1.9 三维坐标、变换

大多数 React 的 UI 组件是根据 layout 布局自动放置的，这些 layouts 以向右为 X 正方向，Y 向下，在其

他的语言中左上角(0,0),右下角(宽，高)。

在 3D 坐标中这些方向就要变化了，也就有了自己的坐标系

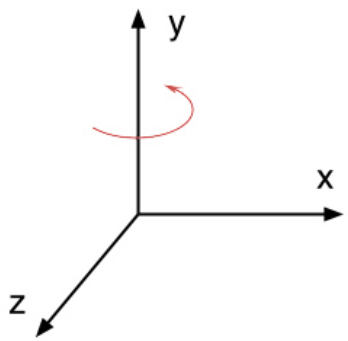
React VR 用的是 the OpenGL® 3D 坐标系

下面介绍 3D 坐标系：

- UI组件可以放置在2D平面内，但是`<View>`可以转换并放在3D空间内;
- 个别组件可以相对于度组件转换、移动、旋转;
- 每一个单独的组件都可以放在3D空间的绝对位置上.

React VR 用的是右手 OpenGL 坐标系统：右手大拇指指向一个轴的正方向，其他手指弯曲的方向就

是正旋的方向。



不像在 React 里面的，Y 轴向上为正，React 是向下为正。

Z 轴指向用户，默认用户站在远点的位置，也就是说朝着 Z 轴负方向望去。要是把物体放在负距离的

位置上如-3 米，如果开始时是可以看到的。下面有一个而例子，通过 **transform** 属性做的转换，后面

再详细讨论转换这个话题。

```
<Text style={{ position: 'absolute', transform: [{translate: [0, 0, -6]}],
    layoutOrigin: [0.5, 0.5] }}
  fontSize={<span className='red'>0.5</span>} text='Office Lobby' />
```

单位

距离、长度单位都是米

旋转角度单位是度

变换

变化可以在 3D 空间内放置各式各样的组件，React VR 把 React 的转换样式扩展到 3D 空间，下面是

我们的一个 mesh 例子的代码：

```
<Mesh style={{
  transform: [{translate: [0, -30, -300]},{ scale : 0.1 },
        {rotateY : this.state.rotation}, {rotateX : -90} ] }}
  source={{url:'./creature/', mesh:'creature.obj', mtl:'creature.mtl',
```

```
  lit: true}}
/>
```

转换在 style 内是以数组的形式存在的，而且执行的顺序是从最后一个到第一个，倒叙执行的：

```
style={{
  transform: [
    {rotateZ : this.state.rotation},
    {translate: [0, 2, 0]}
    { scale : 0.1 },
  ],
}}
```

上面例子中，scale 先执行，其次是 translate，最后是 Z 轴旋转；

最终的结果是沿着 Y 轴移动了 2 米，

然后绕着 Z 轴旋转

下面的例子就和上面的完全不一样了：

```
style={{
  transform: [
    {translate: [0, 2, 0]}
    {rotateZ : this.state.rotation},
    { scale : 0.1 },
  ],
}}
```

这个是先绕着自己的原点旋转，然后在移动 2 米。

Transform 属性

Transform 属性是由一系列的键值对组成的：

```
transform: [
            {TRANSFORM_COMMAND: TRANSFORM_VALUE},
            ...
          ]
```

TRANSFORM_COMMAND 可以是下面的一个：

matrix ：是 16 位数的一个数组

- **translation**[1,0,0,0, 0,1,0,0, 0,0,1,0, Tx,Ty,Tz,1]
- **scale**[Sx,0,0,0, 0,Sy,0,0, 0,0,Sz,0, 0,0,0,1]
- **rotation**通过R值 [R00,R01,R01,0, R10,R11,R12,0, R20,R21,R22,0, 0,0,0,1]

matrix 是最灵活的改变 transforms 的方式，也让开发者用自己的 js 模块接入 React 组件。

举个例子，下面的作用就是先所有轴都缩放 0.01，然后在旋转[3,2,1],代码如下：

```
style={{
  transform: [
    {matrix: [0.01, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0.01, 0, 3, 2, 0, 1]}
  ],
}}
```

rotateX：绕着 X 轴旋转一定角度，默认的单位是度，如果要用弧度，需要加上 **rad** 单位，如**-0.5rad**。

```
style={{
  transform: [
    {rotateX: 10}
```

```
  ],
}}
```

rotateY：绕着 Y 轴旋转一定角度，默认的单位是度，如果要用弧度，需要加上 **rad** 单位，如**-0.5rad**。

```
style={{
  transform: [
    {rotateY: 10}
  ],
}}
```

rotateZ：绕着 Z 轴旋转一定角度，默认的单位是度，如果要用弧度，需要加上 **rad** 单位，如**-0.5rad**。

```
style={{
  transform: [
    {rotateZ: 10}
  ],
}}
```

scale：它的值可以是一个数字或者一个数组，如果是数字，xyz 的值都会缩放。

```
style={{
  transform: [
    {scale: 10}
  ],
}}
```

如果是数组，则长宽高会按不同的比例缩放

```
style={{
  transform: [
```

```
        {scale: [0.01, 0.02, 0.03]}
  ],
}}
```

translate：它的值是一个数组，单位是米，会沿着 xyz 的轴移动不同的距离。

```
style={{
  transform: [
    {translate: 10}
  ],
}}
```

translateX：只沿着 X 轴移动多少米。

```
style={{
  transform: [
    {translateX: 1}
  ],
}}
```

translateY：只沿着 Y 轴移动多少米。

```
style={{
  transform: [
    {translateY: 1}
  ],
}}
```

translateZ：只沿着 Z 轴移动多少米。

```
style={{
  transform: [
    {translateZ: 1}
  ],
```

```
}}
```