



Browser based 3D Geovisualization

Kanishk Chaturvedi, Zhihang Yao, Thomas H. Kolbe

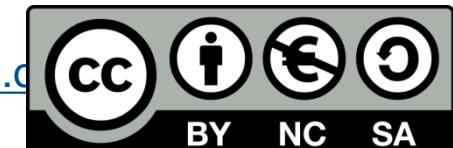
kanishk.chaturvedi@tum.de

Chair of Geoinformatics
Technische Universität München

October 1st, 2016

Document Licensing

This presentation is released under the Creative Commons License CC BY-NC-SA 3.0 as defined here: <https://creativecommons.org/licenses/by-nc-sa/3.0/legalcode>



According to CC BY-NC-SA 3.0 permission is granted to share this document, i.e. copy and redistribute the material in any medium or format, and to adapt it, i.e. remix, transform, and build upon the material under the following conditions:

1. **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
2. **NonCommercial** — You may not use the material for commercial purposes.
3. **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
4. **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

A clarification concerning point 2: non-commercial usage means that this tutorial is not allowed to be used in a commercial training course. It is, however, allowed to use this tutorial for learning about the presented topics within commercial companies or projects.

Outline

- ▶ Session 1: Overview of browser based 3D visualization
 - 3D Graphics and Standards
 - Benefits of browser based 3D visualization
 - HTML5 and WebGL
- ▶ Session 2: Overview of browser based virtual globes for 3D Geo-visualization
 - WebGL Earth
 - OpenWebGlobe
 - Cesium
 - Overview of Cesium virtual globe
- ▶ Session 3: Hands-on with WebGL
- ▶ Session 4: Hands-on with Cesium

3D Graphics – A Primer

3D Coordinate Systems

- ▶ X-Axis
 - Horizontally left to right
- ▶ Y-Axis
 - Vertically top to bottom
- ▶ Z-Axis
 - Positive z coming out of the screen
 - Determines how far into or out of the screen an object can be drawn

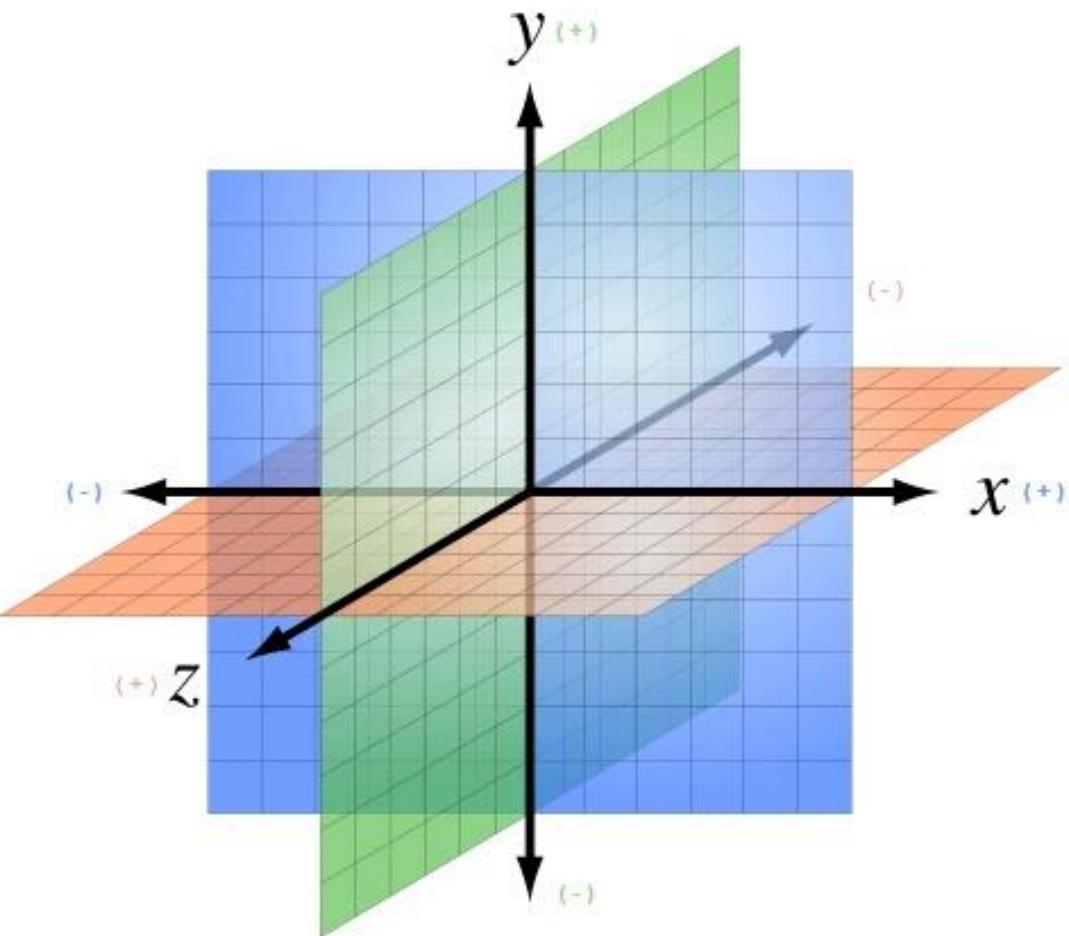


Image source: Parisi, Tony.,2012. *WebGL Up and Running*

Meshes, Polygons, and Vertices

- ▶ Mesh is an object composed of one or more polygonal shapes
- ▶ Constructed out of vertices (x,y,z triples) defining coordinate positions in 3D space
- ▶ Polygons used in meshes are usually triangles and quads
- ▶ 3D meshes are often referred to as models

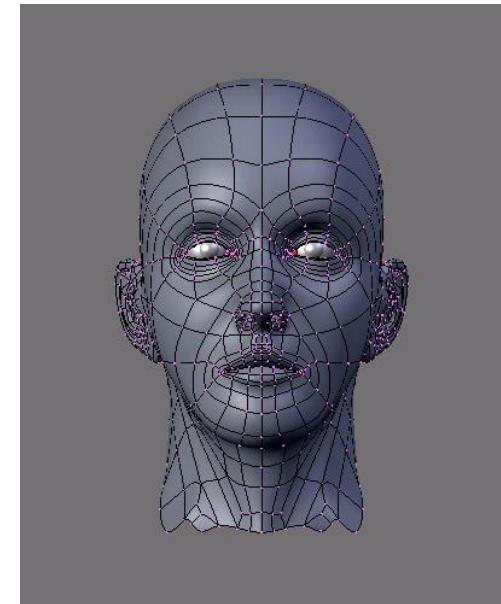


Image source: Parisi, Tony.,2012. *WebGL Up and Running*

Materials, Textures, and Lights

- ▶ Additional attributes of the surface of a mesh (beyond x, y, and z vertex positions)
 - Solid colour
 - Complex piece of information
 - E.g., how light reflects off the object or how shiny the object looks
 - Textures can define the literal surface look
 - E.g., image draped on the surface
 - Materials rely on the presence of one or more lights
 - Defines how a scene is illuminated

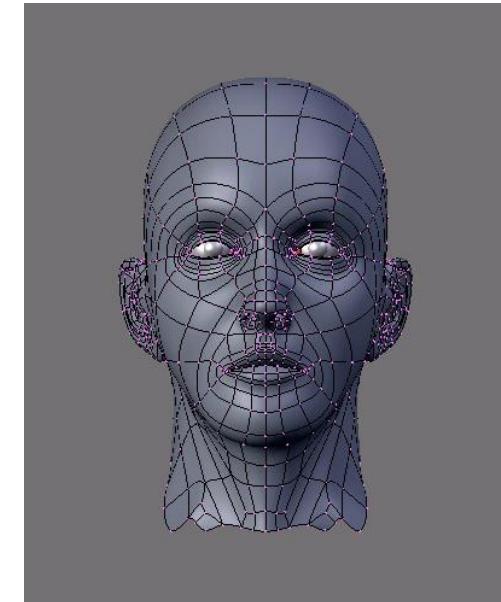


Image source: Parisi, Tony.,2012. *WebGL Up and Running*

Transforms and Matrices

- ▶ Transforms allow a rendered mesh to be **scaled, rotated, and translated around**
 - Without actually changing any value in its vertices
- ▶ A transform is represented by a matrix
 - A mathematical object containing an array of values used to compute the transformed position of vertices.

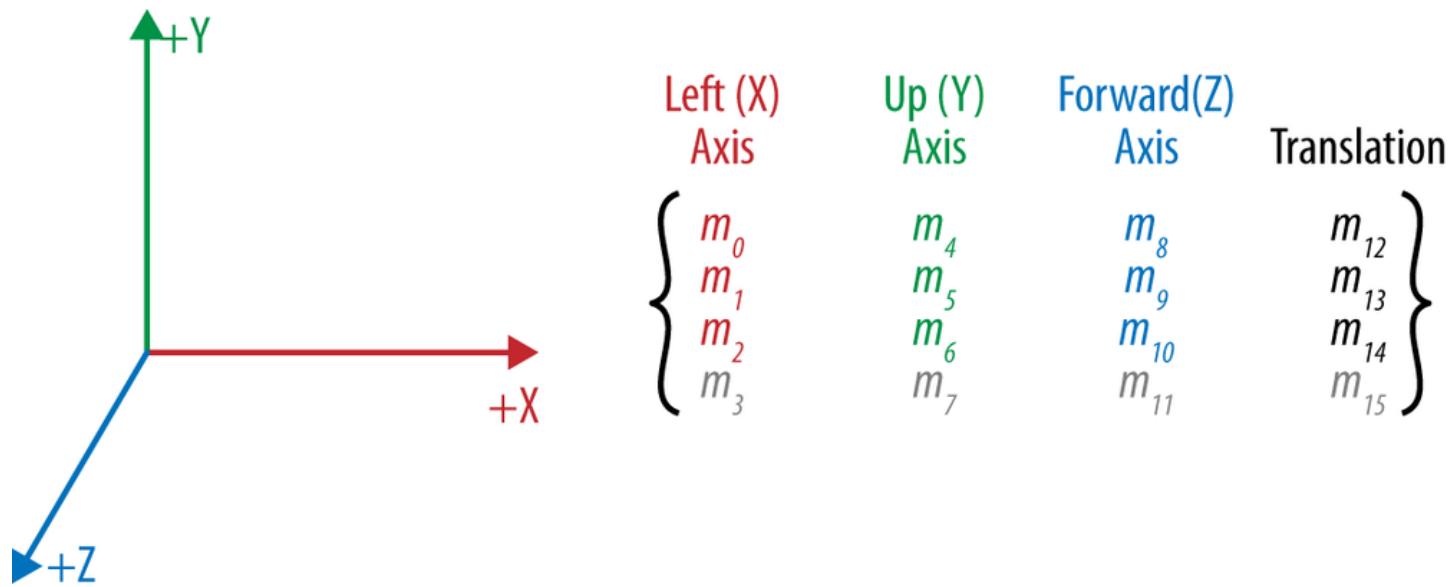


Image source: Parisi, Tony., 2014. *Programming 3D Applications with HTML5 and WebGL*

Cameras, Perspective, and Viewports

- ▶ Camera defines where (relative to the scene) the user is positioned and oriented
- ▶ Perspective is defined by real-world camera properties
 - such as the size of the field of view
- ▶ Viewport is defined by the window or canvas,
 - allows delivering the final rendered image of a 3D scene into a 2D viewport

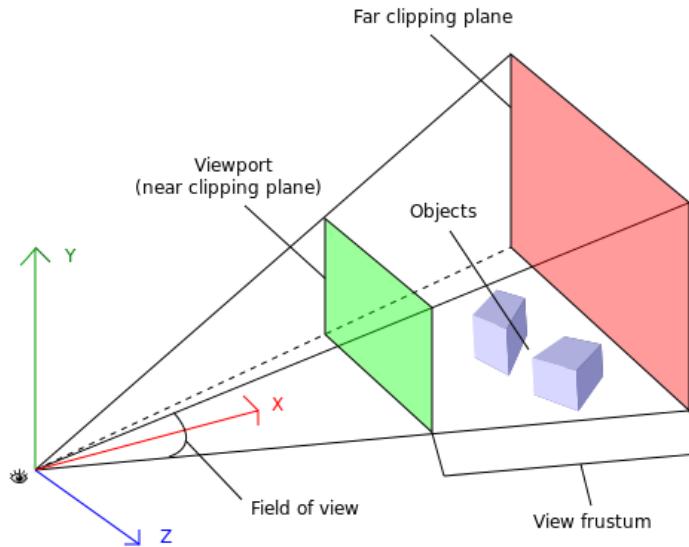


Image source: Parisi, Tony.,2012. *WebGL Up and Running*

Shaders

- ▶ Shaders allows rendering the final image for a mesh
 - Define how vertices, transforms, materials, lights, and the camera interact with one another to create the image
- ▶ A chunk of program code that implements algorithms to get the pixels for a mesh onto the screen
- ▶ Typically defined in a high-level C-like language and compiled into code usable by the GPU

Standards for 3D Graphics

COLLADA

- ▶ COLLABorative Design Activity
- ▶ XML-based schema
- ▶ Exchange format (.dae)
- ▶ Provides comprehensive encoding for
 - Geometry
 - Shaders and effects
 - Physics
 - Kinematics
 - Multiple representations of the same asset
- ▶ Supported by a wide variety of software products



More details: <https://www.khronos.org/collada/>

FBX and OBJ

► FBX

- 3d asset exchange format
- Provided by Autodesk
- Supported by a few proprietary software products
- Mostly useful in digital content creation applications



► OBJ

- Geometry definition file format
- Developed by Wavefront Technologies
- Very simple data format for representing 3D geometry by
 - Position of each vertex
 - UV position of each texture coordinate vertex
 - Vertex normals
 - Faces with list of vertices
 - Texture vertices

More details:

<http://www.autodesk.com/products/fbx/overview>



More details: <http://paulbourke.net/dataformats/obj/>

glTF (intends to become „JPEG for 3D“)

- ▶ New format by Khronos Group
- ▶ Highly suitable for web applications
- ▶ Lightweight format
 - Mesh and animated data can be transmitted as binary
- ▶ Minimal client-side processing
 - Fast loading
- ▶ Run-time neutral
 - Can be implemented in all tools, apps and runtimes
- ▶ Supports materials, animations, skins, cameras and lights
- ▶ Flexible extension system
 - Additional features can be added



More details: <https://github.com/KhronosGroup/glTF>

Web standards for 3D (I)

► Virtual Reality Mark-up Language (VRML)

- 3D on the web began with VRML, in 1995
- Text file format for describing
 - 3D image sequences
 - User interactions (view, movement, rotation)
- VRML viewing requires plug-in for web browsers



More details: <https://www.w3.org/MarkUp/VRML/>

► X3D

- XML-based file format for representing 3D computer graphics
- Successor to VRML
- Supports shaders, geo-location, and many other features
- Being used in domains such as medicine, CAD, GIS and others.
- Like VRML, X3D requires plug-ins to be used on the browsers
- X3DOM is an effort towards plug-in independent X3D support



More details: <http://www.web3d.org/x3d/what-x3d>

Web standards for 3D (II)

- ▶ Universal 3D (U3D)
 - Compressed file format for 3D graphics
 - Proposed by a consortium 3D Industry Forum in 2003
 - 3D objects can be inserted into PDF documents
 - Requires plug-ins for web browsers
- ▶ 3D Markup Language for Web (3DMLW)
 - XML-based file-format for creating 3D and 2D content on the Web
 - Requires plug-ins for web browsers
- ▶ Adobe Flash 3D
 - Support for 3D effects via the addition of new classes and methods
 - Requires proprietary Flash browser plug-in

U3D



Existing 3D standards (Summary)

- ▶ They are all interchange formats
 - Meant for exchanging data for further editing
 - Most of the formats are based on Scene Graph modeling paradigm
 - Can be loaded and rendered very efficiently by 3D viewers
- ▶ Models need a web-friendly runtime format
- ▶ Most solutions work only with certain browsers or with additional plug-ins
 - Plug-ins are **inconvenient to install, troubleshoot and manage**, and
 - Plug-ins **may cause browser crashes and other problems**
- ▶ Future browser generations do not support plug-ins anymore



Browser based 3D Visualization

Browser based 3D Visualization

- ▶ 3D Visualization on the Web is challenging!
 - Large and complex data sets/volumes
 - Intricate textures and shapes
 - Requires high bandwidth and computing power
- ▶ 3D Visualization on the browsers is possible
 - Advancements in computer graphics
 - High computational capacities of devices
 - Faster CPUs, graphics processors, video cards, graphics accelerators
 - State-of-the-art web technologies
 - JavaScript, HTML5, WebGL
- ▶ Active research area
 - Numerous standards have been developed

HTML5

► HTML5

- Mark-up language for structuring and presenting contents on the web
- Open standard format
- Provides common platform for web applications

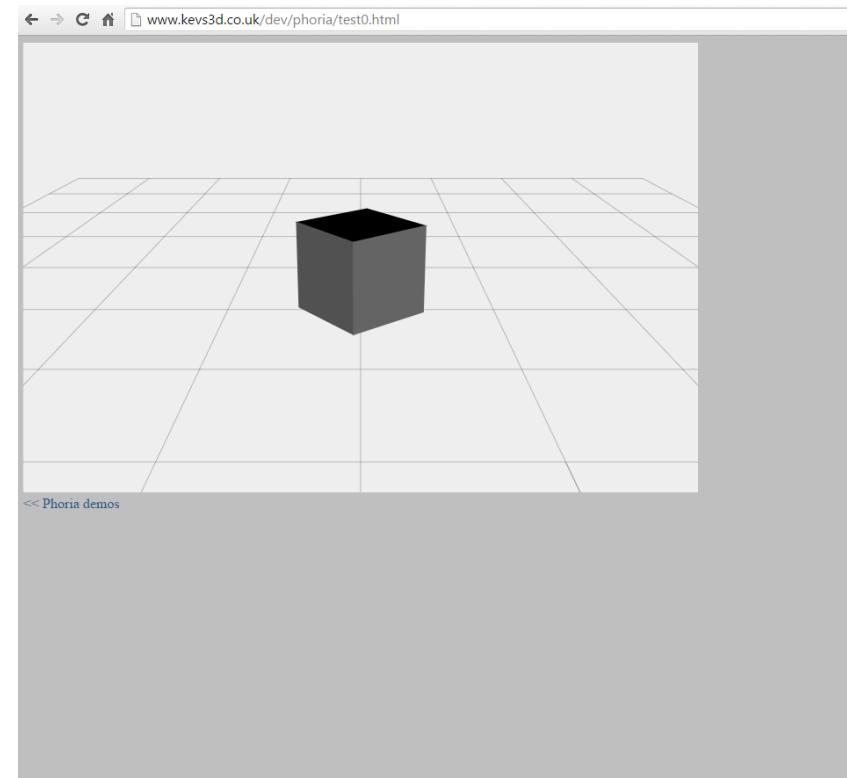
► Introduced new elements

- Audio/Video
- **Canvas**
- Scalable Vector Graphics (SVG)
- Geo-location
- Web Workers
- Web Sockets
- Many more..



HTML5 <canvas>

- ▶ Used to draw graphics, on the fly, via scripting
- ▶ The <canvas> element is only a container for graphics
 - The graphics are drawn by scripting
- ▶ Several methods for drawing
 - Path
 - Boxes
 - Circles
 - 3D Objects



More details: <http://www.kevs3d.co.uk/dev/phoria/test0.html>

WebGL

- ▶ WebGL: JavaScript implementation of OpenGL ES 2.0
 - Operating System independent
 - runs in all recent browsers (Chrome, Firefox, IE, Safari)
 - application can be located on a remote server
 - rendering is done within browser using local hardware
 - uses HTML5 canvas element
 - integrates with standard Web packages and apps
 - CSS
 - jQuery



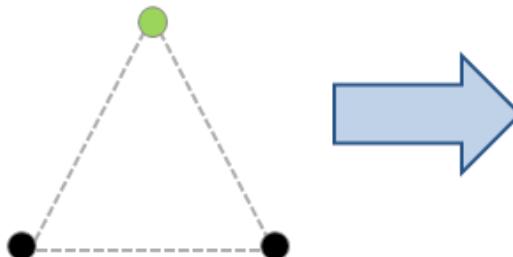
HTML5 and WebGL

- ▶ HTML5 and WebGL have revolutionized the 3D web applications
- ▶ 3D capabilities can now be realized directly within the browsers
- ▶ Without any need for an additional plug-ins or extensions
- ▶ Hardware accelerated 3D functionality on the web
 - Utilizes hardware's graphics card memory
- ▶ Being used to solve complex problems
 - Medicine
 - CAD
 - Geospatial domain
 - Virtual Globes
 - Many more

Components of WebGL

- ▶ **HTML5 and CSS** for the user interface, including a `<canvas>` tag for the 3D scene
- ▶ **JavaScript** to process user events and communicate with servers if necessary
- ▶ **Shaders** running on the GPU to control rendering
 - Vertex shaders - handles the processing of individual vertices
 - Fragment shaders – handles the processing of fragments

In the Vertex Shader:



Vertex Coloring

In the Fragment Shader:



Pixel Coloring

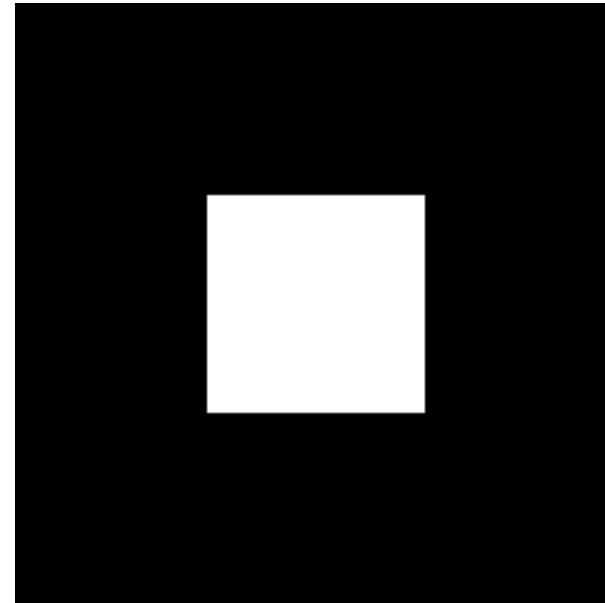
Image source: Cantor, Diego.,2012. *WebGL Beginner's Guide*

WebGL Programming in a nutshell

- ▶ All WebGL programs must do the following:
 1. Create a canvas element.
 2. Obtain a drawing context for the canvas.
 3. Initialize the viewport.
 4. Create one or more buffers containing the data to be rendered (typically vertices).
 5. Create one or more matrices to define the transformation from vertex buffers to screen space.
 6. Create one or more shaders to implement the drawing algorithm.
 7. Initialize the shaders with parameters.
 8. Draw.

Demonstration

- ▶ A simple WebGL example
- ▶ Generate one basic square
- ▶ Has all the elements of a more complex application
 - Vertex Shader
 - Fragment Shader
 - HTML Canvas



Source: Parisi, Tony.,2012. *WebGL Up and Running*

```
function initWebGL(canvas) {
    var gl;
    try {
        gl = canvas.getContext("experimental-webgl");
    } catch (e) {
        var msg = "Error creating WebGL Context! " + e.toString();
        alert(msg);
        throw Error(msg);
    }
    return gl;
}

function initViewport(gl, canvas)
{
    gl.viewport(0, 0, canvas.width, canvas.height);
}
var projectionMatrix, modelViewMatrix;

function initMatrices()
{
    // The transform matrix for the square - translate back in Z for the camera
    modelViewMatrix = new Float32Array(
        [1, 0, 0, 0,
         0, 1, 0, 0,
         0, 0, 1, 0,
         0, 0, -3.333, 1]);

    // The projection matrix (for a 45 degree field of view)
    projectionMatrix = new Float32Array(
        [2.41421, 0, 0, 0,
         0, 2.41421, 0, 0,
         0, 0, -1.002002, -1,
         0, 0, -0.2002002, 0]);
}

// Create the vertex data for a square to be drawn
function createSquare(gl) {
    var vertexBuffer;
    vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    var verts = [
        .5, .5, 0.0,
        -.5, .5, 0.0,
        .5, -.5, 0.0,
        -.5, -.5, 0.0
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(verts), gl.STATIC_DRAW);
    var square = {buffer:vertexBuffer, vertSize:3, nVerts:4, primtype:gl.TRIANGLE_STRIP};
    return square;
}

function createShader(gl, str, type) {
    var shader;
    if (type == "fragment") {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    } else if (type == "vertex") {
        shader = gl.createShader(gl.VERTEX_SHADER);
    } else {
        return null;
    }
    gl.shaderSource(shader, str);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert(gl.getShaderInfoLog(shader));
        return null;
    }
    return shader;
}

function initShader(gl) {
    var fragmentShader = createShader(gl, fragmentShaderSource, "fragment");
    var vertexShader = createShader(gl, vertexShaderSource, "vertex");
    // link them together into a new program
    .....

    function draw(gl, obj) {
        // clear the background (with black)
        gl.clearColor(0.0, 0.0, 0.0, 1.0);
        gl.clear(gl.COLOR_BUFFER_BIT);
        // set the vertex buffer to be drawn
        gl.bindBuffer(gl.ARRAY_BUFFER, obj.buffer);
        // set the shader to use
        gl.useProgram(shaderProgram);

        gl.vertexAttribPointer(shaderVertexPositionAttribute, obj.vertSize, gl.FLOAT, false, 0, 0);
        gl.uniformMatrix4fv(shaderProjectionMatrixUniform, false, projectionMatrix);
        gl.uniformMatrix4fv(shaderModelViewMatrixUniform, false, modelViewMatrix);
        // draw the object
        gl.drawArrays(obj.primtype, 0, obj.nVerts);
    }

    function onLoad() {
        var canvas = document.getElementById("webglcanvas");
        var gl = initWebGL(canvas);
        initViewport(gl, canvas);
        initMatrices();
        var square = createSquare(gl);
        initShader(gl);
        draw(gl, square);
    }
}
```

WebGL Code for a basic Square

Popular WebGL Frameworks

- ▶ WebGL is a **low-level API and complex in nature**
 - 150 lines of codes for just a basic square
- ▶ Frameworks are available to create WebGL contents quickly and easily
 - **Avoids complex low-level programming**
- ▶ Popular frameworks:
 - Three.js
 - Cross-browser JavaScript API
 - Create and display animated 3D graphics on browsers
 - Lightweight in nature
 - Scene.js
 - Open-source JavaScript API
 - Intended towards fast rendering of large numbers of individually picked objects

Three.js Code for a basic Square

```
<script src="../libs/Three.js"></script>
<script>
    function onLoad()
    {
        // Grab our container div
        var container = document.getElementById("container");

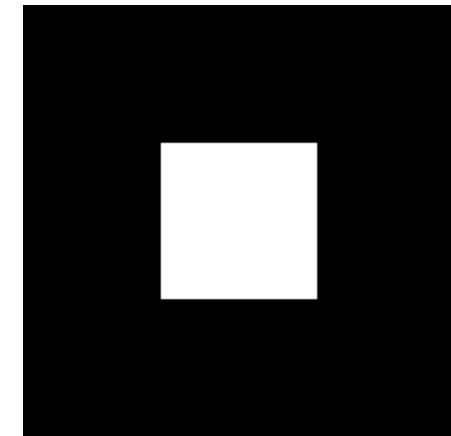
        // Create the Three.js renderer, add it to our div
        var renderer = new THREE.WebGLRenderer();
        renderer.setSize(container.offsetWidth, container.offsetHeight);
        container.appendChild( renderer.domElement );

        // Create a new Three.js scene
        var scene = new THREE.Scene();

        // Create a camera and add it to the scene
        var camera = new THREE.PerspectiveCamera( 45, container.offsetWidth / container.offsetHeight, 1, 4000 );
        camera.position.set( 0, 0, 3.3333 );
        scene.add( camera );

        // Now, create a rectangle and add it to the scene
        var geometry = new THREE.PlaneGeometry(1, 1);
        var mesh = new THREE.Mesh( geometry, new THREE.MeshBasicMaterial( ) );
        scene.add( mesh );

        // Render it
        renderer.render( scene, camera );
    }
</script>
```



Source: Parisi, Tony.,2012. *WebGL Up and Running*

X3DOM

- ▶ Open-source framework for integrating and manipulating X3D scenes as HTML5 DOM elements
- ▶ Rendered via
 - An X3D plug-in
 - Flash/Stage3D
 - **WebGL**
- ▶ Declarative 3D scene description and run-time behaviour
 - Without any low-level JavaScript coding
- ▶ Library in a single JavaScript file (*x3dom.js*)
- ▶ When loaded, creates a new DOM object, *x3dom*
- ▶ When loaded, X3DOM automatically parses the HTML document for 3D models
 - Specifically, 3D-models specified inside `<x3d>` tag

The logo for x3dom, consisting of the word "x3dom" in a bold, lowercase, sans-serif font. The letters are a dark blue color.

Demonstrations

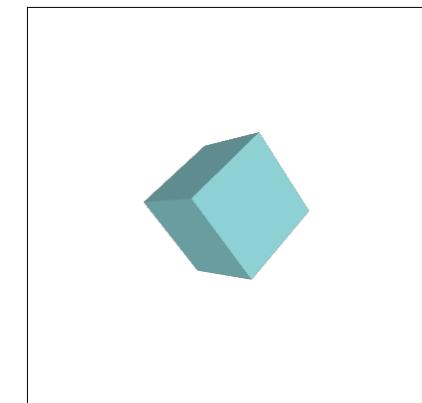
- ▶ Hello World – demonstrates how a simple X3D scene can be displayed in browser

```
<!DOCTYPE html >
<html >
  <head>
    <link rel="stylesheet" type="text/css" href="x3dom.css" />
    <script type="text/javascript" src="x3dom.js"></script>
  </head>
  <body>
    <x3d xmlns="http://www.x3dom.org/x3dom" width="400px" height="400px" >
      <scene>
        <viewpoint position='0 0 10' ></viewpoint>
        <shape>
          <appearance>
            <material diffuseColor='0.603 0.894 0.909' ></material>
          </appearance>
          <box DEF='box' ></box>
        </shape>
      </scene>
    </x3d>
  </body>
</html>
```

Loads X3DOM library

Embedded X3D Content

Demo Link: http://examples.x3dom.org/example/x3dom_helloWorld.html





Session 2:

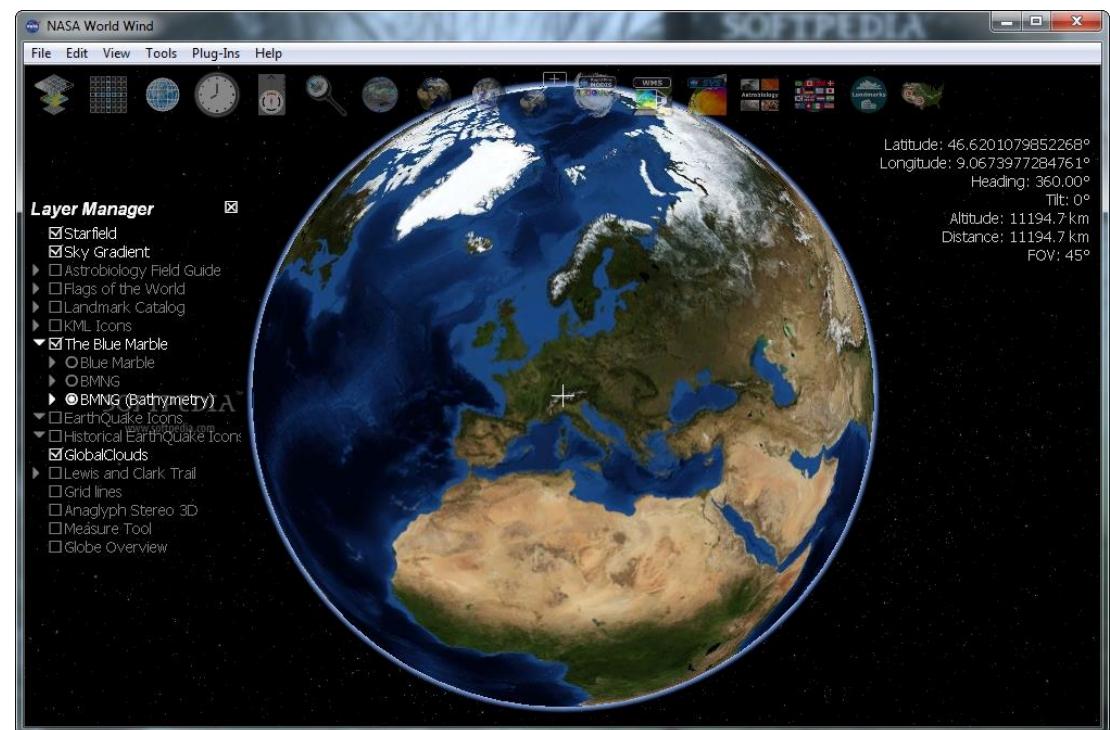
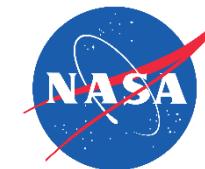
Virtual Globes

Concepts of Virtual Globes

- ▶ 3D-representation of the Earth
- ▶ Medium to visualize and interact with global geospatial data
 - Visualization of georeferenced data
- ▶ Provides sophisticated (and standardized) interaction and navigation capabilities
 - Freely move around in the virtual environment
 - Changing viewing angle and positions
- ▶ Multiple views on the surface of Earth
 - Geographical features (Terrains)
 - Man-made features (Roads or Buildings)
 - Demographic quantities (Population)

Popular Virtual Globes

- ▶ NASA World Wind
 - Open source virtual globe
 - Developed by NASA
 - Java based Software Development Kit
 - Supports
 - USGS satellite imageries,
 - aerial photography,
 - topographic maps,
 - KML and COLLADA files

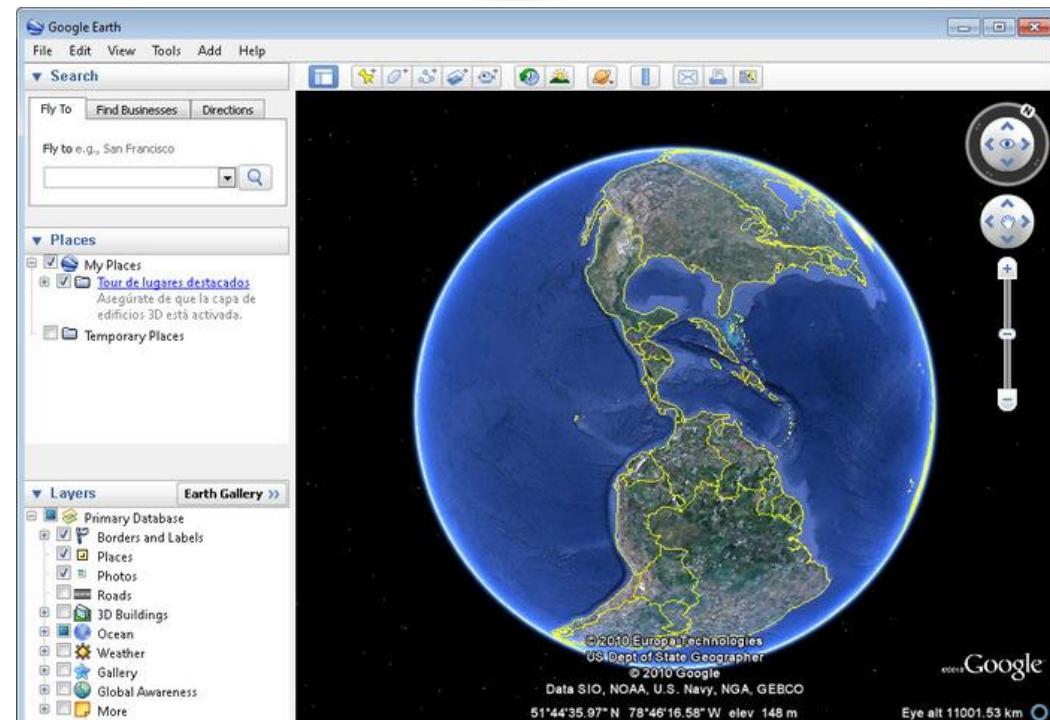


More details: <http://worldwind.arc.nasa.gov/java/>

Popular Virtual Globes

► Google Earth

- Free (not open source) virtual globe
- Most popular till date
- Supports
 - Satellite imageries,
 - aerial photography,
 - topographic maps,
 - KML and COLLADA files
- Runs on most operating systems and devices (desktop, laptop, mobiles)
 - Google Earth plug-in for web browsers
deprecated in December, 2015.



More details: <https://www.google.com/earth/>

How does WebGL benefit Virtual Globes?

- ▶ Development of web based virtual globes
 - Plug-in free
 - Cross-platform
 - Cross-browser
 - Cross-device
 - Free and open source
 - Seamless performance on browsers
- ▶ WebGL based virtual globes

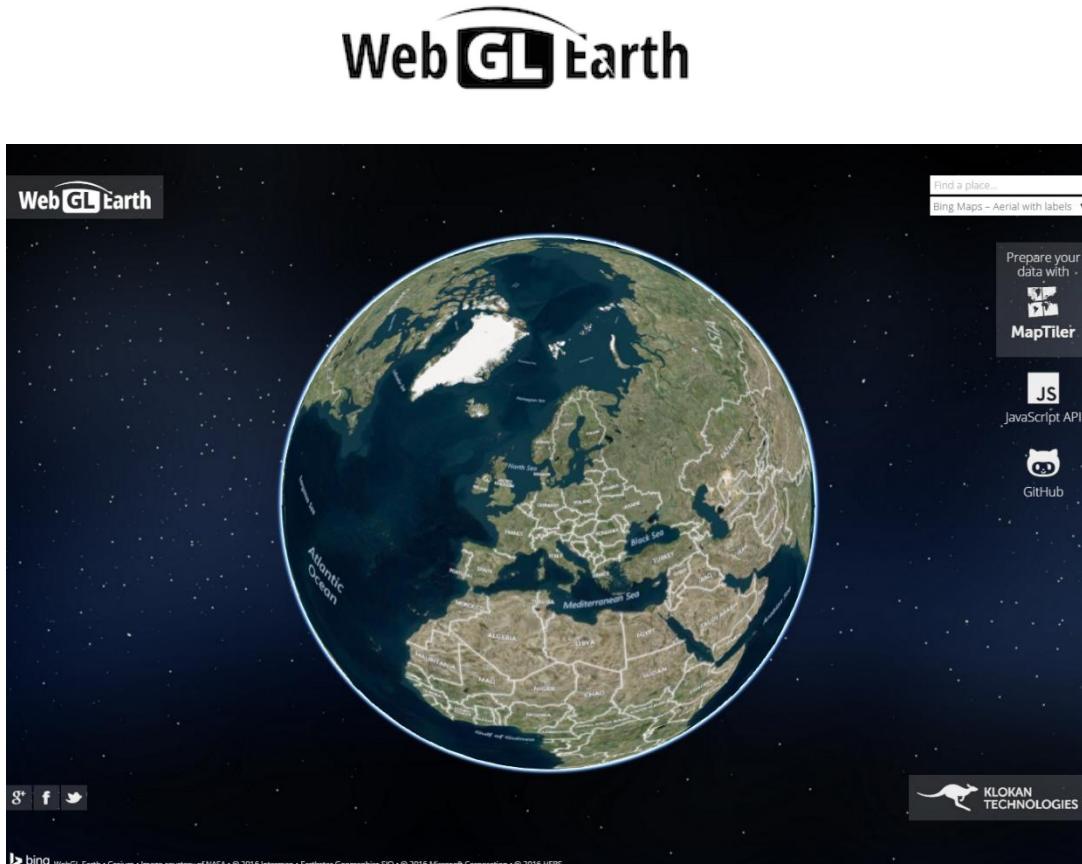
- WebGL Earth
- OpenLayers 3
- OpenWebGlobe
- Cesium

And many more..



WebGL Earth

- ▶ Open source JavaScript API
- ▶ Visualization of
 - maps,
 - satellite imagery and
 - aerial photography
- ▶ Features
 - Supports camera-dependent functionalities (rotation, zoom, tilt)
 - Supports existing maps from multiple sources (OSM, Bing)
 - Supports custom map tiles



More details: <https://www.webglearth.com>

OpenWebGlobe

- ▶ Open source SDK to develop web based applications
- ▶ Processes and interactively visualizes vast volumes of geospatial data
- ▶ Supports
 - Image Data
 - Elevation Data
 - Points of interest
 - Vector data
 - 3D Objects
- ▶ Parallel and scalable computing environment
 - Processes very large amount of data, upto TB
- ▶ Data must be pre-processed



More details: <http://www.openwebglobe.org/>

Cesium

- ▶ Open source JavaScript API
- ▶ Visualize high-resolution worldwide terrain
- ▶ Draw imagery layers using WMS, TMS, OSM, Bing and ESRI standards
- ▶ Draw vector data from GeoJSON and TopoJSON
- ▶ Draw 3D models using COLLADA and glTF
- ▶ Create data-driven time-dynamic scenes with CZML



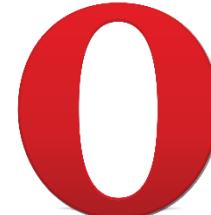
More details: <https://cesiumjs.org/>



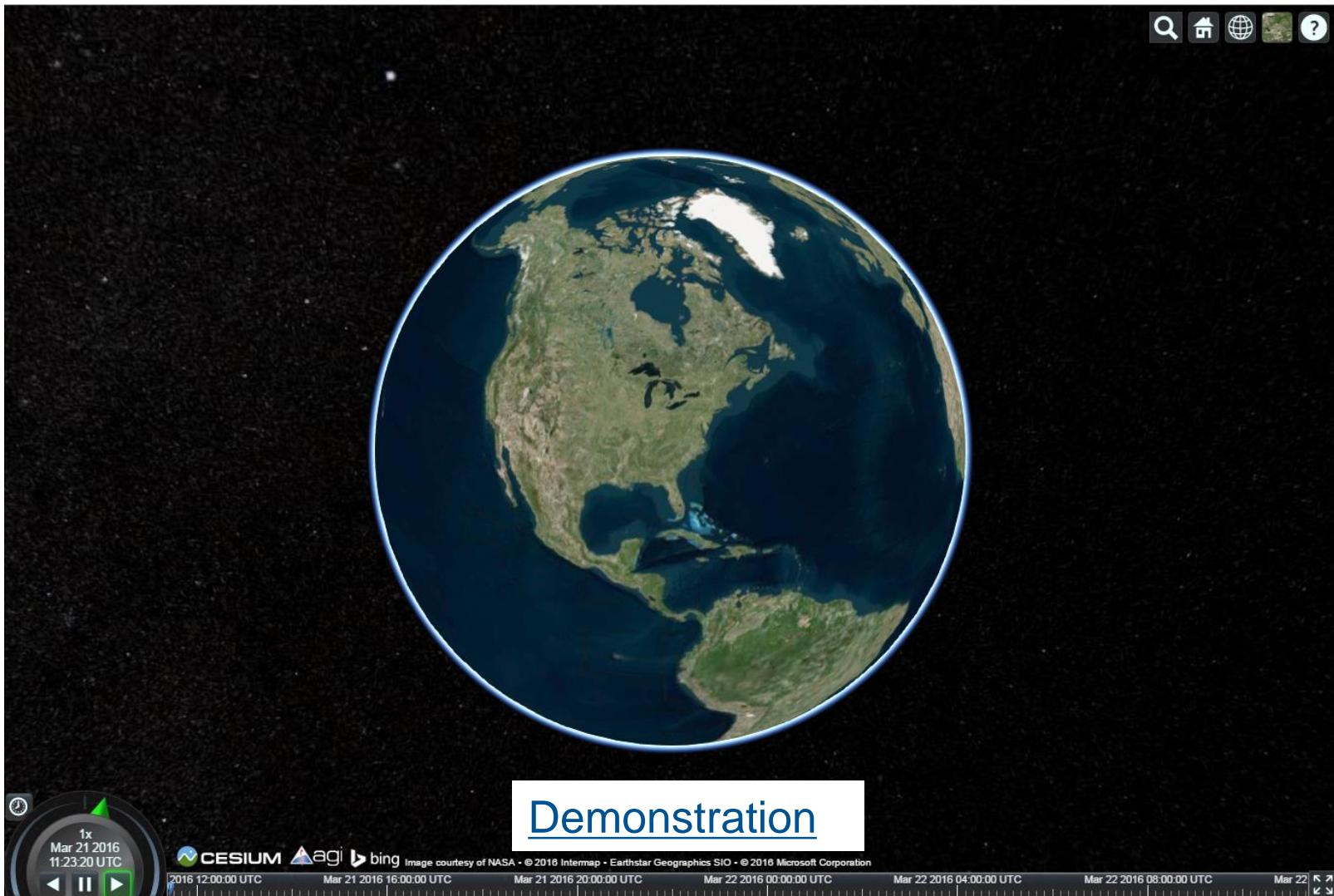
Introduction to Cesium WebGL Virtual Globe

What is Cesium WebGL Virtual Globe?

- ▶ JavaScript library for 3D globes and 2D maps
- ▶ Built using WebGL and HTML5 standards
- ▶ Runs almost everywhere (including mobile devices)
- ▶ Apache 2.0 License
- ▶ Tuned for time-dynamic data
- ▶ Founded by US based company AGI, supported by a large community



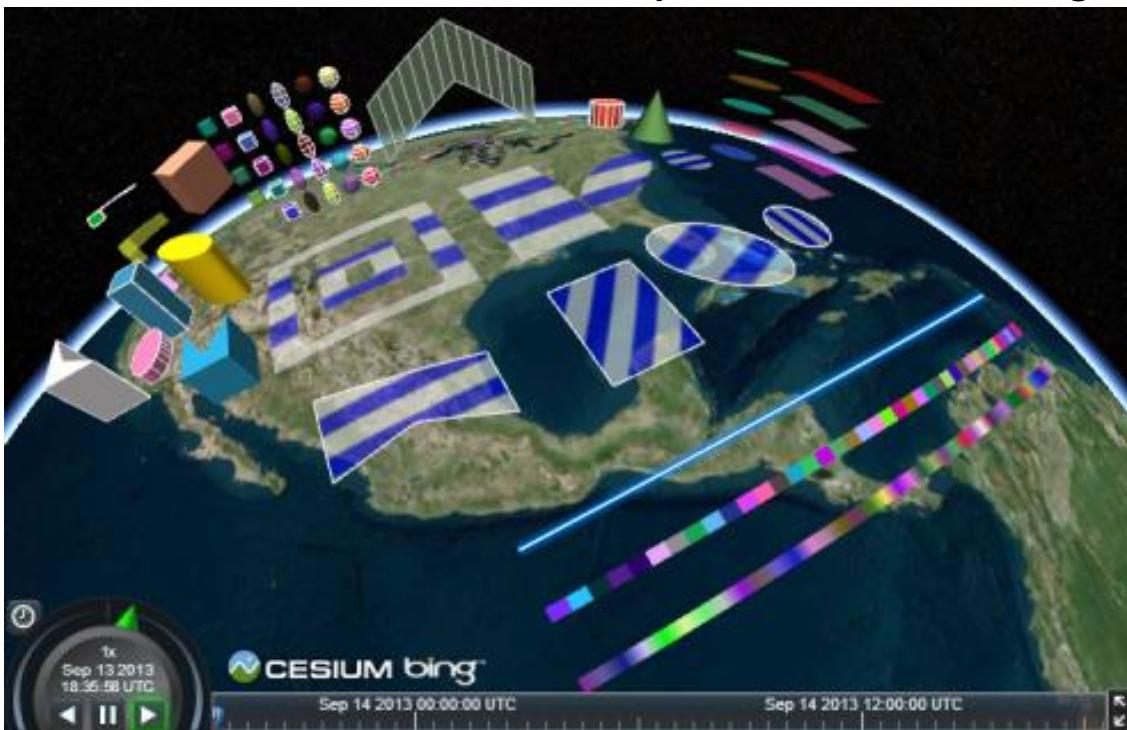
Cesium Viewer



Cesium Features

Vector Geometry

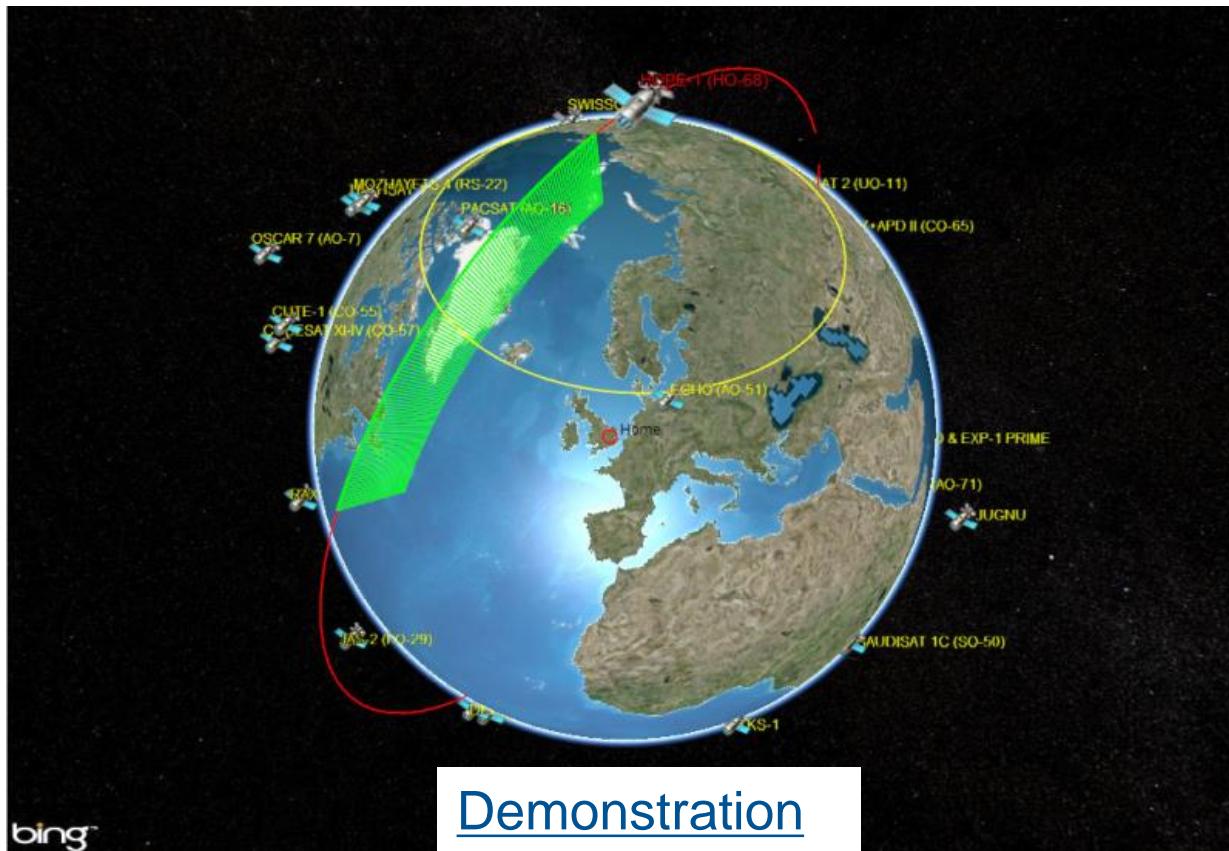
- ▶ Direct support for
 - KML
 - GeoJSON
 - TopoJSON
- ▶ Cesium API draws a wide variety of features and geometry



Demonstration

Time-Dynamic Visualization

- ▶ High-fidelity time-dynamic simulation
- ▶ Real-time telemetry streaming
- ▶ 4D visualization
- ▶ CZML



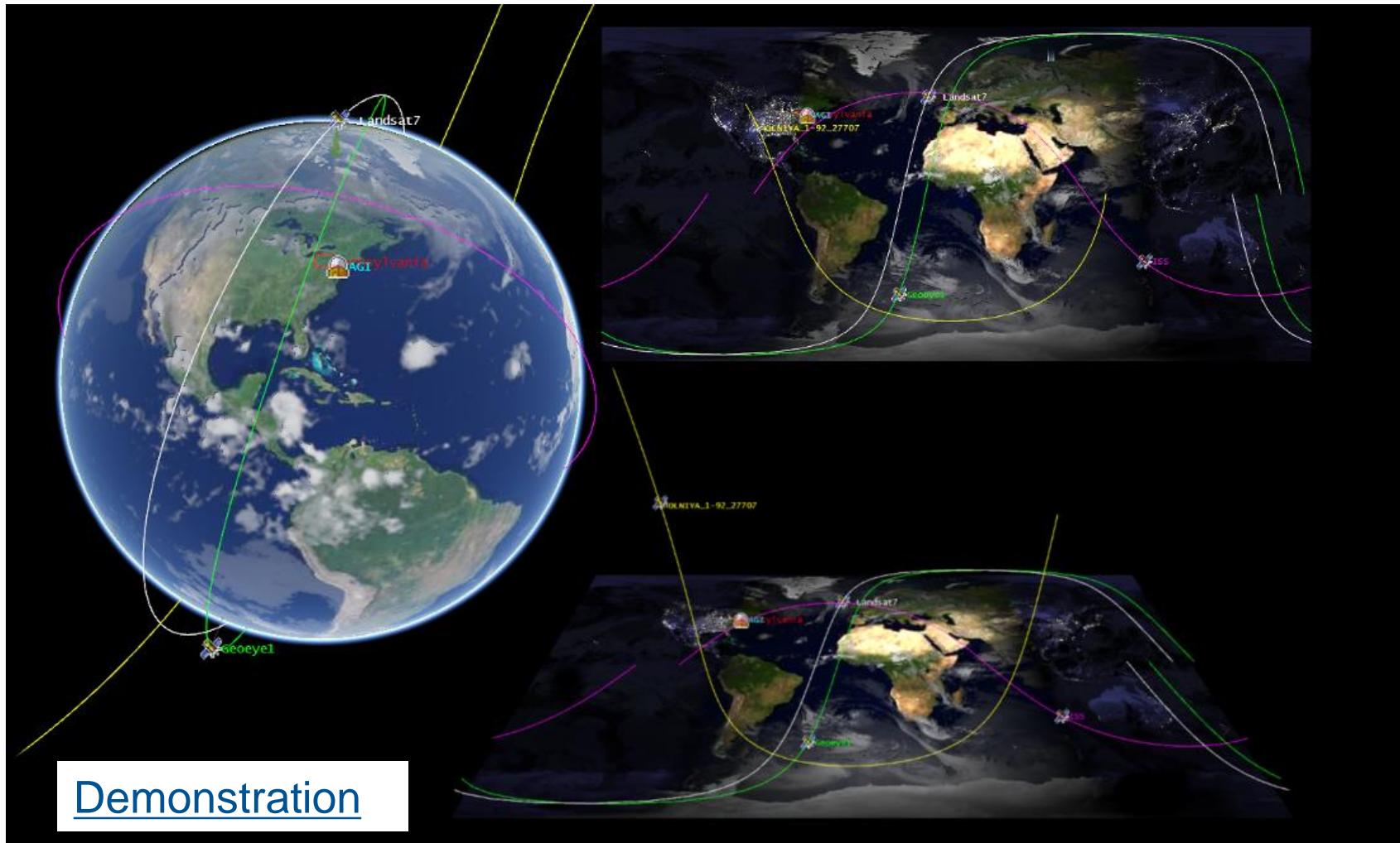
CZML

- ▶ Short for Cesium Modeling Language
- ▶ A streamable JSON scene description for data-driven visualization
- ▶ Describes properties that change over time
- ▶ Supports time-variation for
 - Lines
 - Points
 - Models
 - Graphical primitives

More details: <https://github.com/AnalyticalGraphicsInc/cesium/wiki/CZML-Guide>

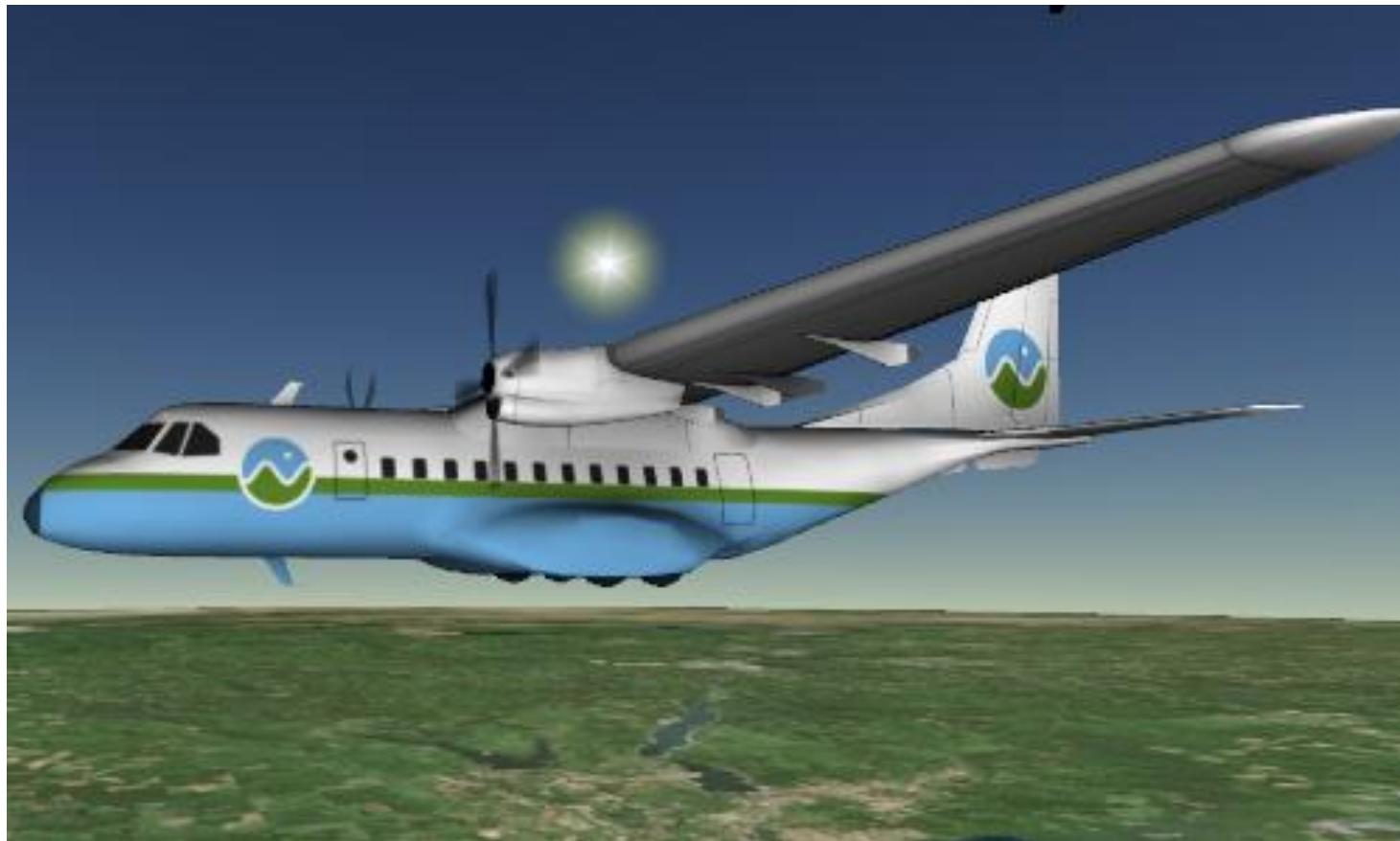
3D, 2D, and 2.5D Columbus View

- ▶ Use one API for three views



3D Models

- ▶ Visualize 3D models using glTF



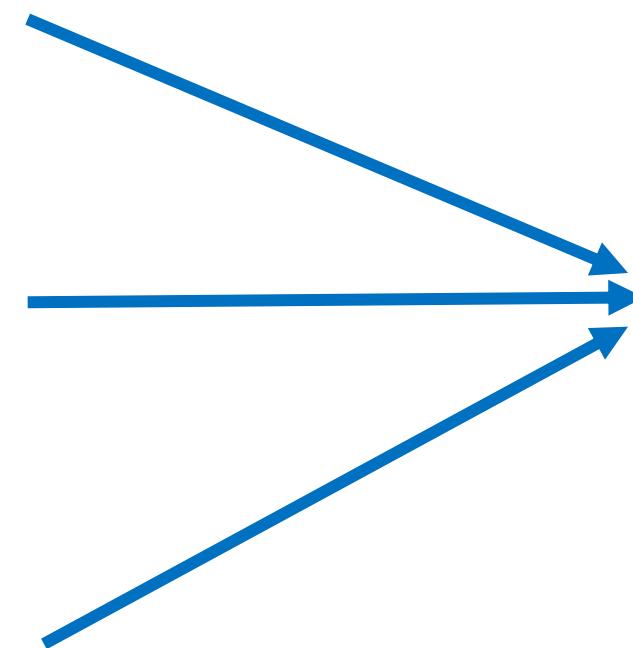
[Demonstration](#)

glTF Conversion

COLLADA

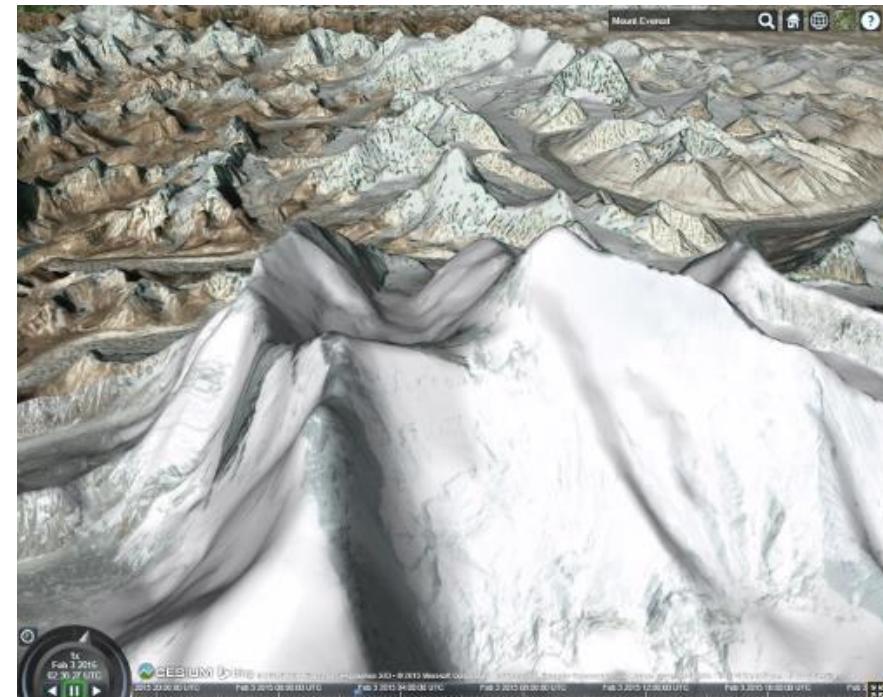


glTF™



Global Terrain

- ▶ Stream high-resolution global terrain,
using
 - Open-quantized mesh format
 - Traditional height maps
 - External terrains
- ▶ Cesium provides free access to
30 m resolution SRTM DEM
- ▶ User specified DEMs and DSMs
are possible



[Demonstration](#)

Imagery Layer

- ▶ Stream imagery
 - Supports a wide array of open standards
 - Custom tiling schemes



[Demonstration](#)

Cesium Sandcastle

Demonstration

The screenshot shows a Cesium-based 3D geovisualization application. On the left, there is a code editor window displaying JavaScript code for creating a 3D aircraft model. The main area shows a 3D rendering of a white and blue airplane flying over a satellite map of a city and river system. A navigation bar at the bottom includes a clock, time information (Mar 21 2016 11:36:15 UTC), and a timeline showing dates from Mar 21 2016 to Mar 22 2016. Below the main view, there are tabs for 'Gallery' (selected), 'Console', 'Showcases', 'Tutorials', 'Beginner', 'Geometries', 'DataSources', 'CZML', and 'All'. At the bottom, there is a grid of thumbnail images for various Cesium features: 3D Models, Billboards, Camera, Camera Tutorial, Cardboard, Cesium Inspector, Cesium Widget, and Custom DataSource.

```
1 var viewer = new Cesium.Viewer('cesiumContainer', {  
2     scene3DOnly: true,  
3     infoBox: false,  
4     selectionIndicator: false  
5});  
6  
7 function createModel(url, height) {  
8     viewer.entities.removeAll();  
9  
10    var position = Cesium.Cartesian3.fromDegrees(-123.0744619, 44.0503  
11    var heading = Cesium.Math.toRadians(135);  
12    var pitch = 0;  
13    var roll = 0;  
14    var orientation = Cesium.Transforms.headingPitchRollQuaternion(pos  
15  
16    var entity = viewer.entities.add({  
17        name: url,  
18        position: position,  
19        orientation: orientation,  
20        model: {  
21            uri: url,  
22            minimumPixelSize: 128,  
23            maximumScale: 20000  
24        }  
25    });  
26    viewer.trackedEntity = entity;  
27}  
28  
29 var options = [{  
30    text: 'Aircraft',  
31    onSelect: function() {  
32        createModel('../..../SampleData/models/CesiumAir/Cesium_Air.glb'  
33    }  
34 }, {  
35    text: 'Ground vehicle',  
36    onSelect: function() {  
37        createModel('../..../SampleData/models/CesiumGround/Cesium_Groun  
38    }  
39 }, {  
40    text: 'Milk truck',  
41    onSelect: function() {  
42        createModel('../..../SampleData/models/CesiumMilkTruck/CesiumMil  
43    }  
44 }];
```

Cesium Architecture

Cesium Apps

Specific 3D Web applications can be developed on the basis of the following Cesium APIs except "renderer"

Widgets

3D visualization component with navigation, imagery selection, geocoding etc

DataSources

High-level Entity API encapsulating Scene types and time-dynamic properties. CZML, KML Loader

Scene

Out-of-core terrain and imagery engine. 3D models, geometries, and vector data

Renderer

Internal API for abstraction of WebGL Library

Core

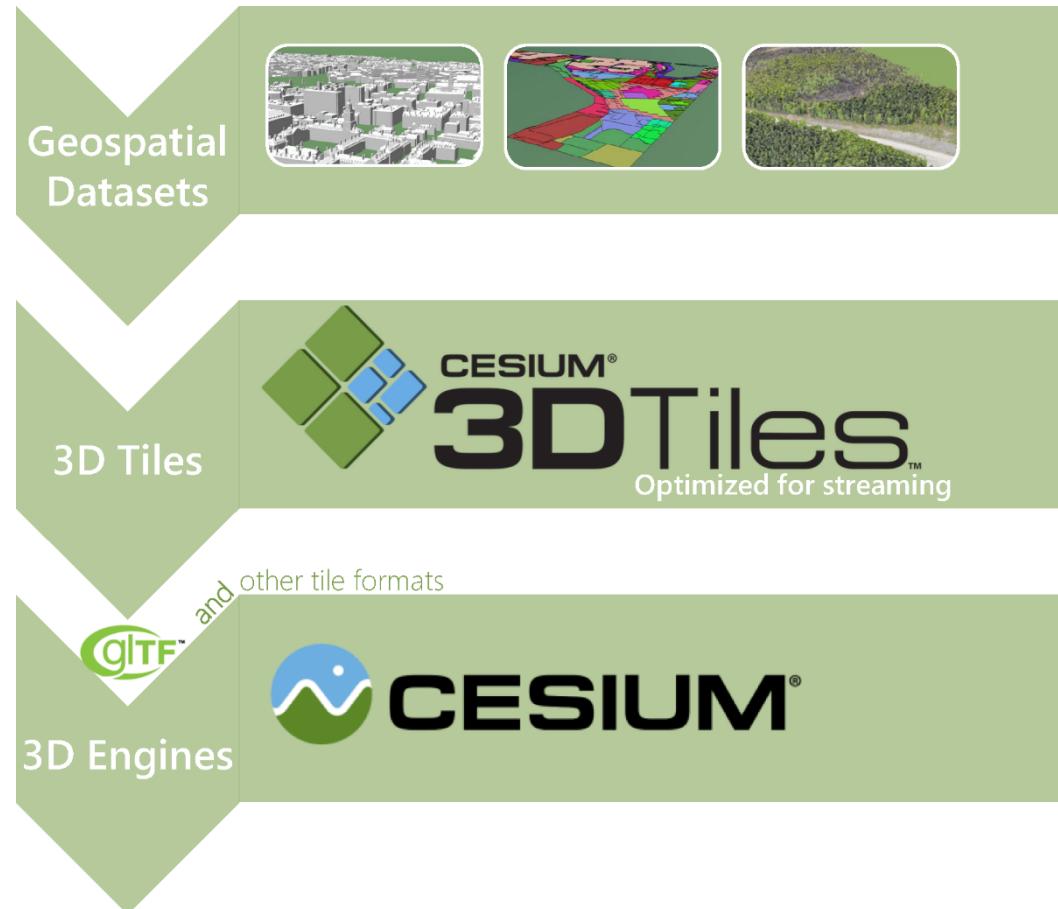
Collection of mathematical and geometrical features such as coordinate transformations



Source: <https://cesiumjs.org/2015/05/15/Graphics-Tech-in-Cesium-Architecture/>

3D Tiles by Cesium

- ▶ An open specification for streaming massive 3D geospatial datasets
 - Buildings
 - Trees
 - Point clouds
 - Temporal data
- ▶ Leverages glTF for models
- ▶ In-progress Work



Source: <http://cesiumjs.org/2015/08/10/introducing-3d-tiles/>



Session 3:

Hands-on with WebGL

Sample Datasets

- ▶ Code examples from *WebGL Up and Running* by Tony Parisi

<https://github.com/tparisi/WebGLBook>

- ▶ Download and extract the folder in C Drive
 - E.g. C:/Workshop

Setting up a local server

- ▶ Install Python
- ▶ Include C:/python27 in Windows Path
- ▶ Open Command Prompt
 - Navigate to C:/Workshop
 - Run `python -m SimpleHTTPServer`

```
C:\Windows\system32\cmd.exe
C:\Workshop>python -m SimpleHTTPServer
```

- ▶ Open a Web Browser
 - Go to <http://localhost:8000/>

Working with Three.js

► Chapter 2 -> example2-1.html

```
<script src="../libs/Three.js"></script>
<script>
    function onLoad()
    {
        // Grab our container div
        var container = document.getElementById("container");

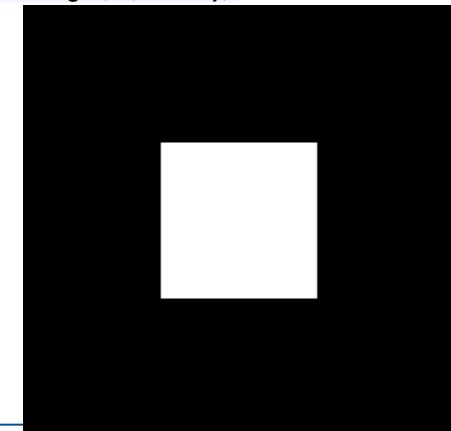
        // Create the Three.js renderer, add it to our div
        var renderer = new THREE.WebGLRenderer();
        renderer.setSize(container.offsetWidth, container.offsetHeight);
        container.appendChild( renderer.domElement );

        // Create a new Three.js scene
        var scene = new THREE.Scene();

        // Create a camera and add it to the scene
        var camera = new THREE.PerspectiveCamera( 45, container.offsetWidth / container.offsetHeight, 1, 4000 );
        camera.position.set( 0, 0, 3.3333 );
        scene.add( camera );

        // Now, create a rectangle and add it to the scene
        var geometry = new THREE.PlaneGeometry(1, 1);
        var mesh = new THREE.Mesh( geometry, new THREE.MeshBasicMaterial( ) );
        scene.add( mesh );

        // Render it
        renderer.render( scene, camera );
    }
</script>
```



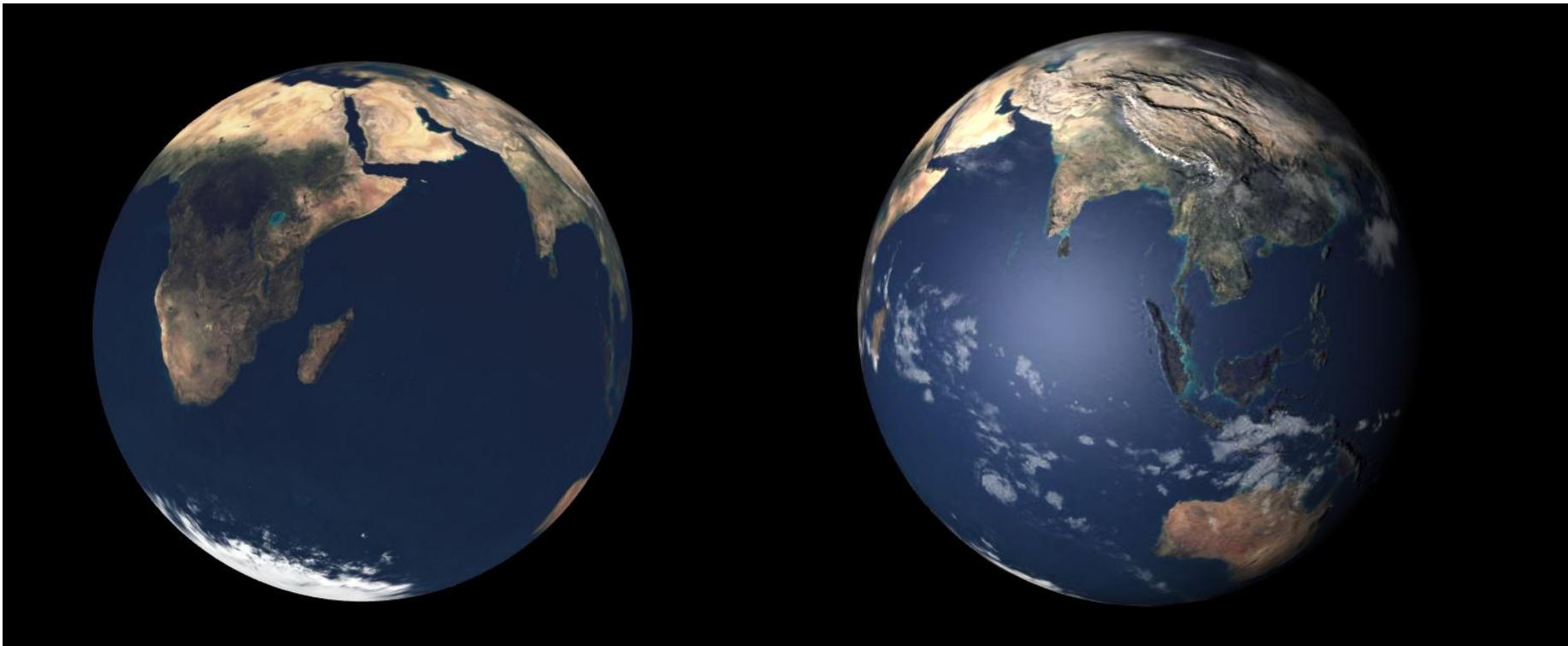
Working with Three.js

- ▶ Chapter 2 -> example2-2.html
 - Cube with an image wrapped onto its faces
 - Supports animation



Basic Earth Example

- ▶ Chapter 3
 - Creating realism with multiple textures

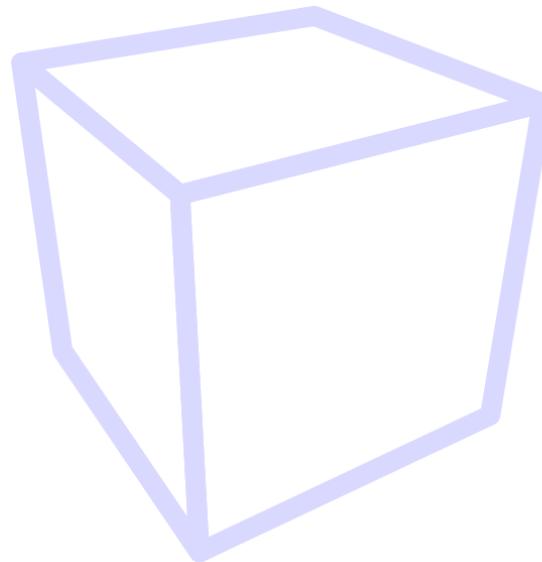


Live Coding with Three.js Library

- ▶ <http://mrdoob.com/projects/htmleditor/>

mndoob.com/projects/htmleditor/

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <style>
6       body {
7         background-color: #ffffff;
8         margin: 0;
9         overflow: hidden;
10      }
11    </style>
12  </head>
13 <body>
14   <script src="http://cdnjs.cloudflare.com/ajax/libs/three.js/r57/three.min.js"></script>
15   <script>
16
17   var camera, scene, renderer;
18   var geometry, material, mesh;
19
20   var init = function () {
21
22     renderer = new THREE.CanvasRenderer();
23     renderer.setSize( window.innerWidth, window.innerHeight );
24     document.body.appendChild( renderer.domElement );
25
26     camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 1, 1000 );
27     camera.position.z = 500;
28
29     scene = new THREE.Scene();
30
31     geometry = new THREE.CubeGeometry( 200, 200, 200 );
32     material = new THREE.MeshBasicMaterial( { color: 0x0000ff, wireframe: true, wireframeLinewidth: 20 } );
33
34     mesh = new THREE.Mesh( geometry, material );
35     scene.add( mesh );
36
37   }
38
39   var animate = function () {
40
41     requestAnimationFrame( animate );
42
43     mesh.rotation.x = Date.now() * 0.0005;
44     mesh.rotation.y = Date.now() * 0.001;
45
46     renderer.render( scene, camera );
47
48   }
49
50   init();
51   animate();
52
53   </script>
54 </body>
55 </html>
```



Exercise 1 – Working with Cube Geometry

```
<script>
    var camera, scene, renderer;
    var geometry, material, mesh;

    var init = function () {
        renderer = new THREE.CanvasRenderer();
        renderer.setSize( window.innerWidth, window.innerHeight );
        document.body.appendChild( renderer.domElement );

        camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 1, 1000 );
        camera.position.z = 500;

        scene = new THREE.Scene();
        geometry = new THREE.CubeGeometry( 200, 200, 200 );
        material = new THREE.MeshBasicMaterial( { color: 0x000000} );
        mesh = new THREE.Mesh( geometry, material );
        scene.add( mesh );
    }

    var animate = function () {
        requestAnimationFrame( animate );
        mesh.rotation.x = Date.now() * 0.0005;
        mesh.rotation.y = Date.now() * 0.001;
        renderer.render( scene, camera );
    }

    init();
    animate();
</script>
```

Try changing the dimensions of the cube

Try changing the colour of the cube

Try changing the rotation speed of the cube

Exercise 2 – Adding different geometries

```
<script>
    var camera, scene, renderer;
    var geometry, material, mesh;

    var init = function () {
        renderer = new THREE.CanvasRenderer();
        renderer.setSize( window.innerWidth, window.innerHeight );
        document.body.appendChild( renderer.domElement );

        camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 1, 1000 );
        camera.position.z = 500;

        scene = new THREE.Scene();
        geometry = new THREE.SphereGeometry( 100, 100, 100 );
        material = new THREE.MeshBasicMaterial( { color: 0x000000} );
        mesh = new THREE.Mesh( geometry, material );
        scene.add( mesh );
    }

    var animate = function () {
        requestAnimationFrame( animate );
        mesh.rotation.x = Date.now() * 0.0005; ← Try changing the rotation speed of the sphere
        mesh.rotation.y = Date.now() * 0.001;
        renderer.render( scene, camera );
    }

    init();
    animate();
</script>
```

Sphere Geometry

Try changing the dimensions of the sphere

Try changing the colour of the sphere

Try changing the rotation speed of the sphere

Exercise 3 – Adding different geometries

- ▶ Try adding different geometries, such as
 - Cylinder
 - Ring
 - Octahedron
 - Torus
 - Others

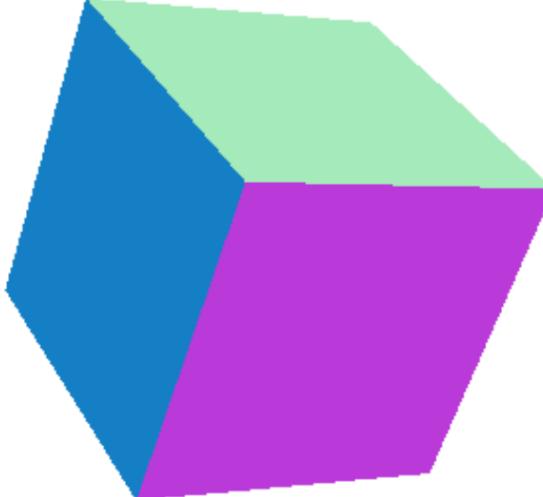
- ▶ The syntax of the geometries have been provided
http://threejs.org/docs/index.html#Manual/Introduction/Creating_a_scene

Future Learning – LiveCodeLab 2.0

- ▶ Live coding and lots of cool demonstrations

[LiveCodeLab 2.0](#) [Demos](#) [Tutorials](#) [Autocode](#) [Hide Code: off](#) [Reset](#)

```
// there you go!
// a simple cube!
//
background white
rotate 0, time/2,time/2
box
```



More details: <http://livecodelab.net/play/index.html#bookmark=simpleCubeDemo>

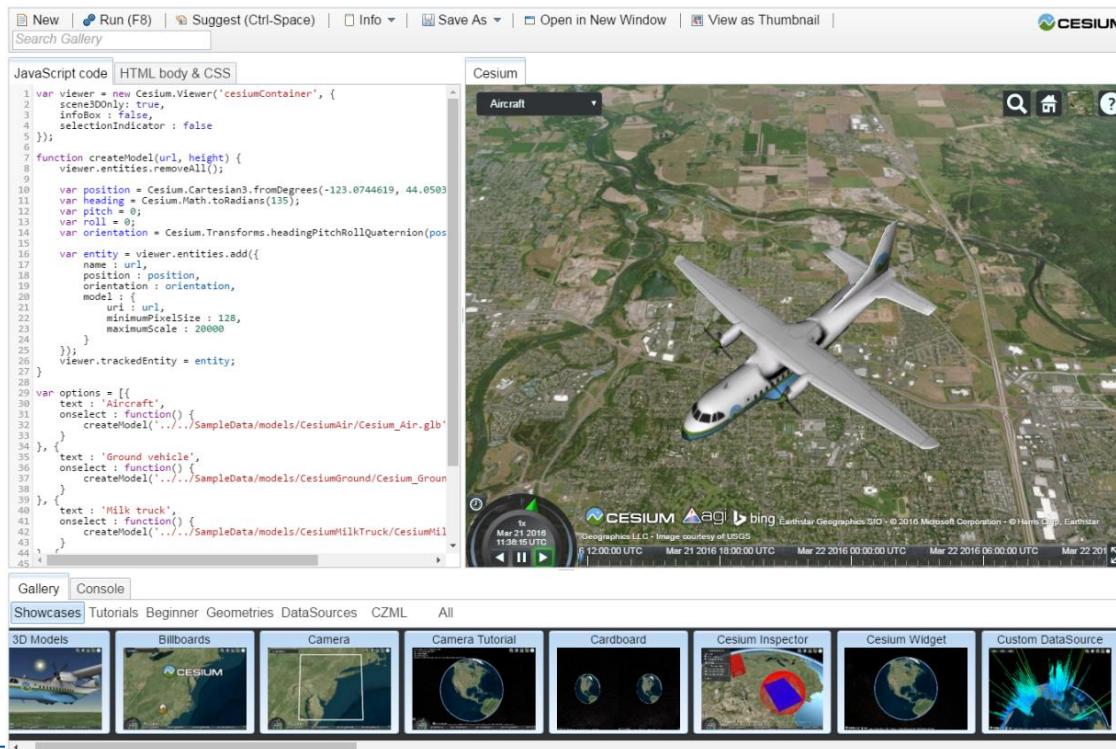


Session 4:

Cesium Hands-on

Cesium Sandcastle – Live Coding

- ▶ Suitable for beginners
 - Easy JavaScript coding
 - Results are visible instantly
- ▶ Sandcastle is available on Cesium server
 - No installation required



Installing Cesium

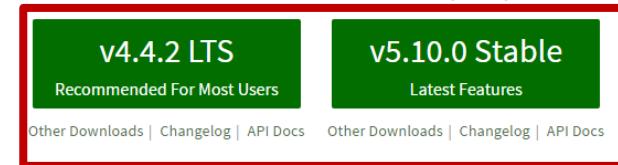
- ▶ For own application development, it is recommended to use a local installation of Cesium
- ▶ Step 1: Install node.js



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Important security notification regarding npm

Download for Windows (x64)



Or have a look at the LTS schedule.

More details: <https://nodejs.org/en/>

Installing Cesium

► Step 2: Download latest version of Cesium

<https://cesiumjs.org/downloads.html>

The screenshot shows the 'Cesium Downloads' page. At the top, there's a green button labeled 'DOWNLOAD CESIUM 1.25' with a download icon. Below it, text says '42.35 MB | 2016-09-01'. A red box highlights this button. Further down, there's a list of contents included in each release, followed by instructions for installing with npm, and a table of all versions.

Cesium is released monthly.

DOWNLOAD CESIUM 1.25
42.35 MB | 2016-09-01

Each Cesium release contains:

- The HTML reference documentation, located in the Build/Documentation directory.
- Cesium.js and all required dependencies, located in the Build/Cesium directory.
- [Asynchronous Module Definition \(AMD\)](#) modules, located in Source.
- HelloWorld.html, the simplest possible Cesium application, located in Apps/HelloWorld.html.
- Cesium Viewer, a simple reference application, at Apps/CesiumViewer.
- Cesium Sandcastle, a live code editor and example gallery, at Apps/Sandcastle.
- A Node.js-based development server for getting up and running quickly.

Install with npm

You can also install Cesium with [npm](#).

```
$ npm install cesium
```

The Cesium npm module includes everything you need for app development.

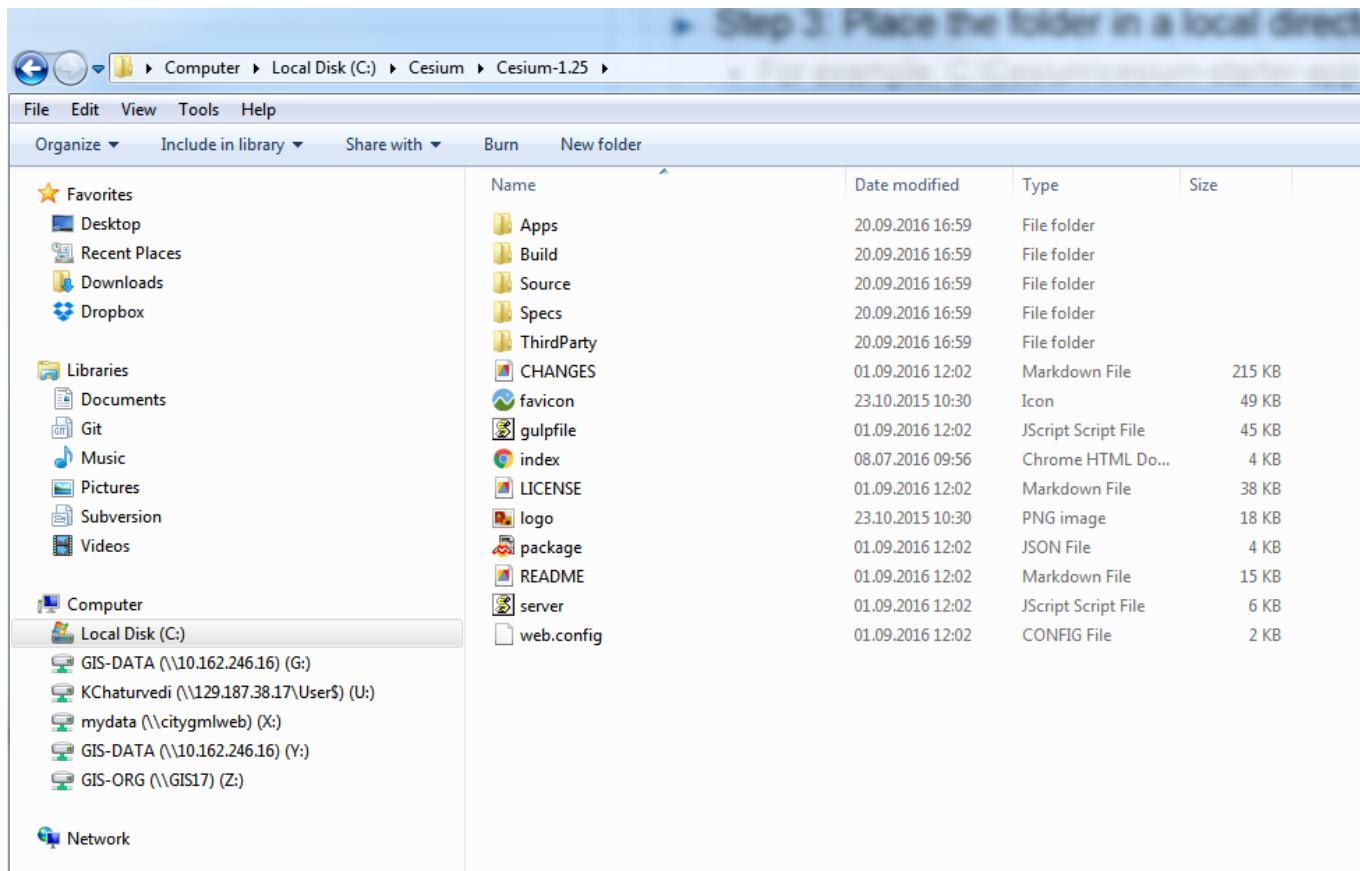
- **Source** - A collection of modules that can be used with any AMD-aware loaders, such as requirejs, webpack, and browserify. This is our preferred development method and ideal for those who have embraced module-based web development.
- **Build/Cesium** - Combined and minified Cesium.js and associated files, suitable for deploying an application.
- **Build/CesiumUnminified** - Combined and unminified Cesium.js and associated files, along with extra error checking and validation, suitable for development.

All Versions

Version	Download	Date	Notes
1.25	Cesium-1.25.zip (42.35 MB)	2016-09-01	<ul style="list-style-type: none">• Blog post• Source• Documentation, Code Examples, Tests, etc.

Installing Cesium

- ▶ Step 3: Place the folder in a local directory and unzip
 - For example, C:\Cesium\Cesium-1.25



Running Cesium (under Windows)

- ▶ Step 1: Open command prompt
- ▶ Step 2: Go to C:\Cesium\Cesium-1.25
- ▶ Run
 - npm install

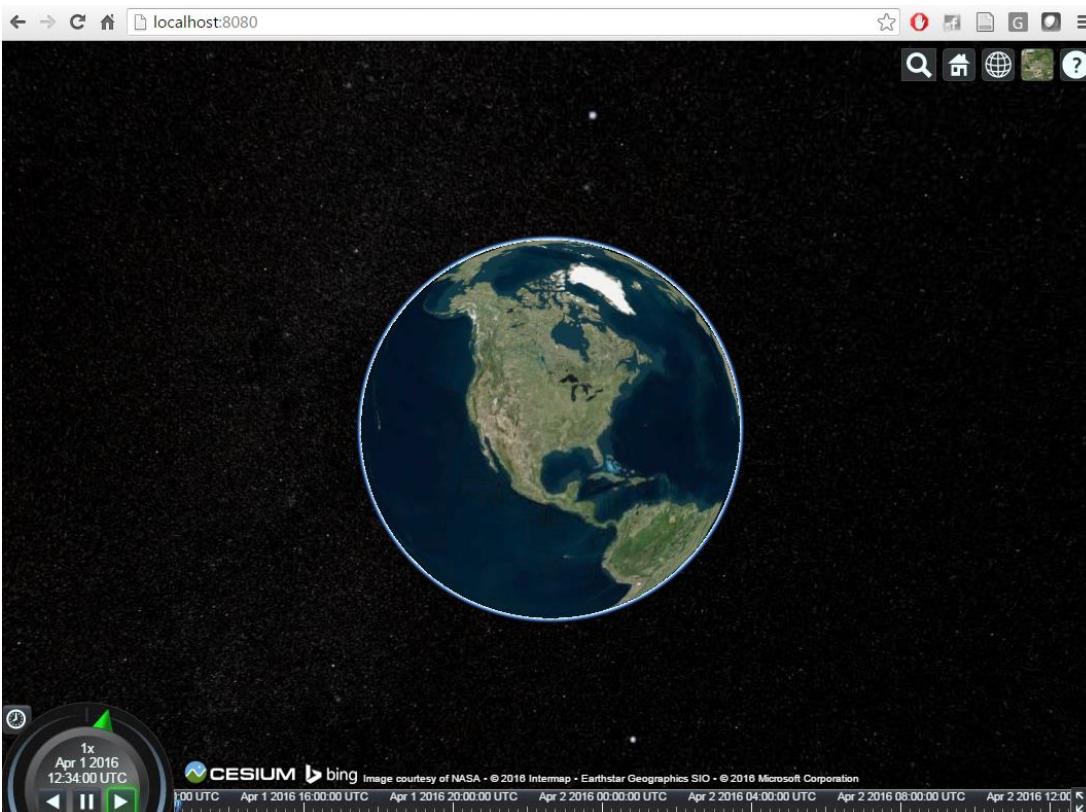
```
C:\Windows\system32\cmd.exe
C:\Cesium\Cesium-1.25>npm install
```

- node server.js

```
C:\Windows\system32\cmd.exe
C:\Cesium\Cesium-1.25>node server.js
```

Running Cesium

- ▶ Step 4: Open a web browser
 - Go to <http://localhost:8080> and select HelloWorld.html



- ▶ Cesium is up and running!

Working with examples

- ▶ Retrieve folders *SampleData* and *SampleScripts* from the location
www.3dcitydb.org/3dcitydb/fileadmin/TUM_Workshop/
- ▶ Copy folders *SampleData* and *SampleScripts* in the Cesium Folder
C:\Cesium\cesium-starter-app-master

Name	Date modified	Type	Size
.settings	23.03.2016 15:23	File folder	
node_modules	23.03.2016 15:24	File folder	
SampleData	01.04.2016 14:14	File folder	
SampleScripts	01.04.2016 14:21	File folder	
Source	23.03.2016 15:23	File folder	
ThirdParty	23.03.2016 15:23	File folder	
	23.04.2015 01:58	Text Document	1 KB
.project	23.04.2015 01:58	PROJECT File	1 KB
index	23.04.2015 01:58	Chrome HTML Do...	1 KB
LICENSE	23.04.2015 01:58	MD File	2 KB
package.json	23.04.2015 01:58	FMEFormatFile.F...	1 KB
Procfile	23.04.2015 01:58	File	1 KB
README	23.04.2015 01:58	MD File	6 KB
server	23.04.2015 01:58	JScript Script File	6 KB

- SampleScripts contains example HTML files for test purposes
- SampleData contains sample data files for loading

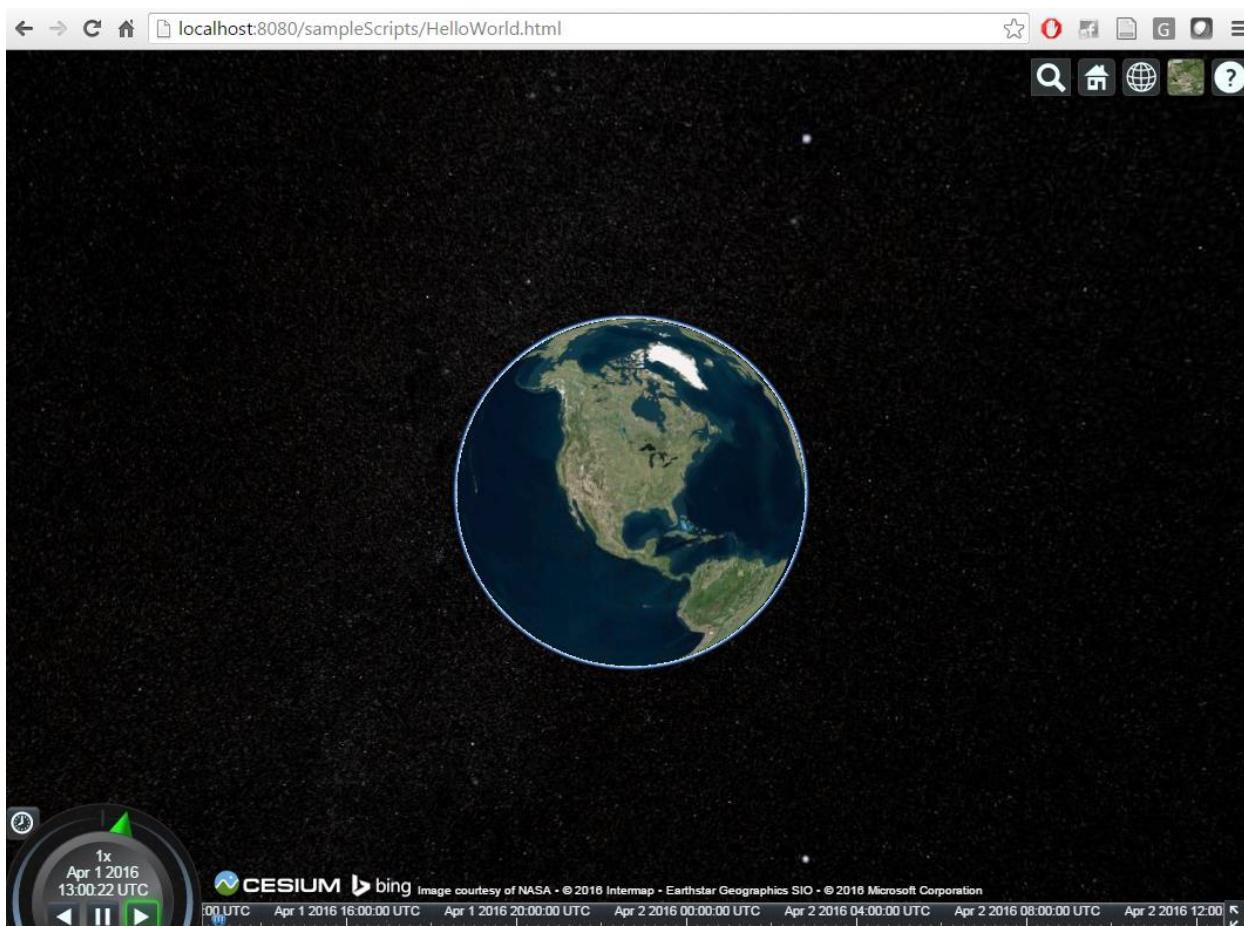
Basic Hello World Example

► HelloWorld.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Hello World</title>
  <script src="../ThirdParty/Cesium/Cesium.js"></script> ← Calling Cesium JavaScript
  <style>
    @import url(..../ThirdParty/Cesium/Widgets/widgets.css);
    html, body, #cesiumContainer {
      width: 100%; height: 100%; margin: 0; padding: 0; overflow: hidden;
    }
  </style>
</head>
<body>
  <div id="cesiumContainer"></div>
  <script>
    var viewer = new Cesium.Viewer('cesiumContainer'); ← Enabling Cesium Viewer
  </script>
</body>
</html>
```

Basic Hello World Example

- ▶ Open HelloWorld.html in the browser,
 - E.g., type <http://localhost:8080/sampleScripts>HelloWorld.html>



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Adding simple geometry</title>
<script src="../ThirdParty/Cesium/Cesium.js"></script>
<style>
@import url(..ThirdParty/Cesium/Widgets/widgets.css);
html, body, #cesiumContainer {
    width: 100%; height: 100%; margin: 0; padding: 0; overflow: hidden;
}
</style>
</head>
<body>
<div id="cesiumContainer"></div>
<script>

    var viewer = new Cesium.Viewer('cesiumContainer', { infoBox : false });
    var entities = viewer.entities;

    var stripeMaterial = new Cesium.StripeMaterialProperty({
        evenColor : Cesium.Color.WHITE.withAlpha(0.5),
        oddColor : Cesium.Color.BLUE.withAlpha(0.5),
        repeat : 5.0
    });

    entities.add({
        rectangle : {
            coordinates : Cesium.Rectangle.fromDegrees(-92.0, 20.0, -86.0, 27.0),
            outline : true,
            outlineColor : Cesium.Color.WHITE,
            outlineWidth : 4,
            stRotation : Cesium.Math.toRadians(45),
            material : stripeMaterial
        }
    });
    viewer.zoomTo(viewer.entities);

</script>
</body>
</html>
```

Adding simple geometries

AddingGeometry.html

Defining material

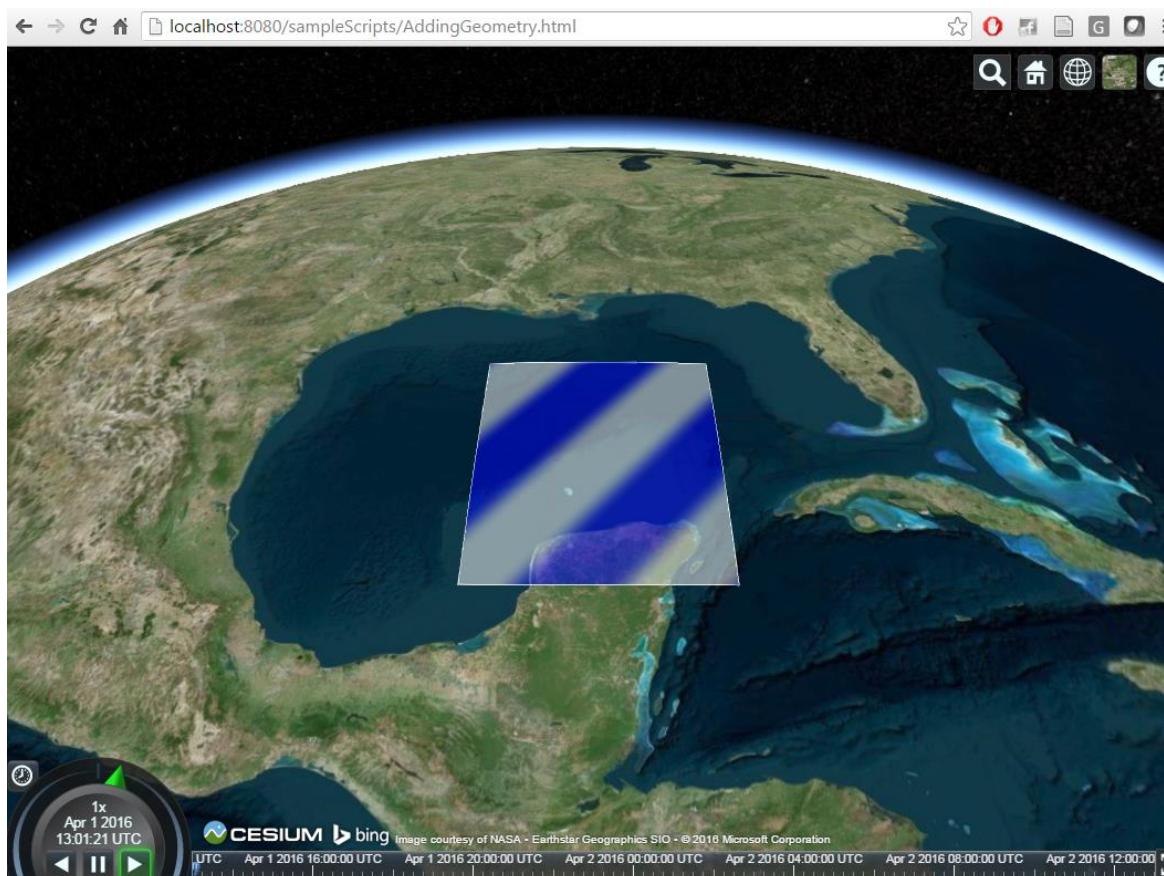


Creating simple rectangle



Adding simple geometries

- ▶ Open AddingGeometry.html in the browser,
 - E.g., type <http://localhost:8080/sampleScripts/AddingGeometry.html>



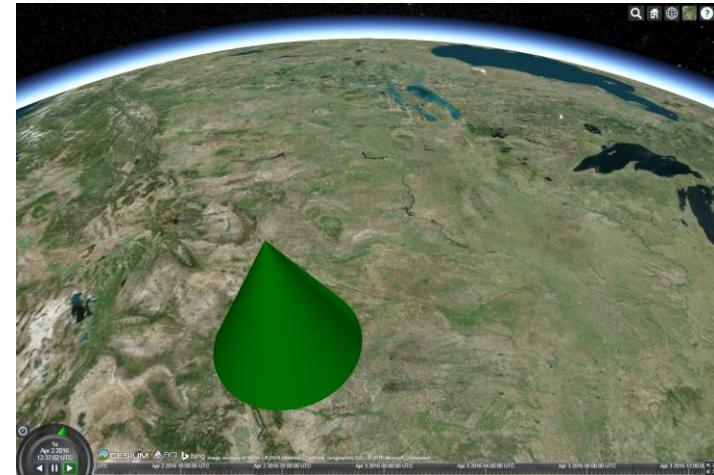
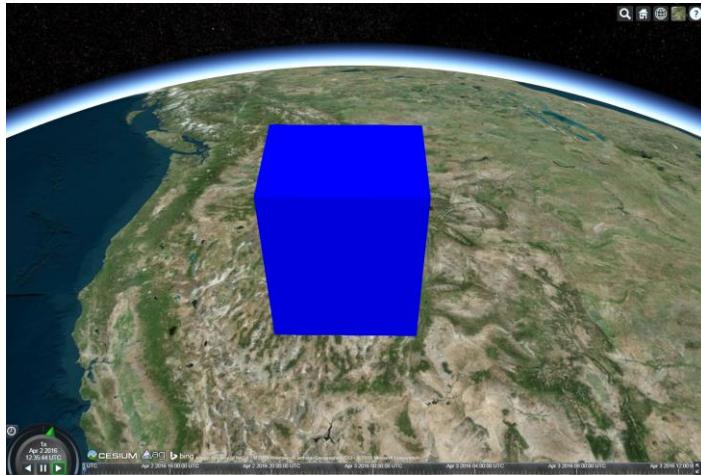
Exercise

► Exercise 1:

- Change the dimensions and colour of the rectangle

► Exercise 2:

- Create a new HTML file ,AddingBox.html‘, having a Box geometry
- Create a new HTML file, AddingCone.html‘ having a cone geometry



```
<!DOCTYPE html>
.....
<body>
<div id="cesiumContainer"></div>
<script>
  var czml = [
    {
      "id" : "document",
      "name" : "CZML Point - Time Dynamic",
      "version" : "1.0"
    }, {
      "id" : "point",
      "availability" : "2012-08-04T16:00:00Z/2012-08-04T16:05:00Z",
      "position" : {
        "epoch" : "2012-08-04T16:00:00Z",
        "cartographicDegrees" : [
          0, -70, 20, 150000,
          100, -80, 44, 150000,
          200, -90, 18, 150000,
          300, -98, 52, 150000
        ]
      },
      "point" : {
        "color" : {
          "rgba" : [50, 0, 200, 255]
        },
        "outlineColor" : {
          "rgba" : [200, 0, 20, 20]
        },
        "pixelSize" : 15
      }
    }
  ];
  var viewer = new Cesium.Viewer('cesiumContainer');
  viewer.dataSources.add(Cesium.CzmlDataSource.load(czml));
</script>
</body>
</html>
```

Adding simple CZML

AddingCZML.html

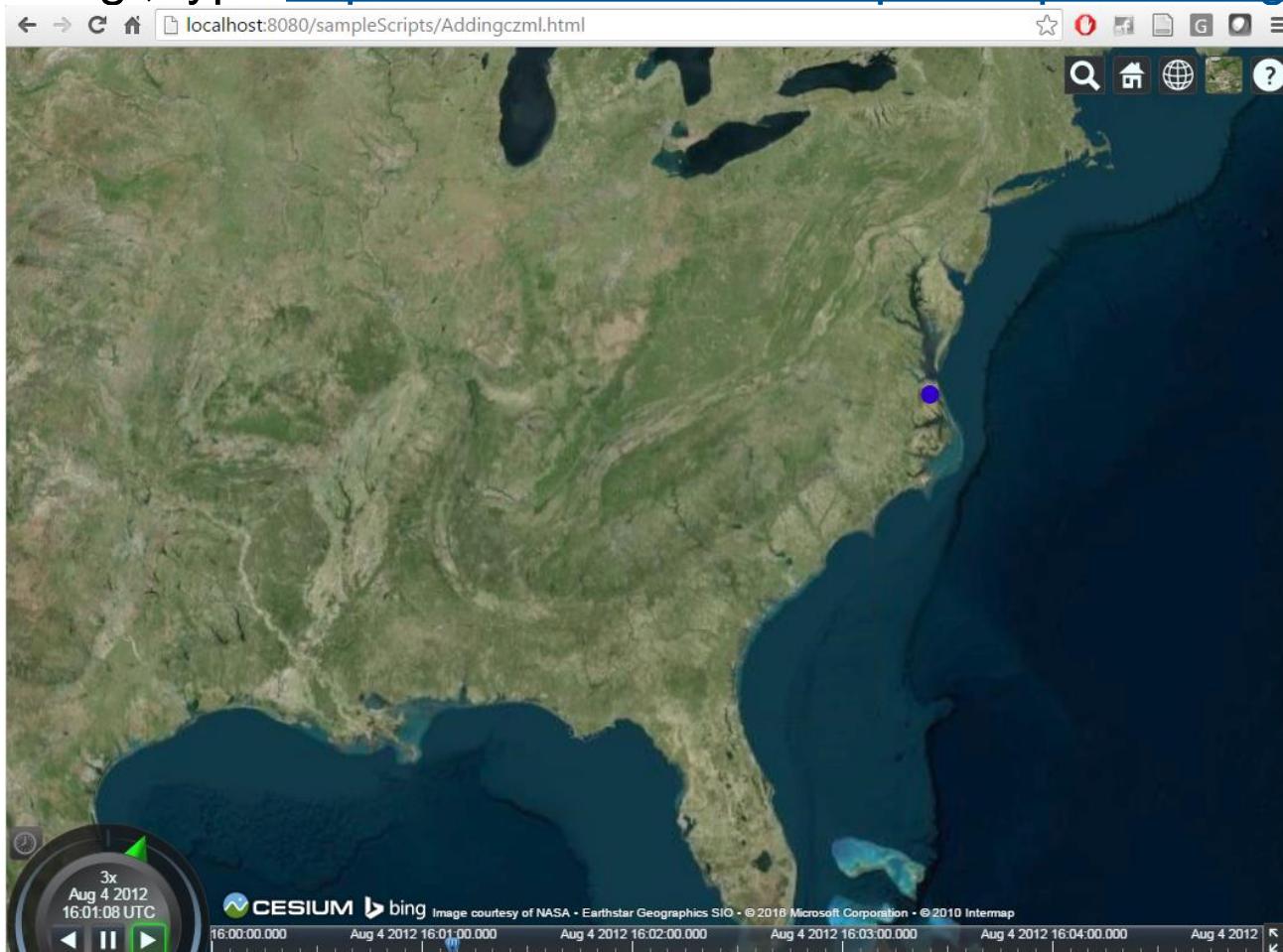


Defining CZML

Adding CZML source

Adding simple CZML

- ▶ Open AddingCZML.html in the browser,
 - E.g., type <http://localhost:8080/sampleScripts/Addingczml.html>



Exercise

- ▶ Can you try adding one more dynamic point moving in parallel to this point?

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Adding KML</title>
<script src="..ThirdParty/Cesium/Cesium.js"></script>
<style>
    @import url(..ThirdParty/Cesium/Widgets/widgets.css);
    html, body, #cesiumContainer {
        width: 100%; height: 100%; margin: 0; padding: 0; overflow: hidden;
    }
</style>
</head>
<body>
    <div id="cesiumContainer"></div>
    <script>

        var viewer = new Cesium.Viewer('cesiumContainer', { infoBox : false });
        var entities = viewer.entities;

        var options = {
            camera : viewer.scene.camera,
            canvas : viewer.scene.canvas
        };

        viewer.dataSources.add(Cesium.KmlDataSource.load('..SampleData/bikeRide.kml', options)).then(function(dataSource){
            viewer.clock.shouldAnimate = false;
            var rider = dataSource.entities.getById('tour');
            viewer.flyTo(rider).then(function(){
                viewer.trackedEntity = rider;
                viewer.selectedEntity = viewer.trackedEntity;
                viewer.clock.multiplier = 30;
                viewer.clock.shouldAnimate = true;
            });
        });

    </script>
</body>
</html>
```

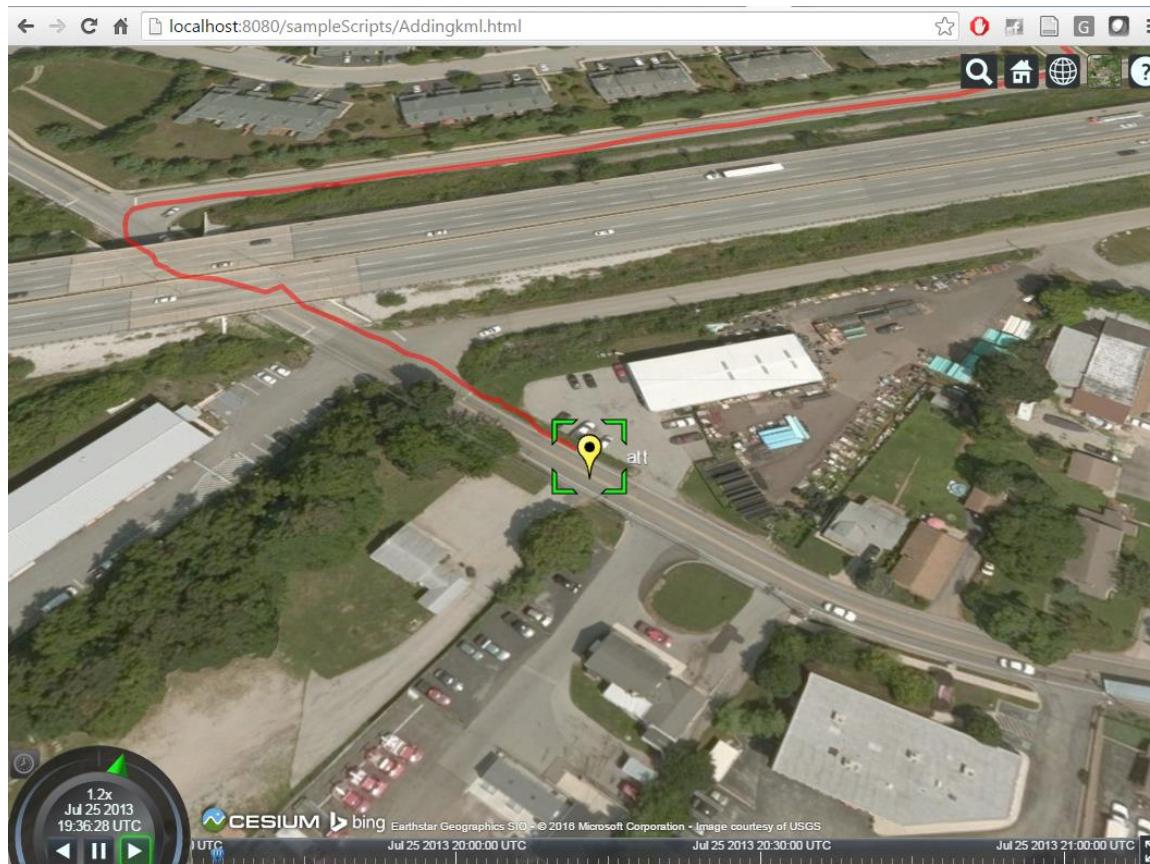
Adding Datasource - KML

AddingKML.html

Adding KML source

Adding Datasource - KML

- ▶ Open AddingKML.html in the browser,
 - E.g., type <http://localhost:8080/sampleScripts/AddingKML.html>



Simple COLLADA Building stored as KMZ file

► BuildingGeometry.html

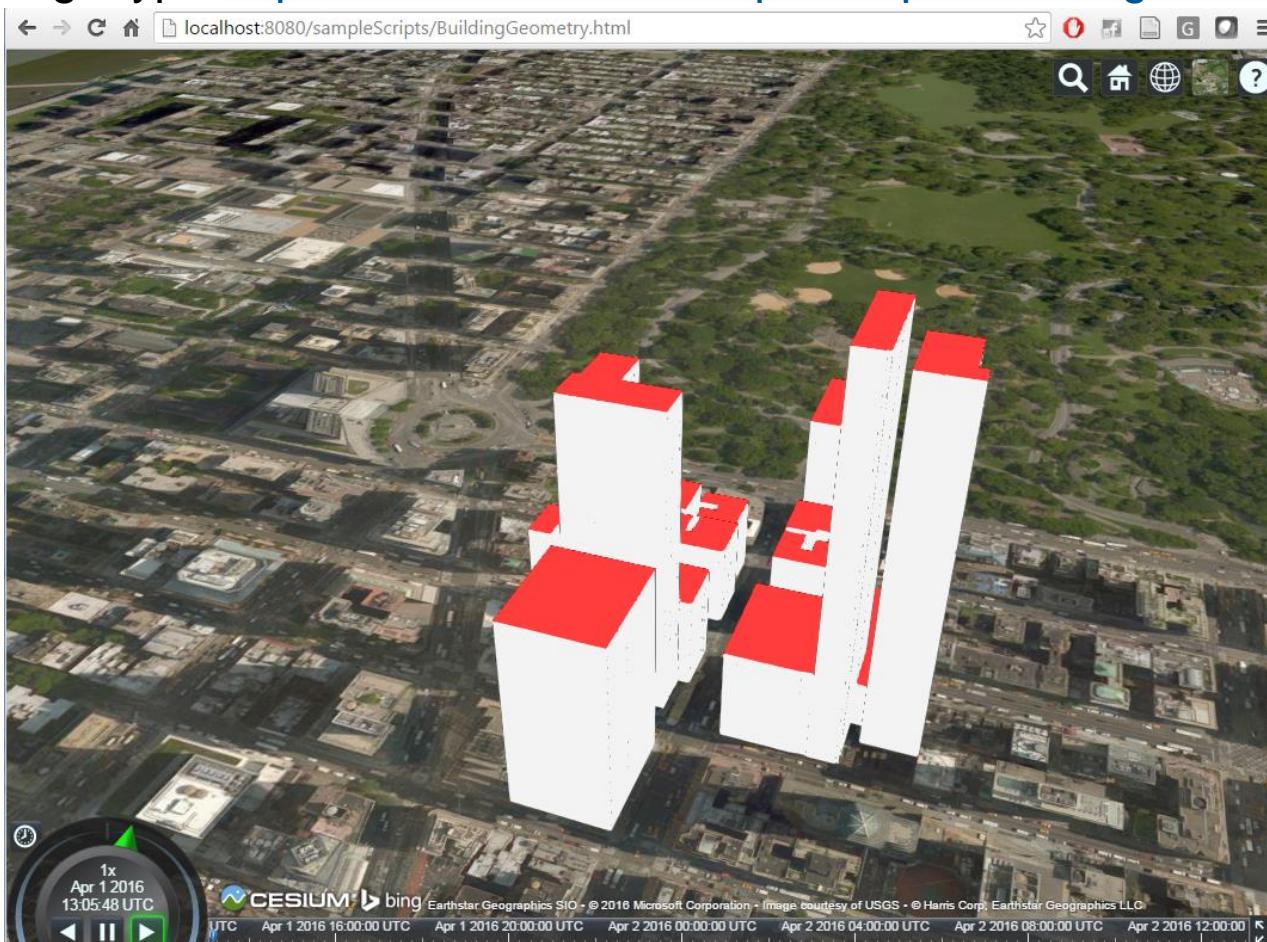
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Adding KMZ</title>
  <script src="../ThirdParty/Cesium/Cesium.js"></script>
  <style>
    @import url('../ThirdParty/Cesium/Widgets/widgets.css');
    html, body, #cesiumContainer {
      width: 100%; height: 100%; margin: 0; padding: 0; overflow: hidden;
    }
  </style>
</head>
<body>
  <div id="cesiumContainer"></div>

  <script>
    var viewer = new Cesium.Viewer('cesiumContainer');

    var promise = viewer.dataSources.add(Cesium.KmlDataSource.load('../SampleData/Building_test_Tile_1_1_geometry.kmz'));
    viewer.flyTo(promise);
  </script>
</body>
</html>
```

Simple COLLADA Building stored as KMZ file

- ▶ Open BuildingGeometry.html in the browser,
 - E.g., type <http://localhost:8080/sampleScripts/BuildingGeometry.html>



Exercise

- ▶ Create a new HTML file ,AddingKMZ.html‘, having a Box geometry
 - Load KMZ File *Building_test_Tile_0_0_geometry.kmz* in a similar manner