1. Motivation
2. Examples
3. Principles
4. Tiling algorithms
5. Tile payloads
6. Declarative styling
7. Future work

Image from Bentley ContextCapture 3D Tiles export, https://www.linkedin.com/pulse/contextcapture-web-publishing-cesium-aude-camus
Sean Lilley

Cesium Developer
Patrick Cozzi

Started Cesium

Books on Virtual Globes, WebGL, OpenGL

Contribute to Open Standards and Formats

Teach Computer Graphics

Open-Source Geospatial Community Service

http://www.seas.upenn.edu/~pcozzi/
1. Motivation
1.1 A brief personal history

- 1981 – birth
- 2008 – HLOD thesis
- 2011 – Streaming imagery
- 2014 – Point clouds
- 2015 – 3D Tiles
- 2016 – Almost done
2008 – MS Thesis: Visibility Driven Out-of-Core HLOD Rendering
2011 – Streaming imagery in Cesium

Without tile processing
With unbounded cache
Without tile priority details
Without layers

Update

* Throttled texture creation

* Walk tile tree

* Throttled HTTP request
  Async response

Image queue

Render
  * draw back-to-front
  * clear render queue

Tile tree

Texture pool

Have tile

Need tile

Request queue

Render queue

With anisotropic filtering
2014 – Point cloud prototype
2015 – 3D Tiles

Geospatial Datasets

3D Tiles

Optimized for streaming

3D Engines
and other tile formats
1.2 Industry trends
Ease of distribution


Bottom image from http://cesiumjs.org/2015/01/27/Migrating-from-Earth-to-Cesium/
Cost and availability of supply

Open data policies

Crowdsourcing

nationalmap.gov.au

www.opentreemap.org
Cost and availability of supply

Ease and cost of data acquisition

For more examples, see http://cesiumjs.org/presentations/RenderingMassiveGeospatialDatasetsInCesium.pdf
Industry trends

easy distribution +
large, low cost supply =

disruption

(pardon the business-ish speak)
Industry trends

Why haven’t we disrupted 3D yet?

Something is missing…
Massive heterogeneous 3D geospatial datasets and other 3D engines.
Massive heterogeneous 3D geospatial datasets → Open specification for streaming massive heterogeneous 3D geospatial datasets → 3D Tiles → CESIUM and other 3D engines
Massive heterogeneous 3D geospatial datasets

// Complete Cesium code example to load a 3D Tiles tileset

```javascript
var viewer = new Cesium.Viewer('cesiumContainer');

viewer.scene.primitives.add(new Cesium.Cesium3DTileset({
    url : 'path-to-tileset.json'
}));
```
Every building in Manhattan to every satellite in space

1.1 million buildings in New York City

Point cloud of the Church of St Marie at Chappes, France.

High resolution terrain and imagery

Trees in Philadelphia

3D vector data
Foreshadowed at Web3D 2014

Future 3D Standards?

- CZML
- Terrain, Buildings, Point Clouds, Trees, …
  - Streaming, HLOD, culling metadata, crack filling, imposters, compression, semantics, incremental updates, textures, normals, procedural shading, optimized for runtime!
3D Tiles in the ecosystem

3DPS

- data discovery
- payload can be 3D Tiles

CityGML
- KML, GML, COLLADA, LAS...
- convert to 3D Tiles for streaming massive datasets
- preserve attributes

Khronos Group
- GTF
- individual 3D models
- can be payload in a 3D Tiles tile

WMTS, ...
- 3D Tiles complement existing 2D standards

Cesium®
- massive 3D datasets with attributes and styles
- optimized for runtime streaming

and other 3D engines

Cesium®
2. Examples
3D Tiles Showcases

https://youtu.be/KoGc-XDWPDE
• 1.5 million buildings with 3D Tiles
• Buildings clamped to terrain
• Separate terrain and imagery layers
• **22.2 GB** KML/COLLADA converted to **1.5 GB** 3D Tileset (0.97 GB gzipped)

**KML + COLLADA**
NYC: CityGML and OSM

- ~1.1 million buildings
- Mouse over highlight
- **Dynamic styling** with per-building attributes
- Separate imagery layer
- **12.84 GB** CityGML converted to **1.85 GB** 3D Tileset (**727 MB** gzipped)

CityGML

OSM extract

.json / COLLADA

See http://cesiumjs.org/NewYork/
- Stream 100K+ buildings
- Mouse over highlight
- Per-building attributes
  - Display
  - Style
- Combine with terrain, imagery, and vector data

See http://www.cybercity3d.com/
• Textured buildings
• Mouse over highlight
• Per-building attributes
  • Display
  • Dynamic Styling
• Combine with terrain and vector data

See http://www.cityzenith.com/
Just presented **Using glTF for streaming CityGML 3D City Models** here at Web3D.
ContextCapture reality mesh export

- Buildings, terrain, trees, etc.
- Publish data and app to the Web
- Overlay vector data

See https://www.linkedin.com/pulse/contextcapture-web-publishing-cesium-aude-camus
- High-resolution terrain, buildings, and imagery
- Runtime annotation

See http://cesiumjs.org/2016/03/08/Vricon-and-3D-Tiles/
• Separate tileset each for buildings, trees, windmills, etc.
• 3DPS for data discovery
3. Principles
How much geospatial data?

Terabytes is not uncommon

NVIDIA GeForce 980 GTX – 4 GB
A few examples

- **2006** – Google: 70 TBs of compressed satellite imagery in BigTable
- **2010** - Oregon Department of Geology and Mineral Industries Lidar Program Data: 290 billion points
- **2012** – Microsoft: 302 TBs of imagery
- **2014** – MapBox adds 1 million NYC buildings to OSM
2D tiling has limited use in 3D

- OK for imagery
- OK-ish for terrain
- What about 3D buildings, point clouds, vector data, and massive models?
2D tiling for terrain and imagery

https://youtu.be/f6NEi0F_PVw
2D tiling has limited use in 3D

- Sub-optimal subdivisions for non-uniform datasets
- No 3D subdivision, e.g., for point clouds
- Doesn't easily handle objects that overlap two tiles
- 3D requires multiple LODs in the same view
  - Need error metric for LOD selection
  - Need to avoid cracking visual artifacts
• The foundations for 3D tiling were created by
  – Graphics research
  – Movie industry
  – Game industry
• 3D Tiles brings these techniques to geospatial
Hierarchical Level of Detail (HLOD)

```plaintext
visit(tile)
{
    if (computeSSE(tile) <= pixel tolerance)
    {
        render(node);
    }
    else
    {
        foreach (child in tile.children)
        {
            visit(child);
        }
    }
}
```

Images: [Cozzi08], Code: [Ulrich02]
Hierarchical Level of Detail (HLOD)

• Easy to add culling. Exploit spatial coherence to discard entire sub-trees.
  – View frustum
  – Occlusion / horizon / fog
  – Backface
  – Need tight bounding volume
3D Tiles use case

• Massive datasets
  – Tiling, LOD, cache management culling, batching, instancing, compression

• Heterogeneous datasets
  – Data fusion, common algorithms

• Non-uniform datasets
  – Require a variety of spatial data structures

• Interaction and styling
  – Per-feature properties/attributes/metadata and uniquely style individual features
3D Tiles

• **Spec**
  – Spatial data structure defined in JSON
  – Tile formats: binary with embedded JSON
    • Ready to render
  – Declarative styling
  – In progress on [GitHub](https://github.com)

• **Software ecosystem**
  – Cesium implementation in progress
  – Several significant adoptions
3D Tiles

• Spatial data structure supports many tiling approaches
  – Server/tool can decide what is optimal for a dataset
• Client/runtime traverses generic spatial data structure
Tilesets and tiles

- **tileset.json**
  - Defines spatial layout for the tileset
  - Points to external tiles

- **Tiles**
  - Contain the tile’s actual content
  - Binary + embedded JSON
3D Tiles

Tile
- bounding volume
  - box
  - region
  - sphere
- geometric error
- refine
- content
  - bounding volume (box, region, or sphere)
  - url
    - Separate file with tile contents, streamed on demand
- children[]
tileset.json

```json
{
  "properties": {
    "Height": {
      "minimum": 1,
      "maximum": 241.6
    }
  },
  "geometricError": 494.50961650991815,
  "root": {
    "boundingVolume": {
      "region": [-0.0005682966577418737, 0.8987233516605286, 0.00011646582098558159, 0.8998683398325034, 0, 241.6]
    },
    "geometricError": 268.37878244706053,
    "content": {
      "url": "0/0/0.b3dm",
      "boundingVolume": {
      /* ... */
    },
    "children": [..]
  }
}
```

- **Tileset metadata**
- **Geometric error if tileset is not rendered**
- **Bounding volume of root tile**
- **Geometric error if only root tile is rendered**
- **Url to tile content**
- **Bounding volume of content**
- **Children of this tile**
Bounding volumes

See http://cesiumjs.org/2015/06/24/Oriented-Bounding-Boxes/
Spatial coherence

- Child’s **content** must be in parent’s bounding volume
  - Does not imply child’s bounding volume is in parent’s bounding volume
Spatial data structures

Grid
Spatial data structures

**Quadtree**

**Tight-fitting Quadtree**
Spatial data structures

Adaptive loose-ish Quadtree
Spatial data structures

Octree

K-d tree
Geometric error

• How many “meters wrong” is the image of a tile’s children (and their sub-trees) are not rendered?
• At runtime, used to compute screen-space error based on view parameters
Geometric error: terrain example

Images: [Ring13.c], Algorithm: [Garland95]
Refinement: replacement vs. additive

**Replacement**

\[ \mathcal{O}(n\log n) \text{ tileset} \]

```c
visit(tile)
{
    if (computeSSE(tile) \leq\ pixel tolerance)
    {
        render(node);
    }
    else
    {
        foreach (child in tile.children)
        {
            visit(child);
        }
    }
}
```

**Additive**

\[ \mathcal{O}(n) \text{ tileset} \]

```c
visit(tile)
{
    render(tile);
    if (computeSSE(tile) > pixel tolerance)
    {
        foreach (child in tile.children)
        {
            visit(child);
        }
    }
}
```
Content bounding volume
Additive refinement

http://youtu.be/1useSwpuM8w
Tileset of tilesets

tiles.json

empty

empty

tiles.json
tiles.json

tiles.json
tiles.json
Tile Formats

- Small, **fast to download**
  - Compression
- **Fast to decode**, create WebGL resources
  - “Ready to render” with batching or instancing
- Include **per-feature** properties/attributes/metadata
- Utilize glTF
  - Benefit from glTF’s open-source ecosystem
glTF

- GL Transmission Format
- Efficient **open-standard** runtime 3D model format by **Khronos**
  - Khronos also publishes COLLADA, WebGL, OpenGL, ...
- **JSON + binary** geometry/textures/animations
- Strong brand and **open-source** community
  - Converters
  - Validators
  - Pipeline tools
  - 3D engine importers

[Khronos.org/glTF](https://khronos.org/gltf)
3D Tiles tile formats

- Batched (.b3dm)
- Instanced (.i3dm)
- Point Cloud (.pnts)
- Vector (.vctr)
- Composite (.cmpt)
- Add your own!
Tile formats

- Batched (.b3dm)
Tile formats

- Instanced (.i3dm)
Tile formats

- Point Cloud (.pnts)
Tile formats

- **Vector (.vctr)**

Work in progress. Data thanks to the Federal Office of Topography swisstopo!
Tile formats

- Composite ( .cmpt )
Declarative styling

```json
{
    "show": "${Area} > 0",
    "color": {
        "conditions": {
            "${Height} < 60": "color('#13293D')",
            "${Height} < 120": "color('#1B98E0')",
            "true": "color('#E8F1F2', 0.5)"
        }
    }
}
```
4. Tiling algorithms
Discrete Level of Detail

https://youtu.be/TCzR6u9hd9w
Sample tilesets: https://github.com/AnalyticalGraphicsInc/3d-tiles-samples
Discrete LOD

- Quick and dirty. Easy
  - Just decimate model at different LODs
- **Popping** artifacts
  - Future work: alpha blending or vertex morphs
- Downloads **duplicate** data at each LOD
  - True of replacement refinement in general
  - Future work: just send deltas
- Not appropriate for massive models
Hierarchical Level of Detail

https://youtu.be/tBG4bWB2_-w
HLOD with replacement refinement

{  
  "asset": { /* ... */ },  
  "geometricError": 1024,  
  "root": {  
    "boundingVolume": { "region": [ /* ... */ ] },  
    "geometricError": 512,  
    "refine": "replace",  
    "content": {  
      "url": "0/0/0.b3dm"  
    },  
    "children": [  
      "boundingVolume": { "region": [ /* ... */ ] },  
      "geometricError": 256,  
      "content": { /* ... */ },  
      "children": [ /* ... */ ]  
    },  
    "children": [  
      "boundingVolume": { "region": [ /* ... */ ] },  
      "geometricError": 256,  
      "content": { /* ... */ },  
      "children": [ /* ... */ ]  
    },  
    // ...  
  }  
}
HLOD with replacement refinement
HLOD with replacement refinement
HLOD with replacement refinement
HLOD with replacement refinement
HLOD with replacement refinement
HLOD with replacement refinement
HLOD with replacement refinement

- Appropriate for massive models
- Tile with quadtree, octree, ...
- Union of leaf tiles is original model
- Root tile is drastically decimated model
- Interior tiles are decimated subset of model with decreasing geometric error farther down the tree
HLOD with replacement refinement

- **Popping** artifacts
  - Future work: alpha blending or vertex morphs
- **Cracks** – often use skirts to fill
- Downloads *duplicate* data at each LOD
HLOD with additive refinement

http://youtu.be/1ZwzCBXPE-0
HLOD with additive refinement

```json
{
    "asset": { /* ... */ },
    "geometricError": 1024,
    "root": {
        "boundingVolume": { "box": [ /* ... */ ] },
        "geometricError": 512,
        "refine": "add",
        "content": {
            "url": "0/0/0.pnts"
        },
        "children": [{
            "boundingVolume": { "box": [ /* ... */ ] },
            "geometricError": 256,
            "content": { /* ... */ },
            "children": [ /* ... */ ]
        }, {
            "boundingVolume": { "box": [ /* ... */ ] },
            "geometricError": 256,
            "content": { /* ... */ },
            "children": [ /* ... */ ]
        }],
        // ...
    }
}
```
HLOD with additive refinement
HLOD with additive refinement
HLOD with additive refinement
HLOD with additive refinement
HLOD with additive refinement

• Does not download duplicate data
• Children can be rendered before all siblings are downloaded
• Does not require decimation
  – Great for lots of simple models, e.g., many building datasets
• Can render too much when zoomed in close
LOD with imposters

https://youtu.be/OSJLQFD01IU
using custom shader for glTF, could also be .vctr
LOD with imposters

- Different tree levels have different data
- In general, can switch between any representation
  - Point ➔ billboard ➔ 3D model
  - Point ➔ polyline ➔ polygon
  - Extruded footprint ➔ building with rooftop ➔ textured building
  - Low-resolution textured building ➔ Highly textured building
Tight-fitting quadtree

https://youtu.be/_M3Kiu3GvQw
Tight-fitting quadtree
Tight-fitting quadtree

```
{
  "asset": { /* ... */ },
  "geometricError": 12,
  "root": {
    "transform": [ /* ... */ ],
    "boundingVolume": { "region": [ /* ... */ ] },
    "geometricError": 12,
    "refine": "replace",
    "children": [
      {
        "boundingVolume": { "region": [ /* ... */ ] },
        "geometricError": 12,
        "children": [
          {
            "boundingVolume": { "region": [ /* ... */ ] },
            "geometricError": 0,
            "content": {
              "url": "trees.i3dm"
            }
          }
        ]
      }
    ]
  }
}
```

Non-leaf tiles have the same geometric error

No content, used for culling

Leaf tiles have content
Tight-fitting quadtree

- Interior tiles accelerate culling
- Tight subdivision avoids wasted space and deep trees possible with a traditional quadtree
Grid

```
{
  "asset": { /* ... */ },
  "geometricError": 30,
  "root": {
    "transform": [ /* ... */ ],
    "boundingVolume": { "region": [ /* ... */ ] },
    "geometricError": 30,
    "refine": "replace",
    "children": [
      {
        "boundingVolume": { "region": [ /* ... */ ] },
        "geometricError": 0,
        "content": {
          "url": "tile1.b3dm"
        }
      },
      // ...
    ]
  }
}
```

Root tile with many children
No interior tiles, no accelerated culling
Separate tilesets

https://youtu.be/HSiCrb8U26A
Separate tilesets

- **Independently** tile optimally for the input dataset
- Only stream layers that are **visible**
- Alternatively use composite tiles for a heterogeneous tileset
  - Reduces number of requests
  - Less optimal subdivision
Request Volumes

https://youtu.be/PgX756Yzjf4
When the viewer is inside this bounding volume, the tile is rendered

https://github.com/AnalyticalGraphicsInc/3d-tiles/issues/101
Tile expiration

https://youtu.be/JIPLRtFzn5g
Tile expiration

```
{
  "asset": { /* ... */ },
  "geometricError": 346.4,
  "root": {
    "expire": {
      "duration": 5
    },
    "boundingVolume": {
      "sphere": [ /* ... */ ]
    },
    "geometricError": 0,
    "refine": "add",
    "content": {
      "url": "points.pnts"
    }
  }
}
```

Re-request **points.pnts** every five seconds
Tile expiration

- Client re-requests expired tile and sub-tree
- Work in progress
  - Duration or explicit date
  - Store in tile so it can change

5. Tile payloads
Tile Format Principles

• Same spirit as glTF
  – Utilize open-source glTF ecosystem

• Fast to download
  – Binary blob (little endian) with embedded JSON (utf-8 string)
  – Compression

• Fast to decode, “ready to render”
  – Flatten node hierarchy, texture atlases, pre-batch, vertex cache optimize
  – Pre-triangulate
    – Minimize WebGL draw calls

• Preserve per-feature properties
• Often served pre-gzipped
Compression

• Optional **quantization** for positions
  – Store \( x, y, z \) each as uint16
  – Stays **compressed in GPU** memory
  – **Decode for free** in the vertex shader
  – ~1.5m error in 100,000m scale
  – Tiles get smaller farther down the tree

• `glTF WEB3D_quantized_attributes` extension

• Future
  – Context-aware, e.g., Open3DGC
  – Variable bit based on bounding volume
  – YCoCg for color?
  – What do **you suggest**?
Compression

- Optional **oct-encoding** for normals and texture coordinates
  - For lighting:
    - **oct16P** – 2 bytes for 3D normal
      - Mean error: 0.31485, max error: 0.63575 degrees
  - For rotation (**i3dm**):
    - **oct32P** – 4 bytes for 3D normal
      - Mean error: 0.00122, max error: 0.00246 degrees
  - Stays **compressed in GPU** memory
  - **Decode for cheap** in the vertex or fragment shader

Figure from http://jcgt.org/published/0003/02/01/
Tile formats

- Batched (.b3dm)
- Instanced (.i3dm)
- Point Cloud (.pnts)
- Vector (.vctr)
- Composite (.cmpt)
Batched (.b3dm)

- Pre-batched binary glTF
- Buildings, terrain, ...
- Per-feature `batchId` to identify features in a batch

20-byte header

- `magic` (unsigned char[4])
- `version` (uint32)
- `byteLength` (uint32)
- `batchLength` (uint32)
- `batchTableByteLength` (uint32)

Body

- `batchTable`
- Binary glTF

External data
Batched (.b3dm)

- Pre-batched binary glTF
- Per-feature `batchId` to identify features in a batch
- Example vertex buffer:

<table>
<thead>
<tr>
<th>Positions for all features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normals for all features</td>
</tr>
<tr>
<td><code>batchId 0</code></td>
</tr>
<tr>
<td><code>batchId 1</code></td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td><code>batchId n</code></td>
</tr>
</tbody>
</table>

Points to Batch Table
Batched (.b3dm)

- At runtime, in a vertex or fragment shader, use `batchId` to lookup color/show in a texture
- Fast styling – just change the texel
Batched (.b3dm)

- `batchId` points into Batch Table to access per-feature properties, which can be used for
  - Styling
  - Display
  - Web services

```json
{
    "id": ["unique id", "another unique id"],
    "displayName": ["Building name", "Another building name"],
    "yearBuilt": [1999, 2015],
    "address": [{"street": "Main Street", "houseNumber": "1"},
                 {"street": "Main Street", "houseNumber": "2"}]
}
```
Instanced (.i3dm)

- Single model with slight variations, called **instances**
- Trees, traffic lights, bolts, …
- Efficiently rendered with WebGL instancing
Instanced (.i3dm)

36-byte header (first 20 bytes)

- magic (unsigned char[4])
- version (uint32)
- byteLength (uint32)
- featureTableJSONByteLength (uint32)
- featureTableBinaryByteLength (uint32)

36-byte header (next 16 bytes)

- batchTableJSONByteLength (uint32)
- batchTableBinaryByteLength (uint32)
- gltfByteLength (uint32)
- gltfFormat (uint32)

Body

- featureTable
- batchTable
- url (UTF-8) or Binary glTF

External data
Instanced (.i3dm)

- Feature table contains **per-instance**:
  - Positions (optionally **quantized**)
  - Rotations (optional, optionally **oct-encoded**)
  - Scale (optional)
  - batchId (optional)
Instanced (.i3dm)

- Feature table semantics reference binary data
  - **Example:** `POSITION_QUANTIZED` points to `INSTANCES_LENGTH` array of `uint16[3]`
Instanced (.i3dm)

- **glTF**
  - url to `external` glTF to share among tiles
  - `embed` in tile to minimize number of requests
- Use `composite` tile for multiple models
Point Cloud (.pnts)

- Still working on this, should stabilize in 1-2 months
  - Follow https://github.com/AnalyticalGraphicsInc/3d-tiles/issues/22
Point Cloud (.pnts)

- Feature Table (not shown in diagram) will contain
  - Positions (optionally **quantized**)
  - Color (optional, RGB or RGBA, compression)
  - Normals (optional, optionally **oct-encoded**)
  - Scale (optional)
  - batchId (optional)
• At runtime, generate vertex shader for
  – Point size attenuation
  – Point culling
  – Show/color/size based on declarative styling – executing on the GPU **in parallel**!
  – Lighting, hidden surfaces, …
  – Edge-preserving blur
Vector (.vctr)

- Still working on this, should stabilize in 2-3 months
  - Follow https://github.com/AnalyticalGraphicsInc/cesium/compare/3d-tiles...vector-tiles-polygons

Work in progress. Data thanks to the Federal Office of Topography swisstopo!
Vector (.vctr)

- Polygons, polylines, points (labels/billboards)
  + extrusions, Cesium-style geometries
- Feature and Batch Tables like .i3dm and .pnts
- Pre-triangulate, ready to render
• Combine heterogeneous tiles into one tile
  – Composites of composites too

• Examples
  – Buildings (b3dm) + trees (pnts)
  – Multiple tree species (i3dm)
• Users turn layers on/off a lot?
  – Separate tilesets
  – Also allows more optimal subdivision
• Users look at heterogeneous datasets
  – Composite tiles
  – Fewer requests
  – Likely less optimal subdivision
6. Declarative styling
Principles

• Declaratively describe per-feature:
  – Show/hide
  – Color (\texttt{color.rgb})
  – Translucency (\texttt{color.a})

• Write expressions using
  – Subset of \texttt{JavaScript}
  – \texttt{Augmented} for styling

• Access \texttt{per-feature properties} (from the Batch Table)

• Applying new styles should be \texttt{fast}
// Applying a style to a tileset

var viewer = new Cesium.Viewer('cesiumContainer');

viewer.scene.primitives.add(new Cesium.Cesium3DTileset({
    url: 'path-to-tileset.json'
})));

tileset.style = new Cesium.Cesium3DTileStyle({
    "show": true,
    "color": "rgb(100, 255, 190)"
});
• **Built-in Color type with CSS-like constructor functions**

```
"color" : "color('#BAA5EC')" // 3 or 6-digit hex, optional alpha
"color" : "color('cyan', 0.5)"
"color" : "rgb(100, 255, 190)" // also rgba()
"color" : "hsla(0.9, 0.6, 0.7, 0.75)" // also hsla()
```
Styling language

- Access to per-feature properties using `{{}}`

  ```json
  "show" : "${ZipCode} === '19341'"
  
  "color" : "(${{Temperature} > 90) ? color('red') : color('white'))"
  ```
Styling language

- **Built-in RegExp type for string compares**

  
  "show" : "(RegExp('^Chest').test(${County})) &&
              (${YearBuilt} >= 1970)"
• **Use conditions to create color ramps and other functions**
  – Execute like an if...else series

```json
"color" : {
    "conditions" : {
        "($\{Height\} >= 100.0)" : "color('#0000FF')",
        "($\{Height\} >= 70.0)" : "color('#00FFFF')",
        "($\{Height\} >= 50.0)" : "color('#00FF00')",
        "($\{Height\} >= 30.0)" : "color('#FFFF00')",
        "($\{Height\} >= 10.0)" : "color('#FF0000')",
        "($\{Height\} >= 1.0)" : "color('#FF00FF')"
    }
}
```
Styling language

• Create non-visual expressions with `meta`.

```javascript
"meta" : {
    "description" : "Hello, ${featureName}.",
    "volume" : "${height} * ${width} * ${depth}"
}
```

• Evaluate per feature:

```javascript
var feature = scene.pick(/* screen x, y position */);
if (Cesium.defined(feature)) {
    var meta = tileset.style.meta;
    console.log("Description: " + meta.description.evaluate(feature));
    console.log("Volume: " + meta.volume.evaluate(feature));
}
```
Cesium styling API

- Change per-feature properties at runtime
  - The style is automatically reevaluated

```javascript
// 3D Tiles style:
"color" : "(${Temperature} > 90) ? color('red') : color('white')"

// In JavaScript using the Cesium API:
feature.setProperty('Temperature', 91);

// Can also get properties, and get/set color/show
feature.getProperty('Temperature'); // 91
feature.color; // Cesium.Color.RED
```
Cesium styling API

• Allows custom styles, e.g., to implement another styling language

```javascript
var style = new Cesium.Cesium3DTileStyle();
style.show = {
    evaluate: function(feature) {
        return true;
    }
};
style.color = {
    evaluateColor: function(feature, result) {
        return Cesium.Color.fromBytes(255, 255, 255, 255, result);
    }
};
tileset.style = style;
```
7. Future work

We need your input!
Explicit tiling schemes

- For well known tiling schemes, e.g., quadtree
  - Allow random access, e.g., $z/y/x$
  - Avoid large tileset.json and tilesets of tilesets
  - Inflexible, requires same tile format throughout tile

```json
{
    "asset": { /* ... */ },
    "tilingScheme": "quadtree",
    "format": "b3dm",
    "boundingVolume": {
        "region": [ /* ... */ ] // could default to global
    }
}
```

github.com/AnalyticalGraphicsInc/3d-tiles/issues/92
Time-dynamic tiles

- Similar to streaming video
  - “keyframe animation for massive models”
- Introduce a new composite tile (.cmpt)
  - Each tile in temporal composite also has a time stamp
  - Temporal composite (or tileset.json) can point to previous/next tiles in the time series, perhaps like a skip list
  - Future future work: only store deltas

| time interval | times 0..n | tiles 0..n | previous/next tiles |

github.com/AnalyticalGraphicsInc/3d-tiles/issues/102
Download multiple children in one request

- Fewer files, fewer requests
  - Especially useful for replacement refinement when all sibling tiles need to be downloaded
  - Client could use prefetch strategies
  - Similar to glTF buffer and bufferView

```
"tiles": {
  "level1-id": "0/0/0.b3dm"
},
// ...
"content": {
  "tile": "level1-id",
  "byteOffset": 0,
  "byteLength": 1024
}
```

github.com/AnalyticalGraphicsInc/3d-tiles/issues/9
New tile formats

• Finish vector (.vctr)
• Consider OSM tile, higher-order surfaces
  – Generate geometry at runtime from OSM constructs, e.g., roofs
  – Compute/memory trade-off
• Terrain
• Create your own tile format!

github.com/AnalyticalGraphicsInc/3d-tiles/tree/master/TileFormats/OpenStreetMap
Optimize additive refinement

- Add spatial data structure to each tile
- Each tile in the nested spatial data structure has \((\text{offset}, \text{count})\)
- Only render subset of tile that is visible
- Natural extension to \(\text{content.boundingVolume}\)

github.com/AnalyticalGraphicsInc/3d-tiles/issues/11
Expand declarative styling

- Expand standard library, e.g.
  - `Boolean inShadow();`
  - `JulianDate simulationTime;`

- Refer other features, e.g.,
  - `color : ramp(palette, distance(${thisFeature.location}, ${anotherFeature.location}))`

[github.com/AnalyticalGraphicsInc/3d-tiles/issues/2](https://github.com/AnalyticalGraphicsInc/3d-tiles/issues/2)
Package tileset in a single file

• Could be as simple as a database with tileset.json and rows with
  – **key**: relative path to tile
  – **value**: tile contents

[github.com/AnalyticalGraphicsInc/3d-tiles/issues/89]
How can we work together?
Consider **3D Tiles** or **Cesium** as a platform for your research.

- Tiling algorithms
- New tile formats
- Compression algorithms
- Rendering optimizations

• Tell us about it!
• We can showcase your work with **3D Tiles** or **Cesium**. Email Sarah Chow, **schow@agi.com**
We’re hiring!

Software Developers and Geospatial Data Wranglers

cesiumjs.org/jobs
Join us for the Cesium BOF

Tuesday, 10:30am-12:00pm. Room 203B

- Cesium State of the Union. Patrick Cozzi, Cesium
- 3D Tiles and OGC's 3D Portrayal service standard. Ralf Gutbell, Fraunhofer
- Making Drone Data Useful in Cesium. Chris Cooper, Propeller
- Publishing Reality Meshes to Web from Bentley ContextCapture using Cesium 3D Tiles. Makai Smith, Bentley
- Baking AO in the glTF Pipeline. Gary Li, Cesium
Check out 3D Tiles

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

Contact

Patrick Cozzi
pcozzi@agi.com
@pjcozzi

Thanks to