CITYGML, 3D TILES AND CESIUM
A Data Fusion Symphony

Shehzan Mohammed
Cesium
zmohammed@agi.com
@shehzanm

CESIUM AGI
What is CityGML?

• CityGML is a data model for 3D city models.
  • Open community standard of the OGC.
• Allows hierarchical specification of geometry and attributes.
• Rich semantic information.
  • Describes types of objects.
  • Relationship between objects.

CityGML Model: http://www.geores.de
Limitations of CityGML

- Data exchange format - stored as XML files
- Not suitable for streaming.
  - NYC DoITT CityGML: 12.8GB
  - Berlin CityGML: 49.3GB (Textured)
- Not suitable for rendering.
  - Need significant processing for large datasets.
• Specification for streaming massive heterogeneous 3D geospatial datasets.

• Open Specification and Open Source implementation in Cesium.

• Allows custom tiling schemes.

• Declarative Styling.

• Undergoing OGC Community Standardization process.

https://github.com/AnalyticalGraphicsInc/3d-tiles
Why CityGML to 3D Tiles?

• 3D Tiles is designed to stream data formats like CityGML.
  • “Massive heterogeneous 3D geospatial datasets”

• 3D Tiles is optimized for streaming & rendering.

• Can preserve attribute and semantic information.

• Support for textures.

• Styling and more.

• Added as an in-kind contribution to OGC Testbed 13 (2017).
Demos – New York City DoITT CityGML
Demos - Berlin CityGML
CityGML to 3D Tiles

Parsed Using libCityGML

Cesium 3D Tiling Pipeline

- Per-Object glTF
  - Geometry
  - Materials
  - Textures
  - Metadata
- Terrain Clamping
- Texture Atlas
- Batching & Compositing
- Tiling Algorithms
  - Adaptive Grid
  - Georeferenced

Multithreaded

3D Tiles
Terrain Clamping

• Buildings cannot be directly put on terrain from ellipsoid.
  • They can either become underground or float in the air.

• Sample heights at buildings and offset.
  • Watch out of terrain fall-offs!
Terrain Clamping

NYC Buildings Not Clamped to Terrain

NYC Buildings Clamped to Terrain

Terrain Slope View

Terrain Slope View
Tiling Algorithms

• User experience is only as good as the tiling algorithm.
  • Tiles too large – too heavy for streaming and rendering.
  • Tiles too small – increased streaming overhead.

• Balance between content in top-level tiles vs lower levels.

• Additive vs Replacement.

• Take instanced models into account.

*: Simplification
2D Tiling / TMS / Grid

• Tiled as Quadtrees.
  • Terrain, imagery etc. have existing tiling algorithms.
  • Usually uniform quadtrees.
  • Regular shapes/subdivisions.

• For buildings:
  • May not take cost of tile contents into account.
  • Can create very heavy or light tiles in the same dataset.
    • Downtown vs Suburb vs Rural
2D Tiling / TMS / Grid

Tiles with same area but more geometry

Tiles with same area but less geometry
Adaptive Tiling

- Non-uniform subdivision.
  - Uses number of buildings, tile volume, density etc as heuristics.
  - Early termination, merging up.
- Pick “important” buildings first, higher up in the tree.
  - Visible from further away.
  - Smaller buildings in the lower levels.
  - More suitable for datasets with both batched and instanced models.
- Allows heuristics for instanced models.
  - A single tree may not warrant it’s own tile, but a forest does.
Adaptive Tiling
Adaptive Tiling
Batching and Compositing

• Inefficient to stream each model individually.
• Need to combine similar types of models.
• Combine buildings into Batched 3D Model (B3DM) Tile.
• For instanced models, create Instanced 3D Model Tiles.
  • Keep glTF separate to reduce streaming cost.
• Composite tiles combine batched and multiple instanced tiles.
  • Example: Downtown with buildings, trees, lamps, furniture etc.
Buildings and Trees

CityGML Model: http://www.geores.de
Buildings and Trees

Shapefile / OpenTreeMap Data
Buildings from: data.sfgov.org
Trees from: OpenTreeMap
Buildings and Trees on Terrain

Shapefile / OpenTreeMap Data
Buildings from: data.sfgov.org
Trees from: OpenTreeMap
Optimizing Geometry

• Geometry, although stored as binary in glTF, can be compressed.
  • Both on disk and in memory.

• Many algorithms exist – Quantization, Oct-encoding etc.

• Newcomer: **Draco** ([https://github.com/google/draco](https://github.com/google/draco))
  • Compression library for Meshes and Point Clouds
  • C++ / WebAssembly / Javascript
Optimizing Geometry using Draco

- Extension to glTF 2.0.
- Encode using Javascript wrapper
- Decode on client using WebAssembly
  - Decoded to quantized form.
- In our tiling pipeline:
  - Almost no server side overhead.
  - Small start-up cost on client.
- Big win for hosted datasets!

<table>
<thead>
<tr>
<th>New York City Dataset Payload Type</th>
<th>Tileset Size</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>glTF 2.0 w/o Gzip</td>
<td>3067 MB</td>
<td>1x</td>
</tr>
<tr>
<td>glTF 2.0 w/ Gzip</td>
<td>738 MB</td>
<td>~4x</td>
</tr>
<tr>
<td>glTF 2.0 w/ Draco w/o Gzip</td>
<td>179 MB</td>
<td>~17x</td>
</tr>
<tr>
<td>glTF 2.0 w/ Draco w/ Gzip</td>
<td>149 MB</td>
<td>~20x</td>
</tr>
</tbody>
</table>

NYC CityGML Dataset: 12.8GB (900MB Gzipped)
Optimizing Textures

• Textures make datasets look infinitely better.
• However they come with 2 significant challenges:
  • Increased computation (draw calls).
  • Increased memory usage.
Textures – Optimize Compute

• Solution is to create a Texture Atlas.

• Texture Atlas is a single large texture file that combines many textures.
  • Reduces the compute by a factor of N (number of textures in atlas).

• Remember: Blank spaces use memory!
  • Minimize.
Textures – Optimizing Memory

• Textures are expensive on GPU.
  • A single 1 megapixel RGB texture = 12MB on GPU.
  • 1GB = ~85 such textures

• Berlin CityGML Dataset has ~36GB of JPEG textures.

• Spatial subdivision reduces active loaded size, but not enough.

• Compressing textures (DXT, Crunch etc) is an option.
  • Not universal!
  • Increased processing time – server and client side.
Textures – Optimizing Memory

• Create duplicate levels of tiles – with reduced texture sizes.

• Reducing texture size reduces memory usage exponentially.
  • 1024 x 1024 RGB texture = 12MB on GPU.
  • 512 x 512 RGB texture = 3MB on GPU.
  • 256 x 256 RGB texture = 0.75MB on GPU.

• Highest resolution textures only when camera is near.

• No deterioration in visual experience at runtime.

• Fast on server side.
Textures – Optimizing Memory

Berlin CityGML Model
Textures – Optimizing Memory
Preserving Attribute Metadata

• CityGML has rich metadata information schemas.
• Preserving this information is vital to users.
• We use 3D Tiles Batch Table to preserve all attributes.
3D Tiles Styling

• 3D Tiles provides a rich styling language that allows modifying color and visibility based on properties.

• JSON / Javascript.

Berlin – Style by distance
From Reichstag (bottom-center)

New York – Color by Height

Berlin
Style by Latitude
3D Tiles Styling and Attributes
Finally...

• CityGML to 3D Tiles coming soon to CESIUM ION™.

• Much more on 3D Tiles during

  Data Fusion in 3D Tiles
  by Sean Lilley
  At 2:45pm in Gateway II
  (Next talk in this room)
Questions?

https://cesium.com/

We’re Hiring: https://cesium.com/jobs/

: @shehzanm
: https://github.com/AnalyticalGraphicsInc

Slides: https://cesium.com/presentations/