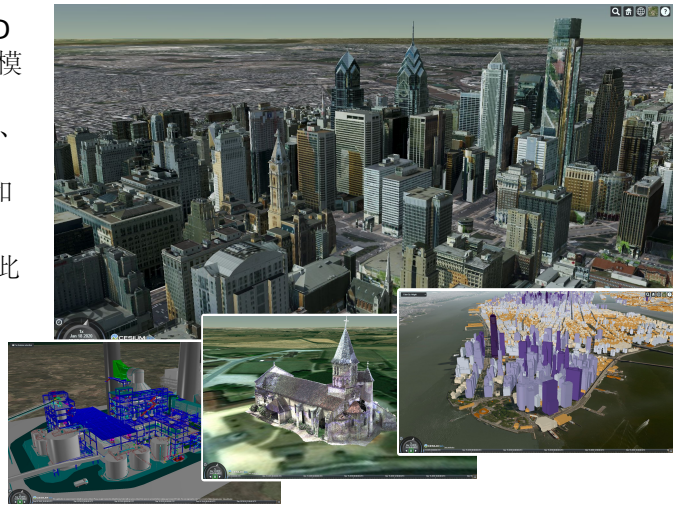
3D Tiles

3D Tiles 是一个用于共享、可视化、融合、交互和分析大量异构 3D 地理空间内容的跨桌面、Web 和移动应用程序的开放规范。3D Tiles 建立在 glTF之上，glTF 是一种用于高效流式传输和渲染 3D 模型和场景的开放标准。

3D 地理空间内容包括 摄影测量/大型模型、BIM/CAD、3D 建筑、实例化特征和点云，可以转换为 3D Tiles 并组合成单个数据集，以无缝衔接的形式实现实时分析，包括测量、可见性分析、样式和过滤。

3D Tiles 是一种空间数据结构，它支持分层细节级别(HLOD)，因此只有可见的瓦片才会被流式传输和渲染，从而提高整体性能。



本文概括总结了 3D Tiles 规范支持的主要概念：

• 瓦片集与瓦片的一般概念，以及它们是如何将大量数据组成可以有效流式传输的元素。	1. 概览: 瓦片集示例 2. 瓦片集与瓦片
• 3D Tiles 如何实现分层空间数据结构，用于高效渲染和交互。	3. 边界包围体 4. 空间数据结构
• 层次细节级别 (HLOD) 的概念,，可以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 Tiles 背后的概念，以实现高效渲染和交互。	7. 使用 3D Tiles 优化渲染 8. 3D Tiles 中的空间查询
• 3D Tiles 中不同瓦片格式的技术细节	9. 瓦片格式: 介绍 10. 瓦片格式
• 使用附加功能扩展基本规范的可能性	11. 扩展
• 通过基于元数据设置信息可视化的内容样式	12. 样式声明
• 地理空间坐标系和几何数据压缩等基本元素如何集成到 3D Tiles	13. 通用定义

如果你正在寻找 3D 瓦片集平台用于开始开发项目，推荐使用 Cesium ion (https://cesium.com/ion)，它可以托管 3D 瓦片集，并且允许用户上传自己的数据以创建、托管和流式传输 3D Tiles。

CESIUM®

1. 概览: 一个瓦片集示例

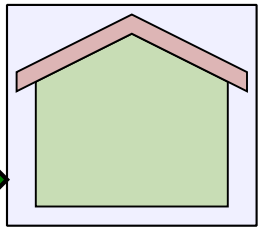
3D Tiles 的核心元素是 瓦片集(tileset)。 瓦片集是按层次结构组织的一组瓦片。瓦片集使用 JSON 的形式来描述内部信息。下面是一个瓦片集的简单例子，介绍了最重要的概念和要素。本文后续章节会更加详细地解释每一个概念。

```
{
  "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"
        },
      }
    ]
  }
}
```

每个瓦片可以引用一个外部瓦片集。这允许将多个较小的瓦片集组合成更大的瓦片集。

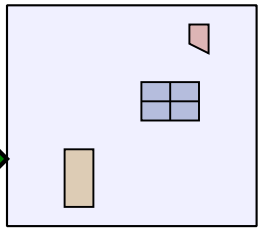
瓦片集的 JSON 内容包含资产和一般属性的基本描述。几何误差(详情参见第5章) 用于确定合适渲染根瓦片。

每个瓦片包含一个包围体(详情参见第3章)，它包含了瓦片的内容和所有子元素。它还包含一个几何误差，用于确定何时应呈现子项。

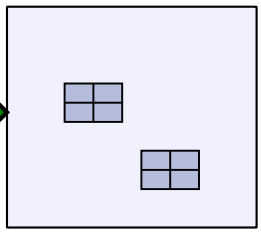


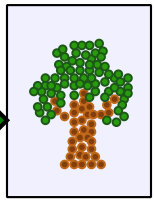
在这个示例中，第一个子项瓦片(房子)是指比较低层级的 3D 模型。此示例中的模型格式为 .b3dm，即 *Batched 3D Model* 缩写(详情参见第 10.2章)

这个子项瓦片(房子)包含其他模型(门窗)的详细信息。当更高层级被激活时它们会被渲染。

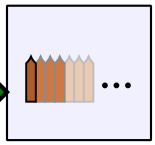


在这个示例中，这些被包含的子项瓦片(门窗)会被添加到父模型节点的低层级(low-level)中。或者这些子项瓦片可以包含更详细的模型以替换低层级模型。这是两种 3D Tiles 支持的细化策略(详情参见第 6 章)。



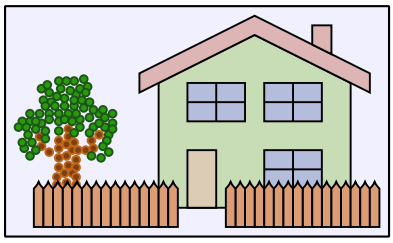


不同的瓦片格式可以组合在一个瓦片集中：在这个示例中，第一个子瓦片(房子)为 b3dm，即批量处理 3D模型。另外一个子项瓦片(树)为点云(详情参见第 10.4 章)。



此外还有瓦片(栅栏)是实例化的 3D模型(详情参见第 10.3 章)，它是一个简单的几何体，且在不同位置多次使用和被渲染。

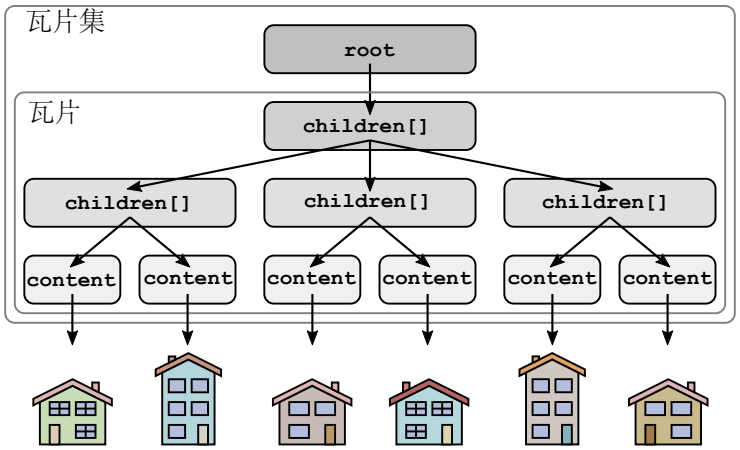
当渲染瓦片集时，瓦片以适当的细节级别组合，生成最终渲染结果：



2. 瓦片集和瓦片

一个瓦片集是由一组瓦片构成，是一个分层的数据结构，被称为 树(tree)。瓦片集顶部为一个根瓦片，每个瓦片又可能包含有子瓦片。
一个瓦片也可以引用另一个瓦片集。因此可以将多个瓦片集以灵活和分层的方式组合成一个更大的瓦片集。

每一个瓦片可以引用可渲染的内容。此内容可以具有不同的格式，例如可以表示纹理地形表面、3D模型或点云。在第 9 章 “瓦片格式：介绍”中会详细相接不同类型的可能性。



瓦片集和瓦片都是用 JSON 形式描述。瓦片集 JSON 文件包含有关瓦片集本身的基本信息以及瓦片的描述。

瓦片集属性：

根瓦片： 瓦片集的根瓦片(root)位于瓦片层次结构最顶部。

几何误差： 几何误差(geometricError) 用于量化如果地形设置不渲染时会出现的视觉误差阈值。当视觉误差超过某个阈值值，则考虑渲染瓦片集及其包含的瓦片。在第 5 章“几何误差”会有详细讲解。

属性摘要： 瓦片的可渲染内容可能具有相关联的属性。例如当瓦片包含建筑物时，每个建筑物的高度都可以存在瓦片中。瓦片集对象包含所有瓦片的可用属性(properties)值的最小值和最大值。

元数据： 有关 3D Tiles 版本的信息和特定应用程序的版本信息可以储存在 资产(asset) 属性中。

瓦片属性：

内容： 通过 URI 引用与图块关联的实际可渲染内容(content)。

子项： 瓦片的层次结构被建模为一棵树：每个瓦片可能有子项，这些子项被放置在 children 中。

包围体： 每一个瓦片都有一个关联的包围体，不同类型的包围体可能会存储在 boundingVolume 属性中。第 3 章“边界包围体”中介绍了可能的边界体类型。每个边界包围体都包含瓦片的内容和所有子项内容，从而产生空间连贯的边界体层次结构。

几何误差： 瓦片的可渲染内容(renderable)可能具有不同级别的细节。对于瓦片而言 几何误差(geometricError) 属性量化了瓦片中内容和最高细节级别相比的简化程度。在第 5 章“几何误差”中会讲解具体用法，以便确定合适应考虑渲染子瓦片。

细化策略： 当具有一定细节级别的瓦片的视距误差超过阈值时，则考虑对子瓦片进行渲染。将来自子瓦片的附加细节并入渲染过程的方式由 细化策略(refine)属性决定。不同的细化策略会在第 6 章“细化策略”中进行讲解。

3. 边界包围体

每一个瓦片集是由一组以分层方式组织的瓦片构成。每一个瓦片都有一个相关联的边界包围体(bounding volume) 。这产生了可用于优化渲染和高效空间查询的分层空间数据结构。此外，每个瓦片都可以包含实际的可渲染内容，这些内容也具有边界包围体。与瓦片边界包围体相反，内容边界包围体仅与内容紧密相贴合，可用于可见性查询和视椎体剔除，以进一步提高渲染性能。

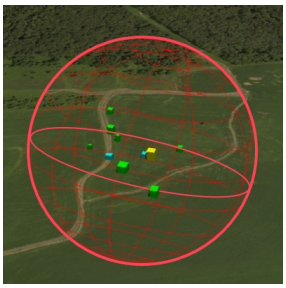
3D Tiles 格式支持不同类型的边界包围体，以及不同的层次组织策略。

边界包围体的类型

3D Tiles 支持不同类型的边界包围体，从而可以选择最合适底层数据结构的类型：

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

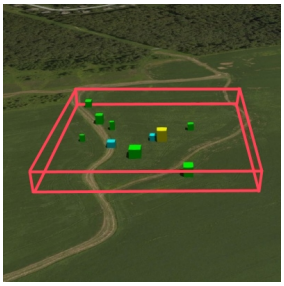
Center
Radius



包围球(sphere) 是一个简单的边界包围体，允许简单有效的交叉(相交)测试。它由球心位置和半径长度定义。

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

Center
x-axis
y-axis
z-axis



包围盒(box)是一种定向的边界框，使得边界更紧密地贴合常规几何体，如 CAD 模型。 它由包围盒中心位置和 3 个 3D 向量定义。这 3 个向量定义出 x、y、z 轴的方向和一半长。
(注：所谓半长即 长、宽、高的一半的长度)

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

West
South
East
North
Min. height
Max. height

包围区域(region)是一种描述特定区域的边界框，特别适用于地理信息系统，因为该区域由经纬度定义，用于描述该区域的西、南、东和北 4 个方位，加上最小和最大高度。

经度和纬度基于由 EPSG 4979 定义的 WGS 84 标准，单位为弧度。
最小和最大高度以 WGS 84 标准椭球的地面为准，单位为米。

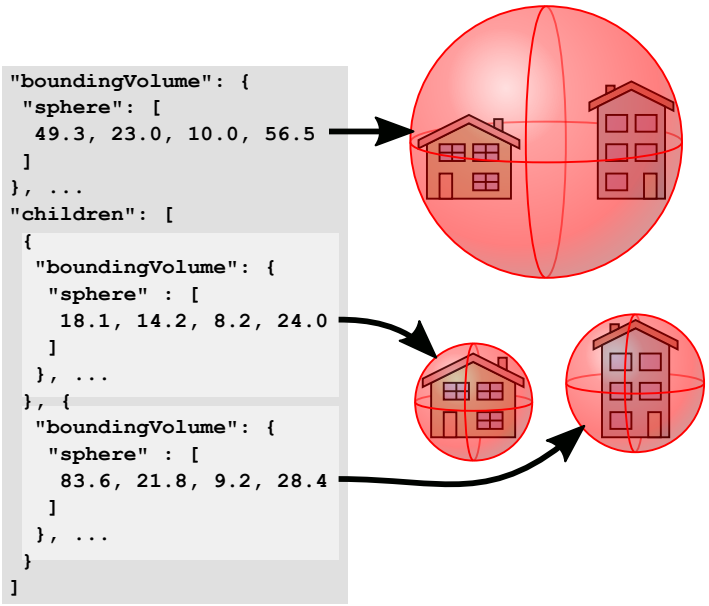
WGS84 和 EPSG4979 的详细信息，请参见第 13 章“通用定义”。

4. 空间数据结构

瓦片集中的瓦片被组织成树状数据结构。每个瓦片都有一个关联的边界包围体。这允许对不同的空间数据结构进行建模。以下内容将介绍可以使用 3D Tiles 建模的不同类型的空间数据结构。运行时引擎可以通用地使用空间信息以优化渲染性能，耳语空间数据结构的实际特征无关。

空间连贯

所有类型的边界包围体和空间数据都具有空间一致性。 这意味着父级瓦片的边界包围体始终包含其所有子项的内容。
空间连贯性对于可见性测试和相交测试至关重要。当某个对象不与其他父级瓦片边界包围体相交时，它也不会与该父级瓦片的任何子项内容相交。
相反，当某个对象确实与其他父级瓦片边界包围体相交时，则需测试与其他父级瓦片子项的边界包围体是否相交。



当必须测试对象与其他实际瓦片内容集合相交时，这可用于快速修剪层次结构的大部分，只留下几个叶子瓦片。

空间数据结构的类型

有不同的方式可以构建具有边界包围体的分层数据结构。这些方法通常会产生不同类型的空间数据结构，具体取决于用于构建的确切策略。

一个简单的策略为：通过沿特定轴递归拆分瓦片的内容，来构建简单的空间层次结构，直到满足停止标准。当内容在每个级别仅沿单个轴拆分时，结果是 k-d 树。3D Tiles 还支持多路 k-d 树，其中每个级别沿一个轴有多个拆分。

当内容在每个级别沿 x、y、z 轴拆分时，结果是一个 八叉树(octree)。当内容在内容边界包围体的中心被分割时，结果是一个 统一的八叉树(uniform octree)。当内容在不同的点被分隔时，结果是一个 不均匀的八叉树(non-uniform octree)。

在层次结构的每个级别上，可以纯粹从父节点的空间上创建边界包围体。或者，可以根据节点的实际内容计算每个节点的边界包围体。这对于稀疏数据集(sparse datasets)特别有用，因为它可以确保每个层级都 紧密贴合边界包围体。

有时需要在无法分割内容的单个特征(模型)的情况下分割瓦片。因此，可以构建 松散八叉树(loose octrees) 和 松散 k-d 树(loose k-d trees)，其中子项的边界包围体重叠，这些树仍然保持空间连贯性，因为父边界包围体仍然包裹着所有子项内容。

一般情况下，瓦片集可以被划分为 非均匀网格(non-uniform grid)。由于 3D Tiles 中空间层次结构中的每个节点有可能有任意数量的子节点，并且非叶子瓦片不需要包含可渲染的内容，因此网格单元仍然可以按层次结构组织，以便支持层次剔除。

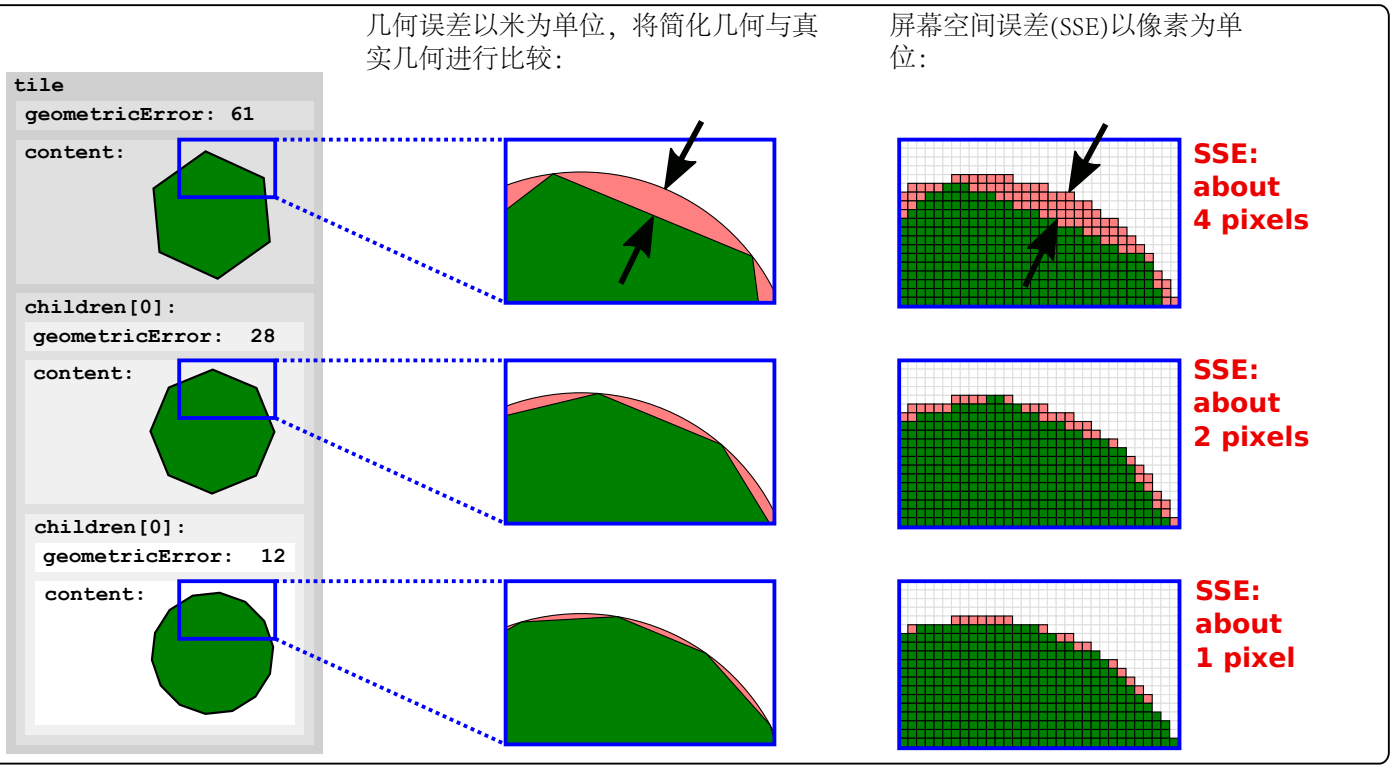
3D Tiles 中的通用空间数据结构允许对所有这些构造方法的结果进行建模。每个瓦片是层次结构中的一个节点，可以选择引入几何内容或仅存储包围其所有子项内容的边界包围体，并且可以在单个瓦片集中存储不同类型的边界包围体。

5. 几何误差

3D Tiles 的目标之一是有效地将大量数据集以流式传输到运行时引擎中，并且运行时可以有效呈现这些内容。因此，瓦片集中瓦片的层次结构包含了层次细节级别(HLOD)的概念：层次结构顶部的瓦片包含具有低细节级别的可渲染内容。子瓦片包含更高细节级别的内容。渲染运行时可以动态选择在性能和渲染质量之间提供最佳折中的细节级别。

确定应该渲染哪个细节级别的关键点是 几何误差(geometricError)。每个瓦片集和瓦片都有一个 geometricError 属性，它量化了简单几何与实际几何的误差。

运行时将此几何误差转换为屏幕空间误差(SSE：screen-space error)。屏幕空间误差(SSE)根据屏幕上的像素量化了多少几何误差是可见的。



当屏幕空间误差(SSE)超过某个阈值时，运行时将呈现更高级别的细节。对于瓦片集，几何误差用于确定是否应该渲染根瓦片。对于瓦片，几何误差用于确定是否应渲染瓦片的子项内容。

5.1 计算屏幕空间误差(SSE)

运行时可以使用几何误差提供的信息来找到性能和渲染质量之间的最佳折中：对于任何给定的瓦片集或瓦片，运行时可以确定有多少视觉效果，即使此时还没有实际下载或渲染瓦片内容；也可以通过渲染瓦片来提高视觉渲染质量。

为此，运行时必须计算给定几何误差的屏幕空间误差。以像素为单位的实际屏幕空间误差将取决于试图配置(查看器的位置和方向)，以及渲染最终图像的分辨率。

因此，屏幕空间误差(SSE) 的计算必须考虑这些因素。

最终的实现还取决于所使用的视图投影的类型。但对于标准透视投影，计算屏幕误差的一种可能方法如下：

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

其中 geometricError 是存储在瓦片集或瓦片中的几何误差(以米为单位)，screenHeight 是渲染屏幕的高度(以像素为单位)，tileDistance 是瓦片离眼睛的距离(以米为单位)，fovy 是 y 方向视椎体张开的角度(以弧度为单位)。

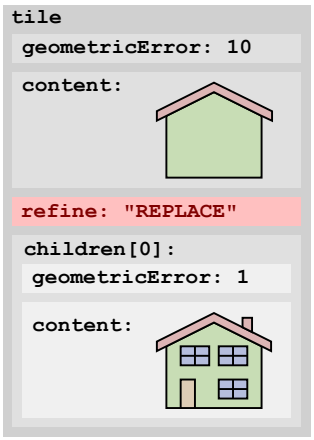
6. 细化策略

渲染瓦片集时，运行时会遍历瓦片的层次结构，检查每一个瓦片的几何误差，并计算相关的屏幕空间误差。当此屏幕空间误差超过某个阈值时，运行时将递归考虑对子项进行渲染。包含可渲染内容的子项瓦片具有更高级别的细节和更小的屏幕空间误差。瓦片子项的内容可以通过两种细化策略中的一种来增加细节级别。细化策略由 refine 属性值决定，该属性值可以是 REPLACE(替代) 或 ADD(添加)。

替代:

子瓦片包含父瓦片更高细节级别的**完整内容**。

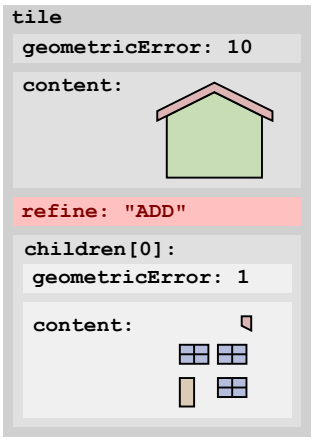
渲染子瓦片时，此内容将替换父瓦片的内容。



添加:

子瓦片包含父瓦片详细的**其他内容**。

渲染子瓦片时，此内容将在父瓦片的内容之上呈现。



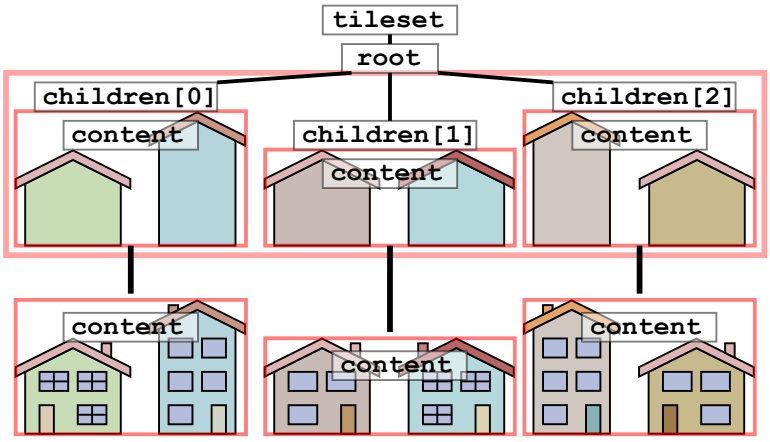
地形中的根瓦片必须设置 细化(refine)属性，子瓦片也可以单独设置自己的细化属性。若子瓦片未设置细化属性则会继承于该子瓦片的父级细化属性。

7. 使用 3D Tiles 优化渲染

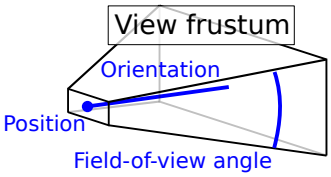
3D Tiles 是专门为了高效渲染和流式从传输海量异构数据集而设计的。以下简要得讲述一下运行时如何使用 3D Tiles 的概念来交互式渲染巨大的瓦片集的。我们将展示瓦片集和瓦片的层次结构、相关的边界包围体、几何误差，以及不同的细化策略如何一起发挥作用。

下面示例中的瓦片集包含一组建筑物，每个元素的边界包围体以红色显示。

瓦片集(tileset)包含一个根瓦片(root)，而根瓦片(root)又包含三个子瓦片(children)。每个子瓦片都包含可呈现的内容(content)。
在示例中，每个子瓦片的内容由两个建筑物组成，存储为批处理的 3D 模型。在此级别(children)，内容存储的细节级别较低，这意味着这些子瓦片具有较高的几何误差。
每个子瓦片拥有更详细的相似内容(content)，这些内容(content)具有更低的几何误差。



在渲染期间，运行时会维护一个视椎体(view frustum)，该视椎体由相机的位置(position)、方向(orientation)和视场角定义(field-of-view angle)。可以检测视椎体与瓦片集、瓦片边界包围体的交集。

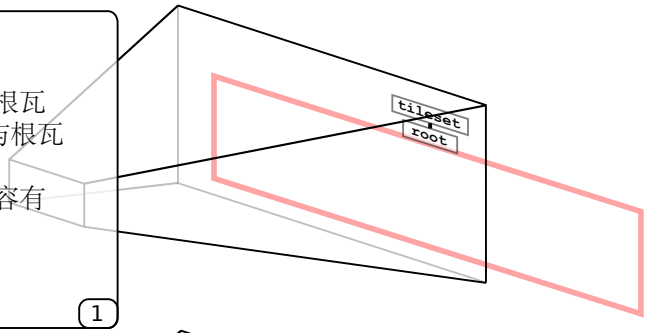


7.1 3D Tiles渲染优化：流程示意

一开始，运行时会加载瓦片集对应的 JSON 文件，并检测视椎体与根瓦片(root)的边界包围体的交集。在右侧这个示例中，检测到视椎体与根瓦片确实相交，这意味着该瓦片集可以被考虑用于渲染。
此时，还没有加载可渲染的内容：瓦片集 JSON 更加详细的信息内容有待分析，以确定是否有内容会需要被渲染。

(视椎体检测到瓦片集相交)

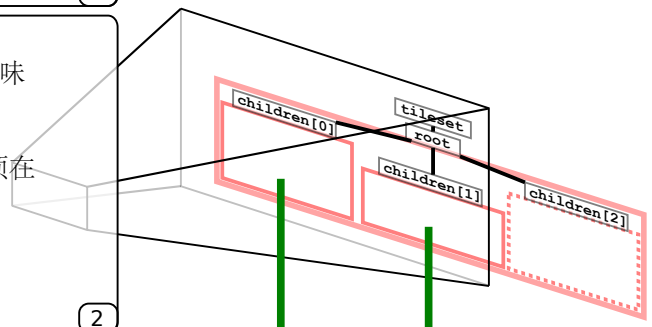
1



接下来，运行时检测所有子瓦片(children)是否与视椎体相交。
在右侧示例中，检测到视椎体与三个子瓦片中的其中两个相交。意味着相交的这两个子项的内容必须要加载(load)。

(视椎体检查到与前两个子项相交，那么开始分析并加载这两个子项在当前级别中的内容)

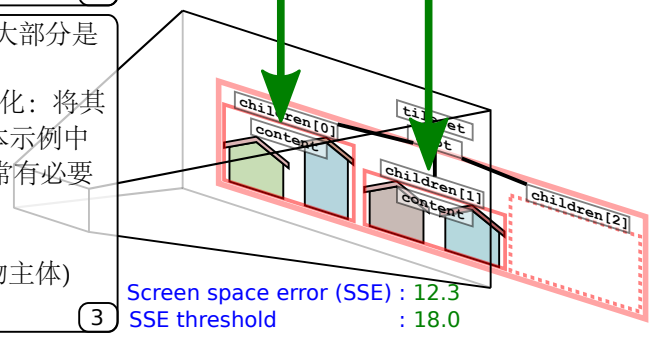
2



在此示例中，建筑物(content)处于低细节级别。因此即使瓦片集的大部分是可见的，由于其低复杂性，运行时也可以有效地渲染内容。
在渲染期间，运行时会观察与需要渲染的瓦片的几何误差的相对变化：将其转换为屏幕空间误差，以估算视觉表示的呈现效果。只要阈值(在本示例中阈值被设置为 18.0) 不被超过，不渲染这些内容对于优化渲染是非常有必要的。

(渲染此级别中的两个子项内容，请注意此级别中的内容仅为建筑物主体)

3

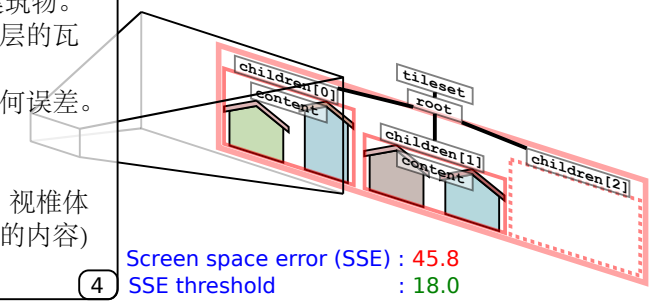


Screen space error (SSE) : 12.3
SSE threshold : 18.0

用户可以 与渲染的瓦片集进行交互。例如，用户可以放大到某个建筑物。这会导致屏幕空间误差增大，当超过一定阈值时，会考虑渲染下一层的瓦片。
这些瓦片包含相同内容，具有更高级别的细节，因此具有更小的几何误差。运行时只需加载和渲染与新视椎体相交的瓦片内容。

(当用户放大视图，将进入下一个视觉级别，此时视椎体发生变化，视椎体检测到只与第一个子项相交，因此开始加载该子项下一级别更详细的内容)

4

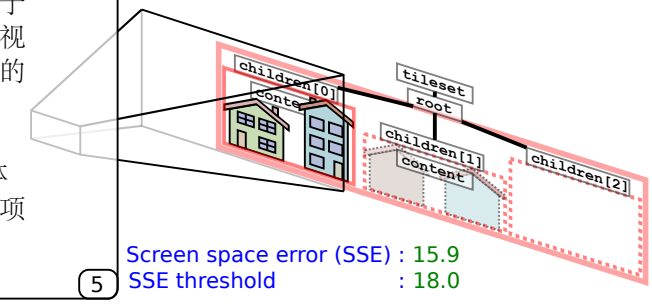


Screen space error (SSE) : 45.8
SSE threshold : 18.0

根据选定的细化策略加载和渲染具有更高细节级别的视觉内容。由于其较低的几何误差，导致屏幕空间误差低于阈值，这意味着更高的视觉渲染要求。但是此时只有一小部分的瓦片是可见的，因此高细节的内容仍然可以有效渲染。

(此时只渲染第一个子项内容，且在这个级别中，会渲染建筑的主体和门窗。尽管需要渲染的局部内容变多了，但是由于此时第二个子项是不需要被渲染的，因此渲染依然会是流畅的。)

5



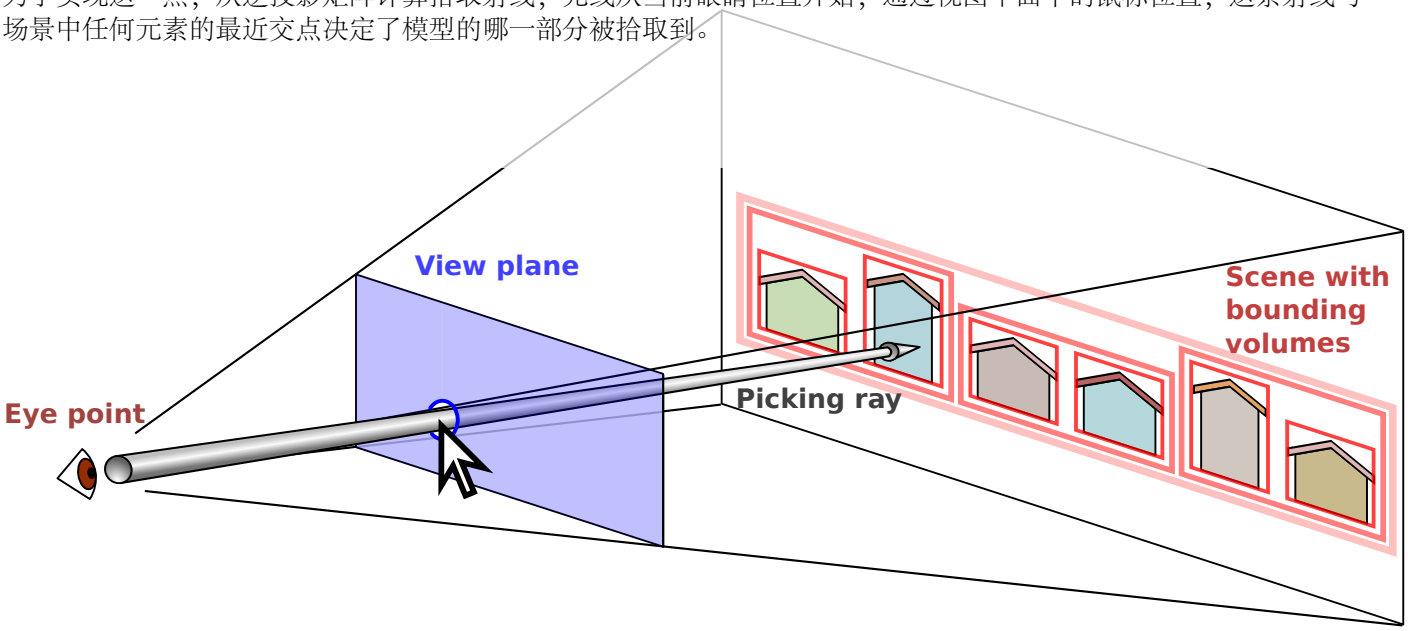
Screen space error (SSE) : 15.9
SSE threshold : 18.0

上面渲染流程示例讲解了 3D Tiles 如何在任何规模下做到了 渲染性能 和 视觉质量(细腻程度)的平衡。运行时可以检测视椎体是否与瓦片的边界包围体相交，并且只有在相交情况下才会加载和渲染瓦片内容。运行时最初只显示具有低细节级别的瓦片内容，并且仅在屏幕空间误差超过阈值时(例如当用户放大到接近一倍时)才会按需下载和渲染更高级别细节的内容。

8. 3D Tiles 的空间查询

具有边界包围体的瓦片的层次结构允许有效的空间查询。 空间查询的一个示例是光线投射(ray casting)：当运行时渲染瓦片集时，用户可以通过选择单个模型或 特征与场景进行交互，例如突出显示模型或显示元数据。

为了实现这一点，从逆投影矩阵计算拾取射线，光线从当前眼睛位置开始，通过视图平面中的鼠标位置，这条射线与场景中任何元素的最近交点决定了模型的哪一部分被拾取到。

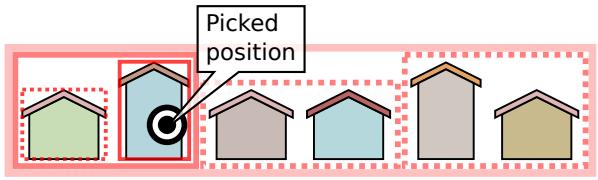


从瓦片集的根瓦片开始进行相交检测。当光线与该图块的边界包围体相交时，会检测光线与每一个子瓦片是否相交。当找到与子瓦片的相交点时，可以检测该瓦片内容的实际几何体与射线的交点，以确定实际拾取位置。

在相交检测期间，树(tree)的空间一致性使得可以快速修剪层次结构中不相交的部分：当射线与父瓦片的边界包围体不相交时，可以跳过对应子瓦片的相交检测。
在右侧示例中，使用虚线显示的边界包围体不与拾取射线相交，遍历可能会在那里停止。

用于查找被射线选中的瓦片的伪代码:

```
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;
}
```



对于某些类型的瓦片，使用鼠标不足以可以获取到是否被拾取：例如 成批3D模型 .b3dm (第10.2节) 或分批点云(第10.4节)，多个不同的模型或模型的一部分(特征)可以组合成一个单一的几何图形。在这些情况下，几何体的顶点会使用 批次(batch) ID 进行扩展，用于标识要素。这使得可以通过鼠标点击拾取到相关特征。

9. 瓦片格式：介绍

瓦片的可渲染内容由瓦片的 JSON 中的 URI 属性记录引用。对于不同的模型类型，有以下几种不同的格式：

- **批处理 3D 模型**：异构模型，例如纹理地形或 3D 建筑
- **实例化 3D 模型**：同一个 3D 模型的多个实例
- **点云**：大量的点

瓦片集可以包含瓦片格式的任意组合。允许将具有不同格式的瓦片组合成复合瓦片(第10.5节)。

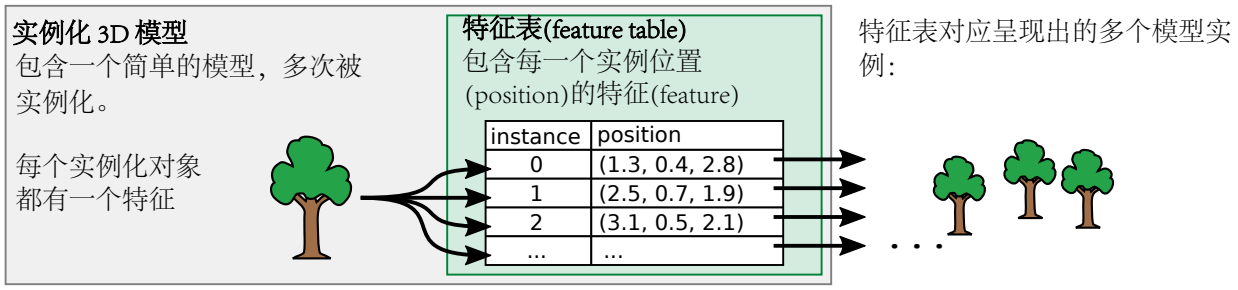
瓦片的可渲染内容包含不同的功能特点。对于批处理 3D 模型，几何的每个部分都可能是一个特征。例如当多个建筑物组合成一个批处理 3D 模型时，每个建筑物都可能是一个特征。对于实例化 3D 模型，每一个实例都是一个特征。对于点云有两个选项：特征可以是单个点，也可以是代表模型可识别部分的一组点。

(这里提到的 特征 是指该模型内容的一些属性参数，例如坐标等)

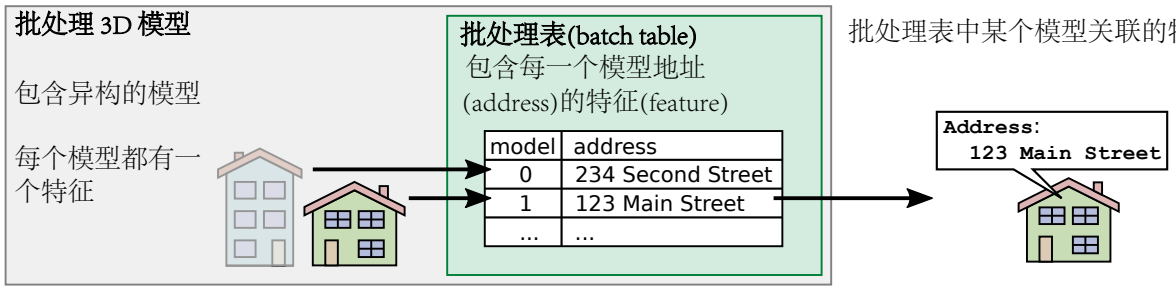
特征表 与 批处理表

所有切片格式(符合切片除外)的一个共同元素是 特征表(feature table) 和 批处理表(batch table)。特征表 和 批处理表 的确切内容取决于瓦片格式，但他们的结构和布局对于所有瓦片格式都是相同的。

特征表包含渲染特征所需的属性。例如，在实例化 3D 模型中，实例的位置存储在特征表中：

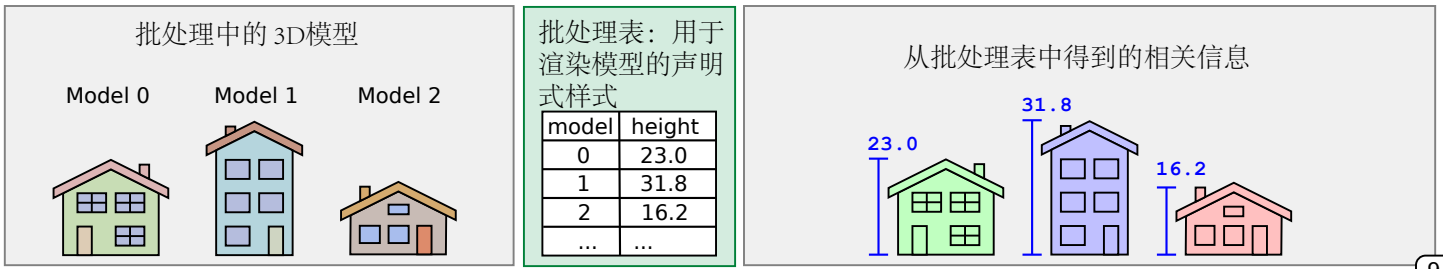


批处理表(batch table)可以包含每个特征的附加的、特定于应用程序的属性。例如在批处理 3D 模型中，与每个模型关联的元数据：



外观：声明式样式

存储在批处理表中的信息不是渲染所必须的。但是批处理表通常包含用于声明式样式的元数据，在第 11 章中会有详细介绍。例如，批处理表可能包含瓦片中的一组建筑物的高度信息。可以在 3D Tiles 样式语言中访问此信息，以修改模型的外观。例如，建筑物可以根据其高度使用不同的颜色进行渲染：

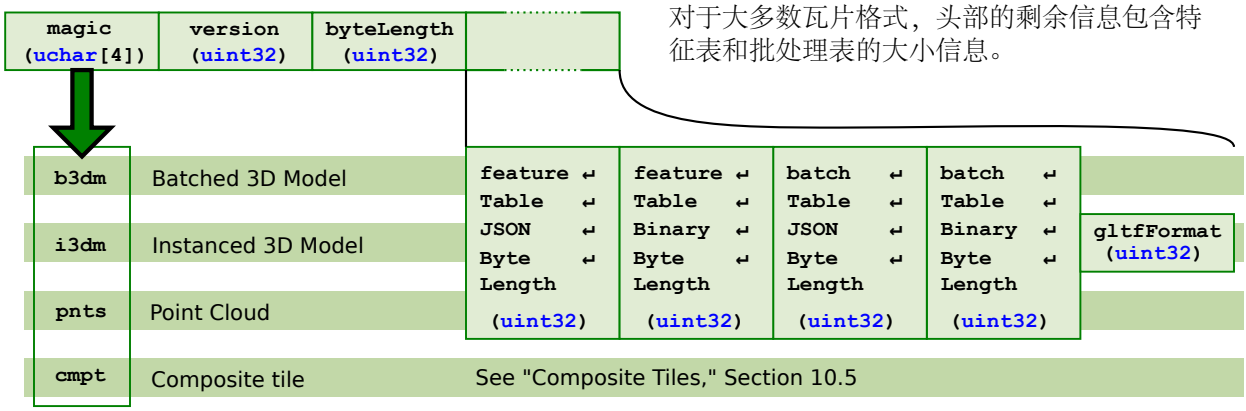


10. 瓦片格式

瓦片的实际可渲染内容存储为二进制 blob，这个 blob 由一个带有结构信息的头(header)和一个包含实际负载的主体(body)组成：

瓦片头部信息

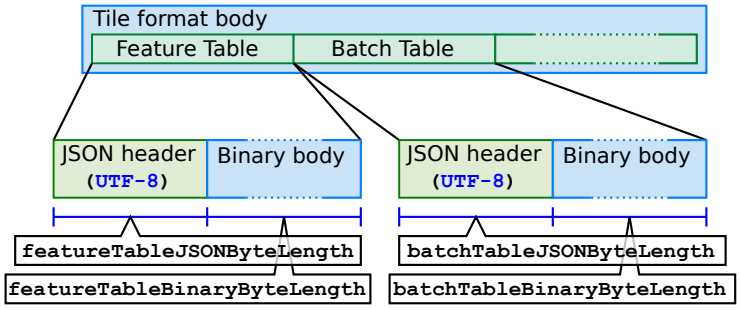
每个瓦片格式的头部信息以四个简写字符(four magic bytes)的序列开始。这些字符表示一个 4 位的字符串，用于确定瓦片格式。他们之后是瓦片格式的版本号，以及瓦片数据的总长度，以字节为单位。
头部信息的确切大小、内容和结构取决于瓦片格式，在读取简写字符(magic bytes)后即可知道。瓦片格式以及瓦片的确切数据布局显示在以下部分中，这些部分解释了每一种瓦片格式。



对于 featureTableJSONByteLength、featureTableBinaryByteLength、batchTableJSONByteLength 和 batchTableBinaryByteLength 描述的大小(以字节为单位) 对应的特征表和批处理表的各部分的总和，是指包含瓦片格式体实际表数据。

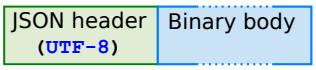
瓦片主体信息

瓦片主体信息包含瓦片数据的实际有效载荷。对于所有瓦片格式(复合瓦片除外)，该主体可能包含一个特征表和批处理表，以及其他二进制数据，用于特定瓦片格式。
特征表 和 批处理表每个都包含一个 JSON 头部信息和一个二进制体的长度。
每个元素由瓦片格式头部的信息决定。

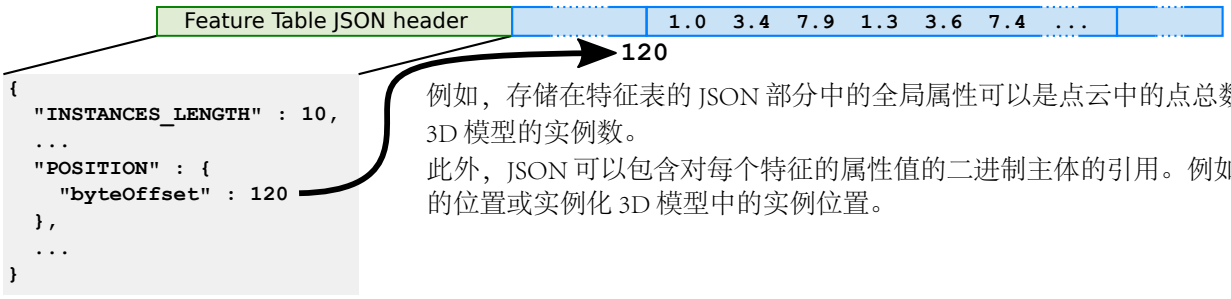


10.1. 瓦片格式：特征表和批处理表

特征表 和 批处理表 存储在瓦片数据的主体中。它们具有相同的结构：一个由 JSON 字符串描述的头部信息(header)和一个由二进制数据组成的主体(body)：



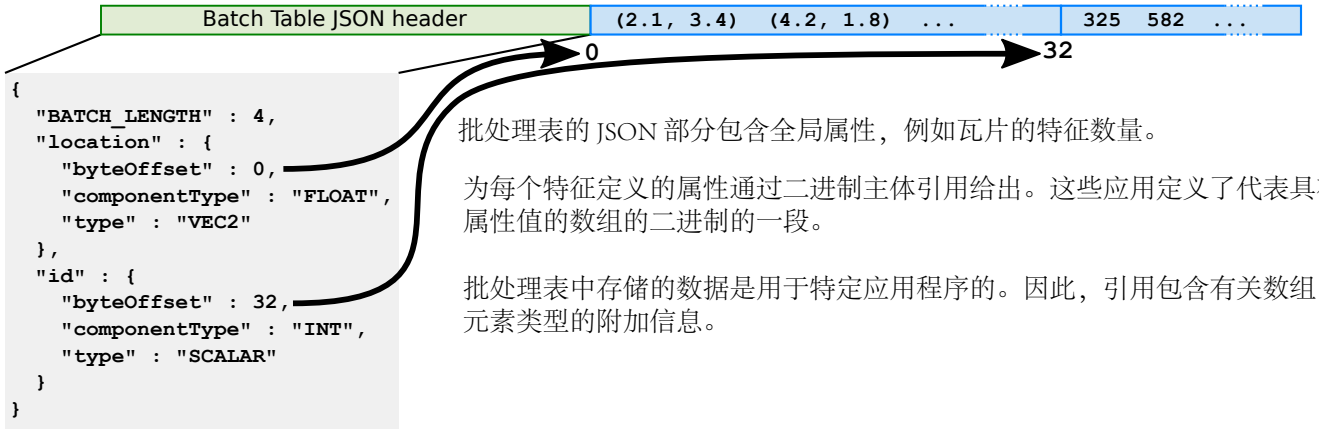
JSON 可能包含引用整个瓦片的全局属性。此外 JSON 可能包含对二进制主体(body)的引用。支持的确切属性集取决于瓦片格式，但所有特性都定义了二进制主体的元素，这些元数包含瓦片中显示的每个特征的值数组。



例如，存储在特征表的 JSON 部分中的全局属性可以是点云中的点总数，或是实例化 3D 模型的实例数。

此外，JSON 可以包含对每个特征的属性值的二进制主体的引用。例如，点云中心点的位置或实例化 3D 模型中的实例位置。

一个属性所指的二进制主体部分由主体内的 字节偏移索引(byteOffset) 决定的。数据的类型取决于属性语义。例如，POSITIONS 属性是指表示 32 位浮点值数组的主体部分，用于表示位置的 x、y 和 z 坐标。



批处理表的 JSON 部分包含全局属性，例如瓦片的特征数量。

为每个特征定义的属性通过二进制主体引用给出。这些应用定义了代表具有属性值的数组的二进制的一段。

批处理表中存储的数据是用于特定应用程序的。因此，引用包含有关数组中元素类型的附加信息。

二进制主体中数据数据的位置由 byteOffset 给出。类型表示数组的元素是标量还是向量。componentType 是定义标量或向量分量的类型。下表包含每个组件类型包含的字节数，以及每个类型包含的元素数。这可用于计算二进制主体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. 瓦片格式：批处理 3D 模型(b3dm)

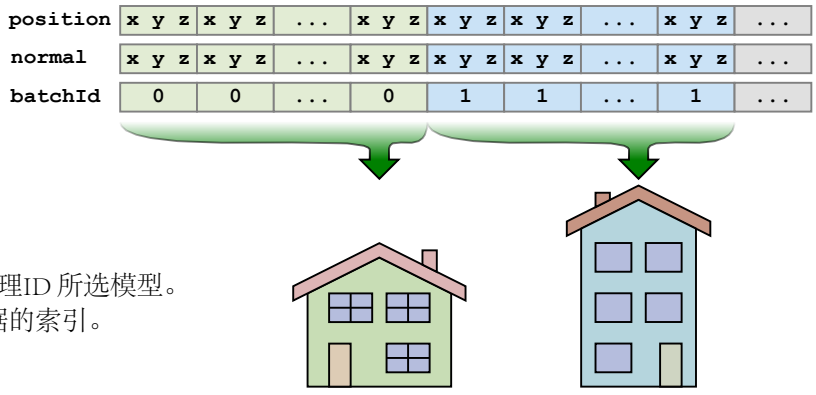
批处理 3D 模型瓦片包含异构模型的渲染效果。这些模型可以是地形或 3D 建筑，例如用于建筑信息管理(BIM)或工程的应用程序。

批处理 3D 模型中的是合集渲染数据存储为二进制 glTF——GL传输格式的二进制形式。这种格式允许单个模型甚至带有纹理和动画的完整 3D 场景，且文件体积小，可以通过网络高效传输，并由运行时引擎直接渲染。glTF中的集合数据存储在缓冲区中，该缓冲区的结构是将缓冲区划分为不同部分，分别代表不同属性，如顶点位置或法线。第 13章 “通用定义”中提供了有关 glTF 的更多资源的摘要和链接。

术语“批处理”是指多个模型的几何数据，每个模型由顶点位置、可选的法线和纹理坐标组成，可以合并到一个缓冲区中，以提高渲染性能：单个模型的数据 buffer 可以直接赋值到 GPU 内存中，从而减少赋值操作，并且可以将其构建为最小化绘制调用次数的形式。

当多个模型组合成一个缓冲区时，仍然必须支持单个模型的样式和交互。例如可以通过用不同的颜色渲染模型来突出显示模型或通过单击鼠标确定已选择哪个模型。在 3D Tiles 中，这是通过使用额外的顶点属性扩展缓冲区来实现的。

模型的几何数据使用 batchId 属性来进行标记和区分。它将每个顶点的 批次ID(batchID) 存储为整数，具有相同 ID 的顶点是同一模型的一部分。



批次ID(bitchID)可用于识别模型的交互或样式：
当用户单击批处理 3D 模型时，运行时可以确定 批处理ID 所选模型。
在批处理表中，批处理ID 用作查询样式信息或元数据的索引。

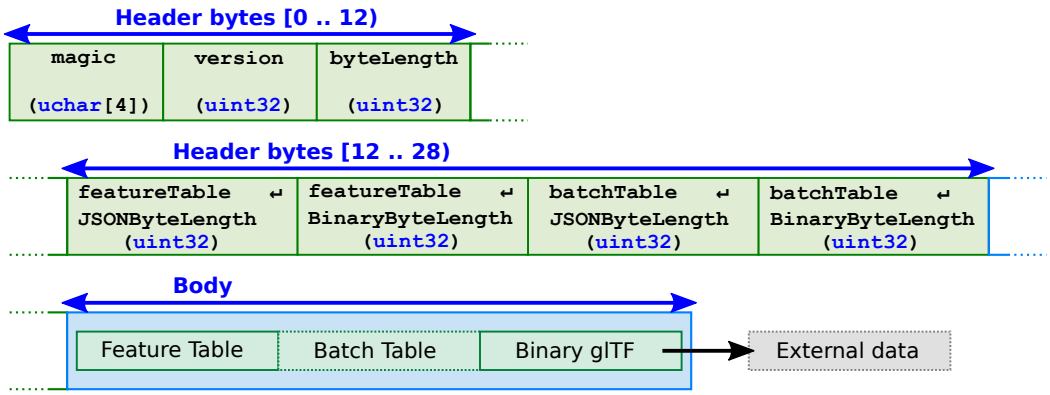
批处理 3D 模型：属性

批处理 3D 模型的特征表仅包含全局属性。BATCH_LENGHT 指批处理表中可区分模型的总数量，RTC_CENTER 用于保存相对给定中心位置时的中心位置。

属性	类型	描述
BATCH_LENGTH	**uint32**	批处理中可区分模型(特征)的数量。如果二进制 glTF 没有 batchID 属性，则该字段必须为 0
RTC_CENTER	**float32[3]**	相对给定中心位置时的相对中心位置

批处理3D模型：数据结构

下图显示了在批处理 3D 模型的头部和主体中，有关数据结构和类型的示意：

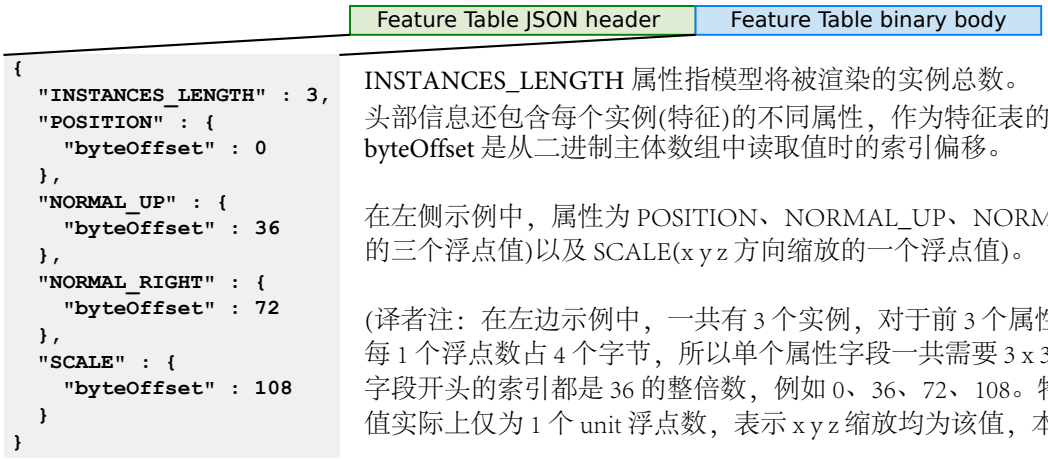


10.3. 瓦片格式：实例化 3D 模型(i3dm)

在许多应用场景中，复杂场景中多次包含相同的模型，这些模型的变化很小，例如树木、CAD 功能(如螺栓) 或 室内设计元素(如家具)。

因此，3D Tiles 支持实例化(instanced) 3D 模型，其中单个模型被多次使用和渲染。该模型的每个外观都是一个示例(或特征)，并且这些实例可以通过不同的变换进行渲染——例如，在不同的位置。

模型被存储为二进制 glTF(请参阅“第 13 章 通用定义”)。它可以直接存储在二进制中，或者是二进制 glTF 文件的 URI。有关将渲染多少个实例以及这些实例的位置和方向的信息存储在特征表中。特征表的第一部分包含 JSON 数据，第二部分包含二进制数据。



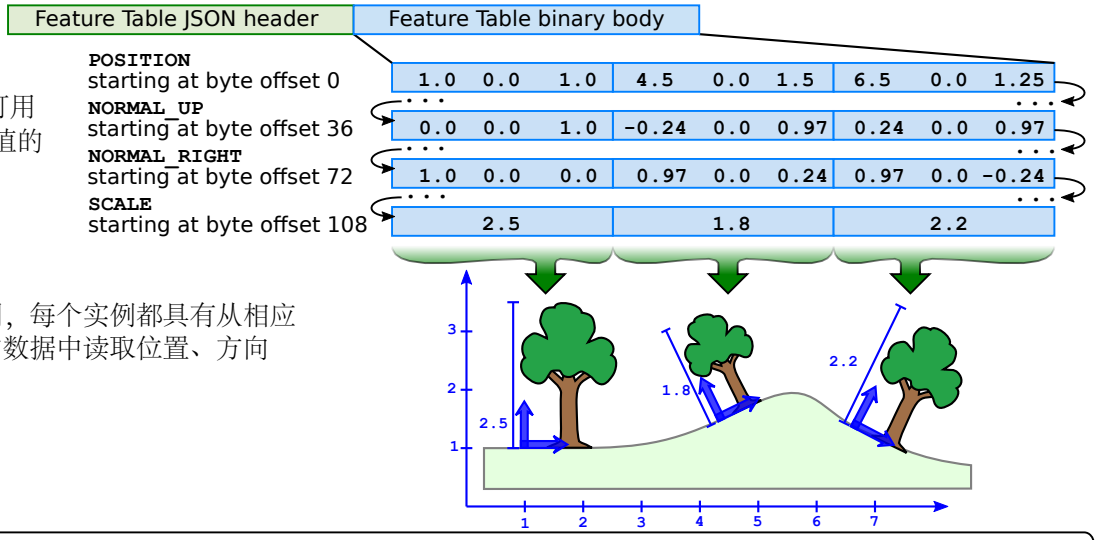
INSTANCES_LENGTH 属性指模型将被渲染的实例总数。

头部信息还包含每个实例(特征)的不同属性，作为特征表的二进制主体的引用。byteOffset 是从二进制主体数组中读取值时的索引偏移。

在左侧示例中，属性为 POSITION、NORMAL_UP、NORMAL_RIGHT(每个由 x y z 坐标的三个浮点值)以及 SCALE(x y z 方向缩放的一个浮点值)。

(译者注：在左边示例中，一共有 3 个实例，对于前 3 个属性字段需要 3 个 uint 浮点数，每 1 个浮点数占 4 个字节，所以单个属性字段一共需要 3 x 3 x 4 = 36 个字节，因此每一个字段开头的索引都是 36 的整倍数，例如 0、36、72、108。特别说明第 4 个字段 SCALE 的值实际上仅为 1 个 unit 浮点数，表示 x y z 缩放均为该值，本小节后面会有进一步说明)。

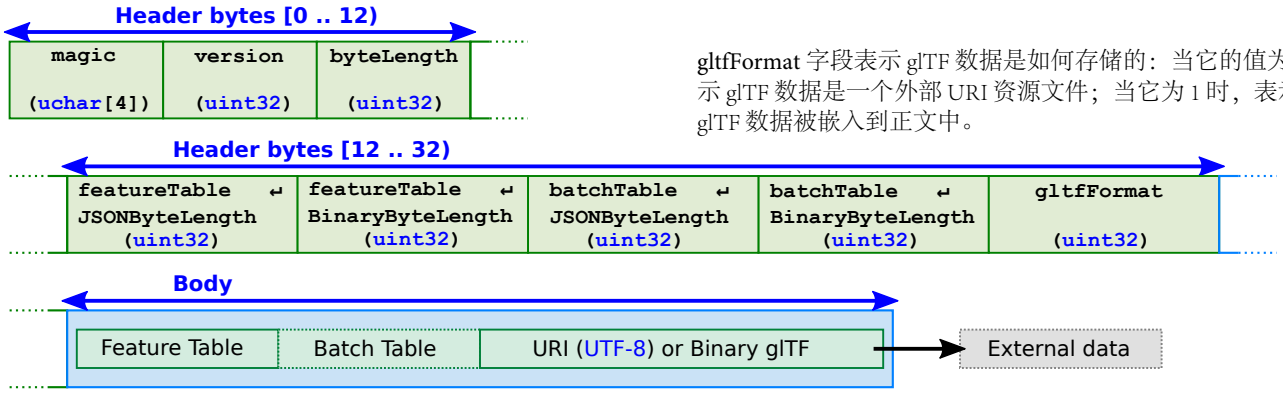
来自 JSON 部分的属性信息可用于访问表示每个实例的属性值的数组的二进制主体字段。



将同一个模型渲染成多个实例，每个实例都具有从相应实例的特征表二进制部分中的数据中读取位置、方向(法线)和缩放。

实例化 3D 模型：数据结构

下图显示了在实例化 3D 模型的头部和主体中，有关数据结构和类型的示意：



gltfFormat 字段表示 glTF 数据是如何存储的：当它的值为 0 时，表示 glTF 数据是一个外部 URI 资源文件；当它为 1 时，表示二进制 glTF 数据被嵌入到正文中。

实例化 3D 模型：属性

实例化 3D 模型的特征表可能包含以下属性：

实例总数 使用 INSTANCES_LENGTH 属性用来存储模型被实例化的总数。这决定了存储每个实例属性的数组的长度。在批处理表存储每个实例的元数据的情况下，它还确定批处理表的大小。

批处理ID 使用 BATCH_ID 属性来存储每个实例的标识符，该批处理 ID 被当做索引，可用于在批处理表中查找每个实例的属性。

位置 使用 POSITION 属性来存储位置信息，即 x y z 对应的坐标值。

相对中心点 或者使用 POSITINS_QUANTIZED 属性来存储相对中心位置。
也可以使用 RTC_CENTER 属性来存储相对中心位置。

量化的体积 使用 POSITIONS_QUANTIZED 属性存储相对量化的体积。由 QUANTIZED_VOLUME_OFFSET 定义偏移，由 QUANTIZED_VOLUME_SCALE 定义缩放。

方向 每个实例的方向可以用两个向量定义：
使用 NORMAL_UP 定义向上方向，使用 NORMAL_RIGHT 定义向右方向。
或者使用压缩的 oct 编码的法线表示，对应的为 NORMAL_UP_OCT16P 和 NORMAL_RIGHT_16P

默认方向 如果单个实例没有指定方向，则使用 EAST_NORTH_UP 属性存储指示每个实例默认的方向，该方向为 WGS84 标准椭球上的 东/北/上 参考系的方向。

缩放 使用 SCALE 属性存储每个实例的缩放信息，请注意 SCALE 为一个单值，暗含的意思为 x y z 三个缩放分量均为 SCALE 对应的值。若想表达 x y z 均不相同，则使用 SCALE_NON_UNIFORM 属性来存储。

有关量化位置、otc 编码的法线、相对中性位置的概念，以及 WGS84 椭球的更多信息，会在第 13 章“通用定义”中找到。

实例化 3D 模型：属性摘要

下面总结了实例化 3D 模型的特征表中可能包含的属性。

全局属性：

属性	类型	描述
INSTANCES_LENGTH	**uint32**	实例中个数
QUANTIZED_VOLUME_OFFSET	**float32[3]**	相对中心位置偏移
QUANTIZED_VOLUME_SCALE	**float32[3]**	相对缩放
EAST_NORTH_UP	**boolean**	实例方向是否使用 WGS84 椭球上的 东/北/上 坐标参考系
RTC_CENTER	**float32[3]**	相对中心位置

私有属性：

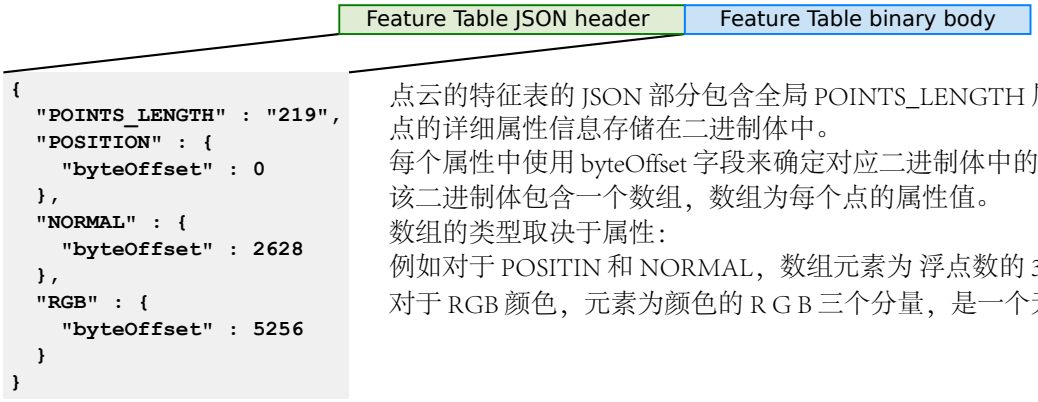
每个属性都是一个参考到特征表的二进制主体的一部分。此部分包含具有实际属性值的数组数据。
数组元素的数据类型由属性的类型决定。
数组的长度有实例的数量决定。

属性	类型	描述
POSITION **POSITION_QUANTIZED**	**float32[3]** **uint16[3]**	实例的位置
NORMAL_UP **NORMAL_UP_OCT32P**	**float32[3]** **uint16[2]**	实例向上方向的单位向量
NORMAL_RIGHT **NORMAL_RIGHT_OCT32P**	**float32[3]** **uint16[2]**	实例向右的单位向量
SCALE	**float32**	x y z 三轴相同的缩放值
SCALE_NON_UNIFORM	**float32[3]**	x y z 三轴不同的缩放值
BATCH_ID	**uint8/16/32**	批处理ID，用于在批处理表中查找示例的元数据

10.4. 瓦片格式：点云(pnts)

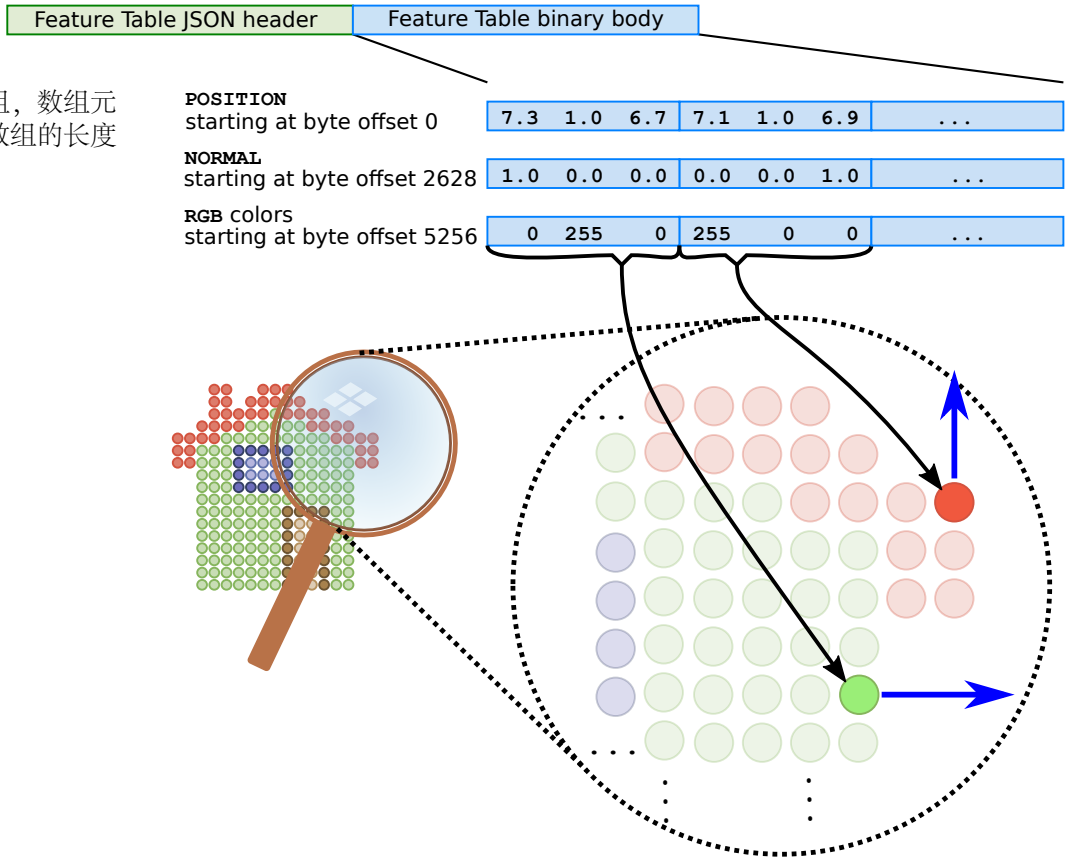
从建筑物或环境等现有结构获取 3D 数据的常用方法是通过摄影测量或激光雷达扫描。此获取过程的结果是一个点云，其中每个点由位置和定义其外观的附加属性构成。

点云(Point Clouds)是 3D Tiles 支持的一种格式，它允许流式传输大量点云以进行 3D 可视化。
有关点的位置和其他视觉属性的信息存储在特征表中，该表由 JSON 头部和二进制主体组成。




```
{
  "POINTS_LENGTH" : "219",
  "POSITION" : {
    "byteOffset" : 0
  },
  "NORMAL" : {
    "byteOffset" : 2628
  },
  "RGB" : {
    "byteOffset" : 5256
  }
}
```

点云的特征表的 JSON 部分包含全局 POINTS_LENGTH 属性，用于表示点的总数量。
点的详细属性信息存储在二进制体中。
每个属性中使用 byteOffset 字段来确定对应二进制体中的开始索引位置。
该二进制体包含一个数组，数组为每个点的属性值。
数组的类型取决于属性：
例如对于 POSITIN 和 NORMAL，数组元素为 浮点数的 3D 向量；
对于 RGB 颜色，元素为颜色的 R G B 三个分量，是一个无符号的 uint8。



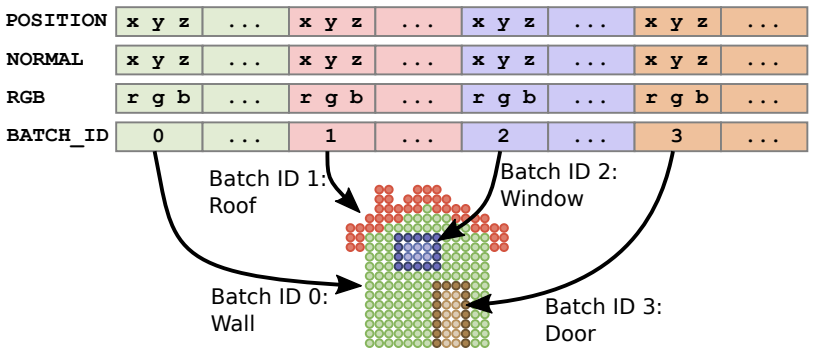
每个部分代表一个属性值数组，数组元素的类型取决于属性类型。数组的长度由点的总个数决定。

批处理点云

在点云中，可以定义代表不同特征的点组。例如，可能存在代表门、窗户或房顶的点组。
在 3D Tiles 中，这些组是通过为点分配 批处理ID 来定义的，类似于用于批处理的 3D 模型的概念。

对于批处理点云，特征表的 JSON 内容包含了一个 BATCH_LENGTH 属性，它表示点组的总数量。同时 JSON 还包含 BATCH_ID 属性，它指的是二进制主体的一部分，该部分包含带有点的批处理ID 的数组，存储为 8、16、或 32 位整数值。
可以将批处理ID当做索引，在批处理表中查找该组点的元数据。

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



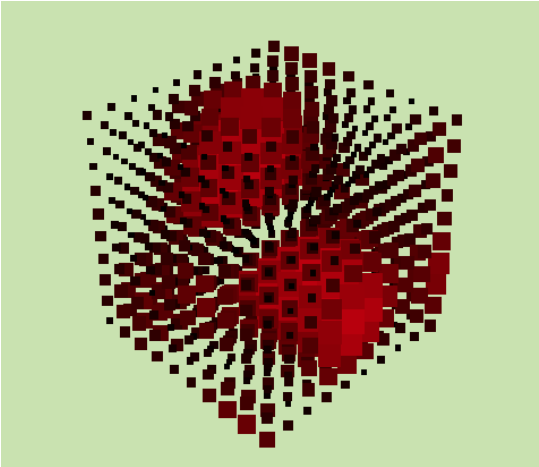
外观：点云的样式声明

样式声明，如第 11 章所示，可以应用点云以促进不同类型的信息可视化。
例如，点云可能由规则网格组成，填充特定体积的点。然后批处理表可以包含在相应位置进行检测。

在右侧示例中，每个点都有一个关联的 温度(temperature)值存储在批处理表中。然后可以将批处理表中的特定信息映射到点的视觉属性 3D Tiles 样式语言中。

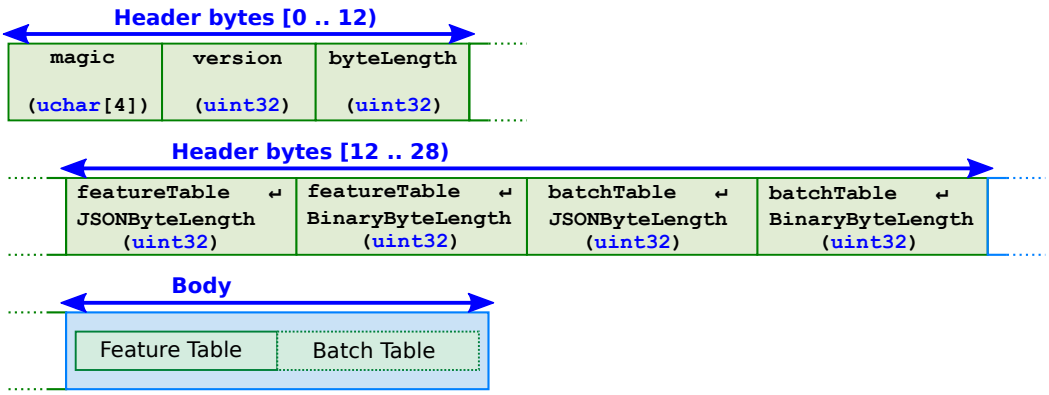
这里温度属性影响渲染点的颜色和大小：

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



点云：数据结构

下图显示了在点云的头部和主体中，有关数据结构和类型的示意：



点云：属性

在特征表中，点云可能存在以下属性：

点云总个数 使用 POINTS_LENGTH 属性存储点云总个数。这也决定了存储每个点属性的数组长度。

批处理ID总数 使用 BATCH_LENGTH 属性存储批次(点组)总个数。它是所有唯一值 BATCH_ID 的总个数。

批处理ID 使用 BATCH_ID 属性存储某个点组的唯一索引 ID。所有具有相同 BATCH_ID 的点都属于同一个点组。

位置 使用 POSITION 属性存储点的位置信息。或使用 POSITINS_QUANTIZED 属性存储压缩、量化后的位置信息。

相对中心点 使用 RTC_CENTER 属性存储相对于中心点位置信息。

量化体积 或使用 POSITIONS_QUANTIZED 属性村塾压缩、量化后的位置信息，使用 QUANTIZED_VOLUME_OFFSET 定义位置偏移量，使用 QUANTIZED_VOLUME_SCALE 定义缩放。

法线 使用 NORMAL 属性定义点的法线单位向量。
或使用 NORMALS_OCT16P 属性存储压缩的、oct 编码的法线。

颜色 使用 RGB 或 RGBA 属性来表示点的颜色。
或使用 RGB565 属性存储压缩的颜色值。

恒定颜色 使用 CONSTANT_RGBA 属性存储默认点的颜色值。当没有定义某个点的颜色时，将使用该颜色值作为这个点的 颜色。

有关量化位置、otc 编码的法线、相对中心位置的概念的更多信息，可以在第 13 章“通用定义”中找到。

点云：属性总结

下表总结了可能包含在点云特征表中的属性。

全局属性：

属性	类型	描述
POINTS_LENGTH	**uint32**	点云总个数
QUANTIZED_VOLUME_OFFSET	**float32[3]**	量化的 x y z 轴偏移量
QUANTIZED_VOLUME_SCALE	**float32[3]**	量化的 x y z 轴缩放量
CONSTANT_RBGA	**uint8[4]**	默认点的颜色
BATCH_LENGTH	**uint32**	批次ID的总个数
RTC_CENTER	**float32[3]**	相对于中心点的位置

私有属性：

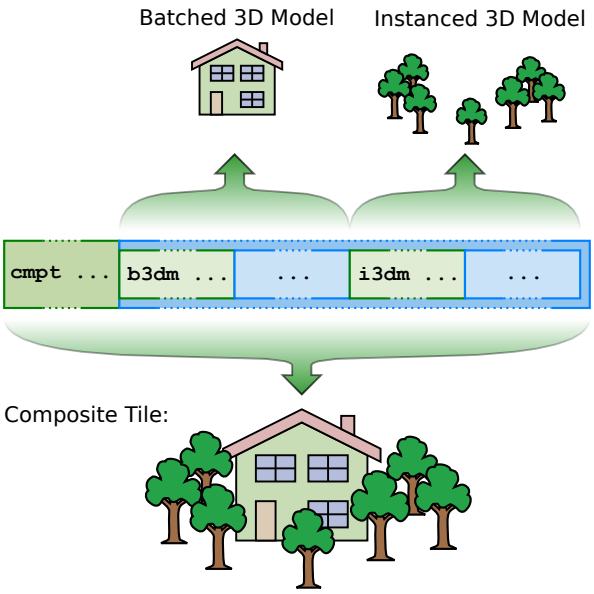
每个属性是对特征表二进制主体部分的引用。此部分包含具有实际属性值的数组数据。
数组元素的数据类型由属性的类型决定。数组的长度有点云中点的总个数决定。

属性	类型	描述
POSITION **POSITION_QUANTIZED**	**float32[3]** **uint16[3]**	点的位置，即坐标 x y z 对应的值。
RGBA	**uint8[4]**	点的 RGBA 颜色值
RGB	**uint8[3]**	点的 RGB 颜色值
RGB565	**uint16**	使用 RGB颜色的 16 位压缩形式，其中 5 位代表红色，6 位代表绿色，5 位代表蓝色
NORMAL **NORMAL_OCT16P**	**float32[3]** **uint8[2]**	定义点的法线的单位向量
BATCH_ID	**uint8/16/32**	批处理ID，用于查找属于该点组中点的元数据。

10.5. 瓦片格式：复合瓦片(cmpt)

3D Tiles 支持流式异构数据集。多种瓦片格式可以组合在一个瓦片集中。还可以在复合瓦片(composite tiles)中组合不同格式的多种瓦片以获得额外的灵活性。

我们拿这个符合瓦片的场景来举例。
在地里空间应用程序中：
一组建筑物可以存储在批处理 3D 模型中；
一组树木可以存储为实例化 3D 模型；
当这些元素出现在相同的地理位置时，将这些模型组合在单个 复合瓦片中会很有用：可以通过单个请求将给定地理位置的可渲染内容作为单个瓦片获取。



在单个复合瓦片中的子瓦片被称为内部瓦片。复合瓦片也可以继续嵌套，这意味着每个内部瓦片都可以再次成为复合瓦片。
例如 批处理 3D 模型、实例化 3D 模型 等等都可以继续组合。

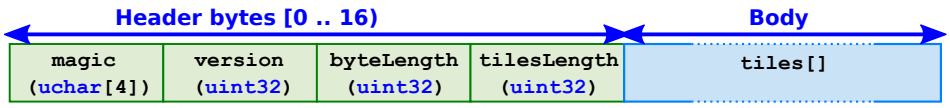
复合瓦片作为瓦片的一种格式，其头部信息以 魔法直接(magic bytes)作为开头，后面跟着版本号、以字节为单位的瓦片数据总长度等。

头部信息的公共部分后跟着一个整数，指示在复合瓦片中组合的瓦片数量。

符合瓦片的主体仅由一组内部瓦片(子瓦片)组成。每个内部瓦片都是一个表示瓦片的二进制 blob 数据：它以通用头部信息开发，指示内部瓦片的数据格式、版本和长度，然后是相应的瓦片格式的主体。

复合瓦片：数据结构

下图显示了在复合瓦片的头部和主体中，有关数据结构和类型的示意：



11. 扩展

3D Tiles 提供了一种机制来扩展具有新功能的基本规范：3D Tiles 中每一个 JSON 对象都可能包含一个 扩展(extensions)字典。这个字典的键名是扩展的名称，键值是扩展的对象值。数据供应商(提供者)可以设定扩展字段、结构和语义规范。

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

左侧示例显示了数据供应商(提供者)在 JSON 瓦片中的上下文中添加了 瓦片碰撞体的自定义字段。
扩展名为 VENDOR_collision_volume，通过其重心和半径定义了一个边界包围球，类似于 3D 瓦片中已支持的边界包围球。

在瓦片集或其子项之中使用的扩展名必须列在顶级 extensionsUsed 字典中。当需要某个扩展来正确加载和渲染瓦片集时，它也必须列在 extensionsRequired 字典中。这样可以实现在加载瓦片集时检查这些字典，同时检查它们是否支持瓦片集使用或需要的扩展。

12. 样式声明

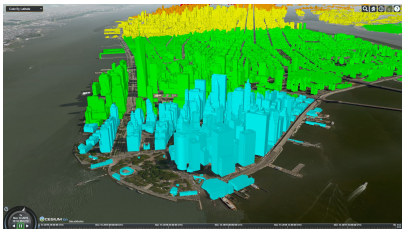
根据瓦片格式，瓦片可渲染内容可能包含不同的功能。对于批处理 3D 模型，使用 batchId 用于标识不同模型特征。对于实例化 3D 模型，每个实例都是一个特征。对于点云有两种选择：当点被批处理时，具有相同 batchId 的每组点都是一个特征，否则另外一种情况就是每个点都是一个特征。

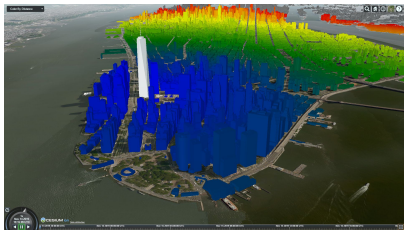
3D Tiles 允许在运行时通过样式声明来修改特征的外观：使用包含一组表达式的 JSON 定义样式。
这些表达式的值决定了要素的可见性或颜色。运行时引擎可以评估这些表达式并将样式应用到基于用户交互和存储的批处理表的某些功能属性中。

例如，在每个建筑物都是一个特征的批处理 3D 模型中，建筑物的颜色可能会在运行时根据不同的标准进行修改。以下是 JSON 样式的示例，该示例会根据建筑物的高度将其渲染成不同的颜色：

```
{
  "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)"]
    ]
}
```

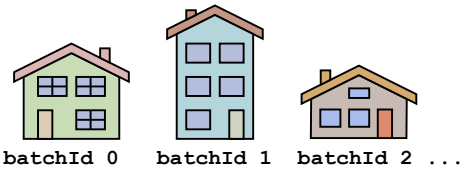
样式声明的其他示例：
可能会根据其他因素将建筑物设置为不同颜色，例如纬度(左) 或 与某个地表的距离(右)





设置批处理 3D 模型的样式

在此示例中，批处理 3D 模型包含多个建筑物。每个建筑物都有自己的 batchId。

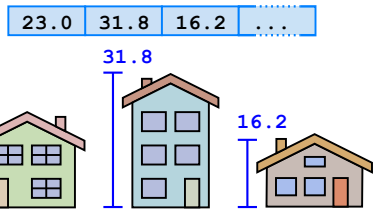


该模型还有一个批处理表，批处理表的 JSON 头部包含 高度(height)属性。它指向批处理表的二进制主体的一个数据片段。该数据片段表示建筑物的高度数组。该 batchId 被用于查找建筑物的高度。

Batch table JSON:

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

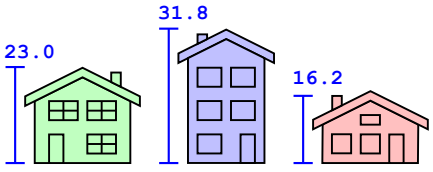
Batch table binary:



渲染模型时，样式 JSON 将应用于模型。样式决定了每个特征(建筑物)的颜色。在本示例中，颜色取决于 高度(height)的属性值。

```
{
 "color" : {
  "conditions" : [
    ["(${height} < 20)", "color('#FFC0C0')"],
    ["(${height} < 30)", "color('#C0FFC0')"],
    ["(${height} < 40)", "color('#C0C0FF')"]
  ]
 }
}
```

当建筑物高度低于 20、30、40 时会被渲染成红色、绿色 和 蓝色。



JSON 样式

传递给运行时引擎的 JSON 样式包决定瓦片外观的属性：

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

对于批处理 3D 模型和实例化 3D 模型，样式中 show 用于决定其是否可见，样式中 color 决定呈现的颜色。对于点云，样式还可能包含一个 pointSize 属性，用于确定点在渲染时的大小。

样式属性的值使用 JavaScript 子集的表达式语言编写的。这种语言的类型还包括对几何计算和表示颜色的向量类型。这些向量类型源自 GLSL，因此可以在着色器中轻松、高效地实现可以用这些类型表示的表达式。

(译者注：这种 JavaScript 子集的表达式语言在下文中被称为 样式语言)

表达式

样式语言支持 JavaScript 支持的大多数一元运算符、二元运算符、三元条件运算符，以及常见的内置数学函数。例如 abs、max 或 cos。

样式语言包含向量类型 vec2、vec3 和 vec4。其中 vec4 类型可用于表示 RGBA 颜色。

向量的分量 vec2、vec3、vec4 可以使用点表示访问类型。例如 4D 向量 v 可以解释为 x y z 和 w 坐标，并且可以使用 vx、vy、vz、vw 访问。同样等效地可以被解释为 RGBA 的四个颜色分量，即 vr、vg、vb、va 访问。

或者可以使用数组符号获取分量，例如 4D 向量 v 可以使用 v[0]、v[1]、v[2]、v[3] 访问对应分量。

颜色使用 vec4 类型，包含 RGBA 的颜色分量，这样可以比较方便使用函数创建不同颜色值。例如一些函数可以使用类似 CSS 中定义的颜色方式来定义颜色。
例如：

- `color('red')` 颜色单词
- `color('blue', 0.5)` 颜色单词 + 透明度
- `color('#00FFFF')` RGB颜色值
- `rgb(100, 255, 90)` RGB颜色值，每个数值取值范围 [0, 255]
- `hsl(1.0, 0.6, 0.7)` HSL 色相、饱和度和亮度，每个数值取值范围 [0, 1]

样式语言中定义了几个内置函数可以用于数字以及某些向量类型：
clamp：可以将数字或向量分量限制在某个范围内

mix：允许数字或向量之间进行线性插值
length、**distance**、**normalize**、**dot**：分别对应求向量长度、距离、归一化、点乘(内积)
cross：用于计算三维向量的叉乘(外积)
样式表达式可以解析为抽象语法树(AST)并进行评估。计算表达式的类型必须与样式属性的类型匹配。

变量

样式表达式中也可以包含变量。这些变量指的是瓦片中要素的某些属性。这些属性的实际值存储在批处理表中。
变量使用 ECMAScript 2015 模板字面量语法编写：字面量 **${property}** 中 **property** 是区分大小写的属性名称。

以下是根据存储的在批处理表中的高度为要素，分配颜色的样式示例：高度大于 50 将被着色为红色，否则将被着色为白色。

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

条件

样式属性也可以使用条件来编写。条件有两个表达式组成：布尔值、一个计算结果为布尔值的表达式。多条件可以组合为一个数组。

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

左侧为使用条件的示例：
如果与特征关联的温度大于 100 则返回红色。如果大于 50 则返回黄色，否则将返回绿色。条件按顺序进行判断执行。如果不满足任何条件，则样式属性将是未定义的(undefined)。

定义变量

除了引用批处理表中属性作为变量外，样式还可以定义自己的变量。这些自定义变量被定义在 defines 字段中。
每个定义都包含映射到表达式的新变量的名称。
表达式可能不引用其他定义，但可能引用批处理表中的现有属性作为变量。

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

自定义变量 distanceToPoint 表示某个点距离点 vec2(1,2) 的距离，而这个点由存储在批处理表中的两个变量 ${x} 和 ${y} 构成。

自定义变量 color 用于确定颜色：当距离超过阈值时，物体会被渲染成红色，否则通过距离(${distanceToPoint})来设置颜色(0x0000FFF)的透明度，并将该颜色作为渲染物体的颜色。

正则表达式

样式语法支持正则表达式，因此可以根据特征表中存储为字符串的属性值制定渲染条件。

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

示例：具有 ${name} 字段且进行给定正则表达式测试，将计算的布尔值结果作为属性 show 的值。左侧示例中正则表达式为：允许 Building 后面跟一个空格或数字。

点云的样式语法

当样式语法应用于点云时，样式还可以参考存储在特征表中的语义，例如 位置、颜色和法线。
点的位置可以使用 ${POSITION}。当位置以量化形式存储时，指在应用量化缩放之后、添加量化偏移之前的位置。
相反，${POSITION_ABSOLUTE} 是指应用量化缩放和偏移后的位置。
可以使用 ${COLOR} 作为颜色关键词。
点的法线使用 ${NORMAL} 关键词。如果法线存储在八进制编码(oct-encoded)表中，那么这指的是解码后的法线。

关于量化位置、八进制编码(oct-encoded)法线的更多信息，可以在第 13 章中找到。

13. 通用定义

13.1. glTF——GL传输格式

3D Tiles 中批处理 3D 模型和实例化 3D 模型使用的是二进制 glTF 模型。这是 GL 传输格式的二进制形式，它是一种用于高效传输 3D 内容的开放规范，由 Khronos Group 维护。

https://github.com/KhronosGroup/glTF

13.2. 世界地理空间系统标准椭球(WGS84)

应用于地理空间的世界标准椭球。该椭球空间位置以纬度和经度的形式给出，以弧度为单位，在 EPSG 4979 中定义的 WGS84 标准椭球。

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

最小和最大的高度区域为 WGS84 椭球的上方或下方，单位为米。

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

13.3. 相对中心位置(RTC_CENTER)(`RTC_CENTER`)

(此处提供的信息基于 http://help.agi.com/AGIComponents/html/BlogPrecisionsPrecisions.htm)

创建 3D Tiles 是为了是大量 3D 内容交互式可视化成为可能，尤其是地理空间数据。这意味着必须能够以高精度渲染 3D 对象，即使他们与虚拟世界的中心距离很远。

在 OpenGl、Direct3D 或 Vulkan 等常见图形 API 中，3D 对象的顶点位置通常存储为单精度(32位)浮点数。

由于精度优先，无法准确表示到渲染坐标系原点距离较大的对象：不同的顶点坐标将具有相同的内部表示为单精度值

理论值	实际 32位 单精度值
131072.01	131072.0156250
131072.02	131072.0156250
131072.03	131072.0312500

尽管 131072.01 和 131072.02 值不同，但它们对应的 32位单精度值却是相同的，都是 131072.0156250。

这种准确性的缺乏导致渲染有伪影——最明显的是靠近渲染对象并进行缩放时会出现视觉抖动。为了环节这个问题，3D Tiles 支持一种用于补偿由大坐标值引起的误差技术。这种技术称为 相对中心(RTC)渲染。

批量 3D 模型的顶点位置为 glTF 模型中的一个属性。对于实例化 3D 模型和点云，特征表包含实例和点的位置属性。这些位置的坐标值通常存储为单精度、32位浮点数。为了支持 RTC 渲染，瓦片格式允许在其特征表中指定 RTC_CENTER 属性。此属性特定用于应用程序的坐标系的中心，如果存在该属性则假定所欲位置都相对于该中心。

为了考虑顶点的相对位置，RTC_CENTER 用于修改渲染 模型 - 视图 - 矩阵(Model-View-Matrix)。最初这是一个矩阵 MV_{GPU}，双精度存储在 CPU 中。将 RTC_CENETER 使用原始模型-试图矩阵转换为眼睛坐标：

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

用于渲染的 模型-视图-矩阵则是矩阵 MV_{GPU} ，以单精度存储，通过用生成的 RTC_CENTER 替换原始 模型-视图-矩阵的最后一列来创建 RTC_CENTER_{Eye}。

使用这种技术，可以避免在模型的位置出现大值，由于后续渲染管线的精度有限，导致渲染伪影：相对于位置可以给出较小的值 RTC_CENTER，并且修改后的 模型-视图-矩阵 将他们正确的转换为眼睛坐标系进行渲染。

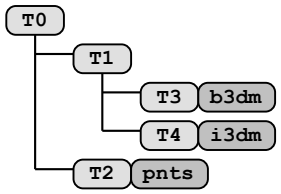
13.4. 3D Tiles 中的变换

3D Tiles 中可渲染的内容可以处在不同的坐标系中。例如一个城市的瓦片集可以包含单个建筑物的嵌套瓦片集，后者可以拥有它自己的坐标系。

为了在局部坐标系之间进行转换，每个瓦片都有一个可选的 变换(transform) 属性。该属性是一个列阵式 4x4 仿射变换矩阵，将 瓦片 的坐标系变换为父级坐标系，未定义时默认为单位矩阵。

瓦片的变换矩阵会影响瓦片和它的内容的位置、发现和边界包围体。特征的位置(例如实例化 3D 模型中的实例，或 点云中的点) 与变换矩阵相乘，将他们从本地坐标系带入父瓦片的坐标系。法线乘以左上角 3x3 矩阵逆转置的 3 维矩阵，以适当地考虑非均匀缩放。

除了明确定义为 EPSG:4979 坐标中的 “区域” 边界包围体外，边界包围体将使用矩阵进行变换。



T0

T0 · T1

T0 · T1 · T3

T0 · T1 · T4

T0 · T2

作图为瓦片集 T0 ... T4 和不同内容类型的瓦片集示例。
该图显示了瓦片的层次结构。右侧显示了为每个瓦片计算的变换：每个瓦片的变换计算为从根瓦片到相应瓦片的路径上所有变换的乘积。

如果瓦片的内容使用相对中心位置，则 RTC_CENTER 必须将视为顶点的附加转换。

13.4.1 3D Tiles 和 glTF 的坐标系

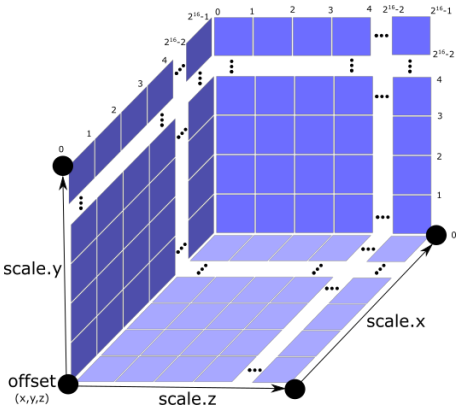
3D Tiles 中的坐标系将 z 轴定义为局部笛卡尔坐标系(即 z 轴为朝上)。相比之下，glTF 将 y 轴视为向上。当 glTF 资源嵌入到批处理 3D 模型和实例化 3D模型中时，必须通过在运行时转换 glTF 资源来考虑这些不同的约定，以便 z 轴指向上。此外，每个 glTF 资源都有自己的带有变换的节点层次结构。有关这些不同变换如何一起发挥作用的详细信息，以及他们应用于可渲染的内容的顺序，可以在 3D Tiles 规范中找到。

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

13.5. 量化位置和 Oct 编码法线

实例化 3D 模型瓦片和点云瓦片格式中的位置可以有不同的设置形式：
当使用 POSITION 属性时它们是存储为 3 个 32 位浮点数，包含实例或点的 x y z 坐标分量。

对于大量实例或点，3D Tiles 提供了一种更紧凑的存储方式：
当使用 POSITION_QUANTIZED 设置位置时，它们是 3 个 16 位无符号整数。
QUANTIZED_VOLUME_OFFSET 和 QUANTIZED_VOLUME_SCALE 存储在特征表中。



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

同样，点和实例的发现可以作为由 3 个浮点数组成的向量构成，即在 NORMAL、NORMAL_UP 和 NORMAL_RIGHT 属性中。对于更多的实例和点，这些法线可以以压缩的形式存储，由 _OCT16P 这些属性名称作为后缀。
这种压缩形式由双向映射组成：法向量从单位球体的八分圆映射到八面体的面，然后投影到平面并展开为单位正方形。
使用这种压缩方式可以仅使用单个 16位 值去表示 3D 法线向量。

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