Cesium Workshop

- http://52.4.31.236/tutorials/Cesium-Workshop/
- https://cesiumjs.org/tutorials/Cesium-Workshop/

Overview

Welcome to the Cesium community!

We're happy to have you.

In order to get you developing your own web map application as soon as possible, this tutorial will walk you through the development of a simple Cesium application from beginning to end.

This tutorial will touch on many of the most important aspects of the Cesium API, but is no means comprehensive (Cesium has a lot of features!).

Our goal is to introduce the fundamentals and the tools you'll need to explore the rest of Cesium.

We'll create a simple application for visualizing sample geocache locations in New York City.

We'll load and style multiple types of 2D and 3D data and create several camera and display options that a user can set interactively.

Finally, as high-tech geocachers, we'll load a 3D drone model to scout the geocache locations for us to take full advantage of our 3D visualization.

By the end of this tutorial, you will have a working overview of Cesium's features and know how to configure a Cesium viewer, load datasets, create and style geometry, work with 3D Tiles, control the camera, and add mouse interactivity to your application.

Setup

Just a few setup steps before we can get to development.

1. Make sure your system is Cesium compatible by visiting Hello World
2. Install cesium-workshop. Either clone or download the zip and extract the contents.
3. In your console, navigate to the root `cesium-workshop` directory.
4. Run `npm install`
5. Run `npm start`

The console should tell you “Cesium development server running locally. Connect to http://localhost:8080”. Don't close the console! We'll need to keep this process running.

Next, navigate to http://localhost:8080 in your browser. You should see our workshop application up and running. Stuck? The Getting Started Tutorial goes more in depth about Cesium setup.

The Application Directory

Now for a tour of our application directory!

Note that this application directory was designed to be as simple as possible, and totally ignores the many varied modern JS frameworks in use today.

But once you have a handle on things, fee free to experiment!

- Source : Our application code and data.
- ThirdParty : External libraries, in this case just Cesium
- LICENSE.md : Therms of use for this application
- server.js : The server we'll run our application from.

Now take a look at `index.html`.

This creates a `div` for our Cesium widget and a few basic input element.

Observe that Cesium Widget is just an ordinary `div` that can be styled and interacted with like any other `div`.

There are a few crucial lines to set this up:

First we include `Cesium.js` in a script tag in the HTML head. This defines the **Cesium** object, which contains the entire Cesium library.

…

Development Resources

For this tutorial and throughout the rest of your Cesium development career, we encourage you to rely on the following resources:

- Reference Documentation : A complete guide to the Cesium API containing many code snippets.
- Sandcastle : A live-coding environment with a large gallery of code examples.
- Tutorials : Detailed introductions to areas of Cesium development.
- Cesium Forum : A resource for asking and answering Cesium-related questions.

Any time you get stuck, odds are one of these resources will have the answers you’re looking for.

The Workflow

Really stuck? You can follow along in sandcastle with a simplified version of the app (no UI):

- The complete code
- The commented code

Now let’s get started!

Creating the Viewer

The basis of any Cesium application is the Viewer, an interactive 3D globe with lots of functionality right out of the box.

Add the viewer to the 'cesiumContainer' `div` by uncommenting the first line.

snippet.javascript

```
var viewer = new Ceisum.Viewer('cesiumContainer');
```

There's a lot included in that one line! You should see a basic globe like this:

By default, the scene handles mouse and touch input.

Try exploring the globe using the default camera controls:

- Left click and drag - Pans the camera over the surface of the globe.
- Right click and drag - Zooms the camera in and out.
- Middle wheel scrolling - Also zooms the camera in and out.
- Middle click and drag - Rotates the camera around the point on the surface of the globe.

In addition to the globe itself, the Viewer comes with some helpful widgets by default, labeled in the above image.

1. **Geocoder** : A location search tool that flies the camera to queried location. Uses Bing Maps data by default.
2. **Home Button** : Flies the viewer back to a default view.
3. **Scene Mode Picker** : Switches between 3D, 2D and Columbus View (CV) modes.
4. **Base Layer Picker** : Chooses the imagery and terrain to display on the globe.
5. **Navigation Help Button** : Displays the default camera controls.
6. **Animation** : Controls the play speed for view animation.
7. **Timeline** : Indicates current time and allows users to jump to a specific time using the scrubber.
8. **Credits Display** : Displays data attributions. Almost always required!
9. **Fullscreen Button** : Makes the Viewer fullscreen.

snippet.javascript

```
var viewer = new Cesium.Viewer('cesiumContainer', {
    scene3DOnly: true,
    selectionIndicator: false,
    baseLayerPicker: false
});
```

This will create a viewer without selection indicators, base layer picker or scene mode picker widgets, since these will be unnecessary for our app. For the full set of Viewer options, see the Viewer

documentation.

Adding Imagery

The next key element of our Cesium application is imagery. This is the set of images that tile over our virtual globe at various resolutions.

To provide optimal performance, Cesium only requests and renders imagery tiles that are visible in the current view and that are at a resolution (also known as zoom level) appropriate to the camera's distance from the globe's surface and the globe's maximumScreenSpaceError.

For a more visual understanding of how imagery works, check out the Cesium Inspector.

Cesium provides lots of tools for working with imagery layers, such as color adjustment and layer blending, Some code examples:

- Adding basic imagery * Adjusting imagery colors * Manipulating and ordering imagery layers * Splitting imagery layers

Cesium provides support for imagery from many different providers out of the box.

Supported Imagery Formats:

- WMS
- TMS
- WMTS (with time dynamic imagery)
- ArcGIS
- Bing Maps
- Google Earth
- Mapbox
- Open Street Map servers
- Single tile

By default, Cesium uses Bing Maps for imagery. Be careful, different data providers have different attribution requirements - make sure you have permission to use data from a particular provider, crediting them in the credits container if applicable.

The imagery packaged with the Viewer is mostly for demo purposes.

In order to use the Bing imagery set in our application, we need to acquire a Bing key of our own.

Set the Bing key with a line like this (at the top of our application, before the viewer is created):

snippet.javascript

```
Cesium.BingMapsApi.defaultKey =
'AsarFiDvISunWhi137V7l5Bu80baB73npU98oTyjqKOb7NbrkiuBPZfDxgXTrGtQ'; //
For use in this application only. Do not reuse!
```

Again, different imagery providers will have different requirements for usage. Now that we have

permission to use this imagery set, we can actually add the imagery layer. First, we create an ImageryProvider, passing in a data url and a few configuration options, then we add the `ImageryProvider` to viewer.imageryLayers.

snippet.javascript

```
// Add Bing imagery
viewer.imageryLayers.addImageryProvider(new
Cesium.BingMapsImageryProvider({
    url : 'https://dev.virtualearth.net',
    mapStyle: Cesium.BingMapsStyle.AERIAL // Can also use
Cesium.BingMapsStyle.ROADS
}));
```

With the above code additions, our application should look like this when you zoom in:

This is actually the same as the default imagery styling, but feel free to play with the `BingMapsStyle` to see the differences.

For more information on Imagery, see our Imagery Layers Tutorial.

Adding Terrain

Cesium supports streaming and visualizing global high-resolution terrain and water effects for oceans, lakes, and rivers.

Mountain peaks, valleys, and other terrain features really show the benefit of a 3D globe compared to a 2D map.

Like imagery, the Cesium engine will stream terrain data from a server, only requesting and rendering tiles as needed based on the current camera position.

Here are some demos of terrain datasets and configuration options:

- ArcticDEM : High resolution arctic terrain
- PATerrain : High resolution Pennsylvania terrain
- Terrain display options : Some terrain configuration options and formats
- Terrain exaggeration: Making terrain elevation differences more dramatic

Supported Terrain Formats :

- Our own quantized-mesh format
- Heightmap
- Google Earth Enterprise

In order to add terrain data, we create a CesiumTerrainProvider, specifying a data url and a few configuration options, then adding the provider as viewer.terrainProvider.

snippet.javascript

```
// Load STK World Terrain
viewer.terrainProvider = new Cesium.CesiumTerrainProvider({
    url : 'https://assets.agi.com/stk-terrain/world',
    requestWaterMask : true, // required for water effects
    requestVertexNormals : true // required for terrain lighting
});
```

`requestWaterMask` and `requestVertexNormals` are configuration options which tell Cesium to request extra data for water and lighting effects. By default these are set to false.

Finally, now that we have have terrain, we need just one more line to make sure objects behind the terrain are correctly occluded. Only the front-most objects will be visible.

snippet.javascript

```
// Enable depth testing so things behind the terrain disappear.
viewer.scene.globe.depthTestAgainstTerrain = true;
```

We now have terrain and animated water. New York is pretty flat, so feel free to explore in order to see the new terrain in action.

For a particularly obvious example, you can navigate to a more rugged area like the Grand Canyon or San Francisco.

For more information on terrain, see the Terrain Tutorial.

Configuring the Scene

Now just a little more setup to start our viewer in the right location and time.

This involves interacting with cesium-workshop : a 3D Cartesian point – when used as a position it is relative to the center of the globe in meters using the Earth fixed-frame (ECR) - Cartographic : a position defined by latitude/longitude (in radians) and height off the globe's surface - Heading Pitch Roll : A rotation (in radians) about the local axes in the East-North-Up frame. Heading is the rotation about the negative z axis. Pitch is the rotation about the negative y axis. Roll is the rotation about the positive x axis. - Quaternion : A 3D rotation represented as 4D coordinates.

…

Camera Control

This Camera and controls what is currently visible.

We can control the camera by setting its position and orientation directly, or by using the Cesium camera API, which is designed to flexibly specify camera position and orientation over time.

Some of the most commonly used methods are :

- Camera.setView(options) : set camera at specific position and orientation immediately
- Camera.zoomIn(amount) : move camera forward along the view vector
- Camera.zoomOut(amount) : move camera backwards along the view vector
- Camera.flyTo(options) : animates movement from current position to a new position
- Camera.lookAt(target, offset) : orient and position the camera to look at target point with given offset
- Camera.move(direction, amount) : move the camera in any direction
- Camera.rotate(axis, angle) : rotate the camera about any axis

To get an idea of what the API can do, check out these camera demos :

- Camera API Demo
- Custom Camera Controls Demo

Now let's try one of these methods by moving the camera to New York.

Set the initial view with **camera.setView()**,

using a Cartesian3 and a HeadingPitchRoll for position and orientation :

snippet.javascript

```
// Create an initial camera view
var initialPosition = new
Cesium.Cartesian3.fromDegrees(-73.998114468289017509,
40.674512895646692812, 2631.082799425431);
var initialOrientation = new
Cesium.HeadingPitchRoll.fromDegrees(7.1077496389876024807,
-31.987223091598949054, 0.025883251314954971306);
var homeCameraView = {
    destination : initialPosition,
    orientation : {
        heading : initialOrientation.heading,
        pitch : initialOrientation.pitch,
        roll : initialOrientation.roll
    }
};
// Set the initial view
viewer.scene.camera.setView(homeCameraView);
```

The camera is now positioned and oriented to look down at Manhattan, and our view parameters are saved in a object that we can pass to other camera methods.

In fact, we can use this same view to update the effect of pressing the home button. Rather than having it return us to the default view of the globe from a distance, we can override the button to bring us to that initial view of Manhattan.

We can adjust the animation by adding a few more options, then add an event listener that cancels the default flight, and calls `flyTo()` our new home view :

snippet.javascript

```
// Add some camera flight animation options
homeCameraView.duration = 2.0;
homeCameraView.maximumHeight = 2000;
homeCameraView.pitchAdjustHeight = 2000;
homeCameraView.endTransform = Cesium.Matrix4.IDENTITY;
// Override the default home button
viewer.homeButton.viewModel.command.beforeExecute.addEventListener(func
tion (e) {
    e.cancel = true;
    viewer.scene.camera.flyTo(homeCameraView);
});
```

For more on basic camera controls, check out our Camera Tutorial.

Clock Control

Loading and Styling Entities

Now that we've set the stage for our application with viewer configuration, imagery and terrain, we can go ahead and add the main focus of our application, sample geocache data.

For easy visualization, Cesium supports popular vector formats GeoJson and KML, as wellas our own vector format, CZML.

Regardless of the initial format, all spatial data in Cesium are represented using the Entity API, a geometry library that provides flexible visualization in a format that is efficient for Cesium to render.

A Cesium Entity is a data object that can be paired with a styled graphical representation and positioned in space and time.

The sandcastle gallery provieds many examples of simple entities.

To get up to speed on the basics of the Entity API, take a break from this applicatoin and read Visualizing Spatial Datafirst.

관련 문서

Plugin Backlinks: 아무 것도 없습니다.