**自定义几何图形和样式**

本教程将向您介绍底层的API提供的几何图形和外观系统。这是一个高级主题，用于扩展cesumjs，具有自定义网格、形状、体积和外观。

**Geometry概述**

CesiumJS可以使用entities创建不同的几何体类型（如多边形和椭球体）。

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

viewer.entities.add({

rectangle : {

coordinates : Cesium.Rectangle.fromDegrees(**-**100.0, 20.0, **-**90.0, 30.0),

material : **new** Cesium.StripeMaterialProperty({

evenColor: Cesium.Color.WHITE,

oddColor: Cesium.Color.BLUE,

repeat: 5

})

}

});



在本教程中，我们将深入了解构成它们的Geometry和Appearance外观类型。Geometry定义了基本的结构，即组成基本图形包括:三角形、直线或点。Appearance定义图形的着色，包括其完整的GLSL顶点和片段着色器，以及渲染状态。

使用geometries and appearances的好处是:

* **性能**- 当绘制大量静态图形时，使用geometry可以直接将它们组合成一个几何体，以减少CPU开销并更好地利用GPU性能。
* **灵活** - Primitive结合了geometry和appearance。通过分离它们，我们可以独立地修改它们。我们可以添加与许多不同外观兼容的新几何图形，反之亦然。.
* **底层访问控制** - 外观提供了接近底层的渲染访问，而不必担心直接使用渲染器的所有细节。Appearance很容易编写GLSL脚本代码并使用用户自定义的渲染状态。

同时也有一些缺陷:

* 直接使用geometries and appearances需要更多代码和对图形更深入的理解。Entities处于适合映射应用程序的抽象级别；geometries and appearances具有更接近传统3D引擎的抽象级别。
* 组合几何图形对静态数据有效，不一定适用于动态数据。.

让我们开始应用primitive，简单绘制个图形:

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

**var** scene **=** viewer.scene;

**var** instance **=** **new** Cesium.GeometryInstance({

geometry : **new** Cesium.RectangleGeometry({

rectangle : Cesium.Rectangle.fromDegrees(**-**100.0, 20.0, **-**90.0, 30.0),

vertexFormat : Cesium.EllipsoidSurfaceAppearance.VERTEX\_FORMAT

})

});

scene.primitives.add(**new** Cesium.Primitive({

geometryInstances : instance,

appearance : **new** Cesium.EllipsoidSurfaceAppearance({

material : Cesium.Material.fromType('Stripe')

})

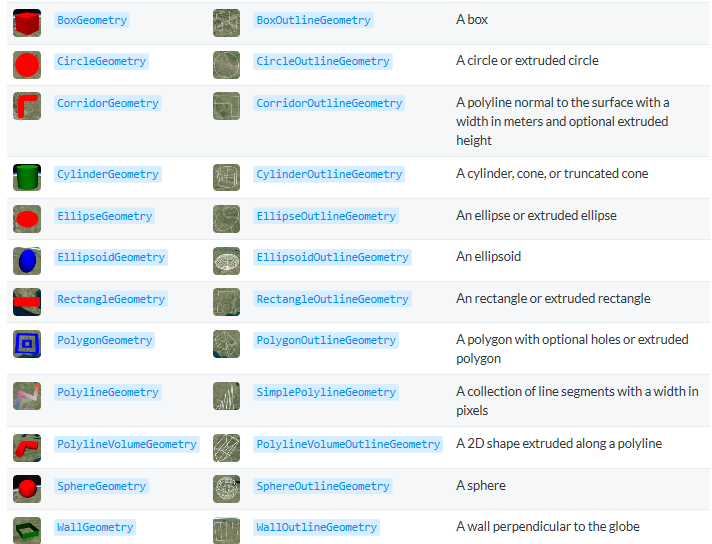
}));

我们没有使用矩形entity，而是使用了通用[Primitive](https://cesium.com/docs/cesiumjs-ref-doc/Primitive.html),，它结合了geometry instance和appearance。现在，除了instance是geometry的容器外，我们不会区分 [Geometry](https://cesium.com/docs/cesiumjs-ref-doc/Geometry.html)和[GeometryInstance](https://cesium.com/docs/cesiumjs-ref-doc/GeometryInstance.html)。

因为它在曲面上，所以我们可以使用 [EllipsoidSurfaceAppearance](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidSurfaceAppearance.html)。这样可以根据几何体位于曲面上或位于椭球体上方的恒定高度进行假设，从而节省内存。

**Geometry类型**

CesiumJS支持多种类型，详细内容可查看：https://cesium.com/docs/tutorials/geometry-and-appearances/#geometry-types



**Geometry组合**

当我们使用一个primitive绘制多个静态几何体时，我们看到了性能上的好处。例如，在一个primitive中绘制两个矩形。

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

**var** scene **=** viewer.scene;

**var** instance **=** **new** Cesium.GeometryInstance({

geometry : **new** Cesium.RectangleGeometry({

rectangle : Cesium.Rectangle.fromDegrees(**-**100.0, 20.0, **-**90.0, 30.0),

vertexFormat : Cesium.EllipsoidSurfaceAppearance.VERTEX\_FORMAT

})

});

**var** anotherInstance **=** **new** Cesium.GeometryInstance({

geometry : **new** Cesium.RectangleGeometry({

rectangle : Cesium.Rectangle.fromDegrees(**-**85.0, 20.0, **-**75.0, 30.0),

vertexFormat : Cesium.EllipsoidSurfaceAppearance.VERTEX\_FORMAT

})

});

scene.primitives.add(**new** Cesium.Primitive({

geometryInstances : [instance, anotherInstance],

appearance : **new** Cesium.EllipsoidSurfaceAppearance({

material : Cesium.Material.fromType('Stripe')

})

}));



我们用一个不同的矩形创建了另一个实例，然后将这两个实例都提供给了primitive。这将绘制两个具有相同外观的实例。

某些外观允许每个实例提供唯一的属性。例如，我们可以使用PerInstanceColorAppearance用不同的颜色对每个实例进行着色。

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

**var** scene **=** viewer.scene;

**var** instance **=** **new** Cesium.GeometryInstance({

geometry : **new** Cesium.RectangleGeometry({

rectangle : Cesium.Rectangle.fromDegrees(**-**100.0, 20.0, **-**90.0, 30.0),

vertexFormat : Cesium.PerInstanceColorAppearance.VERTEX\_FORMAT

}),

attributes : {

color : **new** Cesium.ColorGeometryInstanceAttribute(0.0, 0.0, 1.0, 0.8)

}

});

**var** anotherInstance **=** **new** Cesium.GeometryInstance({

geometry : **new** Cesium.RectangleGeometry({

rectangle : Cesium.Rectangle.fromDegrees(**-**85.0, 20.0, **-**75.0, 30.0),

vertexFormat : Cesium.PerInstanceColorAppearance.VERTEX\_FORMAT

}),

attributes : {

color : **new** Cesium.ColorGeometryInstanceAttribute(1.0, 0.0, 0.0, 0.8)

}

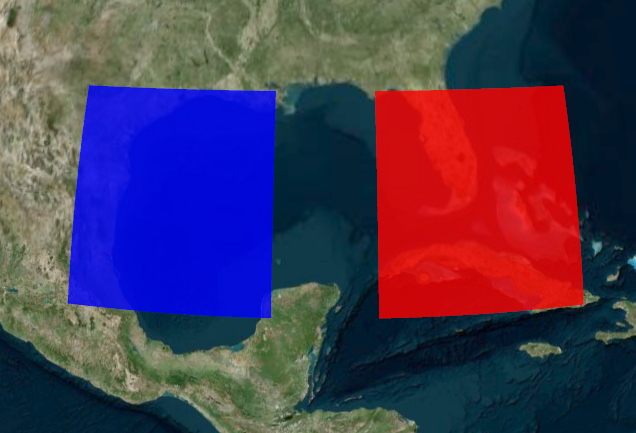
});

scene.primitives.add(**new** Cesium.Primitive({

geometryInstances : [instance, anotherInstance],

appearance : **new** Cesium.PerInstanceColorAppearance()

}));



每个实例都有一个颜色属性。primitive是用PerInstanceColorAppearance构造的，它使用每个实例的color属性来确定着色。

组合几何可以让CesiumJS高效地绘制许多几何图形。下面的示例绘制2592个不同颜色的矩形。

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

**var** scene **=** viewer.scene;

**var** instances **=** [];

**for** (**var** lon **=** **-**180.0; lon **<** 180.0; lon **+=** 5.0) {

**for** (**var** lat **=** **-**85.0; lat **<** 85.0; lat **+=** 5.0) {

instances.push(**new** Cesium.GeometryInstance({

geometry : **new** Cesium.RectangleGeometry({

rectangle : Cesium.Rectangle.fromDegrees(lon, lat, lon **+** 5.0, lat **+** 5.0),

vertexFormat: Cesium.PerInstanceColorAppearance.VERTEX\_FORMAT

}),

attributes : {

color : Cesium.ColorGeometryInstanceAttribute.fromColor(Cesium.Color.fromRandom({alpha : 0.5}))

}

}));

}

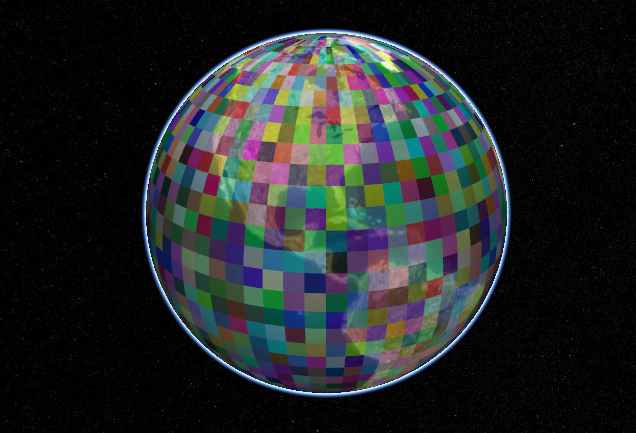
}

scene.primitives.add(**new** Cesium.Primitive({

geometryInstances : instances,

appearance : **new** Cesium.PerInstanceColorAppearance()

}));



**Picking拾取**

实例合并后可以独立访问。为实例指定一个id，并使用它来确定是否选中[Scene.pick](https://cesium.com/docs/cesiumjs-ref-doc/Scene.html#pick).

下面的示例创建一个具有id的实例，并在单击时将消息写入控制台。

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

**var** scene **=** viewer.scene;

**var** instance **=** **new** Cesium.GeometryInstance({

geometry : **new** Cesium.RectangleGeometry({

rectangle : Cesium.Rectangle.fromDegrees(**-**100.0, 30.0, **-**90.0, 40.0),

vertexFormat: Cesium.PerInstanceColorAppearance.VERTEX\_FORMAT

}),

id : 'my rectangle',

attributes : {

color : Cesium.ColorGeometryInstanceAttribute.fromColor(Cesium.Color.RED)

}

});

scene.primitives.add(**new** Cesium.Primitive({

geometryInstances : instance,

appearance : **new** Cesium.PerInstanceColorAppearance()

}));

**var** handler **=** **new** Cesium.ScreenSpaceEventHandler(scene.canvas);

handler.setInputAction(**function** (movement) {

**var** pick **=** scene.pick(movement.position);

**if** (Cesium.defined(pick) **&&** (pick.id **===** 'my rectangle')) {

console.log('Mouse clicked rectangle.');

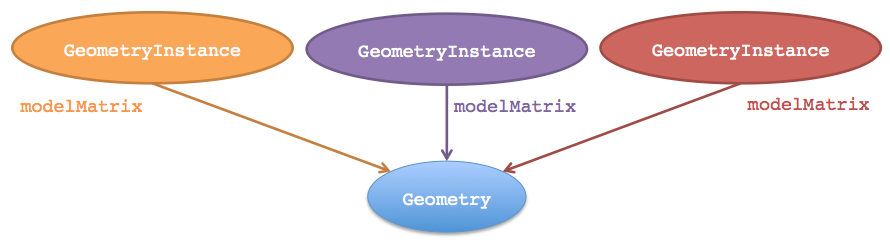
}

}, Cesium.ScreenSpaceEventType.LEFT\_CLICK);

使用id可以避免在构造primitive之后在内存中保留对完整instance（包括geometry）的引用。

**几何实例(Geometry instances)**

Instances可用于在场景的不同位置、缩放和旋转同一geometry。这是可能的，因为多个实例可以引用同一个Geometry，并且每个实例可以具有不同的modelMatrix。这允许我们只计算一次Geometry，并多次重用它。



以下示例将创建一个[EllipsoidGeometry](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidGeometry.html) 和两个instances。每个实例引用相同的椭球体几何体，但使用不同的modelMatrix放置它，导致一个椭球体位于另一个椭球体之上。

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

**var** scene **=** viewer.scene;

**var** ellipsoidGeometry **=** **new** Cesium.EllipsoidGeometry({

vertexFormat : Cesium.PerInstanceColorAppearance.VERTEX\_FORMAT,

radii : **new** Cesium.Cartesian3(300000.0, 200000.0, 150000.0)

});

**var** cyanEllipsoidInstance **=** **new** Cesium.GeometryInstance({

geometry : ellipsoidGeometry,

modelMatrix : Cesium.Matrix4.multiplyByTranslation(

Cesium.Transforms.eastNorthUpToFixedFrame(Cesium.Cartesian3.fromDegrees(**-**100.0, 40.0)),

**new** Cesium.Cartesian3(0.0, 0.0, 150000.0),

**new** Cesium.Matrix4()

),

attributes : {

color : Cesium.ColorGeometryInstanceAttribute.fromColor(Cesium.Color.CYAN)

}

});

**var** orangeEllipsoidInstance **=** **new** Cesium.GeometryInstance({

geometry : ellipsoidGeometry,

modelMatrix : Cesium.Matrix4.multiplyByTranslation(

Cesium.Transforms.eastNorthUpToFixedFrame(Cesium.Cartesian3.fromDegrees(**-**100.0, 40.0)),

**new** Cesium.Cartesian3(0.0, 0.0, 450000.0),

**new** Cesium.Matrix4()

),

attributes : {

color : Cesium.ColorGeometryInstanceAttribute.fromColor(Cesium.Color.ORANGE)

}

});

scene.primitives.add(**new** Cesium.Primitive({

geometryInstances : [cyanEllipsoidInstance, orangeEllipsoidInstance],

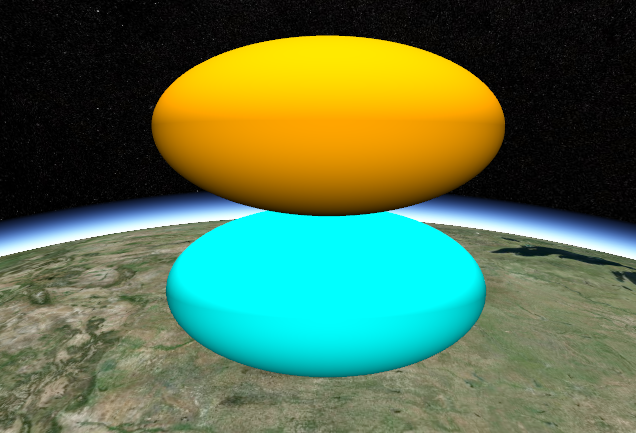
appearance : **new** Cesium.PerInstanceColorAppearance({

translucent : **false**,

closed : **true**

})

}));



**更新属性**

Update the per-instance attributes of the geometries after they are added to the primitive to change the visualization. Per-instance attributes include:

将geometry添加到primitive后，更新geometry的每个实例属性以更改可视化效果。每个实例属性包括：

* Color: [ColorGeometryInstanceAttribute](https://cesium.com/docs/cesiumjs-ref-doc/ColorGeometryInstanceAttribute.html) 决定实例的颜色，primitive必须要有 [PerInstanceColorAppearance](https://cesium.com/docs/cesiumjs-ref-doc/PerInstanceColorAppearance.html).
* Show：确定实例可见性的布尔值。可用于任何实例。

下面的代码改变geometry的颜色:

**var** viewer **=** **new** Cesium.Viewer('cesiumContainer');

**var** scene **=** viewer.scene;

**var** circleInstance **=** **new** Cesium.GeometryInstance({

geometry : **new** Cesium.CircleGeometry({

center : Cesium.Cartesian3.fromDegrees(**-**95.0, 43.0),

radius : 250000.0,

vertexFormat : Cesium.PerInstanceColorAppearance.VERTEX\_FORMAT

}),

attributes : {

color : Cesium.ColorGeometryInstanceAttribute.fromColor(**new** Cesium.Color(1.0, 0.0, 0.0, 0.5))

},

id: 'circle'

});

**var** primitive **=** **new** Cesium.Primitive({

geometryInstances : circleInstance,

appearance : **new** Cesium.PerInstanceColorAppearance({

translucent : **false**,

closed : **true**

})

});

scene.primitives.add(primitive);

setInterval(**function**() {

**var** attributes **=** primitive.getGeometryInstanceAttributes('circle');

attributes.color **=** Cesium.ColorGeometryInstanceAttribute.toValue(Cesium.Color.fromRandom({alpha : 1.0}));

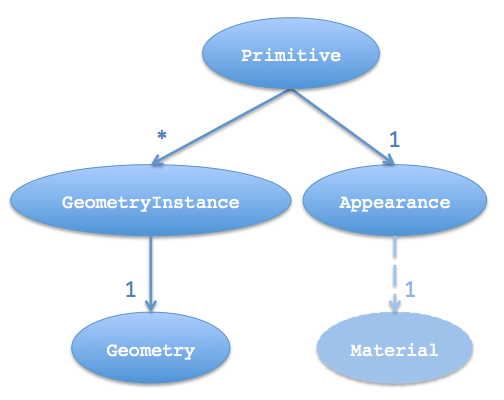
},2000);

Geometry的属性可以用primitive.GetGeometryInstanceAttribute获取. attributes的属性可以直接更改。

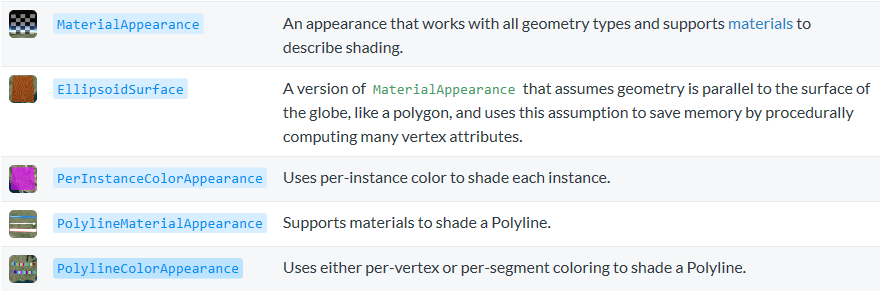
**样式**

Geometry defines structure. The other key property of a primitive, appearance, defines the primitive’s shading, i.e., how individual pixels are colored. A primitive can have many geometry instances, but it can only have one appearance. Depending on the type of appearance, an appearance will have a [material](https://github.com/CesiumGS/cesium/wiki/Fabric) that defines the bulk of the shading.

Geometry定义结构。primitive的另一个关键属性appearance,定义了primitive的着色，即单个像素的着色方式。primitive可以有多个几何体实例，但只能有一个appearance。根据appearance的类型，appearance将具有mateail定义着色的大部分的材质。



CesiumJS 具有以下 appearances类型:



Appearances define the full GLSL vertex and fragment shaders that execute on the GPU when the primitive is drawn. Appearances also define the full render state, which controls the GPU’s state when the primitive is drawn. We can define the render state directly or use higher-level properties like [closed](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#closed) and [translucent](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#translucent), which the appearance will convert into render state. For example:

Appearances定义绘制primitive时在GPU上执行的完整GLSL顶点和片段着色器。Appearances还定义了完整的渲染状态，该状态控制绘制primitive时GPU的状态。我们可以直接定义渲染状态，也可以使用更高级的属性，比如[closed](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#closed)和[translucent](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html" \l "translucent)，Appearances设置为渲染状态。例如：

*// Perhaps for an opaque box that the viewer will not enter.*

*// - Backface culled and depth tested. No blending.*

**var** appearance **=** **new** Cesium.PerInstanceColorAppearance({

translucent : **false**,

closed : **true**

});

*// This appearance is the same as above*

**var** anotherAppearance **=** **new** Cesium.PerInstanceColorAppearance({

renderState : {

depthTest : {

enabled : **true**

},

cull : {

enabled : **true**,

face : Cesium.CullFace.BACK

}

}

});

创建appearance后，无法更改其renderState属性，但可以更改其material。我们还可以更改primitive的外观属性。

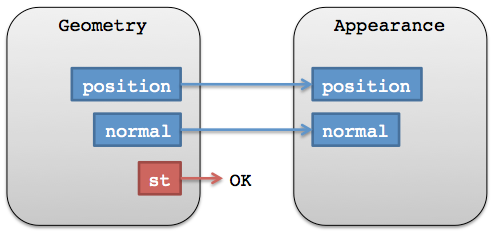
大多数外观还具有“ [flat](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#flat)”和“ [faceForward](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html" \l "faceForward)”特性，这些特性间接控制GLSL着色器。

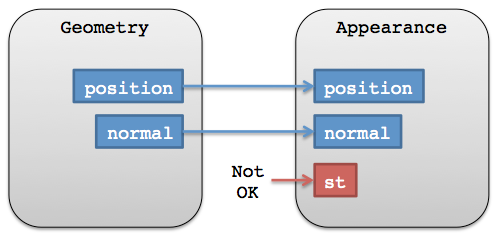
* flat -平面阴影。不要考虑照明。.
* faceForward –开启光照时，翻转法线使其始终面向观众。避免背面出现黑色区域，例如墙的内侧l.

**几何图形和样式的兼容**

并非所有appearance都适用于所有geometry。例如，EllipsoidSurfaceAppearance 不适用于WallGeometry ，因为墙不在球体表面上。

要使appearance与geometry兼容，它们必须具有匹配的顶点格式，这意味着geometry必须具有外观所需的数据作为输入。创建几何体时可以提供VertexFormat





geometry的vertexFormat确定它是否可以与其他几何体组合。两个几何体不必是同一类型，但它们需要匹配的顶点格式。

为了方便起见，appearances要么有一个[vertexFormat](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#vertexFormat) ，要么有一个[VERTEX\_FORMAT](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidSurfaceAppearance.html#VERTEX_FORMAT)静态格式传递给一个geometry

**var** geometry **=** **new** Ceisum.RectangleGeometry({

vertexFormat : Ceisum.EllipsoidSurfaceAppearance.VERTEX\_FORMAT

*// ...*

});

**var** geometry2 **=** **new** Ceisum.RectangleGeometry({

vertexFormat : Ceisum.PerInstanceColorAppearance.VERTEX\_FORMAT

*// ...*

});

**var** appearance **=** **new** Ceisum.MaterialAppearance(*/\* ... \*/*);

**var** geometry3 **=** **new** Ceisum.RectangleGeometry({

vertexFormat : appearance.vertexFormat

*// ...*

});

**学习资源**

查看API文档:

[All geometries](https://cesium.com/docs/cesiumjs-ref-doc/index.html?filter=Geometry)

[All appearances](https://cesium.com/docs/cesiumjs-ref-doc/index.html?filter=Appearance)

[Primitive](https://cesium.com/docs/cesiumjs-ref-doc/Primitive.html)

[GeometryInstance](https://cesium.com/docs/cesiumjs-ref-doc/GeometryInstance.html)

更多关于材质样式的,查看:

[Fabric](https://github.com/CesiumGS/cesium/wiki/Fabric)(https://github.com/CesiumGS/cesium/wiki/Fabric compatibility)

关于未来计划,查看 :

[Geometry and Appearances Roadmap](Geometry%20and%20Appearances%20Roadmap ) (https://github.com/CesiumGS/cesium/issues/766)