Bringing massive 3D geospatial data to the web

Sean Lilley
sean@cesium.com

WebGL + WebGPU meetup - April 26, 2022
About Cesium

Building an open platform for 3D geospatial

● CesiumJS: open-source JS library for 3D globes
  ○ Started in 2011
  ○ Early adopters of WebGL
  ○ One of the first glTF loaders

● Exponential growth of geospatial data lead to creation of 3D Tiles

● Increasing amount of semantically-rich data lead the development of 3D Tiles Next
What we’ll cover

- Background on 3D Tiles
- What’s new in 3D Tiles Next
- How it’s implemented efficiently in WebGL
- Open standard for massive 3D geospatial data
- 3D Tiles provides a spatial data structure for glTF
- Each “tile” is a glTF
- Hierarchical level of detail
  - Leaves are full resolution
  - Each parent is a simplification of its children
- Only stream what you need for a given view
● This works well for all sorts of data
  ○ Massive models (e.g. photogrammetry)
  ○ Instanced features
  ○ Point clouds
  ○ 3D buildings
Demo: 3D Tiles Inspector

Data: Aerometrex  
https://demos.cesium.com/ferry-building
Upcoming revision for 3D Tiles

Goals:
- More robust and efficient semantic **metadata**
- **Implicit tiling**: compact, sparse quadtrees and octrees for massive simulations and analytics
- Cleaner integration with the **glTF ecosystem**

https://cesium.com/blog/2021/11/10/introducing-3d-tiles-next/
Metadata

- Augment real-world data with semantics from AI and machine learning
- Metadata at many granularities
- Decoupled design: semantics, type
Per-Texel Metadata

**Property textures** for fine grained surface properties

**Feature ID textures** for feature identification

Data source: Aerometrex

Data source: Maxar
Implicit Tiling

- Compact representation of octrees and quadtrees
- Tiles located directly by (level, x, y, [z])
- Random access provides
  - Accelerated spatial queries
  - Efficient traversal at runtime
  - Efficient partial updates for changing scenes

S2 Base Globe Sandcastle
3D Tiles Next and glTF

- 3D Tiles now references glTF directly
- Leverage more glTF tools & extensions
  - Compression: KTX 2.0, MESHOPT, Draco
  - GPU instancing
  - More physically-based materials: specular, clearcoat, sheen, etc.
- New glTF extensions for fine-grained metadata
  - EXT_mesh_features: https://github.com/KhronosGroup/glTF/pull/2082
  - EXT_structural_metadata: https://github.com/KhronosGroup/glTF/pull/2151
CesiumJS renderer

- Built on WebGL
- Global scale precision - coordinates are in the 6 million+ meter range
- Hybrid multi-frustum logarithmic depth buffer and emulated double precision
  - Eliminates z-fighting and jittering artifacts
  - Relative to center rendering for tiles
  - GPU relative to eye rendering for point collections, polylines, etc
- Highly tuned 3D Tiles streaming engine and glTF engine

https://cesium.com/blog/2018/05/24/logarithmic-depth/
3D Engine Design for Virtual Globes
Rendering the Whole Wide World on the World Wide Web
Using Multiple Frustums for Massive Worlds, Rendering Massive Virtual Worlds Course, SIGGRAPH 2013
**Batching**

- WebGL draw calls can be expensive, we want to minimize how many we make.
- Objects in the glTF are batched together and differentiated by a feature ID vertex attribute.
- In the shader, the ID can be used to look up rendering details (colors, material properties, transformations).
In CesiumJS, we assign features a 32-bit integer pick ID and encode it as RGBA bytes in a texture.

- Render the pickId in its own pass
  - We only render a small viewport around the mouse, typically 3x3 pixels
- Read rendered pixels to determine which feature was clicked
• When rendering many similar-shaped models (e.g. trees, streetlights), redundant to store the geometry over and over
• Store the geometry once, and then only the attributes that change (e.g. model matrices)
Custom Shaders

- New experimental feature – user-defined shaders for styling tilesets and models
- Very useful in combination with feature IDs and structural metadata

Custom Shaders Sandcastle
Demo: Photogrammetry Classification

Data: Aerometrex

https://demos.cesium.com/ferry-building
Demo: OWT Uncertainty Textures

Data: Maxar

https://demos.cesium.com/owt-uncertainty
Demo: S2 Base Globe

Data: Maxar

https://demos.cesium.com/owt-globe
What’s next?

- 3D Tiles Next is becoming 3D Tiles 1.1
  - Update to OGC community standard - now open for public comments! [https://www.ogc.org/pressroom/pressreleases/4687](https://www.ogc.org/pressroom/pressreleases/4687)
  - Latest specification draft: [https://github.com/CesiumGS/3d-tiles/tree/draft-1.1/specification](https://github.com/CesiumGS/3d-tiles/tree/draft-1.1/specification)
- Finishing up CesiumJS implementation
- To learn more...
  - Introducing 3D Tiles Next (blog post)
  - CesiumJS sandcastle demos
  - 3D Tiles Ecosystem & Resources (in progress)
  - 3D Tiles Sample Data
- We’re hiring! [https://cesium.com/careers/](https://cesium.com/careers/)
Appendix
- Open source JavaScript library purpose-built for 3D geospatial on the web
- Custom 3D engine designed for high performance and precision at geospatial scale
- Started in 2011, first open source release in 2012
- Early adopter of WebGL

Data source: TRI

https://github.com/CesiumGS/cesium/wiki/CesiumJS-Features-Checklist
Implicit Tiling

- Efficient binary representation of quadtrees/octrees
- Tile and content availability stored in bitstreams
- Tiles located directly by (level, x, y, [z])
- Random access provides
  - Accelerated spatial queries
  - Efficient traversal at runtime
  - Efficient partial updates for changing scenes
- S2 bounding volumes avoid distortion at the poles
Global scale precision

- Global cartesian coordinate system for WGS84 and other ellipsoid models
- Double precision in JavaScript and single precision on GPU
- High precision is achieved through a hybrid multi-frustum logarithmic depth buffer and emulated double precision
  - Eliminates z-fighting and jittering artifacts
  - Relative to center rendering for tiles
  - GPU relative to eye rendering for point collections, polylines, etc
- Multiple domains supported natively in a common coordinate system
  - Satellites in geostationary orbit 42,164,000 meters from center of the Earth
  - Individual points in a point cloud at millimeter scale
  - And everything in-between: cities, undersea, underground
  - All in the same application
Performance

- Highly optimized 3D Tiles and terrain streaming engines
  - Hierarchical level of detail driven by screen space error
  - Multiple culling techniques - frustum culling, horizon culling, and fog culling - to reduce the number of tiles streamed
  - Tile request prioritization to favor loading “important” tiles first - perceived performance
  - Camera flight destination preloading
  - Level of detail skipping
- 3D Tiles quality is important too
  - Geometry and texture compression to keep network transfer and GPU memory low
  - High quality decimation - generate as few levels of detail as possible = less tiles streamed, better user experience
  - Adaptive subdivision for better culling
- Performance sensitive JavaScript
  - Avoid heap allocations in update loop - use scratch variables instead
  - Web workers for anything that could stall the main thread: decompression, polygon triangulation, etc

Data source: Nearmap

Data source: Maxar
Point cloud compression

===============================================

File size
===============================================

* # points: 313,605,932
* Properties: position, color
* Compression Error: 0.01 meters

* LAS (format 3) = 12.3 GiB
* pnts = 4.38 GiB
* glb = 4.68 GiB
* pnts draco = 1.19 GiB
* glb meshopt = 1.81 GiB

* pnts gzip = 3.72 GiB
* glb gzip = 3.67 GiB
* pnts draco gzip = 0.85 GiB
* glb meshopt gzip = 0.86 GiB

===============================================

Performance
===============================================

* Network speed (throttled): 50Mbps
* pnts = 33.3 seconds
* pnts draco = 9.25 seconds
* glb meshopt = 9.46 second (*)

* Experimental meshopt decoder doesn't use Web Workers yet - expecting to see even better decode performance

Data: Aalto University with support from City of Helsinki
https://zenodo.org/record/5578198##YioTWBNKiu4
Mesh compression

File size

<table>
<thead>
<tr>
<th>Format</th>
<th>Size (GB)</th>
<th>Compressed Size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + png</td>
<td>3.70</td>
<td>(3.45 gzip)</td>
</tr>
<tr>
<td>glb + jpeg</td>
<td>1.01</td>
<td>(0.82)</td>
</tr>
<tr>
<td>glb + ktx2/etc1s</td>
<td>1.15</td>
<td>(0.95)</td>
</tr>
<tr>
<td>glb + jpeg + draco</td>
<td>0.46</td>
<td>(0.45)</td>
</tr>
<tr>
<td>glb + jpeg + meshopt</td>
<td>0.55</td>
<td>(0.50)</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>0.68</td>
<td>(0.64)</td>
</tr>
</tbody>
</table>

Load Time (local disk, throttled 50Mbps)

<table>
<thead>
<tr>
<th>Format</th>
<th>Load Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + jpeg</td>
<td>24.2</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s</td>
<td>23.5</td>
</tr>
<tr>
<td>glb + jpeg + draco</td>
<td>11.15</td>
</tr>
<tr>
<td>glb + jpeg + meshopt</td>
<td>12.95</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>12.61</td>
</tr>
</tbody>
</table>

Load Time (local disk, no throttling)

<table>
<thead>
<tr>
<th>Format</th>
<th>Load Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + jpeg</td>
<td>5.16</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s</td>
<td>4.65</td>
</tr>
<tr>
<td>glb + jpeg + draco</td>
<td>5.69</td>
</tr>
<tr>
<td>glb + jpeg + meshopt</td>
<td>5.07</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>4.21</td>
</tr>
</tbody>
</table>

Memory Usage

<table>
<thead>
<tr>
<th>Format</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + jpeg</td>
<td>834 MB (texture) + 108 MB (geometry)</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>105 MB (texture) + 72 MB (geometry)</td>
</tr>
</tbody>
</table>

Data: Aerometrex - 2cm resolution photogrammetry - downtown San Francisco
Mesh compression

File size

<table>
<thead>
<tr>
<th>Format</th>
<th>Size</th>
<th>(Compressed Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + png</td>
<td>3.70 GiB</td>
<td>(3.45 GiB gzip)</td>
</tr>
<tr>
<td>glb + jpeg</td>
<td>1.01 GiB</td>
<td>(0.82 GiB)</td>
</tr>
<tr>
<td>glb + ktx2/etc1s</td>
<td>1.15 GiB</td>
<td>(0.95 GiB)</td>
</tr>
<tr>
<td>glb + jpeg + draco</td>
<td>0.46 GiB</td>
<td>(0.45 GiB)</td>
</tr>
<tr>
<td>glb + jpeg + meshopt</td>
<td>0.55 GiB</td>
<td>(0.50 GiB)</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>0.68 GiB</td>
<td>(0.64 GiB)</td>
</tr>
</tbody>
</table>

Load Time (local disk, throttled 50Mbps)

<table>
<thead>
<tr>
<th>Format</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + jpeg</td>
<td>24.2 seconds</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s</td>
<td>23.5 seconds</td>
</tr>
<tr>
<td>glb + jpeg + draco</td>
<td>11.15 seconds</td>
</tr>
<tr>
<td>glb + jpeg + meshopt</td>
<td>12.95 seconds</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>12.61 seconds</td>
</tr>
</tbody>
</table>

Load Time (local disk, no throttling)

<table>
<thead>
<tr>
<th>Format</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + jpeg</td>
<td>5.16 seconds</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s</td>
<td>4.65 seconds</td>
</tr>
<tr>
<td>glb + jpeg + draco</td>
<td>5.69 seconds</td>
</tr>
<tr>
<td>glb + jpeg + meshopt</td>
<td>5.07 seconds</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>4.21 seconds</td>
</tr>
</tbody>
</table>

Memory Usage

<table>
<thead>
<tr>
<th>Format</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>glb + jpeg</td>
<td>834 MB (texture) + 108 MB (geometry)</td>
</tr>
<tr>
<td>glb + ktx2/basis/etc1s + meshopt</td>
<td>105 MB (texture) + 72 MB (geometry)</td>
</tr>
</tbody>
</table>