**相机**

CesiumJS中的摄影机控制场景的视图。操纵摄影机的方法有很多种，例如旋转、缩放、平移和飞到目的地。CesiumJS有鼠标和触摸事件处理程序来与摄像机交互，还有一个API来以编程方式操作摄像机。了解如何使用cameraapi和自定义相机控件。

**默认相机控制**

| **Mouse Action**  **鼠标状态** | **3D** | **2D** | **Columbus View** |
| --- | --- | --- | --- |
| Left click + drag | Rotate around the globe | Translate over the map | Translate over the map |
| Right click + drag | Zoom in and out | Zoom in and out | Zoom in and out |
| Middle wheel scrolling | Zoom in and out | Zoom in and out | Zoom in and out |
| Middle click + drag | Tilt the globe | No action | Tilt the map |

Set the camera position and orientation programatically with the [setView](https://cesium.com/docs/cesiumjs-ref-doc/Camera.html" \l "setView) function. destination can be a [Cartesian3](https://cesium.com/docs/cesiumjs-ref-doc/Cartesian3.html) or [Rectangle](https://cesium.com/docs/cesiumjs-ref-doc/Rectangle.html), and orientation can be heading/pitch/roll or direction/up. The heading, pitch, and roll angles are defined in radians. Heading is the rotation from the local north direction where a positive angle is increasing eastward. Pitch is the rotation from the local east-north plane. Positive pitch angles are above the plane. Negative pitch angles are below the plane. Roll is the first rotation applied about the local east axis.

使用setView功能函数以编程方式设置相机位置和方向。目的地可以是[Cartesian3](https://cesium.com/docs/cesiumjs-ref-doc/Cartesian3.html)或[Rectangle](https://cesium.com/docs/cesiumjs-ref-doc/Rectangle.html)，方向可以是heading/pitch/roll或direction/up。

航向、俯仰和旋转角以弧度定义。航向是指从正北向东增加的旋转角度。俯仰是基于向北、向东两个向量确定的平面，正俯仰角高于平面，负俯仰角低于平面。旋转是绕局部东轴的旋转角度。

camera.setView({

destination : **new** Cesium.Cartesian3(x, y, z),

orientation: {

heading : headingAngle,

pitch : pitchAngle,

roll : rollAngle

}

});

viewer.camera.setView({

destination : Cesium.Rectangle.fromDegrees(west, south, east, north),

orientation: {

heading : headingAngle,

pitch : pitchAngle,

roll : rollAngle

}

});

有参数都是可选的。如果未指定，参数值将默认为当前相机位置和方向。

使用以下方法设置摄像机位置：直视地球，航向朝北：

camera.setView({

destination : Cesium.Cartesian3.fromDegrees(longitude, latitude, height),

orientation: {

heading : 0.0,

pitch : **-**Cesium.Math.PI\_OVER\_TWO,

roll : 0.0

}

});

**自定义相机鼠标和键盘事件**

让我们创建我们自己的事件处理程序，使相机朝着鼠标的方向看，并在按键时向前、向后、向左、向右、向上和向下移动。首先禁用默认事件处理程序。添加以下代码 (在var viewer = ...后面):

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

**var** canvas **=** viewer.canvas;

canvas.setAttribute('tabindex', '0'); *// needed to put focus on the canvas*

canvas.onclick **=** **function**() {

canvas.focus();

};

**var** ellipsoid **=** viewer.scene.globe.ellipsoid;

*// disable the default event handlers*

scene.screenSpaceCameraController.enableRotate **=** **false**;

scene.screenSpaceCameraController.enableTranslate **=** **false**;

scene.screenSpaceCameraController.enableZoom **=** **false**;

scene.screenSpaceCameraController.enableTilt **=** **false**;

scene.screenSpaceCameraController.enableLook **=** **false**;

创建变量以记录当前鼠标位置，并创建标记以跟踪摄影机的移动方式：:

**var** startMousePosition;

**var** mousePosition;

**var** flags **=** {

looking : **false**,

moveForward : **false**,

moveBackward : **false**,

moveUp : **false**,

moveDown : **false**,

moveLeft : **false**,

moveRight : **false**

};

添加事件处理程序以在单击鼠标左键时设置标志并记录当前鼠标位置:

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

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

flags.looking **=** **true**;

mousePosition **=** startMousePosition **=** Cesium.Cartesian3.clone(movement.position);

}, Cesium.ScreenSpaceEventType.LEFT\_DOWN);

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

mousePosition **=** movement.endPosition;

}, Cesium.ScreenSpaceEventType.MOUSE\_MOVE);

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

flags.looking **=** **false**;

}, Cesium.ScreenSpaceEventType.LEFT\_UP);

创建键盘事件处理程序，以切换摄影机移动的标志。我们将为以下键和行为设置标志:

* w will move the camera forward（前进）.
* s will move the camera backward（后退）.
* a will move the camera to the left（左移）.
* d will move the camera to the right（右移）.
* q will move the camera up（向上）.
* e will move the camera down（向下）.

**function** getFlagForKeyCode(keyCode) {

**switch** (keyCode) {

**case** 'W'.charCodeAt(0):

**return** 'moveForward';

**case** 'S'.charCodeAt(0):

**return** 'moveBackward';

**case** 'Q'.charCodeAt(0):

**return** 'moveUp';

**case** 'E'.charCodeAt(0):

**return** 'moveDown';

**case** 'D'.charCodeAt(0):

**return** 'moveRight';

**case** 'A'.charCodeAt(0):

**return** 'moveLeft';

default:

**return** **undefined**;

}

}

document.addEventListener('keydown', **function**(e) {

**var** flagName **=** getFlagForKeyCode(e.keyCode);

**if** (**typeof** flagName **!==** 'undefined') {

flags[flagName] **=** **true**;

}

}, **false**);

document.addEventListener('keyup', **function**(e) {

**var** flagName **=** getFlagForKeyCode(e.keyCode);

**if** (**typeof** flagName **!==** 'undefined') {

flags[flagName] **=** **false**;

}

}, **false**);

现在我们要在指示事件发生的标志为真时更新摄影机。我们可以向包含以下代码的时钟上的onTick事件添加一个侦听器:

viewer.clock.onTick.addEventListener(**function**(clock) {

**var** camera **=** viewer.camera;

});

下一步，使相机朝着鼠标光标的方向看。将此代码添加到变量声明下的事件侦听器函数中

**if** (flags.looking) {

**var** width **=** canvas.clientWidth;

**var** height **=** canvas.clientHeight;

*// Coordinate (0.0, 0.0) will be where the mouse was clicked.*

**var** x **=** (mousePosition.x **-** startMousePosition.x) **/** width;

**var** y **=** **-**(mousePosition.y **-** startMousePosition.y) **/** height;

**var** lookFactor **=** 0.05;

camera.lookRight(x **\*** lookFactor);

camera.lookUp(y **\*** lookFactor);

}

lookRight和lookUp方法采用一个弧度参数，即旋转角度。我们将鼠标坐标变换为范围（-1.0，1.0），坐标（0.0，0.0）位于画布中心。鼠标离中心的距离决定了转弯的速度。靠近中心的位置移动摄影机的速度较慢，而远离中心的位置移动摄影机的速度更快。

最后，添加代码来移动相机的位置。也要将其添加到事件侦听器函数中:

*// Change movement speed based on the distance of the camera to the surface of the ellipsoid.*

**var** cameraHeight **=** ellipsoid.cartesianToCartographic(camera.position).height;

**var** moveRate **=** cameraHeight **/** 100.0;

**if** (flags.moveForward) {

camera.moveForward(moveRate);

}

**if** (flags.moveBackward) {

camera.moveBackward(moveRate);

}

**if** (flags.moveUp) {

camera.moveUp(moveRate);

}

**if** (flags.moveDown) {

camera.moveDown(moveRate);

}

**if** (flags.moveLeft) {

camera.moveLeft(moveRate);

}

**if** (flags.moveRight) {

camera.moveRight(moveRate);

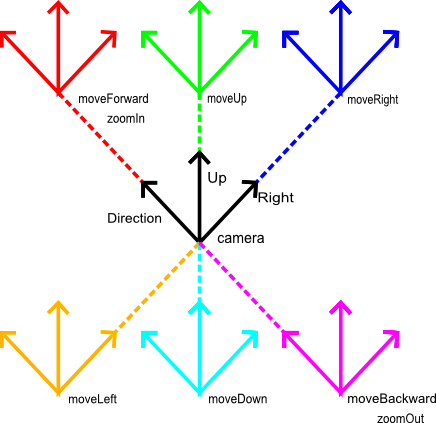
}

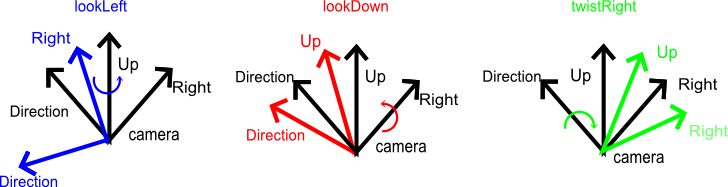
[moveForward](https://cesium.com/docs/cesiumjs-ref-doc/Camera.html?#moveForward), [moveBackward](https://cesium.com/docs/cesiumjs-ref-doc/Camera.html?#moveBackward)，moveUp、moveDown、moveLeft和moveRight方法采用一个以米为单位的参数来移动摄影机。相机在每次按键时移动的距离随着相机到椭球体表面的距离的变化而变化。相机离地面越近，每次按键移动速度越慢。

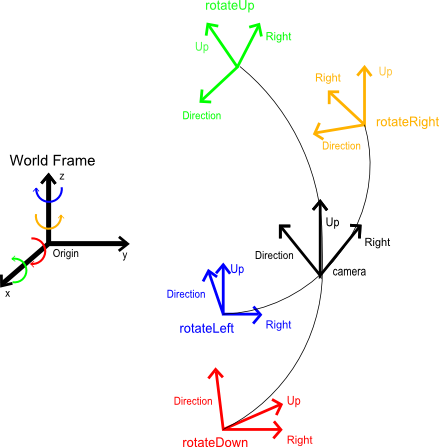
**相机**

[Camera](https://cesium.com/docs/cesiumjs-ref-doc/Camera.html) 代表摄影机当前位置、方向、参照帧和视域截头体的状态。以上的摄影机矢量在每个帧中都是正交的。

1、move\*和zoom\*函数沿摄影机的方向或给定的向量平移摄影机的位置。方向保持不变。



2、look\*和twist\*函数围绕方向、向上或向右向量旋转方向。位置保持不变。

3、rotate\*函数围绕给定向量旋转位置和方向。

函数设置给定范围或位置和目标的相机位置和方向。例如：

**var** west **=** Cesium.Math.toRadians(**-**77.0);

**var** south **=** Cesium.Math.toRadians(38.0);

**var** east **=** Cesium.Math.toRadians(**-**72.0);

**var** north **=** Cesium.Math.toRadians(42.0);

**var** extent **=** **new** Cesium.Extent(west, south, east, north);

camera.viewExtent(extent, Cesium.Ellipsoid.WGS84);

函数创建从相机位置到像素的射线。这对于拾取非常有用，例如：

*// find intersection of the pixel picked and an ellipsoid*

**var** ray **=** camera.getPickRay(mousePosition);

**var** intersection **=** Cesium.IntersectionTests.rayEllipsoid(ray, Cesium.Ellipsoid.WGS84);

**屏幕空间相机控制器**

[ScreenSpaceCameraController](https://cesium.com/docs/cesiumjs-ref-doc/ScreenSpaceCameraController.html) 将用户输入（如鼠标和触摸）从窗口坐标转换为相机运动。它包含用于启用和禁用不同类型输入、修改惯性量以及最小和最大缩放距离的属性。

**学习资源**

学习Sandcastle中关于相机的示例:

[Camera Tutorial](https://sandcastle.cesium.com/?src=Camera%20Tutorial.html).

[Camera](https://sandcastle.cesium.com/?src=Camera.html)

查看API文档

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

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