魔術方塊之架構

step.1 分成三個大類別

1. **APP**
   - 最上層
   - 初始化場景、相機、動畫管理、**MouseEventManager** 、**Cube)**
2. **MouseEventManager**
   - 管理滑鼠事件
   - 對畫面監聽**mouse**事件，並取得旋轉需要的參數
3. **Cube**
   - 初始化**27**塊方塊，位置設置 **for x (-1,0,1) for y (-1,0,1) for z (-1,0,1)**
   - 管理方塊的旋轉

step.2 監聽mouseDown事件

```
onMouseDown() {
    if (this.getIsAnimating()) {
        return
    }
    const mouseCoordInScene = { x: this.mouseXY.x, y: this.mouseXY.y }
    this.tempContainerForMouseDown.mouseStartPosition = mouseCoordInScene
    const intersectObjects = this.getIntersectObjects(mouseCoordInScene)
    if (intersectObjects.length === 0) {  // outside cube
        this.startControl()
        return
    }
    const clickedCube = intersectObjects[0]?.object
    if (!intersectObjects[0] || !clickedCube) {
        return
    }
    const cubePositionType = Cube.getCubePositionType(clickedCube.position)
    if (cubePositionType === 'center') {
        return
    }
    const axisOfTouchedFace = Cube.getAxisByPointPosition(intersectObjects[0].point)  //which face clicked, x , y, z
    this.tempContainerForMouseDown.axisOfTouchedFace = axisOfTouchedFace
    this.tempContainerForMouseDown.positionOfTouchedCube = clickedCube.position
    const bindDoCubeRotateWhenMouseUp = this.doCubeRotateWhenMouseUp.bind(this)
    this.domEl.addEventListener('mouseup', bindDoCubeRotateWhenMouseUp, { once: true })
}
```

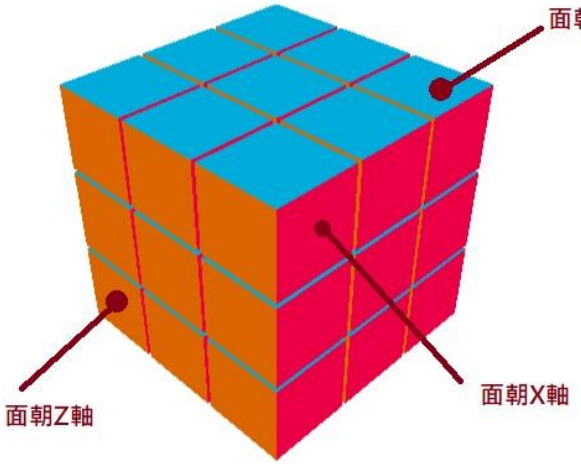
1. 若動畫中，則無視。

2. 取得滑鼠在場景中的座標。
3. 儲存按下時的座標，為了之後計算拖曳向量。

4. 從滑鼠座標映射到場景，取得接觸的物件，若無接觸到物件**=>**滑鼠座標在方塊之外，則監聽滑鼠向量來改變視角。

5. 藉由接觸到的點座標，取得。**mouseDown**到的方塊面的軸，以及接觸到的方塊座標**(**圖一**)**。

6. 當**mouseUp**時，執行**doCubeRotateWhenMouseUp**。



圖一、從哪個軸**mouseDown**的

step.3 取得rotateData

```
doCubeRotateWhenMouseUp() {
    const mouseCoordInScene = { x: this.mouseXY.x, y: this.mouseXY.y }
    const { mouseTrackvector } =
        this.getMouseTrackVector(mouseCoordInScene, this.tempContainerForMouseDown.mouseStartPosition as IPosition)
    const rotateData = this.getCubeRotateData(mouseTrackvector)
    this.app.cube.doRotate(rotateData)
}
```

1. 取得**mouseUp**時的座標。
2. 計算**mouseDown**到**mouseUp**的向量。
3. 計算旋轉所需資料**(**詳細從**3.1**開始**)**。

```
getCubeRotateData(mouseTrackvector: TH.Vector2) {
    const axisLineVectorToScreen = this.getAxisLineVectorToScreenVector(this.app.camera)
    const excludeAxis = this.tempContainerForMouseDown.axisOfTouchedFace
    const touchedCubePosition = this.tempContainerForMouseDown.positionOfTouchedCube as IPosition
    const targetAxisToRotate = this.getAxisToRotate(axisLineVectorToScreen, mouseTrackvector, excludeAxis as TAxis)
    const rotateData = {
        axis: targetAxisToRotate.axis,
        axisPosition: touchedCubePosition[targetAxisToRotate.axis],
        direction: targetAxisToRotate.direction
    } as IDoRotate
    return rotateData
}
```

3.1 計算**XYZ**軸線映射至螢幕的向量。
3.2 去除**mouseDown**的軸**(step2**之**5**。
3.3 計算要旋轉的軸**(**詳細從**3.3.1**開始。
3.4 回傳資料:
axis : 要旋轉的軸
axisPosition : 旋轉第幾層
direction : 方向**(**順逆時針**)**

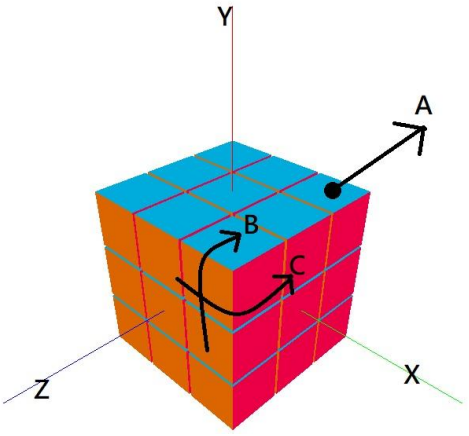
```
getAxisToRotate(axisVector: {
    x: TH.Vector2;
    y: TH.Vector2;
    z: TH.Vector2;
}, mouseTrackVector: TH.Vector2, excludeAxis: TAxis) {
    const dotData = [
        { axis: 'x', dotAbs: 0, cross: 0 },
        { axis: 'y', dotAbs: 0, cross: 0 },
        { axis: 'z', dotAbs: 0, cross: 0 }
    ] as { axis: TAxis, dotAbs: number, cross: number }[]
    const init = dotData
        .filter(d => d.axis !== excludeAxis) //axis of clicked face not rotate
        .map(item => {
            const dot = mouseTrackVector.dot(axisVector[item.axis])
            const dotAbs = Math.abs(dot)
            const cross = mouseTrackVector.cross(axisVector[item.axis])
            return {
                axis: item.axis, dotAbs, cross
            }
        })
    const sortAbs = init.sort((item1, item2) => item1.dotAbs - item2.dotAbs) // find min abs of dot product => the most vertical axis
    const target = sortAbs[0]
    const axis = target.axis
    const direction = target.cross > 0 ? 'counterclockwise' : 'clockwise'
    return {
        axis,
        direction
    }
}
```

3.3.1 參數有三個
axisVector : (同**3.1)**
mouseTrackVector : down到**up**的向量
excludeAxis:根據**step2**之**5**

3.3.2 首先刪除**mouseDown**的軸。
3.3.3 計算**mouseTrackVector**與**XYZ**軸之間的**cos**的絕對值，以及**sin**。
3.3.4 根據**cos**的絕對值中最小的軸，找到要旋轉的軸。
3.3.5 根據要旋轉的軸的**sin**的值來決定順逆時針。
3.3.6 圖解如下頁。

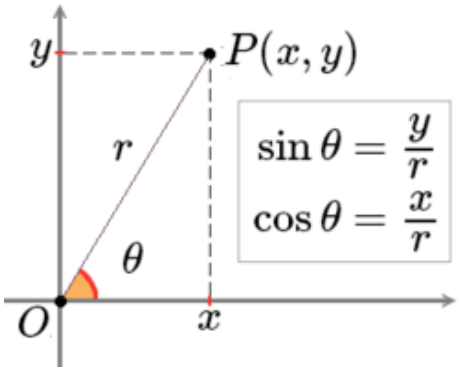
step.3-2 圖解

1.為何要把接觸面的軸，在計算旋轉軸的過程中除去？



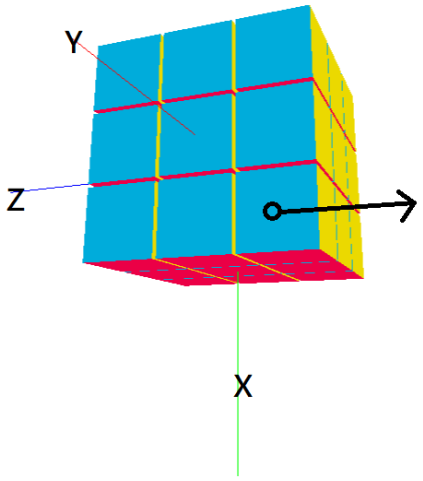
若滑鼠以A向量拖曳，則會面臨到一個問題，
該以B方式旋轉X軸，還是以C方式旋轉Y軸？
選擇較簡單的方式，接觸面的Y軸直接捨去。

2.為何要用與XYZ軸的cos的最小絕對值計算旋轉軸？



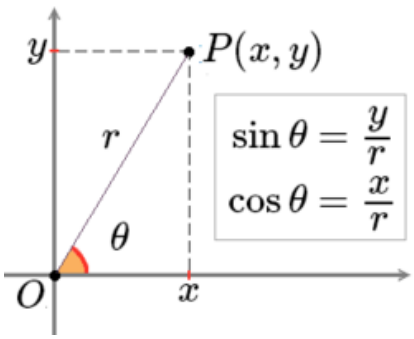
由左圖可知，cos絕對值越小則兩條線越垂直，從力矩來說越垂直越有效率，最符合現實的出力方式，因此以最垂直的軸當旋轉軸，此外，three.js有內建函式可計算cos。

(來源、維基百科)



若以此圖為例，XYZ軸映射到螢幕的向量簡單假設為X軸(0,-1) Y軸(-1,1) Z軸(-1,0)，可明顯看出滑鼠軌跡與X軸最垂直，因此選擇X軸當作旋轉軸，旋轉x=1的九個方塊。

3.如何決定順逆時針？



若把r當作滑鼠向量，則由-y到+y(sin為正數)為順時針繞X軸，由+y到-y(sin為負數)為逆時針繞X軸。

step.4 選取要旋轉的九個方塊，並準備開始動畫

```
doRotate(doRotateData: IDoRotate) {
    this.setIsAnimating(true)
    const targetCubeGroup = this.getCubeGroup(doRotateData.axis, doRotateData.axisPosition)
    if (targetCubeGroup === 'error') {
        this.setIsAnimating(false)
        return
    }
    this.animateRotate(targetCubeGroup, doRotateData.axis, doRotateData.direction)
}
```

4.1 根據**step3-3.4**的資料，選擇旋轉軸上第**N**層的九個方塊，並把此九個方塊加入一個群組準備動畫。

step.5 開始動畫

```
animateRotate(cubeGroup: TH.Group, axis: TAxis, direction: TDirection) {
    const deg = (Math.PI / 2) * (direction === 'clockwise' ? -1 : 1)
    const mixer = new TH.AnimationMixer(cubeGroup)
    const time = [0, animateTime]
    const rotateVector = axis === 'x' ? new TH.Vector3(1, 0, 0) : axis === 'y' ? new TH.Vector3(0, 1, 0) : new TH.Vector3(0, 0, 1)
    const rotateQ = new TH.Quaternion().setFromAxisAngle(rotateVector, deg)
    const originQ = cubeGroup.quaternion
    const rotateKF = new TH.KeyframeTrack('.quaternion', time, [originQ.x, originQ.y, originQ.z, originQ.w, rotateQ.x, rotateQ.y, rotateQ.z, rotateQ.w])
    const clip = new TH.AnimationClip('rotateGroup', animateTime, [rotateKF])
    const action = mixer.clipAction(clip)
    this.setMixer(mixer)
    const whenFinished = () => {
        const arr = [] as TH.Object3D<TH.Event>[]
        cubeGroup.children.forEach(c => {
            const wp = c.getWorldPosition(new TH.Vector3())
            const { x, y, z } = this.fixPosition(wp)
            c.position.setX(x)
            c.position.setY(y)
            c.position.setZ(z)
            const wq = c.getWorldQuaternion(new TH.Quaternion())
            const fixedQ = this.getFixedQuaternion(wq)
            c.quaternion.x = fixedQ.x
            c.quaternion.y = fixedQ.y
            c.quaternion.z = fixedQ.z
            c.quaternion.w = fixedQ.w
            arr.push(c)
        })
        this.scene.add(...arr)
        this.scene.remove(cubeGroup)
        mixer.removeEventListener('finished', whenFinished)
        this.setIsAnimating(false)
        this.cleanMixer() //TODO
    }
    mixer.addEventListener('finished', whenFinished)
    action.setLoop(TH.LoopOnce, 1)
    action.play()
}
```

5.1 根據順逆時針決定要旋轉**-90**還是**90**度。

5.2 初始化動畫用混合器**mixer**。

5.3 根據旋轉軸選擇旋轉的中心向量，根據此向量設置四元數。

5.4 監聽動畫結束時回調，並開始動畫。

5.5.whenFinished細節如下：

5.5.1 由於旋轉的物件是包含九個方塊的群組，而不是九個方塊本身，如果此時把九方塊從群組加到世界座標則會出錯**(**因為方塊本地座標仍在原地**)**。

因此在加回世界座標之前，需先設置世界座標跟世界旋轉。

此外，因**JS**小數精確度的原因，雖然**step 5.3**時，設置了旋轉**90**度的四元數，但並非精確地旋轉**90**度，因此需要修正。

5.6 最後，把方塊從群組加回世界即完成。