I’m Patrick Cozzi, and I’m a developer at Analytical Graphics, Inc. Today I’m going to talk about Cesium – an open-source JavaScript library for creating fast 3D maps.
Today we'll look at

• An overview of Cesium, highlighting its features most important to the geospatial community.

• We'll look at several live demos. In general, if I could replace a slide with a demo, I did. All of these demos are also on the Cesium website, cesium.agi.com.

• Finally, we'll talk about the tools we use to develop Cesium, and lessons learned in starting to build an open-source community.
Let’s start with a demo.

This app, Powder Tracks, is a side-project by one of our developers, Greg Beatty. He went skiing in Utah, and used his phone to record GPS tracks for the routes he skied, and used a helmet-cam to record video.

This app uses Cesium to playback the routes he skied with synchronized video. Cesium provides

• High resolution terrain. Here the data is 10 meter from the National Elevation Dataset.
• High resolution imagery. Here the data is Bing Maps. Cesium provides access to many imagery servers like WMS, TMS, and OpenStreetMap.
• Vector data rendering for the route polylines and video placemarks.
• Picking – sometimes called selection – so the placemarks highlight on mouse over and we can fly to them when we click them.
• Rendering of the stars, sky, and sun.
• 3D camera control and the ability to fly the camera between locations.
• The time control widgets at the bottom of the screen.
Here 3D adds significant value over 2D since terrain is fundamental to understanding the data. In addition, it's just more fun and compelling than a 2D map.

We can also add a second imagery overlay to show the terrain slope. Since the blending is done on the video card, we can use this slider to blend between the slope layer and the base layer.

Finally, remember that Cesium is a JavaScript library – so this is all running in a web browser without an install or plugin.
Here’s another demo – EarthKAM Explorer.

A team of five of us literally coded this app in a weekend for the NASA International Space Apps Challenge – a hackathon ran by NASA with over 9,000 participants. The app took second place in the local Philadelphia judging and received an honorable mention for best use of data in the global judging.

EarthKAM stands for Earth Knowledge Acquired by Middle school Students, and it is a program that allows middle school students to signup for “missions” to take images from the International Space Station by requesting that the ISS’ digital camera take a picture of Earth at a certain time. The result is a dataset of georeferenced polygons with satellite imagery and other metadata like what time the image was taken.

This app uses Cesium to provide a 3D viewer for the EarthKAM datasets. It uses a lot of the same features as Powder Tracks. If we click on a polygon, we fly to it, and its image loads. We also see the ISS orbit so we can see where it took the picture from.

If we zoom to one of the polygons in the ocean, we see the ocean effects provided
by Cesium.

Although the focus of Cesium is on 3D. It also supports 2D with the same API. In fact, it is just one line of code to warp from 3D to 2D. 2D still has the same lighting and shading effects. We can also morph to what we call Columbus view, which is a 2.5D view. This gets interesting when we have objects whose altitudes vary as we'll see in later demos.
Before we dive into Cesium in more detail, I want to show one last demo on a very serious topic – Santa Claus.

Every Christmas Eve, NORAD – the North American Aerospace Defense Command – uses a combination of radar, infrared satellites, Santa Cams, and fighter jets to track Santa’s journey. I believe Rudolph’s red nose actually gives away their location.

For several years, NORAD used Google Earth to show Santa’s position in 3D. Last year, NORAD worked with us to use Cesium so users could have the 3D experience directly in their web browser. Like the previous two demos, this app uses Cesium’s streaming terrain and imagery, camera flights, and placemarks.

During the month of December, the site had a record-breaking 22.3 million unique visitor.
So how do these apps run in a web browser?

I’m a long-time C++ graphics developer using OpenGL. Three years ago if someone told me that I would be coding in JavaScript full-time today, I wouldn’t have believed them. However, WebGL – one of the HTML5 APIs – has brought OpenGL ES 2 – the graphics API used in most phones today – to JavaScript.

There have been previous attempts to bring 3D to the web, but now the timing is right. JavaScript engines continue to increase in performance, and OpenGL-ES-capable hardware is widely available on desktops and mobile.

One reason that WebGL apps can be so fast is that the heavy lifting – the actual 3D rendering – is pushed to the video card (GPU). In extreme cases – where an app is heavily GPU-bound – a WebGL app can run just about as fast as a native app. In Cesium, we go to great lengths to utilize the GPU to optimize performance. We also use other techniques like offline optimizations and pushing work to web workers.
If I had to sum up Cesium as concisely as possible, I would say it is an open-source JavaScript library for fast 3D maps.

In addition to supporting the standard static raster and vector data we encounter in the geospatial world, Cesium is optimized for visualizing dynamic data like the ski tracks, ISS orbits, and Santa we saw in the previous demos. We are also working on CZML, a JSON schema for dynamic data.

We want widespread adoption so we use the Apache 2.0 license, which means Cesium is free for commercial and non-commercial uses.

The images here show apps/engines built by AGI, VT MAK, and NICTA.
Cesium can do quite a bit, but I want to focus on the features most relevant to the geospatial community.

All of the examples I’m going to show are part of Cesium Sandcastle, which is a live code editor with examples for the major Cesium features. It’s pretty straightforward to copy and paste these examples into the Hello World apps that comes with Cesium.
Imagery tiles are requested based on the view parameters. Unlike 2D, not all tiles are at the same level-of-detail. Tiles in the distant are at a lower LOD.

Layering and image processing is done on the GPU.

Sandcastle example


Our current terrain dataset is derived from the Shuttle Radar Topography Mission (SRTM), which has 90-meter spacing between -60 and 60 degrees latitude, and the Global 30 Arc Second Elevation Data Set (GTOPO30), which has 1-kilometer spacing for the entire globe. It is organized as multi-resolution pyramid of height maps using the Tile Map Service (TMS) layout.

We put a lot of work into making the terrain fast and light on the CPU.

- View frustum culling, occlusion culling, and backface culling
- Asynchronous requests
- GPU reprojections
- Web worker transforms

We are teaching part of the Rendering Massive Virtual Worlds course at SIGGRAPH this year based on our experience here.

Sandcastle example –
http://cesium.agi.com/Cesium/Apps/Sandcastle/?src=Terrain.html

Tutorial - http://cesium.agi.com/2013/02/15/Cesium-Terrain-Tutorial/
We call placemarks billboards. We can draw 3D volumes like ellipsoids and are working on a general framework for drawing any type of geometry like cylinders, boxes, walls, etc.

Vector data uses GPU optimizations like batching and culling. Like all of Cesium, vector data has precision handling for large view distances (avoiding z-fighting) and large world coordinates (avoiding jitter). Styles can be applied using our material system. Vector data supports individual selection, what we call picking.

Sandcastle examples
- [http://cesium.agi.com/Cesium/Apps/Sandcastle/?src=Polylines.html](http://cesium.agi.com/Cesium/Apps/Sandcastle/?src=Polylines.html)
- [http://cesium.agi.com/Cesium/Apps/Sandcastle/?src=Polygons.html](http://cesium.agi.com/Cesium/Apps/Sandcastle/?src=Polygons.html)
Visualization without writing client-side code.

We’re also starting on client-side support for KML, and infrastructure to support other vector formats like GeoJSON and TopoJSON

Sandcastle example -
http://cesium.agi.com/Cesium/Apps/Sandcastle/?src=Simple%20CZML%20Demo.html
For the last part of this talk, let’s look at some of the tools we use to build Cesium and the development culture. These are things I would have been very interested in before we started this project.

JSHint runs as we type in Eclipse. We also have a JSHint build target that runs JSHint on the entire source for reviewing pull requests. JSHint is particularly useful for folks new to JavaScript.

Lines of code are without whitespace and comments. 346 engine files and 233 test files.

In addition to AMD deployment, we also support a single minified .js file.

We plan to use Travis to help automate pull requests, but we haven’t done so yet.
Open source since April 2012. In development since February 2011. Went open-source as soon as we could.

Google Summer of Code -

Open Source Culture

- Why do developers contribute
  - Users need features / bug fixes / platform fixes
- What do developers contribute
  - Plugin points, not core architecture (everything is welcome though)
- Legal
  - CLAs can take a long time to go through
  - We still prefer CLA over DCO

Contributors mainly come from users, so to get a strong contributor community, start by building a user community. We were too focused on contributors too early.

Most contributions come in the form of a plugin point. For example, we might get a contribution for a new imagery provider, but not a contribution for the imagery engine, other than bug fixes. Other examples
- Image filters, but not the post-processing framework
- Geometry types, but not the general batching and rendering engine

Like any development team, we generally see a short-term loss bringing a contributor up to speed. However, we see it as a long-term win. Good contributor’s guide, doc, etc. can minimize the effort. Maintaining it all is work.

A CLA from a big company can take six months.

Contributors often forget to signoff on a commit for the DCO.
We use the mailing list a lot. Even if we physically sit right next to each other, we use the mailing list for important discussions.

Same for code reviews. One thing better than code reviews (but not a replacement) is to have two developers working on the same feature. They catch more mistakes and learn the code better, which results in better shared ownership.

Tips for code reviews -

There is no release manager; instead, our community shares the responsibility. Any committer can create the release for a given month, and at any point, they can pass the responsibility to someone else, or someone else can ask for it. This spreads knowledge, avoids stratification, avoids a single point of failure, and is beautifully unstructured.

The Cesium website, cesium.agi.com, has all the demos I showed today, GitHub link, and everything else you need to get started. My contact info is also on this slide.

If you haven’t already, stop by the AGI tables where we are doing Cesium demos.

Finally, Cesium is being integrated into OpenLayers 3 to provide 3D. Learn more about that at Eric Lemoine and Tom Payne’s talk at 3pm today.