React VR 快速入门

前言

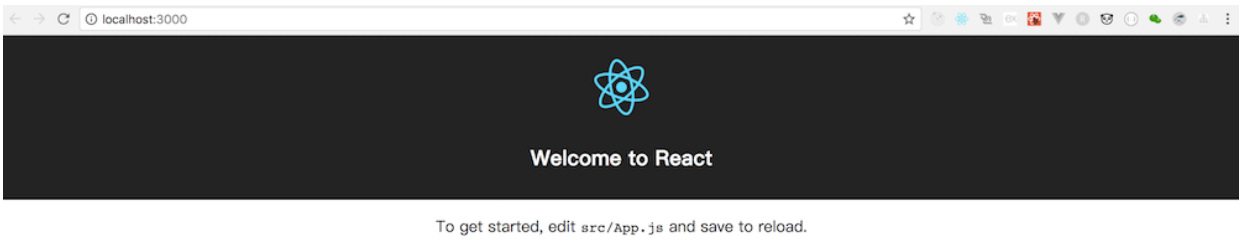
什么是React

React 是一个开放源代码的 JavaScript 库，为HTML呈现的数据提供了视图渲染。 React 视图通常使用指定的像HTML标签一样的组件来进行UI渲染。它目前是最流行的JavaScript库之一，它拥有强大的基础和庞大的社区。

创建一个React App

```
$ npm install -g create-react-app
$ create-react-app my-app

$ cd my-app
$ npm start
```


效果图



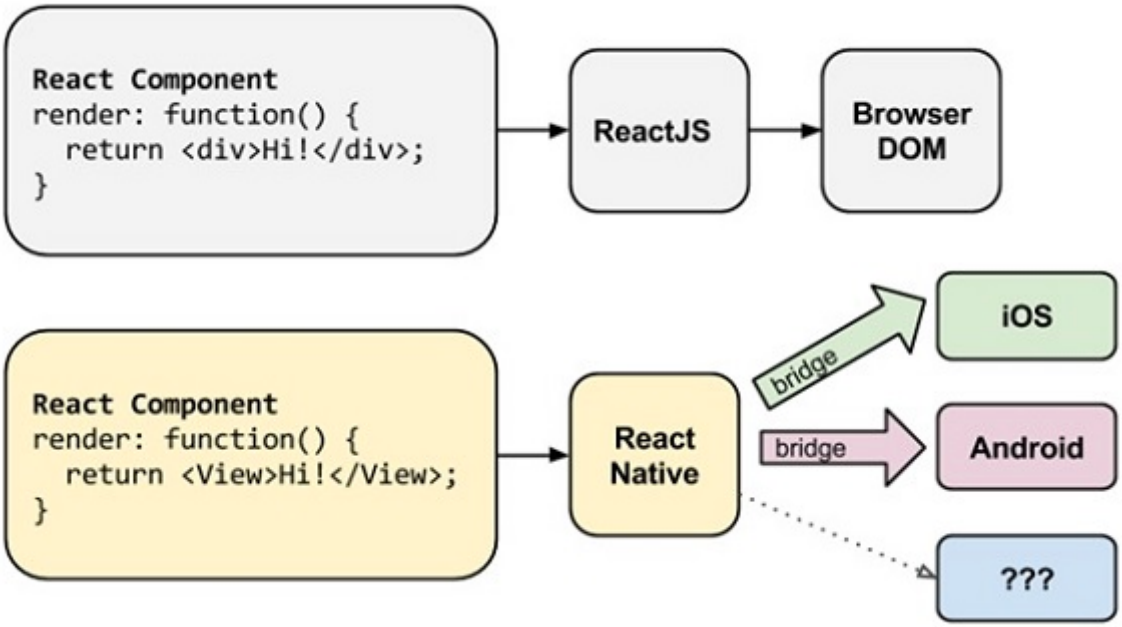
什么是React Native ?

React Native 是仅使用Javascript的移动应用构建框架。它使用与 React 相同的设计，包含丰富的UI库和组件声明。

它使用与常规iOS和Android应用程序相同的基本UI构建块。

使用 React-Native 最赞的地方是还可以通过原生的 Objective-C , Java , 或者 Swift 来构建组件。

- React VS React Native：



- React Native bridge：

创建一个React Native App

- 环境安装

查看官网 ：http://facebook.github.io/react-native/docs/getting-started.html。

- 创建项目

```
$ react-native init my-rn-app
```

- 运行项目

```
To run your app on iOS:
   cd /Users/liyuechun/Pictures/my_rn_app
   react-native run-ios
   - or -
   Open ios/my_rn_app.xcodeproj in Xcode
   Hit the Run button
To run your app on Android:
```

```
    cd /Users/liyuechun/Pictures/my_rn_app
    Have an Android emulator running (quickest way to get
started), or a device connected
    react-native run-android
```

- 效果图

iOS	Android
	

开始使用React VR

React VR旨在允许Web开发人员使用React的声明方法(特别是React Native)来创作虚拟现实（VR）应用程序。

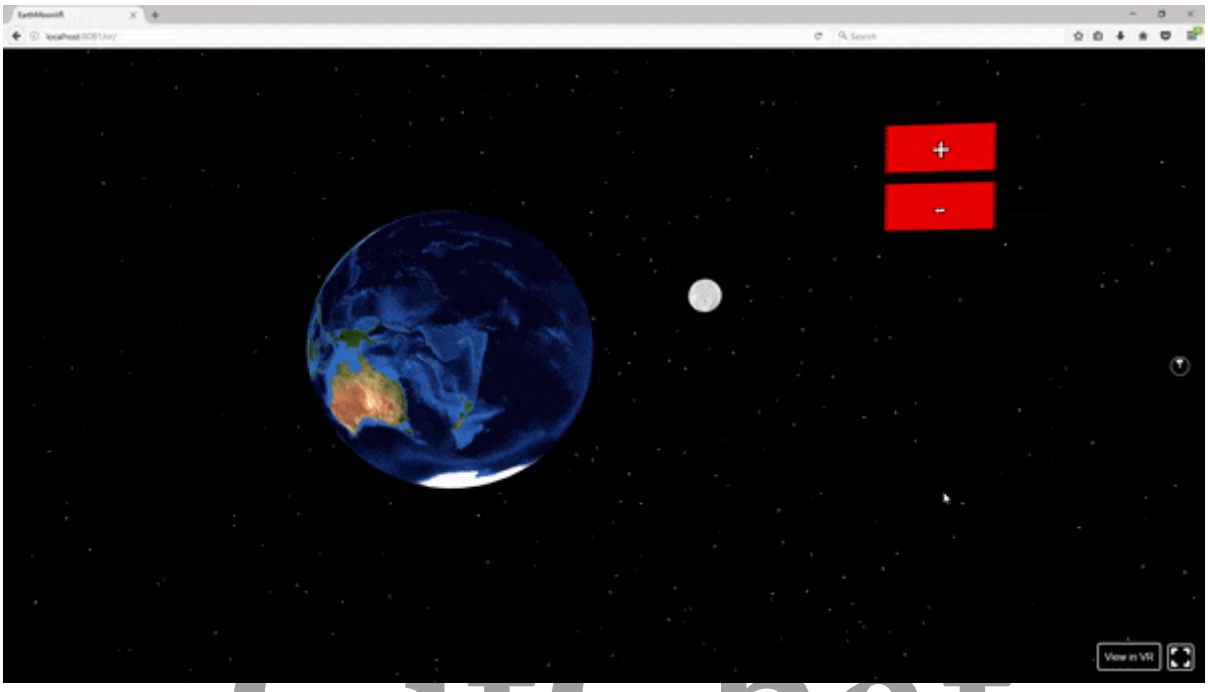
`React VR` 使用Three.js 来支持较低级别的WebVR 和 WebGL API. WebVR是用于访问Web上VR设备的API。 WebGL（Web图形库）是一种无需使用插件即可用于在任何兼容的Web浏览器中渲染3D图形的API。

React VR类似于React Native，因为它使用 `View` ， `Image` 和 `Text` 作为核心组件，并且支持Flexbox布局。 此外，React VR还将 `Pano` ， `Mesh` 和 `PointLight` 等VR组件添加相关库

中。

在本篇文章中，我将带领大家创建一个简单的VR应用程序来学习如何创建一个全景图片，3D对象模型，按钮和flexbox布局的使用场景。我们的模拟应用程序基于React VR的两个 网格 和 布局 的官方示例。

该应用程序将渲染一个能够放大和缩小的 地球 和 月球 的3D模型，效果图如下所示：



这些模型中，它们的尺度和旋转不是 地球–月球系统 的真实复制品。 这种关系只是为了展示React VR的工作原理。 与此同时，我们将解释一些关键的3D建模概念。 一旦掌握了ReactVR，就可以随意创造更多的作品。

你能够从 GitHub找到最后的项目源代码。

要求

到目前为止，虚拟现实是一项相当新的技术，开发或测试我们的VR应用程序的方法很少。

WebVR 和 Is WebVR Ready? 可以帮助您了解哪些浏览器和设备支持最新的VR规范。

但是你也不必过于担心， **你现在不需要任何特殊的设备** ，例如： Oculus Rift, HTC Vive, 或者 Samsung Gear VR 来测试一个WebVR APP。

下面是你现在所需要准备的：

- 一台Windows / Mac电脑。
- Google浏览器。
- 最新版本的Node.js

如果您也有Android设备和Gear VR耳机，您可以安装Carmel Developer Preview浏览器来探索您的React VR 应用程序。

创建项目

首先，我们需要使用 NPM 来安装 React VR CLI 工具：

```
$ npm install -g react-vr-cli
```

使用React VR CLI来创建一个名字叫做 EarthMoonVR 的新项目：

```
$ react-vr init EarthMoonVR
```

在创建过程中您需要等一会儿，这将创建一个 EarthMoonVR 目录，目录里面就是相关项目文件，如果希望创建速度快一些，您可以安装 Yarn 来提高速度。

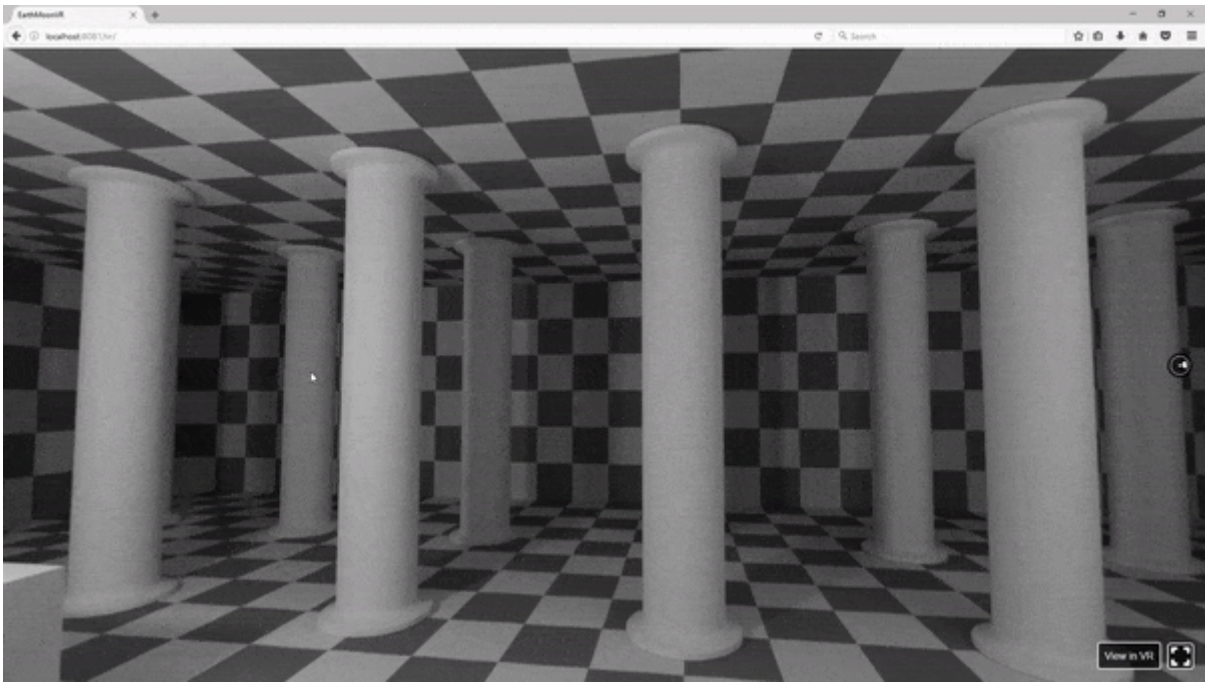
一旦项目创建完毕，可以通过 cd 切换到 EarthMoonVR 文件路径下面：

```
$ cd EarthMoonVR
```


在终端通过 npm start 来启动程序以便查看效果：

```
$ npm start
```

在浏览器中输入 http://localhost:8081/vr 。稍微等一会儿，即可查看到VR效果：



下面是初始化的React VR新项目的项目结构:

```
+-__tests__
+-node_modules
+-static_assets
+-vr
\-.babelrc
\-.flowconfig
\-.gitignore
\-.watchmanconfig
\-index.vr.js
\-package.json
\-rn-cli-config.js
\-yarn.lock
```

我将 `index.vr.js` 文件呈现高亮状态，它包含了您应用程序的源码， `static_assets` 目录包含像图片和3D模型的外部资源文件。

你可以从这里了解更多项目结构。

`index.vr.js` 文件的内容如下:


```
import React from 'react';
import {
  AppRegistry,
  asset,
  StyleSheet,
  Pano,
  Text,
  View,
} from 'react-vr';

class EarthMoonVR extends React.Component {
  render() {
    return (
      <View>
        <Pano source={asset('chess-world.jpg')}/>
        <Text
          style={{
            backgroundColor:'blue',
            padding: 0.02,
            textAlign:'center',
            textAlignVertical:'center',
            fontSize: 0.8,
            layoutOrigin: [0.5, 0.5],
            transform: [{translate: [0, 0, -3]}],
          }}>
```

```
            hello
          </Text>
        </View>
      );
    }
};

AppRegistry.registerComponent('EarthMoonVR', () => EarthMoonVR);
```

我们可以看到React VR使用了ES2015 和 JSX。

这个代码通过React Native packager进行预编译，它提供了(ES2015, JSX)编译和其他资源加载。

在 `render` 函数中， `return` 了一个顶级标签，里面包含：

- `View` 组件, 它是可以包含其他所有组件的容器组件。
- `Pano` 组件, 用于将(`chess-world.jpg`)渲染一张360度的图片。
- `Text` 组件, 用于渲染字体的3D空间。

注意， `Text` 组件通过一个内联的样式对象来设置样式。在React VR中的每一个组件都有一个 `style` 属性来控制控制它的外观和布局。

除了添加 `Pano` 或 `VrButton` 等特殊组件之外，React VR还使用了与React和React Native相同的概念，例如组件，属性，状态，生命周期，事件和弹性布局。

最后，项目根组件应该通过 `AppRegistry.registerComponent` 来进行注册，以便App能够进行打包和正常运行。

现在我们知道代码是做什么用的，接下来我们将 全景图片 拖拽到项目中。

全景图像

通常，我们的VR应用程序中的空间由全景（pano）图像组成，它创建了一个1000米的球体（在React VR距离中，尺寸单位为米），并将用户置于其中心。

一张全景图像允许你从上面，下面，后面以及你的前面去观察它，这就是他们也被称为360的图像或球面全景的原因。

360全景图有两种主要格式：**平面全景图**和**立方体**。 React VR支持两者。

平面全景图

平面全景图由宽高比为2:1的单个图像组成，意味着宽度必须是高度的两倍。

这些照片是通过360度照相机创建的。一个很好的平面图像来源是Flickr，你打开这个网站尝试搜索 equirectangular 关键字，例如：我通过 equirectangular 关键字尝试搜索就找到这张图片:

看起来很奇怪，不是吗？

无论如何，下载最高可用分辨率的照片，将其拖拽到项目中 static_assets 的路径下面，并且修改 render 函数中的代码，如下所示：


```
render() {
    return (
      <View>
        <Pano source={asset('sample_pano.jpg')}/>
      </View>
    );
  }
```

Pano 组件的 source 属性接收一个当前图片位置的 uri 属性值。这里我们使用了 asset 函数，将 sample_pano.jpg 作为参数，这个函数将会返回 static_assets 路径下的图片的正确的 uri 。换句话说，上面的方法等价于下面的方法：

```
render() {
  return (
    <View>
      <Pano source={ {uri:'../static_assets/sample_pano.jpg'} }/>
    </View>
  );
}
```

假设本地服务器一直在运行，当我们在浏览器中刷新页面时，我们将看到如下效果：

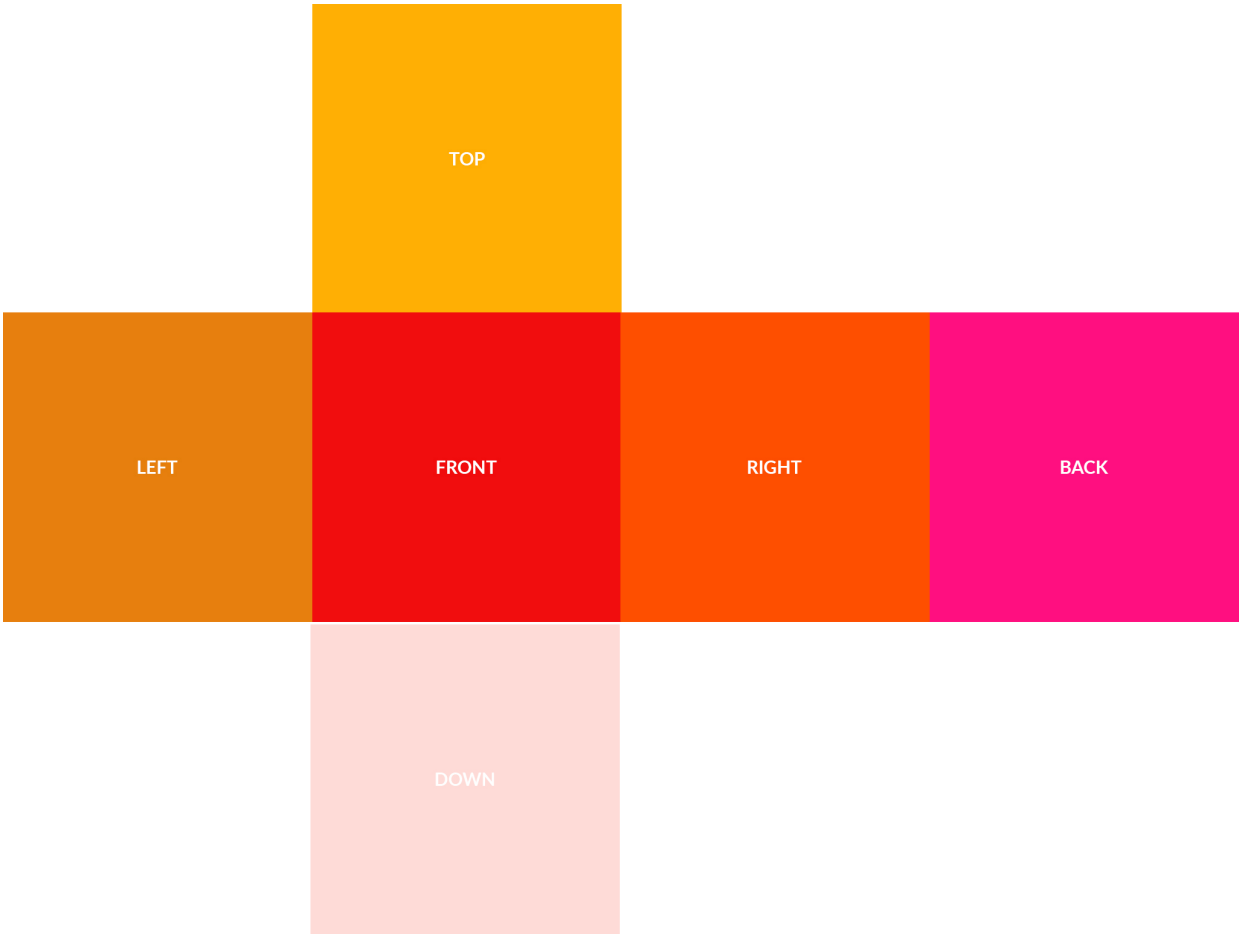
顺便说一下，如果我们想避免在每次更改时都需要重新刷新页面，我们可以通过在URL(http://localhost:8081/vr/?hotreload) 中添加 `?hotreload` 来启用 **热刷新**。

立方体全景图

立方体全景图是360度全景图的其他格式。这种格式使用六个图像作为一个多维数组集的六个面，它将填充我们周围的*球体*。它也被称为天空盒。

基本思想是渲染一个立方体，并将观众置于中心，随后移动。

例如，下面的这张大图中，每一个方位的小图代表立方体的一面：

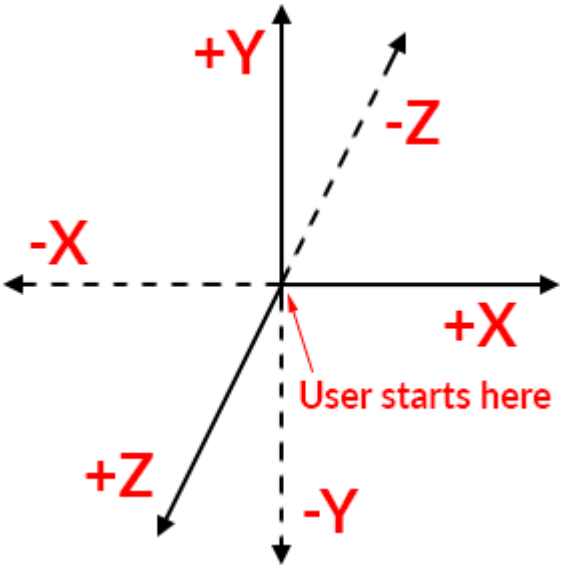


为了能够在React VR中使用立方体全景图，`Pano` 组件的 `source` 属性的 `uri` 值必须指定为一个数组，数组中的每一张图片分别代表 `[+x, -x, +y, -y, +z, -z]` ，如下所示：


```
render() {
  return (
    <View>
      <Pano source={
        {
          uri: [
            '../static_assets/sample_right.jpg',
            '../static_assets/sample_left.jpg',
            '../static_assets/sample_top.jpg',
            '../static_assets/sample_bottom.jpg',
            '../static_assets/sample_back.jpg',
            '../static_assets/sample_front.jpg'
          ]
        }
      } />
    </View>
  );
}
```

在2D布局中，X轴越向右x值越大，Y轴越向下值越大，(0,0)坐标为最左上角，右下角代表元素的宽和高(width, height)。

然而，在3D空间中，React VR使用了同OpenGL使用的右手坐标系统，正X指向右边，正Y指向上边，正Z指向用户的方向。因为用户的默认视图从原点开始，这意味着它们将从负Z方向开始：



你可以从React VR coordinate system here了解更多React VR坐标系统.

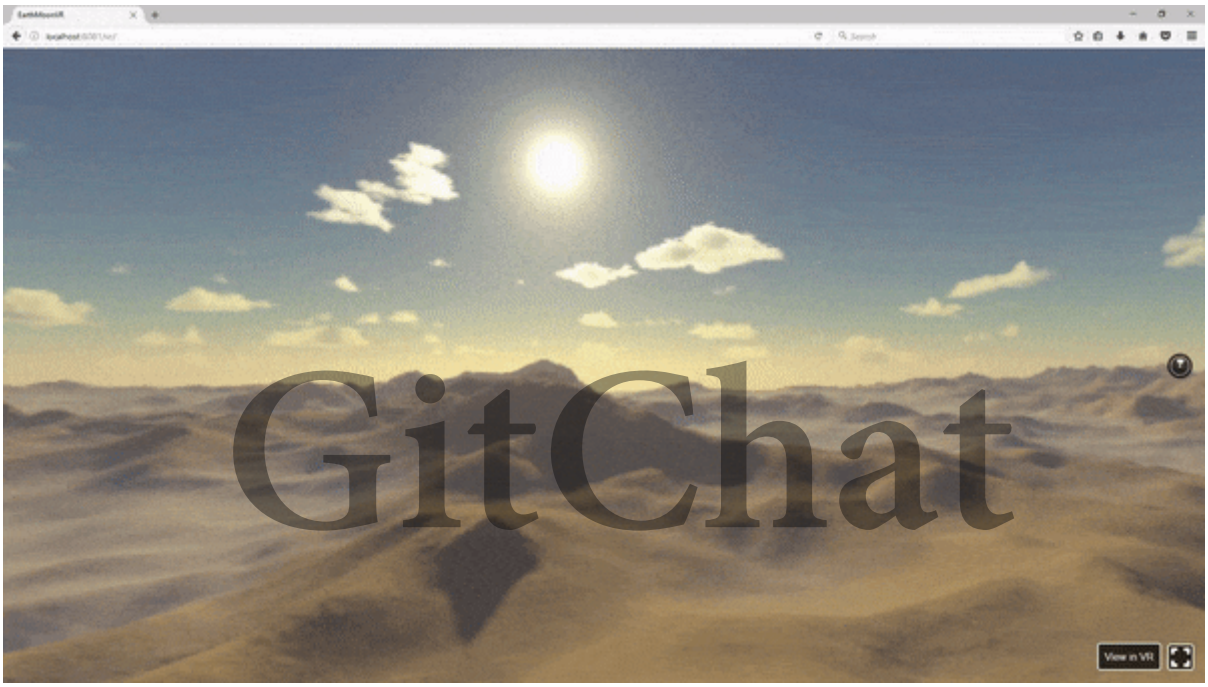
这样，我们的立方体（或天空盒）将如下所示：

Skybox在Unity中使用了很多，所以有很多地方可以找到他们并进行下载。 例如，我从这个页面下载了撒哈拉沙漠。我将图片拖拽到项目中，并修改代码如下所示：

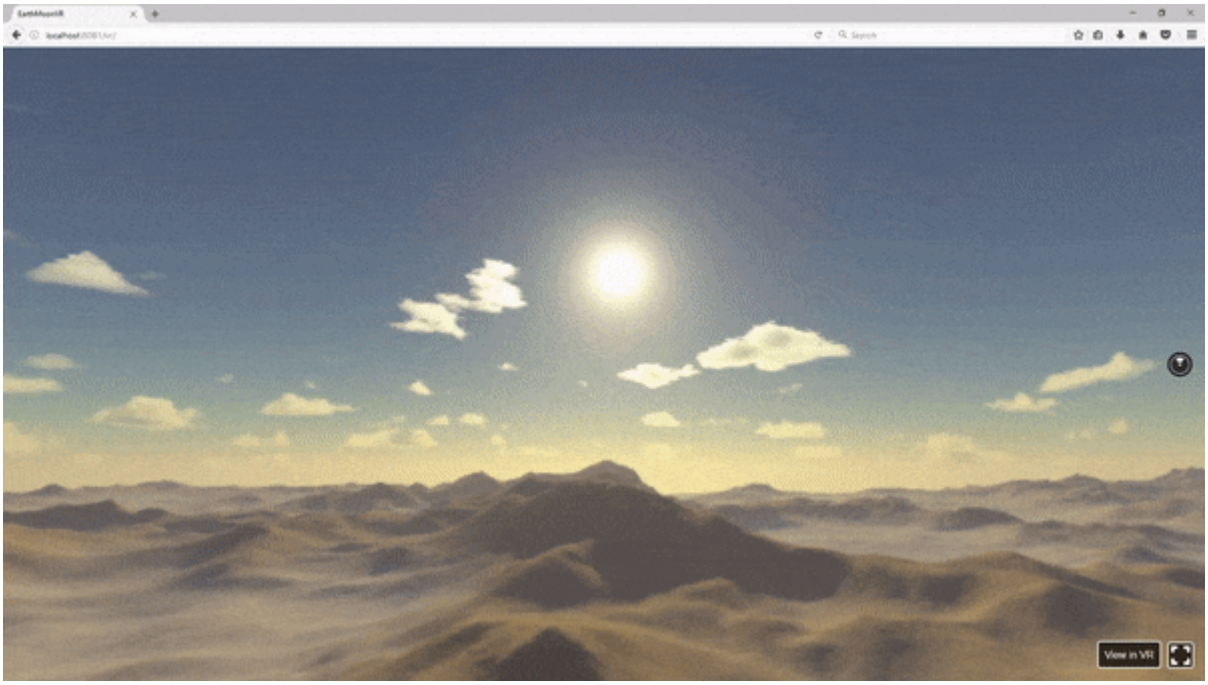
```
render() {
  return (
    <View>
      <Pano source={
        {
          uri: [
            '../static_assets/sahara_rt.jpg',
```

```
              '../static_assets/sahara_lf.jpg',
              '../static_assets/sahara_up.jpg',
              '../static_assets/sahara_dn.jpg',
              '../static_assets/sahara_bk.jpg',
              '../static_assets/sahara_ft.jpg'
            ]
          }
        } />
      </View>
    );
  }
```

效果如下所示：



你能注意到顶部和底部的图像不协调吗？我们可以通过将顶部图像顺时针旋转90度，底部逆时针旋转90度来校正它们：



现在让我们为我们的应用创建一个空间天空盒。

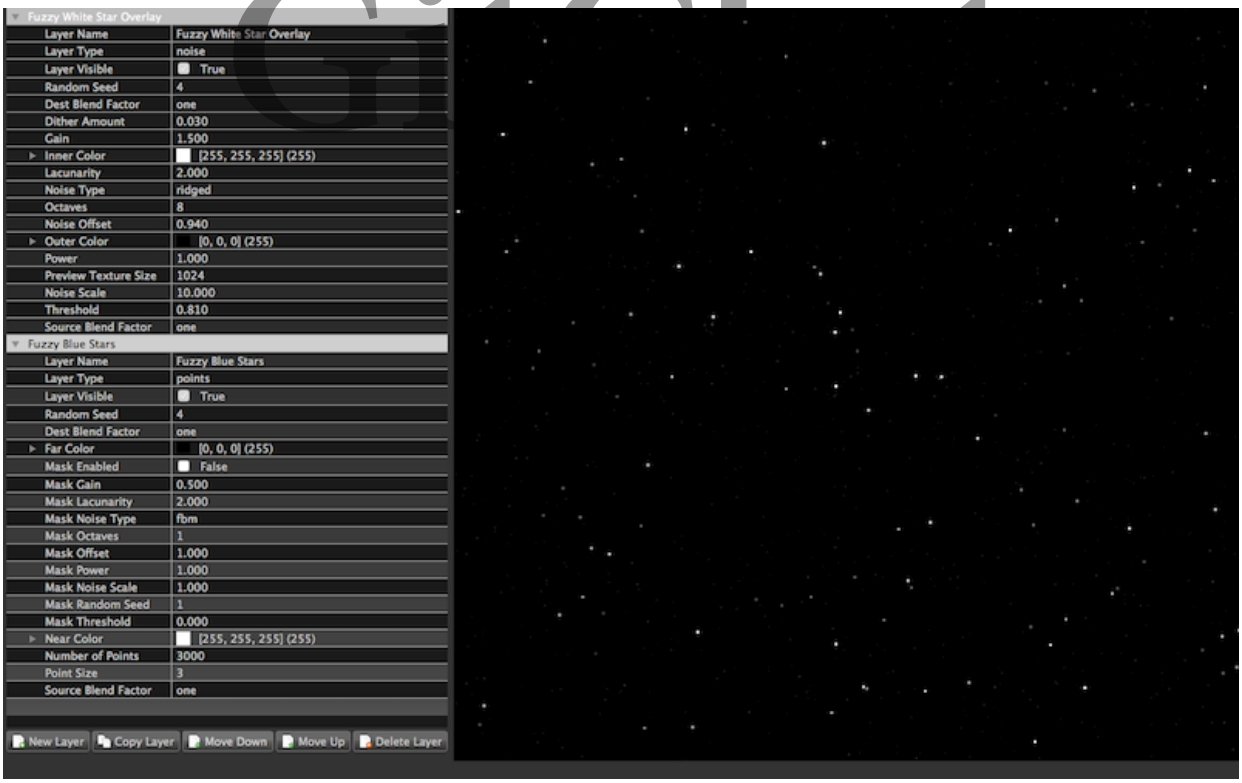
最好的程序是Spacescape， 它一个免费的工具，可在Windows和Mac上创建空间天空盒（包括星星和星云）。

创建一个 sampleSpace.xml 文件，将下面的代码拷贝到 sampleSpace.xml 文件中:

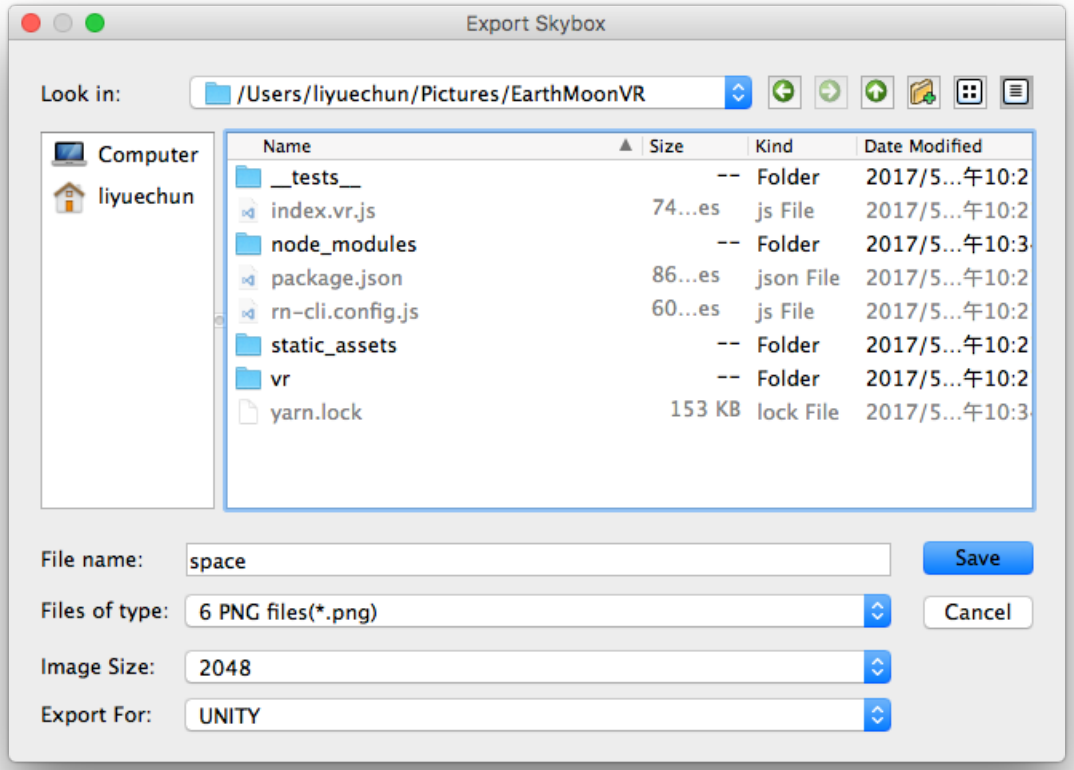

```
<?xml version="1.0" encoding="utf-8" ?>
<spacescapelayers>
    <layer>
        <destBlendFactor>one</destBlendFactor>
        <farColor>0 0 0 1</farColor>
        <hdrMultiplier>1</hdrMultiplier>
        <hdrPower>1</hdrPower>
        <maskEnabled>false</maskEnabled>
        <maskGain>0.5</maskGain>
        <maskLacunarity>2</maskLacunarity>
        <maskNoiseType>fbm</maskNoiseType>
        <maskOctaves>1</maskOctaves>
        <maskOffset>1</maskOffset>
        <maskPower>1</maskPower>
        <maskScale>1</maskScale>
        <maskSeed>1</maskSeed>
        <maskThreshold>0</maskThreshold>
        <name>Fuzzy Blue Stars</name>
        <nearColor>1 1 1 1</nearColor>
        <numPoints>3000</numPoints>
        <pointSize>3</pointSize>
        <seed>4</seed>
        <sourceBlendFactor>one</sourceBlendFactor>
        <type>points</type>
    </layer>
    <layer>
```

```
            <destBlendFactor>one</destBlendFactor>
            <ditherAmount>0.03</ditherAmount>
            <gain>1.5</gain>
            <hdrMultiplier>1</hdrMultiplier>
            <hdrPower>1</hdrPower>
            <innerColor>1 1 1 1</innerColor>
            <lacunarity>2</lacunarity>
            <name>Fuzzy White Star Overlay</name>
            <noiseType>ridged</noiseType>
            <octaves>8</octaves>
            <offset>0.94</offset>
            <outerColor>0 0 0 1</outerColor>
            <powerAmount>1</powerAmount>
            <previewTextureSize>1024</previewTextureSize>
            <scale>10</scale>
            <seed>4</seed>
            <shelfAmount>0.81</shelfAmount>
            <sourceBlendFactor>one</sourceBlendFactor>
            <type>noise</type>
        </layer>
    </spacescapelayers>
```

并且通过 Spacescape 软件打开 sampleSpace.xml 文件，效果图如下所示:



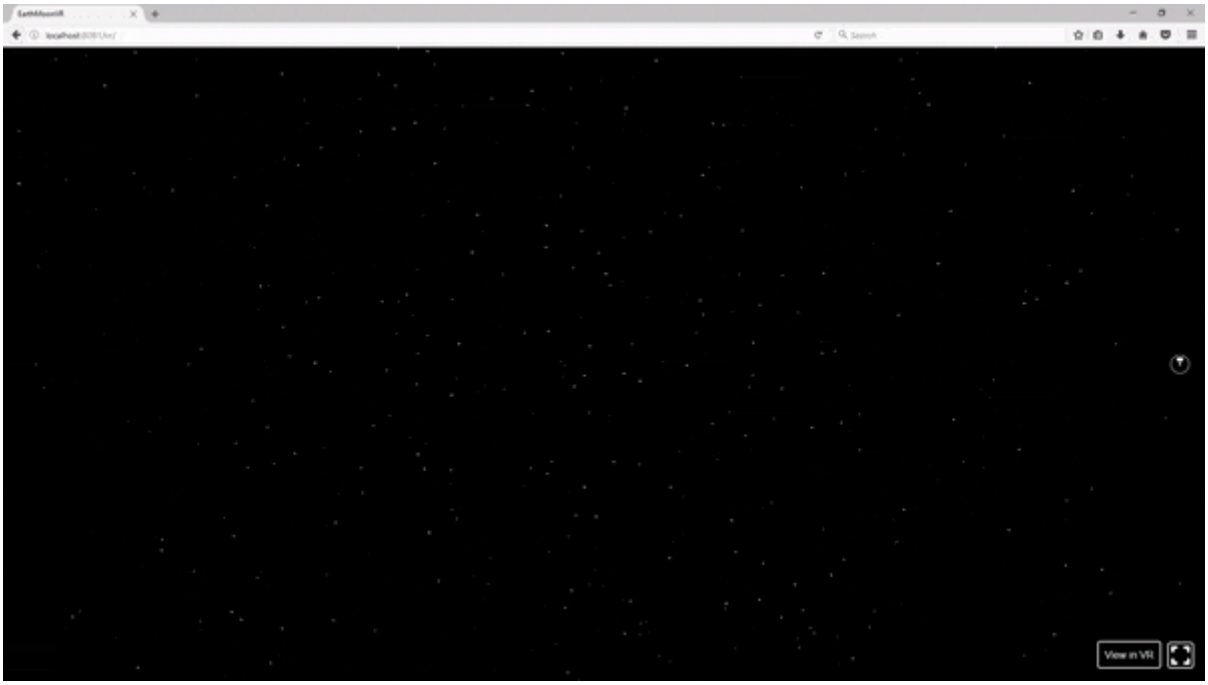
我们可以导出天空盒的六张图像：



如果我们修改代码如下所示：


```
<Pano source={
  {
    uri: [
      '../static_assets/space_right.png',
      '../static_assets/space_left.png',
      '../static_assets/space_up.png',
      '../static_assets/space_down.png',
      '../static_assets/space_back.png',
      '../static_assets/space_front.png'
    ]
  }
}/>
```

将得到如下结果：



现在让我们来讨论讨论3D模型。

3D 模型

React VR 有一个 Model 组件，它支持Wavefront .obj file format 来代表3D建模。

mesh是定义3D对象形状的顶点、边和面的集合。

.obj文件是一个纯文本文件，其中包含几何顶点，纹理坐标，顶点法线和多边形面元素的坐标。

通常，.obj文件引用一个外部.mtl file文件，其中存储了描述多边形视觉方面的材质（或纹理）。

您可以使用Blender, 3DS Max,和Maya等程序创建3D模型并将其导出为这些格式。

还有很多网站可以免费下载或免费下载3D模型。 以下是其中三个很不错的：

- TF3DM
- TurboSquid
- CGTrader

对于我们的应用程序，我们将使用这个3D地球模型和这个来自TF3DM的3D月球模型 。

当我们将地球模型的文件提取到我们应用程序的 `static_assets` 目录时，我们可以看到有一堆图像（纹理）以及.obj和.mtl文件。 如果我们在文本编辑器中打开后者，我们将看到如下所示定义：

```
# 3ds Max Wavefront OBJ Exporter v0.97b - (c)2007 guruware
# File Created: 25.01.2016 02:22:51

newmtl 01___Default
    Ns 10.0000
    Ni 1.5000
    d 1.0000
    Tr 0.0000
    Tf 1.0000 1.0000 1.0000
    illum 2
    Ka 0.0000 0.0000 0.0000
    Kd 0.0000 0.0000 0.0000
    Ks 0.0000 0.0000 0.0000
    Ke 0.0000 0.0000 0.0000
    map_Ka 4096_earth.jpg
    map_Kd 4096_earth.jpg
    map_Ke 4096_night_lights.jpg
    map_bump 4096_bump.jpg
    bump 4096_bump.jpg

newmtl 02___Default
    Ns 10.0000
    Ni 1.5000
    d 1.0000
    Tr 0.0000
    Tf 1.0000 1.0000 1.0000
    illum 2
    Ka 0.5882 0.5882 0.5882
    Kd 0.5882 0.5882 0.5882
    Ks 0.0000 0.0000 0.0000
    Ke 0.0000 0.0000 0.0000
    map_Ka 4096_earth.jpg
    map_Kd 4096_earth.jpg
    map_d 4096_earth.jpg
```


现在我们将添加 Model 组件，如下面的代码所示：

```
<Model source={{obj:asset('earth.obj'), mtl:asset('earth.mtl')}}
lit={true} />
```

lit 属性指定网格中使用的材料应使用Phong shading处理灯。

同时，也不要忘了从 react-vr 中导入 Model 组件：

```
import {
  ...
  Model,
} from 'react-vr';
```

但是，如果我们只将该组件添加到我们的应用程序中，则不会显示任何内容。 我们首先需要添加一个光源。

React VR 有四种光源类型：

- AmbientLight表示全方位，固定强度和固定颜色的光源，可以均匀地影响场景中的所有对象。
- DirectionalLight表示从指定方向平均照亮所有物体的光源。
- PointLight 代表光的起源来源于一个点，并向各个方向传播。
- SpotLight 代表光的起源来源于一个点，并以锥形向外扩散。

您可以尝试所有类型的灯光，看看哪一个可以为您带来最佳效果。 在这种情况下，我们将使用强度值为 `2.6` 的 `AmbientLight` ：

```
import React from 'react';
import {
  AppRegistry,
  asset,
  StyleSheet,
  Pano,
  Text,
  View,
  Model,
  AmbientLight,
} from 'react-vr';

class EarthMoonVR extends React.Component {
  render() {
    return (
      <View>
        ...

        <AmbientLight intensity={ 2.6 }  />

        <Model source={{obj:asset('earth.obj'),
mtl:asset('earth.mtl')}} lit={true} />
      </View>
    );
  }
};

AppRegistry.registerComponent('EarthMoonVR', () => EarthMoonVR);
```

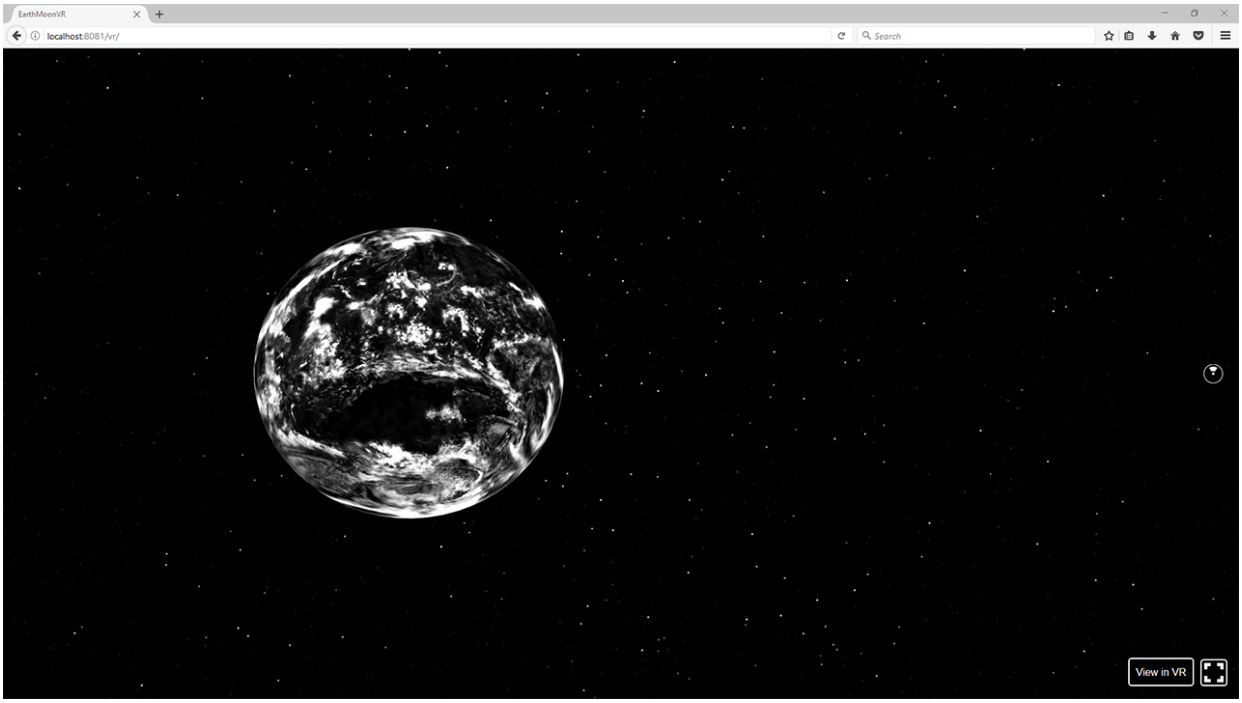
接下来，我们需要给我们的模型一些用于放置、大小、和旋转的样式属性。 通过尝试不同的值，我想出了以下配置：

```
class EarthMoonVR extends React.Component {
  render() {
    return (
      <View>
        ...

        <Model
          style={{
            transform: [
              {translate: [-25, 0, -70]},
              {scale: 0.05 },
              {rotateY: -130},
              {rotateX: 20},
              {rotateZ: -10}
            ],
          }}
          source={{obj:asset('earth.obj'),
mtl:asset('earth.mtl')}}
          lit={true}
        />
      </View>
    );
  }
};

AppRegistry.registerComponent('EarthMoonVR', () => EarthMoonVR);
```


Transforms被表示为一个样式对象中的对象数组，请记住它们最后被应用的单位是米。

`translate` 将您的模型的位置转换为x，y，z空间， `scale` 给您的模型一个大小，并根据提供的度数绕轴旋转。

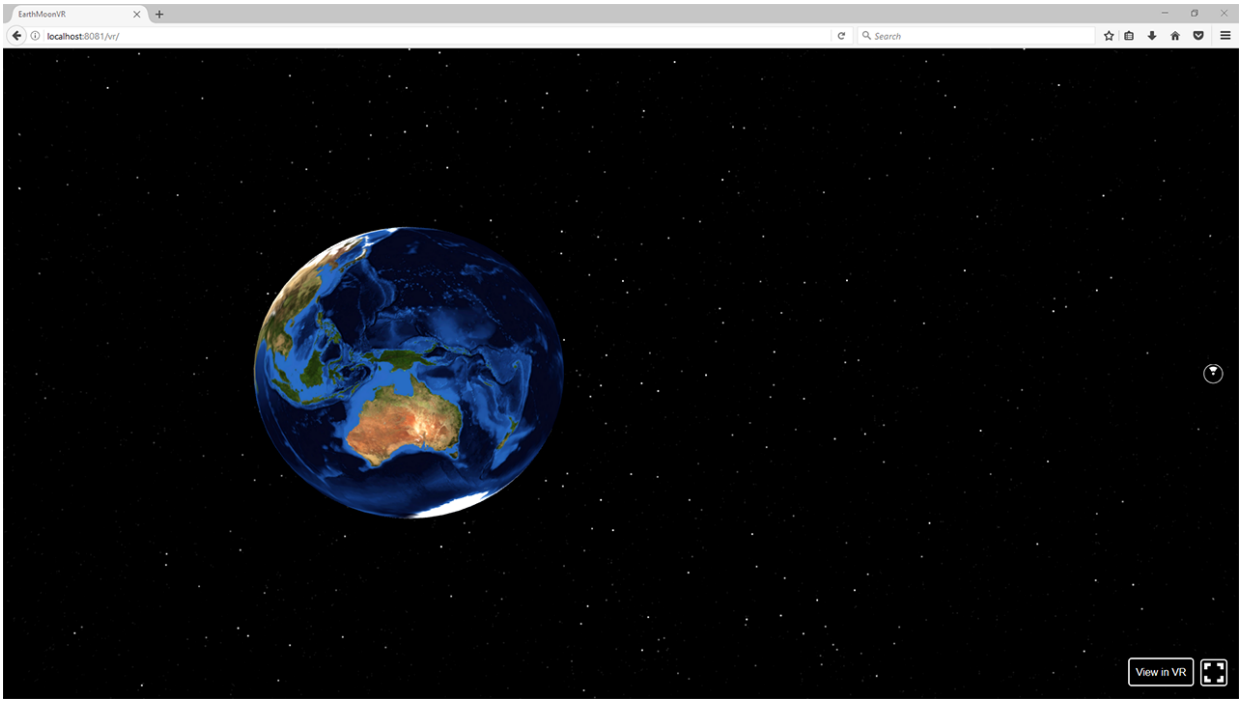
这是效果图：



这个地球模型可以应用多个纹理。 默认情况下它带 *clouds* 纹理，但是我们可以通过用 `4096_earth.jpg` 替换最后三行中的 `4096_clouds.jpg` 来更改.mtl文件中的内容：

```
# 3ds Max Wavefront OBJ Exporter v0.97b - (c)2007 guruware
# File Created: 25.01.2016 02:22:51

newmtl 01___Default
    ...

newmtl 02___Default
    ...
    map_Ka 4096_earth.jpg
    map_Kd 4096_earth.jpg
    map_d 4096_earth.jpg
```


效果图如下：



顺便说一下，如果您的模型不带有.mtl文件，React VR允许您通过下面的代码指定纹理：

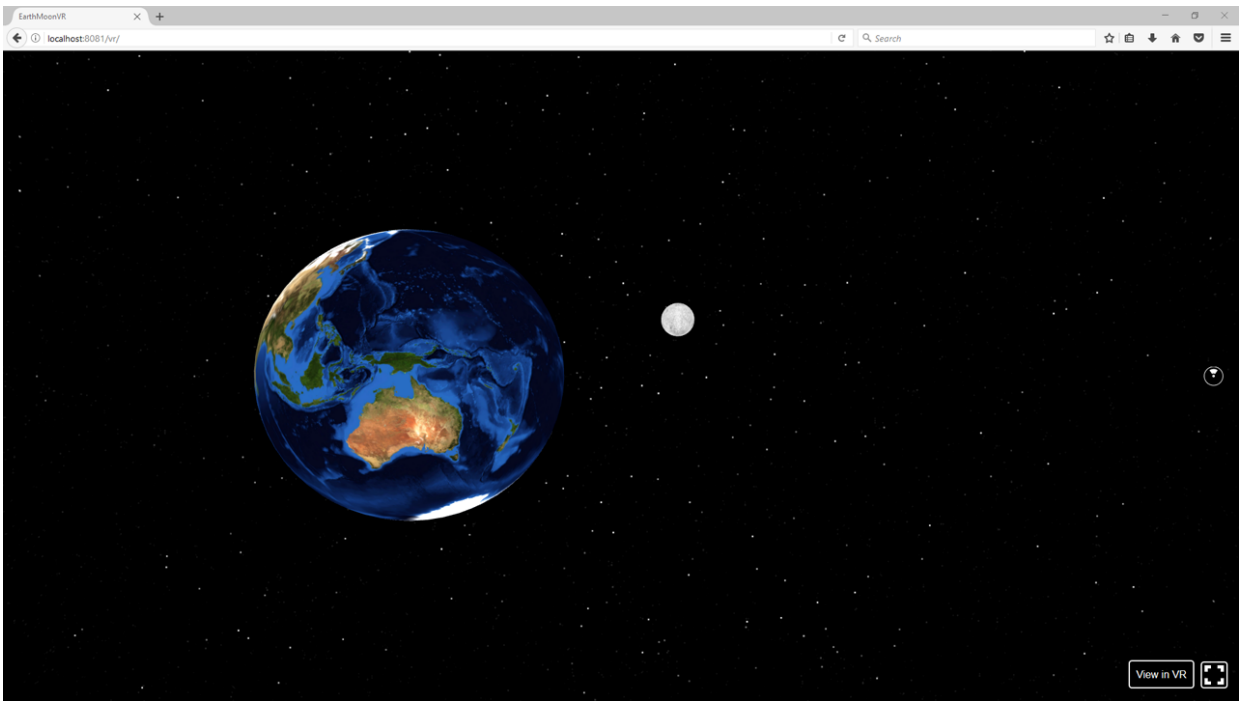
```
<Model
  source={{obj:asset('model.obj'), texture:asset('model.jpg')}}
  lit={true}
/>
```


我们对月球模型做同样的操作，将纹理的路径修复到.mtl文件中，并尝试使用不同的比例和放置值。 您不需要添加另一个光源， AmbientLight 将适用于两种模型。

这是我想出的月球模型的代码：

```
render() {
    return (
      <View>

        ...

        <Model
          style={{
            transform: [
              {translate: [10, 10, -100]},
              {scale: 0.05},
            ],
          }}
          source={{obj:asset('moon.obj'), mtl:asset('moon.mtl')}}
          lit={true}
        />
      </View>
    );
  }
```

效果图：



如果你想在WebVR上下文中了解更多关于360度全景图的信息，你可以查看the developer documentation at Oculus这篇文章。

现在，我们一起来为模型添加动画。

模型动画化

React VR 有一个 动画库来以简单的方式组合一些类型的动画。

在这个时候，只有几个组件可以自己动画(`View` 使用 `Animated.View`，`Text` 使用 `Animated.Text`，`Image` 使用 `Animated.Image`). 这个文档 提醒你可以通过 `createAnimatedComponent` 来创建更多的动画，但是目前为止还找不到更多的相关信息。

另位一种选择是使用requestAnimationFrame，它是基于JavaScript动画API的重要组成部分。

那么我们可以做的就是要有一个状态属性来表示两个模型的Y轴上的旋转值（在月球模型上，为了使它变慢让旋转是地球旋转的三分之一）：

```
class EarthMoonVR extends React.Component {
  constructor() {
    super();
    this.state = {
      rotation: 130,
    };
  }
```

```
  render() {
    return (
      <View>
        ...

        <Model
          style={{
            transform: [
              {translate: [-25, 0, -70]},
              {scale: 0.05 },
              {rotateY: this.state.rotation},
              {rotateX: 20},
              {rotateZ: -10}
            ],
          }}
          source={{obj:asset('earth.obj'),
mtl:asset('earth.mtl')}}
          lit={true}
        />

        <Model
          style={{
            transform: [
              {translate: [10, 10, -100]},
              {scale: 0.05},
              {rotateY: this.state.rotation / 3},
            ],
          }}
          source={{obj:asset('moon.obj'), mtl:asset('moon.mtl')}}
          lit={true}
        />
      </View>
    );
  }
};
```


现在我们来编写一个 rotate 函数，它将通过 requestAnimationFrame 函数调用每一帧，在一定时间基础上更新旋转：

```
class EarthMoonVR extends React.Component {
  constructor() {
    super();
    this.state = {
      rotation: 130,
    };
    this.lastUpdate = Date.now();

    this.rotate = this.rotate.bind(this);
  }
```

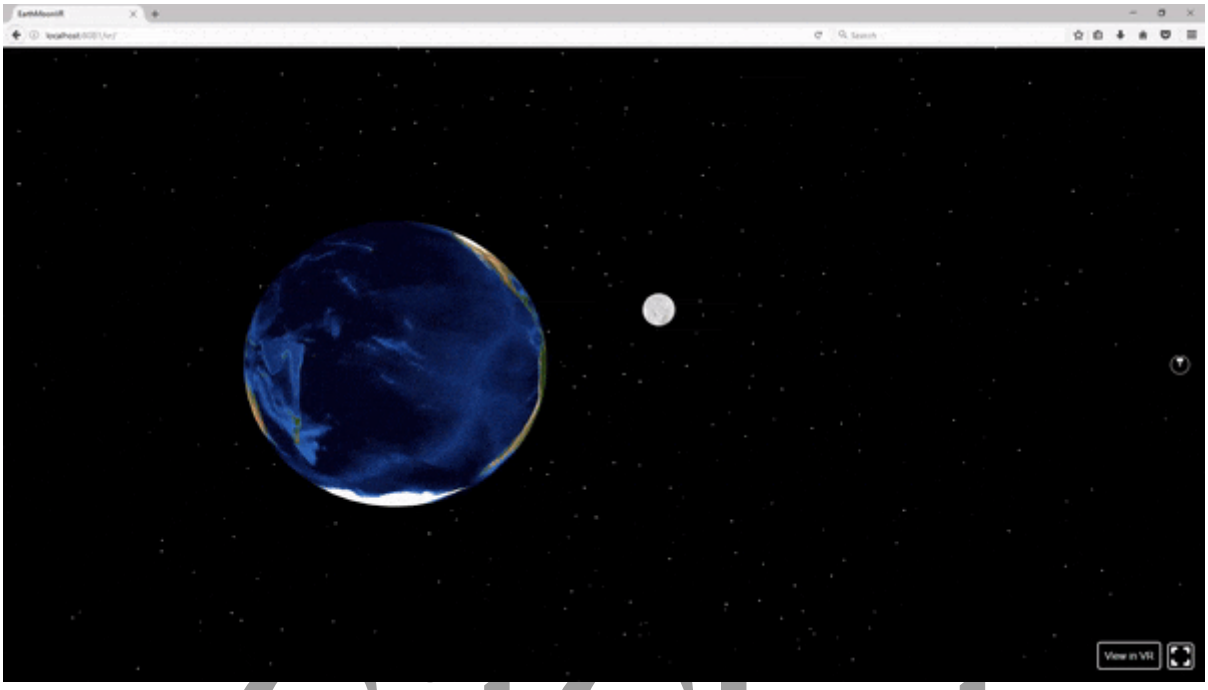
```
  rotate() {
    const now = Date.now();
    const delta = now - this.lastUpdate;
    this.lastUpdate = now;

    this.setState({
        rotation: this.state.rotation + delta / 150
    });
    this.frameHandle = requestAnimationFrame(this.rotate);
  }

  ...
}
```

幻数 150只是控制旋转速度（这个数字越大，旋转速度越慢）。 我们保存由 `requestAnimationFrame` 返回的处理程序，以便当组件卸载并启动 `componentDidMount` 上的旋转动画时，我们可以取消动画：

```
class EarthMoonVR extends React.Component {
  constructor() {
    super();
    this.state = {
      rotation: 130,
    };
    this.lastUpdate = Date.now();

    this.rotate = this.rotate.bind(this);
  }

  componentDidMount() {
    this.rotate();
  }

  componentWillUnmount() {
    if (this.frameHandle) {
      cancelAnimationFrame(this.frameHandle);
      this.frameHandle = null;
    }
  }

  rotate() {
    const now = Date.now();
    const delta = now - this.lastUpdate;
    this.lastUpdate = now;

    this.setState({
        rotation: this.state.rotation + delta / 150
    });
    this.frameHandle = requestAnimationFrame(this.rotate);
```

```
    }

    ...
  }
```

这是效果图（你可能没有注意到，但是月亮旋转得很慢）：



现在让我们来添加一些button来增加一些交互。

添加button并设置样式

为我们的button创建一个新的组件。在实际开发中，我们也能使用 `View` 或者 `VrButton` ，这俩都能设置像 `onEnter` 一样的有效的事件来达到我们的目的。

然而，我们将使用VrButton，因为它有和其他组件不一样的状态机，并且很方便的添加 `onClick` 和 `onLongClick` 事件。

同时，我们为了让button外观更好看一些，我们将使用StyleSheet 来创建一个样式对象，并通过一个样式ID来对button进行引用。

下面是 `button.js` 的内容：

```
import React from 'react';
import {
  StyleSheet,
  Text,
  VrButton,
} from 'react-vr';

export default class Button extends React.Component {
  constructor() {
```

```
    super();
    this.styles = StyleSheet.create({
      button: {
        margin: 0.05,
        height: 0.4,
        backgroundColor: 'red',
      },
      text: {
        fontSize: 0.3,
        textAlign: 'center',
      },
    });
  }

  render() {
    return (
      <VrButton style={this.styles.button}
        onClick={() => this.props.callback()}>
        <Text style={this.styles.text}>
          {this.props.text}
        </Text>
      </VrButton>
    );
  }
}
```


一个 VrButton 没有外观效果，因此我们必须给它添加样式。它也可以包装一个 Image 或 Text 组件。当点击这个button时，我们可以给它传递一个事件函数来接收点击事件。

现在在我们的根组件中，我们倒入 Button 组件并且在 render 函数中，如下所示添加两个Button。

```
...
import Button from './button.js';

class EarthMoonVR extends React.Component {
  ...

  render() {
    return (
      <View>
        ...

        <AmbientLight intensity={ 2.6 }  />

        <View>
          <Button text='+'  />
          <Button text='-'  />
        </View>
```

```
        ...
      </View>
    );
  }
};
```

这两个Button在被触发时将会改变模型的Z坐标值并进行相应的缩放。因此，我们添加一个 zoom 状态机变量值，让它的初始值为 -70 （地球的Z轴值），当我们点击 + 和 - 时会 增加 和 减少 zoom 的值。

```
class EarthMoonVR extends React.Component {
  constructor() {
    super();
    this.state = {
      rotation: 130,
      zoom: -70,
    };
    ...
  }

  render() {
    return (
      <View>

        ...

        <View>
          <Button text='+'
            callback={() => this.setState((prevState) => ({ zoom:
prevState.zoom + 10 }) ) } />
          <Button text='-'
            callback={() => this.setState((prevState) => ({ zoom:
prevState.zoom - 10 }) ) } />
        </View>

        <Model
          style={{
            transform: [
              {translate: [-25, 0, this.state.zoom]},
              {scale: 0.05 },
              {rotateY: this.state.rotation},
              {rotateX: 20},
              {rotateZ: -10}
            ],
          }}
          source={{obj:asset('earth.obj'),
mtl:asset('earth.mtl')}}
          lit={true}
        />
```

```
          <Model
            style={{
              transform: [
                {translate: [10, 10, this.state.zoom - 30]},
                {scale: 0.05},
                {rotateY: this.state.rotation / 3},
              ],
            }}
            source={{obj:asset('moon.obj'), mtl:asset('moon.mtl')}}
            lit={true}
          />
        </View>
      );
    }
};
```

现在我们通过 StyleSheet.create 来给包含两个Button的 View 添加flexbox布局样式。

```
class EarthMoonVR extends React.Component {
  constructor() {
    super();
    ...
    this.styles = StyleSheet.create({
      menu: {
        flex: 1,
        flexDirection: 'column',
        width: 1,
        alignItems: 'stretch',
        transform: [{translate: [2, 2, -5]}],
      },
    });
    ...
  }

  render() {
    return (
      <View>
        ...

        <View style={ this.styles.menu }>
          <Button text='+'
            callback={() => this.setState((prevState) => ({ zoom:
prevState.zoom + 10 }) ) } />
          <Button text='-'
            callback={() => this.setState((prevState) => ({ zoom:
prevState.zoom - 10 }) ) } />
        </View>

        ...
```

```
        </View>
      );
    }
  };
```

在flexbox布局中，子组件会通过 `flexDirection:'column'` 属性值垂直布局，通过 `flexDirection:'row'` 属性值水平布局。在这个案例中，flex设置为1代表两个button大小一样，`flexDirection` 设置为 `column` 代表两个button从上往下排列。`alignItems` 的值为 `stretch` 代表两个Button和父视图宽度一样。

看看 this page on the React Native documentation 和 this one on the React VR documentation 这两篇文章来了解更多关于flexbox布局的相关知识。

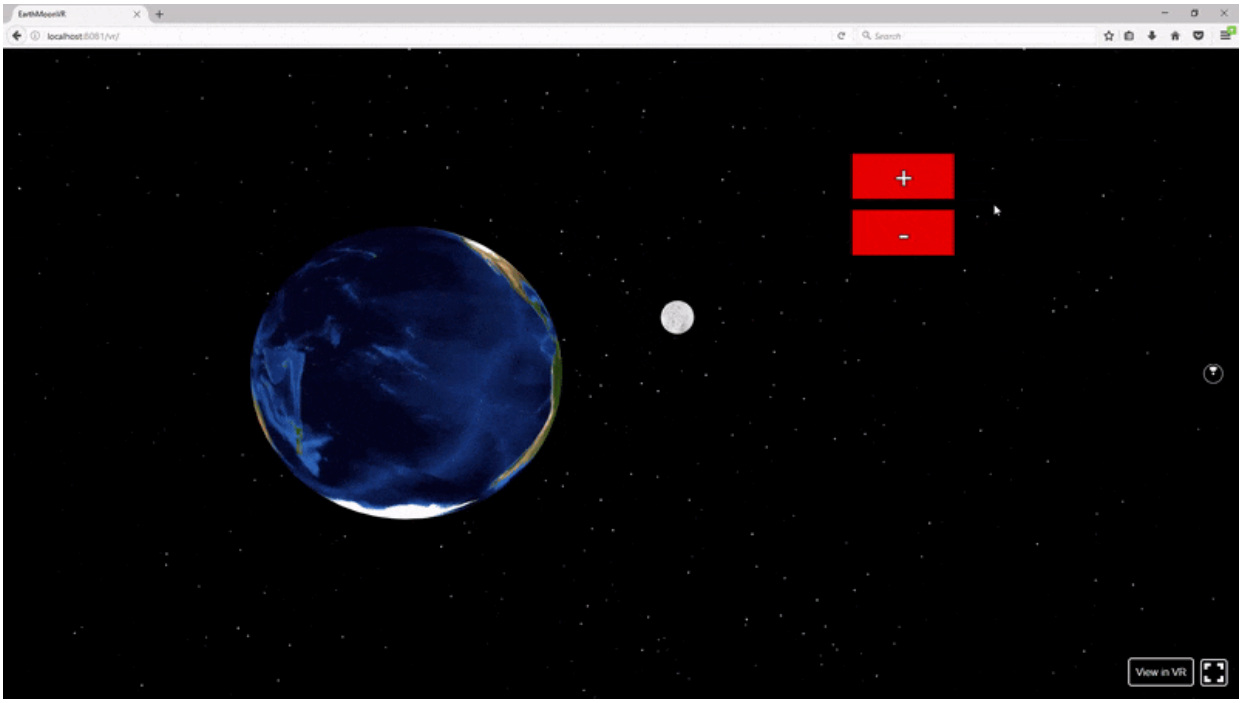
最后，我们可以从 `render` 函数中将天空盒的图片移除，以便 `render` 中的代码看起来不至于那么拥挤：

```
import React from 'react';
import {
  AppRegistry,
  asset,
  StyleSheet,
  Pano,
  Text,
  View,
  Model,
  AmbientLight,
} from 'react-vr';
import Button from './button.js';

class EarthMoonVR extends React.Component {
  constructor() {
    super();
    this.state = {
      rotation: 130,
      zoom: -70,
    };
    this.lastUpdate = Date.now();
    this.spaceSkymap = [
      '../static_assets/space_right.png',
      '../static_assets/space_left.png',
      '../static_assets/space_up.png',
      '../static_assets/space_down.png',
      '../static_assets/space_back.png',
      '../static_assets/space_front.png'
    ];
    this.styles = StyleSheet.create({
      menu: {
        flex: 1,
        flexDirection: 'column',
```

```
        width: 1,
        alignItems: 'stretch',
        transform: [{translate: [2, 2, -5]}],
      },
    });

    this.rotate = this.rotate.bind(this);
  }

  componentDidMount() {
    this.rotate();
  }

  componentWillUnmount() {
    if (this.frameHandle) {
      cancelAnimationFrame(this.frameHandle);
      this.frameHandle = null;
    }
  }

  rotate() {
    const now = Date.now();
    const delta = now - this.lastUpdate;
    this.lastUpdate = now;

    this.setState({
        rotation: this.state.rotation + delta / 150
    });
    this.frameHandle = requestAnimationFrame(this.rotate);
  }

  render() {
    return (
      <View>
        <Pano source={ {uri: this.spaceSkymap} }/>

        <AmbientLight intensity={ 2.6 }  />

        <View style={ this.styles.menu }>
          <Button text='+'
            callback={() => this.setState((prevState) => ({ zoom:
prevState.zoom + 10 }) ) } />
          <Button text='-'
            callback={() => this.setState((prevState) => ({ zoom:
prevState.zoom - 10 }) ) } />
        </View>

        <Model
          style={{
            transform: [
              {translate: [-25, 0, this.state.zoom]},
              {scale: 0.05 },
```

```
                {rotateY: this.state.rotation},
                {rotateX: 20},
                {rotateZ: -10}
              ],
            }}
            source={{obj:asset('earth.obj'),
mtl:asset('earth.mtl')}}
            lit={true}
          />

          <Model
            style={{
              transform: [
                {translate: [10, 10, this.state.zoom - 30]},
                {scale: 0.05},
                {rotateY: this.state.rotation / 3},
              ],
            }}
            source={{obj:asset('moon.obj'), mtl:asset('moon.mtl')}}
            lit={true}
          />
        </View>
      );
    }
};

AppRegistry.registerComponent('EarthMoonVR', () => EarthMoonVR);
```


如果我们测试这个应用程序，我们将看到两个button均触发了相关事件。



总结

React Native 是基于React的一个移动端的JavaScrpit库，而React VR又是基于React Native的适用于虚拟现实的JavaScrpit库。React VR允许我们快速方便的创建VR体验。

有很多相关的社区，如A-Frame和React VR 中文网。如果你想在App中制作360度的VR全景效果，并且如果你了解React / React Native，那么React VR是非常不错的选择。

记住，你能够从GitHub下载本篇文章中的源码。

谢谢阅读！