

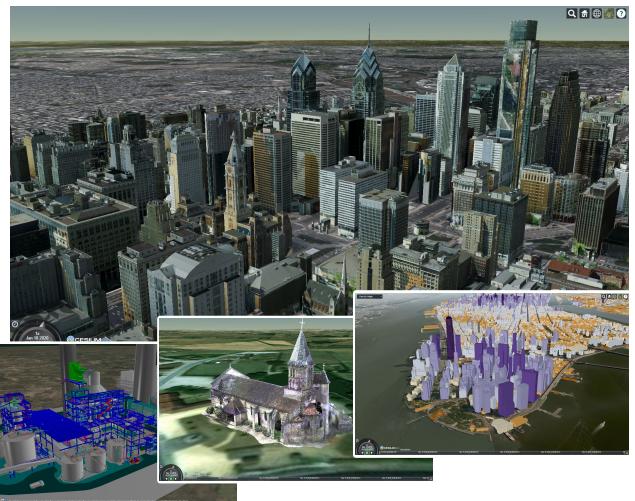


# 3DTiles

3D Tile是一种共享开源的规范，它可以在桌面端，WEB端以及移动端实现大规模三维地理空间内容的可视化，融合，交互和分析。3D Tiles是基于glTF（高效的流媒体和渲染的开放标准）构建的，用于通过流媒体高效的渲染3D模型和场景。

三维地理空间内容，包括摄影测量/大型模型、BIM/CAD、三维建筑、实例化特征和点云，都可以转换为3D Tile，并合并为一个数据集，以实现无缝的性能和实时分析，包括测量、可见度分析、样式和排序。

3D Tile的基础是一种空间数据结构，可以实现分层细节（HLOD）。因此，只有可见的瓦片被流化和渲染，提高了整体性能。



本文概述了3D Tiles规范所支持的主要概念：

- Tileset和Tile的一般概念，以及它们如何使大规模的数据集组织成可以合理流传输的元素成为可能
  - 1. 一览：一个Tileset的例子
  - 2. Tilesets 和 Tiles
- 3D Tile如何实现用于高效渲染和互动的，分层的空间数据结构。
  - 3. 包围体(Bounding Volumes)
  - 4. 空间数据结构
- 分层细节水平(HLOD)的概念，使其有可能在任何比例下平衡渲染性能和视觉质量
  - 5. 几何误差
  - 6. 细化策略
- 如何实现3D Tile背后的概念，以实现高效渲染和互动
  - 7. 3D Tile渲染优化
  - 8. 3D Tile中的空间查询
- 3D Tile中不同Tile格式的技术细节
  - 9. Tile格式：简介
  - 10. Tile格式
- 用附加功能扩展基本规格的可能性
  - 11. 扩展
- 通过基于元数据的内容样式化(styling)，实现信息可视化的能力。
  - 12. 样式声明(Declarative Styling)
- 地理空间坐标系和几何数据压缩等基本要素如何在3D Tile中得到整合
  - 13. 常见定义

如果你正在寻找3D Tileset来起步，Cesium ion (<https://cesium.com/ion>) 也允许用户上传自己的数据来创建、主持和传输3D Tile。



© 2020 by Cesium GS, Inc. Made available under a Creative Commons Attribution 4.0 License (International): <https://creativecommons.org/licenses/by/4.0/>



原文：[Cesium https://github.com/CesiumGS/3d-tiles](https://github.com/CesiumGS/3d-tiles)

翻译：[ElevenIjusee](https://github.com/ElevenIjusee) 更多GIS开源内容见<https://github.com/ElevenIjusee>

# 1. 概览：一个简单的Tilesheet例子

3D Tiles的核心元素是tilesets。一个tileset是一组tiles，以分层结构组织。tileset是用JSON描述的。下面是一个tileset的简单例子，介绍了最重要的概念和元素。每个概念在下面的章节中会有更详细的解释。

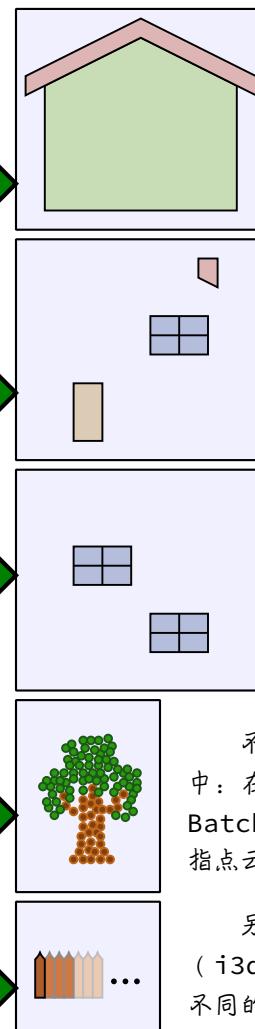
```
{  
  "asset": { ... },  
  "properties": { ... },  
  "geometricError": 100,  
  "root": {  
    "geometricError": 20,  
    "boundingVolume": {  
      "region": [ ... ]  
    },  
    "refine": "ADD",  
    "children": [  
      {  
        "geometricError": 10,  
        "boundingVolume": { ... },  
        "content": {  
          "uri": "house.b3dm"  
        },  
        "children": [  
          {  
            "geometricError": 5,  
            "boundingVolume": { ... },  
            "content": {  
              "uri": "detailsA.b3dm"  
            },  
            {  
              "geometricError": 5,  
              "boundingVolume": { ... },  
              "content": {  
                "uri": "detailsB.b3dm"  
              },  
            }  
          ]  
        },  
        {  
          "geometricError": 10,  
          "boundingVolume": { ... },  
          "content": {  
            "uri": "tree.pnts"  
          },  
        },  
        {  
          "geometricError": 10,  
          "boundingVolume": { ... },  
          "content": {  
            "uri": "fence.i3dm"  
          },  
        },  
        {  
          "geometricError": 10,  
          "boundingVolume": { ... },  
          "content": {  
            "uri": "external.json"  
          },  
        }  
      ]  
    ]  
  }  
}
```

Each tile may refer to an external tileset. This allows combining multiple smaller tilesets into larger ones.

主tileset JSON包含资产的基本描述和一般属性。

几何误差（第5节）用于确定何时应该渲染根tile。

每个tile都包含一个包围盒（第3节），它包含了该tile和所有子tile的内容。它还包含一个几何误差（geometric error），决定什么时候应该渲染子tile。



在这个例子中，第一个子tile指的是一个低细节的三维模型。本例中模型的瓦片格式是Batched 3D Model（第10.2节）。

子tile包含模型的额外细节。当要求更高层次的细节时，它们会被呈现出来。

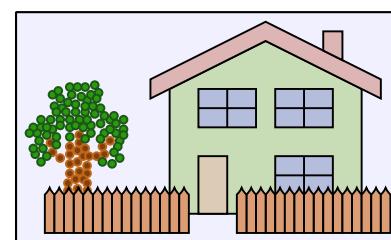
在这个例子中，子tile的内容将被添加到父节点的模型的低级表示中。

或者，子tile可以包含更详细的模型，以取代低层次的模型。这就是3D tile支持的两种重构策略（第6节）。

不同的tile格式可以结合在一个tileset中：在这个例子中，第一个子tile指的是一个Batched 3D模型。根节点的另一个子tile是指点云模型（10.4节）

另一个tile是一个实例化三维模型（i3dm）（第10.3节），一个简单的几何体在不同的位置被多次渲染。

当tileset被渲染时，tile在适当的细节水平上被组合，以产生最终的渲染结果。

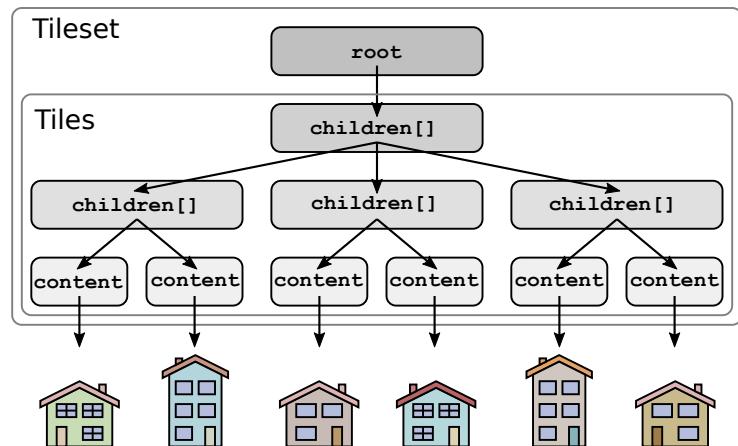


## 2. Tilesets 和 Tiles

tileset是一组tile，它们被组织成一个分层的数据结构，称为结构树。tileset本身包含根tile，每个tile可以有子tile。

一个tile也可以指代另一个tileset。这使得我们可以灵活地、分层次地将tileset组合成更大的瓷砖tileset。

每个tile都可以指代可渲染的内容。这些内容可以有不同的格式，例如，可以代表地形、三维模型或点云。第9节 "tile格式" 中解释介绍了各种可能的tile格式。



tileset和tile都是用JSON描述的。tileset的JSON文件包含tileset本身的基本信息和tile的描述。

### Tileset 属性：

<b>Root tile:</b>	tileset的根属性是代表tile层次结构的根的tile。
<b>Geometric error:</b>	geometric Error属性用于量化如果不对tileset进行渲染会出现的视觉误差。当视觉误差超过某个阈值时，就会考虑对tileset及其包含的tile进行渲染。关于如何使用几何误差的细节在第5节 "几何误差" 中给出。
<b>Property summaries:</b>	tile的可渲染内容可能相关的属性。例如，当tile包含建筑物时，那么每个建筑物的高度可以存储在tile中。tileset的属性对象包含了tileset中所有tile的选择属性的最小值和最大的值
<b>Metadata:</b>	有关3D Tile版本的信息和应用程序特定的版本信息可以存储在tileset的属性中。

### Tile 属性：

<b>Content:</b>	与瓦片相关的实际可渲染内容是通过content属性中的URI引用的。
<b>Children:</b>	tile的分层结构被构建为结构树：每个瓦片都可能有子级，这些子级是一个名为children的数组中的tile。
<b>Bounding Volume:</b>	每个tile都有一个相关的包围体，不同类型的包围体可以存储在boundingVolume属性中。包围盒的可能类型在第3节 "包围体" 中介绍。每个包围体都包含了tile的内容和所有子片的内容，产生了一个空间上连贯的包围体层次构。
<b>Geometric error:</b>	tile的可渲染内容可能有不同的细节层次。对于tiles，几何误差 (geometricError) 属性量化了tile中的内容与最高级别的细节相比的简化程度。如第5节 "几何误差" 所述，几何误差被用来确定何时应考虑对子tile进行渲染。
<b>Refinement strategy:</b>	当具有一定细节水平的tile的视觉误差超过阈值时，就会考虑对子tile进行渲染。来自子tile的额外细节被纳入渲染过程的方式是由细化属性决定的。第6节 "细化" 中解释了不同的细化策略。

### 3. 包围体

每个tileset是一组以分层方式组织的tile，每个tile都有一个相关的包围体。这就产生了一个分层的空间数据结构，可用于优化渲染和高效空间查询。此外，每个tile可以包含实际可渲染的内容，这些内容也有一个包围体。与tile的包围体相比，内容包围体只与内容紧密相连，并可用于可见性查询和视图轮廓的剔除，以进一步提高渲染性能。

3D Tiles格式支持不同类型的包围体，以及不同的分层组织策略。

#### 包围体的类型

3D Tiles支持不同类型的包围体，因此可以选择最适合基础数据结构的类型。

```
"boundingVolume": {  
  "sphere": [  
    10, 5, 15,  
    140.0  
  ]  
}
```



包围球是一个简单的包围盒，可以进行简单而有效的交叉测试。它是由其中心位置和半径来定义的。

```
"boundingVolume": {  
  "box": [  
    0, 0, 10,  
    20, 0, 0,  
    0, 30, 0,  
    0, 0, 10  
  ]  
}
```



定向包围盒将包围体与几何体更紧密地结合在一起，特别是对于像CAD模型这样的常规技术结构。它是由盒子的中心位置和三个三维向量来定义的。这些矢量描述了X、Y和Z轴的方向和半长。

```
"boundingVolume": {  
  "region": [  
    -1.319700,  
    0.698858,  
    -1.319659,  
    0.698889,  
    0.0,  
    20.0  
  ]  
}
```



region特别适合于地理信息系统，因为region的两侧都有经纬度坐标，分别为该地区的西、南、东、北，以及最低和最高高度。

纬度和经度以EPSG 4979中规定的WGS 84为基准，以弧度为单位。

最低和最高高度以米为单位。高于或低于WGS84椭圆体。

关于WGS84和EPSG 4979的细节可在第13节 "通用定义" 中找到。

## 4. 空间数据结构

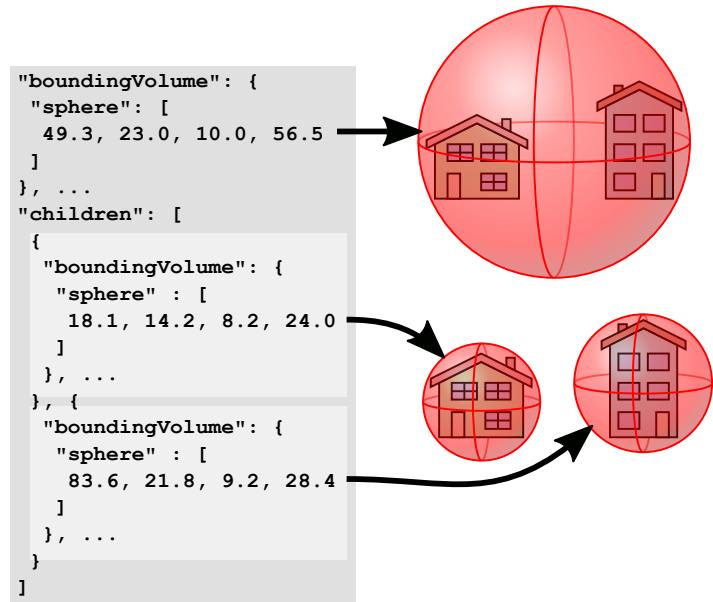
tileset中的tile被组织在一个树形数据结构中。每个tile都有一个相关的包围体。这允许对不同的空间数据结构进行建模。下一节将介绍不同类型的空间数据结构，它们都可以用三维瓦片建模。运行时引擎可以通用地使用空间信息，以优化渲染性能，与空间数据结构的实际特征无关。

### 空间相干性 ( Spatial Coherence )

3D Tiles中所有类型的包围体和空间数据结构都具有空间相干性，这意味着父tile的包围体总是包围着它所有子tile的内容。

空间相干性对于保守的可见性测试和相交测试是至关重要的，因为它保证了当一个对象不与一个tile的包围体相交时，那么它也不会与任何子tile的内容相交。.

相反，当对象确实与一个包围体相交时，那么子代的包围体的子体也会被测试是否相交。这可以用来快速修剪层次结构的大部分，只留下一些leaf tile，在那里对象必须测试与实际tile内容几何体的交集与实际tile内容的几何形状相交。



### 空间数据结构的类型

可以通过不同的方法来构建具有包围体的分层空间数据结构。这些方法通常会产生不同类型的空间数据结构，这取决于用于构建的确切策略。

一个简单的空间层次结构可以通过沿某些轴线递归分割瓦片的内容来构建，直到满足一个停止的标准。当内容在每一级只沿单一轴线分割时，那么结果就是一个k-d树。3D Tiles还支持多向k-d树，其中每一层都有沿一个轴的多个分割。当内容沿x、y和z轴在每一层被分割时，那么结果是一个八叉树。当内容在内容边界体积的中心被分割时，那么结果是一个统一的八叉树 (uniform octree)。当内容在一个不同的点上被分割，那么结果就是一个非均匀的八叉树 (non-uniform octree)。

在层次结构的每一级，包围体可以从父节点的包围体中纯粹地在空间上创建。或者，每个节点的包围体可以从节点的实际内容中计算出来。这对稀疏的数据集特别有用，因为它能确保在层次结构的每一层都有紧密的包围体。

| 有时不可能在不分割内容的单一feature (模型)的情况下分割一个tile。因此，有可能构建松散的八叉树和松散的k-d树，其中子代的包围体会重叠。这些树仍然保持空间相干性，因为父包围体仍然包围着所有子的内容。

在最通常的情况下，tileset可以被划分为一个非统一的网格。由于3D Tiles的空间层次结构中的每个节点可以有任意数量的子节点，而且非leaf tile不需要包含可渲染的内容，所以网格单元仍然可以被组织成一个层次结构，以支持分层拔取。

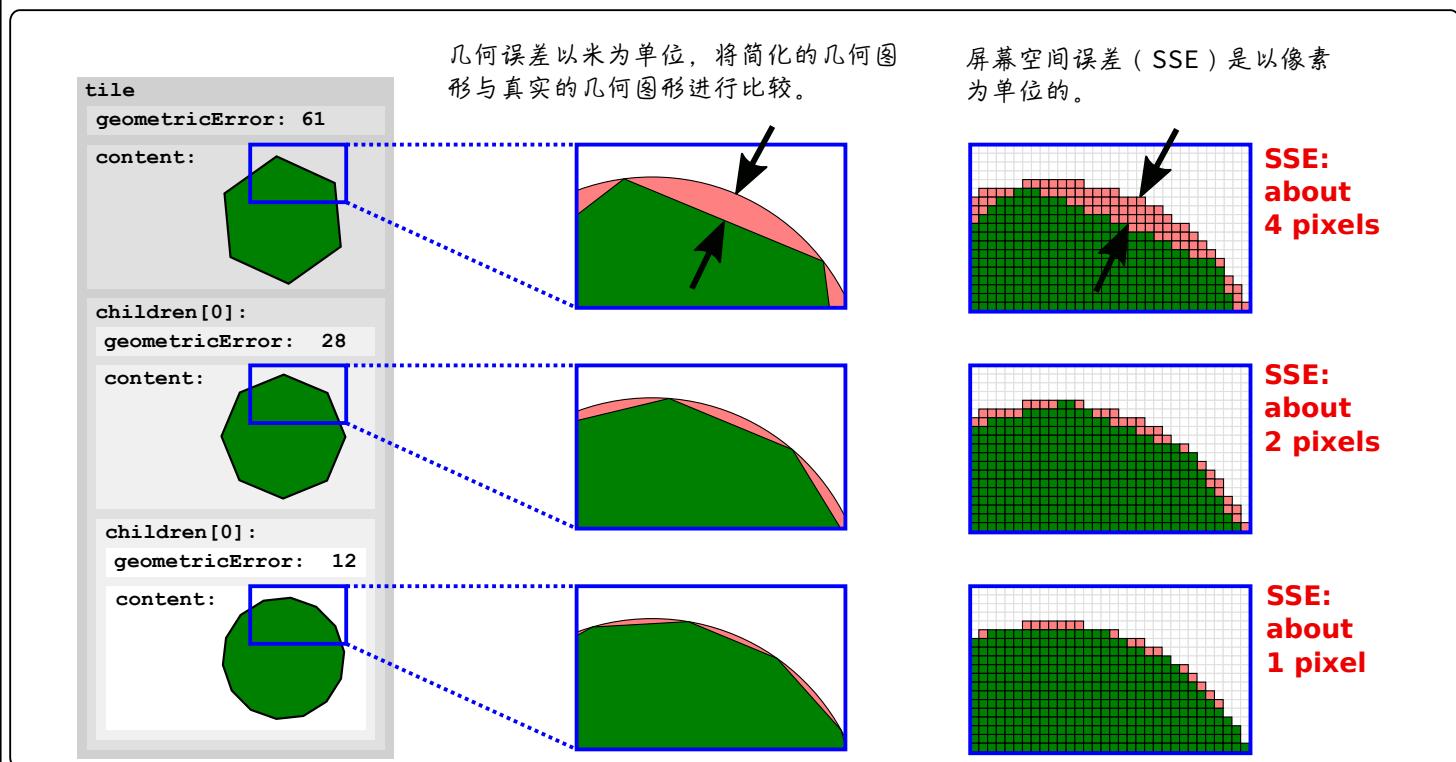
3D Tiles中的通用空间数据结构允许对所有这些构建方法的结果进行建模。每个tile是层次结构的一个节点，可以选择引用几何内容，也可以只存储一个包围其所有下级内容的包围体，而且不同类型的包围体可以存储在一个tileset中。

## 5. 几何误差

3D Tiles的目标之一是合理地将大量的数据集流向正运行的引擎，并且仍然允许运行时合理地渲染这些内容。因此，tileset中tile的分层结构包含了分层细节级别（HLOD）的概念：分层结构顶端的tile包含了低层次的可渲染内容的表示。而子tile则包含更高层次的内容。渲染运行时可以动态地选择在性能和渲染质量之间取得最佳平衡的细节级别。

决定哪个级别的细节应该被渲染的关键是几何误差。每个tileset和每个tile都有一个geometricError属性，它量化了简化的几何图形与实际几何图形相比的误差。

运行时将这个几何误差转化为屏幕空间误差（SSE）。SSE量化了几何误差在屏幕上的可见度，以像素为单位。



### 5.1 计算屏幕空间误差 (SSE)

当SSE超过一定的阈值时，运行时将渲染更高的细节水平。对于一个tileset，几何误差被用来决定是否应该渲染根tile。对于一个tile，几何误差用于确定是否应该渲染该tile的子代。

为了实现这一目标，运行时必须计算出给定几何误差的屏幕空间误差。实际的屏幕空间误差（以像素为单位）将取决于视图配置--即观看者的位置和方向--以及最终图像将被渲染的分辨率。

因此，SSE的计算必须考虑到这些因素。具体的实现方式也取决于所使用的视图投影的类型。但是对于一个标准的透视投影来说，计算SSE的一个可能的方法是如下。

$$sse = (\text{geometricError} \cdot \text{screenHeight}) / (\text{tileDistance} \cdot 2 \cdot \tan(\text{fovy} / 2))$$

其中geometricError是存储在瓦片组或瓦片中的几何错误，screenHeight是渲染屏幕的高度，单位是像素，tileDistance是瓦片与眼点的距离，fovy是视锥在y方向的开口角。

## 6. 细化策略

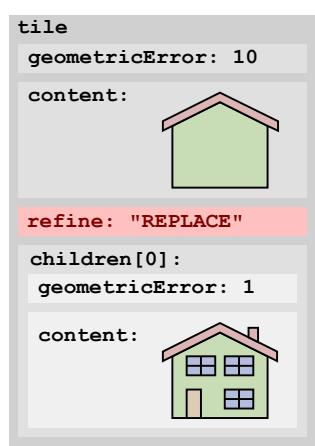
当一个tileset被渲染时，运行时会遍历tile的层次结构，检查每个tile的几何误差，并计算相关的屏幕空间误差。当这个屏幕空间误差超过某个阈值时，运行时就会递归地考虑对子tile进行渲染。包含可渲染内容的子tile具有更高的细节水平和更小的屏幕空间误差。

子tile的内容可以通过两种细化策略中的一种来提高细节水平。细化策略由瓦片的细化属性决定，可以是REPLACE或ADD。

### Replacement:

子tile包含了内容的完整表示，具有更高的细节水平。

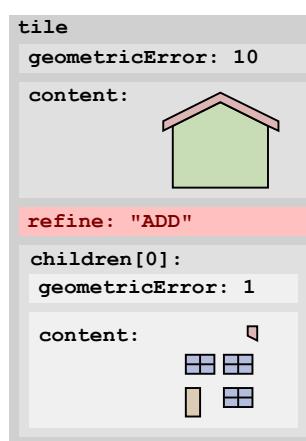
当子tile被渲染时，这个内容将取代父tile的内容。



### Additive:

子tile包含父tile内容的额外细节

当子tile被渲染时，这些内容将被渲染在父tile的内容之上。



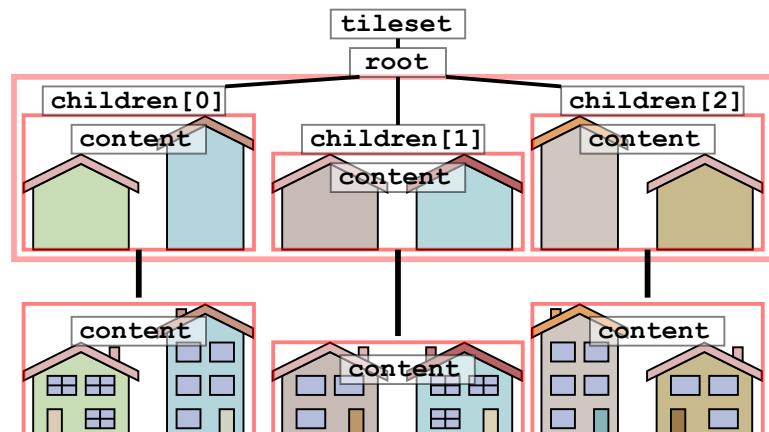
refine属性可以为每个tile单独提供，但只在tileset的根tile中需要。如果不给它，该策略将从父tile中继承。

## 7. 3D Tiles的渲染优化

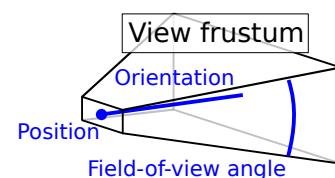
3D Tiles是为高效绘制和流处理海量异构数据集而设计的。下面是一个关于3D Tiles如何被运行时用来交互式地渲染一个巨大的tileset的总结。它显示了tileset和tile的分层结构、相关的包围体、几何误差的概念以及不同的细化策略如何共同发挥作用。

本例中的tileset包含一组建筑物。每个元素的包围体显示为红色。

tileset包含根tile，而根tile又包含三个子tile。每个子tile都包含可渲染的内容。在这个例子中，每个tile的内容包括两个建筑，以Batched 3D模型的形式存储。在这个层面上，内容是以低水平的细节存储的，这意味着tile的几何误差很高。另外，每个子代都以更高细节级别存储相同内容的子代，从而具有更低的几何误差。

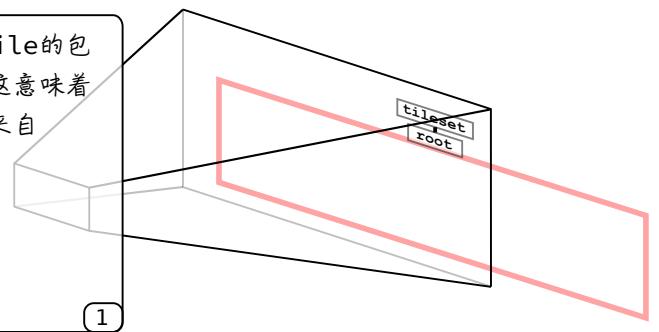


在渲染过程中，运行维持着一个由摄像机的位置、方向和视域角度来定义来定义的视锥。视图视锥可以测试与tileset和tile的包围体的交点。



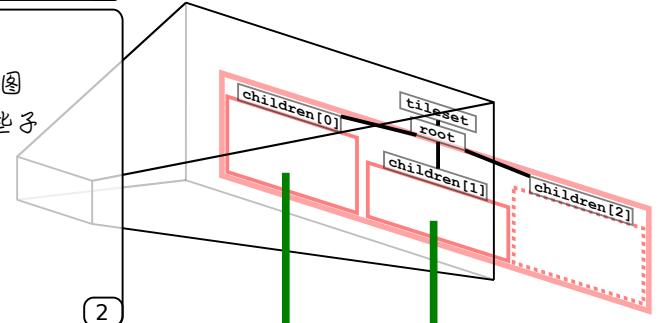
## 7.1. 3D Tiles渲染优化的例子

最初，运行时加载主tileset JSON，并测试视图视锥与根tile的包围体的交集。在这个例子中，视锥确实与根tile的包围体相交，这意味着该tile被考虑进行渲染。此时，没有渲染的内容还没有被加载：来自tileset JSON的信息是足够确定是否有东西要被渲染。

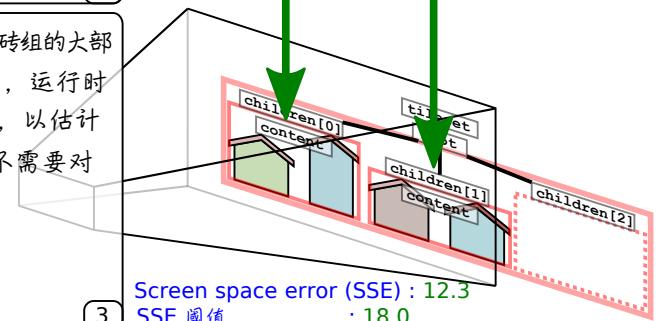


然后，运行时可以测试子tile的包围体是否与视图截锥相交。

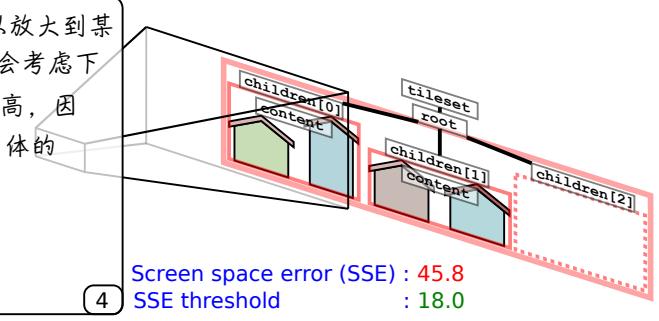
在这个例子中，三个子tile中的两个tile的包围体确实与视图截锥相交。这意味着这些子tile的内容被考虑用于渲染。只有这些子tile的内容必须被加载。



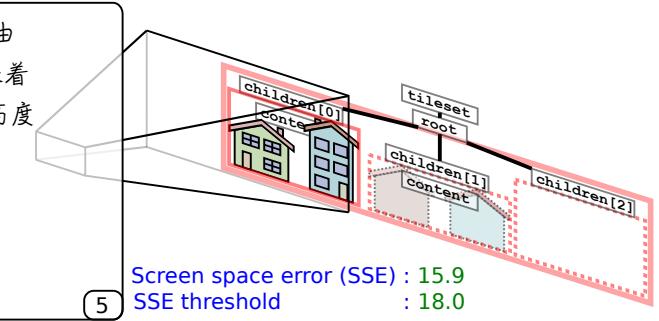
在这个例子中，渲染内容是一个低细节的建筑物。因此，即使瓷砖组的大部分都是可见的，由于其复杂度低，运行时也能合理地渲染内容。在渲染期间，运行时观察与渲染贴图相关的几何误差的影响：它被转换为屏幕空间误差，以估计视觉表现的质量。只要不超过阈值（在本例中可能是18.0），就不需要对渲染的内容进行重新调整。



然后，用户可以与渲染的tileset进行互动。例如，用户可以放大到某一建筑物。这将导致屏幕空间误差增加，当它超过某个阈值时，就会考虑下一级tile的渲染。这些tile包含相同内容的表示，但细节程度更高，因此，几何误差更小。运行时只需加载具有与新视图视锥相交的包围体的tile的内容。



具有较高细节水平的内容被加载，并根据所选择的细化策略进行渲染。由于其较低的几何误差，它使屏幕空间误差下降到阈值以下，这意味着更高的视觉质量。但现在只有小部分的tileset是可见的，具有高度细节的内容仍然可以被正确地渲染出来。



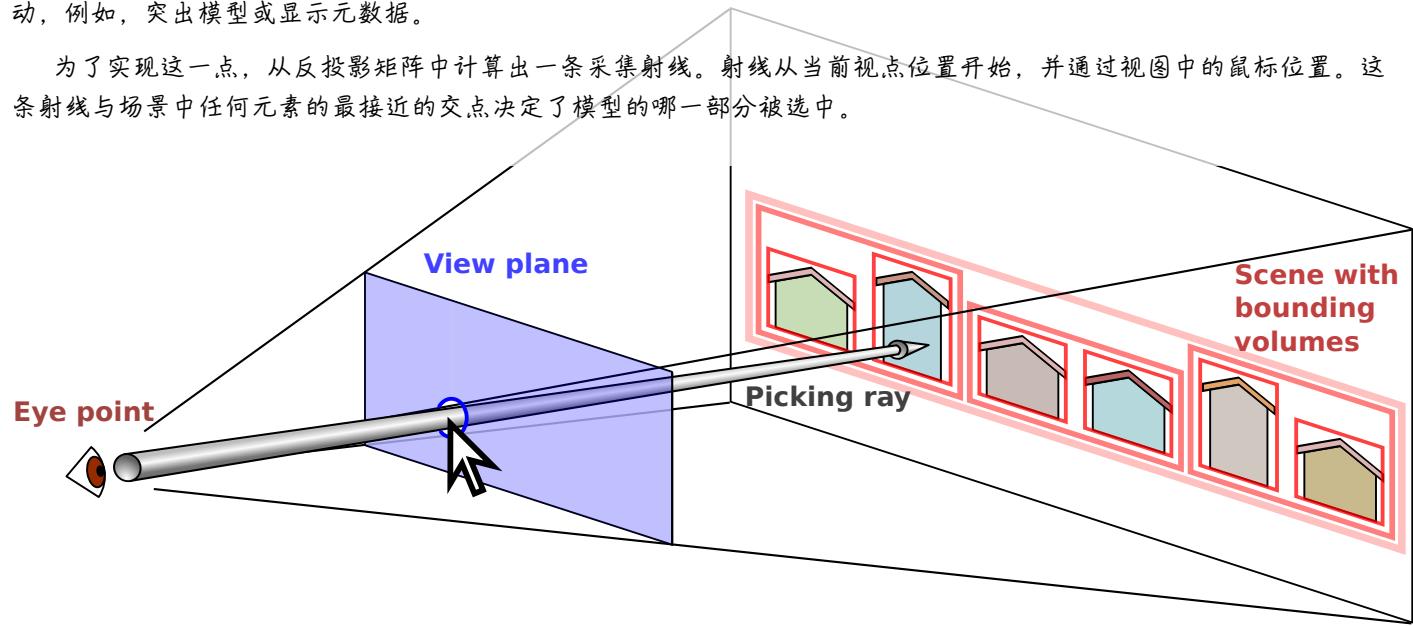
这个例子显示了3D Tiles是如何在任何比例下平衡渲染性能和视觉质量的。运行时可以检测到视图视锥是否与瓦片的包围体相交，只有在这种情况下，tile内容才会被加载。运行时最初可以显示细节程度最低的tile内容，只有在屏幕空间误差超过阈值时——例如，当用户密切放大数据集的一个feature时，才会按要求下载和渲染细节程度更高的内容。

## 8. 3D Tile中的空间查询

3D tiles中带有包围体的分层结构允许高效的空间查询。

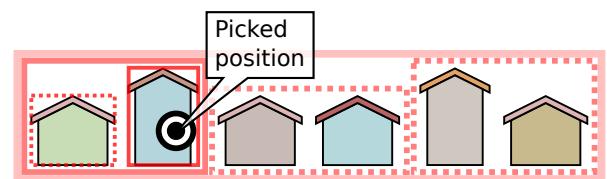
这种空间查询的一个例子是射线投射：当运行渲染瓦片集时，用户可以通过拾取单个模型或feature与场景互动，例如，突出模型或显示元数据。

为了实现这一点，从反投影矩阵中计算出一条采集射线。射线从当前视点位置开始，并通过视图中的鼠标位置。这条射线与场景中任何元素的最接近的交点决定了模型的那一部分被选中。



用射线找到被选中的tile的伪代码

```
function intersect(ray, tile) {
  if (ray.intersects(tile.boundingVolume)) {
    if (tile.isLeaf) {
      return tile;
    }
    for (child in tile.children) {
      intersection = intersect(ray, child);
      if (intersection) return intersection;
    }
  }
  return undefined;
}
```



对交叉点的测试从渲染的tileset的根tile开始。当射线与该tile的包围体相交时，它将被测试是否与每个子tile相交。当发现与一个leaf tile的交集时，可以测试这个tile内容的实际几何数据与射线的交集，以确定实际拾取位置。

在相交测试期间，树的空间相干性使其有可能快速排除层次结构中没有相交的部分：当一个tile的包围体没有相交时，那么这个tile的子代的相交测试可以被跳过。在这个例子中，用虚线表示的物体与拾取射线不相交，而且遍历可能就此停止

对于某些类型的tile，知道某个tile的内容被鼠标点击后可能不适合：在像Batched 3D Models (第 10.2 节) batched Point Clouds (第 10.4 节) 这样的tile格式中，多个不同的模型或模型的一部分 (features) 可以合并成一个几何体。在这些情况下，几何体的顶点会被扩展为一个批处理的ID (batch ID)，它可以识别出该feature。这使我们有可能确定被鼠标点击的实际feature。

## 9. Tile格式: 简介

tile的可呈现内容由tile JSON的一部分URI引用。对于不同的模型类型，这些可呈现的内容可以以不同的格式存储：

- **Batched 3D Model:** 异构模型，如纹理地形或3D建筑
- **Instanced 3D Model:** 同一个三维模型的多个实例
- **Point Clouds:** 大量的点

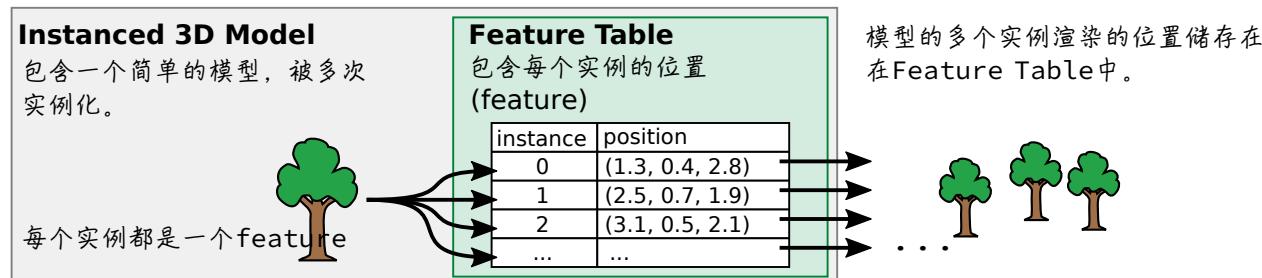
一个tileset可以包含任何tile格式的组合。额外的灵活性是通过允许将不同格式的tile组合成复合tile来实现的（第10.5节）。

一个tile的可渲染内容包含不同的features。对于Batched 3D Model，几何体的每一部分都可以是一个feature。例如，当几个建筑物被组合在一个Batched 3D Model中时，那么每个建筑物可能是一个feature。对于Instanced 3D Model，每个实例都是一个feature。对于点云，有两种选择：一个feature可以是一个单一的点，也可以是代表模型中可识别部分的一组点。

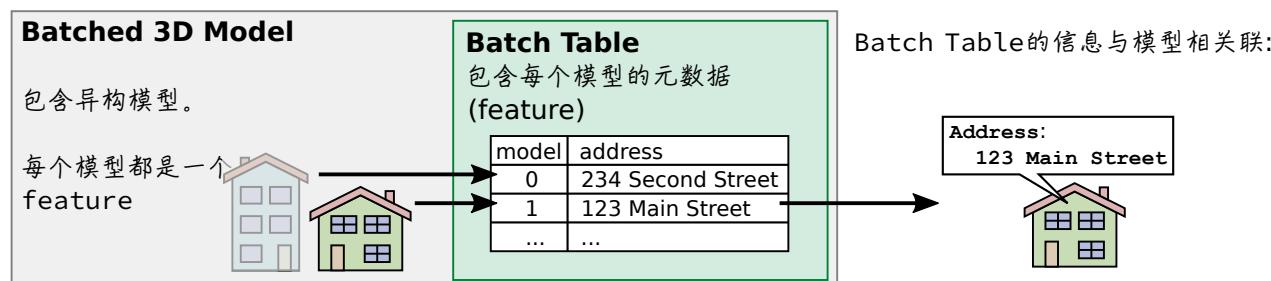
### Feature Table 和 Batch Table

所有tile格式（除复合tile）都有共同的元素，Feature Table和Batch Table。Feature Table和Batch Table的确切内容取决于tile格式，但它们的结构和布局对所有tile格式都是一样的。

Feature Table包含渲染feature所需的属性。例如，在一个实例三维模型中，实例的位置被存储在特征表中：



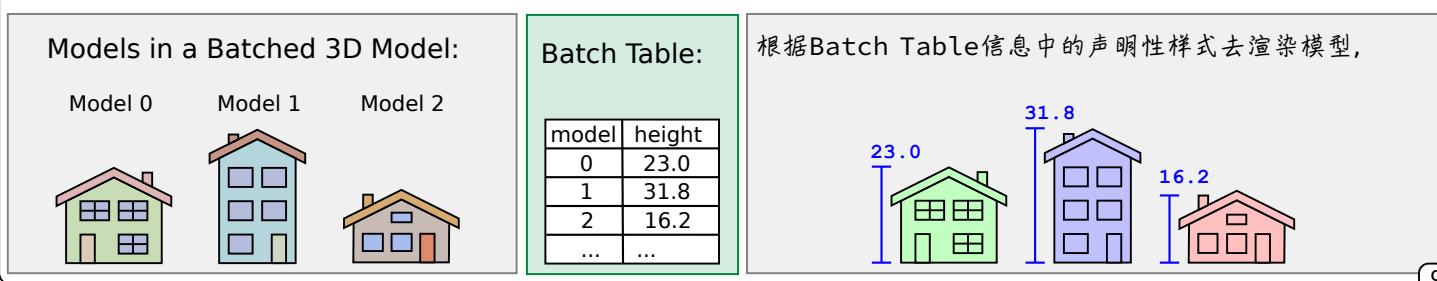
Batch Table可以包含每个feature的额外的、应用特定的属性。例如，在Batched 3D模型中，与每个模型相关的元数据被存储在Batch Table中：



### 展望：声明式样式 ( Styling )

存储在Batch Table中的信息对于渲染来说不是直接需要的。但是Batch Table通常包含元数据，可以用于声明性的样式设计，如第11节所示。

例如，Batch Table可能包含一组出现在tile中的建筑物的高度信息。这些信息可以在3D Tiles styling语言中访问，以修改模型的外观。例如，根据建筑物的高度，可以用不同的颜色来渲染。



## 10. Tile格式

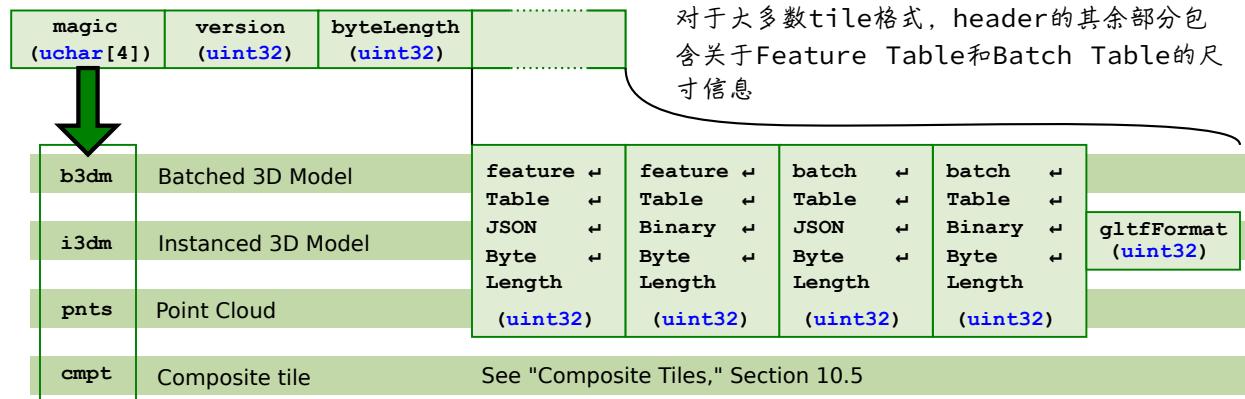
tile的实际可渲染内容被存储为一个二进制的blob。这个blob由一个包含结构信息的header和一个包含实际有效载荷的body组成



### Tile格式的Header

每个tile的header由4个magic bytes组成。这些字节代表一个4个字符的字符串，决定了tile的格式。它们后面是tile格式的版本号，以及tile数据的总长度，包括标题，单位是字节。

header的确切大小、内容和结构取决于tile的格式，因此在读取magic bytes后就可以知道。tile格式的确切数据布局和它们的标题在下面解释每个单独的tile格式的章节中显示。

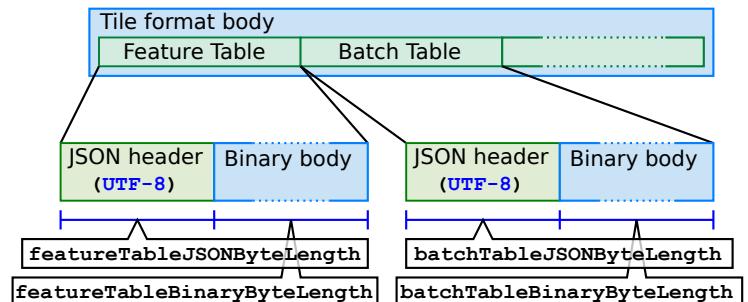


`featureTableJSONByteLength`, `featureTableBinaryByteLength`, `batchTableJSONByteLength`和`batchTableBinaryByteLength`描述了Feature Table和Batch Table各自部分的大小(以字节为单位)，并参考包含实际表数据的tile格式主体。

### Tile的Body

tile格式的body包含tile数据的实际有效载荷。对于所有的tile格式(除了复合tile)，这个body可能包含一个Feature Table和一个Batch Table，以及其他为tile格式所规定的二进制数据。

Feature Table和Batch Table分别由一个JSON header和一个二进制body组成。每个元素的长度由tile格式header的信息决定。

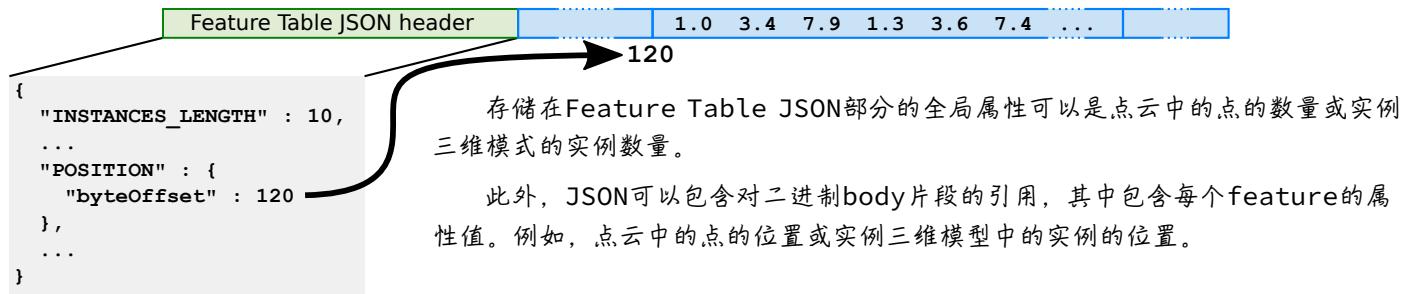


## 10.1. Tile格式：Feature Table 和 Batch Table

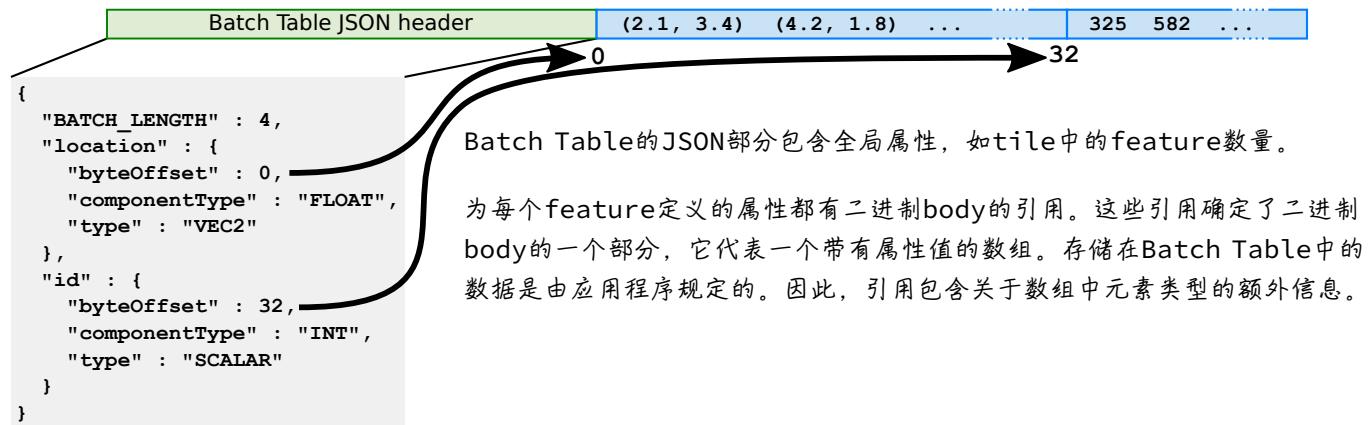
Feature Table和Batch Table被存储在tile格式body中。这两种表都有相同的结构：它们包括一个header部分，被解释为JSON字符串，和一个二进制Body：



JSON部分可能包含引用整个tile的全局属性。此外，JSON部分可能包含对二进制body的引用。支持的属性的确切集合取决于tile格式，但所有属性都定义了二进制body的片段，其中包含tile中出现的每个feature的数值数组。



一个属性所指向的二进制body的部分由body内的byteOffset给出。数据的类型取决于属性的语义。例如，POSITION属性指的是body的一个部分，它代表一个32位浮点值的数组，表示X、Y和Z坐标的位置。



数组数据在二进制体中的位置由byteOffset给出。type表示数组的元素是标量还是矢量。componentType规定了标量或向量组件的类型。

下表包含了每种组件类型所包含的字节数，以及每种类型所包含的元素数。这可以用来计算二进制body中属性数据的字节大小。

componentType	Size in bytes
"BYTE"	1
"UNSIGNED BYTE"	1
"SHORT"	2
"UNSIGNED SHORT"	2
"INT"	4
"UNSIGNED INT"	4
"FLOAT"	4
"DOUBLE"	8

type	Number of components
"SCALAR"	1
"VEC2"	2
"VEC3"	3
"VEC4"	4

## 10.2. Tile格式: Batched 3D Models (b3dm)

一个Batched 3D Models的tile包含异构模型的渲染数据。这些模型可能是地形或三维建筑，例如，用于建筑信息管理 (BIM) 或工程应用。

在一个Batched 3D Models中的实际渲染数据被存储为二进制glTF--GL传输格式的二进制形式。这种格式允许单个模型或甚至带有纹理和动画的完整3D场景以一种紧凑的形式存储，可以方便地通过网络传输并由运行时引擎直接渲染。glTF中的几何数据被存储在一个buffer中，该buffer的结构是将buffer分为几个部分，代表不同的属性，如顶点位置或法线。第13节“常见定义”中给出了关于glTF的总结和更多资源的链接。术语“batched”是指多个模型的几何数据，每个模型由顶点位置和可选的法线和纹理坐标组成，可以合并到一个单一的buffer，以提高渲染性能：单个buffer的数据可以直接复制到GPU内存中，从而减少了复制操作，而且它的结构可以最大限度地减少绘图调用的数量。

当多个模型被合并到一个单一的buffer中时，它必须仍然能够支持各个模型的样式和交互。这可以指通过渲染不同的颜色来突出显示一个模型，或者通过鼠标点击来确定哪个模型已经被选中。在3D Tiles中，这是通过用一个额外的顶点属性扩展buffer来实现的。

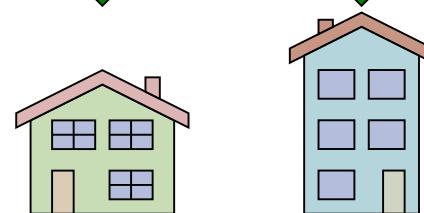
几何数据用batchId属性进行了扩展。

它将每个顶点的批次ID存储为一个整数。具有相同ID的顶点是同一模型的一部分。

position	x	y	z	x	y	z	...	x	y	z	x	y	z	...	x	y	z	...
normal	x	y	z	x	y	z	...	x	y	z	x	y	z	...	x	y	z	...
batchId	0	0	...	0	1	1	...	1	...	...	1	...	...	...	1	...	...	

批处理ID可以用来识别模型，以进行互动或设置样式：

当用户点击一个Batched 3D Model时，可以确定被选中的的batch ID。batch ID作为索引，用于查询Batch Table中的styling信息或元数据。



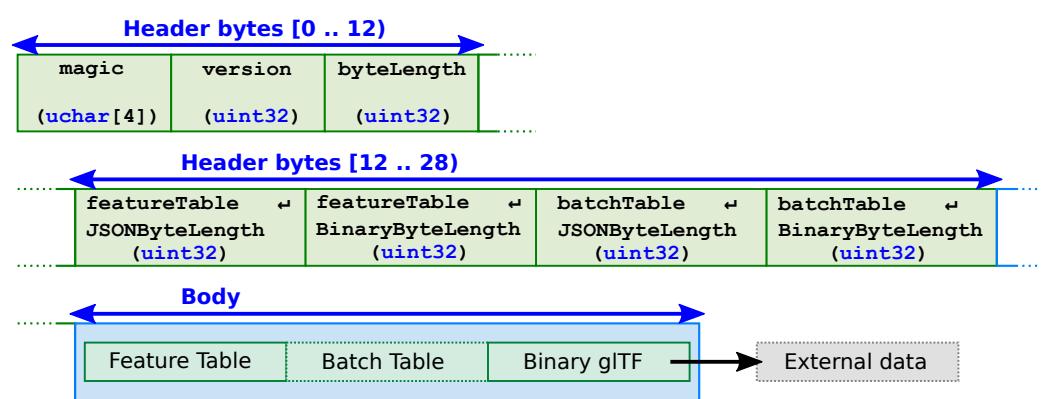
## Batched 3D Models: 属性

批量三维模型的特征表 (Feature Table) 只包含全局属性。批量表的大小以BATCH\_LENGTH属性给出，RTC\_CENTER可以在相对于中心给出位置的情况下存储中心点。

Property	Type	Description
BATCH_LENGTH	uint32	批中可区分的模型 (feature) 的数量。如果二进制 glTF 没有 batchId 属性，这个字段必须是 0。
RTC_CENTER	float32[3]	当位置被定义为相对于中心时的中心位置

## Batched 3D Models: 数据布局 (Data Layout)

下图显示了包含在Batched 3D Models的header和body中的信息的布局和数据类型。：

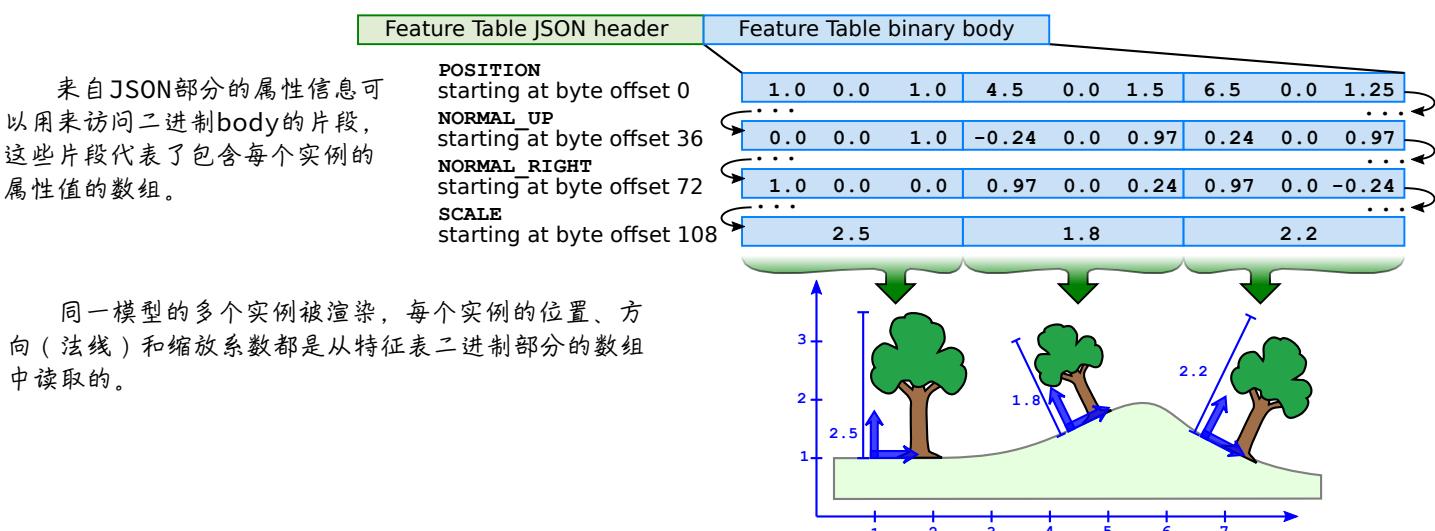
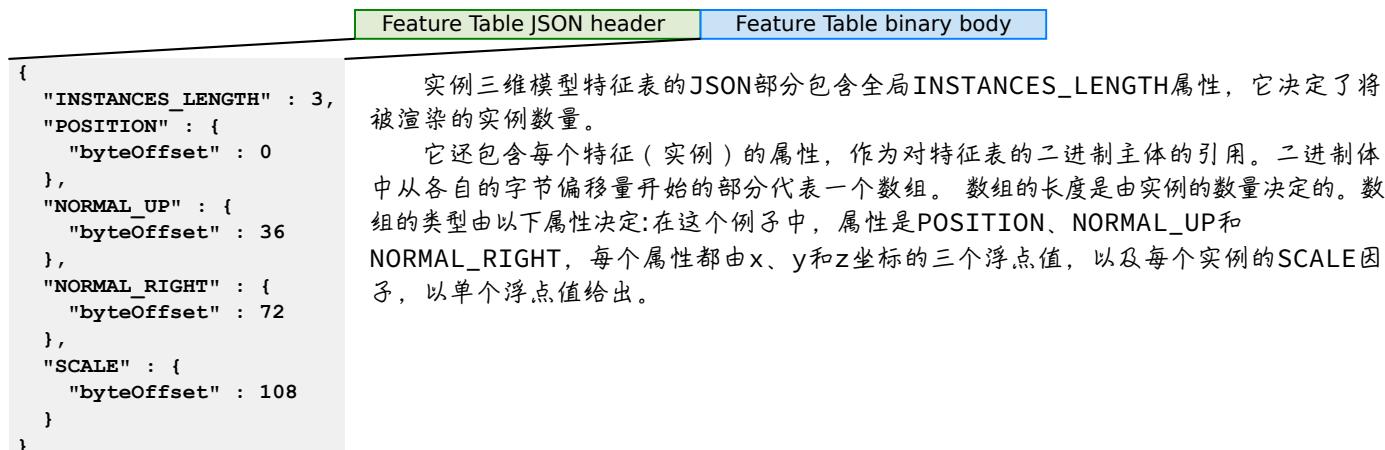


## 10.3 Tile 格式: Instanced 3D Models (i3dm)

在许多应用场景中，复杂的场景多次包含相同的模型，但有微小的变化。这类模型的例子可能是树木、CAD feature (如螺栓) 或室内设计的元素 (如家具)。

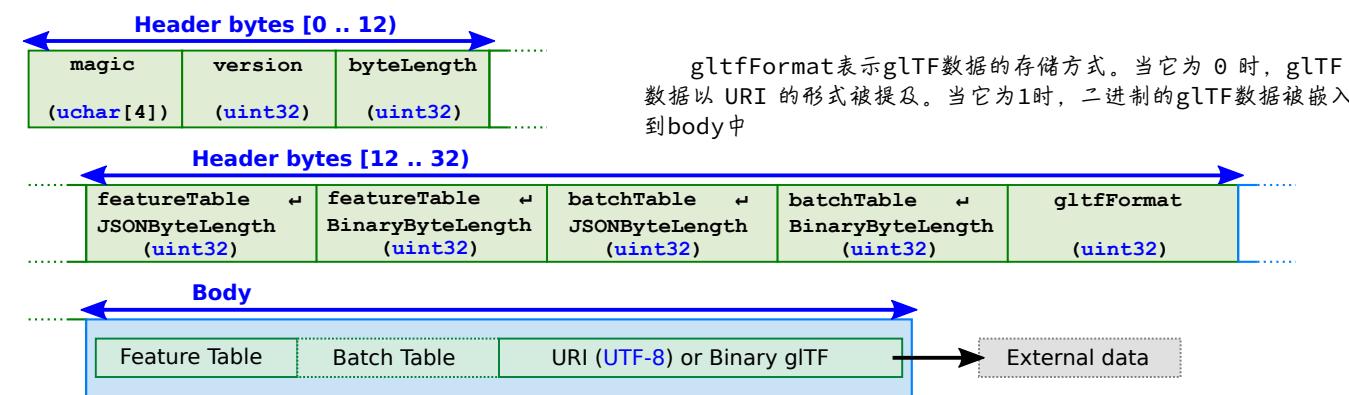
因此，3D tile格式支持Instanced 3D Models (实例三维模型)，即一个模型被多次渲染。这个模型的每一个外观都是一个实例instance (或特征feature)，而且这些实例可以通过不同的变换来渲染——例如，在不同的位置。

实际的模型是以二进制glTF的形式存储的 (见第13节，“通用定义”)。它可以直接存储在二进制体中。或者，主体可以包含一个二进制glTF文件的URI。关于有多少个实例将被渲染的信息，以及这些实例的位置和方向，都存储在特征表feature table中。特征表的第一部分包含JSON数据，第二部分包含二进制数据。



### (实例三维模型) Instanced 3D Models: 数据布局 ( Data Layout )

下图显示了包含在实例三维模型的header和body中的信息的布局和数据类型。



## Instanced 3D Models: 属性 ( Properties )

实例三维模型的特征表可能包含以下属性:

- Number of instances:** 将被渲染的实例的数量，存储在INSTANCES\_LENGTH属性中。这决定了存储每个实例属性的数组的长度。在Batch Table为每个实例存储元数据的情况下，它也决定了Batch Table的大小。
- Batch ID:** 每个实例的一个标识符，存储在BATCH\_ID属性中。这个batch ID作为一个索引，可以用来查询Batch Table中每个实例的属性。
- Position:** 实例的位置，以笛卡尔的x、y和z坐标给出。位置可以直接存储在POSITION属性中，也可以使用POSITIONS\_QUANTIZED属性中的压缩(compressed)、量化(quantized)表示。
- Relative-to-center point:** 相对于中心位置的中心点，可以存储在RTC\_CENTER属性中。当它被定义后，位置就会相对于这个点给出。
- Quantized volume:** 当POSITIONS\_QUANTIZED属性中的位置以其压缩形式给出时，用于量化的体积。体积由QUANTIZED\_VOLUME\_OFFSET和QUANTIZED\_VOLUME\_SCALE决定，前者决定了体积的offset，后者决定了量化后体积的比例。
- Orientation:** 每个实例的方向可以用两个向量来定义：一个向量定义向上的方向，另一个向量定义向右的方向。这些法线可以存储在NORMAL\_UP和NORMAL\_RIGHT属性中，或者使用压缩的、八进制编码的法线表示，其中法线被存储在NORMAL\_UP\_OCT16P 和NORMAL\_RIGHT\_16P 属性中。
- Default orientation:** 当各个实例的方向没有给出时，那么EAST\_NORTH\_UP可以用来表示每个实例将默认为WGS84椭球体上的东/北/上参考框架的方向。
- Scale:** 实例的缩放系数。每个实例的缩放可以是一个统一的缩放，这是一个存储在SCALE属性中的单一值，或者是沿x、y和z轴的缩放系数，通过SCALE\_NON\_UNIFORM属性给出。

关于量化位置(quantized position)和八进制编码的法线表示，关于相对中心位置的概念，以及WGS84椭圆体的进一步信息，可以在第13节 "常用定义" 中找到。

## Instanced 3D Models: 属性摘要 ( Properties Summary )

下表总结了实例三维模型的特征表 ( Feature tables ) 中可能包含的属性。

### 全局属性 ( Global Properties )

Property	Type	Description
INSTANCES_LENGTH	uint32	实例的数量
QUANTIZED_VOLUME_OFFSET	float32[3]	量化后的体积在x、y和z方向的偏移量
QUANTIZED_VOLUME_SCALE	float32[3]	量化体积在X、Y和Z方向的比例
EAST_NORTH_UP	boolean	实例方向是否应默认为WGS84椭球体上的东/北/上参考框架
RTC_CENTER	float32[3]	当位置被定义为相对于中心时的中心位置

### 每个实例的属性 ( Per-instance Properties )

这些属性中的每一个都是对特征表的二进制主体 ( body ) 的一个部分的引用。这一部分包含有实际属性值的数据。数组元素的数据类型是由属性的类型决定的。数组的长度由实例的数量决定。

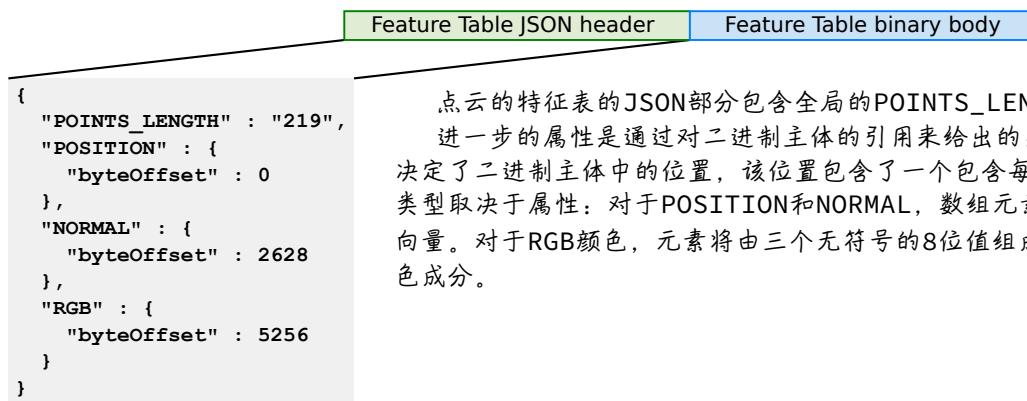
Property	Type	Description
POSITION POSITION_QUANTIZED	float32[3] uint16[3]	实例位置的x、y、z坐标
NORMAL_UP NORMAL_UP_OCT32P	float32[3] uint16[2]	一个定义实例向上方向的单位向量
NORMAL_RIGHT NORMAL_RIGHT_OCT32P	float32[3] uint16[2]	一个定义实例右方向的单位向量
SCALE	float32	实例中所有轴的缩放系数
SCALE_NON_UNIFORM	float32[3]	实例的x、y和z轴分别的缩放系数
BATCH_ID	uint8/16/32	batch ID，用于在batch table中查询实例的元数据 ( metadata )。

## 10.4. Tile Formats: 点云 ( Point Clouds ) (pnts)

从建筑物或环境等现有结构中获取三维数据的常见方法是通过摄影测量或激光雷达扫描。这种获取过程的结果是一个点云，其中每个点都由其位置和确定其外观的附加属性来界定。

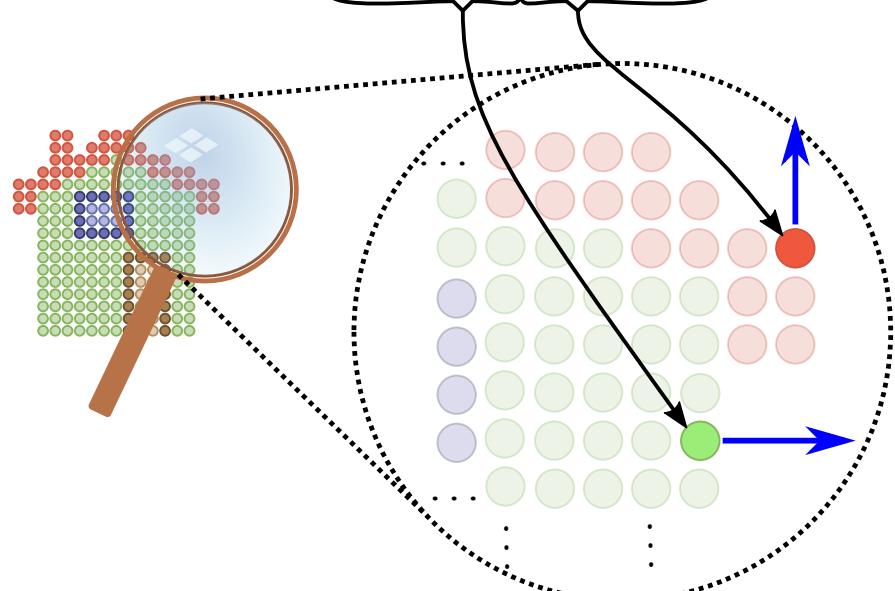
点云格式是3D tile中的一种格式，它可以为3D可视化提供大量的点云流。

关于点的位置和其他视觉属性的信息被存储在特征表 ( Feature Table ) 中，该表由一个JSON header和一个二进制主体 ( body ) 组成。



特征表的二进制部分的每一节都代表一个属性值的数组。数组元素的类型取决于该属性。数组的长度是由点的数量决定的。

Feature Table JSON header	Feature Table binary body
POSITION	starting at byte offset 0     7.3  1.0  6.7   7.1  1.0  6.9   ...
NORMAL	starting at byte offset 2628   1.0  0.0  0.0   0.0  0.0  1.0   ...
RGB colors	starting at byte offset 5256   0  255  0   255  0  0   ...

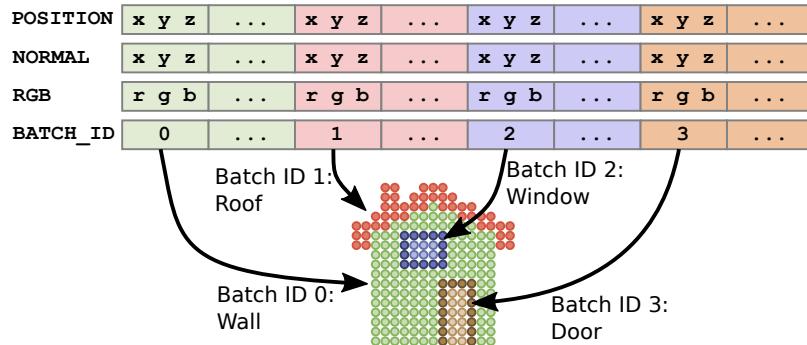


## 批点云 (Batched Point Clouds)

在点云中，有可能定义代表不同特征的点群。例如，可能有一组点代表一扇门、一扇窗或一所房子的屋顶。  
在3D Tiles中，这些组是通过给点分配一个batch ID来确定的，类似于用于Batched 3D Models的概念。

对于batched Point Clouds，特征表的JSON部分包含一个BATCH\_LENGTH属性，用于确定点组的数量，以及一个BATCH\_ID属性，指的是二进制主体的一个部分，其中包含点的batch ID阵列，以8、16或32位整数值存储。然后，batch ID可以作为索引，在Batch Table中查找该组点的元数据(metaData)。

```
{  
    "POINTS_LENGTH" : "219",  
    "BATCH_LENGTH" : 4,  
    "POSITION" : {  
        "byteOffset" : 0  
    },  
    ...  
    "BATCH_ID" : {  
        "byteOffset": 5913,  
        "componentType" :  
            "UNSIGNED_BYTE"  
    }  
}
```

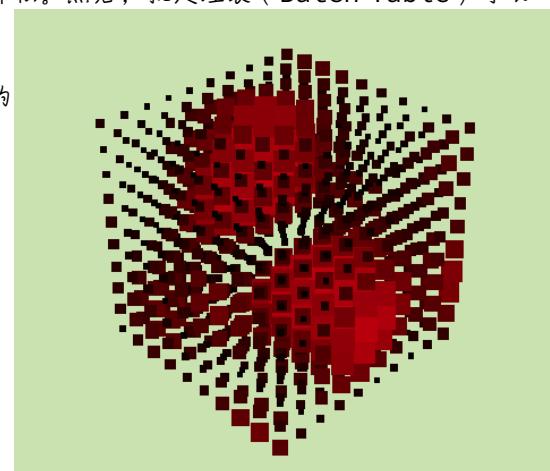


## Outlook: 声明式样式点云 (Declarative Styling Point Cloud)

第11节展示的声明式样式 (Declarative Styling) 可以应用于点云，以促进不同类型的信息可视化。  
例如，一个点云可能由一个有规律的网格点组成，并填满了一定的体积。然后，批处理表 (Batch Table) 可以包含在各自位置进行的测量。

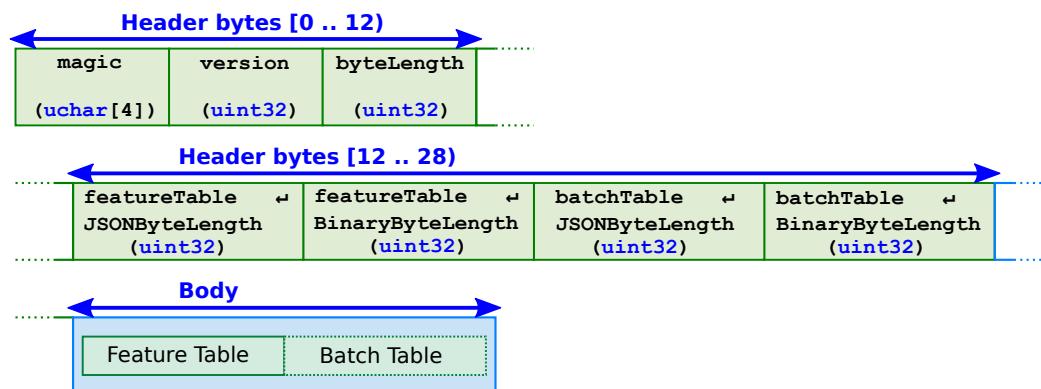
在这个例子中，每个点都有一个相关的 温度值存储在批处理表中。  
这些来自批处理表的领域的特定信息可以通过以下方式被映射到点的视觉属性上，使用3D Tiles styling语言。绘制成为点的视觉属性，这里，温度属性影响了渲染点的颜色和大小。

```
{  
    "color" : "color('red') * ${temperature}",  
    "pointSize" : "5 + ${temperature} * 30"  
}
```



## 点云：布局

下图显示了点云的header和body中所包含的信息的布局和数据类型。



## 点云：属性

点云的Feature Table可能包含以下属性。

<b>Number of points:</b>	点云中的点的数量是由POINTS_LENGTH属性给出的。这决定了存储每个点的属性的数组的长度。
<b>Number of batches:</b>	点云中的批次(batches)(点组)的数量，存储在BATCH_LENGTH属性中。因此，它是BATCH_ID属性中的唯一值的数量，以及Batch Table中的条目数量。
<b>Batch ID:</b>	该点所属batch(group)的标识符被存储在BATCH_ID属性中。所有具有相同batch ID的点都属于一个batch。batch ID作为一个索引，用于在Batch Table中查找这组点的属性。
<b>Position:</b>	点的位置，以笛卡尔的x、y、z坐标给出。这些位置可以直接存储在POSITION属性中，也可以存储在POSITIONS_QUANTIZED属性中的压缩(compressed)、量化(quantized)表示。
<b>Relative-to-center point:</b>	相对于中心的位置的中心点可以存储在RTC_CENTER属性中。当它被定义后，位置就会相对于这个点给出。
<b>Quantized volume:</b>	当POSITIONS_QUANTIZED属性中的位置以其压缩形式给出时，用于量化的体积。体积由QUANTIZED_VOLUME_OFFSET和QUANTIZED_VOLUME_SCALE决定，前者决定了体积的offset，后者是量化体积的比例。
<b>Normal:</b>	描述点的法线的单位向量。这些可以存储在NORMAL属性中，或者使用NORMALS_OCT16P属性，使用法线的压缩、八进制编码表示。
<b>Color:</b>	每个点的颜色可以以RGB或RGBA颜色的形式给出，存储在RGB或RGBA属性中。或者，也可以在RGB565属性中存储颜色的压缩表示。
<b>Constant color:</b>	可以在CONSTANT_RGBA属性中为tile中的所有点提供一种颜色。当没有为单个点定义颜色时，就会使用这种颜色。

关于量化位置和八进制编码的正常表示，以及关于相对于中心的位置概念的进一步信息，可以在第13节“常用定义”中找到。

## 点云：属性摘要 (Properties Summary)

下表总结了可能包含在点云特征表(Feature Table)中的属性。

### 全局属性 (Global Properties)

Property	Type	Description
POINTS_LENGTH	uint32	点的数量
QUANTIZED_VOLUME_OFFSET	float32[3]	量化后的体积在x、y和z方向的偏移量
QUANTIZED_VOLUME_SCALE	float32[3]	量化体积在X、Y和Z方向的比例
CONSTANT_RGBA	uint8[4]	瓦片中所有点的RGBA颜色分量
BATCH_LENGTH	uint32	点的batch的数量
RTC_CENTER	float32[3]	当位置被定义为相对于中心时的中心位置

### 每个点的属性 (Pre-Point Properties)

这些属性中的每一个都是对特征表(Feature Table)的二进制主体的一个部分的引用。这一部分包含有实际属性值的数据。数组元素的数据类型是由属性的类型决定的。数组的长度由点的数量决定。

Property	Type	Description
POSITION	float32[3]	点的位置的x、y、z坐标
POSITION_QUANTIZED	uint16[3]	点的位置的x、y、z坐标
RGBA	uint8[4]	该点的RGBA颜色成分
RGB	uint8[3]	该点的RGB颜色分量
RGB565	uint16	一个压缩的、16位的RGB颜色表示，其中5位为红色，6位为绿色，5位为蓝色。
NORMAL	float32[3]	一个定义点的法线的单位向量
NORMAL_OCT16P	uint8[2]	一个定义点的法线的单位向量
BATCH_ID	uint8/16/32	batch ID，用于查询batch table中各点的元数据。

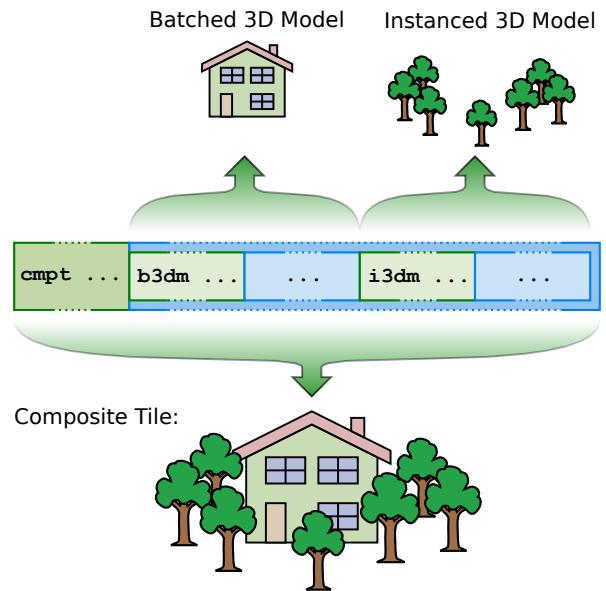
## 10.5. Tile Formats: 复合瓦片 (Composite Tiles) (cmpt)

3D Tiles支持流式异构数据集 (streaming heterogenous datasets)。多种瓦片格式可以结合在一个瓦片组中。也可以在一个复合瓦片中结合多个不同格式的瓦片，以获得额外的灵活性。

使用复合瓦片 (Composite Tiles) (cmpt) 的一个例子可以在地理空间应用中找到。一组建筑物可以存储在一个Batched 3D Model中，而一组树木可以存储为一个Instanced 3D Model (实例3d模型)。当这些元素出现在同一地理位置时，将这些模型组合在一个复合瓦片中是很有用的：然后可以通过一个请求，将特定地理位置的可渲染内容作为一个瓦片获得。

被组合在一个复合瓦片中的瓦片被称为内瓦片 (*inner tiles*)。复合瓦片也可以嵌套，这意味着每个内部瓦片可以再次成为一个复合瓦片。

与其他瓦片格式一样，复合瓦片的header以表示瓦片格式 (cmpt) 的magic bytes开始，然后是一个版本号，以及瓦片数据的总长度，包括header，以字节为单位。

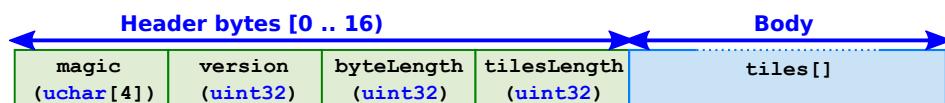


header的公共部分后面是一个整数，表示复合瓦片中被组合的瓦片数量。

复合瓦片的body只是由一个内部瓦片 (*inner tiles*) 的阵列组成。每个内瓦片是一个代表瓦片的二进制blob：它以通用header开始，表明内瓦片数据的格式、版本和长度，然后是各自瓦片格式的body。

### 复合瓦片 (Composite Tiles)：数据布局

下图显示了包含在复合瓦片的header和body中的信息的布局和数据类型。



## 11. 扩展

3D Tiles提供了一个用新功能扩展基本规格的机制：3D Tiles中的每个JSON对象都可以包含一个扩展字典。这个字典的键是扩展的名称，而值可以是扩展的具体对象。供应商 (vendors) 可以提出扩展，并为字典中包含的扩展对象的结构和语义提供一个规范。

```
{  
  ...  
  "extensions": {  
    "VENDOR_collision_volume": {  
      "sphere": [ 5.0, 3.0, 7.0, 10.0 ]  
    }  
  }  
}
```

这个例子显示了在一个假设的供应商扩展中可能被添加到瓦片JSON中的JSON，该扩展为瓦片定义了一个碰撞体积 (collision volume)。扩展的名称是VENDOR\_collision\_volume，它通过中心和半径定义一个碰撞球 (collision sphere)，类似于3D Tiles中已经支持的边界体积 (bounding volumes)。

在一个tileset或其一个子代中使用的扩展名称必须列在瓷砖组的顶层extensionsUsed字典中。当为了正确加载和渲染tileset而需要某个扩展时，它也必须被列在extensionsRequired字典中。实现可以在加载tileset时检查这些字典，并检查它们是否支持tileset使用或需要的扩展。

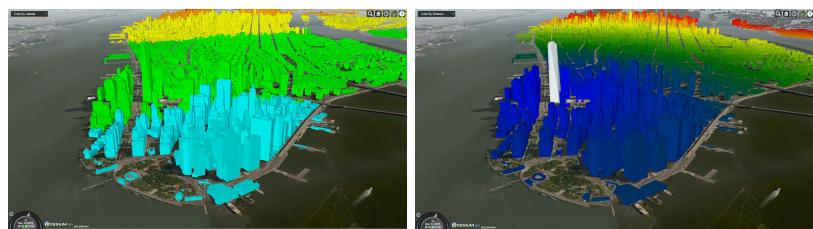
## 12. 声明式样式(Declarative Styling)

根据瓦片格式的不同，瓦片的可渲染内容可能包含不同的特征 (feature)。对于Batched 3D Model，每个用batchId识别的模型都是一个feature。对于实例三维模型 (Instanced 3D Models)，每个实例都是一个feature。对于点云，有两种选择：当点被分批排列时，每一组具有相同batchId的点都是一个feature。否则，每个点都是一个feature。

3D Tiles允许在运行时修改feature的外观，使用声明式样式：用包含一组表达式的JSON来定义一个样式。这些表达式的值决定了feature的可见性或颜色。运行时引擎可以评估这些表达式，并根据用户互动和存储在Batch Table中的feature属性对feature应用样式。

例如，在一个每个建筑物都是一个feature的Batched 3D Model中，建筑物的颜色可以在运行时根据不同的标准进行修改。下面是一个样式JSON的例子，它使建筑物根据其高度被渲染成不同的颜色。

```
{  
  "color": {  
    "conditions": [  
      ["${height} >= 300", "rgba(45, 0, 75, 0.5)"],  
      ["${height} >= 200", "rgb(102, 71, 151)"],  
      ["${height} >= 100", "rgb(170, 162, 204)"],  
      ["${height} >= 50", "rgb(224, 226, 238)"],  
      ["${height} >= 25", "rgb(252, 230, 200)"],  
      ["${height} >= 10", "rgb(248, 176, 87)"],  
      ["${height} >= 5", "rgb(198, 106, 11)"],  
      ["true", "rgb(127, 59, 8)"]  
    ]  
  }  
}
```

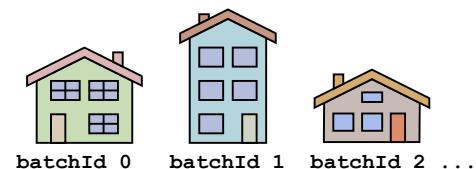


其他声明式样式的例子可以根据其他属性，如它们的纬度（左）或他们与某一地标的距离（右），对建筑物应用一种颜色。

### 给Batched 3D Model设置样式

在这个例子中，一个Batched 3D Model包含多个建筑。每个建筑都有自己的batchId。

该模型也有一个Batch Table。Batch Table的JSON的header包含一个属性 "height"。它指向Batch Table的二进制body的一个段。该段代表一个包含建筑物高度的数组。batchId用来在这个数组中查找建筑物的高度。

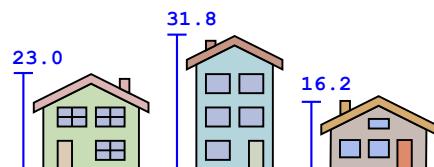


#### Batch table JSON:

```
"height" : {  
  "byteOffset" : 0,  
  "componentType" : "FLOAT",  
  "type" : "SCALAR"  
}
```

#### Batch table binary:

```
23.0 31.8 16.2 ...
```

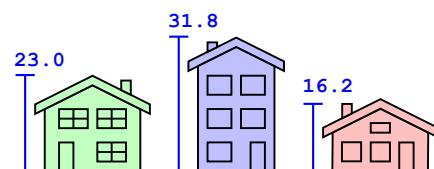


当模型被渲染时，一个样式JSON被应用到模型上。样式决定了每个特征（建筑）的颜色。在这个例子中，颜色取决于 "height" 属性的值。

#### Style JSON:

```
{  
  "color" : {  
    "conditions" : [  
      ["${height} < 20", "color('#FFC0C0')"],  
      ["${height} < 30", "color('#COFFC0')"],  
      ["${height} < 40", "color('#C0C0FF')"]  
    ]  
  }  
}
```

当高度低于20、30或40时，模型将分别以红色、绿色或蓝色呈现：



## Style JSON

样式描述是以JSON格式传递给运行时引擎的。它包含决定瓦片中feature外观的属性。

```
{
  "show" : "true",
  "color" : "color('red')",
  "pointSize" : "3.0"
}
```

对于Batched 3D Model和Instanced 3D Model，样式由一个决定feature的可见性的显示属性和一个决定颜色的颜色属性组成。

对于点云，样式还可能包含一个pointSize属性，它决定了点在被渲染时的大小。

样式的属性值是用一种表达式语言写的，它是JavaScript的一个子集。这种语言的类型还包括矢量类型，对几何计算和表示颜色很有用。这些矢量类型是从GLSL派生出来的，因此可以用这些类型表示的表达式可以很容易地在着色器中实现，而且很有意义。

## 扩展

样式语言(styling language)支持JavaScript所支持的大部分一元和二元运算符，以及三元条件运算符和常见的内置数学函数，如abs、max或cos。

该语言的类型包括向量类型vec2、vec3和vec4。vec4类型用于表示RGBA颜色。

vec2、vec3和vec4类型的矢量成员可以使用点符号来访问。例如，4D矢量v的元素可以被解释为x、y、z和w坐标，并被访问为v.x、v.y、v.z和v.w。

另外，可以使用数组符号来访问组件，4D矢量v的元素可以被访问为v[0]、v[1]、v[2]和v[3]。

颜色用vec4类型表示，包含RGBA颜色成分。有不同的便利函数用于从不同的参数创建颜色。这些函数类似于CSS中定义颜色的方式。比如说。

- `color('red')` 从一个关键词创建
- `color('blue', 0.5)` 从关键词跟透明度创建
- `color('#00FFFF')` 从一个十六进制的RGB字符串
- `rgb(100, 255, 90)` 从[0, 255]中的红色、绿色和蓝色组分创建
- `hsl(1.0, 0.6, 0.7)` 从色调、饱和度和亮度创建，每个值都在[0, 1]范围内

我们定义了几个内置函数，可以应用于数字和某些矢量类型。clamp函数可用于将数字或向量成分限制在某个范围内。混合函数允许在数字或向量之间进行线性插值。长度、距离、归一化和点函数可用于向量的几何计算。十字函数可以计算三维向量的交叉积。

表达式可以被解析成抽象语法树(abstract syntax tree)(AST)并被评估。被评估的表达式的类型必须与样式属性的类型相匹配。

## 变量 (Variables)

在样式中使用的表达式可能包含变量。这些变量指的是tile中feature的属性。这些属性的实际值被存储在Batch Table中。

变量使用ECMAScript 2015模板字面语法编写：字面意义上的\${property}是指feature的一个属性，其中property是区分大小写的属性名称。

下面是一个样式的例子，它根据存储在Batch Table中的features的高度，为feature分配颜色：高度大于50的feature将被染成红色。而所有其他的将是白色。

```
{
  "color" : "(${height} > 50) ? color('red') : color('white')"
}
```

## 条件 (Conditions)

样式属性也可以用条件来写。一个条件由两个表达式组成：一个求值为布尔值的表达式，以及一个当布尔值为真时被求值的表达式。多个条件可以被组合成一个数组。

```
{
  "color" : {
    "conditions" : [
      ["${temperature} > 100", "color('red')"],
      ["${temperature} > 50", "color('yellow')"],
      ["true", "color('green')"]
    ]
  }
}
```

一个使用条件的例子：如果与一个feature相关的温度大于100，那么将返回红色。如果它大于50，那么将返回黄色。否则，将返回绿色。这些条件是按顺序求值的。如果没有满足条件，那么样式属性将是未定义的。

## 定义变量 (Defining Variables)

除了引用Batch Table中的属性的变量外，一个样式也可以定义它自己的变量。这些定义在样式的defines属性中给出。每个定义由新变量的名称组成，它被映射到一个表达式。该表达式不得引用其他定义，但可以引用Batch Table中的现有变量。

```
{
  "defines" : {
    "distanceToPoint" :
      "distance(vec2(${x}, ${y}), vec2(1, 2))"
  },
  "color" : {
    "conditions" : [
      ["${distanceToPoint} > 1.0", "color('red')"],
      ["true", "color('0x0000FF', ${distanceToPoint})"]
    ]
  }
}
```

新定义的distanceToPoint变量表示一个feature到点(1,2)的距离，基于存储在Batch Table中的变量\${x}和\${y}。

新的变量被用来决定样式的颜色。当距离超过一个阈值时，该feature会被渲染成红色。否则，距离被用来控制颜色的不透明度。

## 正则表达式 (Regular Expressions)

样式支持正则表达式，因此可以根据Feature Table中以字符串形式存储的属性值来制定渲染条件。

```
{
  "show" : "regExp('Building\\s\\d').test(${name})"
}
```

一种只显示具有\${name}属性的feature的样式，该属性符合给定的模式：字符串"Building"，后面是空格和一个整数。

## 点云样式 (Styling Point Clouds)

当声明性样式被应用于点云时，该样式也可以参考存储在Feature Table中的语义，如位置、颜色和法线。点的位置可以被称为\${POSITION}。当位置以量化形式存储时，这指的是应用量化比例后，但在添加量化集之前的位置。与此相反，\${POSITION\_ABSOLUTE}指的是应用量化缩放和offset之后的位置。颜色可以用\${COLOR}关键字访问。点的法线可以用\${NORMAL}来访问。如果法线是以八进制编码的方式存储的，那么这指的是解码后的法线。

关于量化位置和八进制编码的正常表示的进一步信息，可以在第13节“常用定义”中找到。

## 13. 常见定义 (Common Definitions)

### 13.1. glTF - GL传输格式

3D Tiles中的Batched 3D Models和Instanced 3D Models可以嵌入以二进制glTF形式存储的模型。这是GL传输格式的二进制形式，它是由Khronos组织维护的关于3D内容精确传输的一个开放规范。

<https://github.com/KhronosGroup/glTF>

### 13.2. 世界大地测量系统椭圆体World Geodetic System Ellipsoid (WGS84)

"区域 (region)"边界体积类型是为地理空间应用准备的。这些界线体的范围以纬度和经度的形式给出，以弧度为单位，以EPSG 4979中规定的WGS 84为基准。

<https://spatialreference.org/ref/epsg/4979/>

区域边界体积内容的最小和最大高度以高于或低于WGS 84椭圆体，以米为单位给出。

<https://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf>

### 13.3. 相对于中心的位置Relative-To-Center Positions (RTC\_CENTER)

(The information presented here is based on <http://help.agi.com/AGIComponents/html/BlogPrecisionsPrecisions.htm>)

3D Tiles的创建是为了使大量的3D内容，特别是地理空间数据，能够以高质量、互动的方式可视化。这意味着必须能够高度准确地渲染三维物体，即使它们与虚拟世界的中心有很大的距离。

在常见的图形API（如OpenGL、Direct3D或Vulkan）中，3D对象的顶点位置通常被存储为单精度（32位）的浮点值。由于这种有限的精度，与渲染坐标系的原点有较大距离的物体不能被准确地表示出来：不同的顶点坐标将有相同的、内部表示的单精度值。

Theoretical value	Actual 32-bit single-precision value
131072.01	131072.0156250
131072.02	131072.0156250
131072.03	131072.0312500

尽管131072.01和131072.02的值不同，但这些值作为32位单精度值的表示方法是相同的。

这种缺乏精确性的情况可能会导致渲染伪影 (artifacts) --最明显的是，在紧贴着某个渲染对象放大的时候，会出现视觉抖动。为了缓解这个问题，3D Tiles支持一种补偿大坐标值所产生的误差的技术。这种技术被称为相对中心 (Relative-To-Center) (RTC) 渲染。

Batched 3D Model的顶点位置作为一个属性存储在模型的 glTF 表示中。对于Instanced 3D Models和点云，Feature Table包含一个实例和点的位置属性。这些位置的坐标值通常被存储为单精度的32位浮点值。为了支持RTC渲染，各瓦片格式允许在其Feature Table中指定RTC\_CENTER属性。这个属性定义了一个应用特定的坐标系的中心，如果它存在，所有的位置都被假定为相对于这个中心给出。

为了考虑顶点的相对位置，RTC\_CENTER被用来修改用于渲染的Model-View-Matrix。最初，这是一个MVGPU的矩阵，以双精度存储在CPU上。RTC\_CENTER通过原始的Model-View矩阵被转换为视点坐标 (eye coordinates)。

$$\text{RTC\_CENTER}_{\text{Eye}} = \text{MV}_{\text{GPU}} * \text{RTC\_CENTER}$$

用于渲染的Model-View-Matrix是一个以单精度存储的MVGPU矩阵，它是通过将原始Model-View矩阵的最后一列替换为  $\text{RTC\_CENTER}_{\text{Eye}}$  的结果。

有了这种技术，就可以避免在模型的位置上出现大的数值，这将会由于后续渲染管道的有限精度而造成渲染伪影：位置可以相对于RTC\_CENTER的小数值给出，修改后的Model-View矩阵将它们正确地转换为视点坐标系进行渲染。

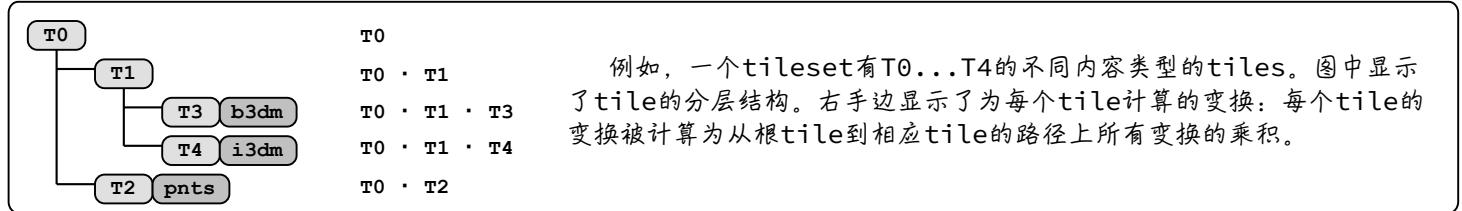
## 13.4. 3D Tiles中的变换 (Transforms in 3D Tiles)

(13. Common Definitions)

3D Tiles中的可渲染内容可以用不同的坐标系给出。例如，一个城市的tileset可以包含一个单一建筑的嵌套tileset，后者可以用它自己的坐标系给出。

为了在本地坐标系之间进行转换，每个tile都有一个可选的转换属性。这个属性是一个列主序的 $4 \times 4$ 仿射矩阵(affine transformation)，将tile的坐标系转换为父坐标系，如果没有定义，则默认为单位矩阵(identity matrix)。

tile的变换矩阵影响tile及其内容的位置、法线和边界体积。feature的位置(如Instanced 3D Model中的实例，或点云中的点)与转换矩阵相乘，使它们从本地坐标系进入父tile的坐标系。法线与矩阵的左上角 $3 \times 3$ 矩阵的转置相乘，以适当考虑到非均匀缩放。除了“区域”边界体积被明确定义为EPSG:4979之外，边界体积也被转换为矩阵。



如果tile的内容使用相对于中心的位置(relative-to-center positions)，那么RTC\_CENTER必须被视为顶点的额外平移。

### 13.4.1 3D Tiles和glTF中的坐标系统

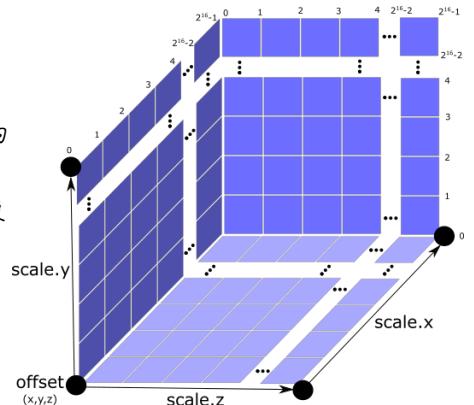
在本地直角坐标系中，3D Tiles将Z轴定义为向上。与此相反，glTF认为y轴是向上的。当glTF资产被嵌入到Batched 3D Models和Instanced 3D Models中时，必须考虑这些不同的惯例，在运行时对glTF资产进行转换，使Z轴指向上方。此外，每个glTF资产都有自己的节点层次结构和变换。关于这些不同的变换如何共同发挥作用的细节，以及它们应用于可渲染内容的顺序，可以在3D Tiles规范中找到。

<https://github.com/CesiumGS/3d-tiles/tree/master/specification#transforms>

## 13.5. 量化位置和Quantized Positions and Oct-encoded Normals

Instanced 3D Model tile瓦片格式中的实例位置以及点云瓦片格式中的点的位置可以以不同的形式给出：当它们以POSITION属性给出时，它们被存储为三个32位的浮点值，包含实例或点的x、y和z坐标。

对于大量的实例或点，3D Tiles提供了一种更紧凑地存储位置的方法：当使用POSITION\_QUANTIZED属性定义位置时，它们用三个16位无符号整数值表示。这是通过存储相对于量化体积的位置来实现的，量化体积由QUANTIZED\_VOLUME\_OFFSET和QUANTIZED\_VOLUME\_SCALE属性决定，这两个属性存储在各自的Feature Table中。



<https://github.com/CesiumGS/3d-tiles/blob/master/specification/TileFormats/Instanced3DModel/README.md#quantized-positions>

同样，点和实例的法线也可以在NORMAL、NORMAL\_UP和NORMAL\_RIGHT属性中以三个浮点值组成的向量形式给出。对于数量较多的点或实例，这些法线可以以压缩形式存储，用这些后缀为\_OCT16P的属性名表示。这种压缩形式由双向映射组成：法向量从单位球体的八面体映射到八面体的面，然后投射到平面上并展开成单位正方形。使用这种压缩，一个三维法向量可以用一个16位的值来表示。

<https://github.com/CesiumGS/3d-tiles/blob/master/specification/TileFormats/PointCloud/README.md#oct-encoded-normal-vectors>