



下载APP



26 | 如何绘制带宽度的曲线？

2020-08-21 月影

跟月影学可视化

[进入课程 >](#)**讲述：月影**

时长 10:58 大小 10.06M



你好，我是月影。

在可视化应用中，我们经常需要绘制一些带有特定宽度的曲线。比如说，在地理信息可视化中，我们会使用曲线来描绘路径，而在 3D 地球可视化中，我们会使用曲线来描述飞线、轮廓线等等。

在 Canvas2D 中，要绘制带宽度的曲线非常简单，我们直接设置上下文对象的 `lineWidth` 属性就行了。但在 WebGL 中，绘制带宽度的曲线是一个难点，很多开发者都在这一步被难住过。

那今天，我就来说说怎么用 Canvas2D 和 WebGL 分绘制曲线。我要特别强调一下，我们讲的曲线指广义的曲线，线段、折线和平滑曲线都包含在内。

如何用 Canvas2D 绘制带宽度的曲线？

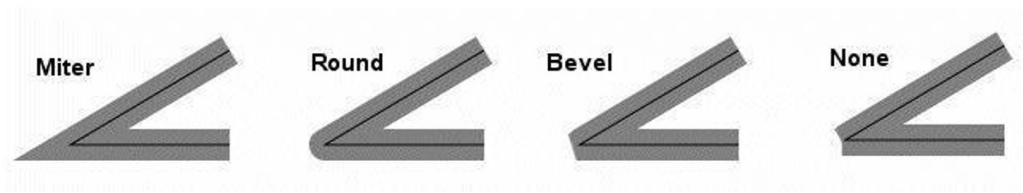
刚才我说了，用 Canvas2D 绘制曲线非常简单。这是为什么呢？因为 Canvas2D 提供了相应的 API，能够绘制出不同宽度、具有特定**连线方式**和**线帽形状**的曲线。

这句话怎么理解呢？我们从两个关键词，“连线方式 (`lineJoin`) ” 和 “线帽形状 (`lineCap`) ” 入手理解。

我们知道，曲线是由线段连接而成的，两个线段中间转折的部分，就是 `lineJoin`。如果宽只有一个像素，那么连接处没有什么不同的形式，就是直接连接。但如果线宽超过一个像素，那么连接处的缺口，就会有不同的填充方式，而这些不同的填充方式，就对应了不同的 `lineJoin`。

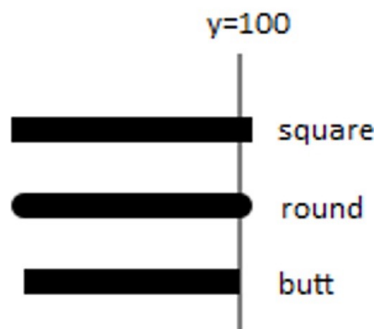


比如说，你可以看我给出的这张图，上面就显示了四种不同的 lineJoin。其中，miter 是尖角，round 是圆角，bevel 是斜角，none 是不添加 lineJoin。很好理解，我就不多说了



4种不同的lineJoin

说完了 lineJoin，那什么是 lineCap 呢？lineCap 就是指曲线头尾部的形状，它有三种类型。第一种是 square，方形线帽，它会在线段的头尾端延长线宽的一半。第二种 round 也叫圆弧线帽，它会在头尾端延长一个半圆。第三种是 butt，就是不添加线帽。



3种不同的lineCap

理解了这两个关键词之后，我们接着尝试一下，怎么在 Canvas 的上下文中，通过设置 lineJoin 和 lineCap 属性，来实现不同的曲线效果。

首先，我们要实现一个 drawPolyline 函数。这个函数非常简单，就是设置 lineWidth、lineJoin、lineCap，然后根据 points 数据的内容设置绘图指令执行绘制。

复制代码

```
1 function drawPolyline(context, points, {lineWidth = 1, lineJoin = 'miter', lineCap = 'butt'}) {
2   context.lineWidth = lineWidth;
3   context.lineJoin = lineJoin;
4   context.lineCap = lineCap;
5   context.beginPath();
6   context.moveTo(...points[0]);
```

```
7   for(let i = 1; i < points.length; i++) {  
8     context.lineTo(...points[i]);  
9   }  
10  context.stroke();  
11 }
```

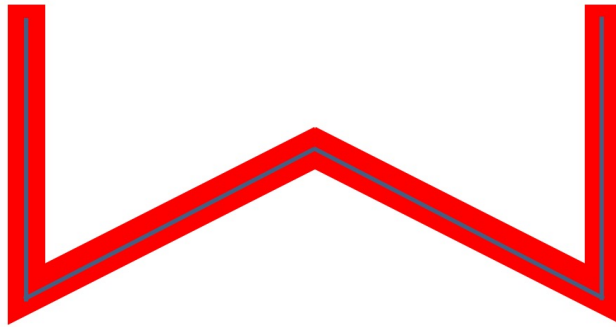
在设置 `lineJoin`、`lineCap` 时候，我们要注意，Canvas2D 的 `lineJoin` 只支持 `miter`、`bevel` 和 `round`，不支持 `none`。`lineCap` 支持 `butt`、`square` 和 `round`。

接着，我们就可以执行 JavaScript 代码绘制曲线了。比如，我们绘制两条线，一条宽度为 10 个像素的红线，另一条宽度为 1 个像素的蓝线，具体的代码：

[复制代码](#)

```
1  const canvas = document.querySelector('canvas');  
2  const ctx = canvas.getContext('2d');  
3  const points = [  
4    [100, 100],  
5    [100, 200],  
6    [200, 150],  
7    [300, 200],  
8    [300, 100],  
9  ];  
10 ctx.strokeStyle = 'red';  
11 drawPolyline(ctx, points, {lineWidth: 10});  
12 ctx.strokeStyle = 'blue';  
13 drawPolyline(ctx, points);
```

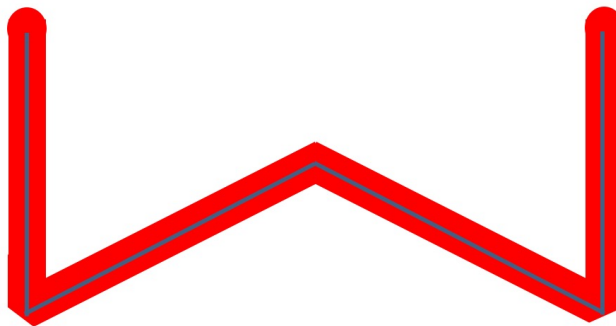
因为我们把连接设置成 `miter`、线帽设置成了 `butt`，所以我们绘制出来的曲线，是尖角并且不带线帽的。



其实，我们还可以修改 `lineJoins` 和 `lineCap` 参数。比如，我们将线帽设为圆的，连接设为斜角。除此之外，你还可以尝试不同的组合，我就不再举例了。

[复制代码](#)

```
1 ctx.strokeStyle = 'red';  
2 drawPolyline(ctx, points, {lineWidth: 10, lineCap: 'round', lineJoin: 'bevel'})
```



除了 `lineJoin` 和 `lineCap` 外，我们还可以设置 `Canvas2D` 上下文的 `miterLimit` 属性，来改变 `lineJoin` 等于 `miter` 时的连线形式，`miterLimit` 属性等于 `miter` 和线宽的最大比值。当我们把 `lineJoin` 设置成 `miter` 的时候，`miterLimit` 属性就会限制尖角的最大值。

那具体会产生什么效果呢？我们可以先修改 `drawPolyline` 代码添加 `miterLimit`。代码如下：

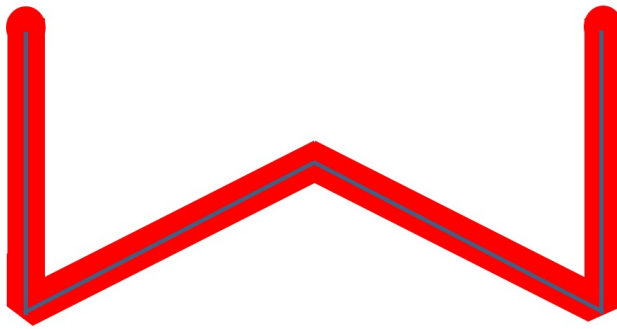
[复制代码](#)

```
1 function drawPolyline(context, points, {lineWidth = 1, lineJoin = 'miter', lin
2   context.lineWidth = lineWidth;
3   context.lineJoin = lineJoin;
4   context.lineCap = lineCap;
5   context.miterLimit = miterLimit;
6   context.beginPath();
7   context.moveTo(...points[0]);
8   for(let i = 1; i < points.length; i++) {
9     context.lineTo(...points[i]);
10  }
11  context.stroke();
12 }
13
```

然后，我们修改参数，把 `miterLimit` 设置为 1.5：

[复制代码](#)

```
1 ctx.strokeStyle = 'red';
2 drawPolyline(ctx, points, {lineWidth: 10, lineCap: 'round', lineJoin: 'miter',
```




你会发现，这样渲染出来的图形，它两侧的转角由于超过了 `miterLimit` 限制，所以表现为斜角，而中间的转角因为没有超过 `miterLimit` 限制，所以是尖角。

总的来说，Canvas2D 绘制曲线的方法很简单，只要我们调用对应的 API 就可以了。但用 WebGL 来绘制同样的曲线会非常麻烦。在详细讲解之前，我希望你先记住 `lineJoin`、`lineCap` 以及 `miterLimit` 这些属性，在 WebGL 中我们需要自己去实现它们。接下来，我们一起来看一下 WebGL 中是怎么做的。

如何用 WebGL 绘制带宽度的曲线

我们先从绘制宽度为 1 的曲线开始。因为 WebGL 本身就支持线段类的图元，所以我们直接用图元就能绘制出宽度为 1 的曲线。

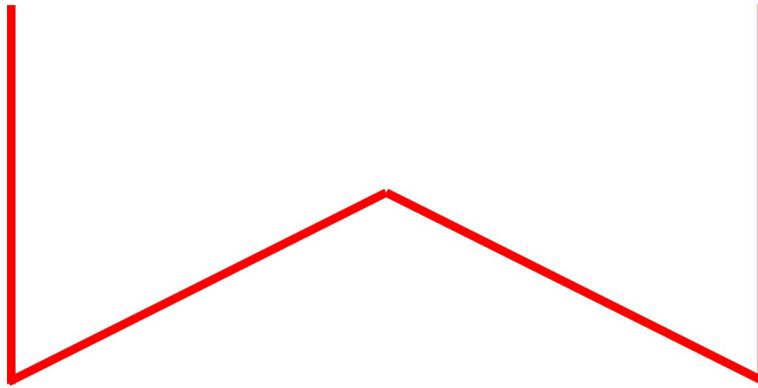
下面，我结合代码来说说具体的绘制过程。与 Canvas2D 类似，我们直接设置 `position` 顶点坐标，然后设置 `mode` 为 `gl.LINE_STRIP`。这里的 `LINE_STRIP` 是一种图元类型，表示以首尾连接的线段方式绘制。这样，我们就可以得到宽度为 1 的折线了。具体的代码和效果如下所示：

 复制代码

```
1 import {Renderer, Program, Geometry, Transform, Mesh} from '../common/lib/ogl/  
2  
3 const vertex = `
```

```
4   attribute vec2 position;
5
6   void main() {
7       gl_PointSize = 10.0;
8       float scale = 1.0 / 256.0;
9       mat3 projectionMatrix = mat3(
10          scale, 0, 0,
11          0, -scale, 0,
12          -1, 1, 1
13      );
14       vec3 pos = projectionMatrix * vec3(position, 1);
15       gl_Position = vec4(pos.xy, 0, 1);
16   }
17 `;
18
19
20 const fragment = `
21     precision highp float;
22     void main() {
23         gl_FragColor = vec4(1, 0, 0, 1);
24     }
25 `;
26
27 const canvas = document.querySelector('canvas');
28 const renderer = new Renderer({
29     canvas,
30     width: 512,
31     height: 512,
32 });
33
34 const gl = renderer.gl;
35 gl.clearColor(1, 1, 1, 1);
36
37
38 const program = new Program(gl, {
39     vertex,
40     fragment,
41 });
42
43 const geometry = new Geometry(gl, {
44     position: {size: 2,
45         data: new Float32Array(
46             [
47                 100, 100,
48                 100, 200,
49                 200, 150,
50                 300, 200,
51                 300, 100,
52             ]
53         )},
54 });
```

```
56 const scene = new Transform();
57 const polyline = new Mesh(gl, {geometry, program, mode: gl.LINE_STRIP});
58 polyline.setParent(scene);
59
60
```



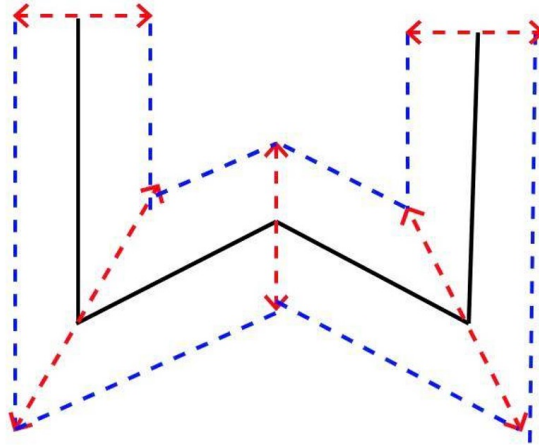
你可能会问，我们不能直接修改 `gl_PointSize`，来给折线设置宽度吗？很遗憾，这是不行的。因为 `gl_PointSize` 只能影响 `gl.POINTS` 图元的显示，并不能对线段图元产生影响。

那我们该怎么让线的宽度大于 1 个像素呢？

通过挤压 (extrude) 曲线绘制有宽度的曲线

我们可以用一种挤压 (Extrude) 曲线的技术，通过将曲线的顶点沿法线方向向两侧移出，让 1 个像素的曲线变宽。

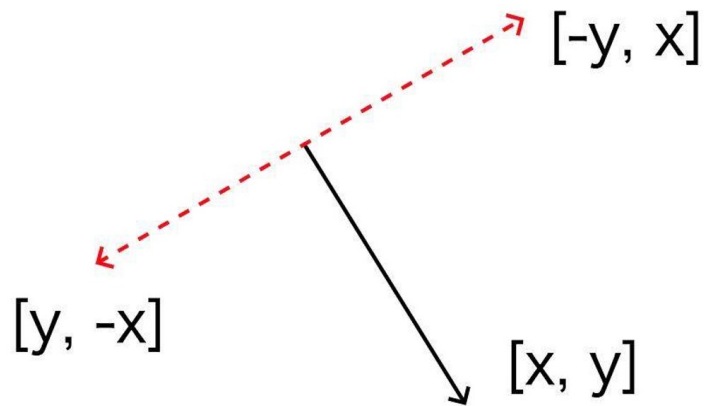
那挤压曲线要怎么做呢？我们先看一张示意图：



挤压线段

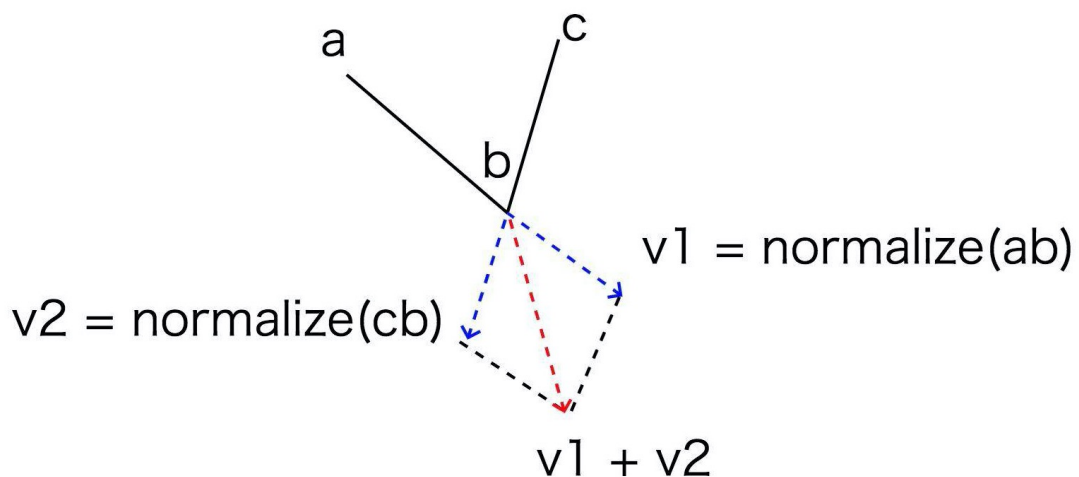
如上图所示，黑色折线是原始的 1 个像素宽度的折线，蓝色虚线组成的是我们最终要生成的带宽度曲线，红色虚线是顶点移动的方向。因为折线两个端点的挤压只和一条线段的方向有关，而转角处顶点的挤压和相邻两条线段的方向都有关，所以顶点移动的方向，我们要分两种情况讨论。

首先，是折线的端点。假设线段的向量为 (x, y) ，因为它移动方向和线段方向垂直，所以我们只要沿法线方向移动它就可以了。根据垂直向量的点积为 0，我们很容易得出顶点的两个移动方向为 $(-y, x)$ 和 $(y, -x)$ 。如下图所示：



折线端点挤压方向

端点挤压方向确定了，接下来要确定转角的挤压方向了，我们还是看示意图。

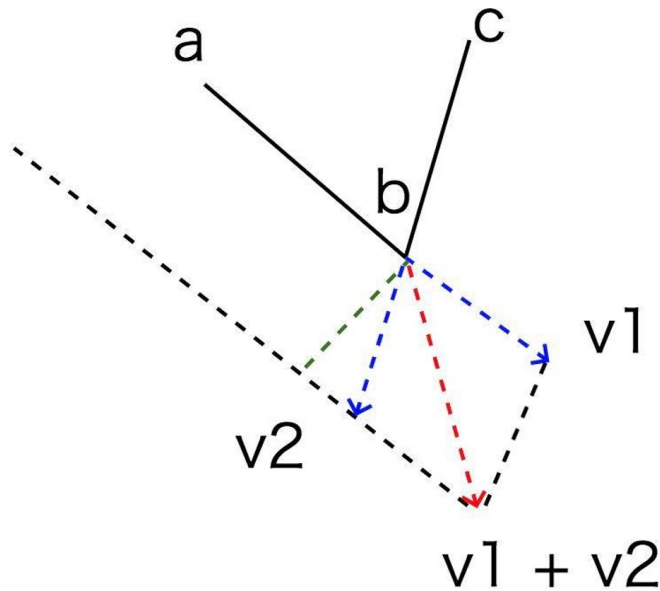


转角的挤压方向示意图

如上图，我们假设有折线 abc ， b 是转角。我们延长 ab ，就能得到一个单位向量 $v1$ ，反向延长 bc ，可以得到另一个单位向量 $v2$ ，那么挤压方向就是向量 $v1+v2$ 的方向，以及相反的 $-(v1+v2)$ 的方向。

现在我们得到了挤压方向，接下来就需要确定挤压向量的长度。

首先是折线端点的挤压长度，它等于 `lineWidth` 的一半。而转角的挤压长度就比较复杂了，我们需要再计算一下。



计算转角挤压长度示意图

绿色这条辅助线应该等于 `lineWidth` 的一半，而它又恰好是 $v1 + v2$ 在绿色这条向量方向的投影，所以，我们可以先用向量点积求出红色虚线和绿色虚线夹角的余弦值，然后用 `lineWidth` 的一半除以这个值，得到的就是挤压向量的长度了。

具体用 JavaScript 实现的代码如下所示：

复制代码

```
1 function extrudePolyline(gl, points, {thickness = 10} = {}) {
2   const halfThick = 0.5 * thickness;
3   const innerSide = [];
4   const outerSide = [];
5
6   // 构建挤压顶点
7   for(let i = 1; i < points.length - 1; i++) {
8     const v1 = (new Vec2()).sub(points[i], points[i - 1]).normalize();
9     const v2 = (new Vec2()).sub(points[i], points[i + 1]).normalize();
10    const v = (new Vec2()).add(v1, v2).normalize(); // 得到挤压方向
11    const norm = new Vec2(-v1.y, v1.x); // 法线方向
12    const cos = norm.dot(v);
```

```

13     const len = halfThick / cos;
14     if(i === 1) { // 起始点
15         const v0 = new Vec2(...norm).scale(halfThick);
16         outerSide.push((new Vec2()).add(points[0], v0));
17         innerSide.push((new Vec2()).sub(points[0], v0));
18     }
19     v.scale(len);
20     outerSide.push((new Vec2()).add(points[i], v));
21     innerSide.push((new Vec2()).sub(points[i], v));
22     if(i === points.length - 2) { // 结束点
23         const norm2 = new Vec2(v2.y, -v2.x);
24         const v0 = new Vec2(...norm2).scale(halfThick);
25         outerSide.push((new Vec2()).add(points[points.length - 1], v0));
26         innerSide.push((new Vec2()).sub(points[points.length - 1], v0));
27     }
28 }
29 ...
30 `

```

在这段代码中，v1、v2 是线段的延长线，v 是挤压方向，我们计算法线方向与挤压方向的余弦值，就能算出挤压长度了。你还要注意，我们要把起始点和结束点这两个端点的挤压也给添加进去，也就是两个 if 条件中的处理逻辑。

这样一来，我们就把挤压之后的折线顶点坐标给计算出来了。向内和向外挤压的点现在分别保存在 innerSide 和 outerSide 数组中。

接下来，我们就要构建对应的 Geometry 对象，所以我们继续添加 extrudePolyline 函数的后半部分。

[复制代码](#)

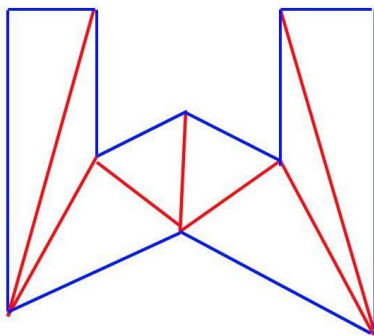
```

1 function extrudePolyline(gl, points, {thickness = 10} = {})
2     ...
3     const count = innerSide.length * 4 - 4;
4     const position = new Float32Array(count * 2);
5     const index = new Uint16Array(6 * count / 4);
6
7     // 创建 geometry 对象
8     for(let i = 0; i < innerSide.length - 1; i++) {
9         const a = innerSide[i],
10            b = outerSide[i],
11            c = innerSide[i + 1],
12            d = outerSide[i + 1];
13
14         const offset = i * 4;
15         index.set([offset, offset + 1, offset + 2, offset + 2, offset + 1, offset

```

```
16     position.set([...a, ...b, ...c, ...d], i * 8);
17   }
18
19   return new Geometry(gl, {
20     position: {size: 2, data: position},
21     index: {data: index},
22   });
23 }
```

这一步骤就非常简单了，我们根据 innerSide 和 outerSide 中的顶点来构建三角网格化的几何体顶点数据，最终返回 Geometry 对象。



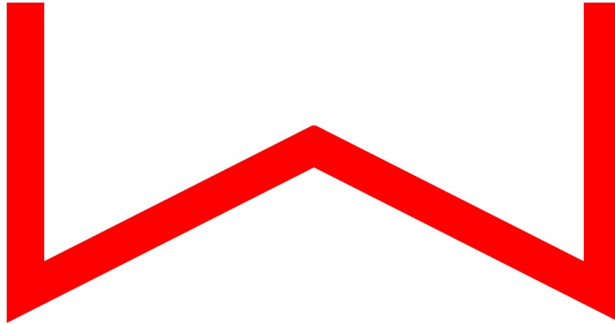
构建折线的顶点数据

最后，我们只要调用 `extrudePolyline`，传入折线顶点和宽度，然后用返回的 `Geometry` 对象来构建三角网格对象，将它渲染出来就可以了。

 复制代码

```
1  const geometry = extrudePolyline(gl, points, {lineWidth: 10});
2
3  const scene = new Transform();
4  const polyline = new Mesh(gl, {geometry, program});
5  polyline.setParent(scene);
6
7  renderer.render({scene});
```

我们最终渲染出来的效果如下图：



这样，我们就在 WebGL 中实现了，与 Canvas2D 一样带宽度的曲线。

当然，这里我们只实现了最基础的带宽度曲线，它对应于 Canvas2D 中的 `lineJoin` 为 `miter`，`lineCap` 为 `butt` 的曲线。不过，想要实现 `lineJoins` 为 `bevel` 或 `round`，`lineCap` 为 `square` 或 `round` 的曲线，也不会太困难。我们可以基于 `extrudePolyline` 函数，对它进行扩展，计算出相应属性下对应的顶点就行了。因为基本原理是一样的，我就不详细说了，我把扩展的任务留给你作为课后练习。

要点总结

这节课，我们讲了绘制带宽度曲线的方法。

首先，在 Canvas2D 中，绘制这样的曲线比较简单，我们直接通过 API 设置 `lineWidth` 即可。而且，Canvas2D 还支持不同的 `lineJoin`、`lineCap` 设置以及 `miterLimit` 设置。

在 WebGL 中，绘制带宽度的曲线则比较麻烦，因为没有现成的 API 可以使用。这个时候，我们可以使用挤压曲线的技术来得到带宽度的曲线，挤压曲线的具体步骤可以总结为三步：

1. 确定端点和转角的挤压方向，端点可以沿线段的法线挤压，转角则通过两条线段延长线的单位向量求和的方式获得。
2. 确定端点和转角挤压的长度，端点两个方向的挤压长度是线宽 `lineWidth` 的一半。求转角挤压长度的时候，我们要先计算方向向量和线段法线的余弦，然后将线宽 `lineWidth` 的一半除以我们计算出的余弦值。
3. 由步骤 1、2 计算出顶点后，我们构建三角网格化的几何体顶点数据，然后将 `Geometry` 对象返回。

这样，我们就可以用 WebGL 绘制出有宽度的曲线了。

小试牛刀

1. 你能修改 `extrudePolyline` 函数，让它支持 `lineCap` 为 `square` 和 `round` 吗？或者让它支持 `lineJoin` 为 `round` 吗？
2. 我想让你试着修改一下 `extrudePolyline` 函数，让它支持 `lineJoin` 为 `bevel`，以及 `miterLimit`。并且，当 `lineJoin` 为 `miter` 的时候，如果转角挤压长度超过了 `miterLimit`，我们就按照 `bevel` 处理向外的挤压。

那通过今天的学习，你是不是已经学会绘制带宽度曲线的方法。那不妨就把这节课分享给你的朋友，也帮助他解决这个难题吧。好了，今天的内容就到这里了，我们下节课再见

源码

课程完整示例代码详见 [🔗 GitHub 仓库](#)

提建议

跟月影学可视化

系统掌握图形学与可视化核心原理

月影

奇虎 360 奇舞团团长

可视化 UI 框架 SpriteJS 核心开发者



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 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 25 | 如何用法线贴图模拟真实物体表面

下一篇 27 | 案例：如何实现简单的3D可视化图表？

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。