Rendering the World With 3D Tiles

Peter Gagliardi
3D Graphics Developer, Cesium
About Me

- Peter Gagliardi
- Drexel CS major, class of 2019
- 3D Graphics Software Developer at Cesium since 2019
- Passionate about computer graphics, procedural art, and fractals

Github: @ptrgags
Cesium
- At the intersection of graphics and geospatial
- We render massive, digital representations of the real world
- Based in Center City, Philadelphia

I work on open source projects (CesiumJS) and open standards (3D Tiles)

Philadelphia Photogrammetry
What’s Unique about Cesium

- Unique problem domain
  - The earth is huge - single precision float coordinates cause jitter
  - The earth’s surface is curved - must convert between geospatial coordinate systems
  - The data is huge - Lidar scans and photogrammetry can be many GB or even TB
- Open source roots
- Real-world graphics use cases
- Variety of platforms, (WebGL, Unreal Engine, VR, etc)
Part 1: How to Render a City?
How to render...

...buildings across the entire globe?

...massive photogrammetry or point clouds?

...highly-detailed 3D models?
Enter 3D Tiles

● Open standard for streaming massive heterogeneous 3D geospatial datasets
  ○ Massive - real-world data can be gigabytes to terabytes in size
  ○ Heterogeneous - triangle meshes, point clouds, terrain
  ○ 3D geospatial - Digital representation of the Earth
● Implementations in CesiumJS, Cesium for Unreal

3D Tiles Specification
3D Tiles Reference Card
Real-world examples

- NASA Perseverance Rover
- Digital Twins of Australia - Blog Post

Also: Aerial imagery of Ukraine visualized in Cesium Stories
2.5D is not 3D

2.5D - Can render the road OR the bridge but not both at once

3D - Can render both the bridge and the road underneath
Bounding Volumes

- Simple geometry (e.g. spheres, boxes) that contains more complex geometry (e.g. a triangle mesh)
- Used for efficiently determining what’s in view
- Game engines also use bounding volumes to speed up collision detection
Bounding Volume Hierarchies (BVHs)

- Tree of bounding volumes with a spatial coherence property
- Spatial coherence - content is completely contained within its bounding volume, and within the bounding volumes of each ancestor in the tree
- Many types:
  - Quadtrees
  - Octrees
  - K-d trees
  - Loose quadtrees/octrees
- Used for efficient culling of complex scenes
Hierarchical Level of Detail (HLOD)

- Tree of 3D content
  - Leaves are full-resolution
  - Each parent is a simplification of its children
- For content near the camera, render the high-resolution version
- For content far away, render the low-res version
- Similar in concept to texture mipmaps, just for 3D geometry
- Also similar to LODs in game engines, just for a whole tree of contents

Live Demo Link
Choosing an LOD

Diagram from the 3D Tiles Reference Card

Depends on Geometry

The geometric error is measured in meters, comparing the simplified geometry to the real geometry:

- Geometric error
- Camera view
- Screen resolution

The screen space error (SSE) is measured in pixels:

- SSE: about 4 pixels
- SSE: about 2 pixels
- SSE: about 1 pixel
For a large scene, back-face culling isn’t enough for performance

Test bounding volumes from the BVH against the view frustum. If completely outside, no need to render!

Live Demo Link
All Together: 3D Tiles

- **Tileset** – the entire tree of tiles
- **Tile** - A single node in the tree, has a bounding volume, content (a 3D model), and geometric error
- A tileset is both a BVH and the contents are organized by HLOD

Live Demo Link
Examples Revisited

Cesium OSM Buildings

Nearmap NYC Photogrammetry

Space Shuttle Discovery - Full article
Part 2: Rendering glTF models
What is glTF?

- OBJ files – simple to write, but large and slow to parse for detailed models
- glTF
  - Geometry (triangles, points, normals, etc.)
  - Materials, scene graph, animations, and more
- 3D model format designed for efficient streaming and runtime rendering
- Open standard maintained by the Khronos Group (same organization behind OpenGL/WebGL and Vulkan)
- Key model format in 3D Tiles
- WebGL calls can be expensive, we want to minimize how many we make.
- Instead of rendering every primitive in a separate draw call, combine geometry into one big array.
- Each part of the geometry gets a unique ID.
- In the shader, the ID can be used to look up rendering details (colors, material properties, transformations).
Same concept as in the picking assignment for this class

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 draw call
  ○ One difference from the assignment – we only render a small viewport around the mouse, typically 3x3 pixels!

Read rendered pixels to determine which feature was clicked

Limitations – doesn’t scale well to millions of features, no room for additional info in the shader.
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).
Physically-Based Rendering

- More realistic lighting of models
- Two main artist workflows:
  - Metallic/roughness – easier to use
  - Specular/glossiness – harder to get right, more expressive
- Image based lighting (IBL) is used for metallic surfaces but fast enough for real-time use
- glTF has many new PBR extensions
  - Clearcoat
  - Sheen
  - Transmissive
  - Volumetric
Part 3: What’s new in CesiumJS/3D Tiles?
We want to support:
- Larger datasets
- Behaviors, info encoded in data
- Accessibility
- Richer context for simulations and games

3D Tiles Next is a collection of new extensions for 3D Tiles & glTF
- Structural Metadata
- Implicit Tiling
Feature Ids

- Closely related to the batching concept discussed earlier
- More ways of defining features
  - Per texel
  - Per vertex
  - Per GPU instance
- Useful for classifying models
- We also want to support storing arbitrary integer IDs that could be used e.g. to query an API/database
- Enrich data with real-world context through metadata properties

CDB Yemen Sandcastle
Custom Shaders

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

Explicit Tiling: Provides flexibility of tileset structure

Every bounding volume is described explicitly

Implicit Tiling: Provides random access and efficient storage for quadtrees/octrees

Root Bounding volume is implicitly subdivided for each descendant tile

Template URI patterns are listed once

Subtree files contain availability bitstreams
Bonus: An Artistic Use for 3D Tiles Metadata
Fractal Generator - **chaos-game-3d**

- Personal passion project from my spare time combining my interest in fractals with exploring 3D Tiles
- Based on the [Chaos game algorithm](https://en.wikipedia.org/wiki/Chaos_game) for generating fractals
- Metadata from the algorithm (iteration numbers, cluster IDs, UV coordinates) + custom shaders opens up many methods for coloring and animation.
My fractal generator generates tilesets offline. This can potentially get expensive to compute.

With an elapsed time uniform, I can show/hide points based on the iteration number.

This way I can view the motion of the points without doing any expensive calculations at runtime!
What we look for in candidates

- Passion projects
  - Please send us GitHub links/blog posts!
- Backend, frontend, 3D, XR
- Interest in graphics and performance
- Experience with good software development practices (git, unit testing, CI) a plus

From https://twitter.com/CesiumJS/status/1214665776969502725
We’re Hiring

https://cesium.com/careers
Contact Information / Resources

Peter Gagliardi

peter@cesium.com

GitHub, Twitter: @ptrgags

Cesium Website

CesiumJS GitHub Repo

3D Tiles GitHub Repo