



下载APP



41 | 实战（五）：如何实现3D地球可视化（下）？

2020-10-12 月影

跟月影学可视化

[进入课程 >](#)**讲述：月影**

时长 11:07 大小 10.19M



你好，我是月影。

上节课，我们实现了一个有着星空背景的 3D 地球效果。但这个效果还比较简单，在某些可视化大屏项目中，我们不仅要呈现视觉效果，还要允许用户与可视化大屏中呈现的内容进行交互。所以这节课，我们会先给这个 3D 地球添加各种交互细节，比如让地球上的国家随着我们鼠标的移动而高亮，接着，我们再在地球上放置各种记号，比如光柱、地标等等。

如何选中地球上的地理位置？



我们先来解决上节课留下的一个问题，为什么我们在绘制 3D 地球的时候，要大费周章地使用 topoJSON 数据，而不是直接用一个现成的等角方位投影的世界地图图片作为球体的

纹理。这是因为，我们想让地球能够和我们的鼠标进行交互，比如当点击到地图上的中国区域的时候，我们想让中国显示高亮，这是纹理图片无法实现的。接下来，我们就来看看怎么实现这样交互的效果。

实现坐标转换

实现交互效果的难点在于坐标转换。因为鼠标指向地球上的某个区域的时候，我们通过 SpriteJS 拿到的是当前鼠标在点击的地球区域的一个三维坐标，而这个坐标是不能直接判断点中的区域属于哪个国家的，我们需要将它转换成二维的地图经纬度坐标，才能通过地图数据来获取到当前经纬度下的国家或地区信息。

那如何实现这个坐标转换呢？首先，我们的鼠标在地球上移动的时候，通过 SpriteJS，我们拿到三维的球面坐标，代码如下：

[复制代码](#)

```
1 layer.setRaycast();
2
3
4 globe.addEventListener('mousemove', (e) => {
5   console.log(e.hit.localPoint);
6 });
7 ...
8 skyBox.attributes.raycast = 'none';
9
```

上面的代码中有一个小细节，我们将天空包围盒的 raycast 设置成了 none。为什么要这么做呢？因为地球包围在天空盒子内，这样设置之后，鼠标就能穿透天空包围盒到达地球，如果不这么做，天空盒子就会遮挡住鼠标事件，地球也就捕获不到事件了。这样一来，当鼠标移动到地球上时，我们就可以得到相应的三维坐标信息了。

接下来，我们要将三维坐标信息转换为经纬度信息，那第一步就是将三维坐标转换为二维平面坐标。

[复制代码](#)

```
1 /**
2  * 将球面坐标转换为平面地图坐标
3  * @param {*} x
4  * @param {*} y
5  * @param {*} z
```

```
6   * @param {*} radius
7   */
8   function unproject(x, y, z, radius = 1) {
9     const pLength = Math.PI * 2;
10    const tLength = Math.PI;
11    const v = Math.acos(y / radius) / tLength; // const y = radius * Math.cos(v
12    let u = Math.atan2(-z, x) + Math.PI; // z / x = -1 * Math.tan(u * pLength);
13    u /= pLength;
14    return [u * mapScale * mapWidth, v * mapScale * mapHeight];
15  }
```

这个球面和平面坐标转换，实际上就是将空间坐标系从球坐标系转换为平面直接坐标系。具体的转换方法是，我们先将球坐标系转为圆柱坐标系，再将圆柱坐标系转为平面直角坐标系。具体的公式推导过程比较复杂，我们没必要深入理解，你只要会用我给出的 `unproject` 函数就可以了。如果你对推导原理有兴趣可以回顾 [第 15 课](#)，自己来推导一下，或者阅读 [这篇文章](#)。

拿到了二维平面直角坐标之后，我们可以直接用等角方位投影函数的反函数将这个平面直角坐标转换为经纬度，代码如下：

[复制代码](#)

```
1 function positionToLatLng(x, y, z, radius = 1) {
2   const [u, v] = unproject(x, y, z, radius);
3   return projection.invert([u, v]);
4 }
```

接着，我们实现一个通过经纬度拿到国家信息的函数。这里，我们直接通过 `d3.geoContains` 方法，从 `countries` 数据中拿到对应的国家信息。

[复制代码](#)

```
1 function getCountryInfo(latitude, longitude, countries) {
2   if(!countries) return {index: -1};
3   let idx = -1;
4   countries.features.some((d, i) => {
5     const ret = d3.geoContains(d, [longitude, latitude]);
6     if(ret) idx = i;
7     return ret;
8   });
9   const info = idx >= 0 ? {...countries.features[idx]} : {};
10  info.index = idx;
11  return info;
12 }
```

```
12 }
```

这样一来，我们只要修改 `mousemove` 方法，就可以知道我们的鼠标移动在哪个国家之上了。

[复制代码](#)

```
1 globe.addEventListener('mousemove', (e) => {
2   const [lng, lat] = positionToLatLng(...e.hit.localPoint);
3   const country = getCountryInfo(lat, lng, countries);
4   if(country.properties) {
5     console.log(country.properties.name);
6   }
7 });
```

高亮国家地区的方法

下一步，我们就可以实现一个方法来高亮鼠标移动到的国家或地区了。要高亮对应的国家或地区，其实处理起来并不复杂。我们先把原始的非高亮的图片另存一份，然后根据选中国家的 `index` 信息，从 `countries` 原始数据中取出对应的那个国家，用不同的填充色 `fillStyle` 再绘制一次，最后更新 `texture` 和 `layer`，就可以将高亮区域绘制出来了。代码如下：

[复制代码](#)

```
1 function highlightMap(texture, info, countries) {
2   if(texture.index === info.index) return;
3   const canvas = texture.image;
4   if(!canvas) return;
5
6
7   const idx = info.index;
8   const highlightMapContxt = canvas.getContext('2d');
9
10
11   if(!imgCache) {
12     imgCache = new OffscreenCanvas(canvas.width, canvas.height);
13     imgCache.getContext('2d').drawImage(canvas, 0, 0);
14   }
15   highlightMapContxt.clearRect(0, 0, mapScale * mapWidth, mapScale * mapHeight);
16   highlightMapContxt.drawImage(imgCache, 0, 0);
17
18
19   if(idx > 0) {
20     const path = d3.geoPath(projection).context(highlightMapContxt);
```

```
21 highlightMapContxt.save();
22 highlightMapContxt.fillStyle = '#fff';
23 highlightMapContxt.beginPath();
24 path({type: 'FeatureCollection', features: countries.features.slice(idx, i
25 highlightMapContxt.fill();
26 highlightMapContxt.restore();
27 }
28 texture.index = idx;
29 texture.needsUpdate = true;
30 layer.forceUpdate();
```

仔细看上面的代码你会发现，这里我们实际上做了两点优化，一是我们在 texture 对象上记录了上一次选中区域的 index。如果移动鼠标时，index 没发生变化，说明鼠标仍然在当前高亮的国家内，没有必要重绘。二是我们保存了原始非高亮图片。之所以这样做，是因为我们只要将保存的非高亮图片通过 drawImage 一次绘制，然后再绘制高亮区域，就可以完成地图高亮效果，而不需要每次都重新用 Path 来绘制整个地图了，因而大大减少了 Canvas2D 绘图指令的数量，显著提升了性能。

实现了这个函数之后，我们改写 mousemove 事件处理函数，就能将这个交互效果完整地显示出来了。具体的代码和效果图如下：

[复制代码](#)

```
1 globe.addEventListener('mousemove', (e) => {
2   const [lng, lat] = positionToLatLng(...e.hit.localPoint);
3   const country = getCountryInfo(lat, lng, countries);
4   highlightMap(texture, country, countries);
5 });
6
```

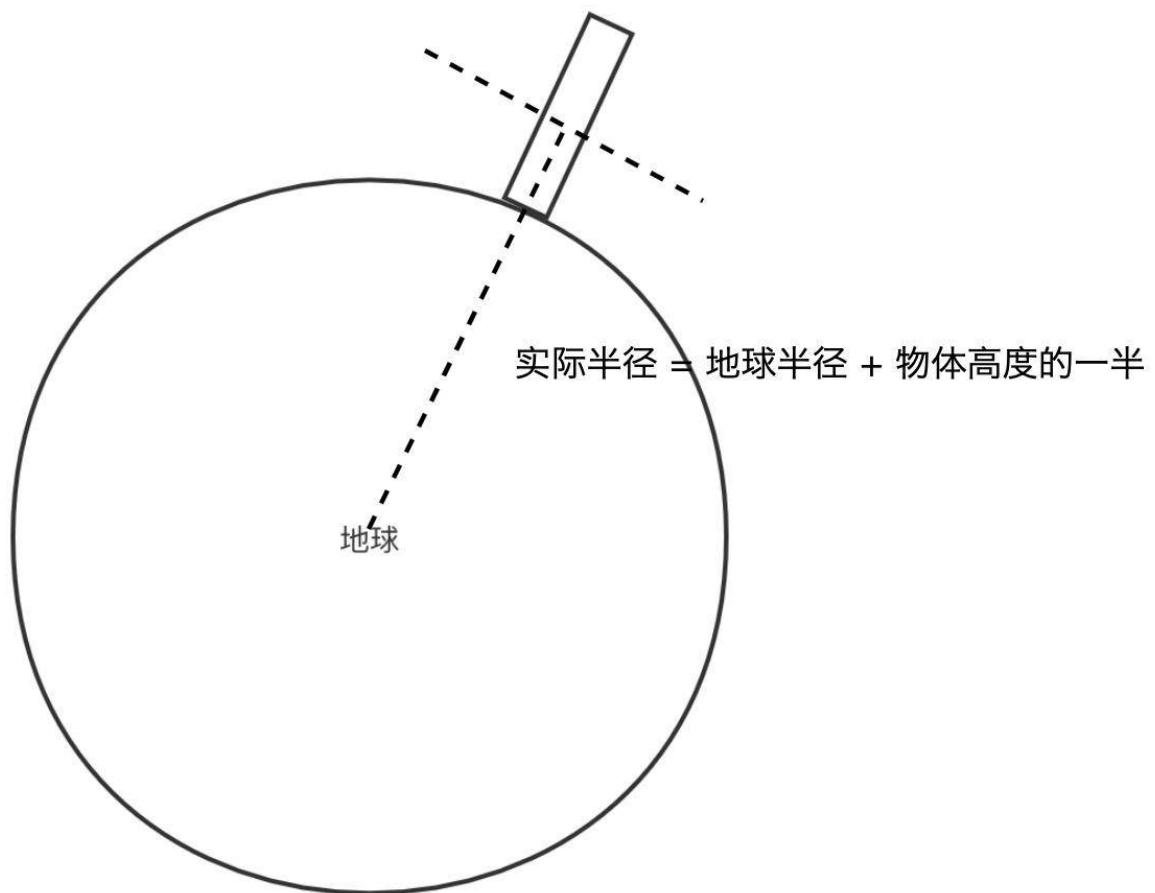


如何在地球上放置标记？

通过选中对应的国家，我们可以实现鼠标的移动的高亮交互效果。接下来，我们来实现另一个交互效果，在地球的指定经纬度处放置一些标记。

如何计算几何体摆放位置？

既然要把物体放置在地球指定的经纬坐标处，那我们接下来的操作依然离不开坐标转换。首先，我们知道几何体通常默认是以中心点为 $(0,0)$ 点，但我们放置的时候，却需要将物体的底部放置在球面上，所以我们需要对球面坐标位置进行一个坐标变换。



因此，我们在实现放置函数的时候，会通过 `latlngToPosition` 先将经纬度转成球面坐标 `pos`，再延展到物体高度的一半，因为球心的坐标是 `0,0`，所以 `pos` 位置就是对应的三维向量，我们使用 `scale` 就可以直接将它移动到我们要的高度了。

[复制代码](#)

```
1 function setGlobeTarget(globe, target, {latitude, longitude, transpose = false}
2   const radius = globe.attributes.radius;
3   if(transpose) target.transpose();
4   if(latitude !== null && longitude !== null) {
5     const scale = target.attributes.scaleY * (attrs.scale || 1.0);
6     const height = target.attributes.height;
7     const pos = latlngToPosition(latitude, longitude, radius);
8     // 要将底部放置在地面上
9     pos.scale(height * 0.5 * scale / radius + 1);
10    attrs.pos = pos;
11  }
12  target.attr(attrs);
13  const sp = new Vec3().copy(attrs.pos).scale(2);
14  target.lookAt(sp);
15  globe.append(target);
16 }
```

这里的 `latlngToPosition` 是前面 `positionToLatlng` 的反向操作，也就是先用 `projection` 函数将经纬度映射为地图上的直角坐标，然后用直角坐标转球面坐标的公式，将它转为球面坐标。具体的实现代码如下：

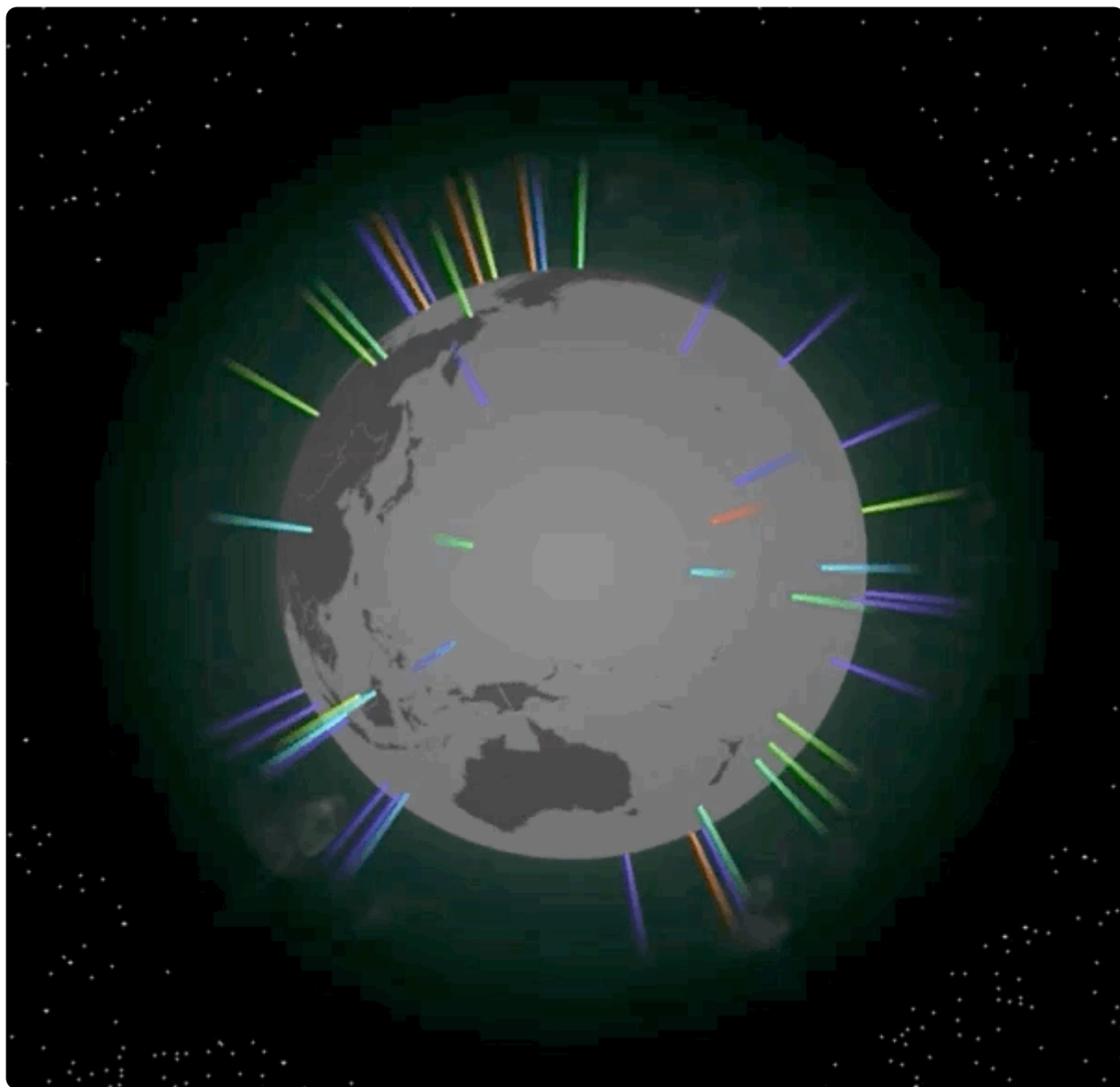
[复制代码](#)

```
1  /**
2   * 将经纬度转换为球面坐标
3   * @param {*} latitude
4   * @param {*} longitude
5   * @param {*} radius
6   */
7  function latlngToPosition(latitude, longitude, radius = 1) {
8    const [u, v] = projection([longitude, latitude]);
9    return project(u, v, radius);
10 }
11
12
13 /**
14 * 将平面地图坐标转换为球面坐标
15 * @param {*} u
16 * @param {*} v
17 * @param {*} radius
18 */
19 function project(u, v, radius = 1) {
20   u /= mapScale * mapWidth;
21   v /= mapScale * mapHeight;
22   const pLength = Math.PI * 2;
23   const tLength = Math.PI;
24   const x = -radius * Math.cos(u * pLength) * Math.sin(v * tLength);
25   const y = radius * Math.cos(v * tLength);
26   const z = radius * Math.sin(u * pLength) * Math.sin(v * tLength);
27   return new Vec3(x, y, z);
28 }
29
```

有了这个位置之后，我们将物体放上去，并且让物体朝向球面的法线方向。这一步我们可以用 `lookAt` 函数来实现。不过，`lookAt` 函数是让物体的 `z` 轴朝向向量方向，而我们绘制的一些几何体，比如圆柱体，其实是要让 `y` 轴朝向向量方向，所以这种情况下，我们需要对几何体的顶点做一个转置操作，也就是将它的顶点向量的 `x`、`y`、`z` 的值轮换一下，让 `x = y`、`y = z`、`z = x`。这么做之后，我们就可以在地球表面摆放几何体了。

摆放光柱

我们先在地球的指定位置上放置一些光柱，光柱通常可以用来标记当前位置是一个重要的地点。光柱效果如下图所示：



这个效果怎么实现呢？因为光柱本身是圆柱体，所以我们可以用 `Cylinder` 对象来绘制。而光柱的光线还会随着高度衰减，对应的 Shader 代码如下：

[复制代码](#)

```
1  const beamVertex = `  
2      precision highp float;  
3      precision highp int;  
4  
5  
6      attribute vec3 position;  
7      attribute vec3 normal;  
8      attribute vec4 color;  
9  
10  
11      uniform mat4 modelViewMatrix;
```

```
12     uniform mat4 projectionMatrix;  
13     uniform mat3 normalMatrix;  
14  
15  
16     varying vec3 vNormal;  
17     varying vec4 vColor;  
18  
19  
20     uniform vec4 ambientColor; // 环境光  
21     uniform float uHeight;  
22  
23  
24     void main() {  
25         vNormal = normalize(normalMatrix * normal);  
26         vec3 ambient = ambientColor.rgb * color.rgb; // 计算环境光反射颜色  
27         float height = 0.5 - position.z / uHeight;  
28         vColor = vec4(ambient + 0.3 * sin(height), color.a * height);  
29         vec3 P = position;  
30         P.xy *= 2.0 - pow(height, 3.0);  
31         gl_Position = projectionMatrix * modelViewMatrix * vec4(P, 1.0);  
32     }  
33     `;  
34  
35  
36  
37  
38     const beamFrag = `  
39         precision highp float;  
40         precision highp int;  
41  
42  
43         varying vec3 vNormal;  
44         varying vec4 vColor;  
45  
46  
47         void main() {  
48             gl_FragColor = vColor;  
49  
50
```

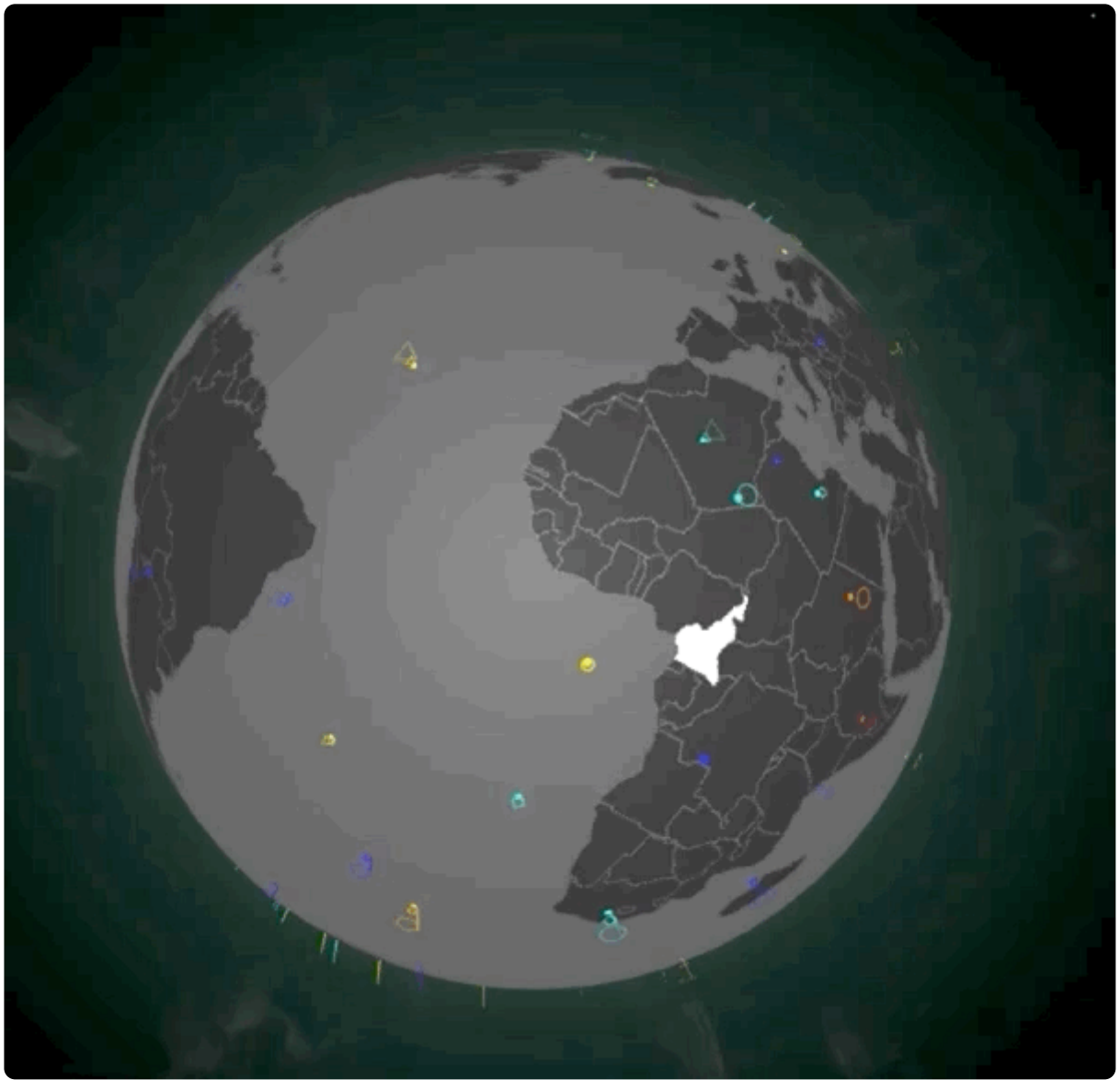
实现 Shader 的代码并不复杂，在顶点着色器里，我们可以根据高度减少颜色的不透明度。另外，我们还可以根据高度对 xy 也就是圆柱的截面做一个扩展： $P.xy *= 2.0 - \text{pow}(\text{height}, 3.0)$ ，这样就能产生一种光线发散（顶部比底部略大）的效果了。

于是，对应的 addBeam 函数实现如下所示。它就是根据参数创建对应的圆柱体对象，并把它们添加到地球对应的经纬度位置上。

```
1 function addBeam(globe, {
2   latitude,
3   longitude,
4   width = 1.0,
5   height = 25.0,
6   color = 'rgba(245,250,113, 0.5)',
7   raycast = 'none',
8   segments = 60} = {}) {
9   const layer = globe.layer;
10  const radius = globe.attributes.radius;
11  if(layer) {
12    const r = width / 2;
13    const scale = radius * 0.015;
14    const program = layer.createProgram({
15      transparent: true,
16      vertex: beamVertex,
17      fragment: beamFrag,
18      uniforms: {
19        uHeight: {value: height},
20      },
21    });
22    const beam = new Cylinder(program, {
23      radiusTop: r,
24      radiusBottom: r,
25      radialSegments: segments,
26      height,
27      colors: color,
28    });
29    setGlobeTarget(globe, beam, {transpose: true, latitude, longitude, scale,
30    return beam;
31  }
32 }
```

摆放地标

除了摆放光柱，我们还可以摆放地标。地标通常表示当前位置产生了一个重大事件。地标实现起来会比光柱更复杂一些，它由一个定位点（Spot）和一个动态的标记（Marker）共同组成。摆放了地标的地图效果如下图所示：




想要实现它，第一步我们还是要实现对应的 Shader。不过，我们这次需要实现两组 Shader。首先是 spot 的顶点着色器和片元着色器，实现起来也非常简单。在顶点着色器中，我们根据 uWidth 扩展 x、y 坐标，根据顶点绘制出一个特定大小的平面图形。在片元着色器中，我们让图形的中心稍亮一些，让边缘亮度随着距离衰减，这么做是为了增强视觉效果。不过，由于分辨率的原因，具体的效果在截图中可能体现不出来，你可以运行示例代码把地球局部放大，来实际观察和体会一下。

[复制代码](#)

```
1  const spotVertex = `  
2    precision highp float;  
3    precision highp int;  
4  
5  
6    attribute vec4 position;  
7  
8  
9    uniform mat4 modelViewMatrix;
```

```
10 uniform mat4 projectionMatrix;  
11 uniform mat3 normalMatrix;  
12  
13  
14 uniform float uWidth;  
15 uniform float uSpeed;  
16 uniform float uHeight;  
17  
18  
19 varying vec2 st;  
20  
21  
22 void main() {  
23     float s = 0.0 + (0.2 * uWidth * position.w);  
24     vec3 P = vec3(s * position.xy, 0.0);  
25     st = P.xy;  
26     gl_Position = projectionMatrix * modelViewMatrix * vec4(P, 1.0);  
27 }  
28 `;  
29  
30  
31  
32  
33 const spotFragment = `  
34     precision highp float;  
35     precision highp int;  
36  
37  
38     uniform vec2 uResolution;  
39     uniform vec3 uColor;  
40     uniform float uWidth;  
41  
42  
43     varying vec2 st;  
44  
45  
46     void main() {  
47         float d = distance(st, vec2(0));  
48         gl_FragColor.rgb = uColor + 1.5 * (0.2 * uWidth - 2.0 * d);  
49         gl_FragColor.a = 1.
```

接着，我们实现 marker 的顶点着色器和片元着色器，它们稍微复杂一些。

 复制代码

```
1 const markerVertex = `  
2     precision highp float;  
3     precision highp int;  
4  
5
```

```
6   attribute vec4 position;
7
8
9   uniform mat4 modelViewMatrix;
10  uniform mat4 projectionMatrix;
11  uniform mat3 normalMatrix;
12
13
14  uniform float uTime;
15  uniform float uWidth;
16  uniform float uSpeed;
17  uniform float uHeight;
18
19
20  varying float time;
21
22
23  void main() {
24      time = mod(uTime, 1.5 / uSpeed) * uSpeed + position.z - 1.0;
25      float d = clamp(0.0, uWidth * mix(1.0, 0.5, min(1.0, uHeight)), time);
26      float s = d + (0.1 * position.w);
27      vec3 P = vec3(s * position.xy, uHeight * time);
28      gl_Position = projectionMatrix * modelViewMatrix * vec4(P, 1.0);
29  }
30 `;
31
32
33  const markerFragment = `
34      precision highp float;
35      precision highp int;
36
37
38      uniform vec2 uResolution;
39      uniform vec3 uColor;
40
41
42      varying float time;
43
44
45      void main() {
46          float t = clamp(0.0, 1.0, time);
47          gl_FragColor.rgb = uColor;
48          gl_FragColor.a = 1.0 - t
```

在顶点着色器里，我们根据时间参数 `uTime` 来调整物体定点的高度。这样，当我们设置 `uHeight` 参数时，`marker` 就能呈现出立体的效果。


有了这两组着色器之后，我们再实现两个函数，用来分别生成 spot 和 marker 的顶点，函数代码如下：

[复制代码](#)

```
1 function makeSpotVerts(radis = 1.0, n_segments) {
2   const vertex = [];
3   for(let i = 0; i <= n_segments; i++) {
4     const theta = Math.PI * 2 * i / n_segments;
5     const x = radis * Math.cos(theta);
6     const y = radis * Math.sin(theta);
7     vertex.push(x, y, 1, 0, x, y, 1, 1.0);
8   }
9   return {
10    position: {data: vertex, size: 4},
11  };
12 }
13
14
15 function makeMarkerVerts(radis = 1.0, n_segments) {
16   const vertex = [];
17   for(let i = 0; i <= n_segments; i++) {
18     const theta = Math.PI * 2 * i / n_segments;
19     const x = radis * Math.cos(theta);
20     const y = radis * Math.sin(theta);
21     vertex.push(x, y, 1, 0, x, y, 1, 1.0);
22   }
23   const copied = [...vertex];
24   vertex.push(...copied.map((v, i) => {
25     return i % 4 === 2 ? 0.33 : v;
26   }));
27   vertex.push(...copied.map((v, i) => {
28     return i % 4 === 2 ? 0.67 : v;
29   }));
30   return {
31     position: {data: vertex, size: 4},
32   };
33 }
```

这两个函数都是用生成正多边形顶点的算法来生成对应的顶点，它们的区别是，spot 只生成一组顶点，因为是平面图形，所以 z 坐标为 0，而 marker 则生成三组不同高度的顶点组成立体的形状。

接着，我们再实现一个初始化函数，用来生成 spot 和 marker 对应的 WebGLProgram，函数代码如下：

 复制代码

```
1 function initMarker(layer, globe, {width, height, speed, color, segments}) {
2   const markerProgram = layer.createProgram({
3     transparent: true,
4     vertex: markerVertex,
5     fragment: markerFragment,
6     uniforms: {
7       uTime: {value: 0},
8       uColor: {value: new Color(color).slice(0, 3)},
9       uWidth: {value: width},
10      uSpeed: {value: speed},
11      uHeight: {value: height},
12    },
13  });
14
15
16  const markerGeometry = new Geometry(layer.gl, makeMarkerVerts(globe.attribut
17
18
19  const spotProgram = layer.createProgram({
20    transparent: true,
21    vertex: spotVertex,
22    fragment: spotFragment,
23    uniforms: {
24      uTime: {value: 0},
25      uColor: {value: new Color(color).slice(0, 3)},
26      uWidth: {value: width},
27      uSpeed: {value: speed},
28      uHeight: {value: height},
29    },
30  });
31
32
33  const spotGeometry = new Geometry(layer.gl, makeSpotVerts(globe.attributes.r
34
35
36  return {
37    program: markerProgram,
38    geometry: markerGeometry,
39    spotGeometry,
40    spotProgram,
41    mode: 'TRIANGLE_STRIP',
42  }
43}
```

最后，我们实现 `addMarker` 方法，将地标添加到地球上。这样，我们就实现了绘制地标的功能。

```
1 function addMarker(globe, {
2   latitude,
3   longitude,
4   width = 1.0,
5   height = 0.0,
6   speed = 1.0,
7   color = 'rgb(245,250,113)',
8   segments = 60,
9   lifeTime = Infinity} = {}) {
10  const layer = globe.layer;
11  const radius = globe.attributes.radius;
12
13
14  if(layer) {
15    let mode = 'TRIANGLES';
16    const ret = initMarker(layer, globe, {width, height, speed, color, segment
17    const markerProgram = ret.program;
18    const markerGeometry = ret.geometry;
19    const spotProgram = ret.spotProgram;
20    const spotGeometry = ret.spotGeometry;
21    mode = ret.mode;
22
23
24    if(markerProgram) {
25      const pos = latlngToPosition(latitude, longitude, radius);
26      const marker = new Mesh3d(markerProgram, {model: markerGeometry, mode});
27      const spot = new Mesh3d(spotProgram, {model: spotGeometry, mode});
28      setGlobeTarget(globe, marker, {pos, scale: 0.05, raycast: 'none'});
29      setGlobeTarget(globe, spot, {pos, scale: 0.05, raycast: 'none'});
30      layer.bindTime(marker.program);
31
32
33      if(Number.isFinite(lifeTime)) {
34        setTimeout(() => {
35          layer.unbindTime(marker.program);
36          marker.dispose();
37          spot.dispose();
38          marker.program.remove();
39          spot.program.remove();
40        }, lifeTime);
41      }
42      return {marker, spot};
43    }
44  }
```

要点总结

今天，我们在上节课的基础上学习了与地球交互，以及在地球上放置标记的方法。

与地球交互的实现过程，我们可以总结为四步：首先我们通过将三维球面坐标转换为经纬度坐标，再通过 topoJSON 的 API 获取当前选中的国家或地区信息，然后在离屏 Canvas 上将当前选中的国家或地区高亮显示，最后更新纹理和重绘 layer。这样，高亮显示的国家或地区就出现在 3D 地球上了。

而要在地球上放置标记，我们先要计算几何体摆放位置，然后实现标记对应的 shader，创建 WebGLProgram，最后将标记添加到地球表面对应经纬度换算的球面坐标位置处，用 lookAt 让它朝向法线方向就可以了。

到这里，地球 3D 可视化的核心功能我们就全部实现了。实际上，除了这些功能以外，我们还可以添加一些更加复杂的标记，比如两个点之间连线以及动态的飞线。这些功能实现的基本原理其实和放置标记是一样的，所以只要你掌握了我们今天讲的思路，就能比较轻松地解决这些需求了。

小试牛刀

实际上，地球上不仅可以放置普通的单点标记，还可以用曲线将两个地理点连接起来。具体的方法是在两个点之间计算弧线或贝塞尔曲线，然后将这些连线生成并绘制出来。在 SpriteJS 的 3D 扩展中有一个 Path3d 对象，它可以绘制空间中的三维曲线。你能试着实现两点之间的连线吗？如果很轻松就能实现，你还可以试着添加动画，实现两点之间的飞线动画效果。（提示：你可以通过官方文档来学习 Path3d 的用法，实现两点之间的连线。在实现飞线动画效果的时候，你可以参照 GitHub 仓库里的代码来进行学习，来理解我是怎么做的）

欢迎把你实现的效果分享到留言区，我们一起交流。也欢迎你把这节课转发出去，我们下节课见！

源码

示例代码详见 [🔗 GitHub 仓库](#)

推荐阅读

[🔗 球坐标与直角坐标的转换](#)

[提建议](#)

更多课程推荐

数据结构与算法之美

为工程师量身打造的数据结构与算法私教课

王争

前 Google 工程师



立省¥40

破90000订阅特惠，到手价¥89

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 [40 | 实战（四）：如何实现3D地球可视化（上）？](#)

下一篇 [国庆策划 | 假期别闲着，一起来挑战“手势密码”](#)

精选留言

[写留言](#)

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。