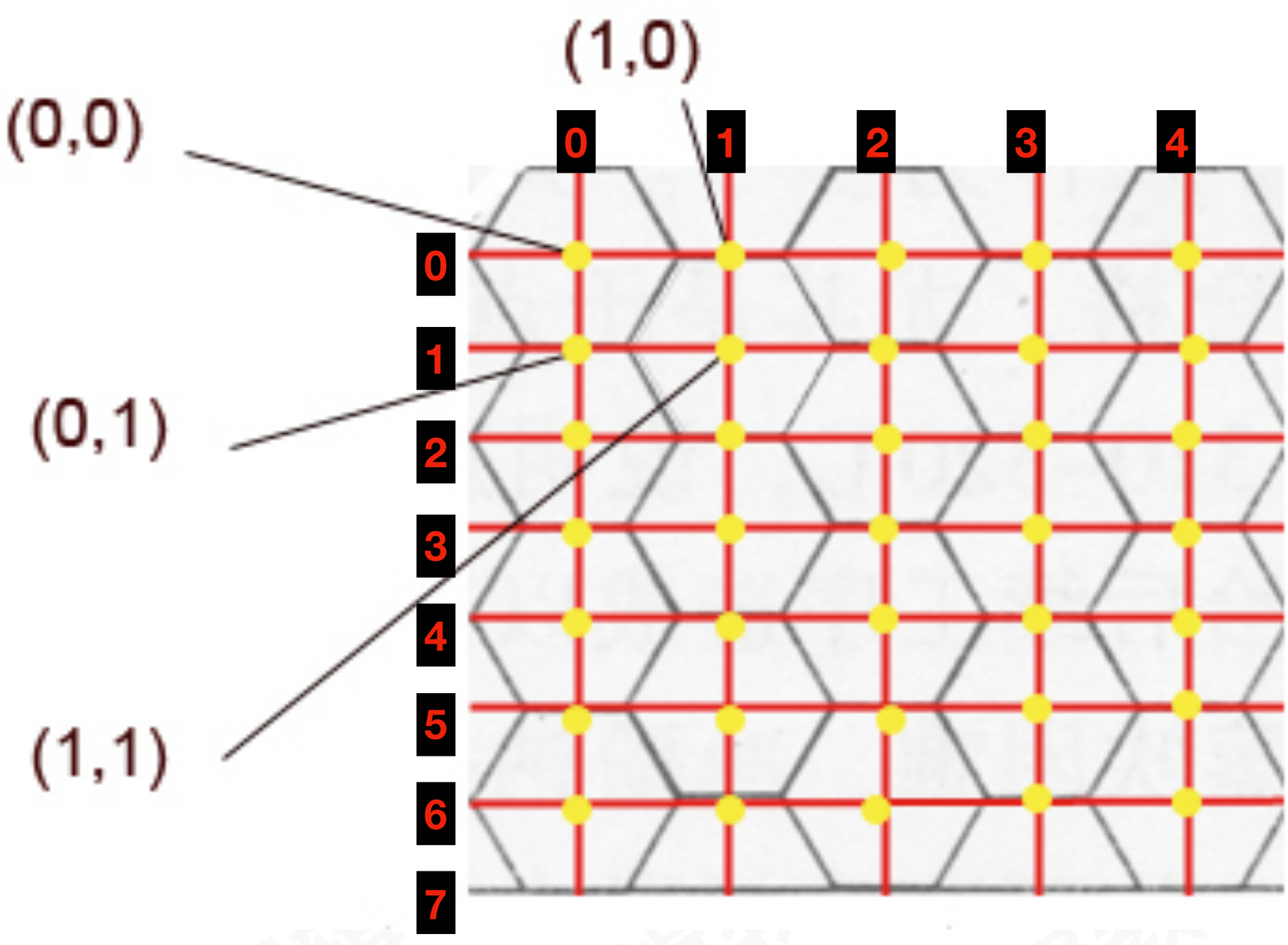
```
vec2(length * 1.5 * float(wx + 1), length * TR * float(wy))
```




```
vec2(length * 1.5 * float(wx + 1), length * TR * float(wy + 1));
```

```
vec2(length * 1.5 * float(wx), length * TR * float(wy + 1));
```

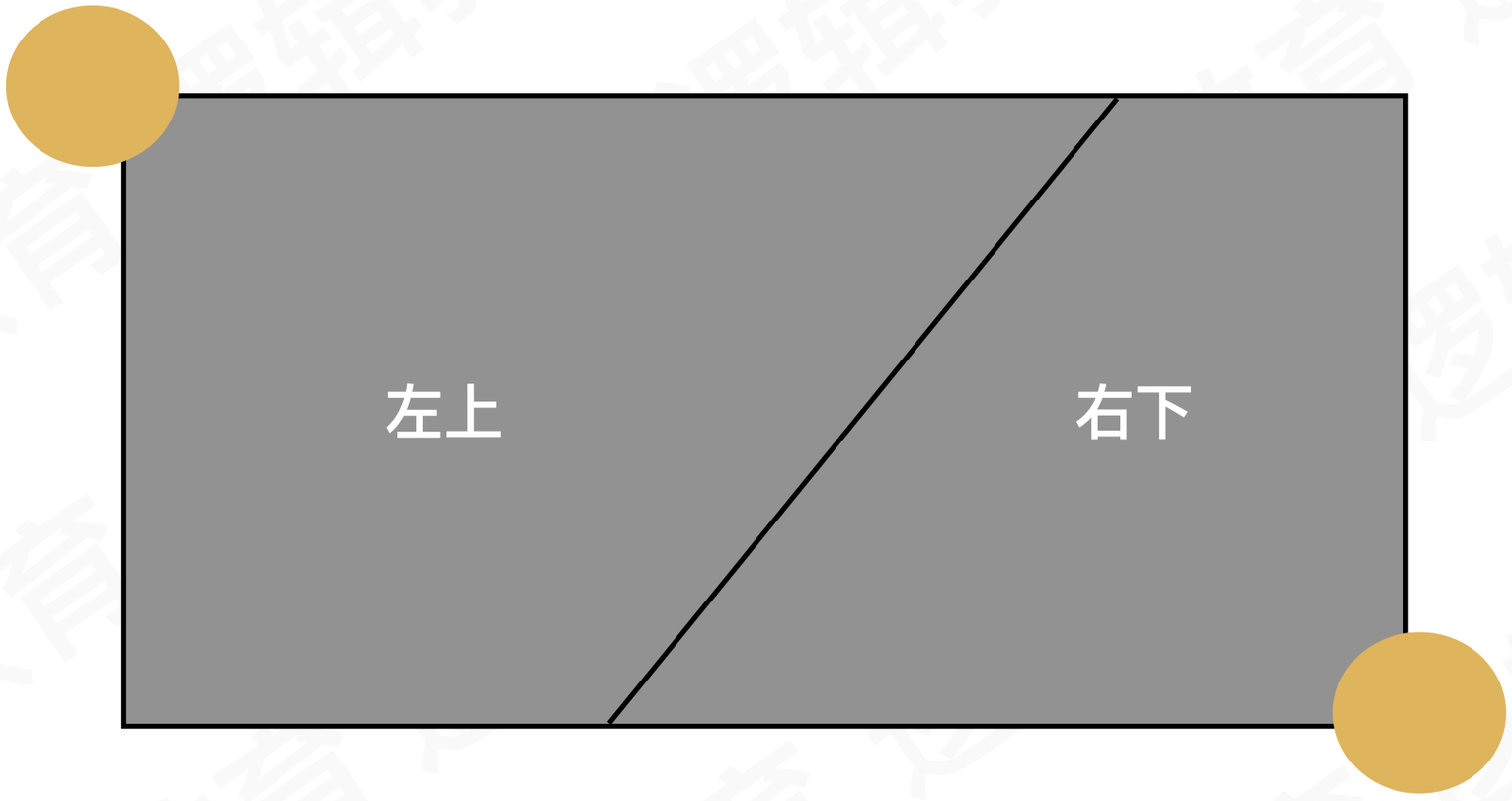
课程研发:CC老师
课程授课:CC老师



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

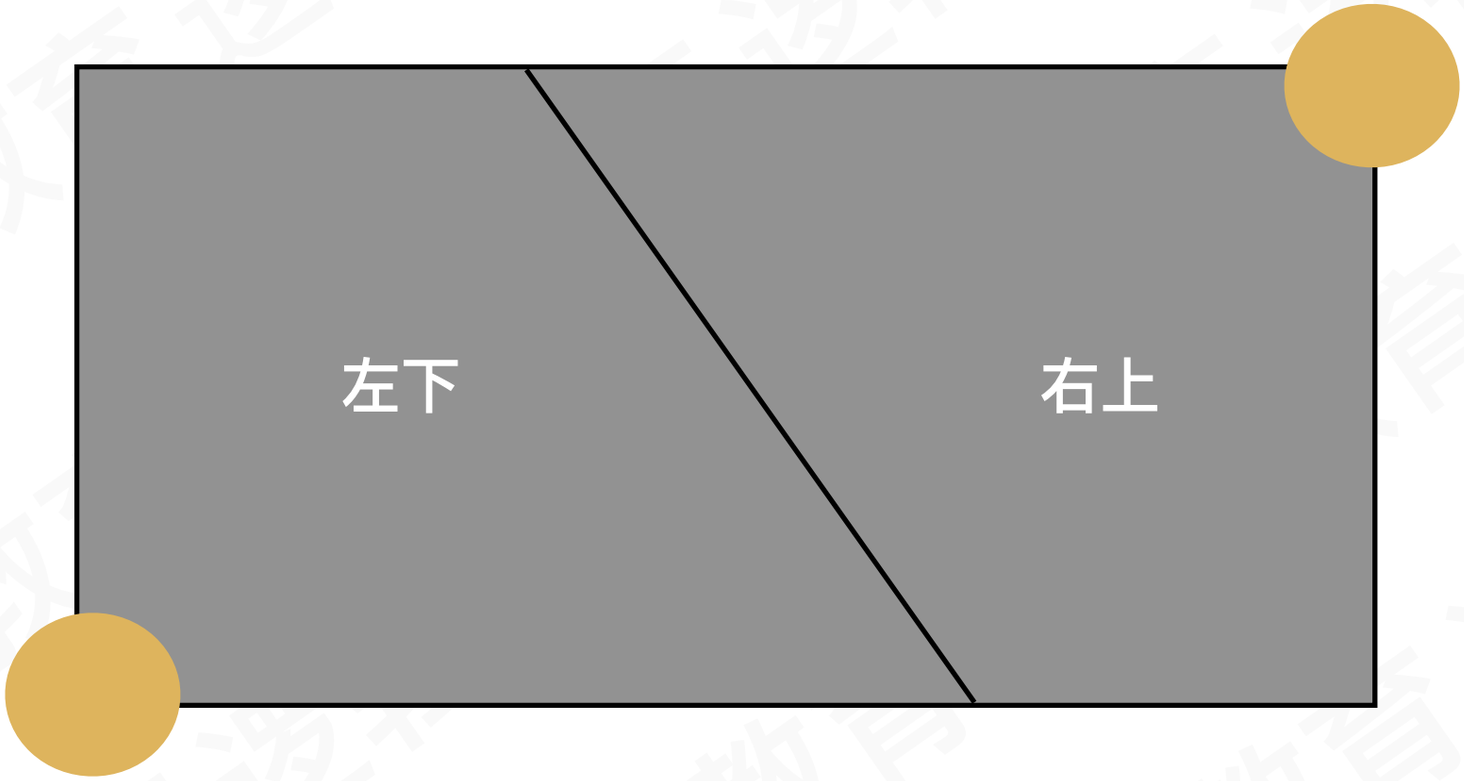
奇数列



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

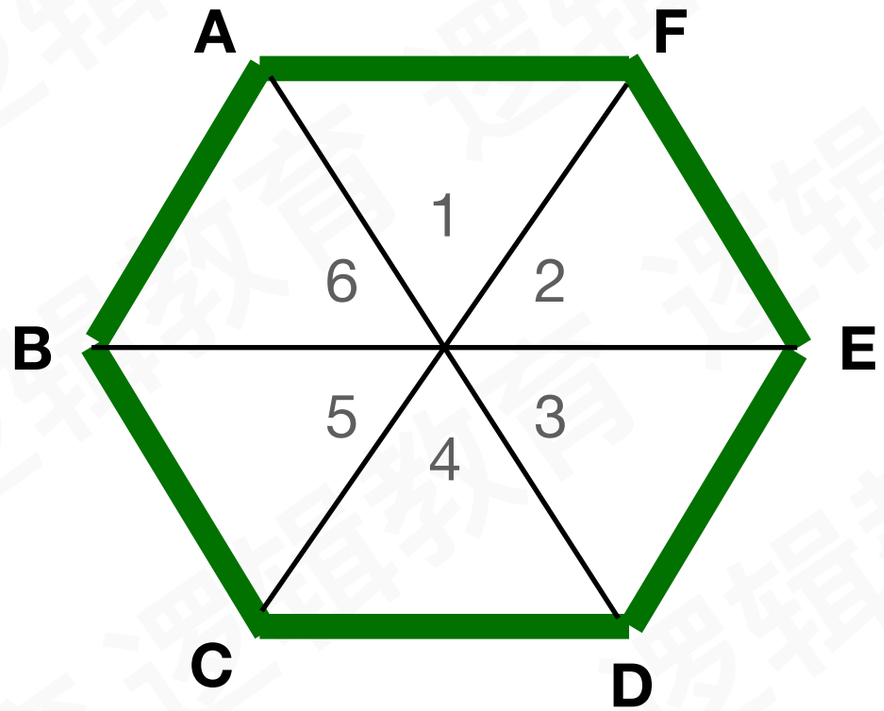
偶数数列



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

三角形马赛克滤镜原理



课程研发:CC老师
课程授课:CC老师