**Custom Geometry & Appearances**

This tutorial will teach you about the Geometry & Appearances system available with the Primitive API. This is an advanced topic for extending CesiumJS with custom meshes, shapes, volumes, and appearances and is not geared towards the typical Cesium user.

If you’re interested in learning how to draw various shapes and volumes on the globe, check out the [Creating Entities](https://cesium.com/docs/tutorials/creating-entities) tutorial.

**Geometry overview**

CesiumJS can create different geometry types using entities, such as polygons and ellipsoids. For example, copy and paste the following into the [Hello World Sandcastle Example](https://sandcastle.cesium.com/) to create a rectangle on the globe with a dot pattern:

Copy to clipboard.Data copied clipboard.

**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

})

}

});



In this tutorial, we go under the hood and look at the [Geometry](https://cesium.com/docs/cesiumjs-ref-doc/Geometry.html) and [Appearance](https://cesium.com/docs/cesiumjs-ref-doc/Appearance.html) types that form them. A geometry defines the primitive’s structure, i.e., the triangles, lines, or points that compose the primitive. An appearance defines the primitive’s shading, including its full GLSL vertex and fragment shaders, and render state.

The benefits of using geometries and appearances are:

* **Performance** - When drawing a large number of static primitives (such as polygon for each zip code in the United States), using geometries directly allows us to combine them into a single geometry to reduce CPU overhead and better utilize the GPU. Combining primitives is done on a web worker to keep the UI responsive.
* **Flexibility** - Primitives combine geometry and appearance. By decoupling them, we can modify each independently. We can add new geometries that are compatible with many different appearances and vice-versa.
* **Low-level access** - Appearances provide close-to-the-metal access to rendering without having to worry about all the details of using the [Renderer](https://github.com/CesiumGS/cesium/wiki/Architecture#renderer) directly. Appearances make it easy to:
  + Write full GLSL vertex and fragment shaders.
  + Use custom render state.

There are also some downsides:

* Using geometries and appearances directly requires more code and a deeper understanding of graphics. Entities are at the level of abstraction appropriate for mapping apps; geometries and appearances have a level of abstraction closer to a traditional 3D engine.
* Combining geometries is effective for static data, not necessarily for dynamic data.

Let’s rewrite the initial code example using geometries and appearances:

Copy to clipboard.Data copied clipboard.

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

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

*// original code*

*//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*

*// })*

*// }*

*//});*

**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({

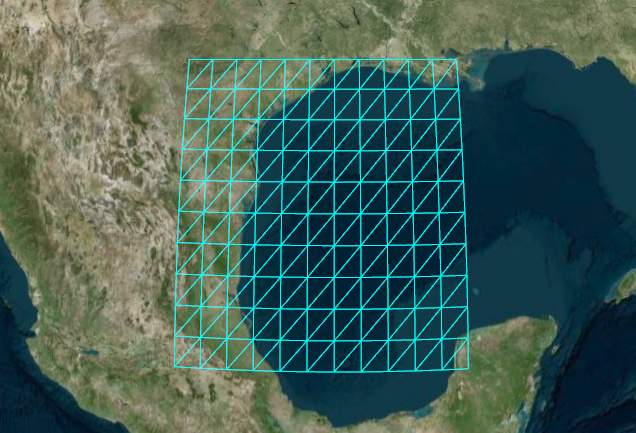
material : Cesium.Material.fromType('Stripe')

})

}));

Instead of using a rectangle entity, we used the generic [Primitive](https://cesium.com/docs/cesiumjs-ref-doc/Primitive.html), which combines the geometry instance and appearance. For now, we will not differentiate between a [Geometry](https://cesium.com/docs/cesiumjs-ref-doc/Geometry.html) and a [GeometryInstance](https://cesium.com/docs/cesiumjs-ref-doc/GeometryInstance.html) other than an instance is a container for a geometry.

To create the geometry for the rectangle, i.e., the triangles covering the rectangular region and that fit the curvature of the globe, we create an [RectangleGeometry](https://cesium.com/docs/cesiumjs-ref-doc/RectangleGeometry.html).

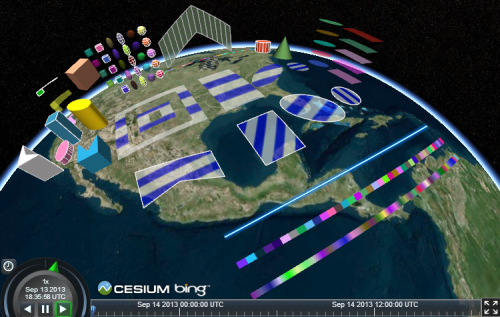


Since it is on the surface, we can use the [EllipsoidSurfaceAppearance](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidSurfaceAppearance.html). This saves memory by making assumptions based on the fact that the geometry is on the surface or at a constant height above the ellipsoid.

**Geometry types**

CesiumJS supports the following geometries:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Box Geometry | [BoxGeometry](https://cesium.com/docs/cesiumjs-ref-doc/BoxGeometry.html) | Box Outline Geometry | [BoxOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/BoxOutlineGeometry.html) | A box |
| Circle Geometry | [CircleGeometry](https://cesium.com/docs/cesiumjs-ref-doc/CircleGeometry.html) | Circle Outline Geometry | [CircleOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/BoxOutlineGeometry.html) | A circle or extruded circle |
| Corridor Geometry | [CorridorGeometry](https://cesium.com/docs/cesiumjs-ref-doc/CorridorGeometry.html) | Corridor Outline Geometry | [CorridorOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/CorridorOutlineGeometry.html) | A polyline normal to the surface with a width in meters and optional extruded height |
| Cylinder Geometry | [CylinderGeometry](https://cesium.com/docs/cesiumjs-ref-doc/CylinderGeometry.html) | Cylinder Outline Geometry | [CylinderOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/CylinderOutlineGeometry.html) | A cylinder, cone, or truncated cone |
| Ellipse Geometry | [EllipseGeometry](https://cesium.com/docs/cesiumjs-ref-doc/EllipseGeometry.html) | Ellipse Outline Geometry | [EllipseOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/EllipseOutlineGeometry.html) | An ellipse or extruded ellipse |
| Ellipsoid Geometry | [EllipsoidGeometry](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidGeometry.html) | Ellipsoid Outline Geometry | [EllipsoidOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidOutlineGeometry.html) | An ellipsoid |
| Extent Geometry | [RectangleGeometry](https://cesium.com/docs/cesiumjs-ref-doc/RectangleGeometry.html) | Extent Outline Geometry | [RectangleOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/RectangleOutlineGeometry.html) | An rectangle or extruded rectangle |
| Polygon Geometry | [PolygonGeometry](https://cesium.com/docs/cesiumjs-ref-doc/PolygonGeometry.html) | Polygon Outline Geometry | [PolygonOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/PolygonOutlineGeometry.html) | A polygon with optional holes or extruded polygon |
| Polyline Geometry | [PolylineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/PolylineGeometry.html) | Polyline Outline Geometry | [SimplePolylineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/SimplePolylineGeometry.html) | A collection of line segments with a width in pixels |
| Volume Geometry | [PolylineVolumeGeometry](https://cesium.com/docs/cesiumjs-ref-doc/PolylineVolumeGeometry.html) | Volume Outline Geometry | [PolylineVolumeOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/PolylineVolumeOutlineGeometry.html) | A 2D shape extruded along a polyline |
| Sphere Geometry | [SphereGeometry](https://cesium.com/docs/cesiumjs-ref-doc/SphereGeometry.html) | Sphere outline Geometry | [SphereOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/SphereOutlineGeometry.html) | A sphere |
| Wall Geometry | [WallGeometry](https://cesium.com/docs/cesiumjs-ref-doc/WallGeometry.html) | Wall Outline Geometry | [WallOutlineGeometry](https://cesium.com/docs/cesiumjs-ref-doc/WallOutlineGeometry.html) | A wall perpendicular to the globe |

[*Live Sandcastle demo*](https://sandcastle.cesium.com/?src=Geometry%20and%20Appearances.html)*.*

**Combining geometries**

We see a performance benefit when we use one primitive to draw multiple static geometries. For example, draw two rectangles in one primitive.

Copy to clipboard.Data copied clipboard.

**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')

})

}));



We created another instance with a different rectangle, and then provided both instances to the primitive. This draws both both instances with the same appearance.

Some appearances allow each instance to provide unique attributes. For example, we can use [PerInstanceColorAppearance](https://cesium.com/docs/cesiumjs-ref-doc/PerInstanceColorAppearance.html) to shade each instance with a different color.

Copy to clipboard.Data copied clipboard.

**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)

}

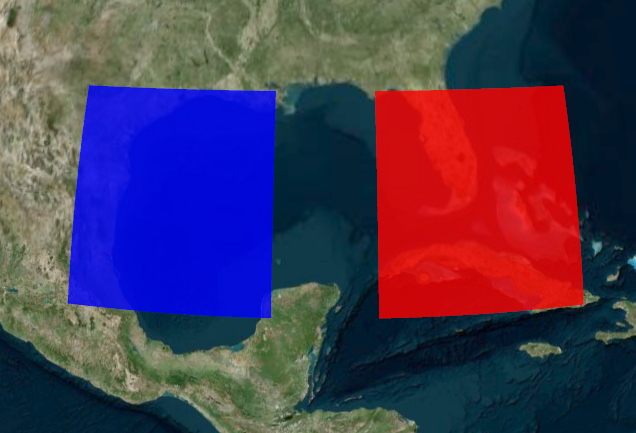
});

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

geometryInstances : [instance, anotherInstance],

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

}));



Each instance has a [Color](https://cesium.com/docs/cesiumjs-ref-doc/Color.html) attribute. The primitive is constructed with a PerInstanceColorAppearance, which uses each instance’s color attribute to determine shading.

Combining geometries allows CesiumJS to efficiently draw A LOT of geometries. The following example draws 2,592 uniquely colored rectangles.

Copy to clipboard.Data copied clipboard.

**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}))

}

}));

}

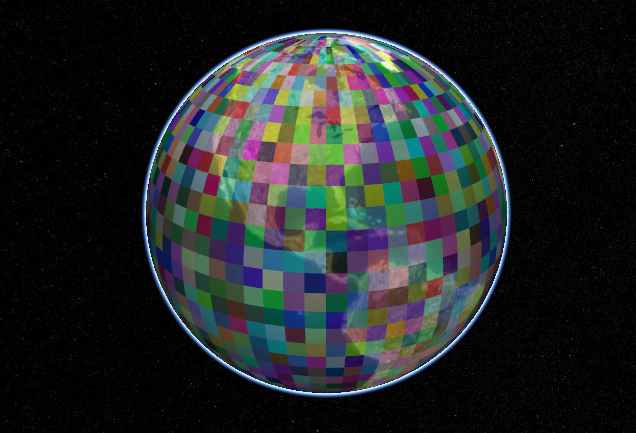
}

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

geometryInstances : instances,

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

}));



**Picking**

Instances are independently accessible after they are combined. Assign an id to an instance and use it to determine if the instance is picked with [Scene.pick](https://cesium.com/docs/cesiumjs-ref-doc/Scene.html#pick).

The following example creates an instance with a id, and writes a message to the console when it is clicked.

Copy to clipboard.Data copied clipboard.

**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.');

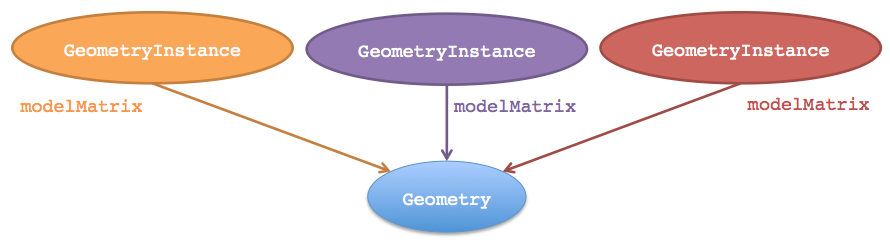
}

}, Cesium.ScreenSpaceEventType.LEFT\_CLICK);

Using id avoids keeping a reference to the full instance, including the geometry, in memory after the primitive is constructed.

**Geometry instances**

Instances can be used to position, scale, and rotate the same geometry in different parts of the scene. This is possible because multiple instances can reference the same Geometry, and each instance can have a different modelMatrix. This allows us to only compute the geometry once and reuse it many times.



The following example creates one [EllipsoidGeometry](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidGeometry.html) and two instances. Each instance references the same ellipsoid geometry, but places it using a different modelMatrix resulting in one ellipsoid being on top of the other.

Copy to clipboard.Data copied clipboard.

**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],

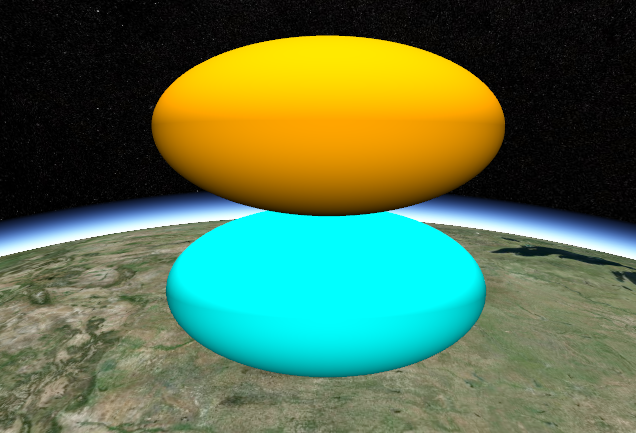
appearance : **new** Cesium.PerInstanceColorAppearance({

translucent : **false**,

closed : **true**

})

}));



**Updating per-instance attributes**

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

* Color : A [ColorGeometryInstanceAttribute](https://cesium.com/docs/cesiumjs-ref-doc/ColorGeometryInstanceAttribute.html) determining the color of the instance. The primitive must have a [PerInstanceColorAppearance](https://cesium.com/docs/cesiumjs-ref-doc/PerInstanceColorAppearance.html).
* Show : A boolean determining the visibility of the instance. Available for any instance.

This example shows how to change the color of the geometry instance:

Copy to clipboard.Data copied clipboard.

**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');

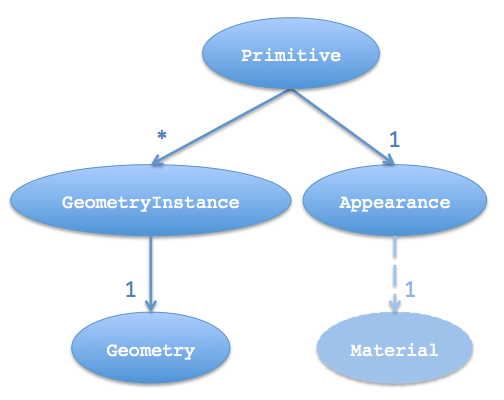
attributes.color **=** Cesium.ColorGeometryInstanceAttribute.toValue(Cesium.Color.fromRandom({alpha : 1.0}));

},2000);

The attributes of the geometry instance can be retrieved from the primitive using primitive.getGeometryInstanceAttributes. The properties of the attributes can be changed directly.

**Appearances**

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.



CesiumJS has the following appearances:

|  |  |  |
| --- | --- | --- |
| MaterialAppearance | [MaterialAppearance](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html) | An appearance that works with all geometry types and supports [materials](https://github.com/CesiumGS/cesium/wiki/Fabric) to describe shading. |
| EllipsoidSurface | [EllipsoidSurface](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidSurfaceAppearance.html) | A version of MaterialAppearance that assumes geometry is parallel to the surface of the globe, like a polygon, and uses this assumption to save memory by procedurally computing many vertex attributes. |
| PerInstanceColorAppearance | [PerInstanceColorAppearance](https://cesium.com/docs/cesiumjs-ref-doc/PerInstanceColorAppearance.html) | Uses per-instance color to shade each instance. |
| PolylineMaterialAppearance | [PolylineMaterialAppearance](https://cesium.com/docs/cesiumjs-ref-doc/PolylineMaterialAppearance.html) | Supports materials to shade a Polyline. |
| PolylineColorAppearance | [PolylineColorAppearance](https://cesium.com/docs/cesiumjs-ref-doc/PolylineColorAppearance.html) | Uses either per-vertex or per-segment coloring to shade a Polyline. |

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:

Copy to clipboard.Data copied clipboard.

*// 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

}

}

});

We can’t change its renderState property once an appearance is created, but we can change its material. We can also change a primitive’s appearance property.

Most appearances also have [flat](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#flat) and [faceForward](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#faceForward) properties, which indirectly control the GLSL shaders.

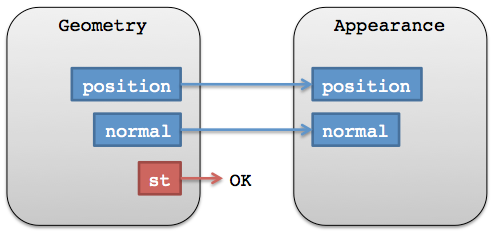
* flat - Flat shading. Do not take lighting into account.
* faceForward - When lighting, flip the normal so it is always facing the viewer. The avoids black areas on back-faces, e.g., the inside of a wall.

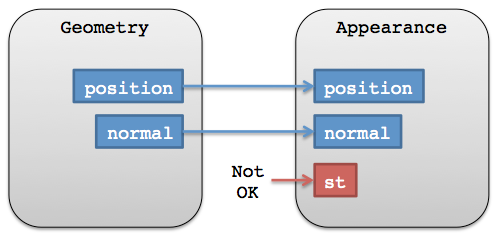
| **flat : true** | **faceForward : false** | **faceForward : true** |
| --- | --- | --- |
| flat | not facing forward | facing forward |

**Geometry and appearance compatibility**

Not all appearances work with all geometries. For example, EllipsoidSurfaceAppearance is not appropriate for WallGeometry since a wall is not on the surface of the globe.

For an appearance to be compatible with a geometry, they must have matching vertex formats, which means the geometry must have the data that the appearance expects as input. A [VertexFormat](https://cesium.com/docs/cesiumjs-ref-doc/VertexFormat.html) can be provided when creating a geometry.





A geometry’s vertexFormat determines if it can be combined with another geometry. Two geometries do not have to be the same type, but they need matching vertex formats.

For convenience, appearances either have a [vertexFormat](https://cesium.com/docs/cesiumjs-ref-doc/MaterialAppearance.html#vertexFormat) property or a [VERTEX\_FORMAT](https://cesium.com/docs/cesiumjs-ref-doc/EllipsoidSurfaceAppearance.html#VERTEX_FORMAT) static constant that can be passed in as an option to the geometry.

Copy to clipboard.Data copied clipboard.

**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

*// ...*

});

**Resources**

In the reference documentation, see:

* [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)

For more on materials, see [Fabric](https://github.com/CesiumGS/cesium/wiki/Fabric).

For future plans, see the [Geometry and Appearances Roadmap](https://github.com/CesiumGS/cesium/issues/766)