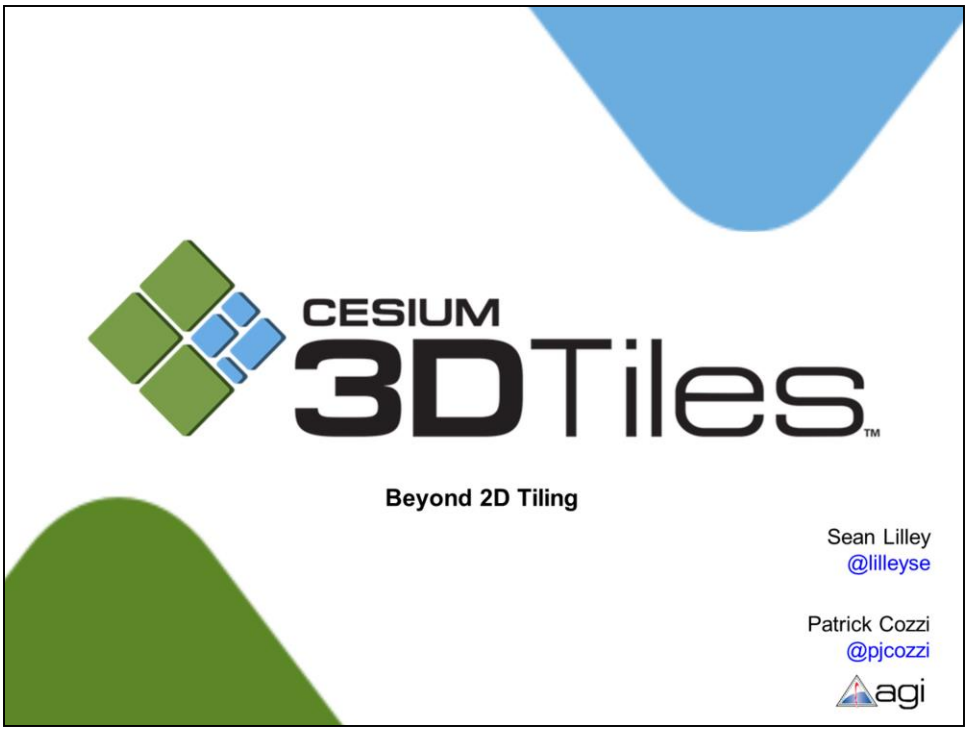



Welcome, thanks everyone for coming!

My name is Patrick Cozzi and this is Sean Lilley.  We are 3D developers at AGI, the company that started Cesium.

Today, we're going to talk about what I believe will be the biggest thing in 3D geospatial since Cesium, and will likely be bigger than Cesium itself: 3D Tiles.

We are at the start of a new era of streaming 3D geospatial data, and I hope this talk spurs your creativity.

I'll cover the motivation, the vision, and some use cases, and Sean will dive into the specifics as best as we can in 35 minutes.


AGI is a space and defense company. About five years ago, our customers had a need for a lightweight cross-platform virtual globe engine. At the time, WebGL was just emerging so we built a JavaScript library - Cesium - which is now actively developed as open source.

These screenshots show Cesium being used for space situational awareness as part of ComSpOC.

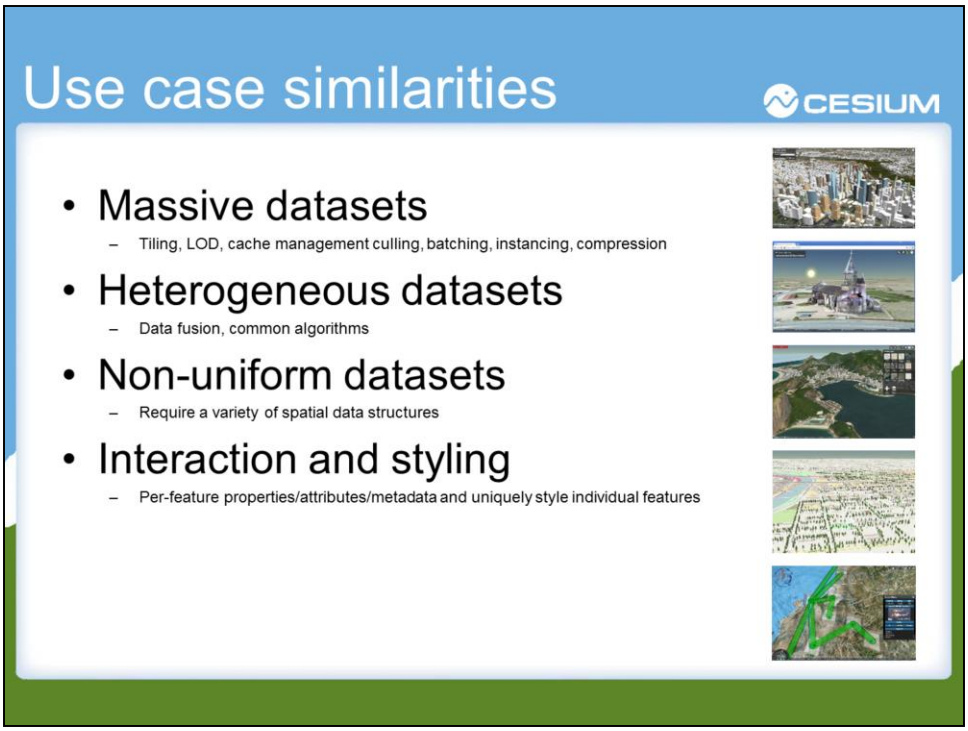
Cesium – [cesiumjs.org](cesiumjs.org)

ComSpOC - [comspoc.com](comspoc.com)


Being open source helped Cesium achieve wide-spread adoption across many industries with many use cases.

Cesium is now used for every building in Manhattan to every satellite in space...literally.

Cesium, of course, does the classic streaming of high-resolution terrain and imagery, which extends naturally enough from 2D. However, our users want to fully take advantage of 3D and stream millions of buildings, perhaps even textured buildings, millions or billions of points, models that can be instanced like trees, and fully 3D vector data, like corridors and polygons, polylines, and points that conform to terrain.




When we look at these use cases, some components are very different, but many things are similar. All of these use cases have massive datasets that are too big to fit into memory or download in their entirety so they require view-dependent streaming, with techniques such as:

* Tiling to create a spatial data structure for use with Hierarchical Level Of Detail (HLOD)

* These includes selecting the right LOD for a given view, efficiently culling tiles that are out of view, and managing a client-side tile cache.

* Features, e.g., individual models, should be batched together or instanced so they are rendered efficiently with WebGL.

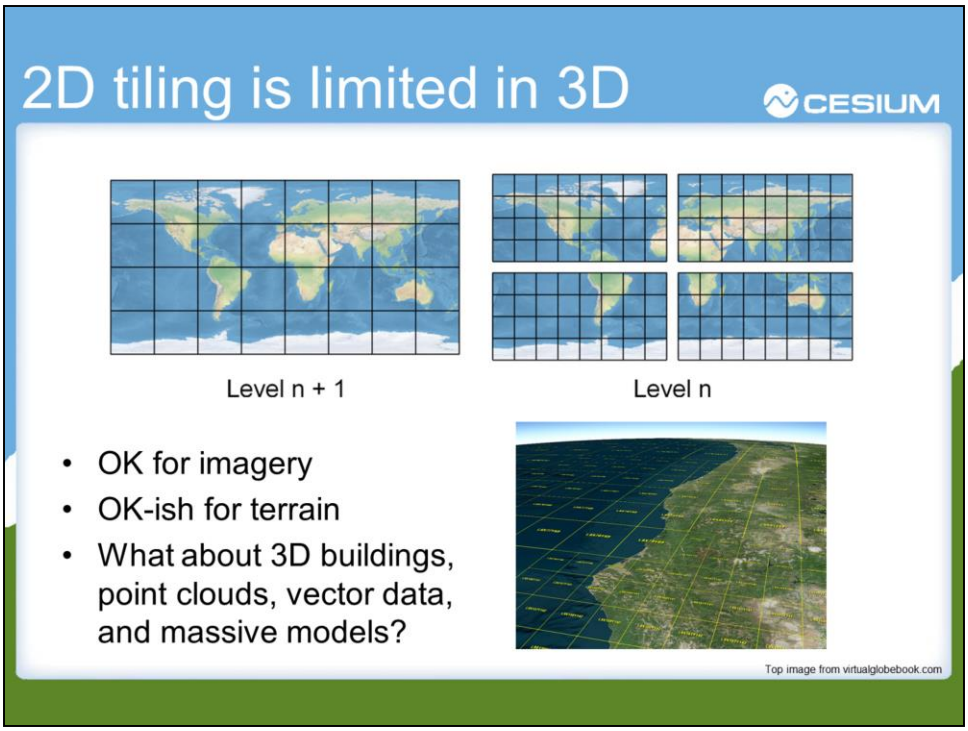
* And the tile payload should come across the wire compressed in such a way that it can be efficiently decompressed.

Many of these use cases have non-uniform data, e.g., buildings are not individually distributed throughout a city and not all 3D building models have the same number of triangles. So we would like the ability to support a variety of spatial data structures in a general manner.

With the exception of terrain and imagery, being able to interact with and style

individual features is key for most users.  We want to know what 3D building or tree is clicked, perhaps display per-feature metadata, and then style each feature based on the metadata.

These different types of data bring a lot of value when combined, e.g., buildings (batched models) and trees (instanced models) on terrain (TIN).



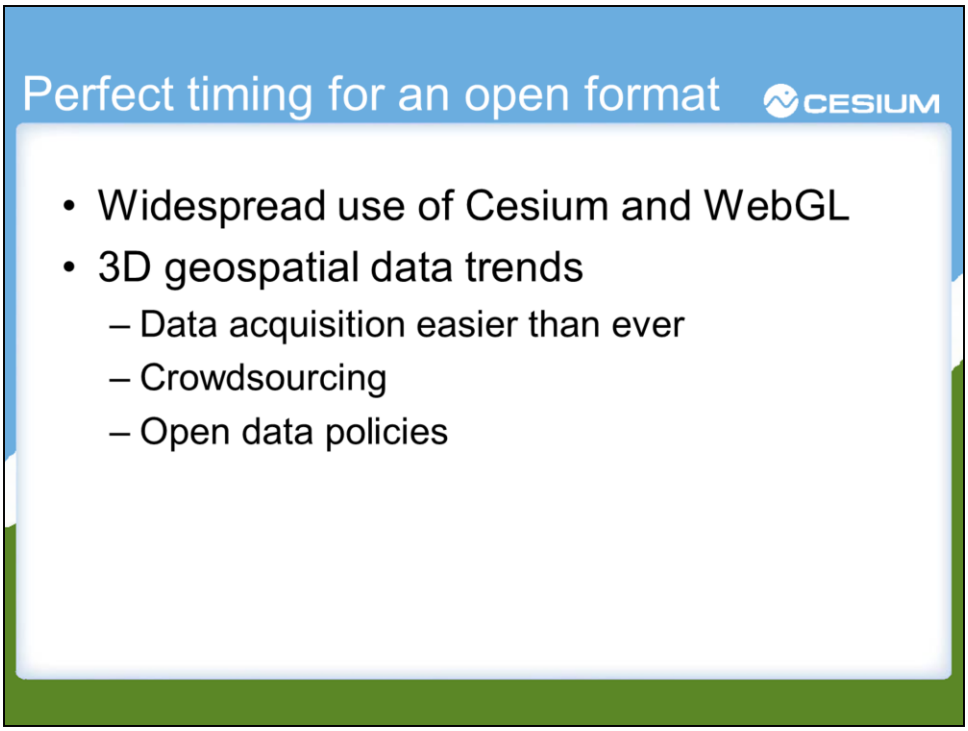

In 2D, tiling is usually done with a quadtree that is uniformly subdivided in longitude and latitude, e.g., TMS. This makes sense for imagery that is evenly distributed.

2D Level of detail is always a top-down view where only one level is visible at a time.

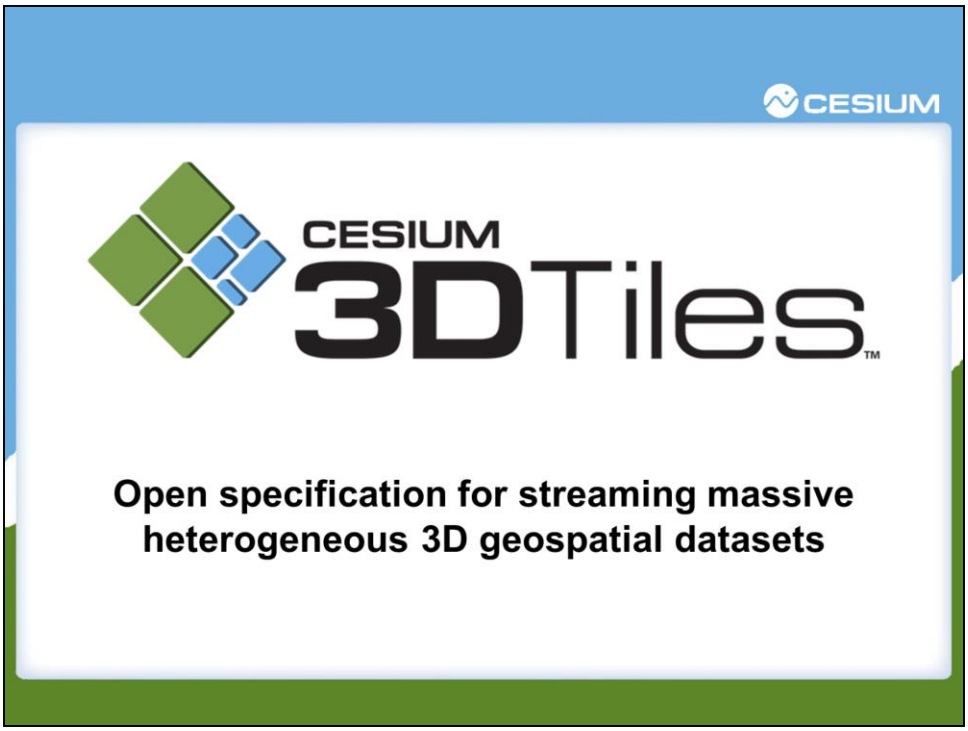
In 3D, this tiling scheme is OK for imagery and OK-ish for terrain, but

* Leads to sub-optimal subdivisions for non-uniform datasets
* Doesn't provide fully 3D subdivision like what is needed for point clouds
* Doesn't easily handle, for example, when a 3D building overlaps two tiles

In 3D, things also become complicated because multiple LODs can be visible in the same view; with higher LODs near the viewer and lower LODs farther away. We need an error metric to select the right tiles and ways to avoid visual artifacts, called cracking, where tiles from different LODs meet.




Given this need, the widespread use of Cesium, and some awesome 3D geospatial data trends, now is great timing for a new open format for these use cases.

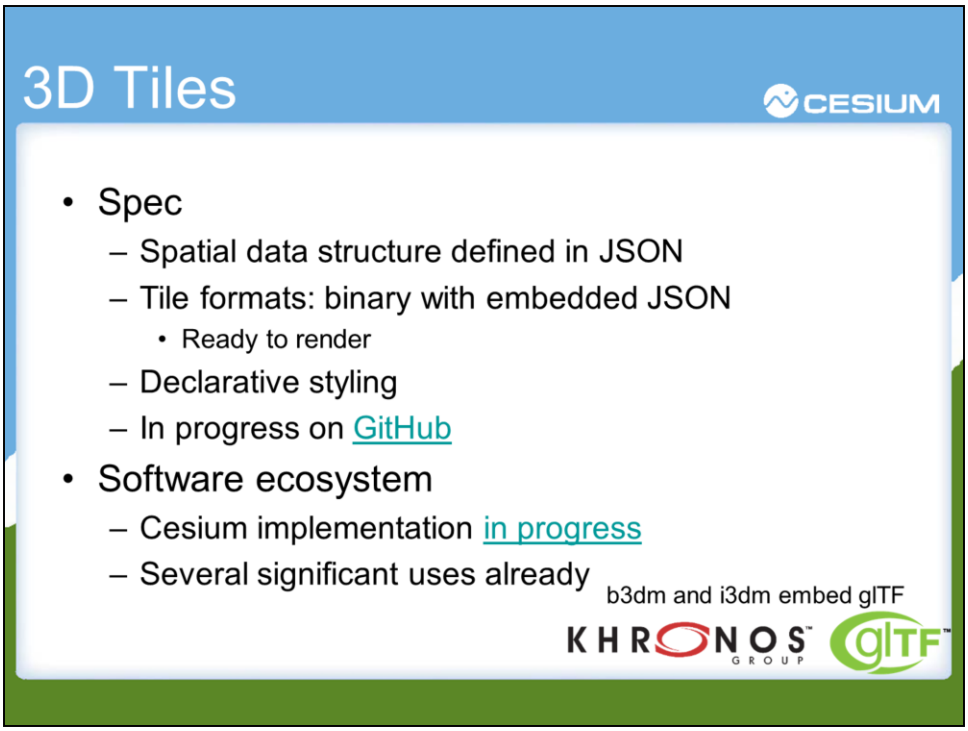



Enter 3D Tiles.

3D Tiles are an open specification for streaming massive heterogeneous 3D geospatial datasets. That is a mouthful, but each of those words is important. 3D Tiles provides the flexibility needed for the overlay but diverse 3D use cases.

I alluded to this at Web3D in 2014:
http://cesiumjs.org/presentations/CesiumCzmlGltf.pdf

And presented some of the technical approaches at MIT a bit later in 2014:
http://cesiumjs.org/presentations/RenderingMassiveGeospatialDatasetsInCesium.pdf




Concretely, what are 3D Tiles?

Two things: a spec and an open-source implementation in Cesium.

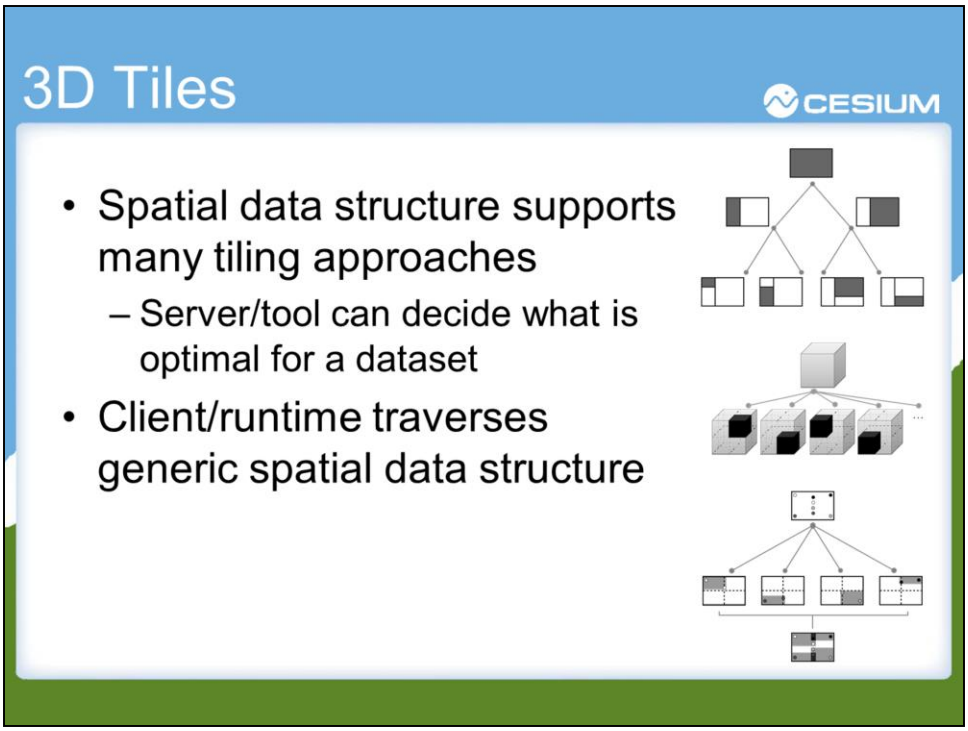
The spec is in Markdown on GitHub. It defines a spatial data structure in JSON. The spatial data structure defines how the tileset is organized.

It also defines tile payload formats for a few types of tiles such as batched 3D models and point clouds.

The 3D model tile formats use glTF, the GL Transmission Format, which is an open-standard from Khronos for runtime 3D models. Khronos is the group behind many graphics formats such as WebGL, OpenGL, and COLLADA. I contributed significantly to the glTF spec.

By utilizing glTF, we are able to take advantage of the glTF loader in Cesium and the growing ecosystem of open-source glTF pipeline tools.

Finally, the spec defines a declarative styling language, which is basically JavaScript expressions that have access to per-feature attributes.




Using techniques that are common in the movie and game industry, 3D Tiles define a general spatial data structure that can be used to create many different hierarchal data structures.

The key insight is that the client is generic and the same code can traverse any of these data structures because traversal relies on the property that a tile's children are inside the tile's bounding volume. This is called spatial coherence.

A converter/server can now tile up a dataset in a way that is optimal for that dataset. For example:

* In the top-right, there is a kd-tree with non-uniform splits, perhaps based on the cost of rendering features in each tile

* In the middle, there is an octree perhaps for rendering a point cloud

* In the bottom right, there is a loose non-uniform quadtree, which is a subdivision strategy we use for one type of 3D buildings dataset.

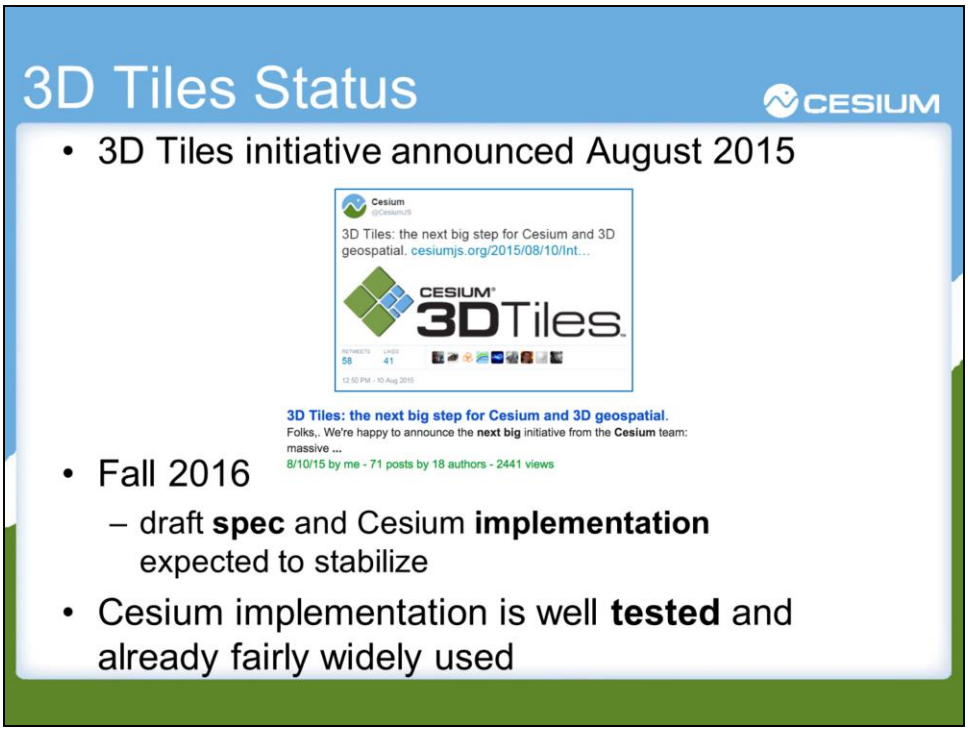
There is not a one-size-fits-all subdivision strategy; the converter/server does what it thinks is optimal for the dataset, then the client handles it in a general

fashion.


Here's an example of declarative styling. Just a few lines of code styles these buildings in Seattle. Each building's height maps to a color ramp where the tallest buildings are also translucent.

For illustration, it also only shows the buildings with an area greater than zero.

Finally, in the top right a building was highlighted on mouse over - that is not part of declarative styling, but is an important use case for interactivity.



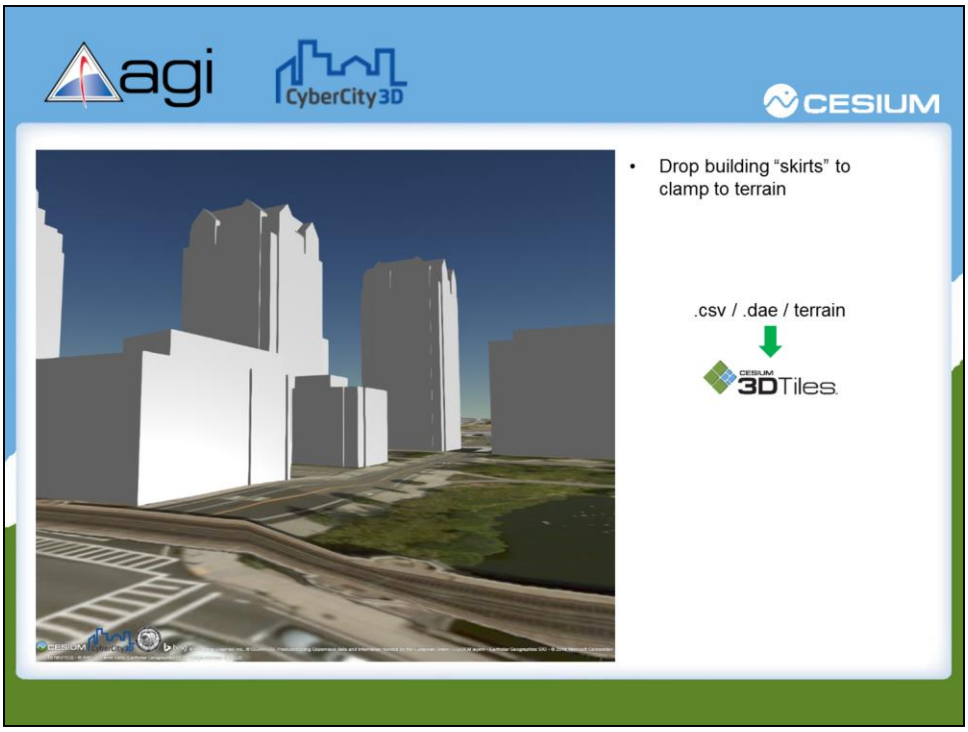

Cesium implementation is 1,172 commits ahead of master;

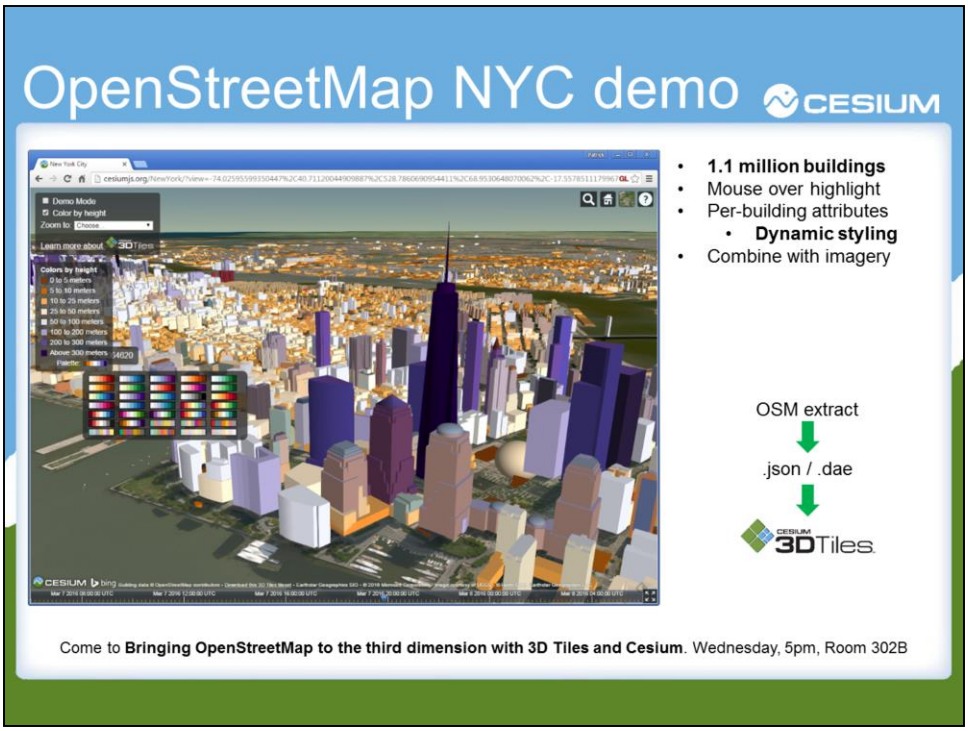
Still needs vector data, and to flush out point cloud and instanced tile formats.

See the roadmap:
https://github.com/AnalyticalGraphicsInc/cesium/issues/3241


Washington, DC: Over 135,000 structures

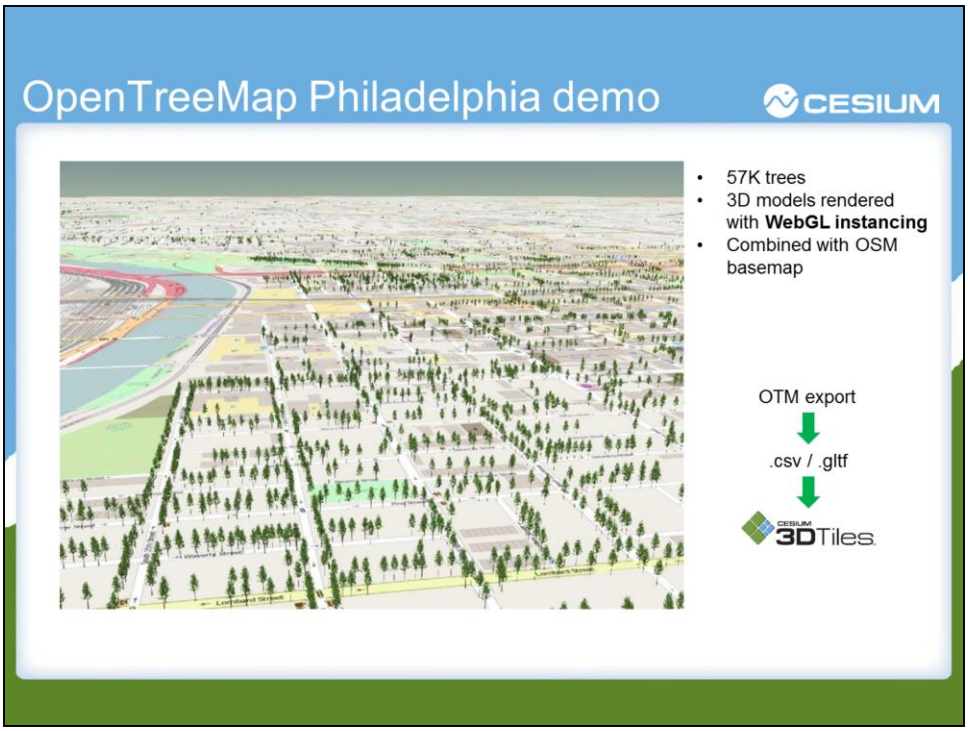
Use additive refinement so the “most important” buildings are streamed first and never replaced. Works well for a large number of simple models – would not work for complex models. 3D Tiles provides the flexibility to create an optimized data structure based on the input dataset.


agi
CyberCity3D
CESIUM
• Drop building “skirts” to clamp to terrain
.csv / .dae / terrain
CESIUM 3DTiles




Demo: http://cesiumjs.org/NewYork/

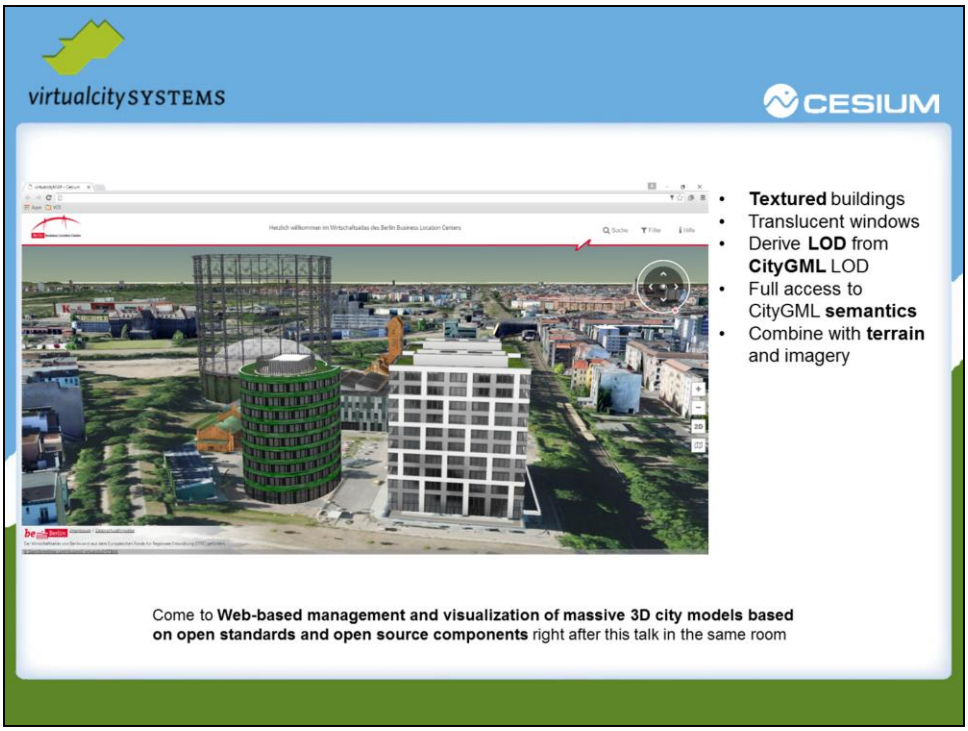
OSM extract from Mapzen: https://mapzen.com/data/metro-extracts/




57,714 trees using instanced tiles (i3dm).

Could also make a tileset where the leaf tiles are full 3D models (with instancing), and interior tiles are vector tiles representing the trees as imposters.

OpenTreeMap data: https://www.opentreemap.org/




~540K full textured LOD2 buildings.

Create a three-level tree (TMS levels 15-17) with different LODs and texture resolutions.

http://www.virtualcitysystems.de/en/

Screenshot: Berlin

Dataset:

Berlin 3D city model

- 539,481 fully textured LoD2 buildings
- 890 km² digital terrain model
- Approx. 370 LoD3 buidings and 3 LoD4 buildings (interior details)
- City trees (SolitaryVegetationObject), bridges, S-Bahn stations, …
- Data volume

16.6 GB CityGML file (XML unzipped)

19.4 GB texture files

3D Tiles / B3DM files for the entire model

Three TMS layers with different LoDs and texture resolutions

Level 15: LoD2 only and low-res textures (2.54 GB)

Level 16: LoD2 only and mid-res textures (4.22 GB)

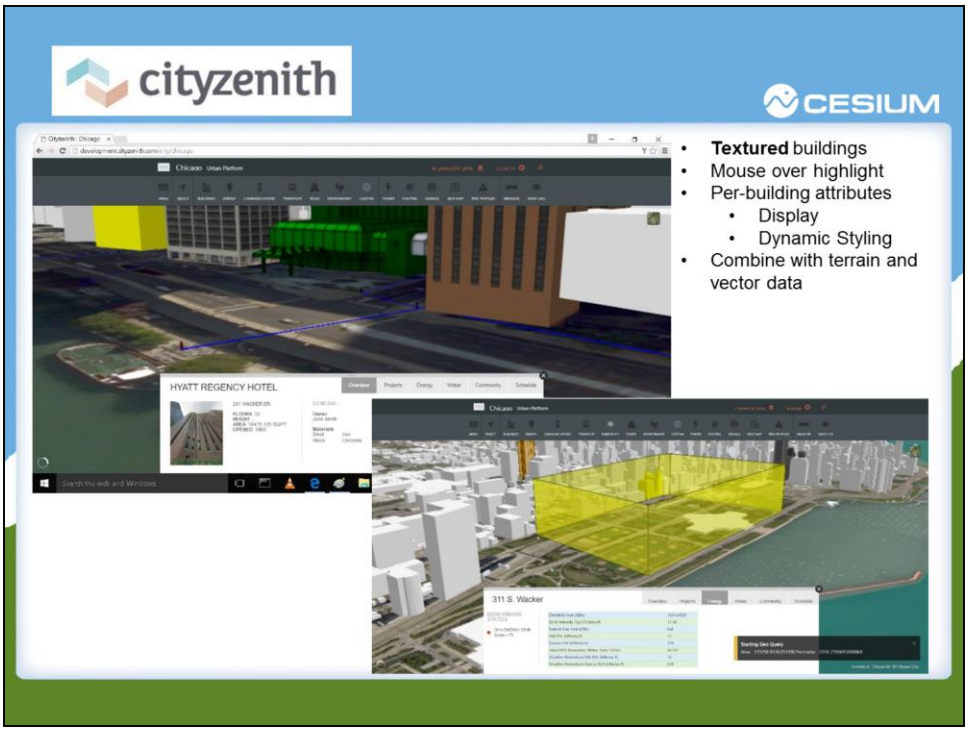
Level 17: LoD2-4 and high-res textures (19.5 GB)

Demo (with just open data):
http://hosting.virtualcitysystems.de/demos/berlin/vcm/index.html?lang=en


Top right: select wall surface of the CityGML building

Bottom left: show building CityGML attributes

Screenshot: Berlin




http://www.cityzenith.com/ - awesome video on their website

5D Smart CityFeatures Video: https://vimeo.com/157494960

Above: Chicago

Annotation and measurement tool using Cesium entities.


Separate tileset each for buildings, trees, windmills, etc.

Fraunhofer IGD:

- Institute for applied research in Visual Computing which focuses among others on the visualization and management of geoinformations
- Applications:
- Screenshot: Visualize plannings of e.g. windparks to local communities and to stakeholder of the infrastructure project
- Visualize huge city models
- Smart City plattforms, etc.

Problem:

- Currently the citymodels and all other objects are added as gltf models
    - Very basic (self implemented) streaming capabilities

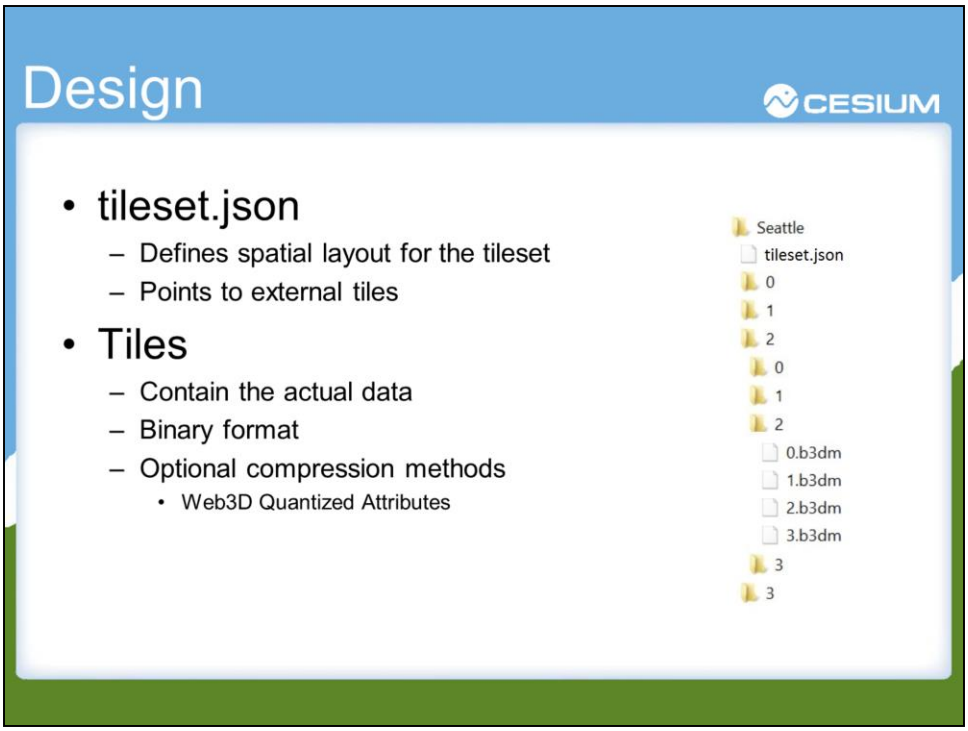
3D-Tiles is currently implemented to have:

- Proper streaming capabilities

- Access to metadata
- Instancing of objects (trees)
- Visualization pointclouds

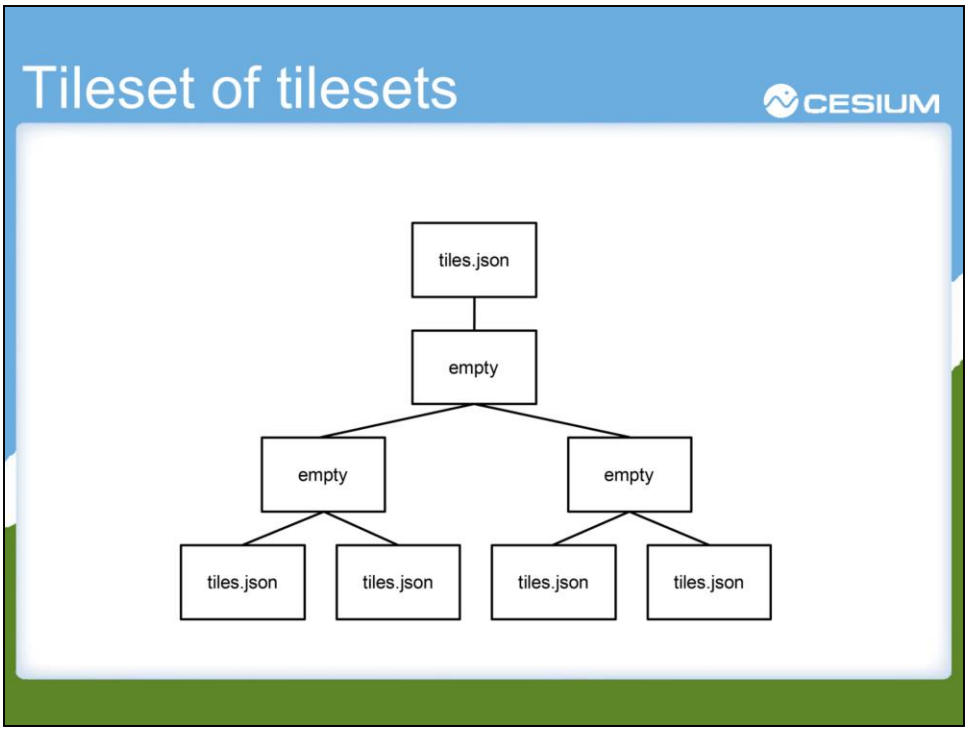

TIN

http://www.vricon.com/

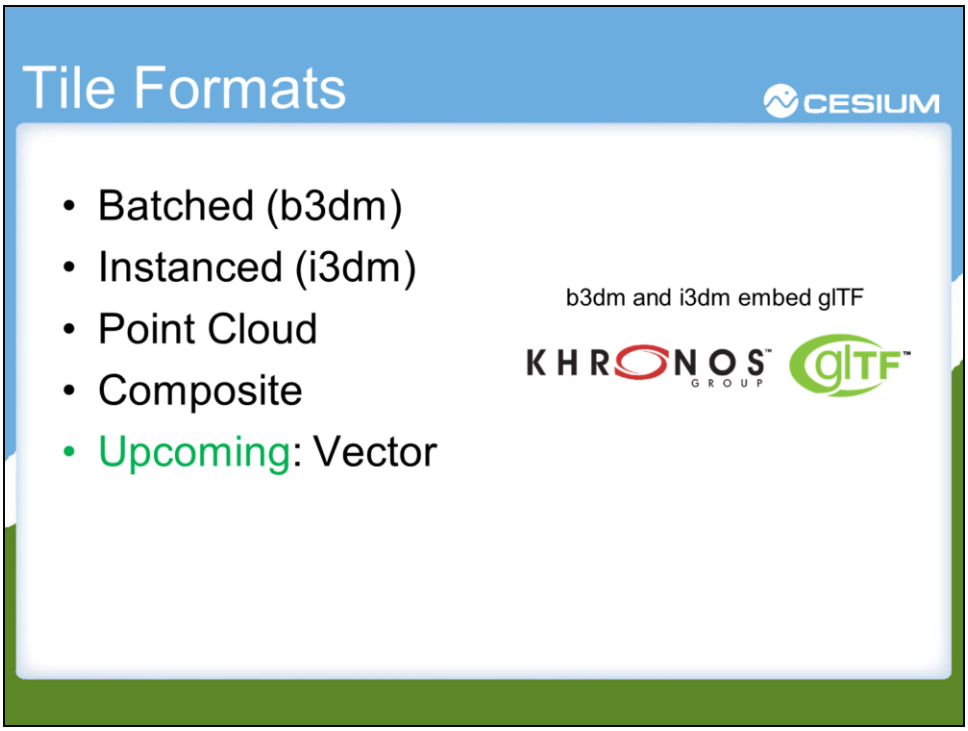



Now I'm going to go into more detail about the 3D Tiles format. The design is separated into two parts. First is the tileset.json file which defines the spatial layout of the tiles and points to the external tile files. The tiles themselves store the actual data and are represented as a binary format. We can also apply optional compression methods to the tiles, such as Web3D Quantized Attributes which can encode position data with fewer bits by storing the deltas relative to a center position. On the side is an example of what a tileset might look like as a folder structure. In this example, the tileset is for Seattle. The tileset.json references all the b3dm files, which are small portions of the city. And this is just one example, your tileset layouts may look different.

Other compression methods: Open3DGC, oct encoding




3D Tiles also supports a tileset of tilesets. If the root tile is your global data set, each smaller tileset.json might be a different city. The tilest is only streamed in once it is in view.



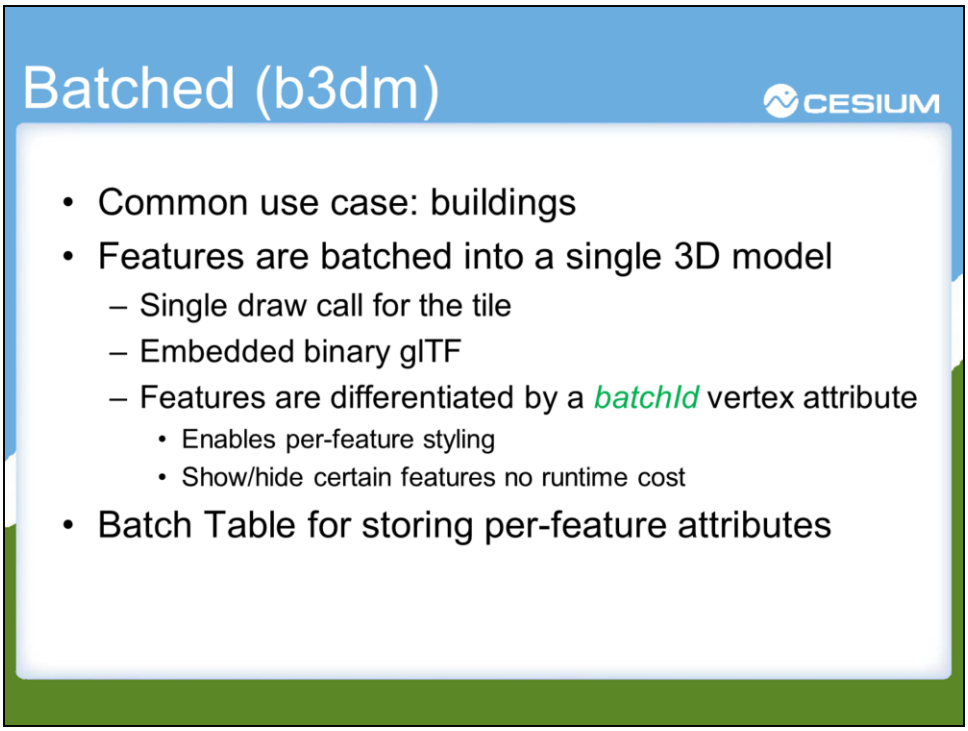

There are a variety of tile formats that are suitable for different types of data. I'll go through each of these individually. And just to note, a tileset will often contain a mix of these tile formats.

b3dm and i3dm embed binary glTF: https://github.com/KhronosGroup/glTF/blob/master/extensions/Khronos/KHR_binary_glTF/README.md

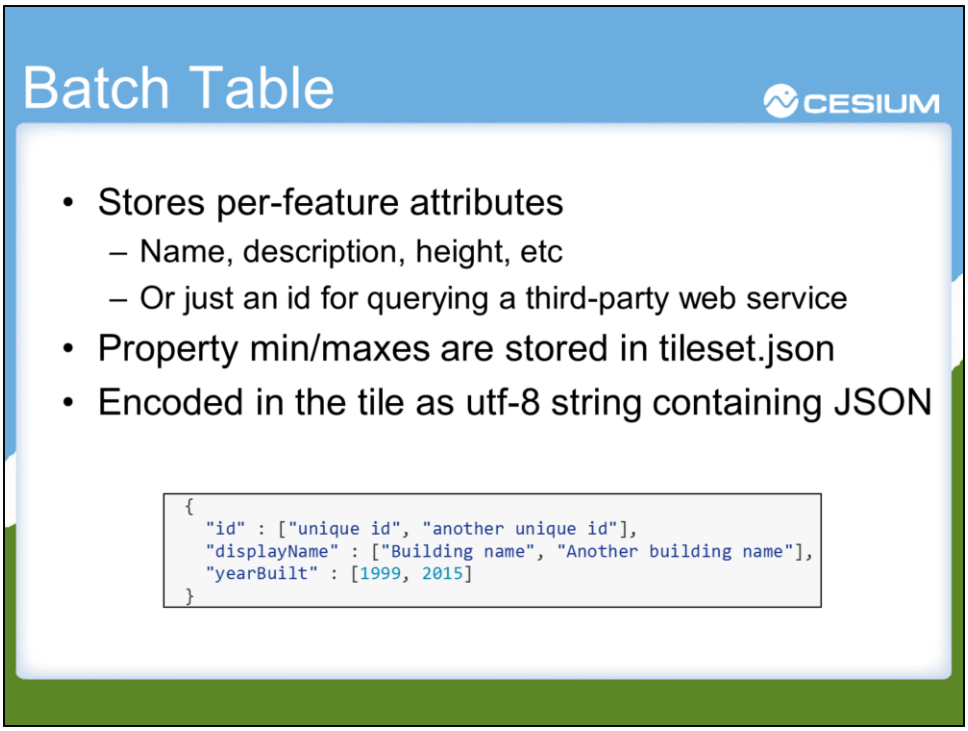
Building on glTF allows us to utilize the glTF software ecosystem.

# Batched (b3dm)

CESIUM



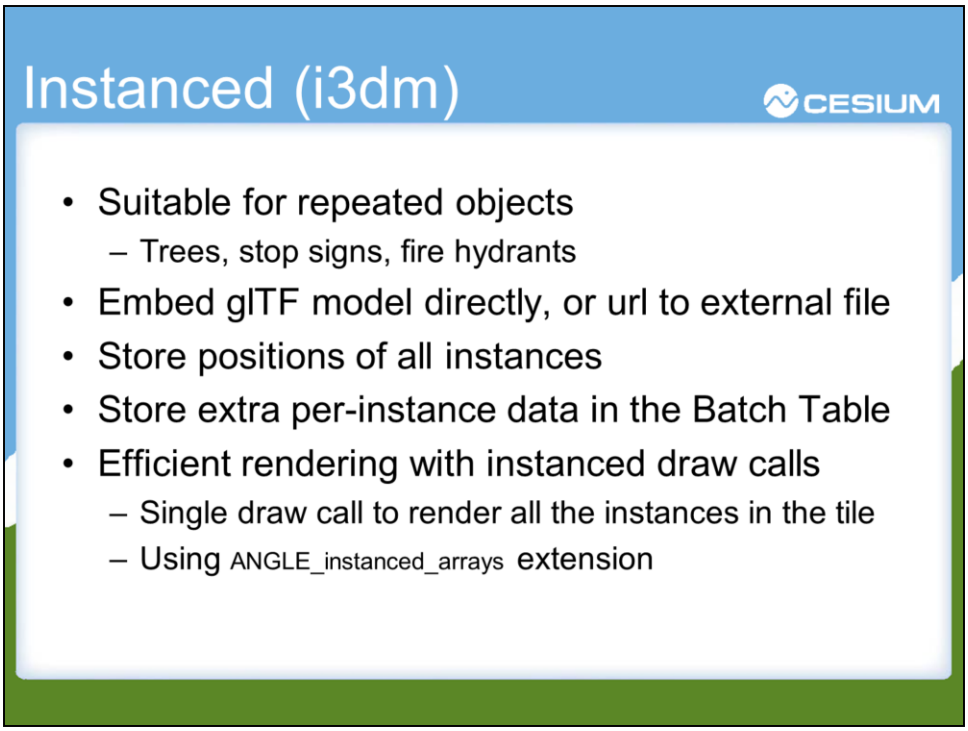

The most common use case for the Batched 3D Model format is buildings. Features are batched into a single 3D model. For the city example, you may have 100 or so buildings that are batched together, and then rendered with a single draw call. Features within the tile are differentiated by a batchId vertex attribute which allows for per-feature styling and per feature showing and hiding, all with no runtime cost. Finally the tile stores what we call a batch table which contains any per-feature attributes.




Some examples of common attributes include name, description, height, etc. Or even easier, just an id that is used for querying a third-party web service. The min and max values of the attributes across all tiles are stored in tileset.json. The batch table is encoded in the tile as JSON.

# Instanced (i3dm)

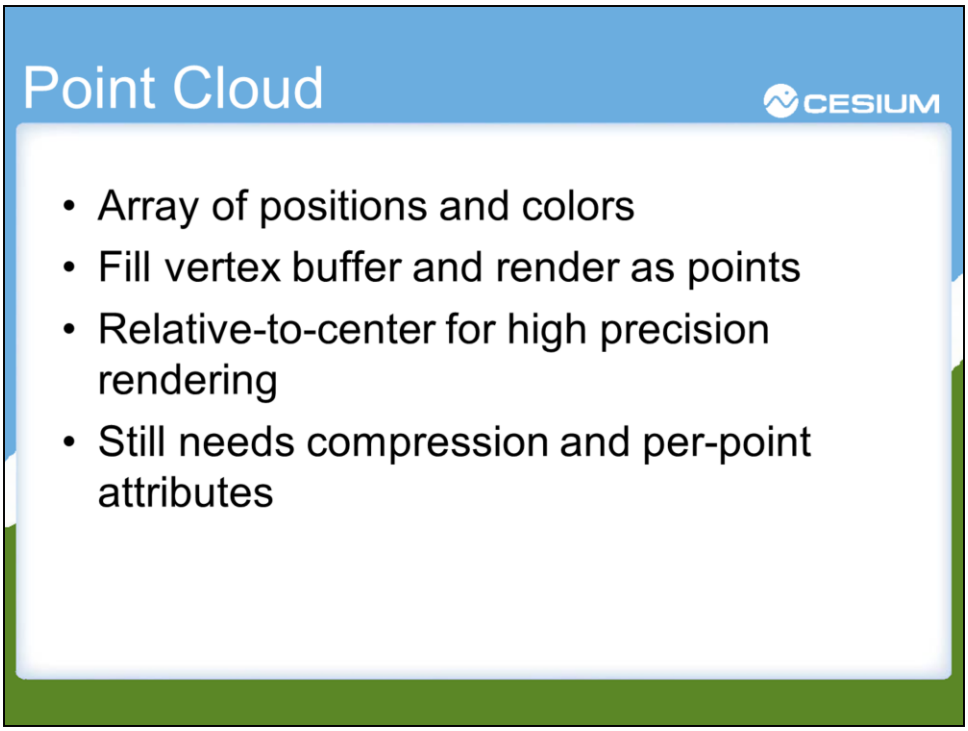
CESIUM



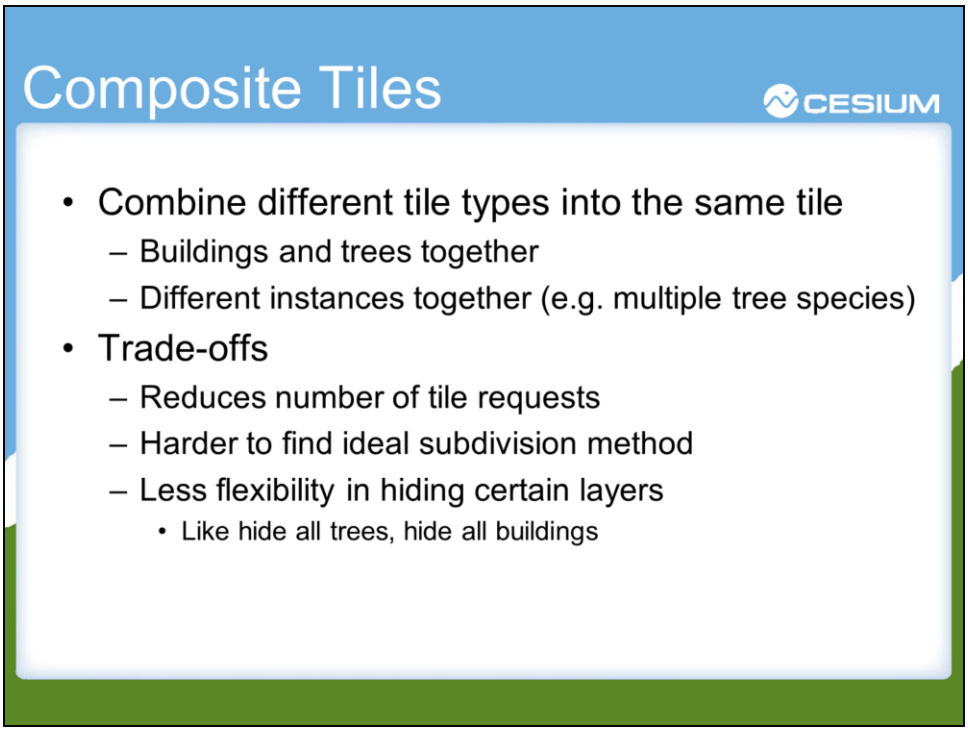

Next is instanced models, for rendering the same model multiple times. This tile type is suitable for repeated objects like trees, stop signs, and others. The 3d model may be stored in the tile directly or the tile may contain a url to the external file. Like with the Batched 3D Model format, any extra per-instance data is stored in the Batch Table. Also like the Batched format our goal is to render the contents of the tile in a single draw call. For this we utilize the instanced drawing extension in WebGL.

# Point Cloud

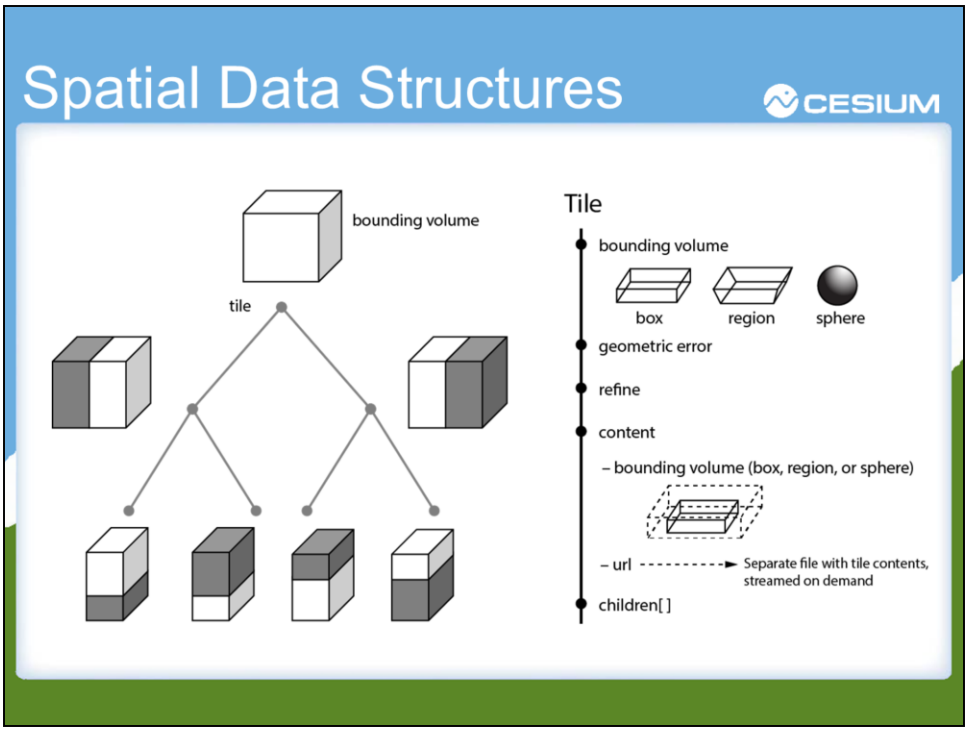
CESIUM



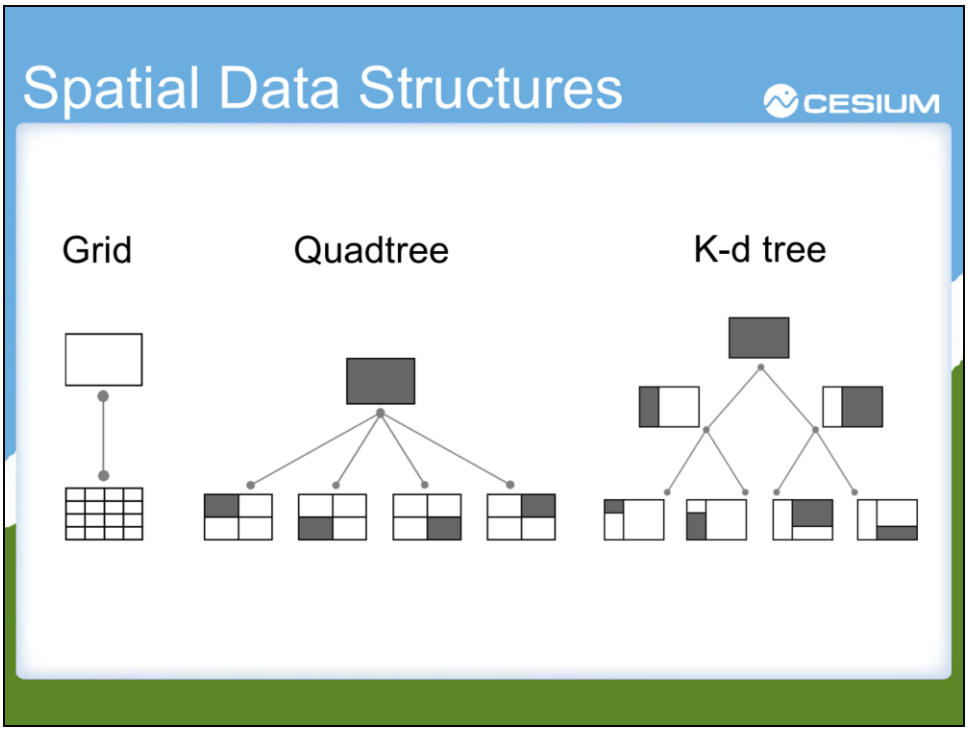

Another format is point clouds. This is the simplest format right now. It stores an array of positions and colors which can be directly uploaded to the GPU and rendered as points. The points are encoded relative to center, and we have a little more work to do in in terms of better compression and storing per-point attributes.



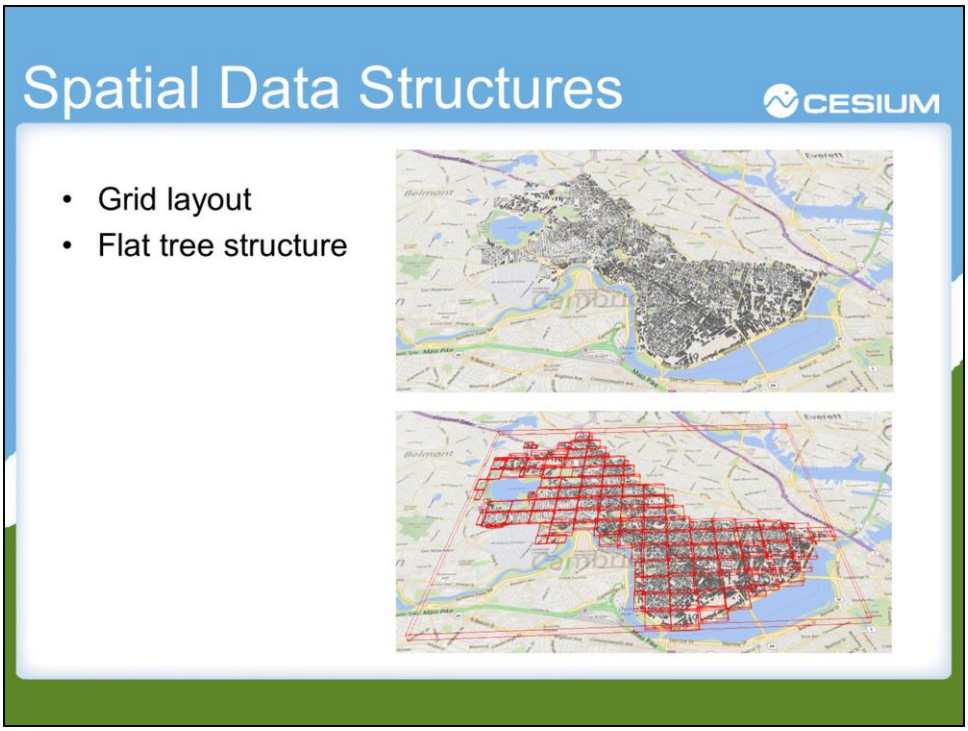

And the final tile format I'm going to talk about is composite tiles, which combine different tile formats into the same tile. For example, if you want to render buildings and trees together, or different instanced models together. However there are tradeoffs. On the positive side, this reduces the number of tile requests. On the negative side, the content generation pipeline may have a harder time finding an ideal subdivision method for diverse types, and less flexibility for hiding certain layers.




Now that I've covered the design and formats, I'm going to talk about the possibilities of using 3d tiles for creating spatial data structures.

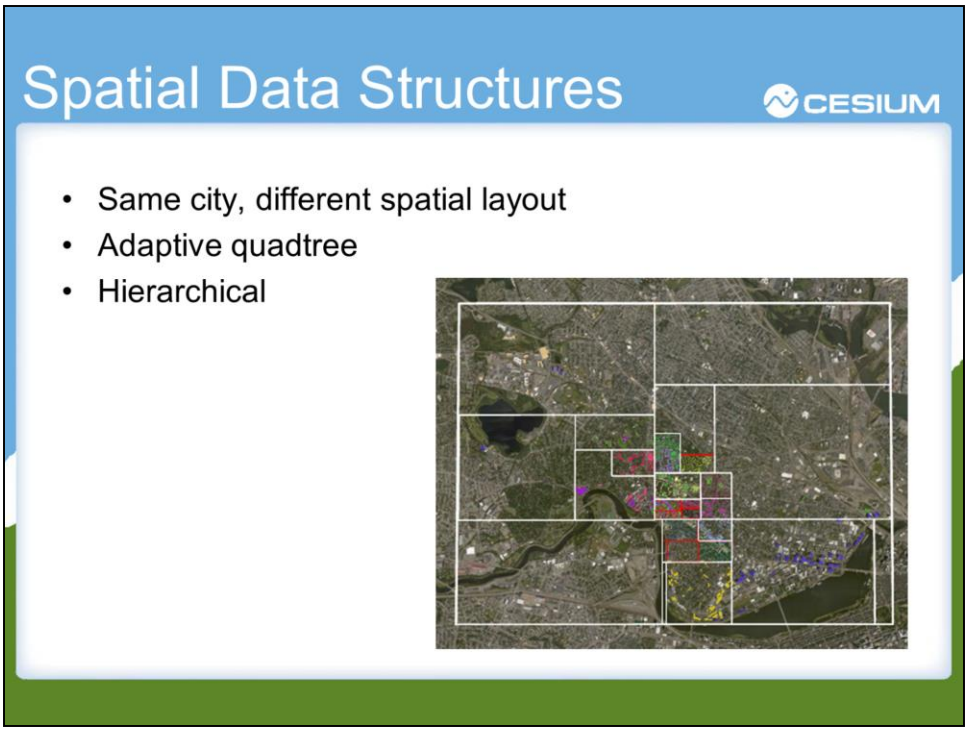



3D Tiles doesn't have a defined layout, its all up the content generator. For example you could divide your tileset into a grid, a quadtree, or a kd-tree.

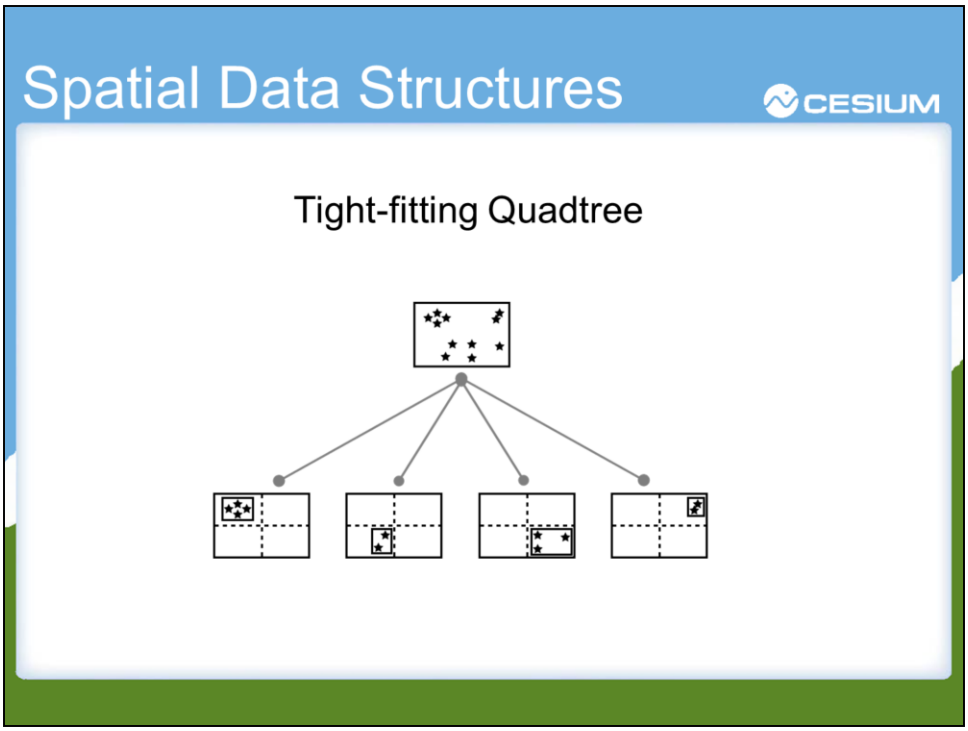



Buildings in Cambridge arranged in a grid layout.

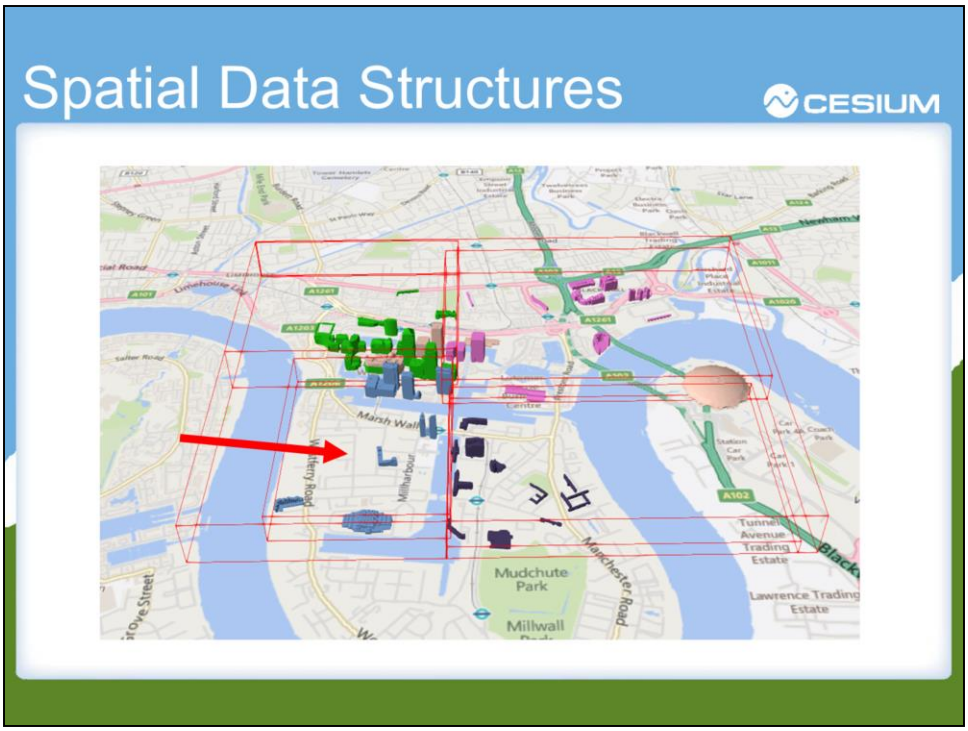
Each red box is a tile. The outer red box is the bounding volume for the tileset.



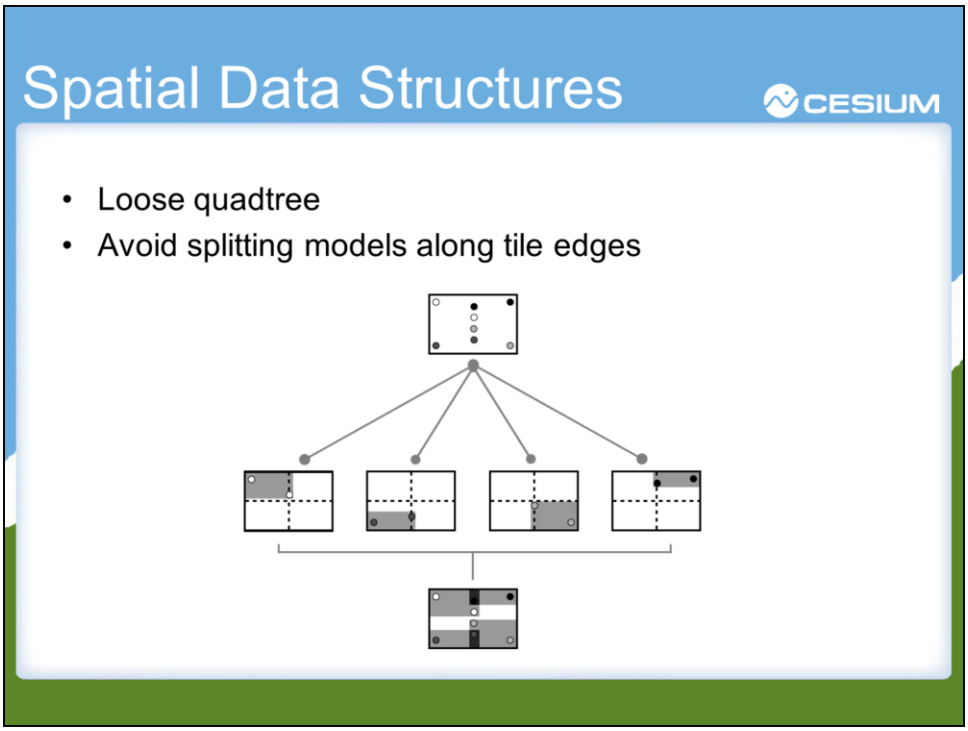

The same city subdivided into an adaptive quadtree. The hierarchical structure will benefit culling.



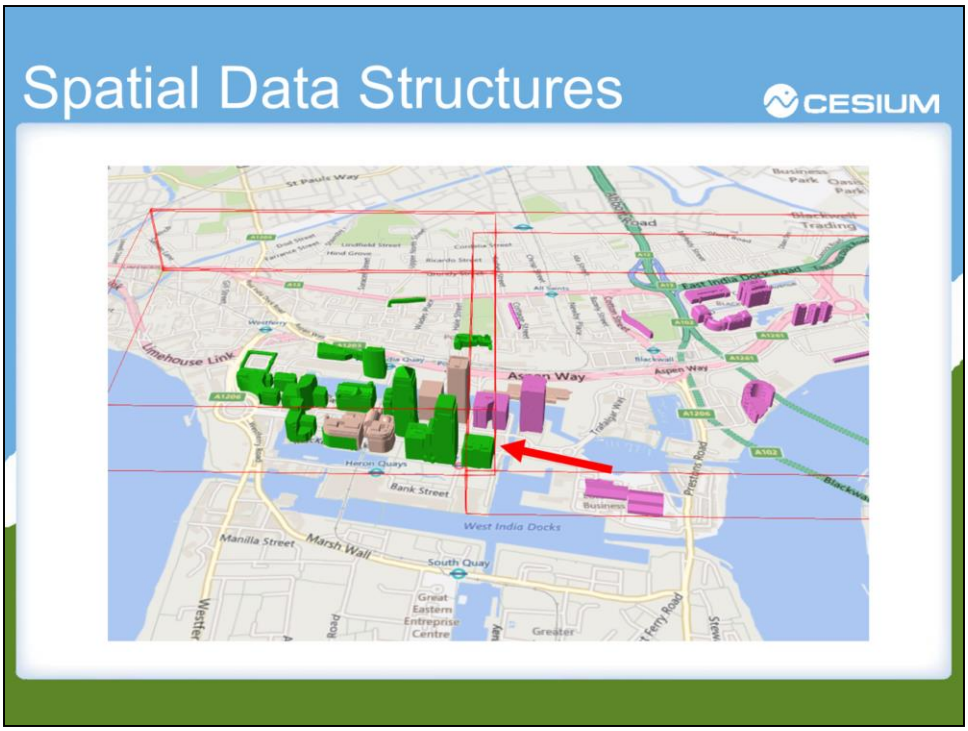

We can be flexible with the bounding volumes by finding tight bounding volumes around the child tiles.



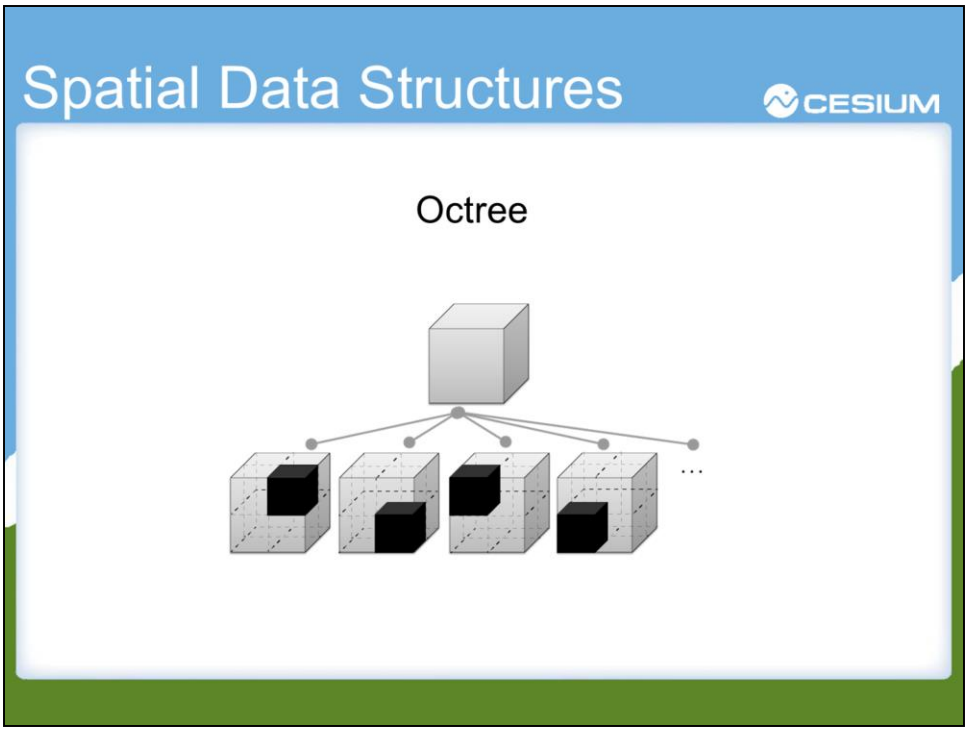

Bottom left tile tightly wraps around its content (blue buildings).



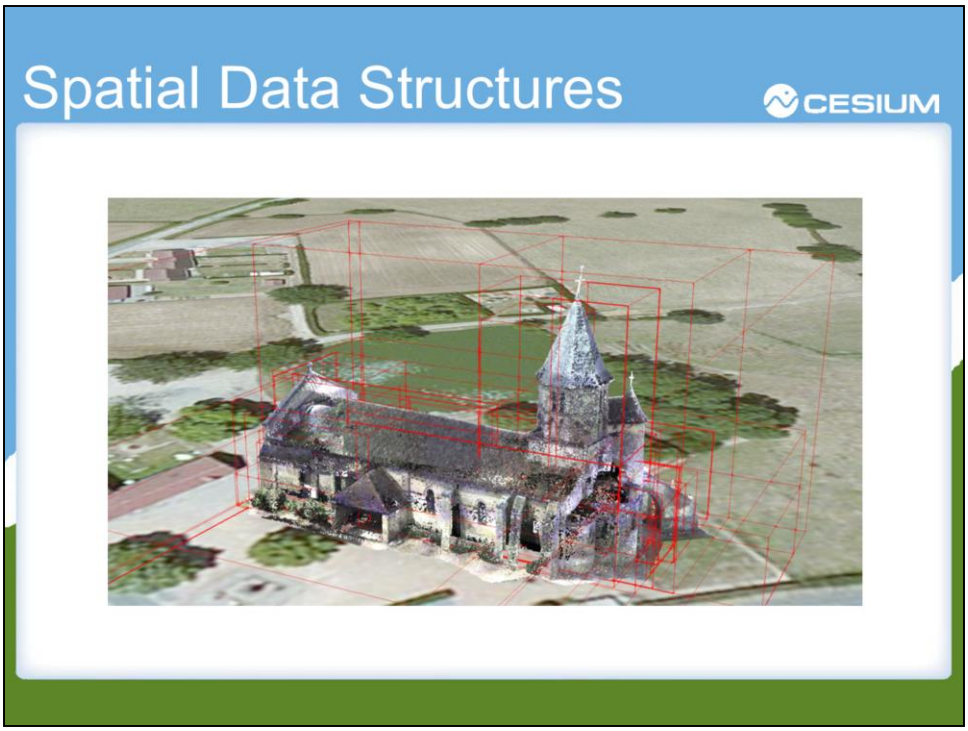

Another helpful subdivision technique is the loose quadtree, where neighboring tiles can overlap in order to avoid splitting models along tile edges.




The green building in the middle would have been split in half, but the left bounding volume extends into the right bounding volume to avoid that. Overlapping neighbor tiles is just fine by the spec.

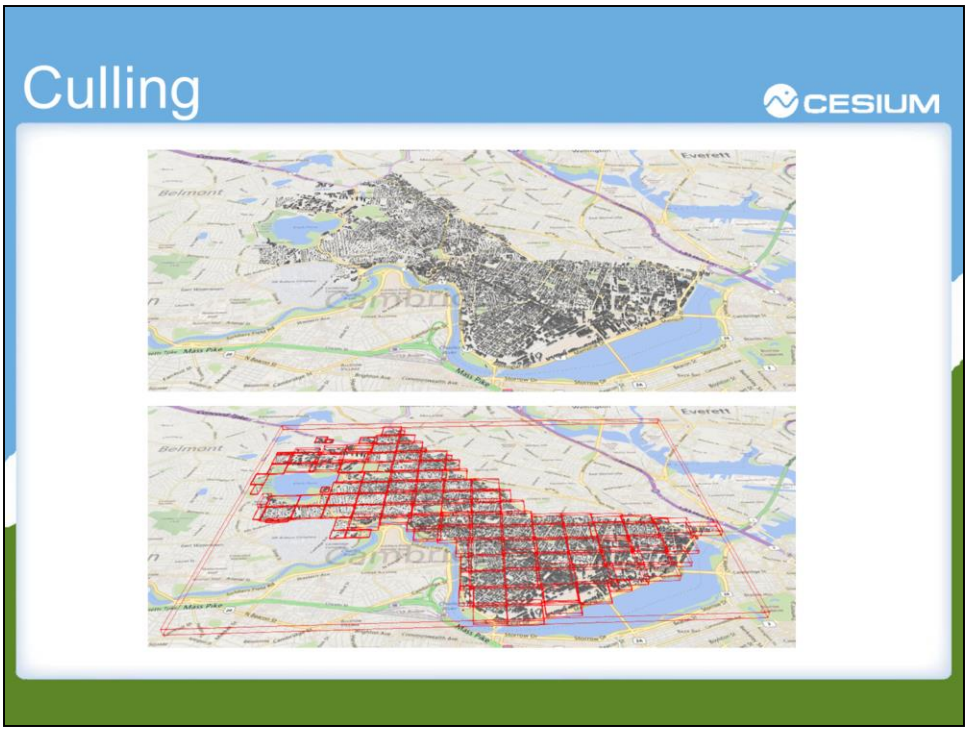



While most of the subdivision techniques we've presented have been 2D based, you can also define 3D subdivisions.

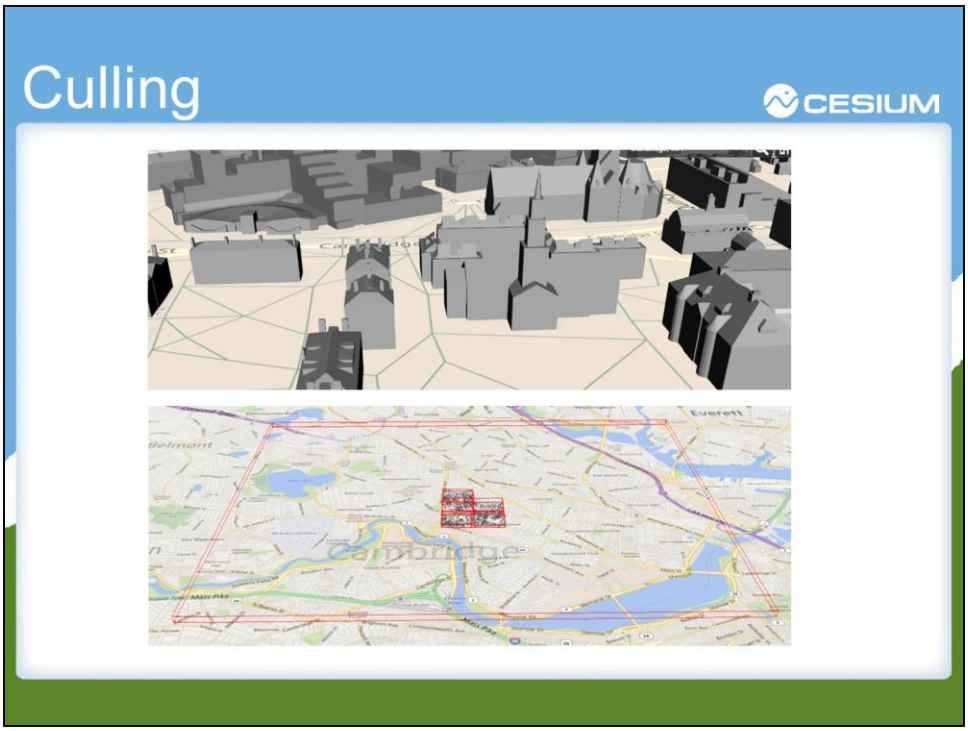



Non-uniform octree subdivision

Need to slice in all 3 dimensions not just latlong

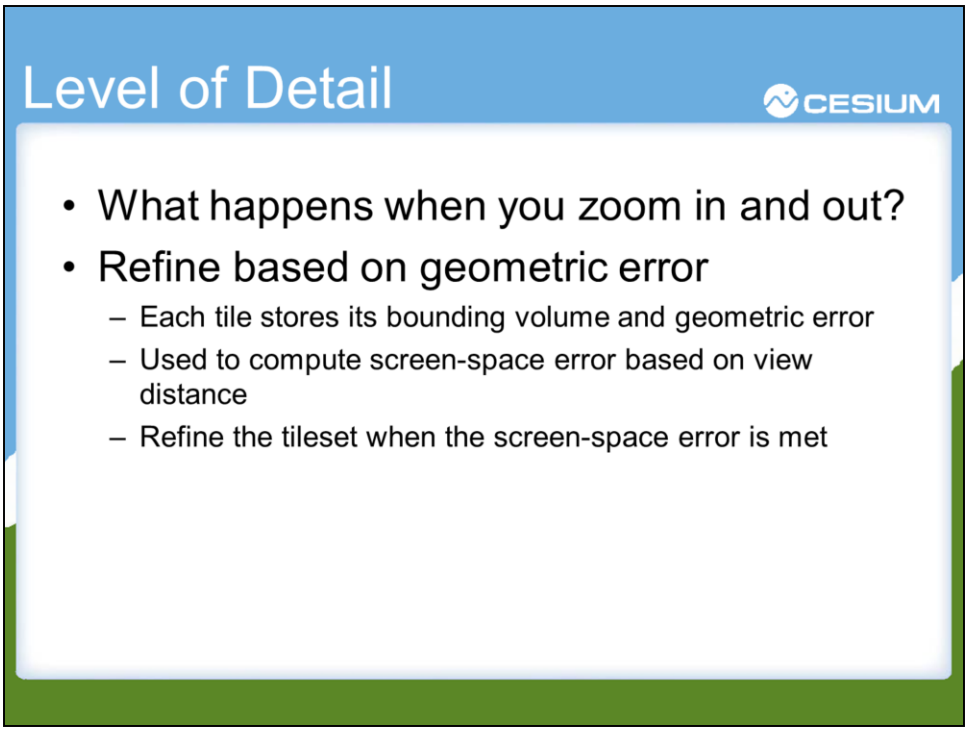



Say you have a tileset organized like this…

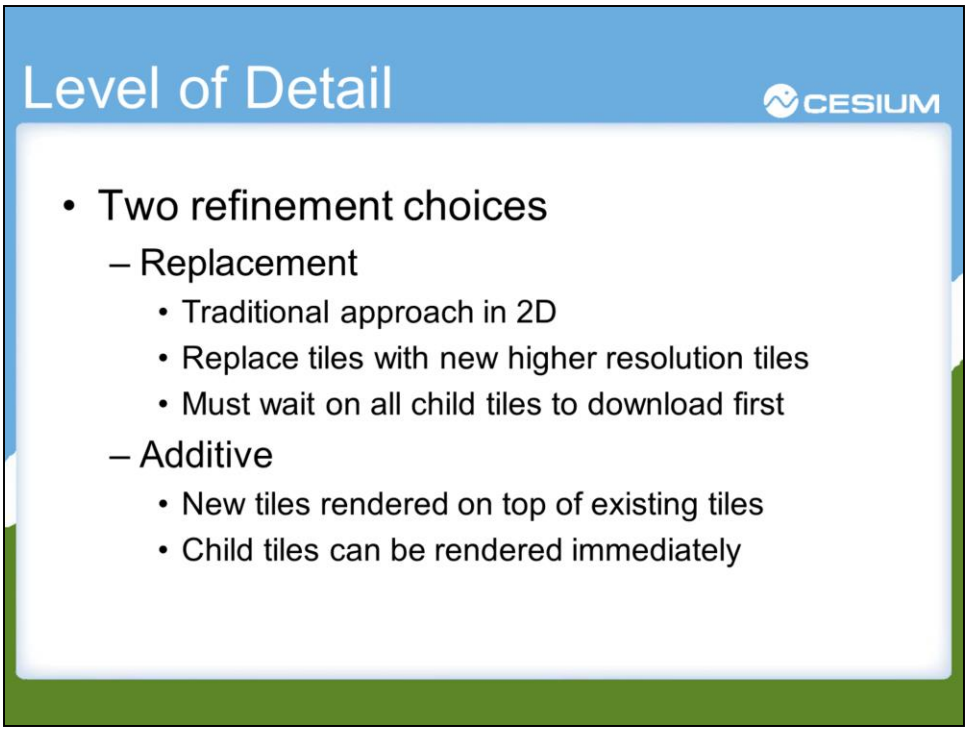



After zooming in, only 3 tiles are visible and rendered.

Culling is based on the data structure, hierarchical structure makes it easier to cull. Plane masking is used for faster culling.




When zooming in or zooming out, or just moving the camera in general, new tiles need to be loaded based on their level of detail. Each tile contains a geometric error which is converted to screen-space error based on the camera's view in the scene. When the error is met, we need to refine the tileset.

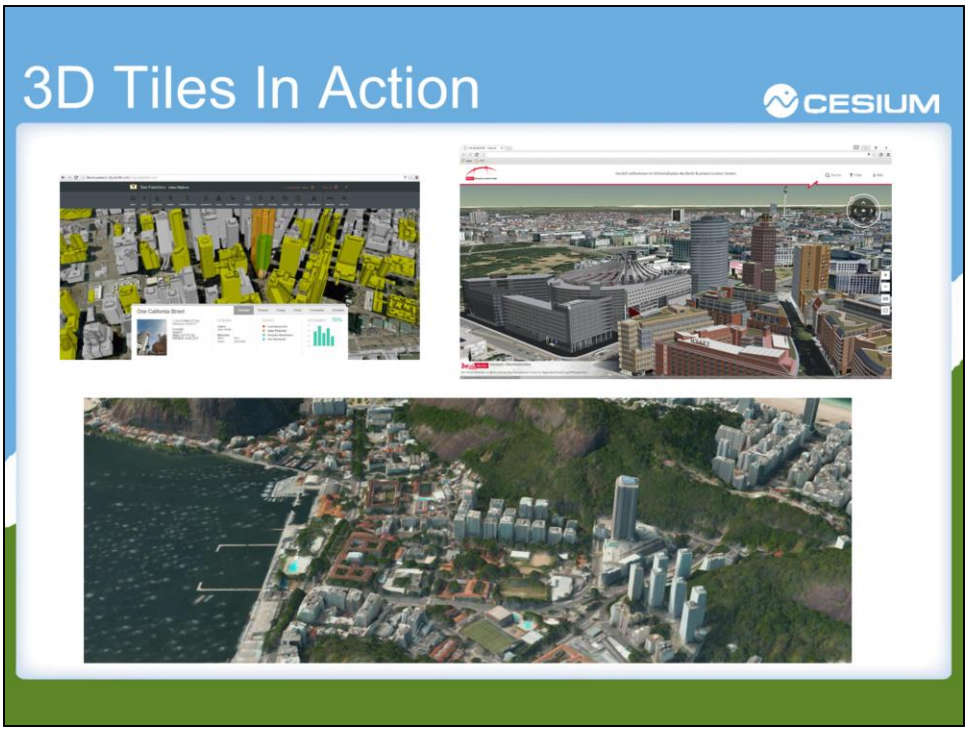



The two refinement choices are replacement and additive refinement. Replacement refinement is the traditional 2D approach where lower LOD tiles are replaced by new higher resolution tiles, and because of that we need to wait for all the child tiles to download first. Additive refinement instead renders new tiles on top of existing tiles, which allows for child tiles to render right as they are loaded in.


Here is an example of additive refinement. From this viewpoint, only the largest buildings need to be rendered. They exist inside the root tile.


Zooming in closer, the child tiles begin to load and render alongside the root tile.


And zooming closer, the grandchild tiles are loaded.




After a lot of technical slides, I just want to leave with some images of 3D Tiles in action to show what is possible. We are excited to see what others do with 3D Tiles.

Top Left: City Zenith - San Francisco

Top Right: virtualcitySYSTEMS - Berlin

Bottom: VRICON


Sign in and vote at foss4gna.org
Evaluate the Sessions
FOSS4G NORTH AMERICA 2016
-1 0 +1


CESIUM
CESIUM
CESIUM
CESIUM
CESIUM
CESIUM
CESIUM
CESIUM
CESIUM

CESIUM

## Check out 3D Tiles

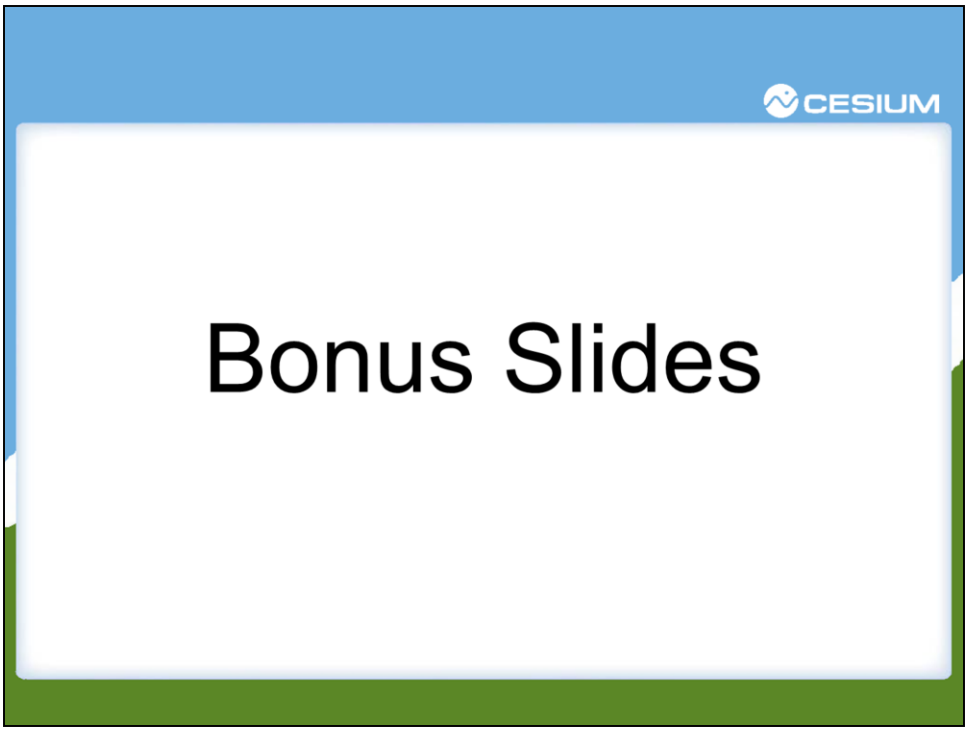
https://github.com/AnalyticalGraphicsInc/3d-tiles

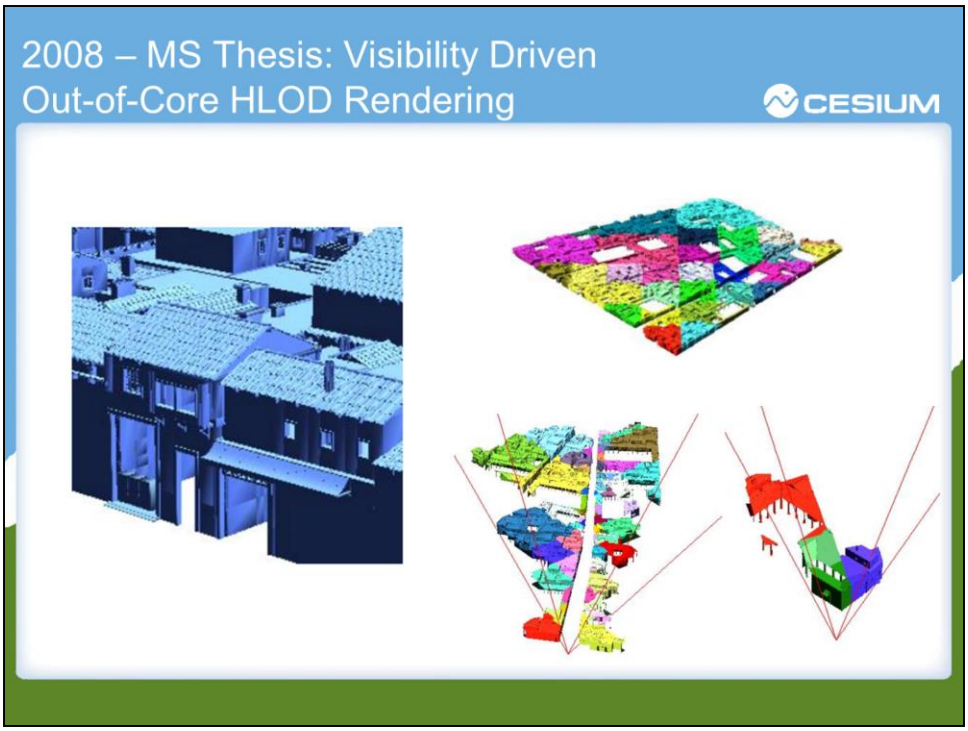
## Contact

agi
Sean Lilley
slilley@agi.com
@lilleyse

## Thanks to

CyberCity3D
Fraunhofer IGD
virtualcitySYSTEMS
VRICON
cityzenith


CESIUM
Bonus Slides




Starting with a brief personal history…

I was interested in rendering massive 3D buildings in 2005 even before grad school at the University of Pennsylvania. I remember mentioning it in my grad school application. My interest came after implementing a quadtree for view frustum culling for VPF (Vector Product Format) for AGI's STK desktop app, and reading the book Real-Time Rendering.

Fast-forward a few years, and I started a master's thesis on rendering 3D buildings using Hierarchical Level of Detail (HLOD). As work progressed, it became clear the the approach as applicable to lots of massive models, not just 3D buildings, hence the final generic title.

Screenshots

•Far left: Pompeii

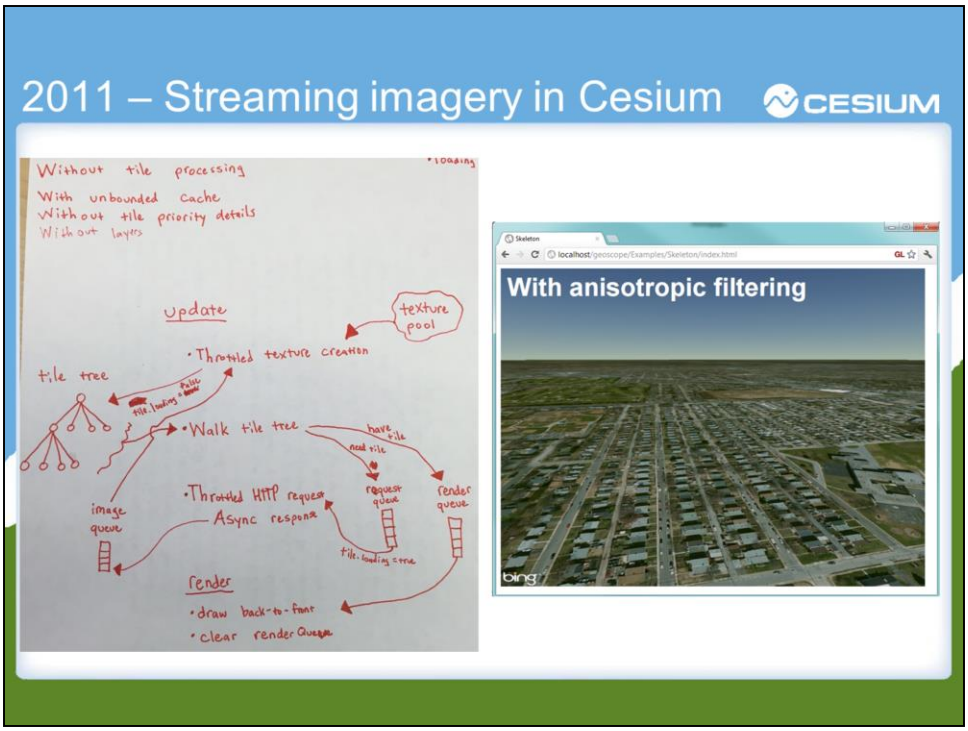
•Top right: colorized nodes

•Bottom

- Left: with view frustum culling

- Right: with view frustum and occlusion culling

MS Thesis: http://blogs.agi.com/insight3d/index.php/2009/02/19/out-of-core-rendering/

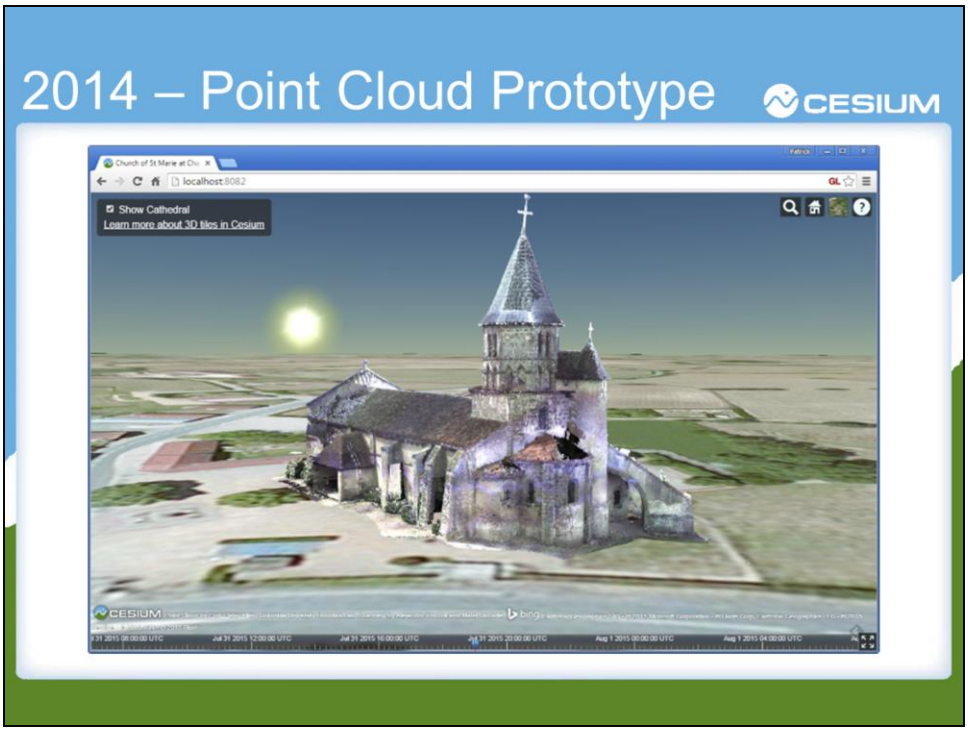



When we started Cesium, it was just a textured ellipsoid. When we added streaming imagery, I helped with the design (left), but Dan Bagnell actually implemented it. Kevin Ring and Scott Hunter then re-implemented it when they added streaming terrain.

The diagram here doesn't tell the whole story of today's implementation, but it is surprisingly not completely wrong.

The screenshot to the right is the earliest screenshot of Cesium's streaming imagery that I know of. Note that it predates the Cesium logo and even Cesium brand, "Geoscope" is in the url.

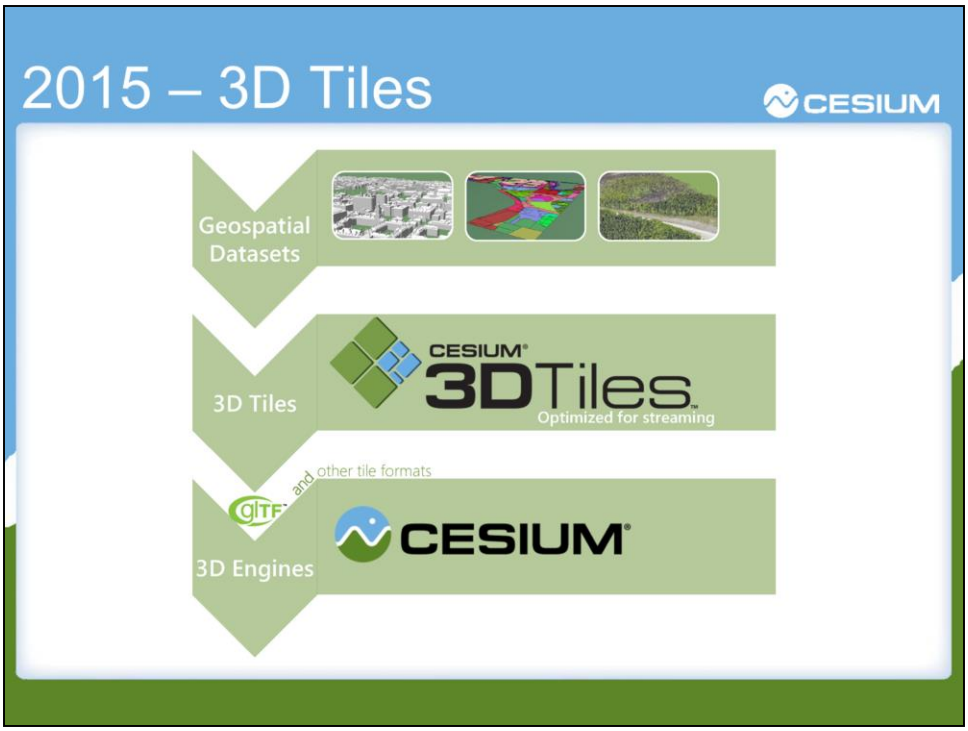
Screenshot from
http://cesiumjs.org/presentations/WebGLForDynamicVirtualGlobes.pdf




In 2014, Tom Fili and I prototyped some point cloud streaming in Cesium using HLOD and Layered Point Clouds LPC…

LPC: http://www.crs4.it/vic/data/papers/spbg04-lpc.pdf

Actual screenshot here is from 3D Tiles, not our prototype.



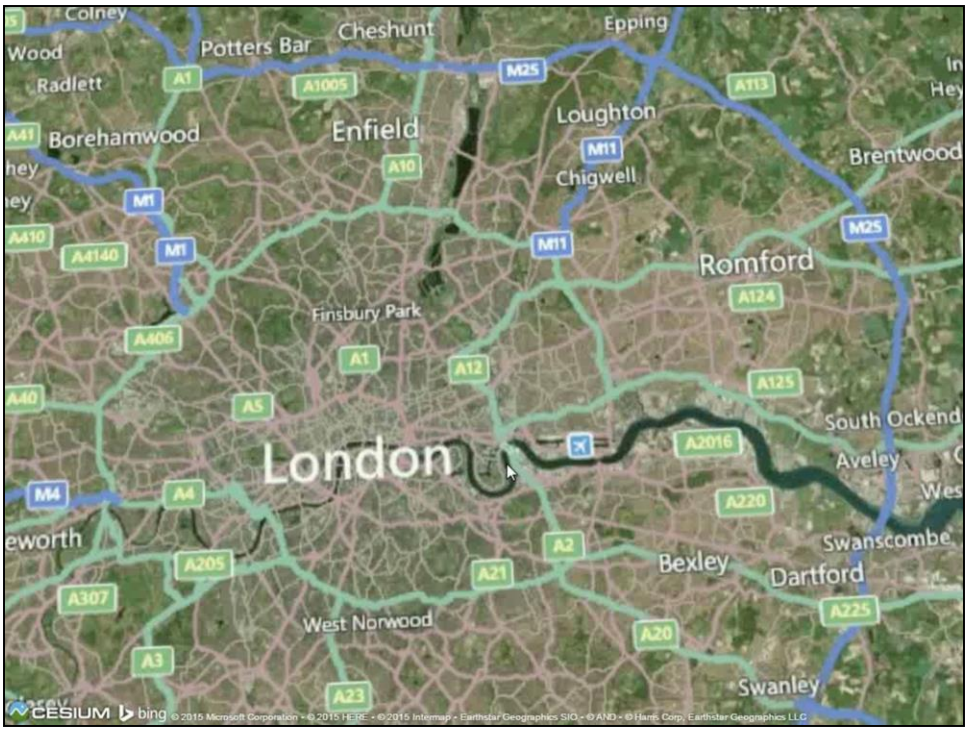

We never actually finished or released the point cloud prototype, but the experience was useful.

We shifted our attention to streaming 3D buildings. Given all our experiences along the way – my initial MS thesis, imagery, terrain, point clouds – it was obvious that that a core, but flexible, approach could be taken, and 3D Tiles were born.

I alluded to this at Web3D last year:
http://cesiumjs.org/presentations/CesiumCzmlGltf.pdf

And presented some of the technical approaches a bit later in the year:
http://cesiumjs.org/presentations/RenderingMassiveGeospatialDatasetsInCesium.pdf

Screenshot of CyberCity 3D buildings for Canary Wharf. Demo:
http://cesiumjs.org/CanaryWharf/

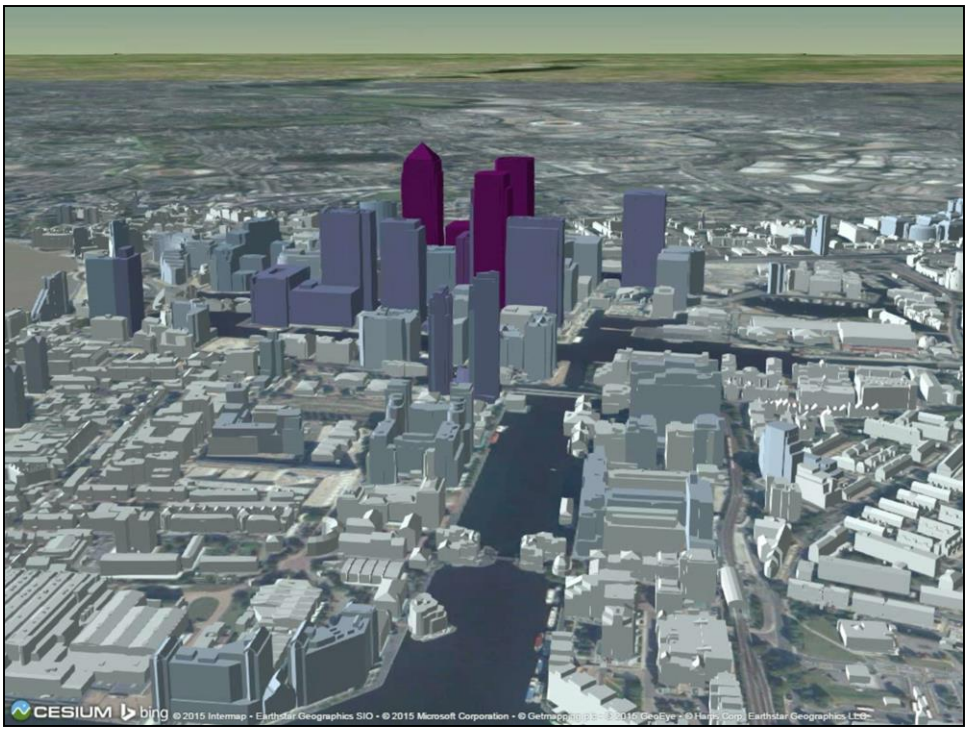



**3D Streaming and interaction**

Video: https://youtu.be/vdoKwp1izFE

Canary Wharf buildings from CyberCity 3D. Base map imagery from Bing maps.

- Buildings stream in based on the view.
- Buildings are individually selectable, e.g., highlight on mouse over.
- Full 3D – any view angle, drag individual buildings, grab the sky, zoom out to a global view and then back in.




**Styling**

Video: https://youtu.be/UmgOQ5TEOYA

- Color based on building height.
- Change color palette at runtime.
- Change number of height “bins” at runtime, i.e., what height ranges map to what colors.


**Streaming with Hierarchical Level of Detail (HLOD)**

Video: http://youtu.be/1useSwpuM8w

- Zoom in and root tile with the "most important" buildings appear. Freeze frame to take a closer look. "Important" buildings are tall and/or have a large footprint. Lots of other heuristics could be used too; it doesn't matter to the runtime engine.
- Zoom in a bit closer and some of the root node's children appear. Each tile is a single WebGL draw call, has a bounding volume used for view frustum culling, and has a geometric error that is used to know when to stop refining down the tree.
    - No individual building LOD
    - This is additive refinement; internal tiles, not just leaf tiles are rendered
    - This is a non-uniform loose-ish quadtree subdivision


**View Frustum Culling**

Video: http://youtu.be/LolnWVe-a9M

- Zoom in so only a subset of tiles are visible.  Tiles outside the view frustum are culled.
    - This is really effective for top-down views.  Horizon views are tough, which is where LOD kicks in to render more coarse tiles.
- Since this is using additive refinement, there are two bounding boxes.
    - One for the tile - using for tree traversal.
    - One for the buildings/content in the tile - used for view frustum culling.
- To avoid splitting or duplicating buildings, the tiles slightly overlap to accommodate buildings near the tile edges.

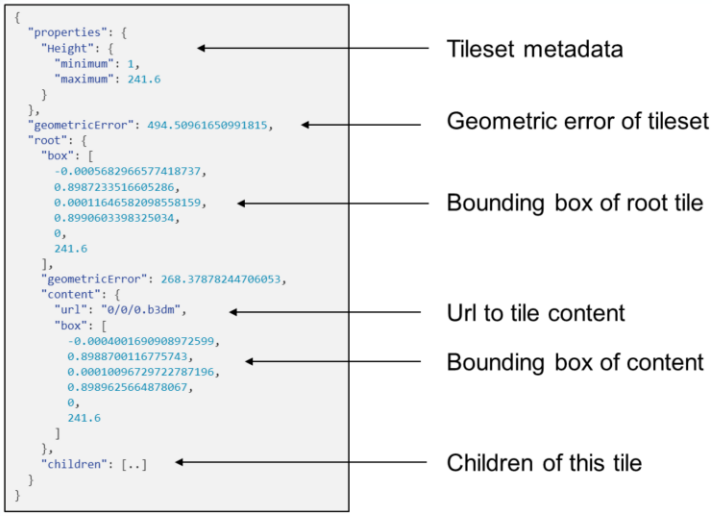

Video - http://youtu.be/1ZwzCBXPE-0

Point Clouds – using the same Cesium runtime code as 3D buildings (the Octree and Quadtree and just a 3D Tiles tree to Cesium) and a different tile format – points instead of glTF.

Zoom out, freeze frame, then zoom in to show just root tile. Unfreeze, then show colorized tiles. Like 3D buildings, this is using additive refinement.

# tileset.json

CESIUM



# Geometric Error

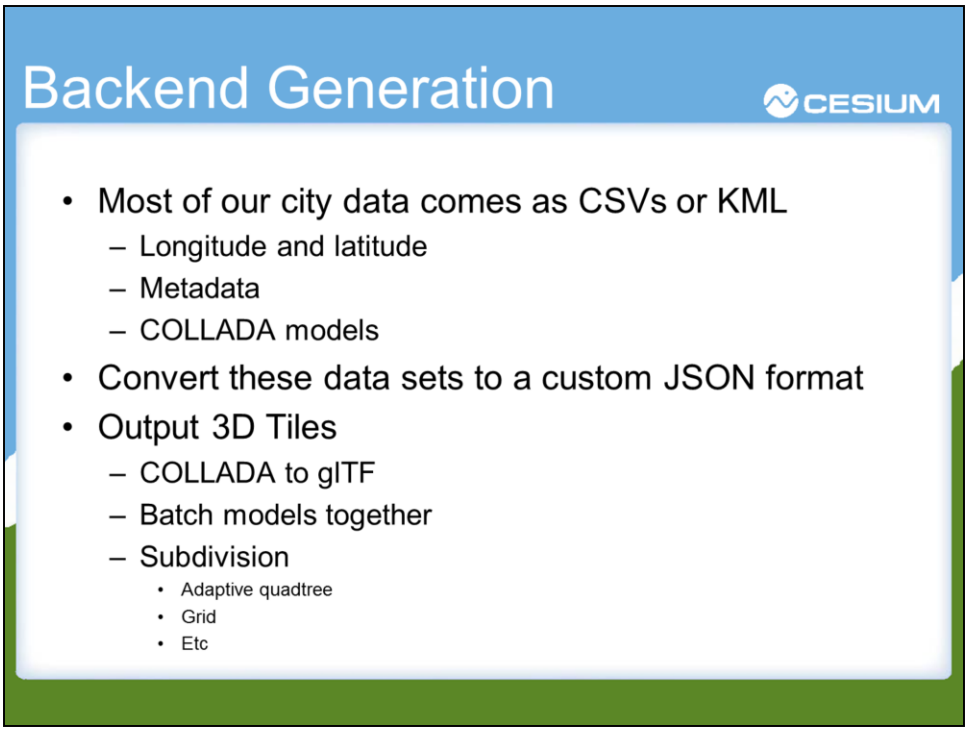
CESIUM

- Each tile contains a geometric error
- Defines the error in meters incurred if its children are not rendered
- At runtime, used to compute screen-space error based on view distance
- Refine the tileset when the screen-space error meets a certain tunable limit

# Vector Data

CESIUM

- For roads, labels, regions
- Define points, polylines, and polygons
- Alternative to KML and other formats
- How to handle cracks between tiles?
  - Adjacent tiles may have different LODs
  - Open problem
- Still in development

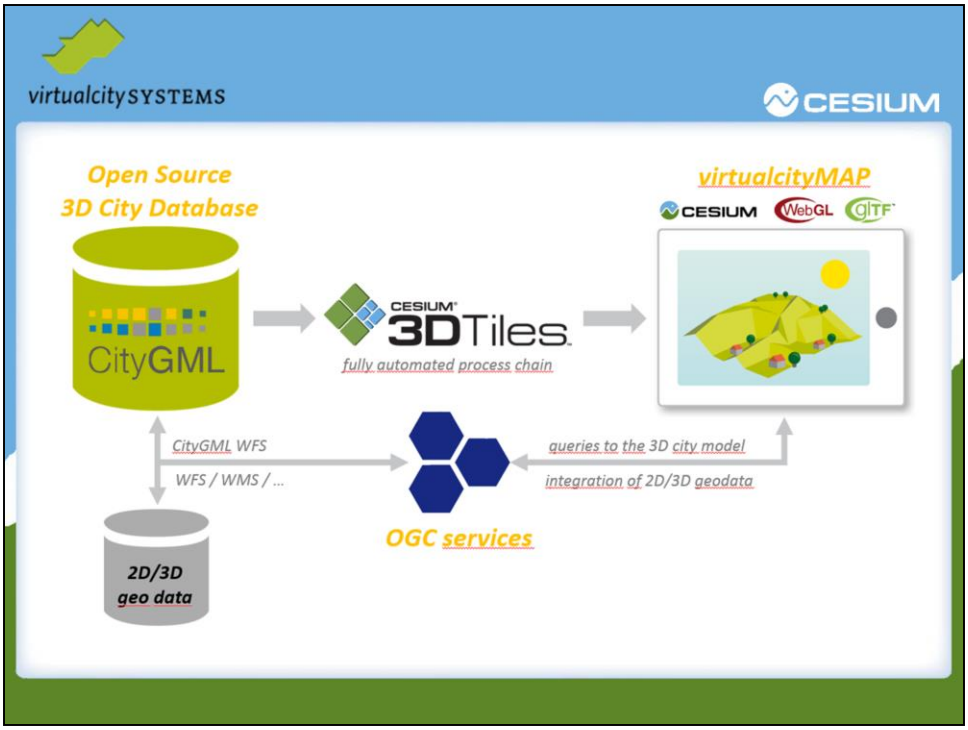



The subdivision technique often depends on the input data, other backend 3D Tiles generators may go with different techniques.

# 3D Tiles and OGC?

CESIUM

- Potential
  - Complement 2D standards: WMS, WMTS
  - Transmission format for 3DPS
  - Streaming for large CityGML datasets
  - Replacement for some KML use cases
- Designed for visualization clients
- Per-feature attributes

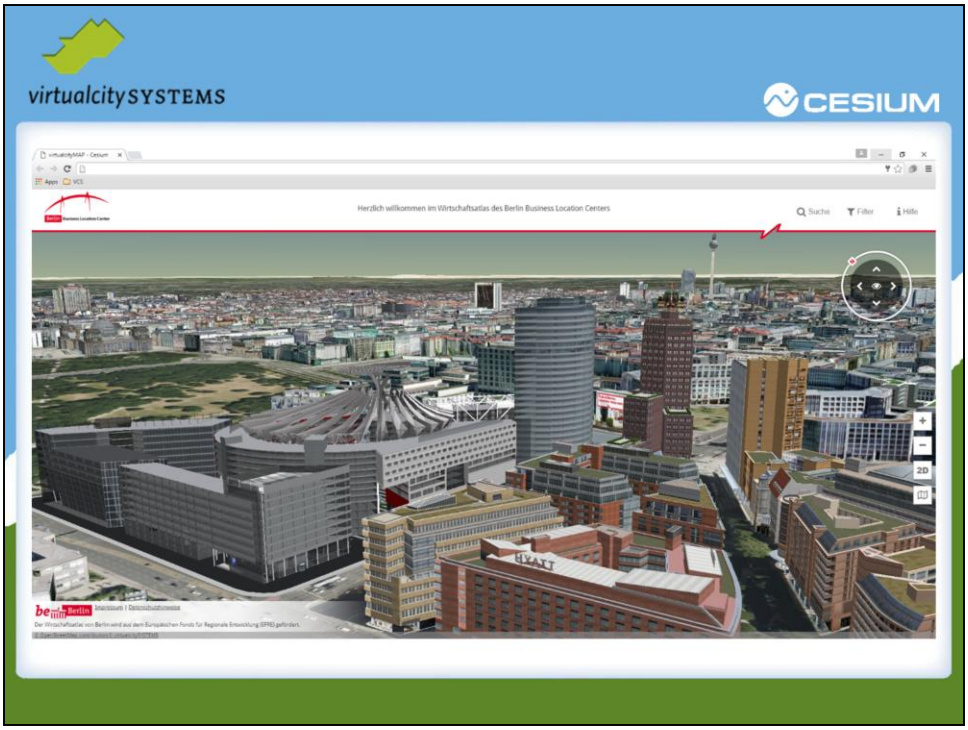

virtualcitySYSTEMS
CESIUM
Open Source
3D City Database
CityGML
CESIUM 3DTiles
fully automated process chain
virtualcityMAP
CESIUM
WebGL
glTF
CityGML WFS
WFS / WMS / ...
queries to the 3D city model
integration of 2D/3D geodata
OGC services
2D/3D
geo data

virtualcitySYSTEMS

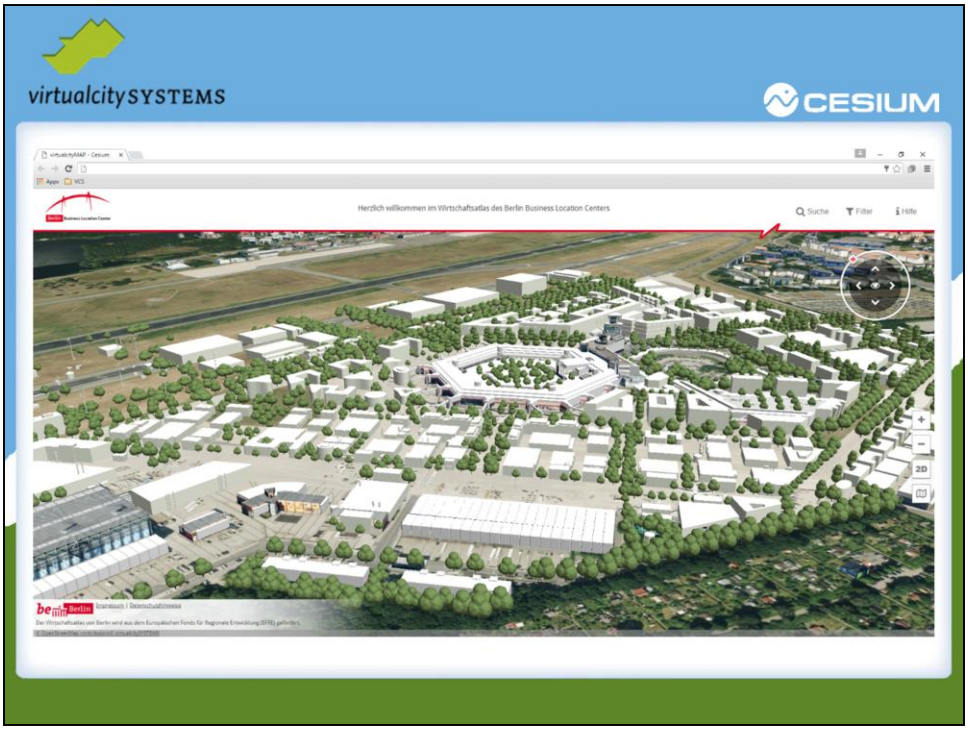
CESIUM

- Berlin 3D city model
  - 539,481 fully textured LoD2 buildings
  - 890 km² digital terrain model
  - Approx. 370 LoD3 buidings and 3 LoD4 buildings (interior details)
  - City trees (SolitaryVegetationObject), bridges, S-Bahn stations, …
  - Data volume
    - 16.6 GB CityGML file (XML unzipped)
    - 19.4 GB texture files

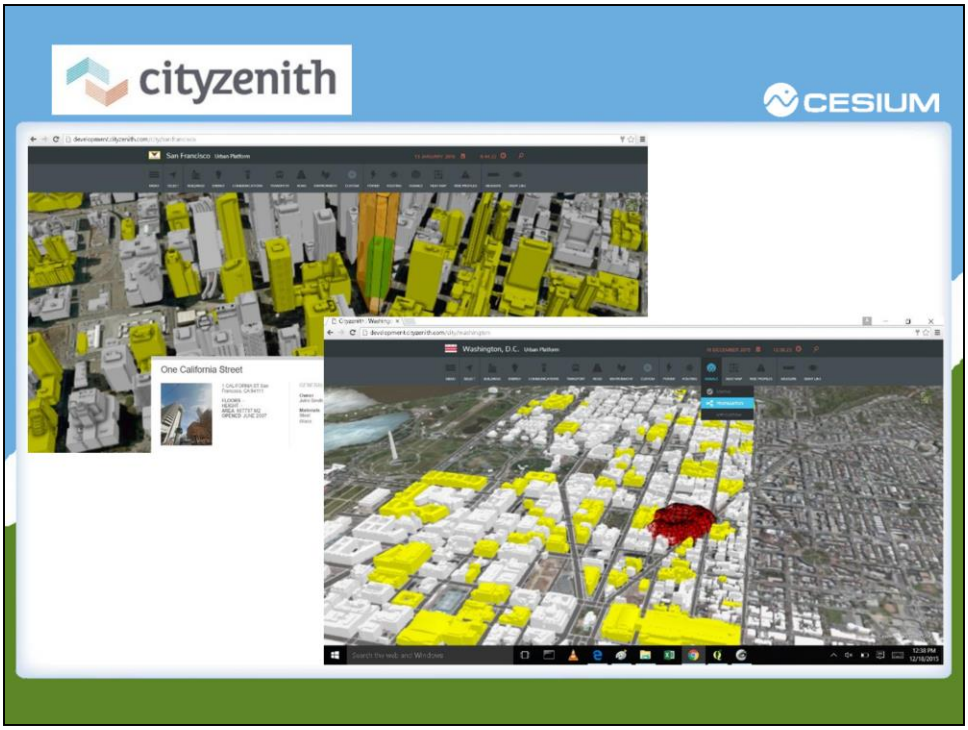
- 3D Tiles / b3dm files for the entire model
  - Three TMS layers with different LoDs and texture resolutions
    - Level 15: LoD2 only and low-res textures (2.54 GB)
    - Level 16: LoD2 only and mid-res textures (4.22 GB)
    - Level 17: LoD2-4 and high-res textures (19.5 GB)




Screenshot: Berlin

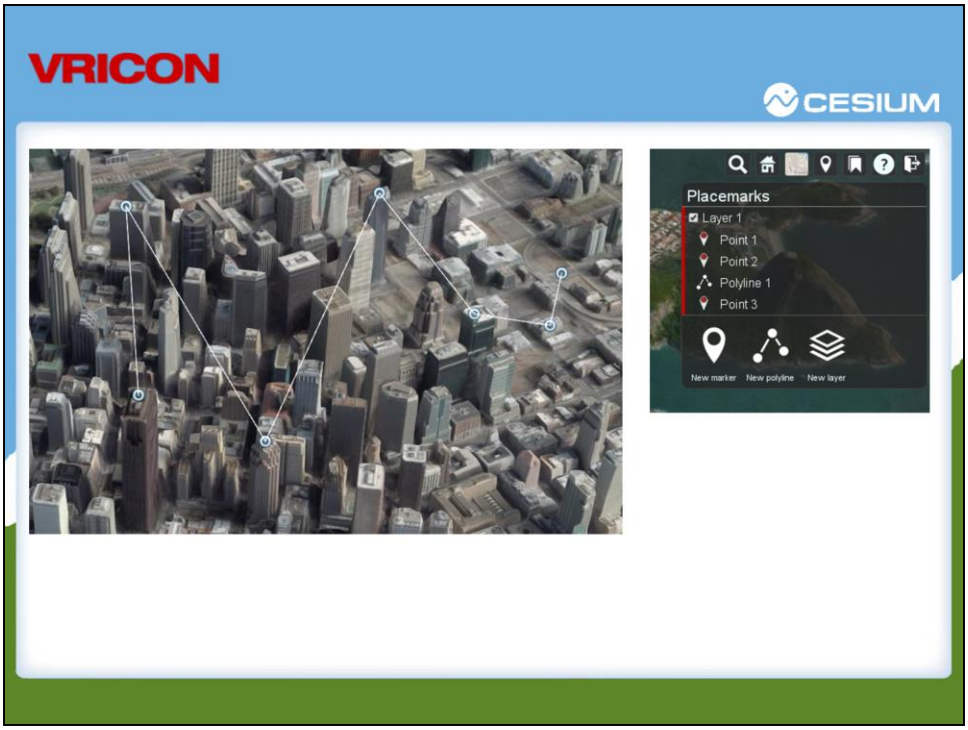



Screenshot: Berlin




Top left - San Francisco.  Prebaked AO

Bottom right – Washington, D.C.



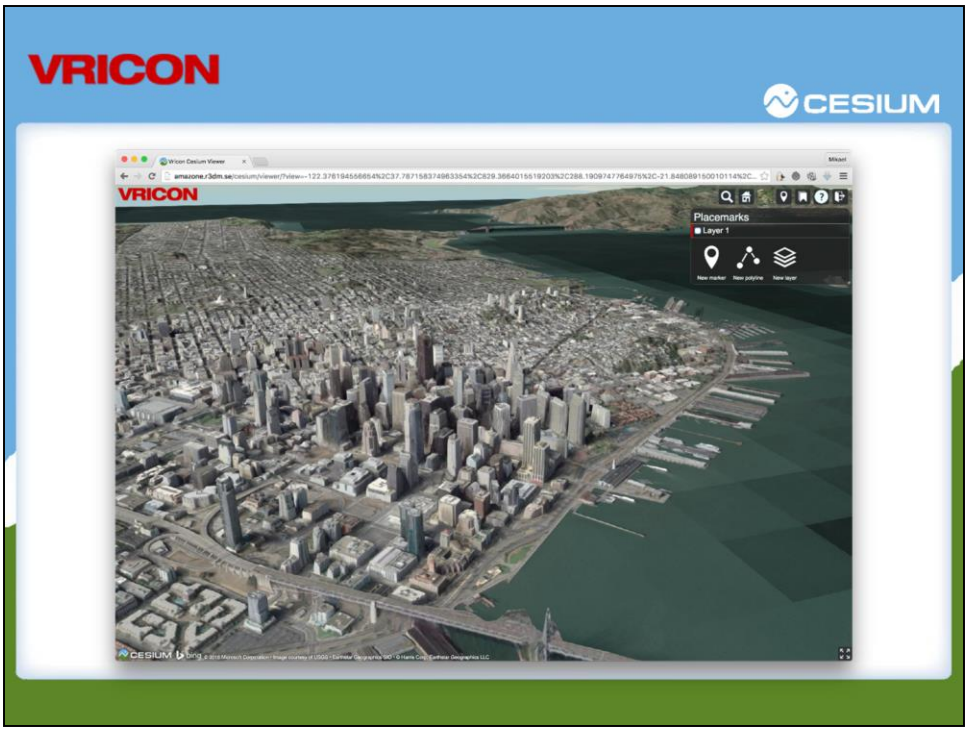

Runtime annotation gets exact click position (e.g., as opposed to the building's center) from depth buffer using Cesium's pickPosition().

VRICON

CESIUM

VRICON

CESIUM




http://www.vricon.com/