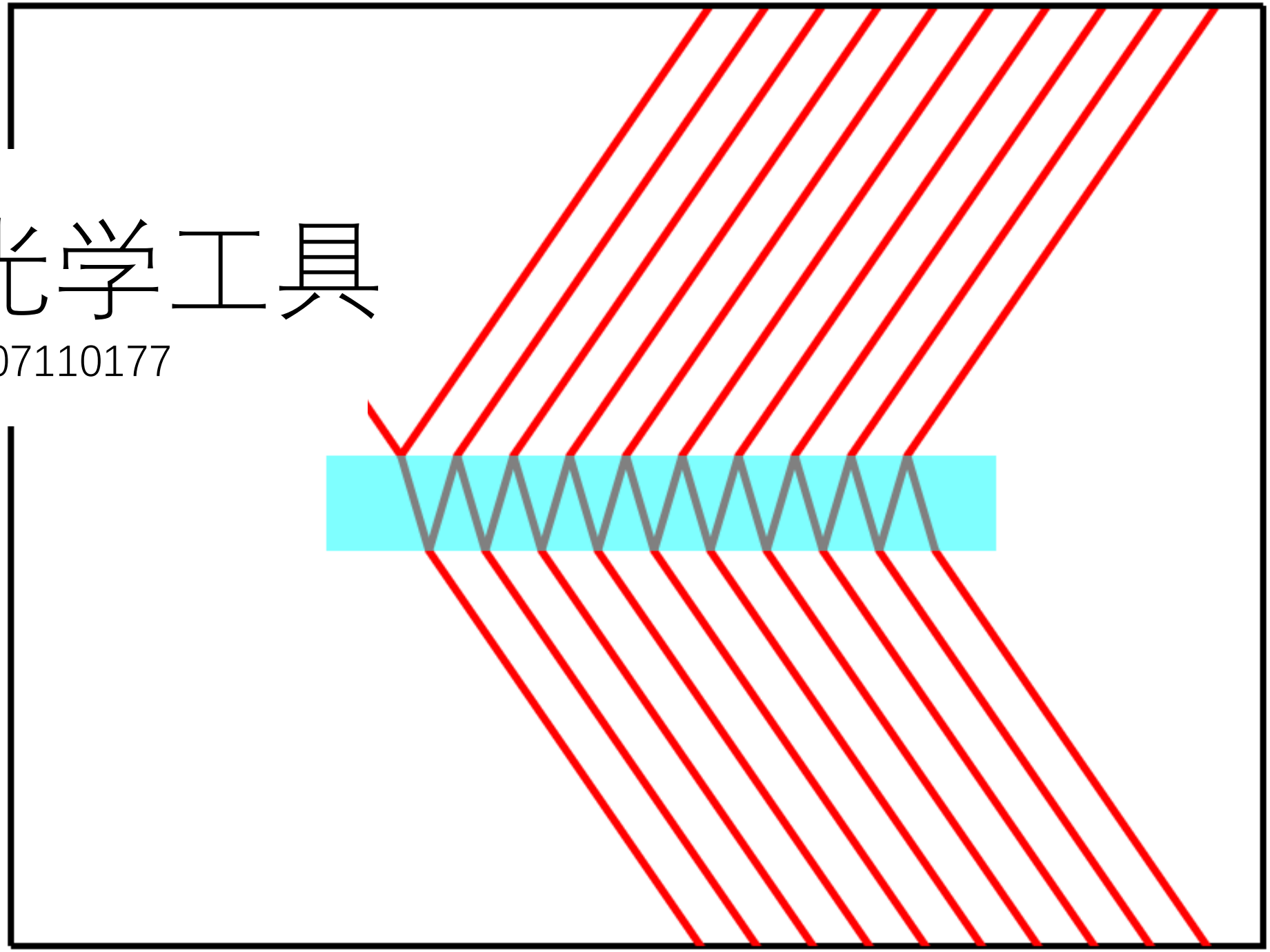
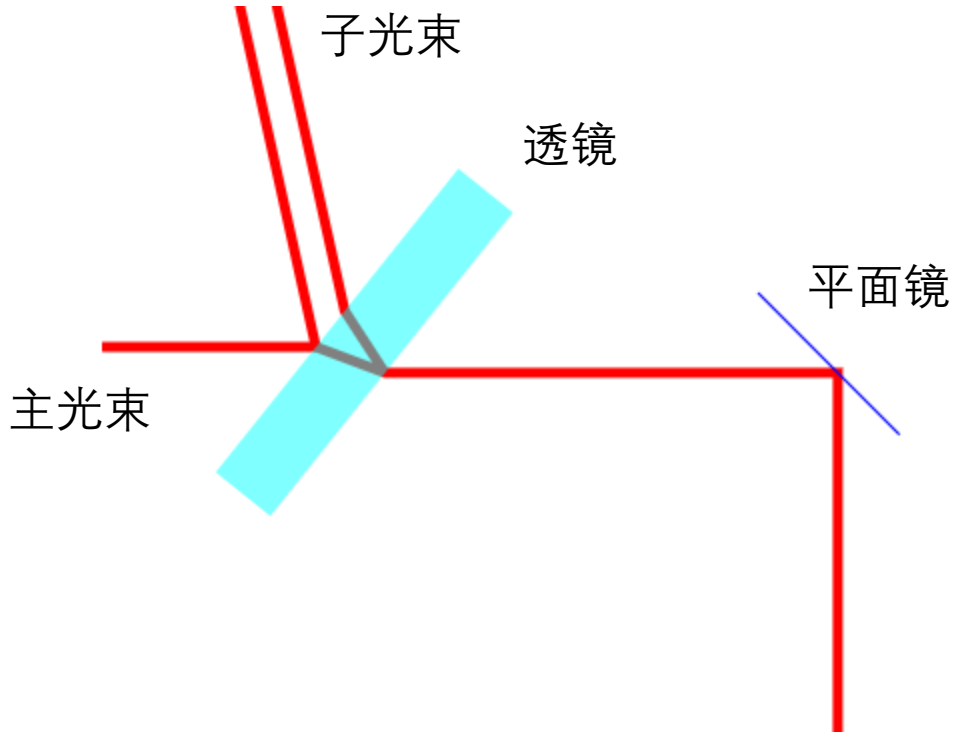
2D几何光学工具

魏雨轩 17307110177



思路——面向对象编程

- 光对象：light
- 子光对象：childlight
- 透镜对象：lens
- 平面镜对象：mirro



光对象light.js——class light

实体属性：

- Type : ”light”;
- length:
- Color:
- Begintheta
- Width
- Childlight列表

控制类属性：

- Anchor（锚点）
- Transferable
- Rotatable

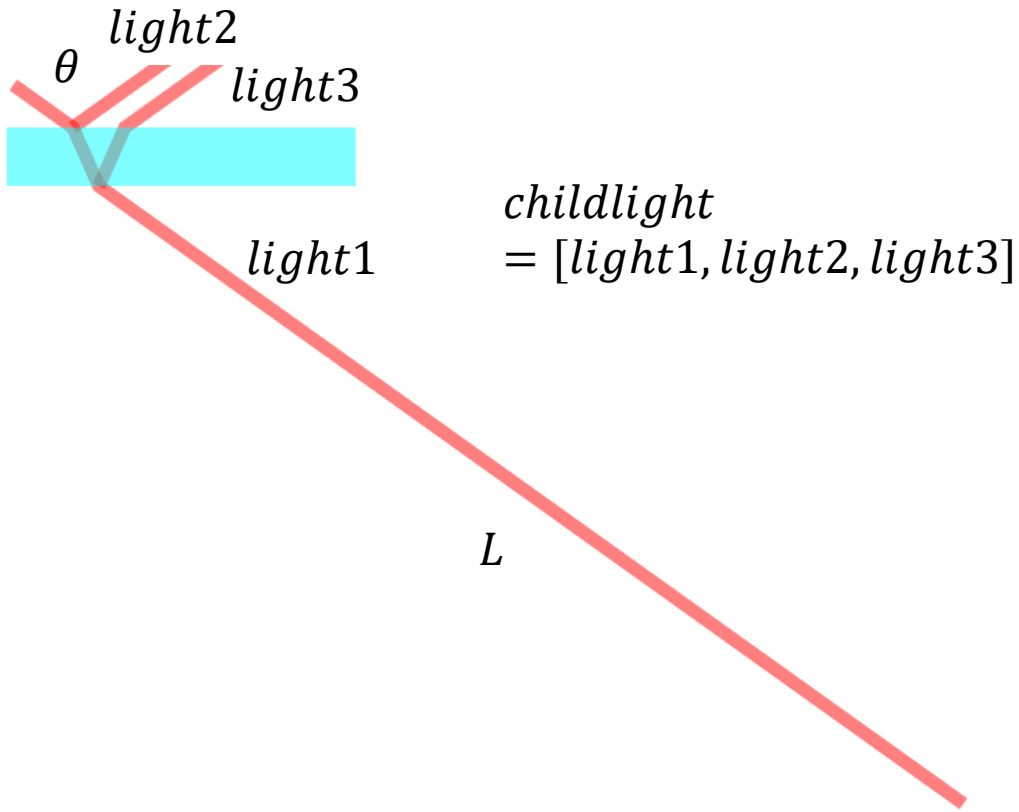
对象方法：

- Addchild, addchilds, deletechild（子光操作）
- Draw（绘图）
- Init（初始化）
- transfer
- Rotate
- Show_info（信息汇总）

光对象class light

部分实体属性：

- Length：光线末端延长量，一般默认很大，可以看成是射线。
- Begintheta：初始光线角度。
- childlight：子光线列表，子光线各自独立传播，都从主光线衍生出来。每发生一次折射增加一束子光线。
- Color：rgba
- Width：宽度



子光对象light.js——class childlight

实体属性：

- path（光路径）
- end（路径末端坐标）
- Theta（路径末端倾角）
- N（路径末端所处折射率）
- Length（继承光对象）
- Width（继承光对象）
- Color（继承光对象）

对象方法：

- Draw（绘图）
- Addpath（添加路径）
- Reflect（输入法线，反射点，得到更新后的反射光线）
- Refract（输入法线，折射点，折射率，得到更新后的折射光线，如果发生全反射，则调用reflect函数）

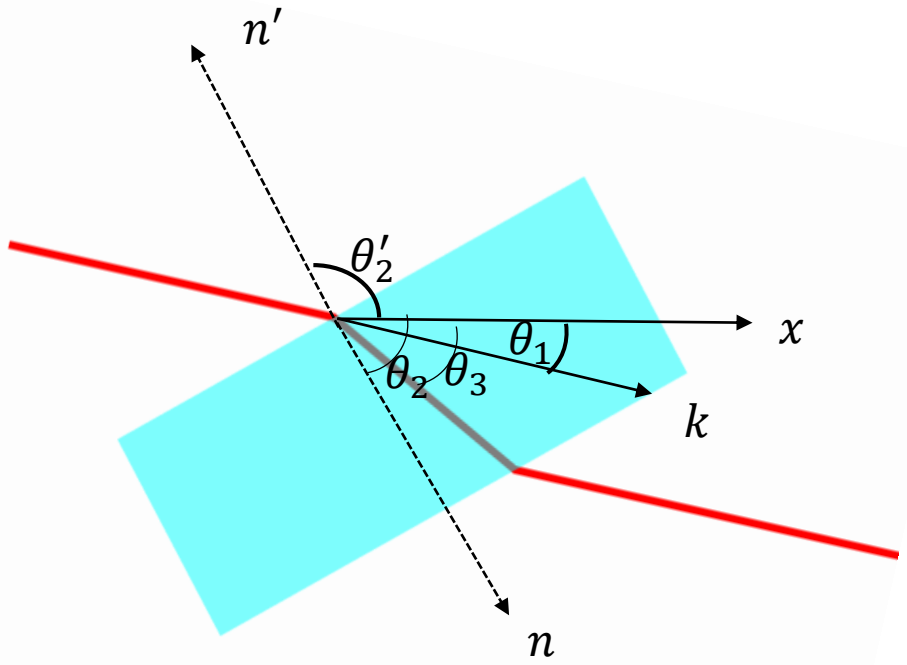
子光对象class childlight——refract

我们很容易写出：

$$\theta_i = \theta_1 - \theta_2$$

$$\theta_r = \arcsin(\frac{\sin(\theta_i)\, n_1}{n_2})$$

$$\theta_3 = \theta_2 + \theta_r$$



光线的角度具有2π的周期性，而法线的角度具有π的周期性，还有计算机在求反三角函数的时候取值范围是$(-\frac{\pi}{2},\frac{\pi}{2})$，如果这些问题不注意，会使得折射光线计算发生错误。

子光对象class childlight——refract

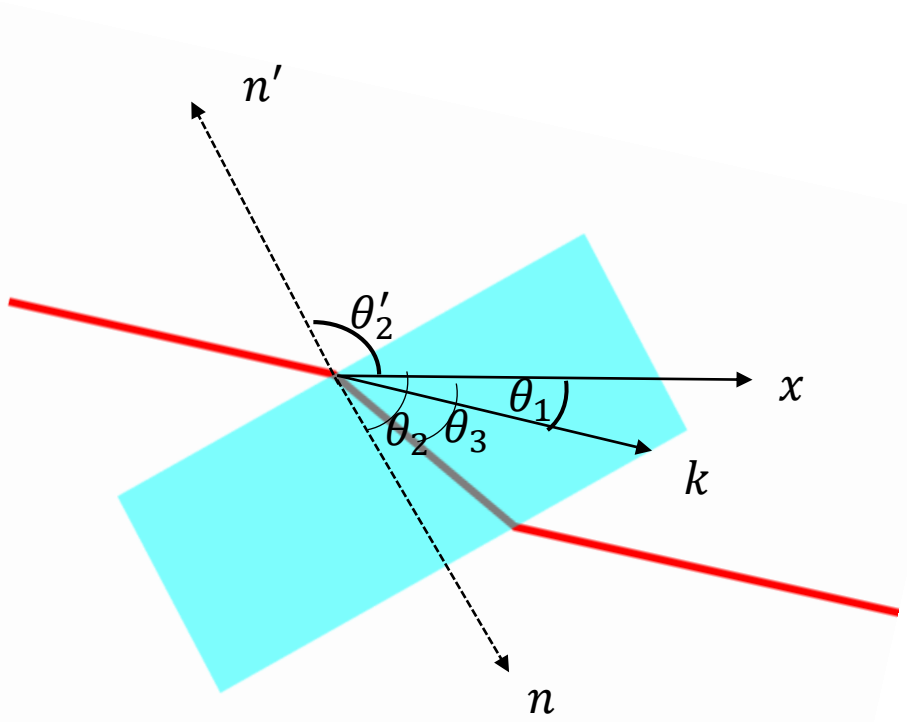
需要把各个角的周期规范化，为了与反三角函数取值对应，把入射角的取值范围设置为$(-\frac{\pi}{2},\frac{\pi}{2})$，使用同余的手段:

$$\theta_i = \left(\theta_1 + 2k\pi - \theta_2 + k'\pi + \frac{\pi}{2}\right)\%(\pi) - \frac{\pi}{2}$$

之后再规范法线的角度:

$$\theta_2 = \theta_1 - \theta_i$$

反三角函数求出$(-\frac{\pi}{2},\frac{\pi}{2})$的$\theta_r$后，使用调节后的法线角度生成出射光线的角度。



透镜类lens.js——class lens

实体属性

- Type:lens
- N（折射率）
- Color
- Xlist（边界横坐标列）
- Ylist（边界纵坐标列）

控制类属性

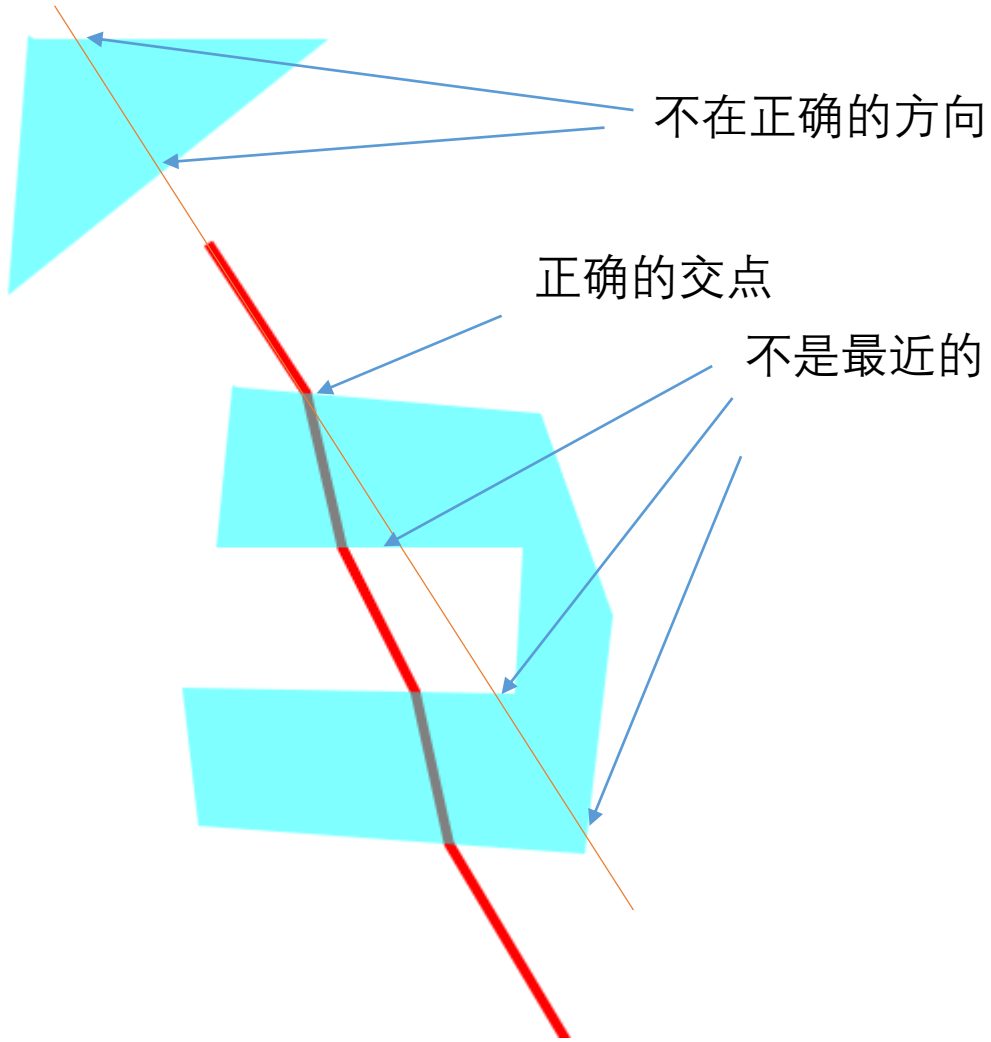
- anchor（锚点）
- Transferable
- Rotatable

对象方法

- Buildfacet, buildfacet2（构建一般边界）
- Buildcircle（构建圆边界）
- Draw
- Transfer
- Rotate
- Checkcross（输入子光线，得到最近的边界交点）
- Show_info

透镜类class lens

Checkcross函数，输入子光线，通过调用子光线的路径末端的点位置和路径末端的光线的方向，得到光线直线表达式，然后计算出和透镜的交点几何，去除不正确的方向的交点，然后寻找最近的交点，返回交点的值，法线方向，折射率。而光线可以处理这些信息，得到反射和折射后的光线。



平面镜类lens.js——class mirros

实体属性

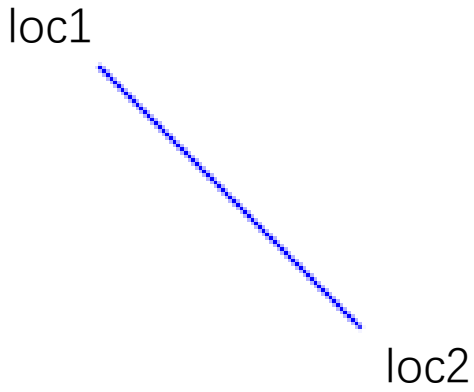
- Type
- Loc1,loc2
- Color

控制类属性

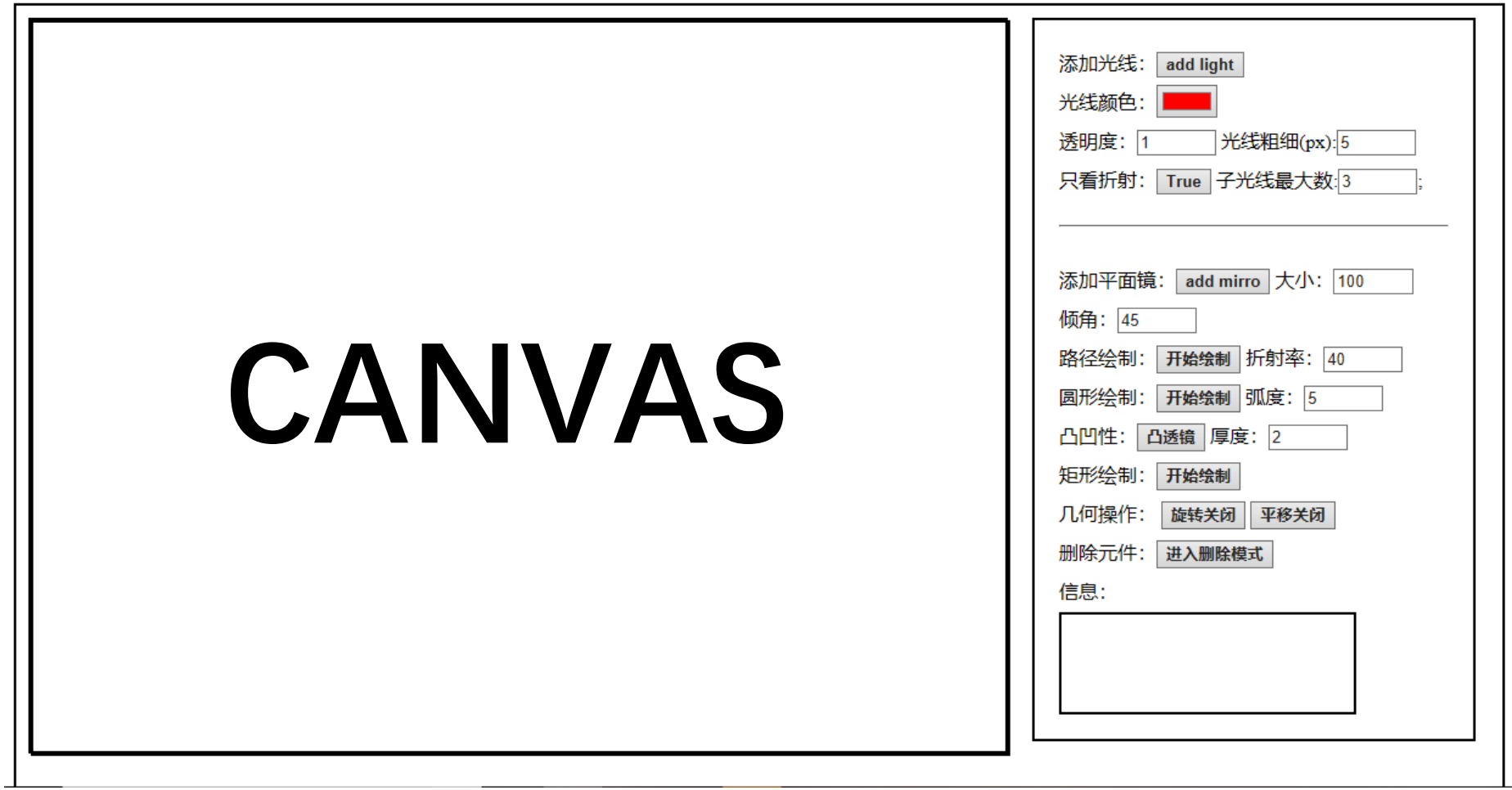
- Anchor
- Transferable
- Rotable

对象方法

- Draw
- Transfer
- Rotate
- Checkcross
- Show_info



主程序——light2D.html



添加光线，光线参数，显示模式

添加光学元件即参数，绘制光学元件，删除元件，光学元素的信息

主程序——light2D.js

实体对象列表
- Light_set
- Mirro_set
- Lens_set

控制参数:
- Refract_only
- Is_transfer
- Is_rotate
- Rotate_ref（旋转锚点）
- Is_paint
- Is_circle
- Cpoint（绘制圆心锚点）
- Angle_info（弧度控制参数）
- Is_concave
- Locp（临时储存锚点）
- Is_rect
- Rect_p（矩形锚点）
- Is_delete

主程序——light2D.js

- 清除画布
- 初始化光线
- 光路计算，直到光路不再改变
- 绘制所有光学元件
- 绘制所有临时参考线
- 回调主函数

```
function main(){
    context.clearRect(0,0,canvas.width,canvas.height)
    init();
    while (light_on()){
        light_on();
    }
    draw_all(context);
    draw_tem(context);
    requestAnimationFrame(main);
}
main();
```

主程序——light2D.js——Light_on()

计算光路程序思路：

- 对所有光线
- 对所有子光线
- 所有元件对子光线做交叉检测，返回最近的那个点。
- 光线折射或反射。
- 如果中途有反射或折射，则返回changed=true，否则为false，循环运行这个程序直到返回false

```
function light_on(){
    var changed = false;
    // 这是一个自动检测所有光学器件的函数，思路是，找出最近的那个，反射或者透射
    // 反射和透射都要算进去，有反射就一定有透射，关于反射率和透射率的计算和光的偏振有关
    // 所以之后可能要加入相位信息，偏振信息，之类的。
    var d = function(p1,p2){
        return Math.abs(p1.x-p2.x)+Math.abs(p1.y-p2.y)
    };
    for (var i=0;i<light_set.length;i++){
        // 每根子光线才能操作 每根子光线的操作只有两类
        /*
        1，反射，不产生新的子光线
        2，折射，必然产生新的子光线，以折射光为原光线的延伸，反射光为新的子光线
        */
        var child_num = light_set[i].childlight.length;
        var beginset = new Array(),thetaset = new Array();nset = new Array();
        for (var ic=0;ic<child_num;ic++){
            // 对每一条子光线操作，
            //console.log(light_set[i].childlight[ic])

            var p0 = light_set[i].childlight[ic].end;
            var theta = light_set[i].childlight[ic].theta;
            let next_ele,next_cross,next_normal,next_action,next_n;
            // 这里一定要用let，否则不会初始化，前面的子光线会都后面的子光线的赋值造成影响。
```

主程序——light2D.js——Light_on()

其中定义了四个参数作为计算下一步光路的参考：

Next_ele:下一个元件

Next_cross:下一个交点

Next_normal:下一个交点对应的法线

Next_action:下一个元件的种类

对于每一束子光线，依次循环镜子类，透镜类，确定四个参数的值，然后根据四个参数，对光线做折射或反射，如果打开了允许折射反射同时出现的开关，则最后一步要把新产生的子光线添加到对应光线的子光线列表中去。最后返回changed

```
    for (var ic=0;ic<child_num;ic++){
        // 对每一条子光线操作，
        //console.log(light_set[i].childlight[ic])

        var p0 = light_set[i].childlight[ic].end;
        var theta = light_set[i].childlight[ic].theta;
        let next_ele,next_cross,next_normal,next_action,next_n;
        // 这里一定要用let，否则不会初始化，前面的子光线会都后面的子光线的赋值造成影响。

        for (var i_m=0;i_m<mirro_set.length;i_m++){…
        }
        for (var i_l=0;i_l<lens_set.length;i_l++){…
        }
        //console.log(next_cross)
        if (next_action === "mirro"){…
        }
        else if (next_action === "lens"){…
        }
    }
    if (!refract_only){…
    }

}
return changed
// 如果变化了那么继续作图
```

主程序——light2D.js——事件响应

这些分别对应着：

添加光线，添加镜子，开启折射反射模式，开启平移，开启旋转，开启任意路径绘制透镜，开启圆形路径绘制透镜，绘制凸透镜还是凹透镜，开启矩形绘制路径，开启删除模式

```
add_light.onclick = function(){…
}

add_mirro.onclick = function(){…
}

ROb.onclick = function(){…
}

transfer.onclick = function(){…
}

rotate.onclick = function(){…
}
// 这样操作保证了不存在两个同时开启的状况，关闭
paint.onclick = function(){…
}

circle.onclick = function(){…
}

concave.onclick = function(){…
}

rect.onclick = function(){…
in_delete.onclick = function(){…
}
```

主程序——light2D.js——mouse_on_ele

此函数的作用是，输入某个坐标值，判断其出于哪类光学类上，返回对应的光学类。

平面镜类：由于镜子的厚度很小，使用一个窄椭圆作为判定区域。

透镜类：利用isPointInPath来判断是否在透镜中。

光线类：判断离光源处的距离，是否小于某一定值。

```
function mouse_on_ele(loc){


    for (var i_m=0;i_m<mirro_set.length;i_m++){…
    }
    for (var i_l=0;i_l<lens_set.length;i_l++){…
    }
    for (var i=0;i<light_set.length;i++){…
    }
}
```

light2D.js———canvas.onmousemove()

以五个模式分为五类

- 平移和旋转的几何操作
- 路径绘制透镜操作
- 圆形绘制透镜操作
- 矩形绘制透镜操作
- 删除操作

```
canvas.onmousemove = function(event){
    var loc = {
        x:event.offsetX,
        y:event.offsetY
    }
    if (!is_paint&&!is_circle&&!is_delete&&!is_rect){···
    }
    else if(is_paint){···
    }
    else if(is_circle){···
    }
    else if(is_rect){···
    }
    else if(is_delete){···
    }


}
```

light2D.js——canvas.onmousemove()

平移操作的思路

- 当鼠标移动到某元件上时，鼠标改为point。
- 当平移模式打开时，捕捉到onmousedown，则改变对象的锚点anchor为当前点击的点，把对象设置为transferable=true。
- 鼠标再次移动时，把新的位置和锚点的位置对比，得到平移矢量，通过对象的transfer函数对对象的坐标平移。并把anchor更新为当前点。
- 捕捉到onmouseup，把对象设置为transferable=false，不再平移。

light2D.js———canvas.onmousemove()

旋转操作思路:

- 在旋转模式打开时，点击对象，则设置对象锚点anchor为当前点击对象，把对象的rotatable=true，之后锚点不再改变，而在绘制过程中，锚点用一个小圆点显示出来，表示旋转操作的圆心。
- 之后移动鼠标后，再次点击对象之后，检测到onmousedown，这时把当前点的值赋给rotate_ref，这个点和平移的anchor功能类似，存储上一个鼠标所在的位置，当鼠标移动时，求这次移动的旋转角，调用对象的rotate方法，使得对象绕着anchor旋转，之后更新rotate_ref。
- 当检测到onmouseup，则把rotate_ref清空，暂停旋转。再次点击可以赋值rotate_ref重新开始旋转。
- 如何更改转动中心anchor，可以点击anchor对应的小黑点，取消anchor，之后便可以重新设置anchor。

light2D.js——canvas.onmousemove()

路径绘制透镜操作思路:

- 在路径模式开启的情况下，点击canvas上的任意一点，把值传递给paintset，之后每点击一次，paintset增加一个点。
- 在paintset的点数大于1时，主循环中的drawtemp()开始起作用，绘制已经选择的paintset中的点和鼠标当前对应的点，鼠标对应的点传递给locp。
- 当paintset点数大于3，在点击距离起始点足够近的区域后，判定为封闭，则完成绘制，把对应的paintset传递给lens_set，并初始化。

light2D.js———canvas.onmousemove()

圆形绘制透镜思路:

- 在圆形绘制模式开启时，选定canvas上某点，传递给Cpoint，作为圆心，之后把鼠标当前值传递给locp。
- 主程序中的drawtemp()，根据所选的弧度，做出参考线，并且弧度参数传递给angle_info
- 当鼠标再次点击后，通过angle_info，和凹凸性，厚度，通过lens的buildcircle，建立弧形的路径，可以得到凹透镜和凸透镜。

矩形绘制思路:

- 和其他类似，传递绘制参数rect_p作图。

删除模式：点击某个对象，把这个对象从相应列表中删除。