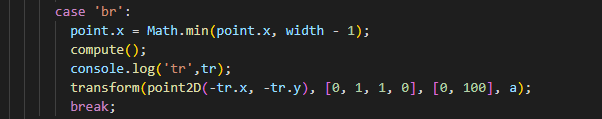


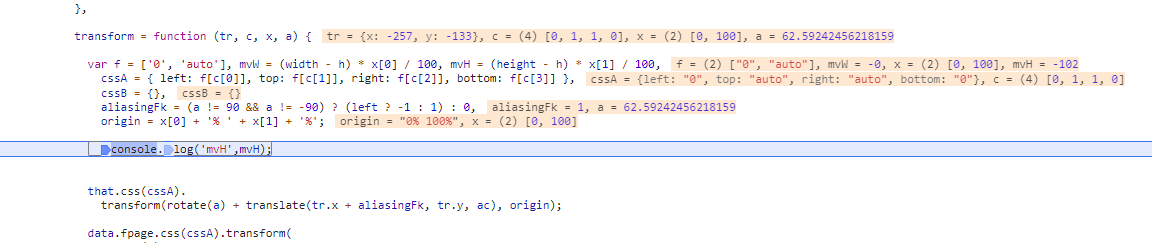
var f = ['0', 'auto'], mvW = (width - h) \* x[0] / 100, mvH = (height - h) \* x[1] / 100,

cssA = { left: f[c[0]], top: f[c[1]], right: f[c[2]], bottom: f[c[3]] },

# cssA得知[0,1,1,0]的意义



以br为例，下断点测试：

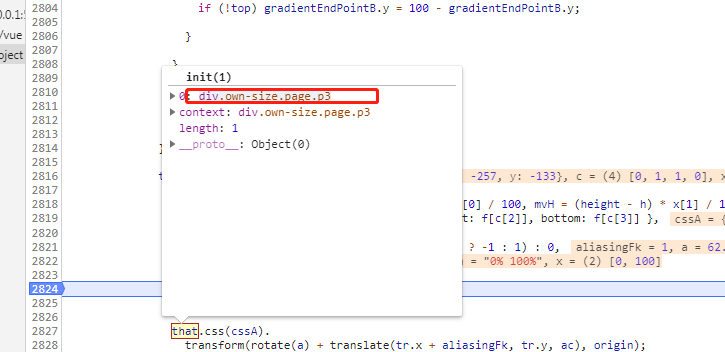


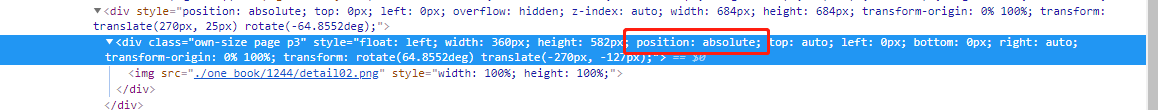
由后面

that.css(cssA).

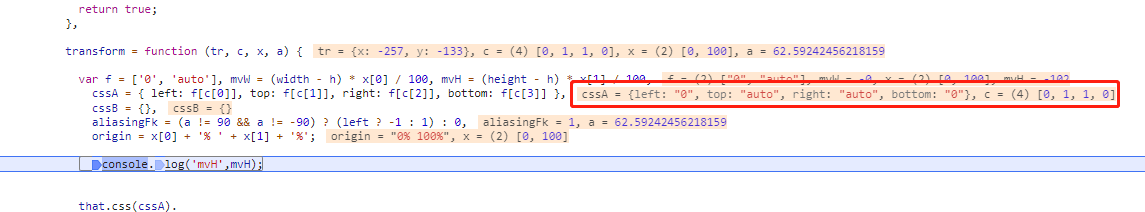
transform(rotate(a) + translate(tr.x + aliasingFk, tr.y, ac), origin);

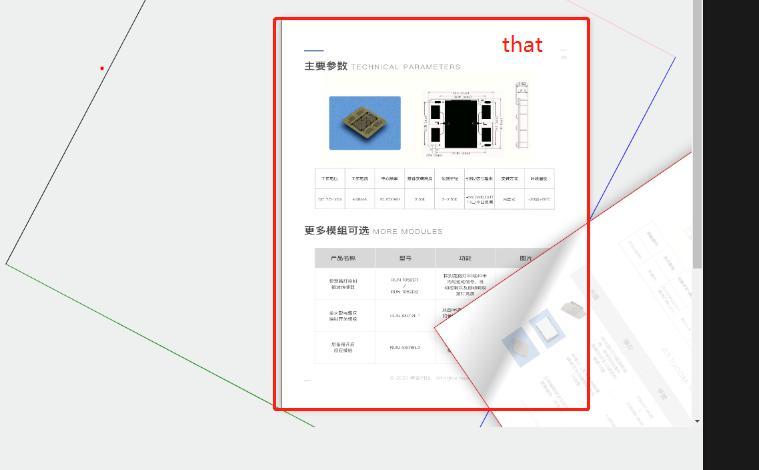
That指是下图



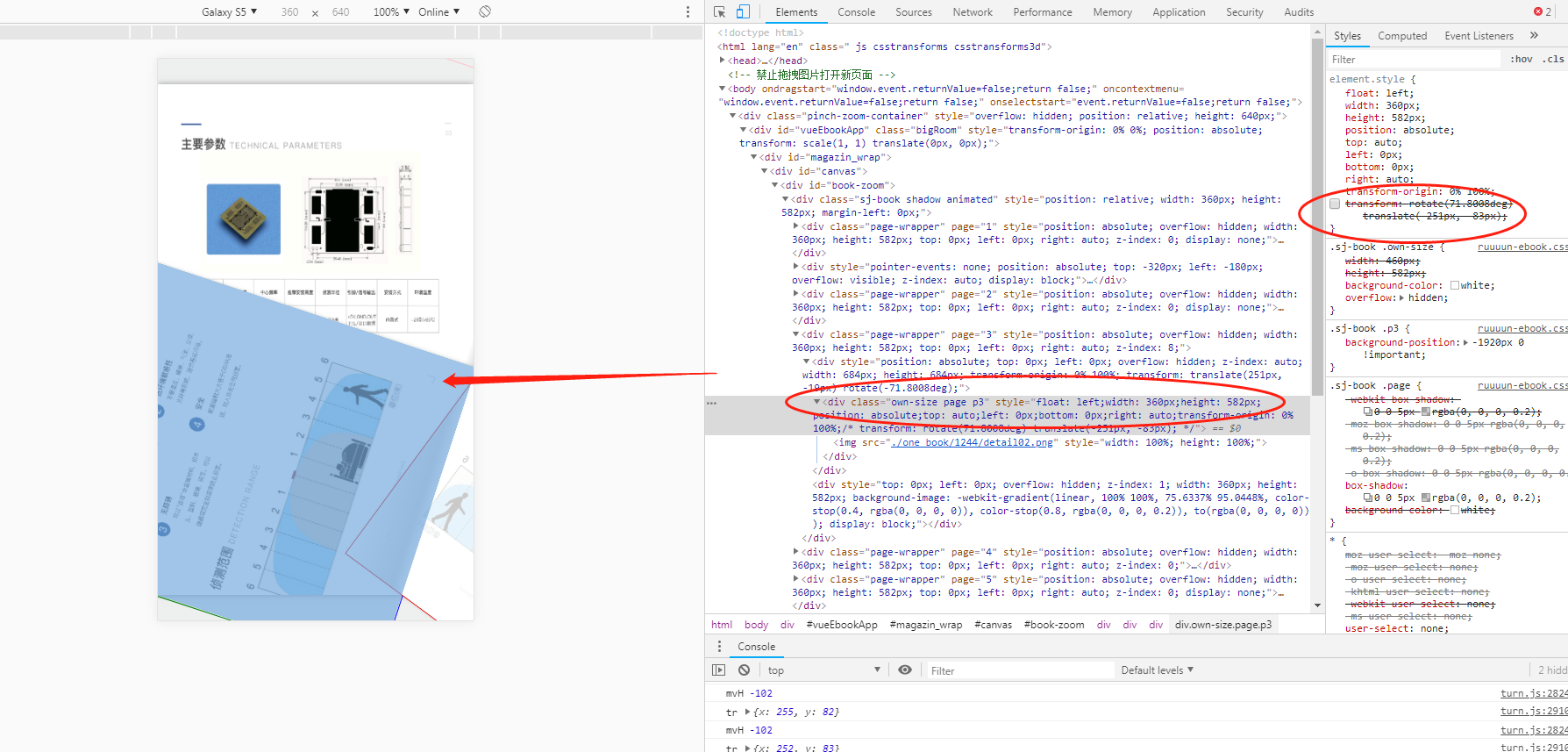


得知，cssA的作用是postion定位，

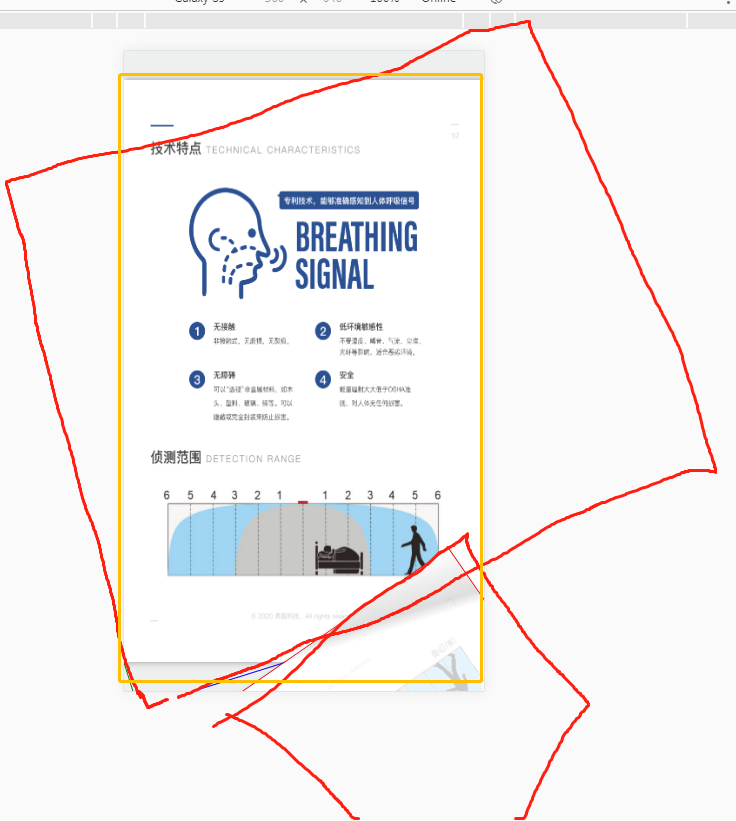




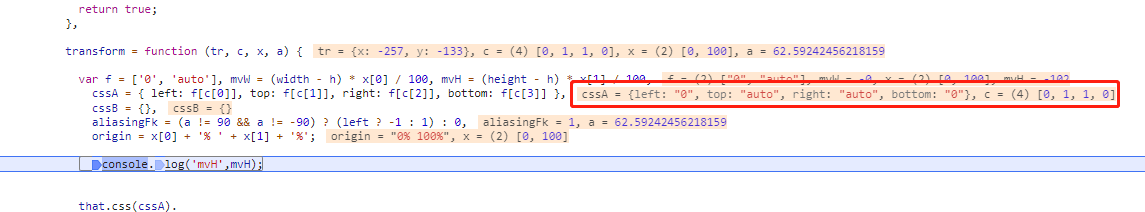
由此得知，是用left和bottom紧贴着父元素的左下角定位，如果先去掉transform,如下图



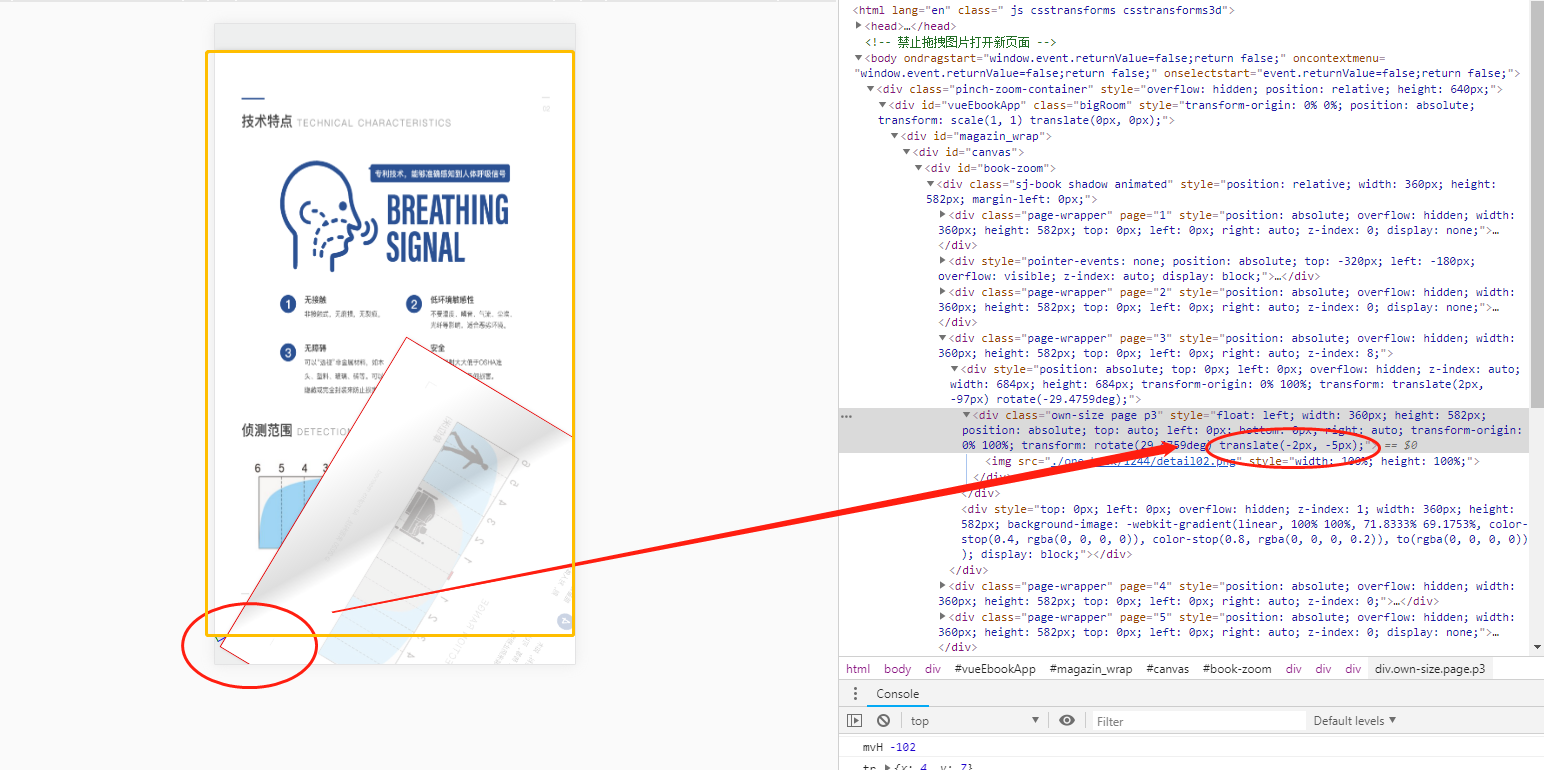
由此得知，cssA的作用是定位当前页，而cssA由第二各参数如[0, 1, 1, 0]确定，得知[0, 1, 1, 0]的作用是确定当前页的定位，至于为什么是0110,看下图：



Br这个角，就是围绕着左下角为中心进行旋转，所以当前页就是用left ,bottom左下角来定位，故就是[0,1,1,0] ，如下图：

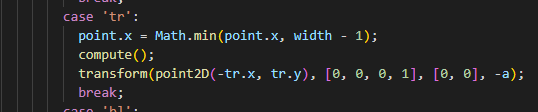


当折角碰到边的时候，怎么做到卡位？我们看下图：

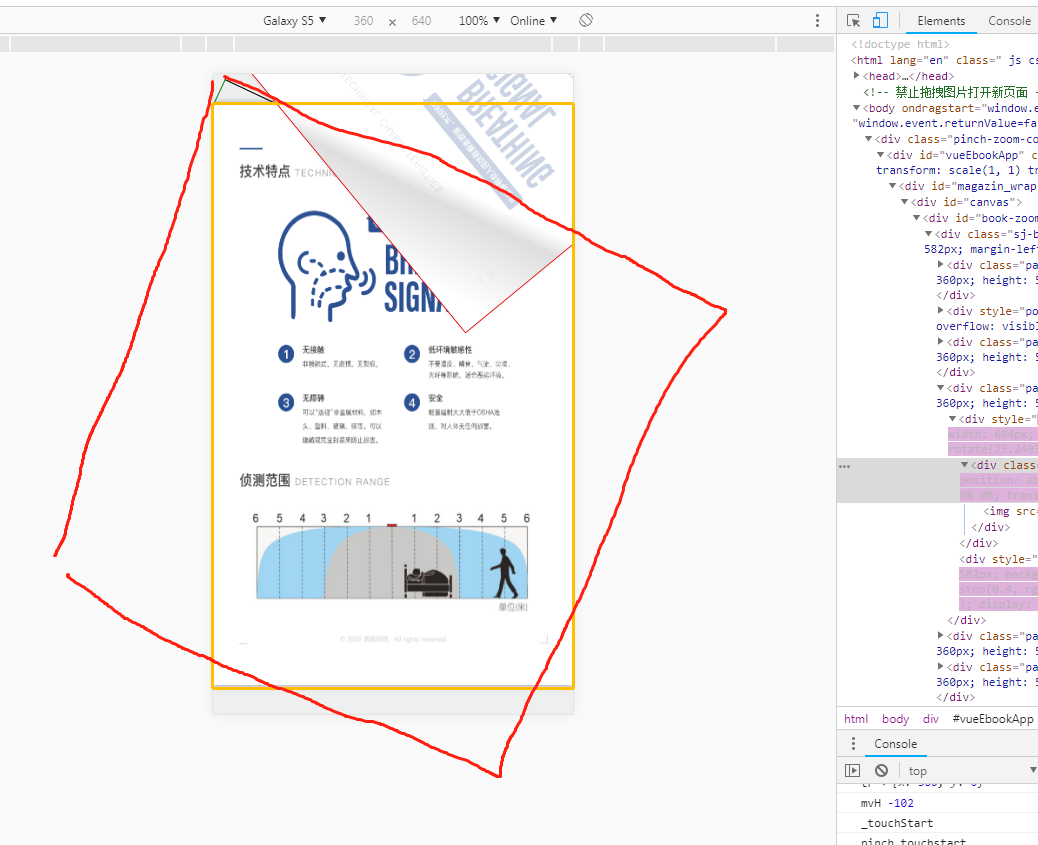


上图，that就是黄色框，cssA控制的部分，我们可以看到，tanslate的参数接近0，这个就是作为判断标准就可以实现卡位效果了。

那么，为什么tr是[0,0,0,1]呢？

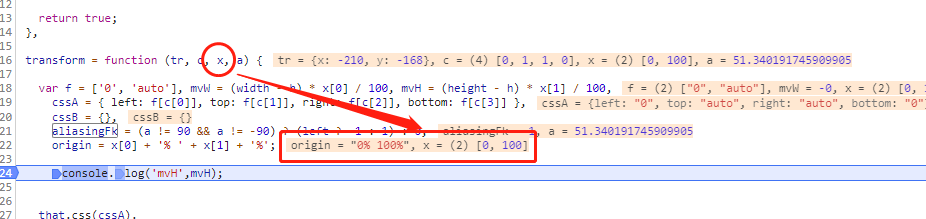


看下图：



黄色是that,父元素是红色框，可以看到通过top+left的定位+translate就可以形成这样的效果，故必须是[0,0,?,1],由于left优先级高于right，所以第三个无所谓，试过了，我把tr改成[0, 0, 1, 1]无变化。

# Origin得知第三个参数是旋转点

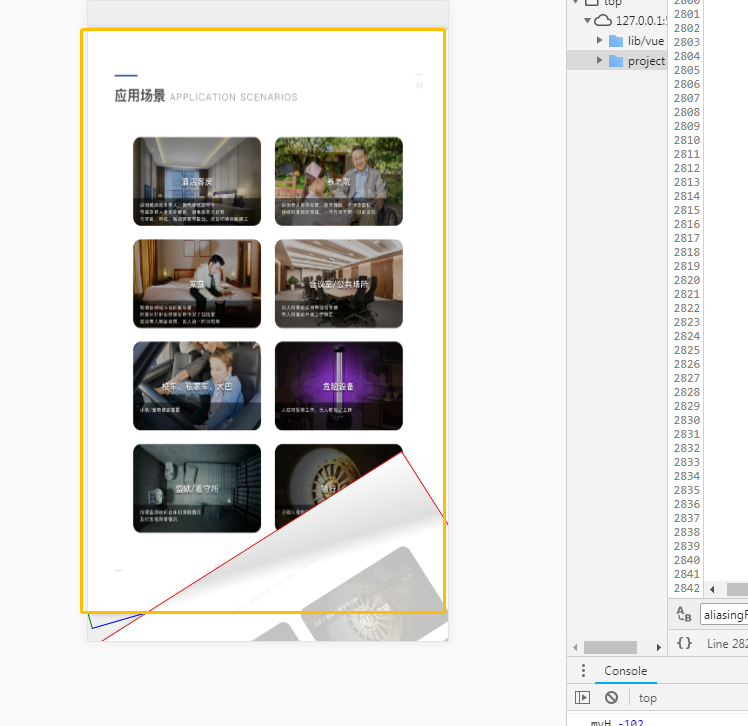


为什么 br 是 0% 100% ?

看后面代码用到了origin：

that.css(cssA).

transform(rotate(a) + translate(tr.x + aliasingFk, tr.y, ac), origin);

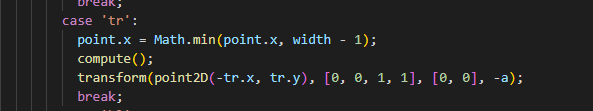


得知，that（黄色）通过left+bottom定位，需要通过调整origin 为0% 100% 实现旋转.

origin的应用 <https://c.runoob.com/codedemo/3391/>

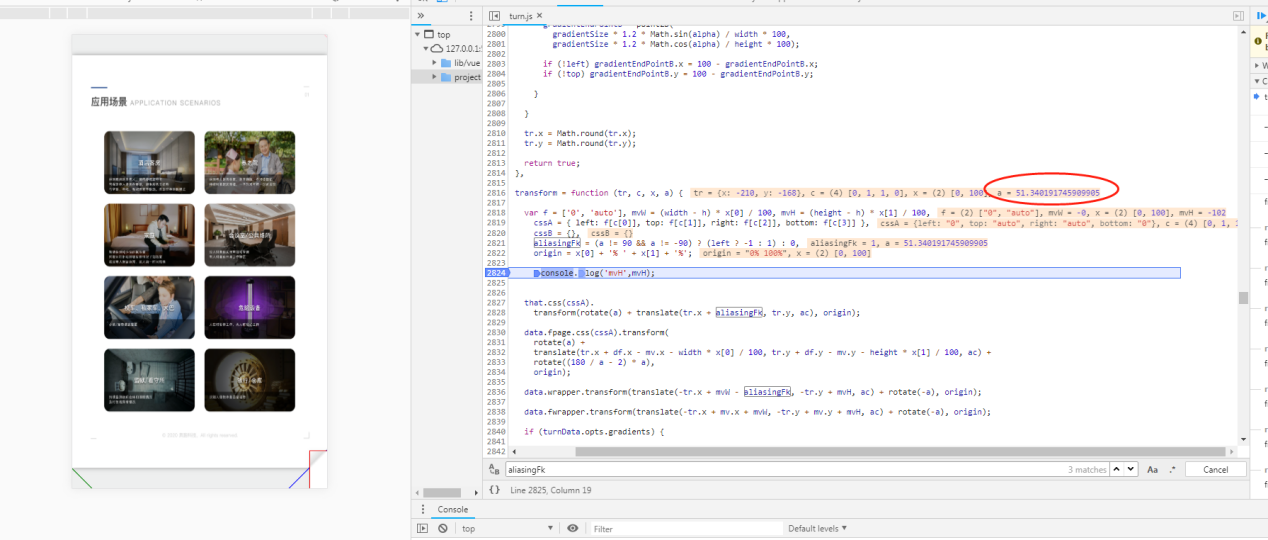
因此，第三个参数是[0,100]

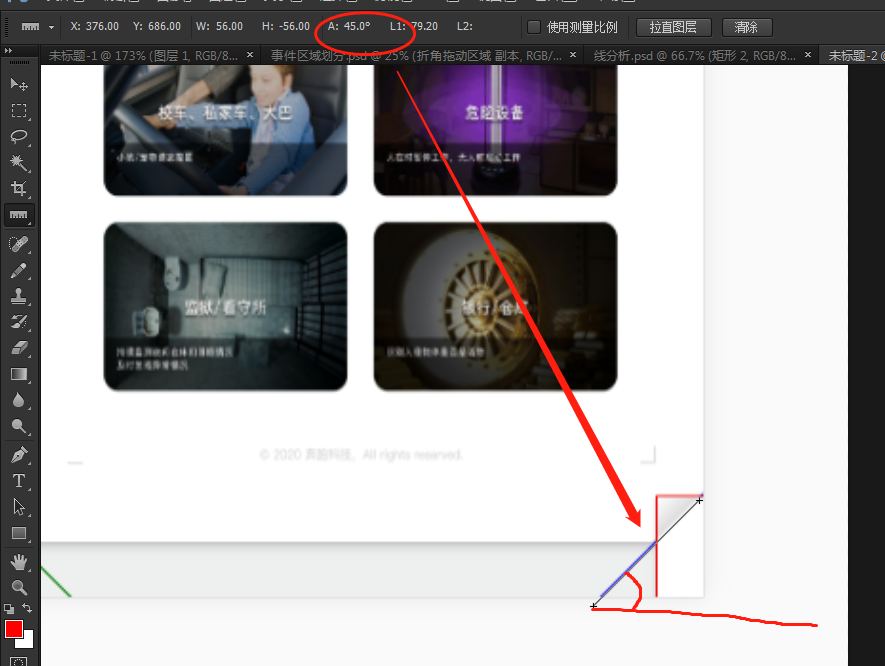
那么，为什么tr是[0,0]?



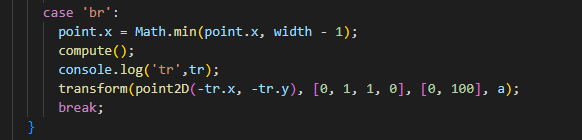
因为tr是left+top定位的，旋转点origin就是0% 0%.

# aliasingFk是折角线的角度



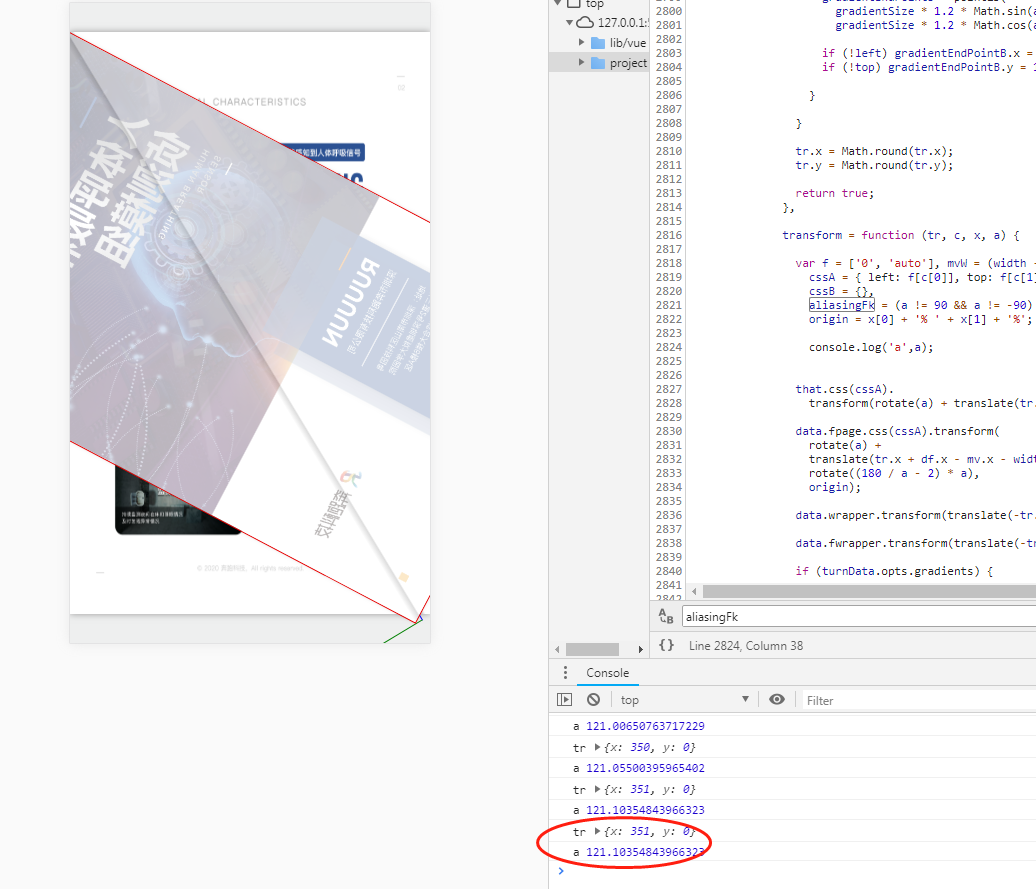


为什么br 是正a?

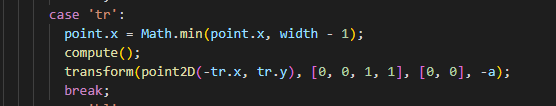


因为br折角线的角度范围就是0到180.

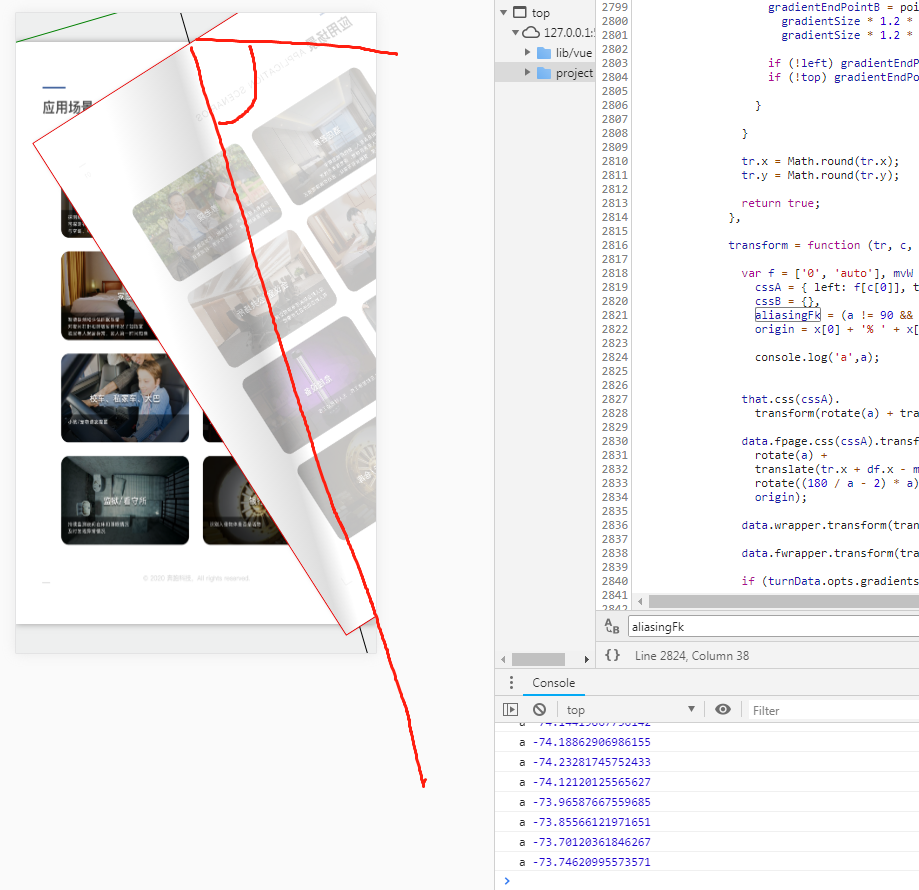
大于90度的情况如下：



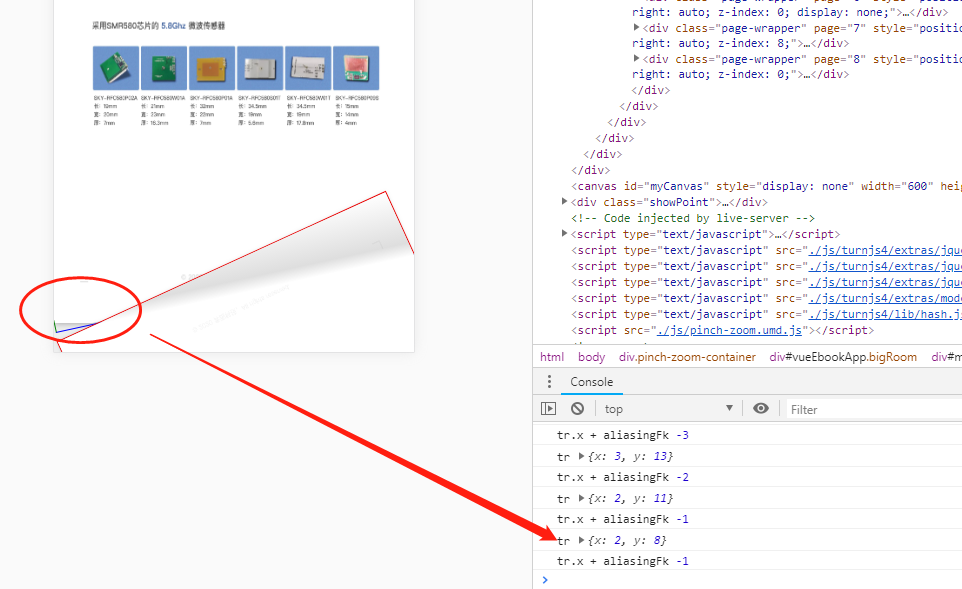
那么为什么tr 是 -a?



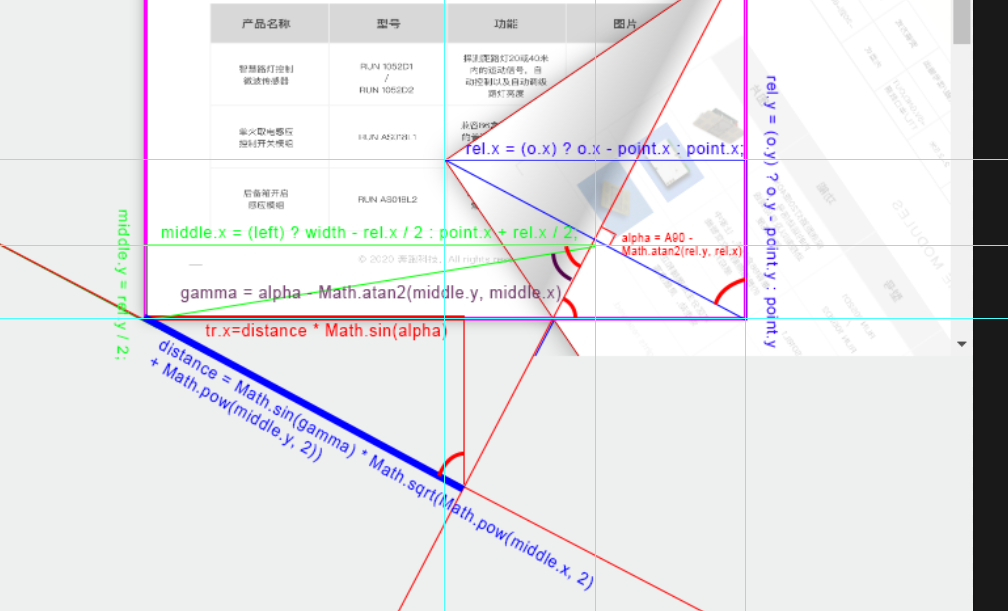
由下图得知，



# Tr的计算是关键！

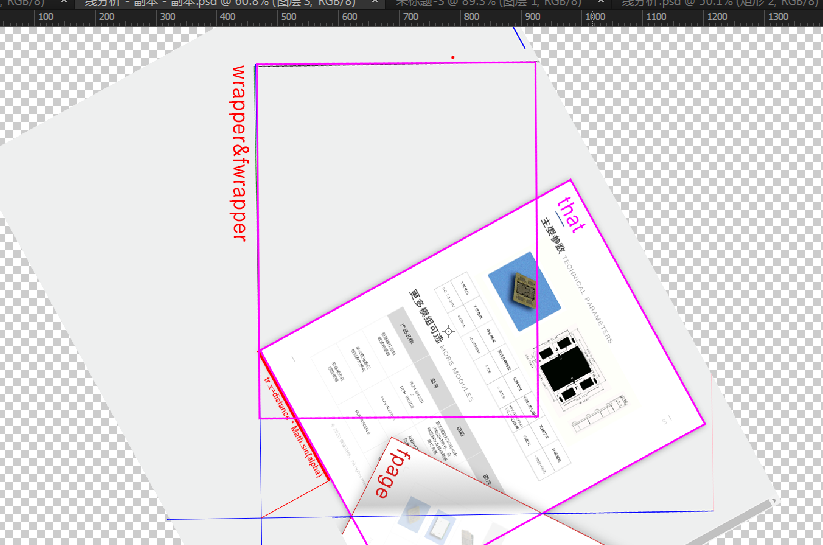


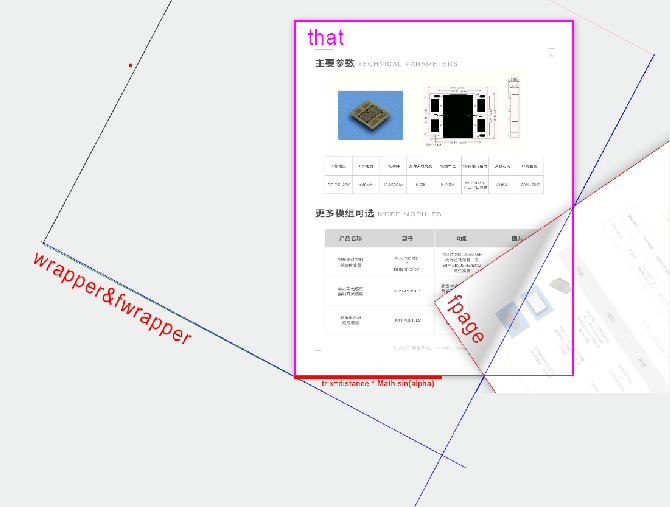
Br为例，如何计算tr?



# Transform

变形之前：

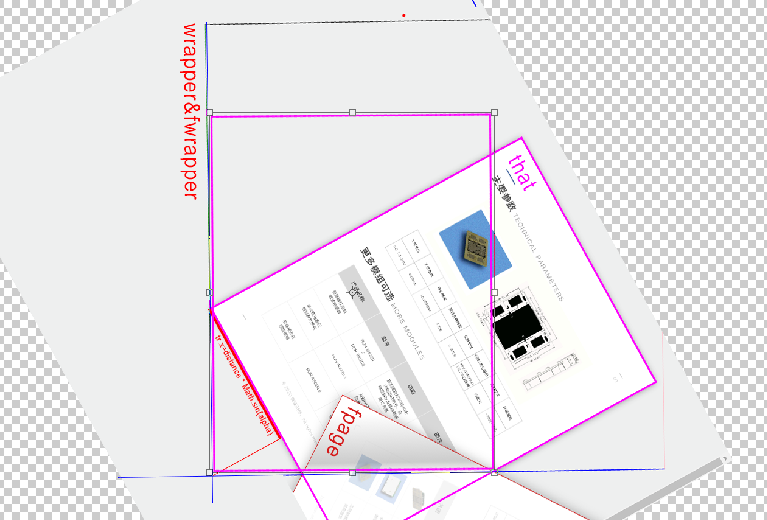




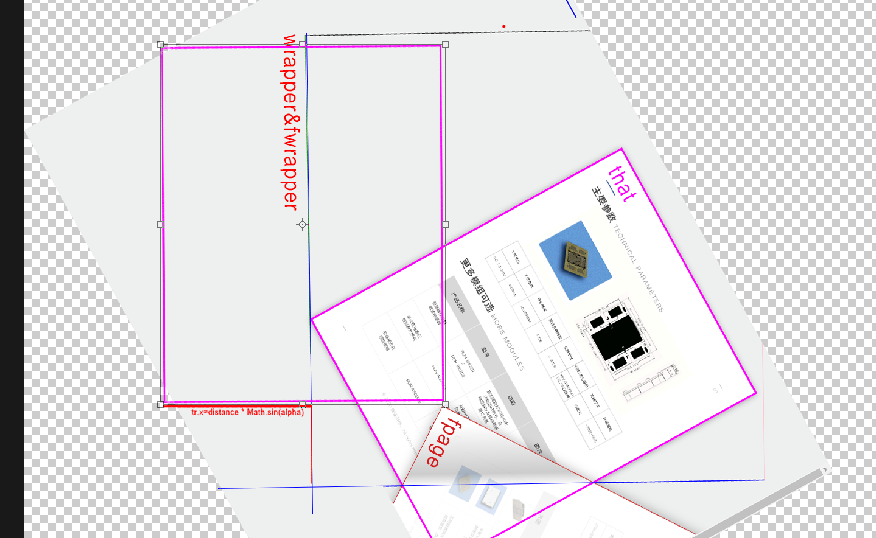
mvW = (width - h) \* x[0] / 100, mvH = (height - h) \* x[1] / 100,

## Wrapper&fwrapper

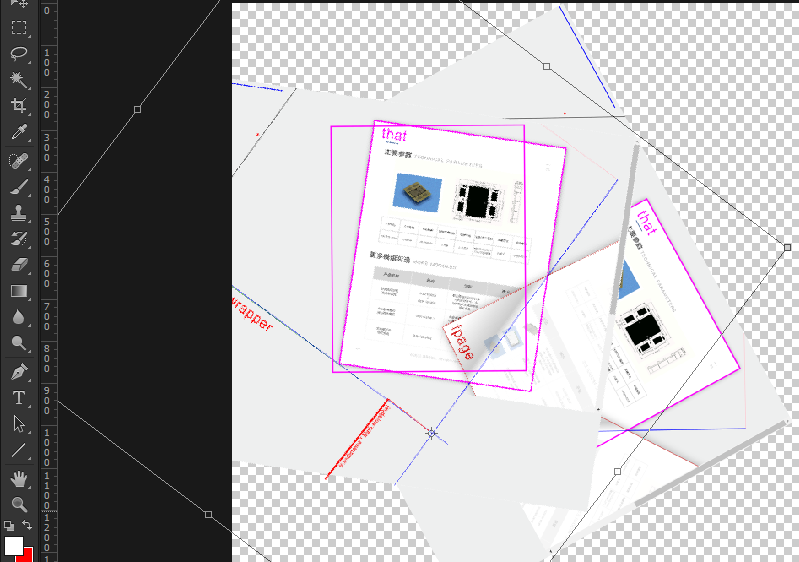
先归位：用mvH/mvW去归位：



再平移，用tr.x tr.y



再旋转：



所以代码就是：

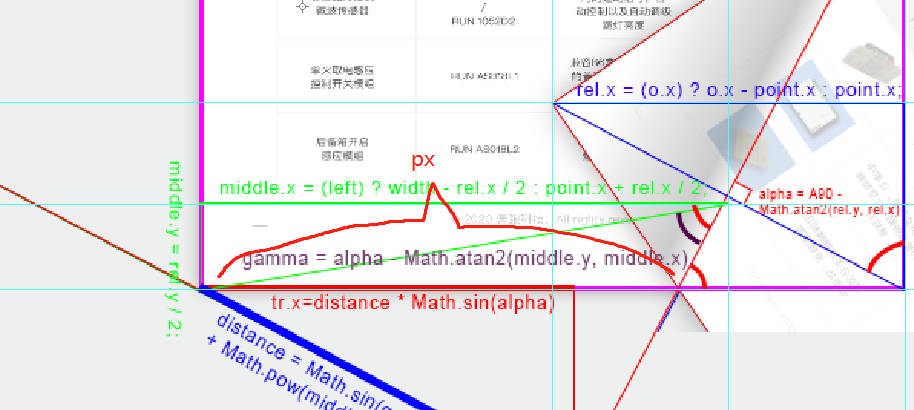
data.wrapper.transform(translate(-tr.x + mvW - aliasingFk, -tr.y + mvH, ac) + rotate(-a), origin);

aliasingFk是1px偏差。

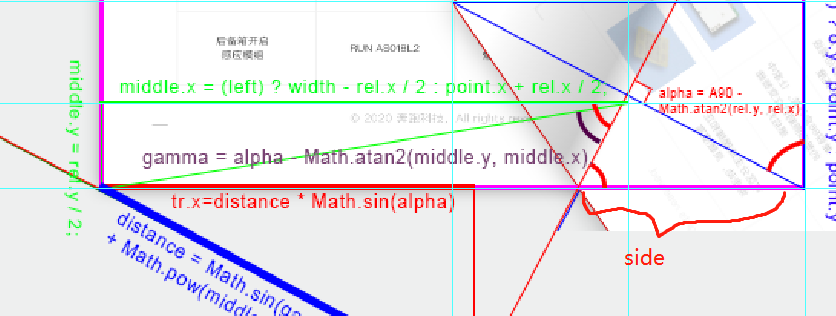
## That:就是wrapper的逆向

## 最难理解的fpage

px = Math.round(tr.y / Math.tan(alpha) + tr.x);

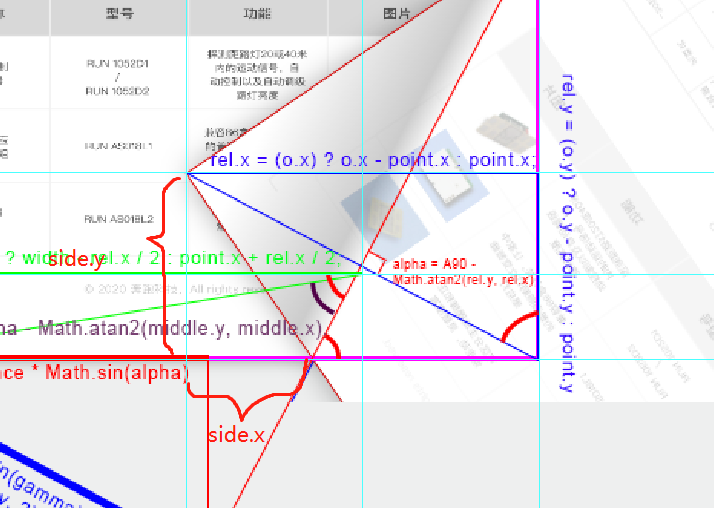


var side = width - px,



sideX = side \* Math.cos(alpha \* 2),

sideY = side \* Math.sin(alpha \* 2);

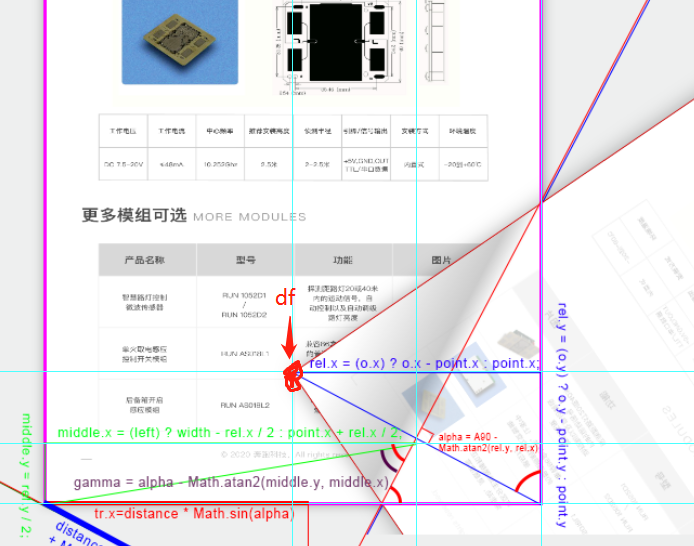


df = point2D(

Math.round((left ? side - sideX : px + sideX)),

Math.round((top) ? sideY : height - sideY));

sideX由于是负值，故df的点就是



data.fpage.css(cssA).transform(

rotate(a) +

translate(tr.x + df.x - mv.x - width \* x[0] / 100, tr.y + df.y - mv.y - height \* x[1] / 100, ac) +

rotate((180 / a - 2) \* a),

origin);

## tr.x + df.x - mv.x - width \* x[0] / 100

### 在alpha < 90 的时候，mv.x为0

简化：

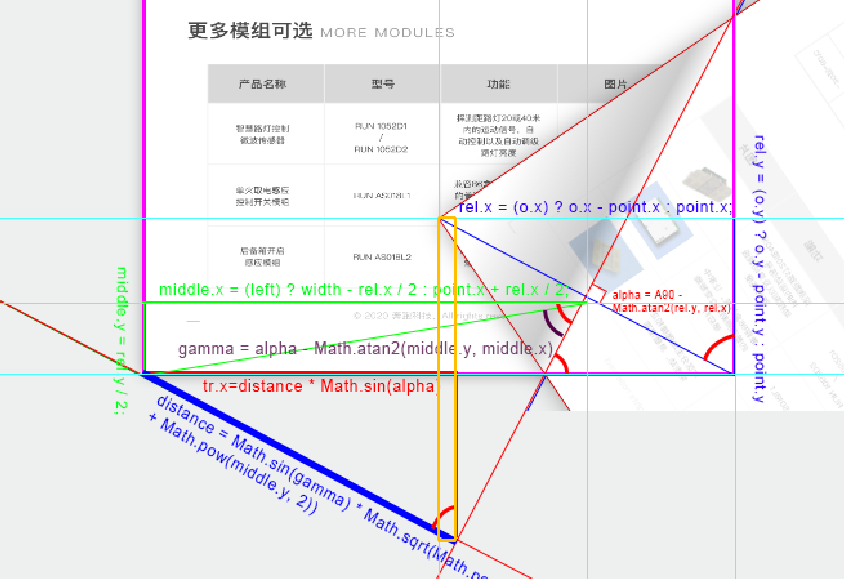
tr.x + df.x - width \* x[0] / 100

横屏 的br ，X[0]= 100,故

简化：

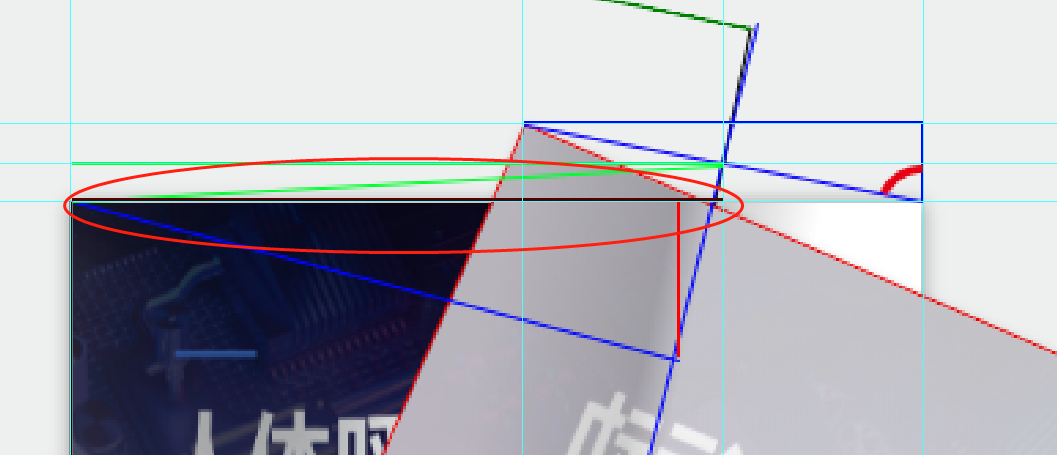
tr.x + df.x – width

一下是变形的矩形：黄色矩形

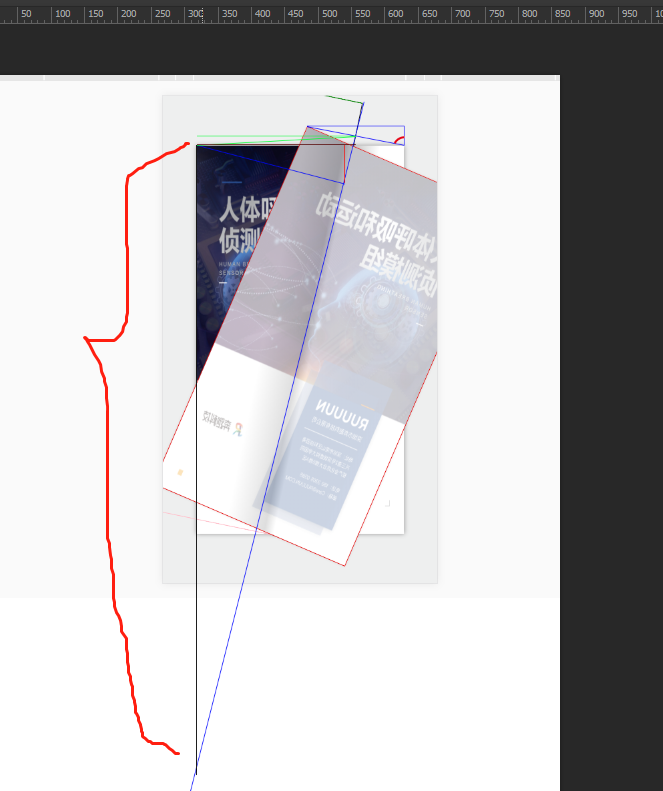


# 第一个if (alpha > A90) {

tr.x = tr.x + Math.abs(tr.y \* rel.y / rel.x);



if (Math.round(tr.x \* Math.tan(PI - alpha)) < height) {



也就是说，当折线与页面左下角相交的时候才为true.

## point.y = Math.sqrt(Math.pow(height, 2) + 2 \* middle.x \* rel.x);

# 第二个if (alpha > A90) {

执行完第一个之后，进入第二个。

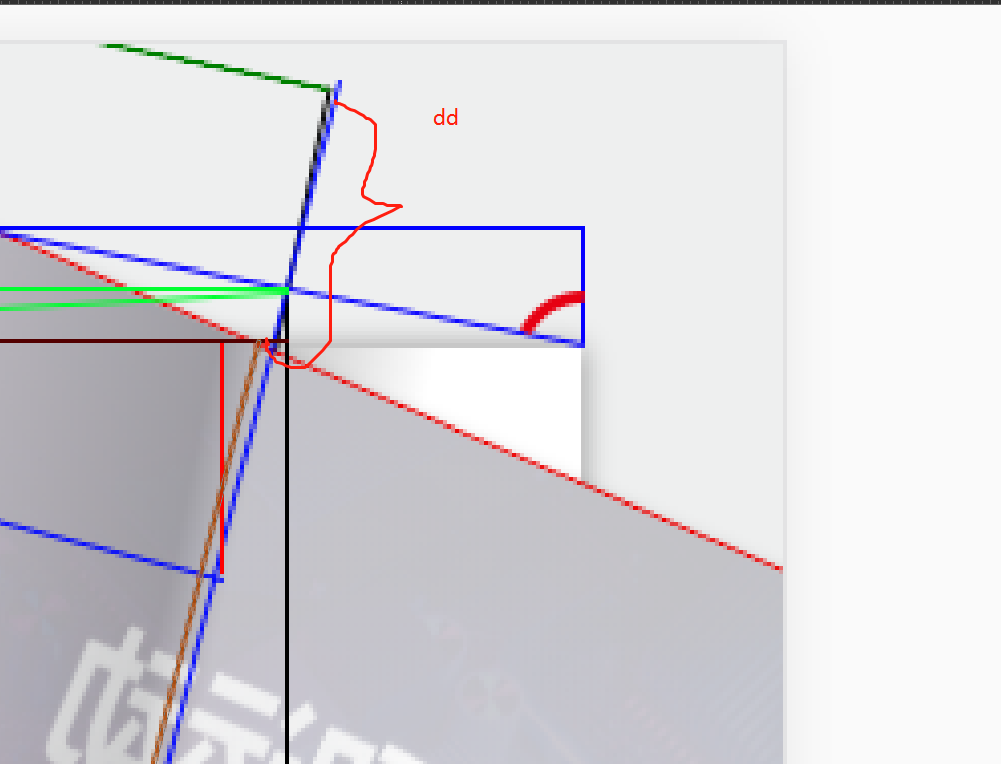
## var beta = PI - alpha, dd = h - height / Math.sin(beta);

h：data.wrapper.height(),

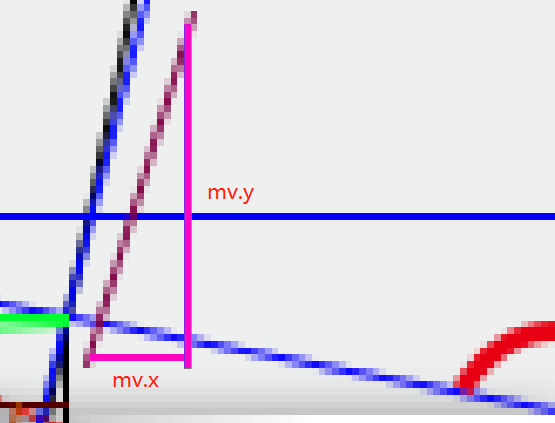
height / Math.sin(beta)：



dd = h - height / Math.sin(beta);



## mv = point2D(Math.round(dd \* Math.cos(beta)), Math.round(dd \* Math.sin(beta)));



top = point.corner.substr(0, 1) == 't',

left = point.corner.substr(1, 1) == 'l',

if (left) mv.x = - mv.x;

if (top) mv.y = - mv.y;

# 阴影变化

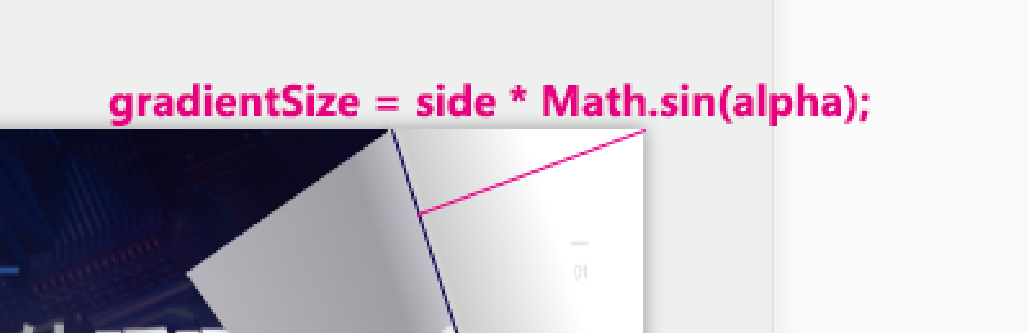
function gradient(obj, p0, p1, colors, numColors) {



-webkit-gradient(type,x1 y1, x2 y2, from(开始颜色值), [color-stop(位置偏移-小数,停靠颜色值),...],to(结束颜色值));

## 阴影A gradientEndPointA

gradientSize = side \* Math.sin(alpha);



endingPoint = flipMethods.\_c2.call(that, point.corner),

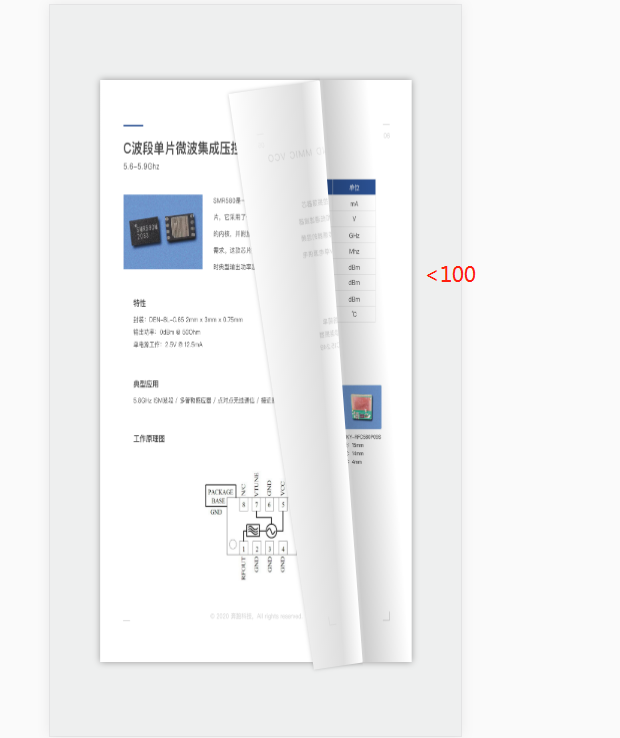
对于tr: endingPoint:(-width,0)

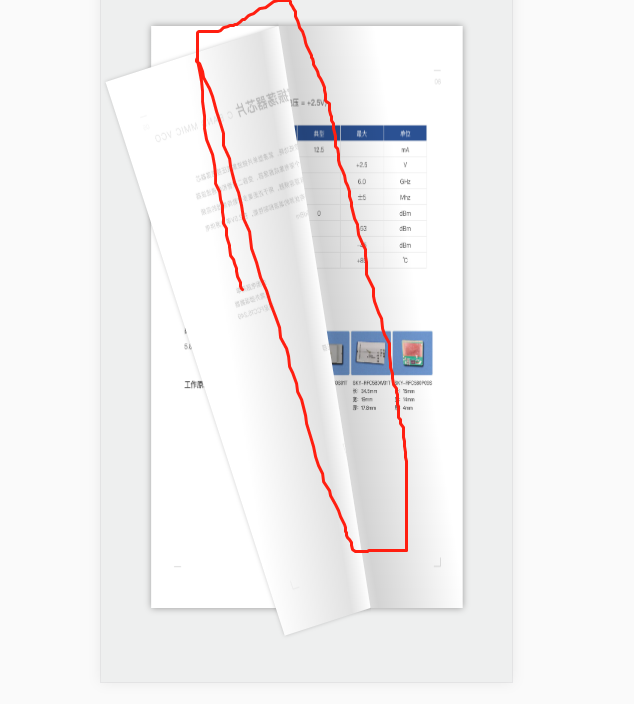
far = Math.sqrt(Math.pow(endingPoint.x - point.x, 2) + Math.pow(endingPoint.y - point.y, 2)) / width;

far 废弃了，不需要

gradientStartVal = gradientSize > 100 ? (gradientSize - 100) / gradientSize : 0;

颜色渐变线的开始位置，百分比数值，这里的100，是渐变颜色的最小宽度，如果翻折的尺寸大于这个100，需要计算超出100的占比作为颜色渐变线的开始位置。

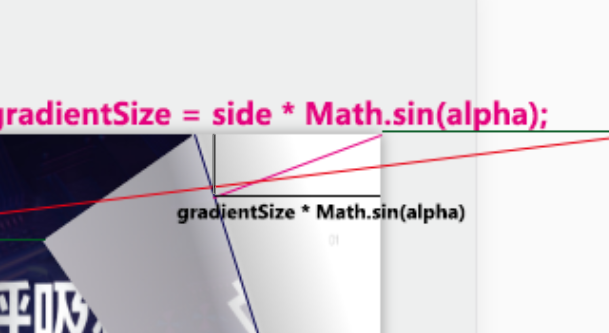




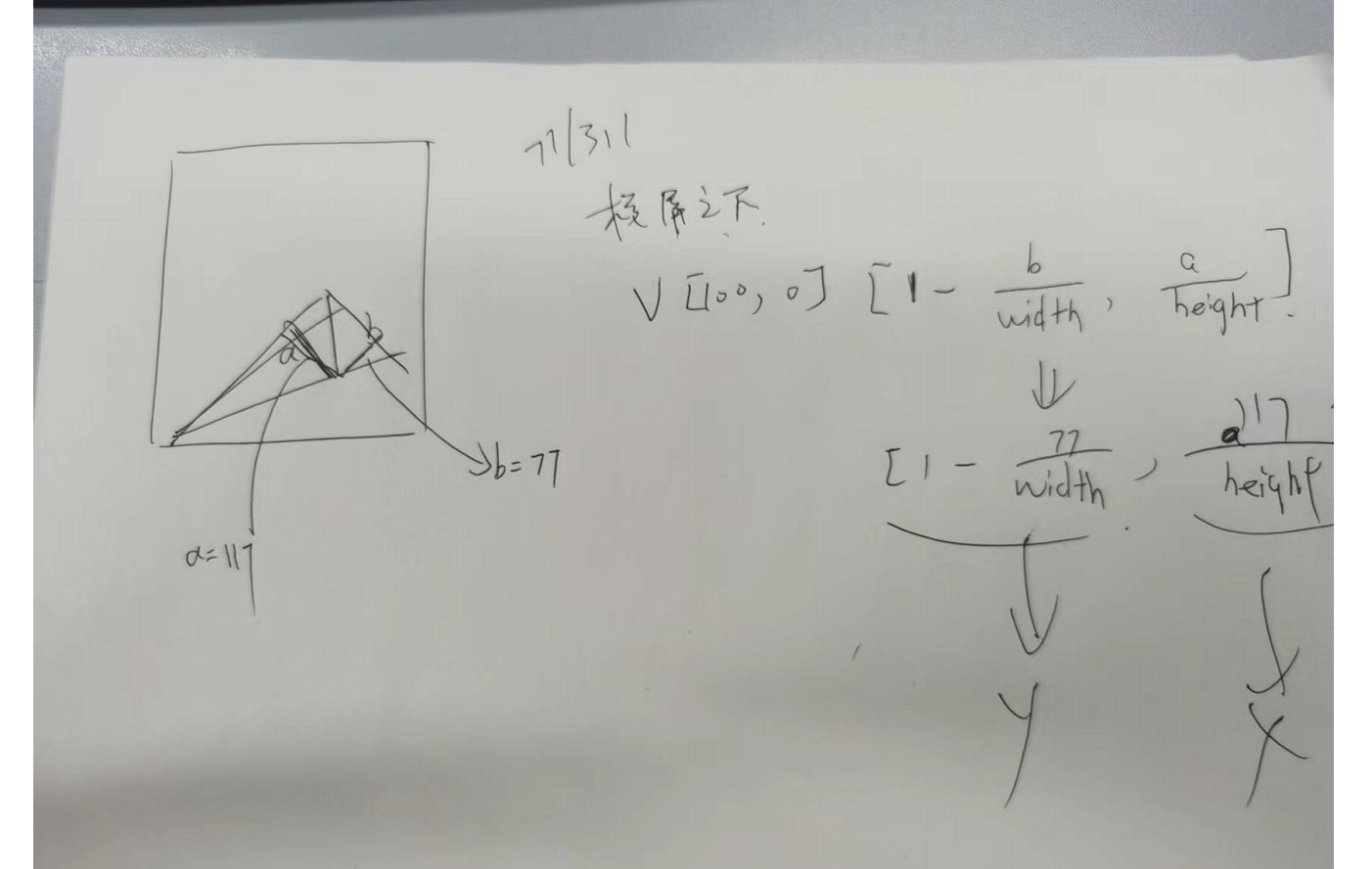
gradientEndPointA = point2D(

gradientSize \* Math.sin(alpha) / width \* 100,

gradientSize \* Math.cos(alpha) / height \* 100);

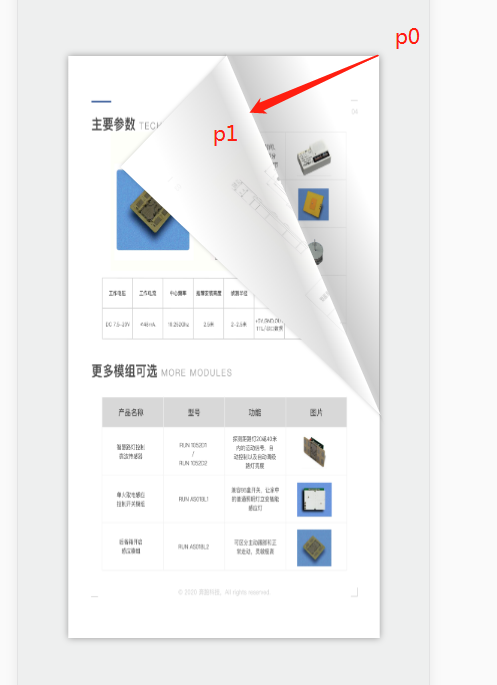


横屏情况下：



### 背面图像翻转修正：

## 阴影B gradientEndPointB

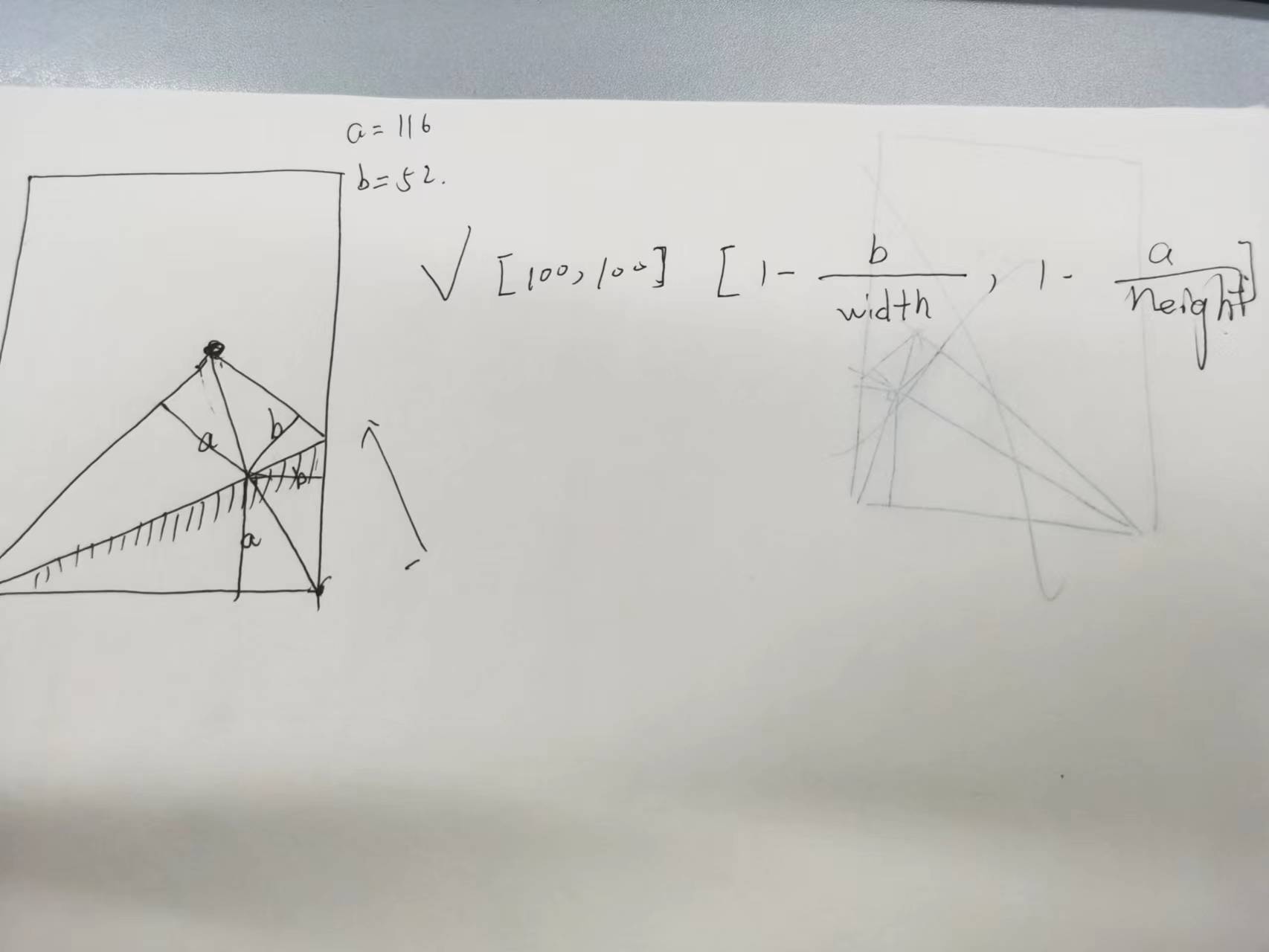


gradientEndPointB = point2D(

gradientSize \* 1.2 \* Math.sin(alpha) / width \* 100,

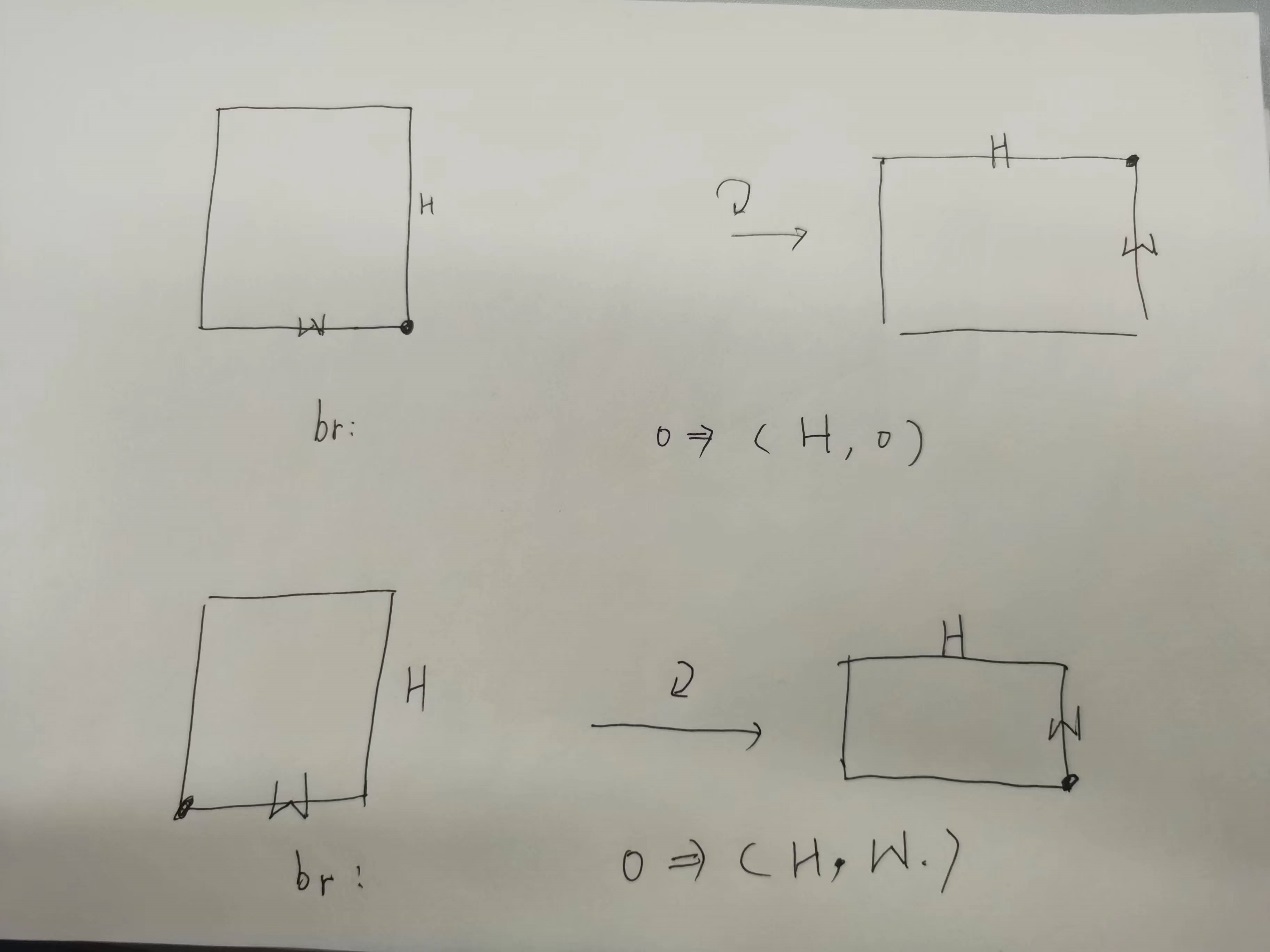
gradientSize \* 1.2 \* Math.cos(alpha) / height \* 100);

横屏情况下：

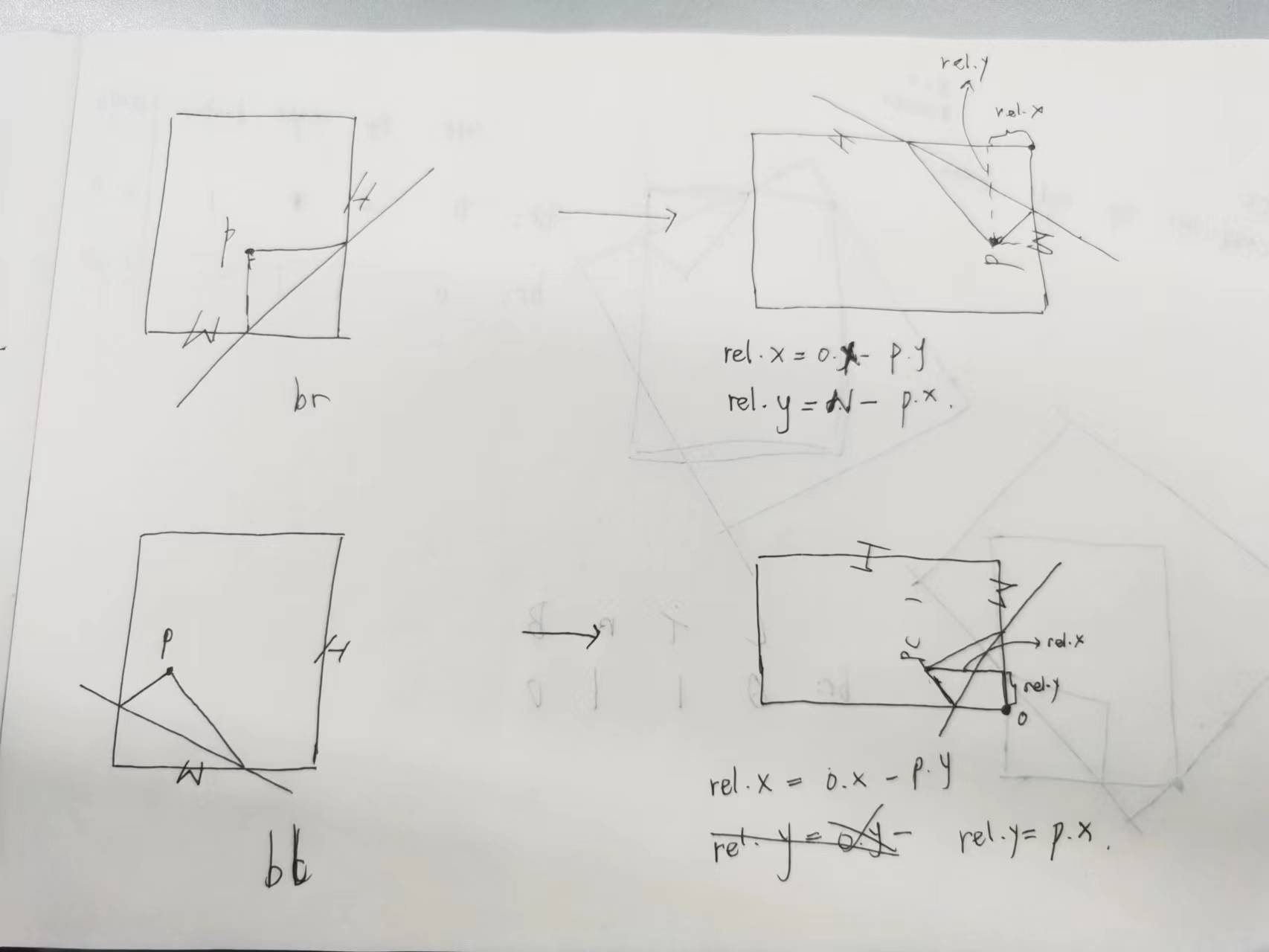


# 最终横屏处理结论

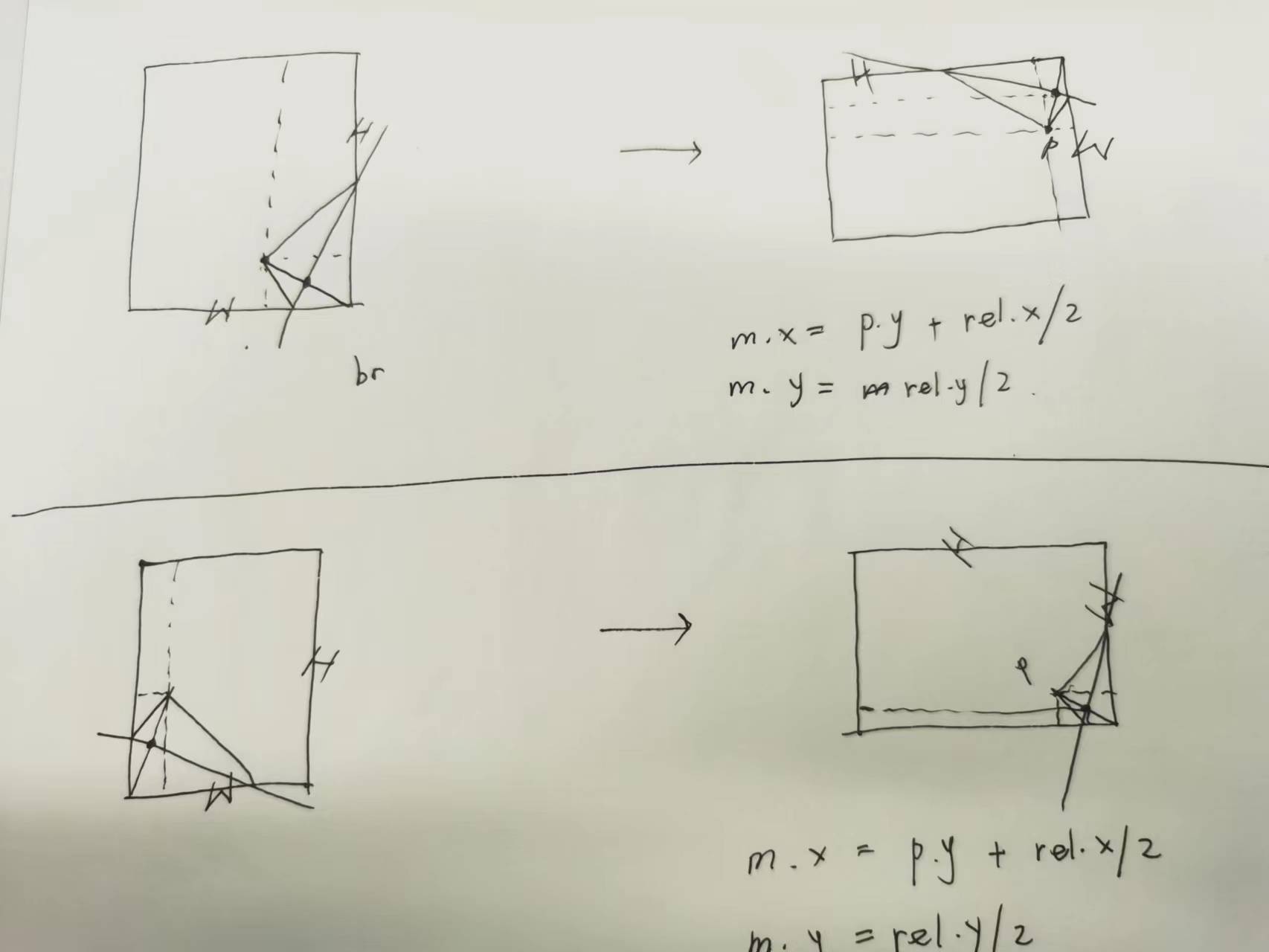
## 横屏处理：o折角原点要逆时针翻转



## 横屏处理，rel的计算：



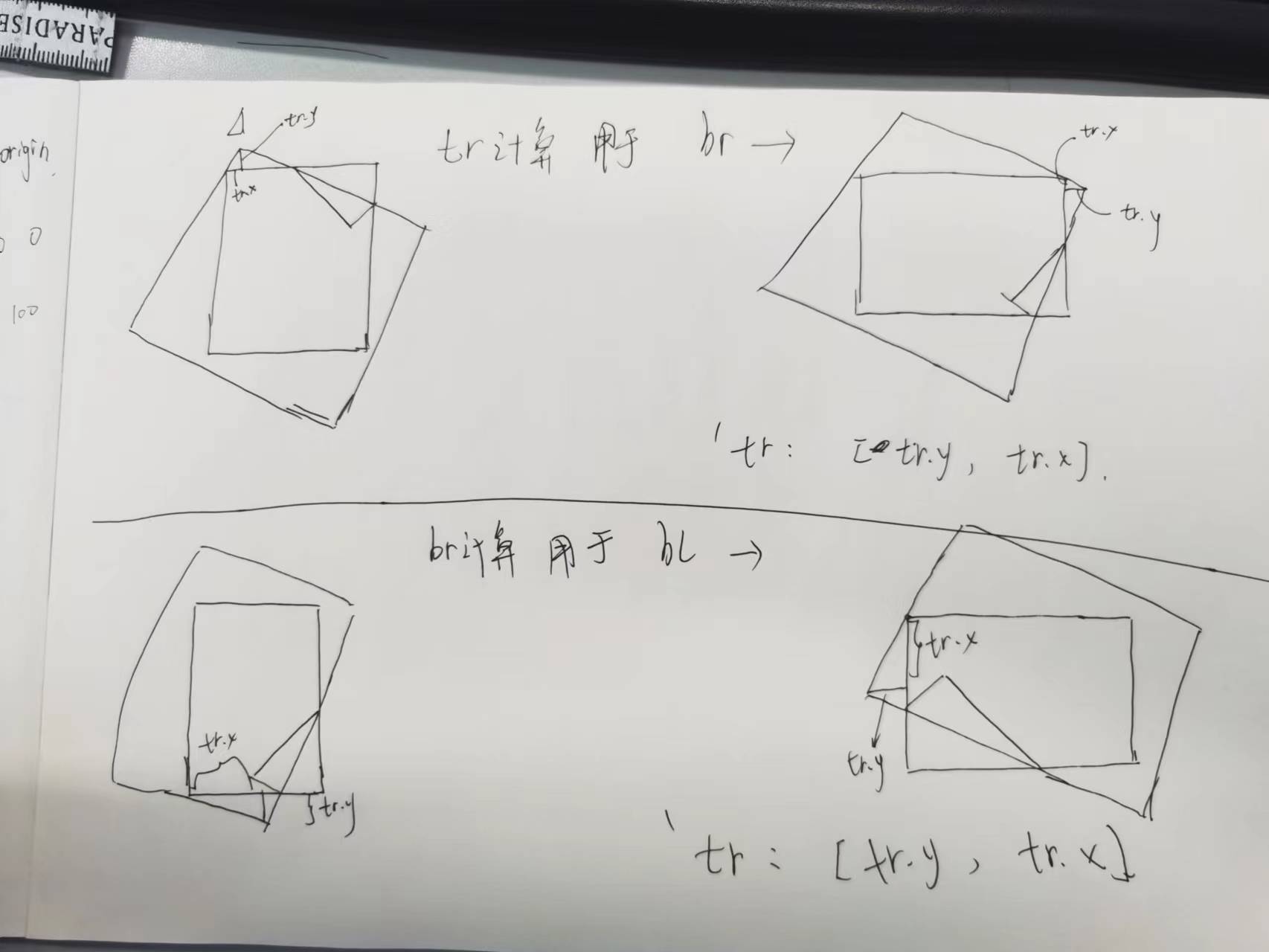
## 横屏处理，middle计算



## 横屏处理：>90度时，height -> width top->left left->top

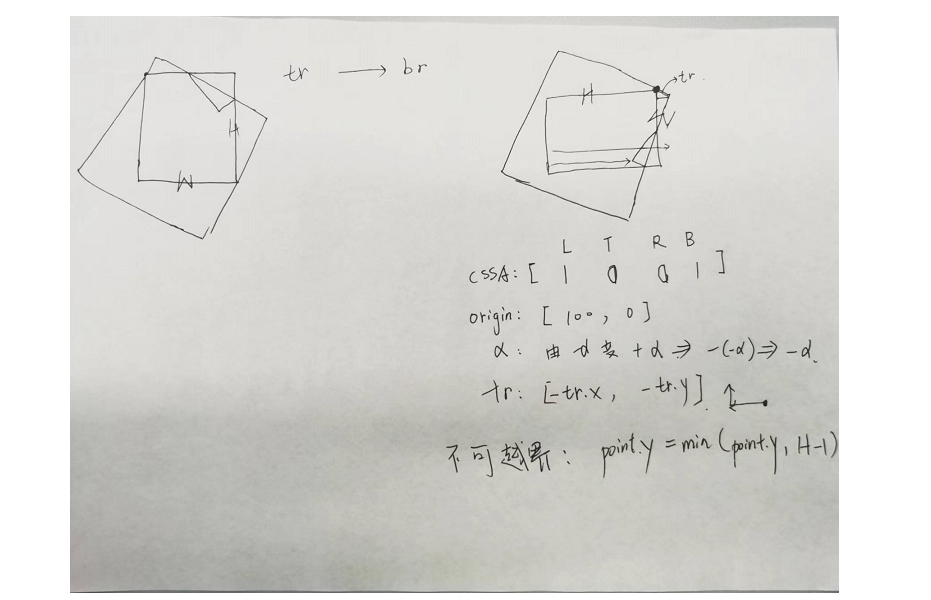
## 横屏处理：df计算，width与height对换，然后left 或者 top 遍历一下可以得到最终值

## 横屏处理，源tr计算用于目标tr:



## 横屏处理，transform传参处理：

tr->br:



br->bl:

