Search

TWAVER DOCUMENT CENTER

- 开发指南 – TWaver HTML5 3D
  - 功能列表及性能指标
  - 3D实战：一步一步学3D开发
  - 3D开发基础
  - 3D对象详解
  - 3D高级开发
  - 附录
    - 3D对象格式说明
    - 常见问题解答
    - API文档

TWAVER DOCUMENT CENTER › 开发指南 – TWaver HTML5 3D › 3D对象详解

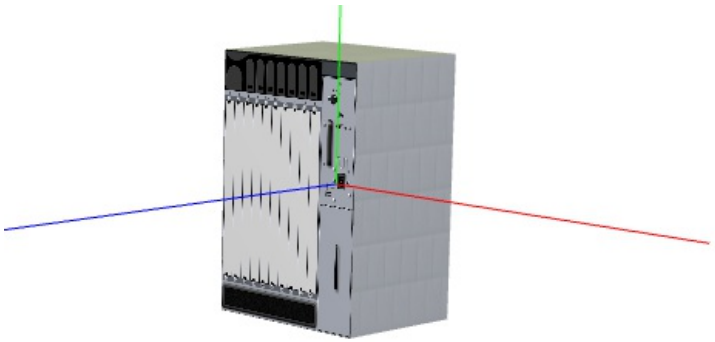
mono.Cube——立方体

Print

IN THIS PAGE [hide]

1 mono.Cube——立方体
2 mono.Sphere——球体
3 mono.Cylinder——圆柱体
4 mono.Torus——圆环
5 mono.PathNode——路径体
6 mono.TextNode——文字
7 mono.ShapeNode——形状块
8 mono.PathCube——路径方块
9 mono.Particle——粒子系统
10 mono.CSG——运算体
11 mono.Billboard——公告牌
12 mono.LatheNode——车削对象
13 mono.Plane——平面对象
14 mono.ComboNode —— 组合体
15 mono.Light——光源对象
16 mono.Terrain——地形对象
17 mono.Line——线条对象

立方体对象（mono.Cube）是一个由6个面组成的立方体，是使用中最常见的3D物体。通过贴图设置，可以用来表示电信中例如设备、机架、仪表等常见物体。立方体的自身坐标原点在其中心位置，如下图：



立方体的每一个面都可以单独设置材质图片、是否纹理、纹理重复次数等等。6个面的名字分别用前、后、左、右、上、下进行区分，具体key值如下。

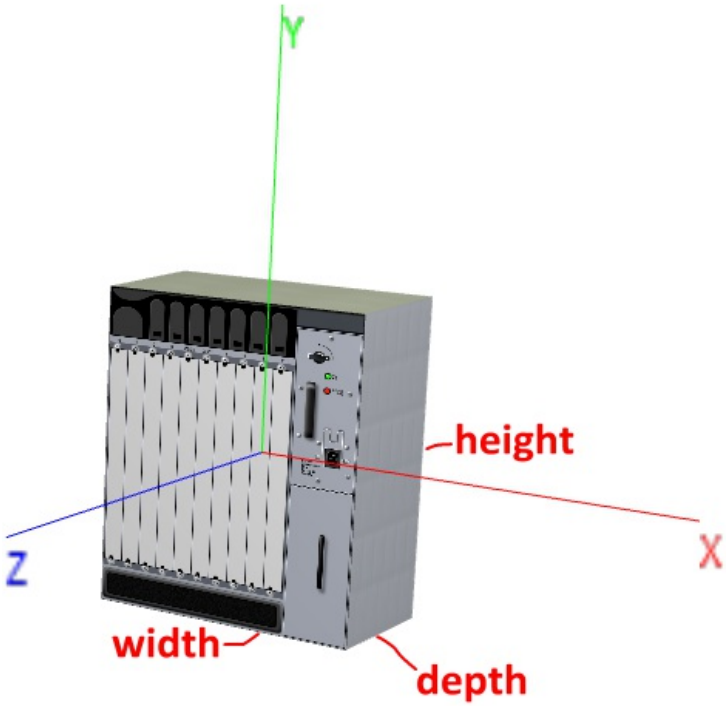
- left：左侧面。例如：设置左侧面贴图，node.setStyle('left.m.texture.image', 'picture.png');
- right：右侧面。例如：设置右侧面贴图，node.setStyle('right.m.texture.image', 'picture.png');
- front：前面（正面）。例如：设置正面贴图，node.setStyle('front.m.texture.image', 'picture.png');
- back：后面（背面）。例如：设置背面贴图，node.setStyle('back.m.texture.image', 'picture.png');
- top：顶面。例如：设置顶面贴图，node.setStyle('top.m.texture.image', 'picture.png');

- bottom：底面。例如：设置底面贴图，node.setStyle('bottom.m.texture.image', 'picture.png');

如果设置style时不指定哪个面前缀，则表示应用于所有面。例如，下面代码将texture-picture.png设置为所有面的贴图，而将左侧单独设置为left-picture.png贴图：

```
node.setStyle('m.texture.image', 'texture-picture.png')
node.setStyle('left.m.texture.image', 'left-picture.png')
```

立方体的尺寸分别为width、height、depth，分别对应x轴的宽度、y轴的高度、z轴的深度。如下图：

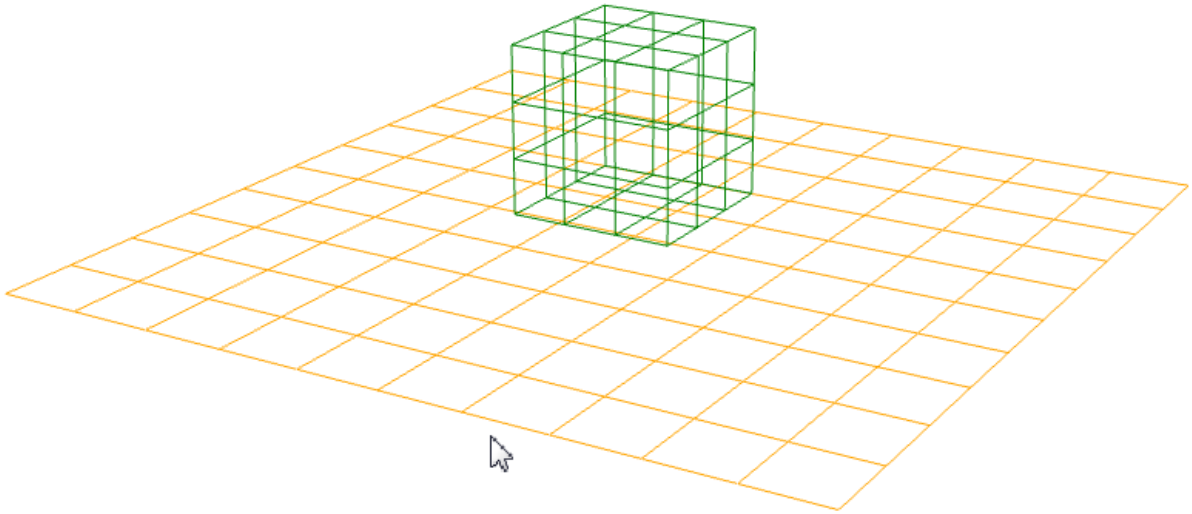


立方体的旋转可以通过setRotation(x, y, z)完成，分别指定在三个轴的旋转角度即可。例如，node.setRotation(Math.PI/2, 0 0)表示将node在x轴旋转90度。也就是：将右手握住x轴，将node沿着手指旋转的方向旋转90度。

```
//构造方法
var node = new mono.Cube(width, height, depth,segmentsW, segmentsH, segmentsD);
//其中参数分别是立方体的宽、高、深、横向切片数量、纵向切片数量和深度切片数量，默认情况下都为1。
```

```
//创建一个segments为3的cube
 var cube = new mono.Cube(100,100,100,3,3,3);
          cube.setStyle('m.wireframe',true).setStyle('m.color','green');
          cube.setPosition(0,70,0);
          cube.setSelectable(false);
          box.add(cube);
```

效果如下：



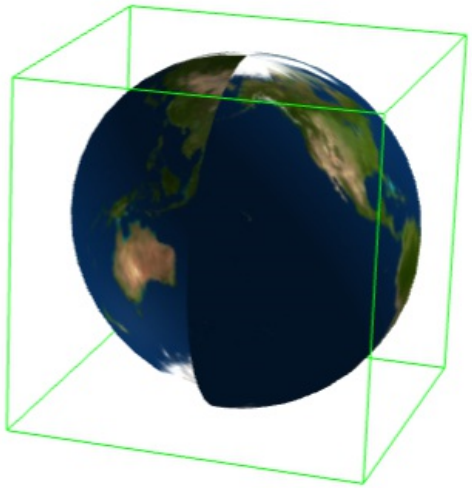
mono.Sphere——球体

球体对象（mono.Sphere）表面是一个曲面，由半径控制其尺寸，其曲面是由一系列纵向+横向均匀切分的小平面拼接而成，圆滑度取决于切分横向和纵向的分片数量，数量越大曲面越圆滑但性能越低。球体的坐标位于其内部中心点。

球体可以在横向和纵向方向上设置开始角度、延伸角度，来创建不完整的球体形状。

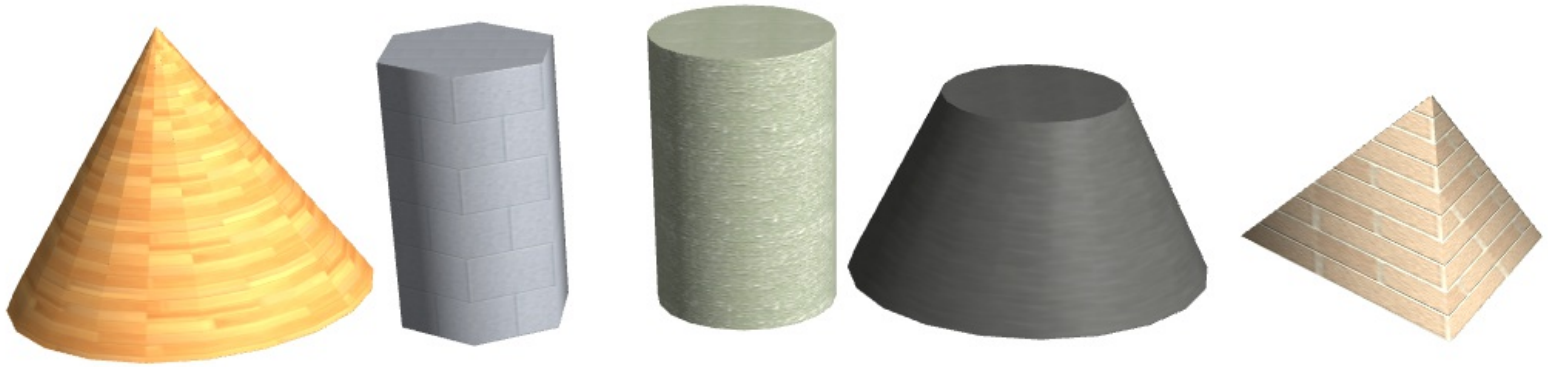
```
1 //Sphere构造函数
2 var node = new mono.Sphere(radius, segmentsW, segmentsH, longitudeStart, longitudeLength, latitudeStart, latitudeL
3 //radius - 球体半径
4 //segmentsW - 横向切片数量，默认值22
5 //segmentsH - 纵向切片数量，默认值15
```

```
//longitudeStart - 经度（纵向）起始角度
//longitudeLength - 经度（纵向）延伸角度
//latitudeStart - 纬度（横向）起始角度
//latitudeLength - 纬度（横向）延伸角度

//下面代码创建了一个不完整的球体
var node=new mono.Sphere(50, 20, 20, 0, Math.PI*1.5, 0, Math.PI);
```



mono.Cylinder——圆柱体

圆柱体对象（mono.Cylinder）表面是由一个纵向直线、横向圆形的曲面组成，由长宽高控制其尺寸，还可设置顶面和底面是否封闭。需要留意的是，圆柱的顶面和底面的尺寸可以不同。侧面圆滑度由分片数量进行控制，分片数量越大侧面越圆滑但性能越低。圆柱体可以有一些特别用法：圆锥（顶面半径为0）、金字塔形状（顶面半径0、侧面分片为4）、半金字塔形状（顶面半径为底面半径的一半、侧面分片为4）等等。可以通过灵活设置上下面的半径、侧面分片数量，来创建很多变种的圆柱体。以下形状都是各种圆柱的变体：



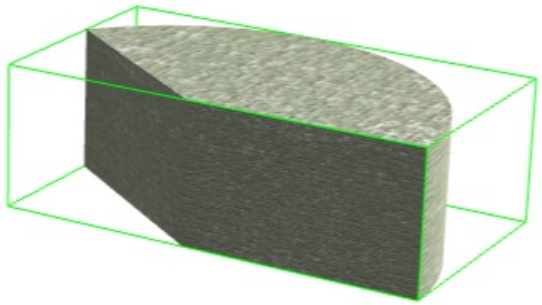
圆柱体的材质分为顶面、底面和侧面，具体key值如下。

- top：顶面。例如：设置顶面贴图，node.setStyle('top.m.texture.image', 'picture.png');
- bottom：底面。例如：设置底面贴图，node.setStyle('bottom.m.texture.image', 'picture.png');
- side：侧面。例如：设置侧面贴图，node.setStyle('side.m.texture.image', 'picture.png');

如果设置style时不指定哪个面前缀，则表示应用于所有面。例如，下面代码将texture-picture.png设置为所有面的贴图，而将侧面单独设置为side-picture.png贴图：

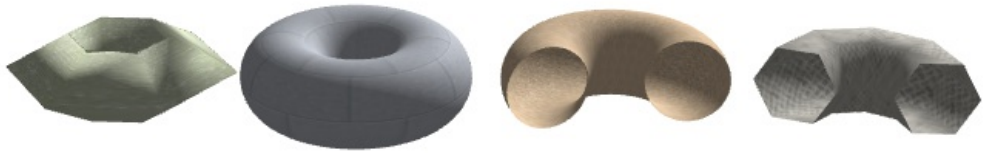
```
node.setStyle('m.texture.image', 'texture-picture.png')
node.setStyle('side.m.texture.image', 'side-picture.png')
```

```
//构造函数
var node = new mono.Sphere(radiusTop, radiusBottom, height, segmentsR, segmentsH, openTop, openBottom,arcLeng
//radiusTop - 顶面圆柱
//radiusBottom - 底面圆柱
//height - 圆柱高度
//segmentsR - 侧面（圆弧）分片数量，默认20
//segmentsH - 高度分片数量。一般高度上不需要分片，默认1
//openTop - 顶面是否镂空。true为镂空，false为封闭
//openBottom - 底面是否镂空。true为镂空，false为封闭
//arcLength - 圆柱的圆弧所占长度。例如Math.PI*2则为完整圆柱，Math.PI/2则为1/4圆柱体
//arcStart - 圆柱的圆弧开始角度

//举例：下面代码创建了一个不完整的圆柱体
var node=new mono.Cylinder(50, 50, 30, 20, 1, false, false,Math.PI*0.8,0);
```



mono.Torus——圆环

圆环（mono.Torus）对象是一个类似“手镯”形状的圆环物体。它的切面和形状都是圆形，其圆滑度都可通过切片数量进行控制。此外，圆环还可以通过角度控制其是否封闭。360度的圆环是一个封闭的圆环，而180度的圆环则是一个被切去一半的半个圆环残体。如果将切面切片和环形切片都设置成3，则会形成一个类似三角管体组成的三角形形状体。典型用法：弧形门把手、弯管连接头体、各种圆环形状体等。以下都是各种圆环的变形体。



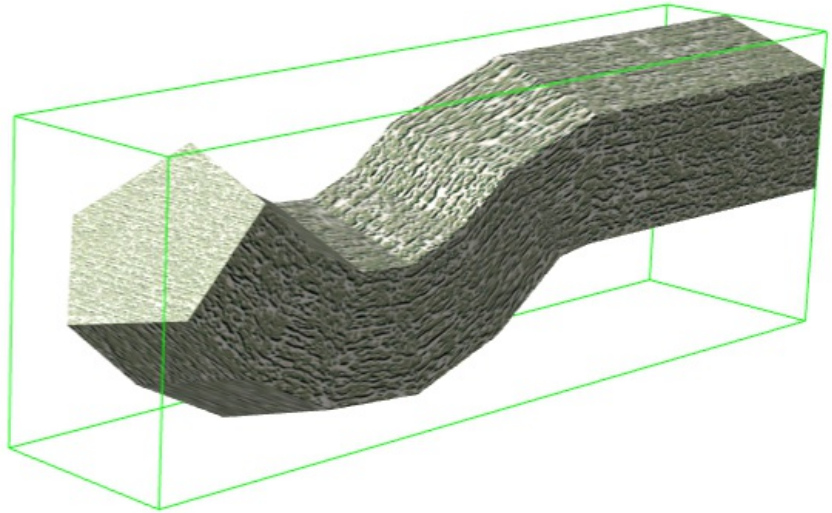
```
1 //构造函数
2 var node = new mono.Torus(radius, tube, segmentsR, segmentsT, arc);
3 //radius - 圆环的半径尺寸
4 //tube - 横切面圆弧的半径尺寸
5 //segmentsR - 圆环的分片数量，默认值8
6 //segmentsT - 横切面圆弧的分片数量, 默认值6
7 //arc - 圆环体所占角度。例如：Math.PI*2则为完整一圈，Math.PI则为半圈
```

mono.PathNode——路径体

路径体（mono.PathNode）这是一种复杂的形状体，由两个任意形状进行控制：切面形状，以及前进走向。最终形状是该切面形状沿着前进走向进行移动而形成的物体。例如，一个圆形切面沿着一个多边形移动，就会形成一个复杂的管线物体。这种形状还可以控制两个端头是否封闭、封闭的形状和尺寸，横切方向是否闭合、闭合角度、闭合样式等。通过控制这些参数，可以创建例如管线、弯管、香肠体、切开的管线等。

下面图中的路径体，路径是一个由直线、曲线等组成的路线，横切面是一个5个分片的圆形。

```
var path = new mono.Path();
path.moveTo(0,100,0);
path.lineTo(200,100,0);
path.curveTo(300,0,0, 400,100,0);

var node=new mono.PathNode(path, 10, 50, 5);
```

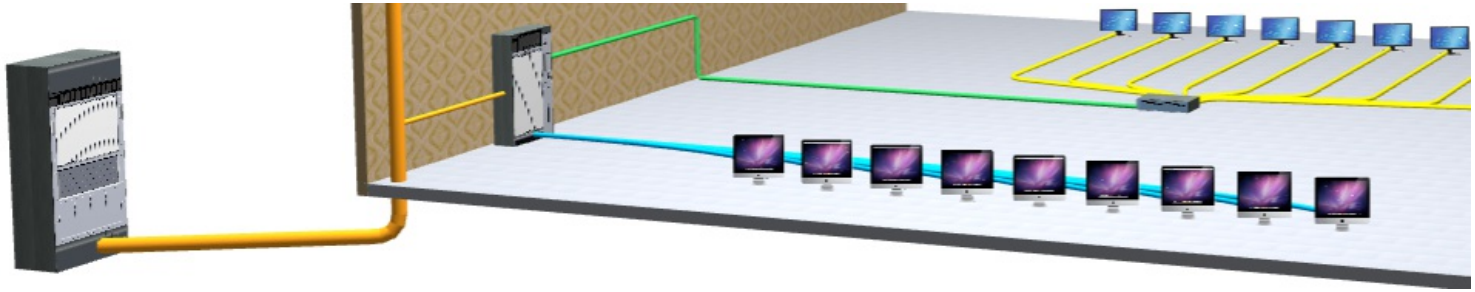


在创建较长路径的管线时，手工指定管线拐角处的曲线弧度会比较繁琐。mono提供了一个内置的方法：

PathNode.adjustPath可以自动将给定的路径进行曲线处理。

```
var path = new mono.Path();
//...
path = mono.PathNode.prototype.adjustPath(path, radius, times);
//其中radius是自动弯角的弧度，times是弯角的分片数量。例如：
path = mono.PathNode.prototype.adjustPath(path,0.5,20);
```

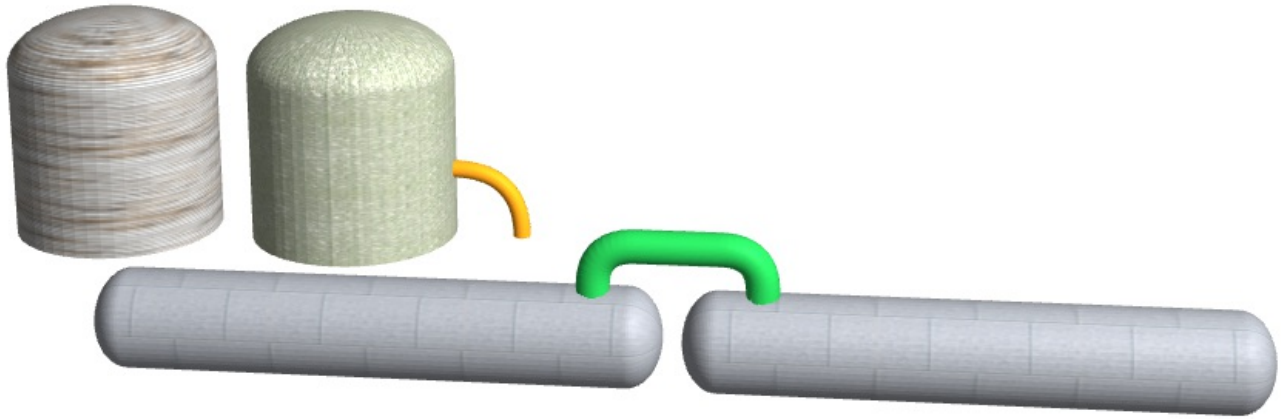
下图展示了使用adjustPath函数的效果：



路径体可以设置两个端头（开始端头、结束端头）是否封闭、平面封闭还是弧形封闭、封闭弧形的尺寸等等。

- 设置端头样式：setStartCap/setEndCap(capStyle); capStyle可以取：’none’ 无封闭（开放）、’plain’ ：平面封闭、’round’：半球形封闭
- 设置端头封闭面尺寸：setStartCapSize/setEndCapSize(size)；size为1，则封闭面尺寸为横截面半径的1倍，2则为2倍，以此类推

下图中的所有对象都是由设置了端头样式的PathNode对象组成：



下面代码使用了弧形封闭且设置了封闭面的尺寸大小：

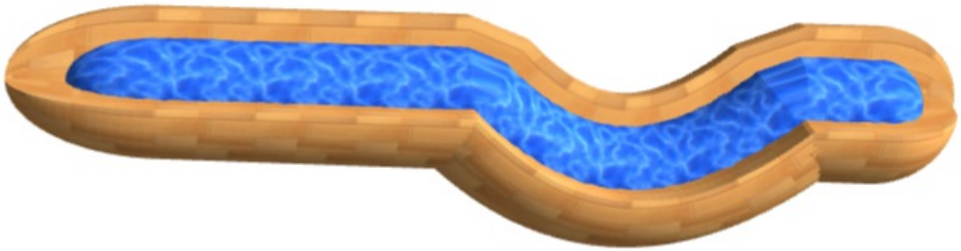
```
1  var path = new mono.Path();
2  path.moveTo(0,100,0);
3  path.lineTo(200,100,0);
4  path.curveTo(300,0,0, 400,100,0);
5  path.lineTo(450,100,0);
6
7  var node=new mono.PathNode(path, 100, 50, 100,'round', 'round');
8  node.setStartCapSize(5);
9  node.setEndCapSize(1);
10 node.setStyle('m.texture.image','../images/wall01_inner_3d.png').setStyle('m.type','phong').setStyle('m.side',mono.Doub
```

路径体可以沿着径向，设置圆弧的完整度。通过构造函数指定arc、arcStart、cutSurface可以设置完整度的长度、角度、样式。

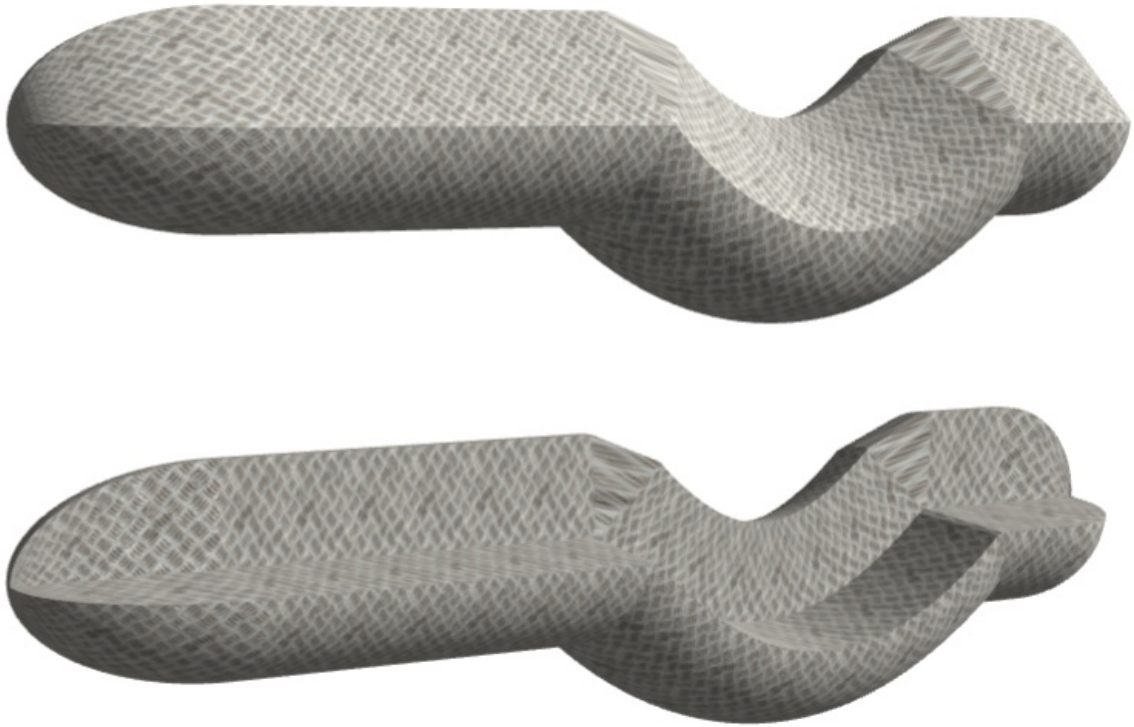
- setArc：横切面圆弧完整度弧度。例如2*Math.PI是完整圆形。
- setArcStart：设置横切面圆弧开始角度。
- setCutSurface：设置不完整弧度切割面的样式。’none’为镂空、’plain’为直连平面、’center’为直中心线拐角切面。

下图演示了不完整弧度、切面样式的例子。用一个路径构造了两个管子，小尺寸管保持完整，大尺寸管切掉1/4：

```
1  var path = new mono.Path();
2  path.moveTo(0,100,0);
3  path.lineTo(200,100,0);
4  path.curveTo(300,0,0, 400,100,0);
5  path.lineTo(450,100,0);
6
7  var node=new mono.PathNode(path, 100, 50, 100,'round', 'round', null, Math.PI*1.5, Math.PI/2, 'center');
8  node.setStartCapSize(2);
9  node.setEndCapSize(1);
10 node.setStyle('m.texture.image','../images/wall01_inner_3d.png').setStyle('m.type','phong').setStyle('m.side',mono.Doub
11 box.add(node);
12
13 node=new mono.PathNode(path, 100, 30, 100,'round', 'round');
14 node.setStartCapSize(2);
15 node.setEndCapSize(1);
16 node.setStyle('m.texture.image','../images/water-texture-1.png').setStyle('m.type','phong').setStyle('m.side',mono.Doubl
17 box.add(node);
```

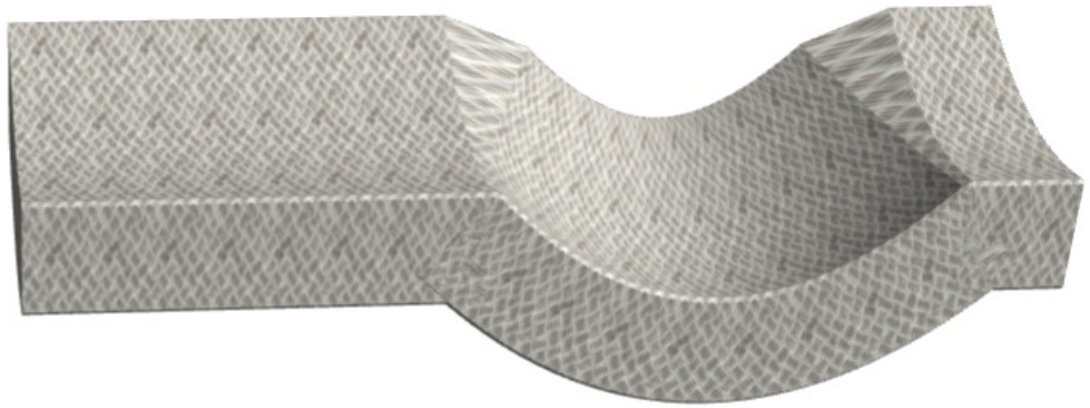


下图中则是展示了cutSurface分别使用’plain’和’center’时的情形：

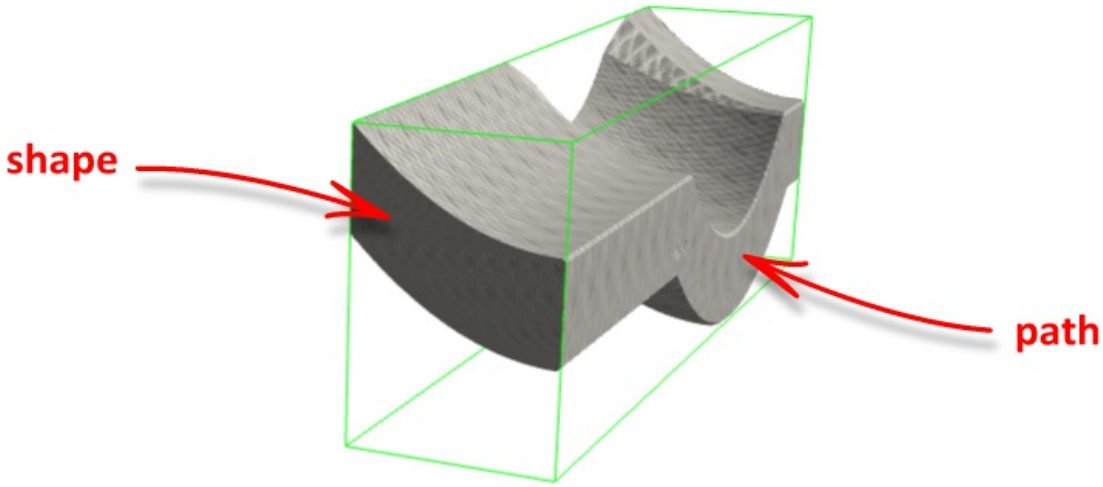


路径体还有一种特别的用法，就是不但可以指定路径走向，还可以指定横切面的任意形状，这个形状也可以通过一个任意路径的走向来定义。这样，就可以创建各种横截面形状的“管子”。下图展示了这一用法：

```
1  var path = new mono.Path();
2  path.moveTo(0,100,0);
3  path.lineTo(200,100,0);
4  path.curveTo(300,0,0, 400,100,0);
5  path.lineTo(450,100,0);
6
7  var shape = new mono.Path();
8  shape.moveTo(0,0,0);
9  shape.curveTo(50,50,0, 100,50,0);
10 shape.lineTo(100,0,0);
11 shape.curveTo(50,0,0, 0,-50,0);
12 shape.lineTo(0, 0,0,0);
13
14 var node=new mono.PathNode(path, 100, 50, 100,'plain', 'plain', shape);
15 node.setStyle('m.texture.image','../images/metal02.png').setStyle('m.type','phong').setStyle('m.side',mono.DoubleSide).
16 box.add(node);
```



其中，横截面形状、路径走向如下图所示：



//构造函数

```
var node = new mono.PathNode(path, segments, radius, segmentsR, startCap, endCap, shape, arc, arcStart, cutSurface
```

//path – 路径走向

//segments – 路径方向分片数量，默认64。**注意：分片数量沿路径方向等距分割，无论路径形状如何。因此，此数值过小可能会影响路径方向形状的圆滑度。过大则会降低效率。开发中应仔细调整合适数值**

//radius – 横截面圆形半径

//segmentsR – 横截面圆形分片数量，默认值8

//startCap – 起始端头样式，’none’镂空、’plain’平面封闭、’round’弧形面封闭

//endCap – 结束端头样式，’none’镂空、’plain’平面封闭、’round’弧形面封闭

//shape – 横截面形状。如不设置则横截面为圆形。默认值空
//arc – 横截面所占弧度。默认为Math.PI*2，完整圆形
//arcStart – 横截面圆弧开始角度，默认0度开始
//cutSurface – 不完整弧度切割面的样式。'none'为镂空、'plain'为直连平面、'center'为直中心线拐角切面

mono.TextNode——文字

文字（mono.TextNode）物体是一串3D化的文字字符。文字在3D中并没有直接的支持方式，所以和其他复杂形状一样，需要给出其具体的形状然后进行切片、分片来模拟。由于3D中没有直接的字体数据，需要额外提供字体的具体形状。在mono中没有内置任何字体信息，需要开发者额外去创建并引入到程序中。在MONO Design编辑器中，提供了几种英文字体形状信息，存储在类似"***_regular.typeface.js"的js文件中。例如，"helvetiker_regular.typeface.js"文件中存储了helvetiker字体的正常体信息，"helvetiker_bold.typeface.js"文件中存储了helvetiker字体的粗体信息，等等。如需要更多的字体，可以到网站http://typeface.neocracy.org/在线提交、生成和下载。一般一个英文字体对应的js文件大约在数百kb左右。由于中文字体形状复杂、字符数量巨大，一般不建议使用中文字符。典型用法：各种文字标识、设备标签。

MONO Design编辑器中提供的字体资源有：

- gentilis_bold.typeface.js
- gentilis_regular.typeface.js
- helvetiker_bold.typeface.js
- helvetiker_regular.typeface.js
- kaushan_script_regular.typeface.js
- optimer_bold.typeface.js
- optimer_regular.typeface.js

其中regular为正常体、bold为粗体。开发者可以将这些js引入程序中直接使用。下图显示了一个具有纹理、染色效果的文字字符串：


```
1 //构造函数
2 var node = new mono.TextNode(text, size,height,font,weight);
3 //text - 文字字符串。请注意要使用字体中包含的字符。例如使用一个英文字体就不能输入中文字符
4 //size - 文字的尺寸。数值越大，文字越高大
5 //height - 文字的厚度（深度）。请注意该参数并不影响文字大小，而是影响文字的厚度
6 //font - 文字字体名称。例如：引入的字体文件是helvetiker_bold.typeface.js，则font设置为'helvetiker'
7 //weight - 粗体或正常体。数值'normal'为正常体,'bold'为正常体
```

下面代码创建了一个字符串，并设置纹理、显示：

```
var node=new mono.TextNode('TWaver Mono Design', 10,1,'optimer','bold');
node.setStyle('m.texture.image','../images/metal02.png').setStyle('m.type','phong').setStyle('m.repeat',new mono.Vec2(3
box.add(node);
```

使用setFont改变textNode的字体。

```
node.setFont('gentilis');
```

mono.ShapeNode——形状块

形状块（mono.ShapeNode）是一个有厚度的、任意形状的3D物体。典型应用：地板、地图块等。下图是用形状块显示的美国Ohio州的3D轮廓图：

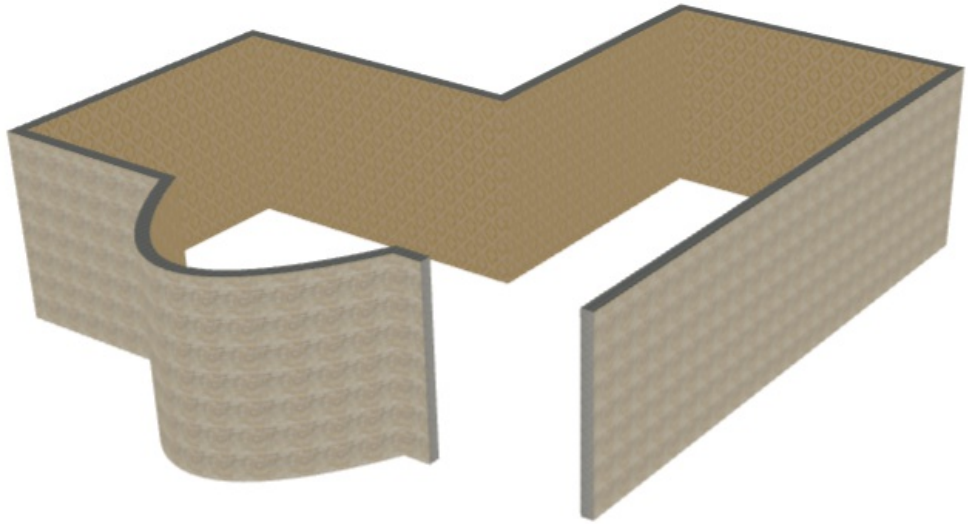
```
//构造函数
var node = new mono.ShapeNode(shapes, curveSegments, amount, horizontal, repeat);
//shapes - 块图形状
//curveSegments - 轮廓边缘分片数量
//amount - 块图的厚度（深度）
//vertical - 形状是水平方向还是垂直方向。true为垂直，false为水平。默认true，垂直
//repeat - 重复纹理块的尺寸大小。越大，纹理图片重复次数越少
```

上面Ohio州轮廓图，可以用下面代码创建：

```
var path=new mono.Path();

path.moveTo(734.98, 266.32);
path.lineTo(718.95, 262.25);
path.lineTo(714.20, 254.88);
path.lineTo(707.70, 255.09);
path.lineTo(708.08, 194.67);
path.lineTo(731.26, 194.02);
path.lineTo(743.43, 198.46);
path.lineTo(738.57, 200.39);
path.lineTo(747.81, 201.93);
path.lineTo(762.18, 199.56);
path.lineTo(783.77, 187.79);
path.lineTo(783.74, 219.64);
path.lineTo(781.16, 220.93);
path.lineTo(782.28, 227.36);
path.lineTo(777.38, 243.14);
path.lineTo(763.05, 251.49);
path.lineTo(761.76, 259.03);
path.lineTo(759.44, 260.32);
path.lineTo(756.78, 257.11);
path.lineTo(751.71, 270.12);
path.lineTo(747.34, 270.98);
path.lineTo(741.76, 263.29);
path.lineTo(734.98, 266.32);

var node=new mono.ShapeNode(path, 100, 10);
node.setPositionX(-700);
node.setPositionY(-150);
node.setStyle('m.texture.image','../images/grass.png').setStyle('m.type','phong');
```

mono.PathCube——路径方块

路径方块物体（mono.PathCube）是一个矩形沿着某个路径移动形成的3D物体。它的纵切面是一个矩形，并沿着一个路径方向进行建模。它最常见的用途就是各种**房间的墙体**。下图是一个典型的路径方块：



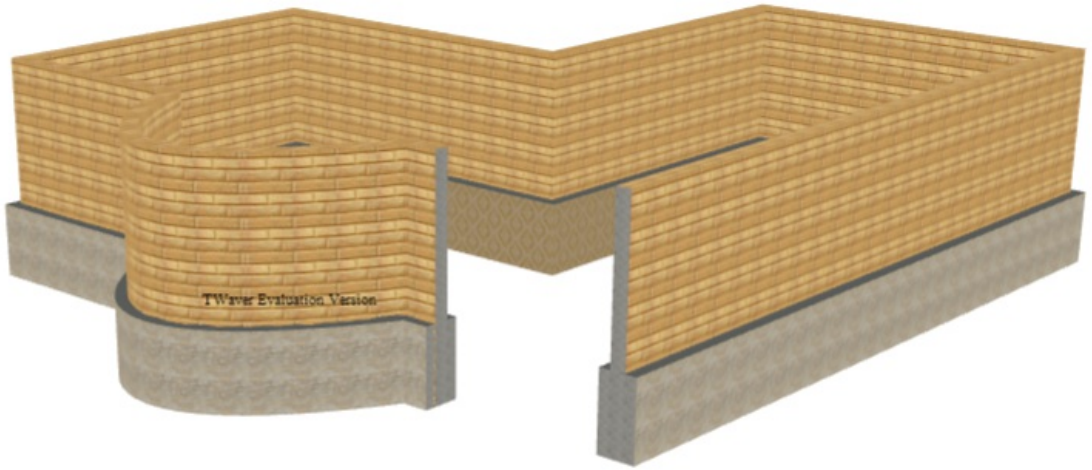
```
//构造函数
var node = new mono.PathCube(path, width, height, curveSegements, repeat);
//path - 走向路径
//width - 横截面立方体宽度
//height - 横截面立方体高度
//curveSegements - 路径方向分片数量
//repeat - 纹理贴图多少尺寸重复一次。例如10则贴图会每隔10重复一次。越大则纹理单位贴图越大。
```

例如，要创建一个上图形状的墙体，并增加一个墙体加强的地基，可以如此写代码：

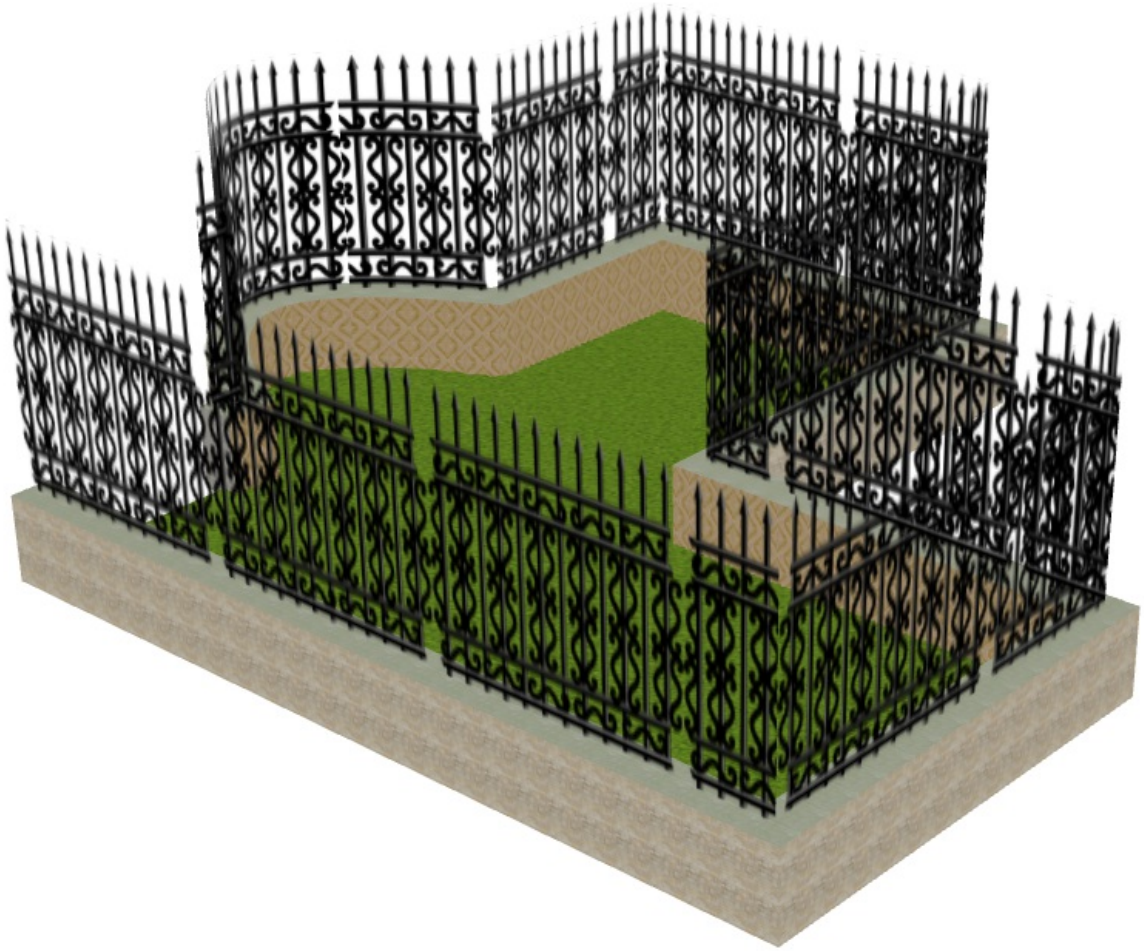
```
var path = new mono.Path();
path.moveTo(0, 0,0);
path.lineTo(1000, 0, 0);
path.lineTo(1000, 500, 0);
path.lineTo(500, 500, 0);
path.lineTo(500, 1000, 0);
path.lineTo(0, 1000, 0);
path.lineTo(0, 700, 0);
path.curveTo(-400,500,0,0,300,0);
path.lineTo(0,250,0);

var wall = new mono.PathCube(path,20,300,32,40);
wall.setStyle('m.texture.image', '../images/wall02_3d.png');
wall.setStyle('inside.m.texture.image', '../images/wall02_3d.png');
wall.setStyle('top.m.texture.image', '../images/wall02_3d.png');
wall.setStyle('bottom.m.texture.image', '../images/wall02_3d.png');
wall.setStyle('aside.m.texture.image', '../images/metal02.png');
wall.setStyle('zside.m.texture.image', '../images/metal02.png');
box.add(wall);

wall = new mono.PathCube(path,50,100,32,40);
wall.setStyle('m.texture.image', '../images/wall04_3d.png');
wall.setStyle('inside.m.texture.image', '../images/wall01_inner_3d.png');
wall.setStyle('top.m.texture.image', '../images/metal08.png');
wall.setStyle('bottom.m.texture.image', '../images/metal08.png');
wall.setStyle('aside.m.texture.image', '../images/metal02.png');
wall.setStyle('zside.m.texture.image', '../images/metal02.png');
```

```
28 box.add(wall);
```

运行效果如下：



还可以使用png局部透明图片，来模拟现实世界中的例如篱笆等特殊墙体。将墙体换成有透明区域的图片后，运行上述例子，效果如下：



mono.Particle——粒子系统

粒子系统（mono.Particle）是一个非常特别的3D物体。和其他复杂形状的3D物体不同，它没有很多具体的面，而只是由一系列内部的顶点组成，每个顶点显示给定的贴图。当它处于静止状态时，它显示一堆空间离散分布的“颗粒物”；当动画来控制顶点的数量和位置，可以实现一种复杂运动的粒子效果。它非常适合模拟例如烟雾、火焰、喷水等效果。

```
1  //构造函数
2  var smoke = new mono.Particle(vertices,colors, material);
3  //vertices - 顶点坐标数组
4  //colors - 颜色数组，每个顶点的颜色
5  //material - 顶点贴图。所有顶点使用相同贴图
6
7  //一般而言，为了让粒子系统更逼真、高效，还需要仔细设置以下参数：
8  smoke.sortParticles = false; //忽略粒子系统中顶点重新排序。重要！
9  smoke.setStyle('m.depthTest',false); //忽略深度检测。重要！
```

```
smoke.setStyle('m.transparent',true); //启用材质透明。材质图片中的透明区域（例如png图片）会有透明、半透明效果。重要！
smoke.setStyle('m.opacity',0.1); //设置材质的不透明度。强行让材质的不透明度为0.1（也就是90%的透明度）。注意：此属性和材质

smoke.setStyle('m.color',0xffffFF); //材质染色，可以通过该颜色对烟雾图片进行染色
smoke.setStyle('m.size',40); //材质的贴图尺寸
smoke.setStyle('m.texture.image','../images/smoke2.png');//设置材质图片
```

下面代码创建了一个静态的、200个顶点的粒子系统：

```
var sphereRadius = 80;
var particleCount = 200;
var smoke = new mono.Particle();
smoke.setClient("sphereRadius", sphereRadius);

for (var p = 0; p &lt; particleCount; p++) {
  var radius = sphereRadius;
  var angle = Math.random() * (Math.PI * 2);
  var pX = Math.sin(angle) * radius, pY = Math.random() * radius, pZ = Math.random() * radius, particle = new mono.V
  particle.velocity = new mono.Vec3(Math.random()*2, Math.random()*2, 0);
  smoke.vertices.push(particle);
}

smoke.sortParticles = false;
smoke.setStyle('m.color',0xffffFF).setStyle('m.size',40).setStyle('m.transparent',true).setStyle('m.opacity',0.1).setStyle('
smoke.setStyle('m.depthTest',false).setStyle('m.depthWrite',false);
smoke.setStyle('m.color', 'red');
```

下面代码进一步添加了动画效果，通过循环调用network的render进行不断刷新，每次刷新再重新计算粒子系统内部顶点的位置，即可达到烟雾飘动的动画效果。可以用于显示机房监控烟雾、温度等信息。

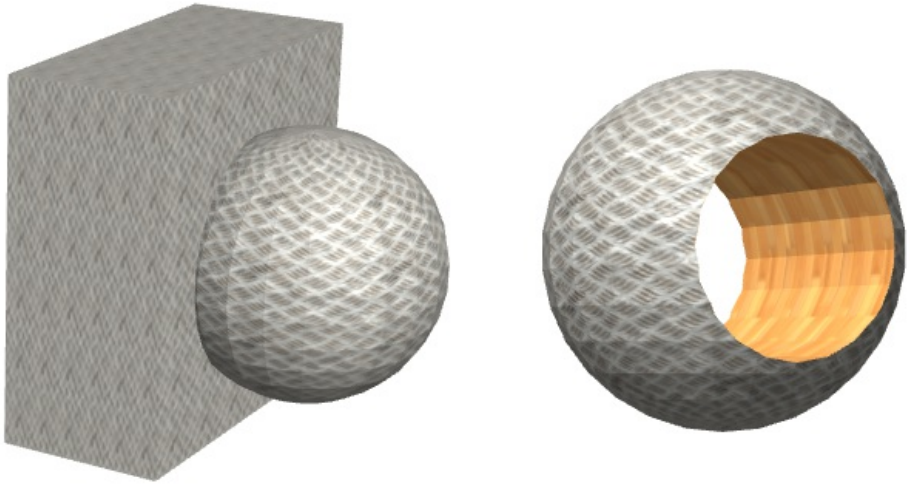
```
var smoke = new mono.Particle();
var network;

function load(){
  var box = new mono.DataBox();
  var camera = new mono.PerspectiveCamera(30, 1.5, 0.1, 10000);
  camera.setPosition(50,200,500);

  network= new mono.Network3D(box, camera, myCanvas);
  var interaction = new mono.DefaultInteraction(network);
  interaction.zoomSpeed = 30;
  network.setInteractions([new mono.SelectionInteraction(network), interaction]);
  mono.Utils.autoAdjustNetworkBounds(network,document.documentElement,'clientWidth','clientHeight');

```

```
15   var pointLight = new mono.PointLight(0xFFFFFF,1);
16   pointLight.setPosition(10000,10000,10000);
17   box.add(pointLight);
18   box.add(new mono.AmbientLight(0x888888));
19
20   var sphereRadius = 80;
21   var particleCount = 200;
22
23   smoke.setClient("sphereRadius", sphereRadius);
24
25   for (var p = 0; p &lt; particleCount; p++) {     var radius = sphereRadius;     var angle = Math.random() * (Math.PI * 2
26       particle.y = 0;
27       var radius = sphereRadius;
28       var angle = Math.random() * (Math.PI * 2);
29       particle.x = Math.cos(angle) * radius;
30     }
31     //continue;
32     var t = Date.now() / 1000 % 3;
33     particle.x += Math.cos(t * particle.velocity.x) * 5;
34     particle.y += Math.sin(t * particle.velocity.y) * 2;
35   }
36   network.render();
37   setTimeout(changeSmoke,20);
38 }
```

效果如下图：

mono.CSG——运算体

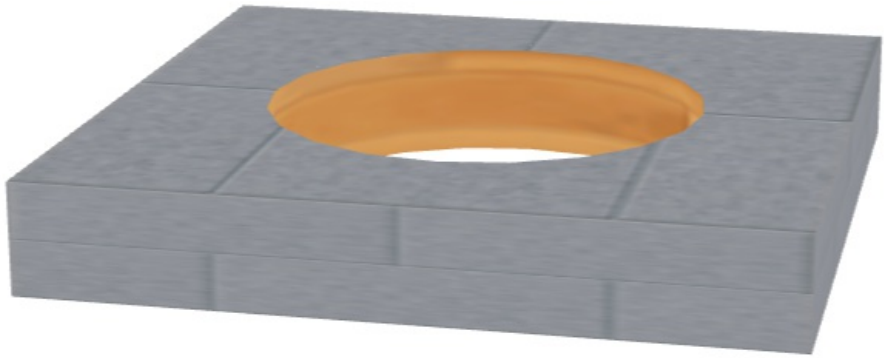
运算体（mono.CSG）是由多个3D物体进行组合运算得出的物体。例如立方体A合并球体B、球体A减掉圆柱体B。目前mono支持的运算有union（合并，A加上B）、substract（减除，A减掉B的部分）、intersect（交叉、A和B的公共部分）。下图显示了两个物体合并和减除的情况：



构造一个运算体物体，需要给定一个具体的3D物体，然后运算体物体再进行相互运算。对于运算的结果，**必须要调用toMesh()函数进行处理**，才能加入DataBox进行显示。

```
1  //创建一个立方体
2  var cube = new mono.Cube(100,15,100);
3  cube.setStyle('m.texture.image', '../images/default_texture.png');
4
5  //创建一个圆柱体
6  var cylinder = new mono.Cylinder(30,30,50);
7  cylinder.setStyle('m.texture.image', '../images/floor.png');
8
9  var csg1=new mono.CSG(cube); //立方体对应的运算体对象
10 var csg2=new mono.CSG(cylinder);  //圆柱体对应的运算体对象
11 var csg=csg1.substract(csg2).toMesh();  //立方体减去圆柱体，生成残留对象，并进行mesh处理，返回运算结果3D对象
12
13 box.add(csg);
```

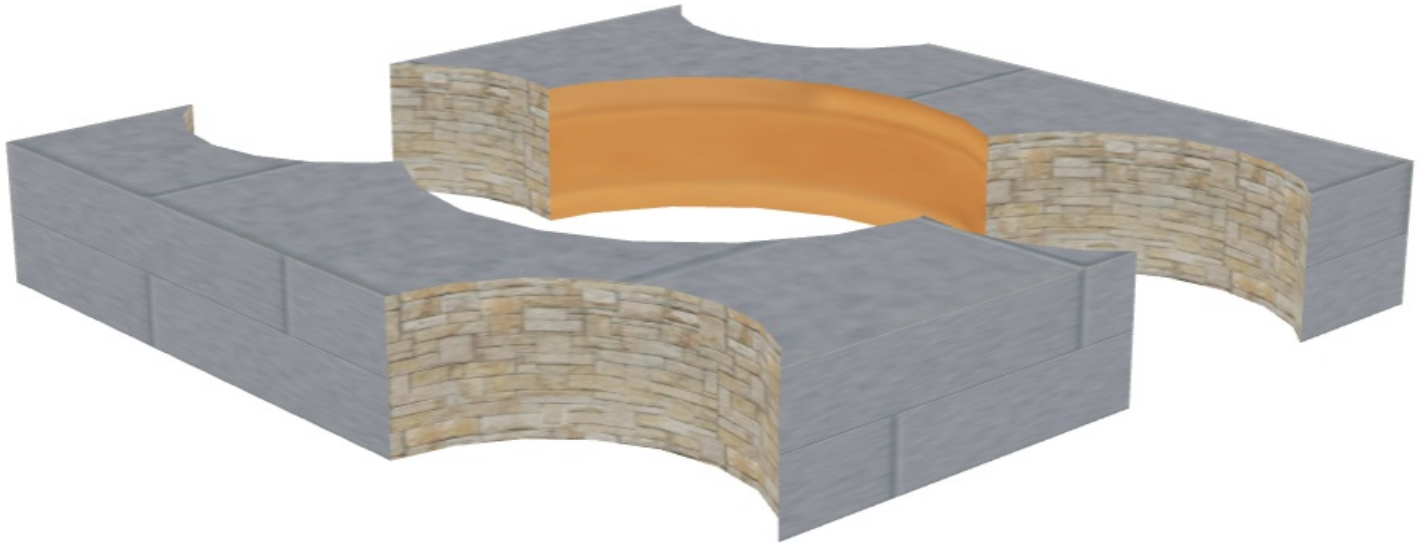
运行上述代码，显示效果：



运算体物体在mesh前，可以持续多次进行运算，最后再进行mesh加入DataBox，以形成更复杂的运算体。例如：

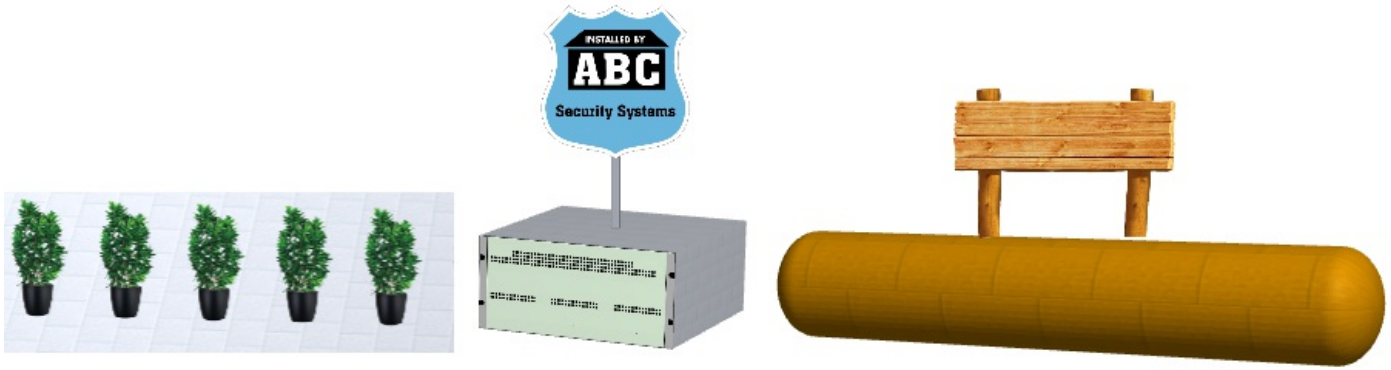
```
1  var cube = new mono.Cube(100,15,100);
2  cube.setStyle('m.texture.image', '../images/default_texture.png');
3
4  var cylinder = new mono.Cylinder(30,30,50);
5  cylinder.setStyle('m.texture.image', '../images/floor.png');
6
7  var csg1=new mono.CSG(cube);
8  var csg2=new mono.CSG(cylinder);
9  var csg=csg1.substract(csg2);
10
11 for(var i=0;i&lt;4;i++){
12   cylinder = new mono.Cylinder(20,20,40);
13   cylinder.setPosition(35-70* (i%2),0, 50-70* (i/2));
14   cylinder.setStyle('m.texture.image', '../images/wall04_3d.png');
15   cylinder.setStyle('m.texture.repeat', new mono.Vec2(4,3));
16   csg=csg.substract(new mono.CSG(cylinder));
17 }
18
19 csg=csg.toMesh();
20 box.add(csg);
```

运行结果如下：



mono.Billboard——公告牌

公告牌对象（mono.Billboard）是一种特殊的3D物体，它只有一个图片组成，而且这张图片会永远朝前面向镜头（也就是我们用户的眼睛），无论场景如何变化。在游戏软件场景中，经常会有这样的场景：一个移动的物体（人、机器等）会在上方显示一个图片，图片上显示了文字、状态信息等，这个图片会永远朝向用户的眼睛，而无论物体如何移动。公告牌的这一特性使得它非常合适制作3D场景中的各种信息展示、文字说明、提示警告等。其典型用法是用于显示设备告警、设备状态信息，也可以显示花草树木等装饰物。下图是一些公告牌的应用场景：



一般公告牌会设置为某设备的自对象，也就是通过billboard.setParent(node)方法为其设置父节点。这样公告牌会跟随父对象一起移动。

```
1  //创建一个设备节点
2  var node = new mono.Cube(20,20,20);
3  node.setStyle('m.texture.image','../images/bbb.png');
4  box.add(node);
5
6  //构造一个公告牌对象
7  var billboard=new mono.Billboard();
8
9  billboard.setStyle('m.texture.image','../images/billboard5.png'); //设置公告牌图片
10 billboard.setScale(20,40,1);  //设置图片的宽高放大比例。第三个z轴参数没有意义，传入1即可
11 billboard.setPosition(0,0,0);  //设置公告牌位置
12
13 billboard.setStyle('m.alignment',mono.BillboardAlignment.bottomCenter); //设置公告牌在附属节点上的对齐位置。附属节点是公
14 billboard.setStyle('m.transparent',true); //设置图片透明模式。此时图片的透明区域会显示为镂空
15
16 billboard.setParent(node); //设置公告牌的父节点（跟随节点）
```

显示效果如下：

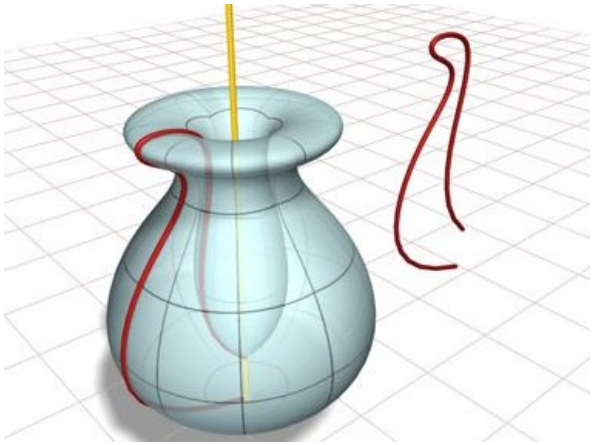
另外我们也可以设置Billboard始终垂直效果：

```
1 billboard.setStyle('m.vertical',true);//设置billboard为垂直效果
```

显示效果如下,左图为未设置垂直效果,右图为始终垂直效果：

mono.LatheNode——车削对象

车削对象（mono.LatheNode）是一个路径体绕某轴线旋转一圈（或一定角度）形成的封闭（或开放）的3D物体。下图中，红色的路径体沿着黄色的轴旋转一周，就形成了一个车削对象。



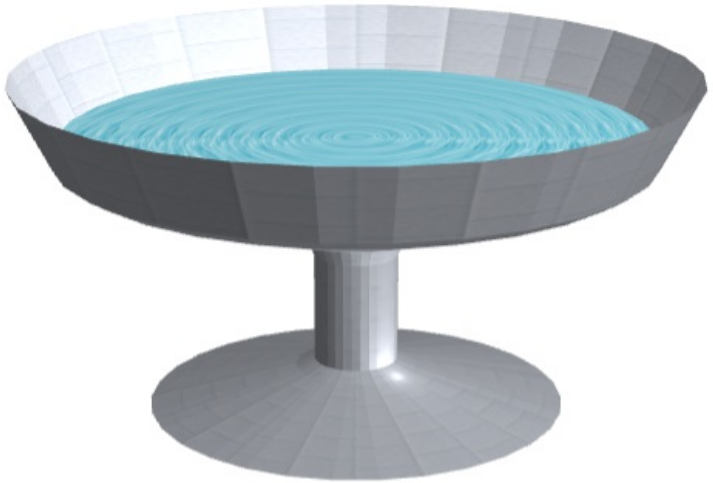
```
//创建车削对象
var node = new mono.LatheNode(path, segmentsH, segmentsR,arc, startClosed, endClosed);
//path - 车削旋转的形状
//segmentsH - 轴向的分片次数。如路径体在轴的方向比较复杂，则需要多分配分片数量，才能得到更平滑的效果。默认64
//segmentsR - 径向的分片次数。车削对象的径向是一个完整的圆（或圆的一部分），要得到光滑的边缘，需要设置较多径向分片。默认20
//arc - 径向旋转角度。如径向旋转角度不足一周（2*Math.PI），则会形成一个残缺的车削对象
//startClosed - 起始面是否封闭。true为封闭，false为镂空
//endClosed - 截止面是否封闭。true为封闭，false为镂空
```

下面代码创建了一个简单的路径体和车削对象：

```
var path = new mono.Path();
//以下z点均会被忽略
path.moveTo(-50, 0, 0);
path.lineTo(-10, 10, 0);
path.lineTo(-10, 40, 0);
path.lineTo(-70, 60, 0);
path.lineTo(-80, 80, 0);

var node = new mono.LatheNode(path, 50, 20, Math.PI*2, true, false);
node.setStyle('m.texture.image','../images/bbb.png').setStyle('m.type','phong').setStyle('m.side',mono.DoubleSide).setSt
box.add(node);
```

效果如下：

如适当设置角度，并进行嵌套，可以很容易做出容器、液体的效果。下图是两个LatheNode组成的物体，内部对象采用水状贴图，模拟了液体效果：



可设置旋转角度，形成残缺旋转体：

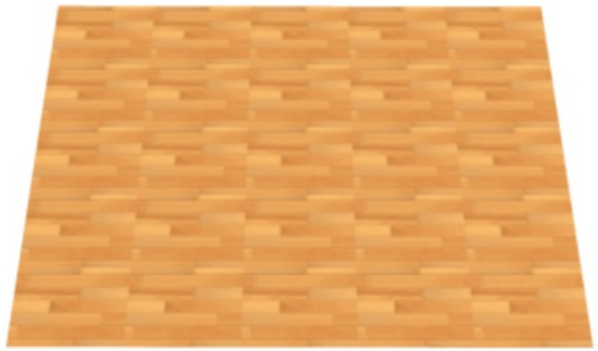
```
1 var node = new mono.LatheNode(path, 50, 20, Math.PI*1.5, true, true);
```

mono.Plane——平面对象

平面（mono.Plane）是一个非常简单的平面对象。它可设置宽、高、贴图，表示一个空间的水平平面或垂直平面。平面对象只能创建一个矩形平面，不能创建任意形状的平面，也没有厚度的概念，使用有一定局限性。如需要创建有厚度、边缘复杂的的平面，可使用mono.ShapeNode。

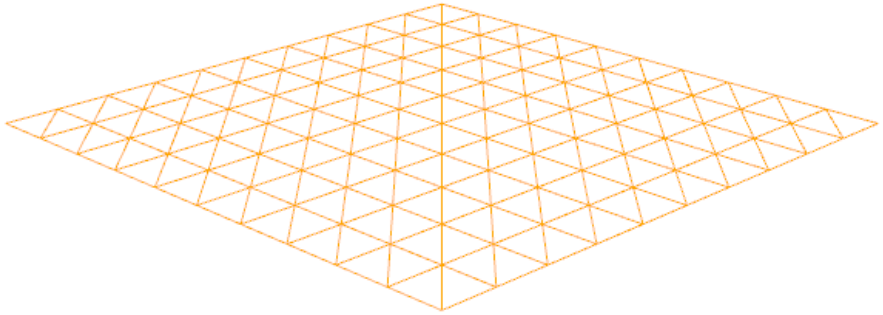
```
1 //创建平面对象
2 var node = new mono.Plane(width, height);
3 //width - 平面宽度
4 //height - 平面高度
5
6 //创建一个简单的平面
7 var node = new mono.Plane(200,200,1,1,true);
8 node.setStyle('m.texture.image','../images/floor.png').setStyle('m.type','phong').setStyle('m.side',mono.DoubleSide).setSty
9 box.add(node);
```

效果如下：



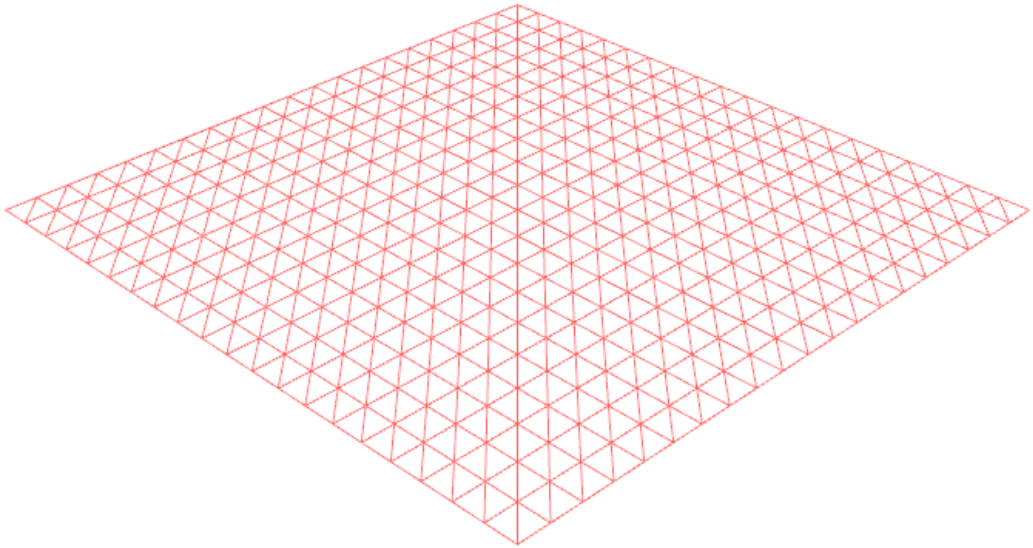
```
//创建wireframe样式的平面对象
 var floor = new mono.Plane(500,500,10,10);
          floor.setStyle('m.wireframe',true).setStyle('m.color','orange').setStyle('m.side', mono.DoubleSide);
          floor.setRotation(Math.PI/2,Math.PI,Math.PI/2);
          floor.setSelectable(false);
          box.add(floor);
```

效果如下：



```
//也可以使用另外一种样式设置方法
 var plane = new mono.Plane(200,200,20,20);
          plane.s({
              'm.color' : '#FF6666',
              'm.wireframe' : true
          });
          plane.setSelectable(false);
          plane.setRotation(Math.PI/2,Math.PI,Math.PI/2);
          box.add(plane);
```

效果如下：



mono.ComboNode —— 组合体

组合体和运算体类似，它和由多个对象根据一定的运算符，运算出来的一个新的物体，它比mono.CSG更加方便，不需要人为去相加、相减，而是通过给定的运算符，mono内部就处理了这几个物体的运算，比如：一个圆形和立方体运算后可以得到下面效果：

代码如下：

```
1 var cube1 = createCube(120,50,50);
```

```
cube1.setPosition(0,0,0);

var cube2 = createCylinder(50);
cube2.setPosition(0,0,0);

var combo = new mono.ComboNode([cube2,cube1],['-']);
box.add(combo);
```

创建ComboNode需要传入两个参数：combos，operators

combos：需要组合的原型对象

operators：组合对象之间的运算符，支持'+'，'-'，'^'（相加、相减、相交）

有了这样的组合体对象，我们就可以组合出各种形状的物体，下面是组合成一个灭火器的样例：

mono.Light——光源对象

mono.Light是所有灯的基类，在3D场景中，光照是非常重要的元素，它模拟了现实世界，在3D场景中的物体可以反射出光照效果。类的定义如下：

```
Light = function(color) {};
```

Light类中包含的方法如下：

- setCastShadow(castShadow):设置是否需要显示灯光的阴影
- setColor(color):设置光照的颜色值
- getColor():获取光照的颜色值
- setAmbient(ambient):设置环境光照的颜色值
- getAmbient(): 获取环境光照的颜色值
- setDiffuse(diffuse):设置散射光的颜色值

- getDiffuse():获取散射光的颜色值
- setSpecular(specular):设置镜面光的颜色值
- getSpecular(): 获取镜面光的颜色值

通过继承Light，主要实现了四种光源，分别为：(环境光源)AmbientLight、(点光源)PointLight、(聚光源)SpotLight、(方向光源)DirectionalLight。类的定义如下：

```
/**
 * 环境光，环境光是那些在环境中进行了充分的散射，无法辨认其方向的光。
 * @param {Number} hex 一个包含RGB的十六进制数组
 */
AmbientLight = function(hex) {};

/**
 * 点光源。在3D场景中创建一个指定颜色的点光源元素
 * @param {Number} hex 一个包含RGB的十六进制数组
 * @param {Number} intensity 光照强度值，如果为空，默认设置为1
 * @param {Number} distance 光照衰减的距离值
 */
PointLight = function ( hex, intensity, distance ) {};

/**
 * 聚光源。在3D场景中创建一个指定颜色的聚光灯元素
 * @param {Number} hex 一个包含RGB的十六进制数组
 * @param {Number} intensity 光照强度值，如果为空，默认设置为1
 * @param {Number} distance 光照衰减的距离值
 * @param {Number} angle 最大可扩散的光照弧度
 * @param {} exponent
 */
mono.SpotLight = function ( hex, intensity, distance, angle, exponent ) {};

/**
 * 方向光源。在3D场景中创建一个指定颜色的方向光照
 * @param {Number} hex 一个包含RGB的十六进制数组
 * @param {Number} intensity 光照强度值，如果为空，默认设置为1
 */
mono.DirectionalLight = function ( hex, intensity ) {};
```

光源中常用的方法如下：

- PointLight.setIntensity(intensity):设置光照的强度
- PointLight.getIntensity():获取光照的强度值
- PointLight.setDistance(distance):设置光照的衰减距离
- PointLight.getDistance():获取光照的衰减距离
- PointLight.setDistance(distance):设置光照衰减的距离值
- PointLight.getDistance():获取光照衰减的距离值
- SpotLight.setAngle(angle):设置最大可扩散的光照弧度
- SpotLight.getAngle():获取最大可扩散的光照弧度
- AmbientLight.clone():将一个点光源对象克隆出新的对象

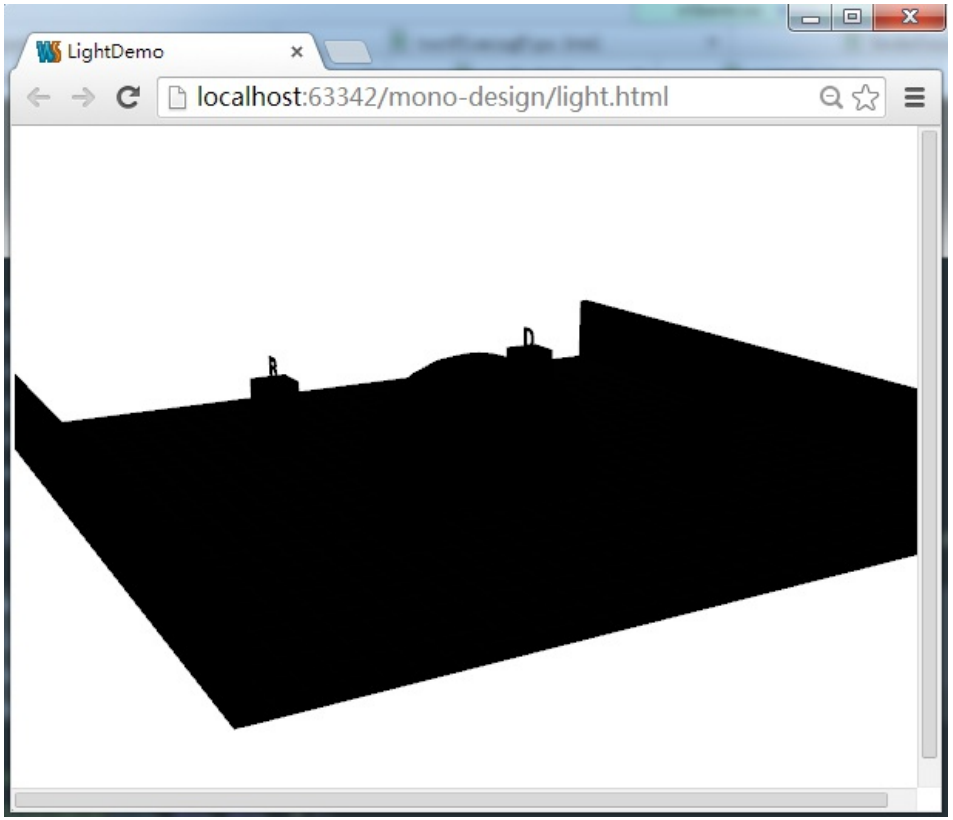
下面代码说明了如何使用光源：

```
var light=new mono.PointLight(0x00ff00,0.5);
    light.setPosition(100,300,600);
    box.add(light);
    box.add(new mono.AmbientLight(0xff00ff));
```

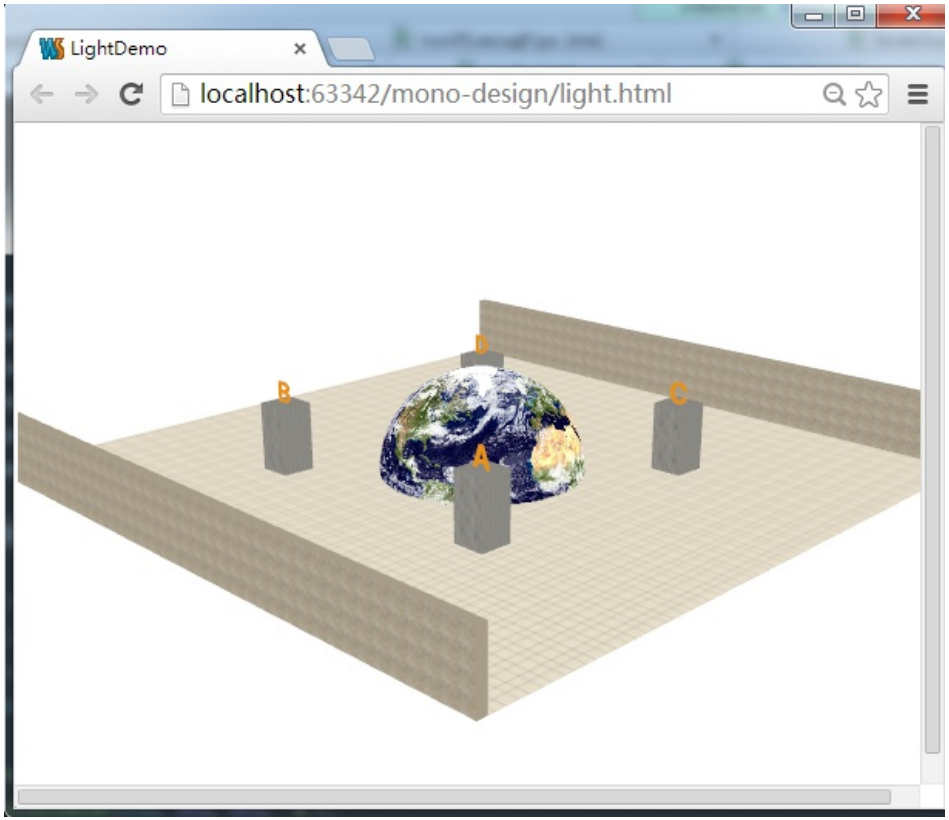
下面例子分别展示了未添加光源、添加AmbientLight 、添加PointLight以及添加SpotLight的效果。

```
1 LightDemo<script src="mono.js" type="mce-text/javascript"></script><script src="twaver.js" type="mce-text/javascrip
2 var network, interaction;        function load(){            var box = new TGL.DataBox();            var camera = new TGL.P
3 // ]]></script>
```

为添加任何灯光效果如下：



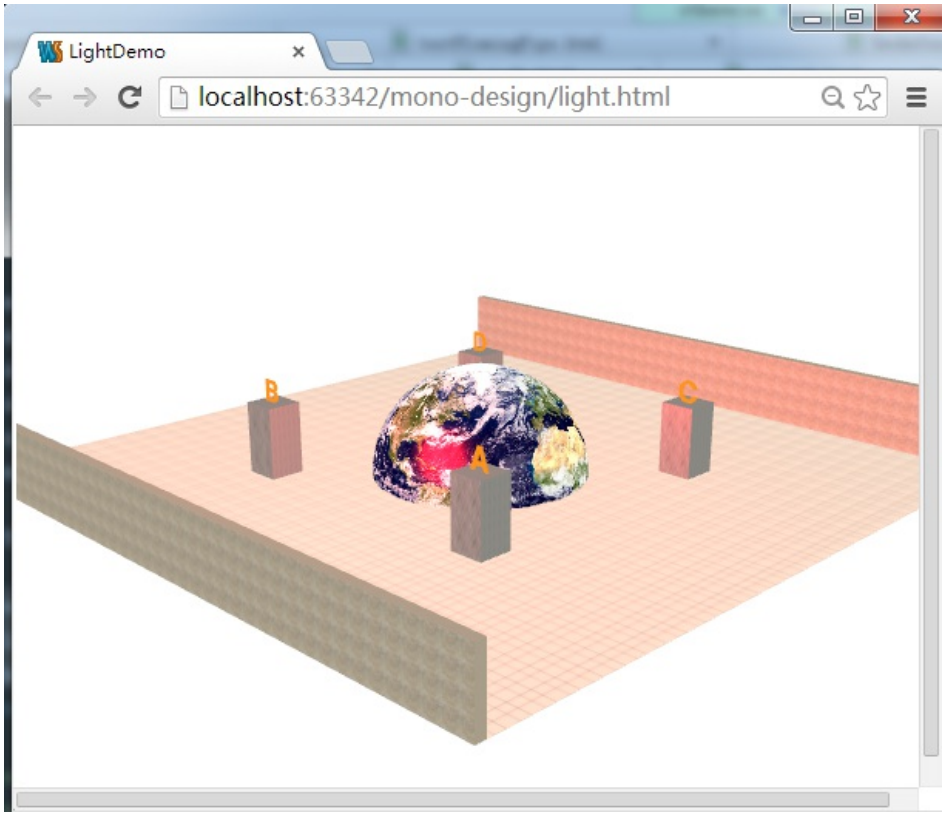
为场景添加AmbientLight效果如下：



为场景添加PointLight

```
1 //添加点光源
2         var light=new TGL.PointLight(0x33ff00,0.5);
3         light.setPosition(100,300,600);
4         box.add(light);
```

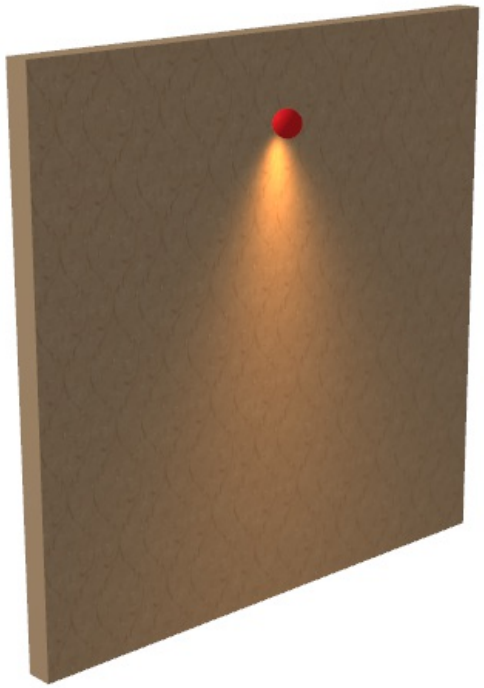
效果如下：



为场景添加SpotLight

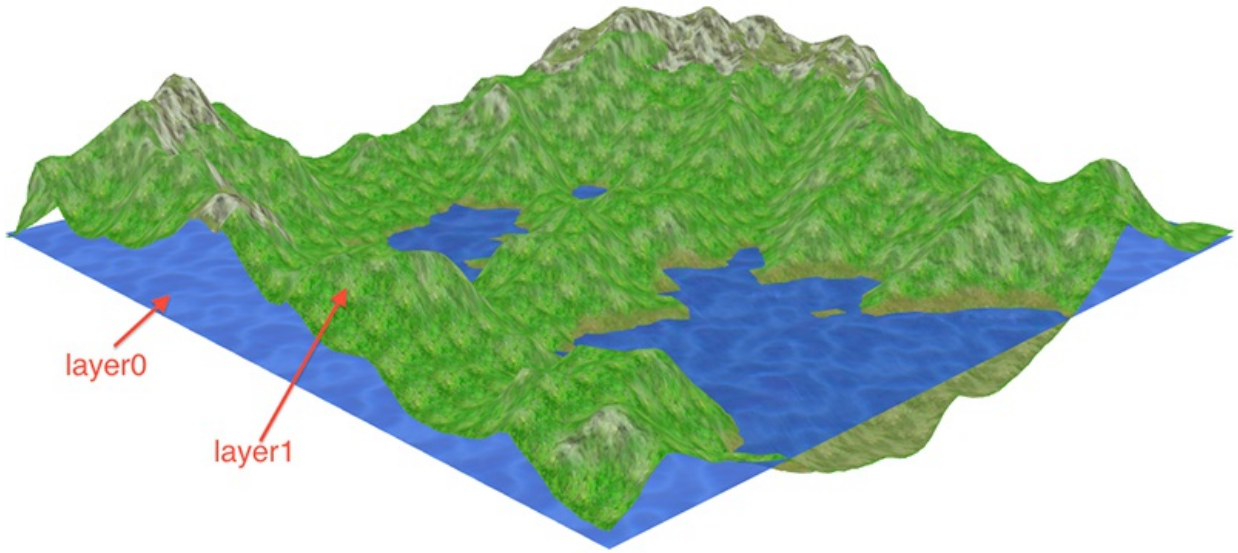
```
1 <script src="../libs/t.js"></script><script>// <![CDATA[
2 var network, interaction;        function load(){            var box = new mono.DataBox();            var camera = new mo
3 // ]]></script>
```

效果如下：



mono.Terrain——地形对象

地形对象（mono.Terrain）有两个面组成，两个面分别为layer0、layer1，例如在海岛的场景中，layer0呈现水面，layer1呈现地形（layer0和layer1不可互换，即layer1表示地形），如下图 T-1



地形对象的每个面都可以单独设置材质素材，纹理重复次数，是否透明，透明度等等

例如：两个面分别设置纹理重复次数

```
terrain.setStyle('layer1.m.texture.repeat', new mono.Vec2(30, 30));
terrain.setStyle('layer0.m.texture.repeat', new mono.Vec2(5, 5));
```

如果设置style时不指定哪个面前缀， 且值不为数组，则表示应用于所有面

例如：设置所有面都不透明

```
terrain.setStyle('m.transparent', false) ;
```

如果设置style时不指定哪个面前缀，且值为数组，则数组元素分别对应一个面

例如：设置layer0和layer1贴图分别为water.jpg和grass.jpg

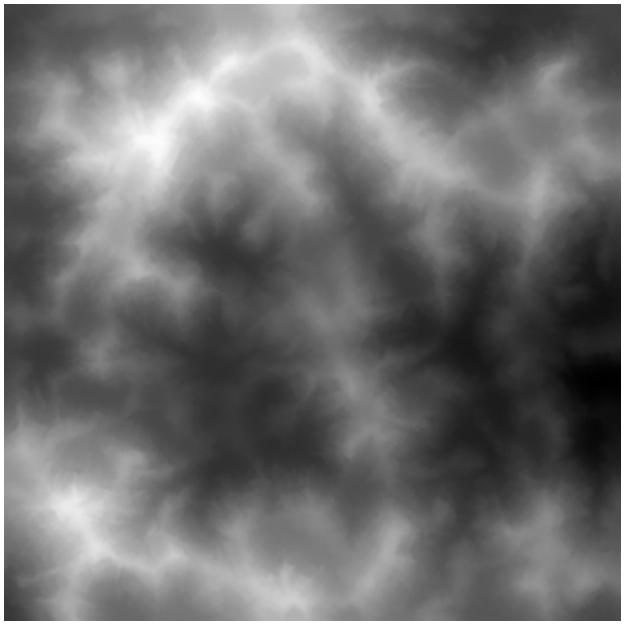
```
terrain.setStyle('m.texture.image', ['./images/water.jpg', './images/grass.jpg']) ;
```

控制面是否显示，例如取消水面的显示

```
terrain.setStyle('layer0.m.visible',false);
```

```
//mono.Terrain构造函数
var terrain = new mono.Terrain(width ,depth, segmentsW , segmentsD , heightUnit, heightMap , baseLayerHeigh);
//width：宽度，X轴方向
//depth：深度，Z轴方向
//segmentsW：横向的切片数量
//segmentsW：横向的切片数量
//heightUnit：layer1上每个点的高度，取决于heightMap图片中对应像素点的颜色（同比例对应）rgb的平均值，所以计算所得值，最大
//heightMap：layer1上每个点的高度，解析于heightMap图片中同比例像素点颜色rgb的平均值，最大值为255。见图-2，图T-1中高度依
//baseLayerHeigh：layer0的高度
```

图T-2



在现实中有不同的地质环境，如草地，森林，高山，黄土高原，湿地，沙滩等等，因此在地形对象中会有使用多张贴图的需求，多张贴图的设置方法，同时地形对象的贴图类型必须为terrain

```
1 terrain.setStyle('layer1.m.type', 'terrain');
2 terrain.setStyle('layer1.m.texture.image', './images/beach.jpg');
3 terrain.setStyle('layer1.m.texture1.image', './images/grass.jpg');
4 terrain.setStyle('layer1.m.texture2.image', './images/rock.jpg');
```

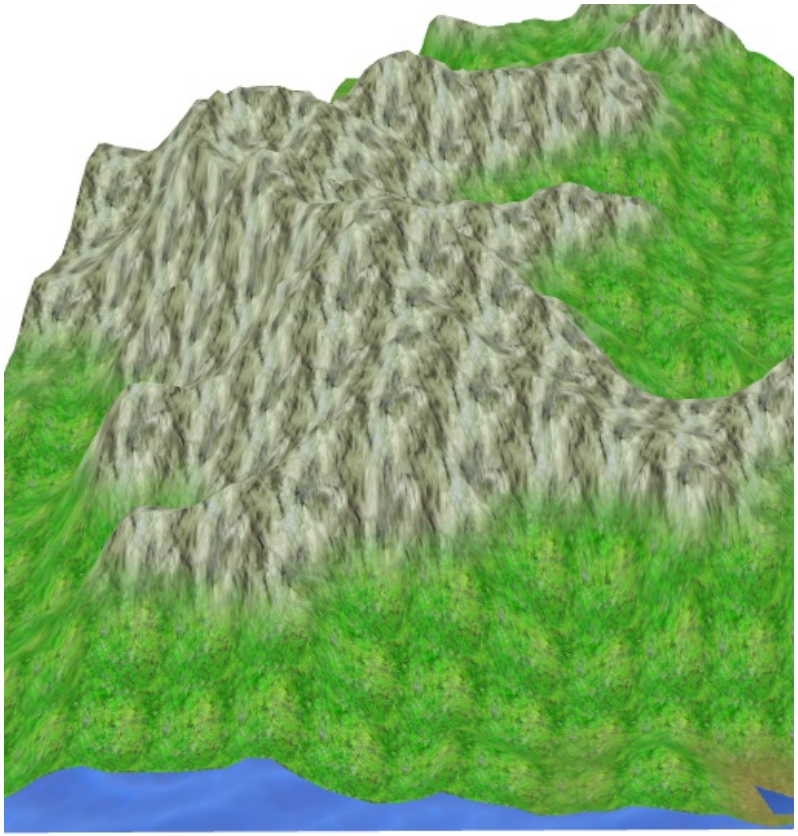
目前最多支持3张贴图，这3张贴图的显示规则一：

设置layer1.m.blendRange属性

```
1 terrain.setStyle('layer1.m.blendRange', new mono.Vec2(0.28,0.5));
```

根据Y值权重所在范围取对应贴图，Y值权重 = Y值/Y值最大值，权重在小于0.28，贴图为beach.jpg；权重大于0.5，贴图为rock.jpg，

显示效果见图-3



显示规则二：

设置layer1.m.textureb.image属性(见图-5)

```
1 terrain.setStyle('layer1.m.textureb.image', './images/terrain_splats.png');
```

列举个数据算式说明

a 代表layer1上像素点的颜色值

b 代表与a同比例在layer1.m.texture.image上的点的颜色值

c 代表与a同比例在layer1.m.texture.image1上的点的颜色值

d 代表与a同比例在layer1.m.texture.image2上的点的颜色值

e 代表与a同比例在layer1.m.textureb.image上的点的颜色值

e.r、e.g、e.b分别代表e的rgb值

a = (b*e.r + c*e.g + d*e.b)/(e.r + e.g + e.b)

效果见图-4

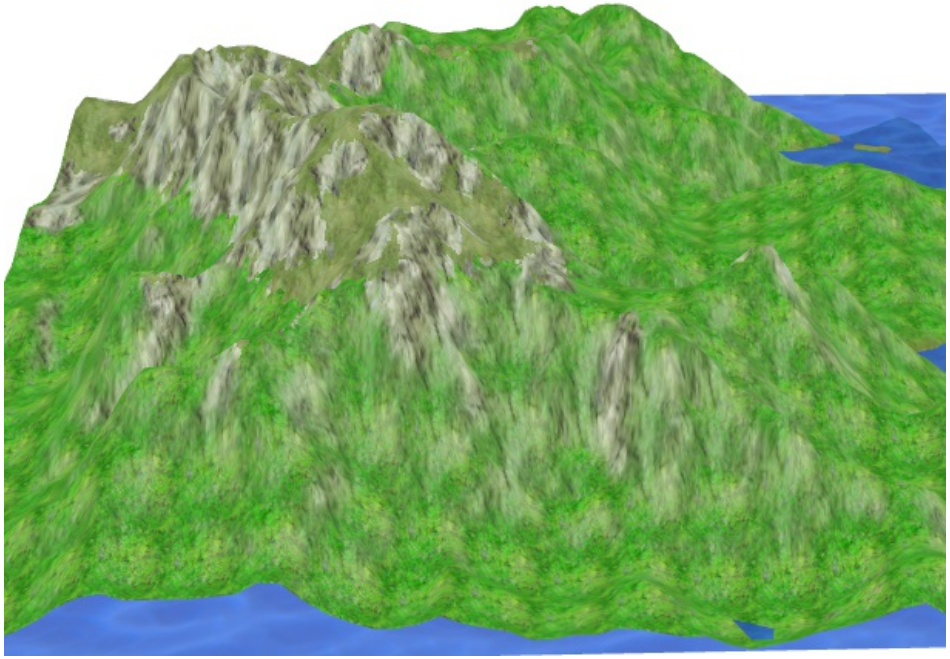
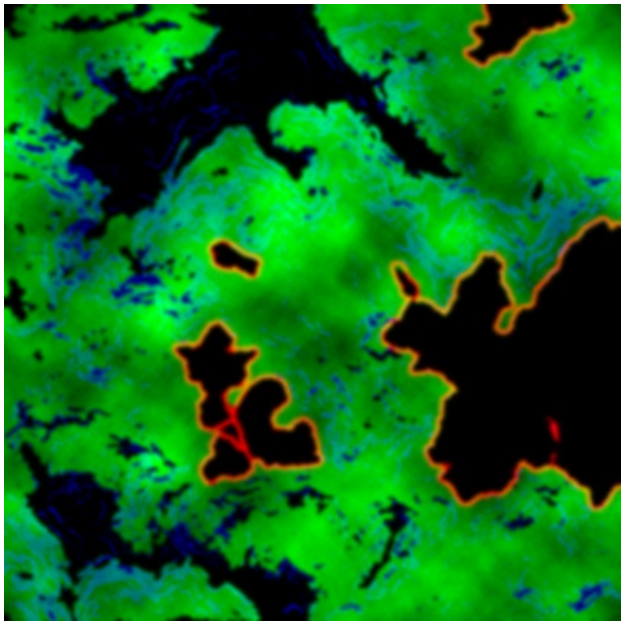


图-5

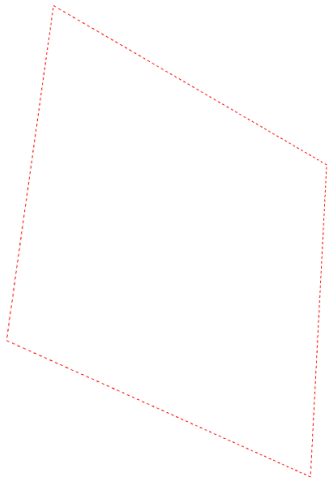


mono.Line——线条对象

线条对象（mono.Line）是即可以充当连接各物体的桥梁，也可以充分展现线条的艺术美。

```
1 //创建Rectangle矩形形状
2 var line = mono.Line.createRectangle(40,40,500);
3 line.setType(TGL.LinePieces);
4 line.setPositionY(-30);
5 line.setPositionX(20);
6 line.setStyle('m.color','red');
7 box.add(line);
```

效果展示如下：



```
1 //创建Helix螺旋状线条
2 var line = mono.Line.createHelix(-10,25,20,10,300);
3 line.setPositionY(30);
4 line.setPositionX(20);
5 line.setStyle('m.color','red');
6 box.add(line);
```

```
1 //创建Ellipse椭圆线条
2  var line = mono.Line.createEllipse(12,8,100,0,Math.PI/2,true);
3  line.setType(TGL.LinePieces);
4  line.setPositionY(30);
5  line.setPositionX(20);
6  line.setStyle('m.color','red');
7  box.add(line);
```