# 城市三维白模场景构建技术和响应式渲染策略研究及实现

## 研究背景与意义

* **背景**：2024年是中国城市数字化、智能化发展的重要战略规划期。城市信息模型（CIM）平台作为智慧城市建设的核心，需要高效、低成本的城市三维场景建模技术来支持城市管理和决策。传统的城市建模方法（如倾斜摄影、激光扫描）成本高、模型资源占用大，且难以满足多层级业务场景的需求。
* **目标**：提出一种城市三维白模的建模方法，设计层级场景结构及渲染优化策略，开发适用于CIM的三维场景建模软件。

## 相关技术理论

### 父模型与子模型之间的关系具备如下特点

1. 层级结构：父亲模型可以包含多个子模型，形成树状结构。
2. 继承与传递：子模型继承父亲模型的属性，对父亲模型的操作可以传递给所有子模型。
3. 独立性与协同性
4. 交互逻辑

### 三维场景优化渲染

#### Level of Detail

1. 离散LOD模型：多个不同质量的副本
2. 连续LOD模型：实时计算视角与物体距离，进行分辨率调整——需要优化顶点计算
3. 多分辨率LOD模型：单个物体不同部分的分辨率和视角有关系——大体积模型/大范围场景

#### 动态加载技术

1. 三维模型预分块：低细节层次模型做为背景，高细节层次模型用以提高体验感，b\*b宫格高细节模型实时动态加载。随着视点移动，计算高模覆盖面积进行加载。
2. 大地形分块加载：地形起伏不大时，切割为均匀的小地形块，根据摄像机位置加载该块附近的地形块，与旧的内容进行对比，加载新的，卸载旧的。

#### 遮挡剔除算法

1. 视锥体裁剪：只有在视锥体内部的物体才会被渲染——复杂场景或者大量遮挡时效率会降低
2. 基于空间分割（Octree、BSP树)：空间划分为多个空间单元，通过递归遍历空间分割树，确定可见物体，加载到渲染列表中——适用于大场景，但消耗更多计算资源

#### OBJ\MTL\Three.js

此处语法、格式不再赘述

## 城市三维场景模型组织结构

### 城市场景核心对象

1. 实体对象：城市中的基础元素，包括但是不限于建筑物、道路、绿地和各类城市设施
2. 聚合对象：城市聚合对象则是由多个实体对象组成的集合体 。

### 城市三维场景白模

1. 低精度模型：重点在几何形状，最小化三角面。
2. 参数化建模
3. 低成本
4. 形状还原

#### 实体三维白模

城市实体对象的三维立体结构，基础元素，注重形状和位置，不关注材质和纹理细节

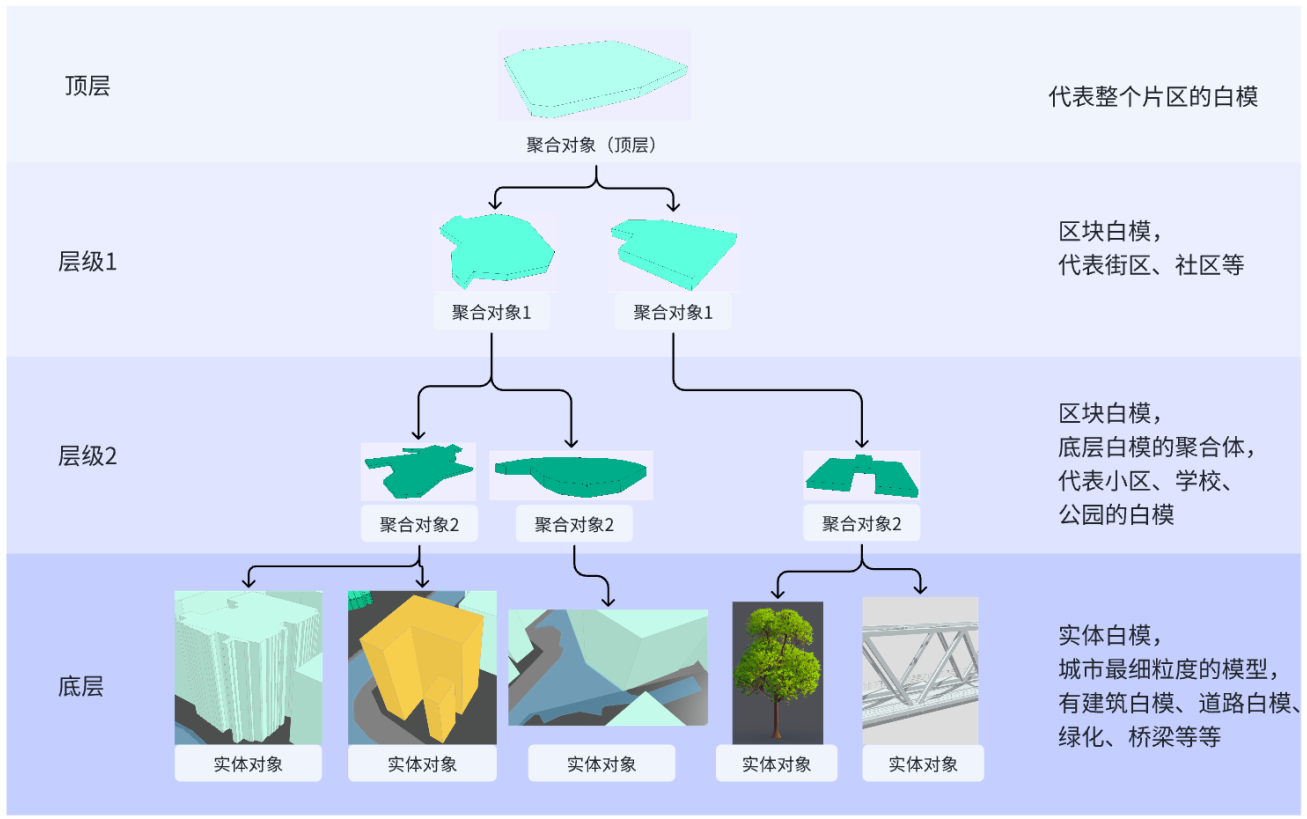
#### 区块三维白模

城市聚合对象的三维立体结构， 抽象地表示城市中的较大单元，注重于父子包含的关系，因为区块三维模型能够替代更细致的三维白模， 与被替代的三维白模之间形成了父子层级关系。

### 层级树

**自底向上**的构建方式： 片区层级树的结构是自底向上的，其构建过程从实体对象开始，逐步向上聚集成更大的聚合对象。

父节点代表了子节点在地理空间中的外边界，即父节点的几何属性是其所有子节点几何体合集的外包络 。



## 建模算法

### Hat概念

**“hat”**类型的城市对象，是对无父节点对象的完全复制体，除了id标识符，几何体轮廓、地理位置、三维白模等属性完全一致，如同一个帽子戴在对象的头上。该对象设计的目的是避免断层和跨层父子的问题， 无论是层级还是对象的插入或删除，都能保证层级树的完整性。

### 参数化建模

1. 必须：外轮廓点集合、外部高度
2. 可选：内轮廓点集、内部凹凸高度、材质和着色信息

### 三角剖分算法

**出于兼容性考虑**：尽管现代的三维引擎已经支持直接编写由三个以上顶点构成的面，无需手动将其切割为三角形面片。但在某些情况下，特别是在处理OBJ文件格式时，仍然需要手动将多边形面切割为三角形面。这是因为一些渲染引擎，如three.js，会根据特定的规则来生成三角形，导致出现错误的面。

#### 简单剖分

loop:

选取一个点+2个相邻点=》三角形

if 三角形在图形内部

删除这个点

直到点集合数量小于等于3

#### 简单剖分+优化

##### 凸多边形过滤

在three.js等三维引擎中，如果面是凸多边形，按照引擎的默认方法生成三角形时不会出现错误面 。在剖分了一定数量的三角形后，剩余部分为凸多边形的凹多边形 ，则没有必要继续剖分。

##### 点密集程度

针对的是那些形状类似凸多边形但存在凸起部分的图形 ，先处理掉这些凸起的轮廓控制点，就能将原图形转变为凸多边形，从而减少三角剖分的数量 。

使用KDE（Kernel Density Estimation ），每个数据点放置高斯核函数及逆行叠加，形成平滑密度估计，三角剖分算法可以从密度最高的点开始剖分 ——即从凸起部分开始剖分。



#### Delaunay算法

##### 特性

将点集组织成无重叠的三角形网络

* 凸性：生成的三角形网络是凸的，即任意三角形的外接圆内不包含其他点。
* 最小角度：在所有可能的三角剖分中， Delaunay剖分尽可能地避免了“瘦长”的三角形，从而使得网络更加均匀。
* 唯一性：不论从区域何处开始构建，最终都将能得到一致的结果。
* 区域性：新增、删除、移动某一个点只会影响临近的三角形。

##### 流程

1. 输入点集P
2. 构建super三角形\*2，包含P在内部——矩形包围盒
3. Super三角形加入三角形-set中
4. 对于P中每一个点p
   1. 绘制三角形-set的外接圆
   2. p所属外接圆多于1
      1. 删除对应外接圆的内三角形的公共边
      2. p与删除公共边后形成的空腔轮廓点连接
      3. 将新的三角形加入三角形set中
5. 最后删除超级三角形以及相关的三角形

#### 三种方法对比

1. 性能接近
2. 优化后简单三角剖分-面数更少
3. Delaunay算法处理复杂轮廓结果稳定

### 交互式轮廓顺序点集捕获算法

#### 算法流程

1. 建模人员通过鼠标点击输入图中轮廓一个内部点的二维坐标。
2. 算法以该点做**四领域洪泛**填充，填充完毕后将结果进行**灰度处理**。
3. 对图像做**膨胀与腐蚀**操作，使得轮廓边缘连续且平滑。
4. 对处理后的轮廓使用**二值图像轮廓提取**，得到轮廓的顺序点集
5. 对轮廓做**多边形逼近**， 减少轮廓顺序点集的数量。

#### 多边形逼近算法——Douglas-Peucker算法

设定一个阈值，对一条曲线的首尾两点做连接得到一条直线，找到曲线中距离该直线最远的点，若该点到直线的距离大于阈值，则以该点与首点、尾点构成两条直线，重复上述操作，直到曲线中所有的点都在阈值内。

### 片区层级树构建与变动

#### 增加层级

一开始会将子层copy一份做为hat填充

#### 删除层级

1. 执行上下级包含关系计算，如果有包含关系就直接连接
2. 如果没有则加一层hat，做为父亲节点

#### 增加对象

首先寻找父亲，没有则添加hat

#### 删除对象

删除后使用hat替代消失部分

### 交互式聚合对象建模

#### 算法流程

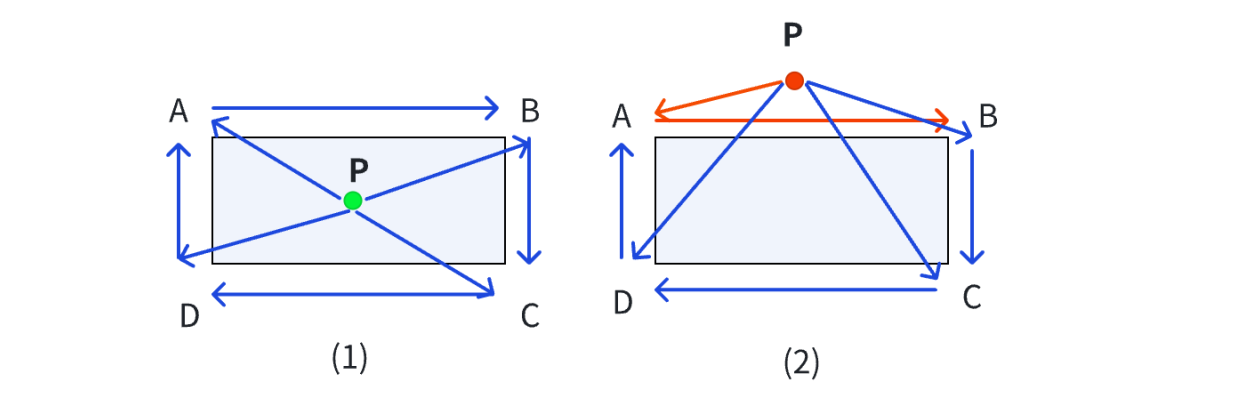
1. 创建新层级，或进入一个已创建完毕的层级。
2. 操作平台输入该层级下一层级的对象轮廓结果，可以是实体对象或聚合对象。
3. 建模人员在图中输入轮廓点，至少三个确保能够形成一个面。
4. 平台计算该面是否在二维上包含下层对象轮廓，如果包含，则构建父子关系。
5. 修改轮廓点，再次检测包含关系。
6. 得到轮廓点集，保存至该层数据中。

#### 检查某点是否位于图形内

根据叉乘数值的正负判断

例如：

（P→B）×（B→C）、（P→C）× （C→D）、（P→D）×（D→A）



### 层级树合并

由于不同片区可能具有不同数量的层级，场景树的高度将以层级数量最多的片区为准。其他片区的层级树将被嵌入到这个最高层级的结构中，最高层级的实体作为其他片区层级树的顶层实体。 用于保持层级一致的实体被标记为“hat”类型，以确保整个场景树的层级数量一致，保持平衡和一致性 。

## 渲染优化策略

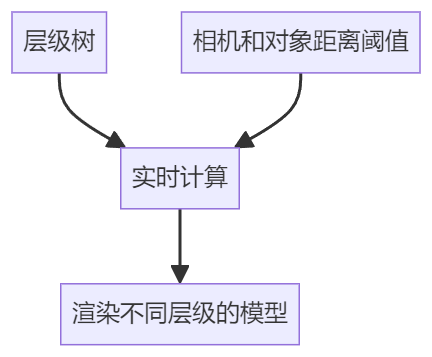
### 响应式动态渲染策略

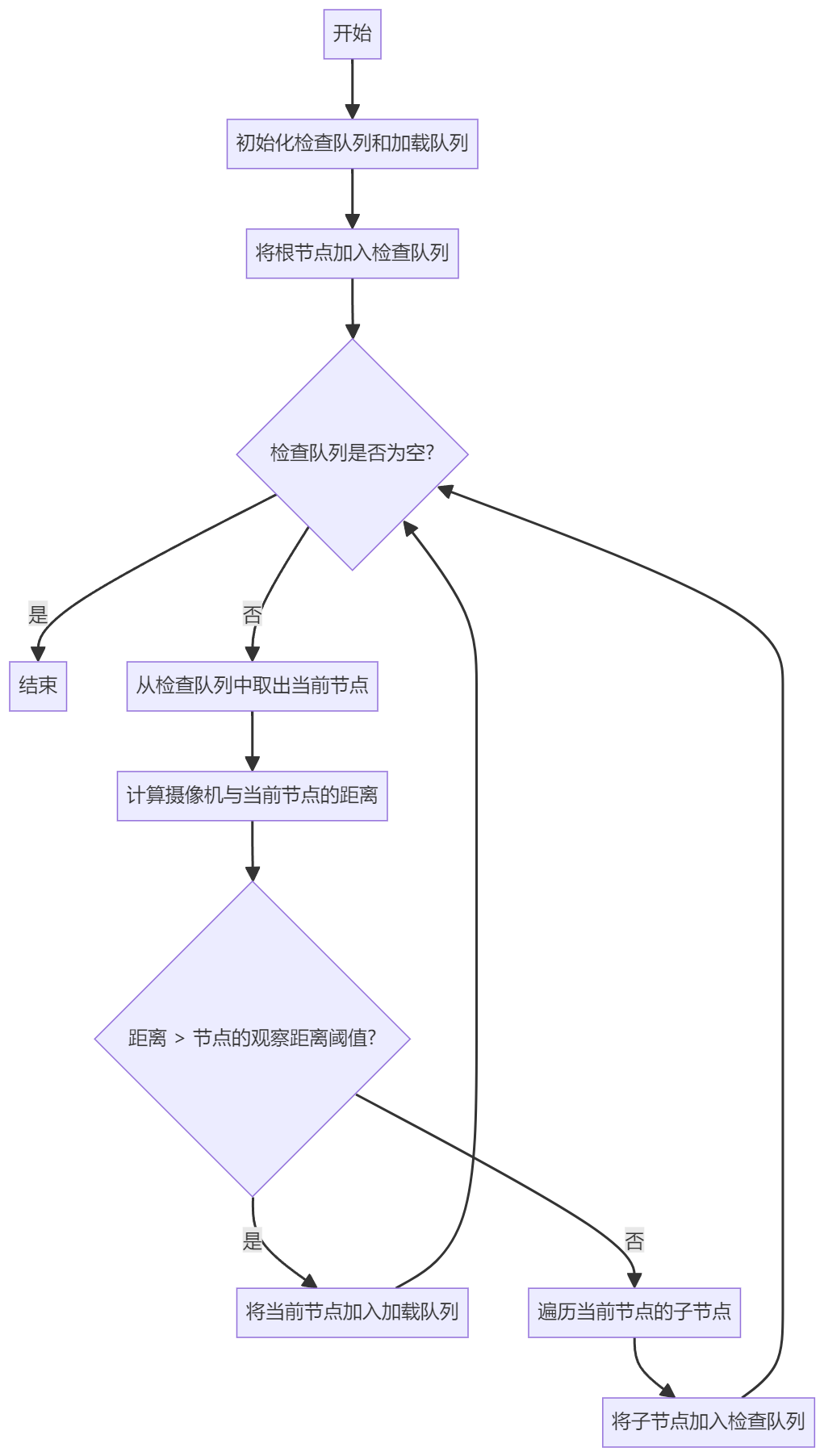
#### 前提

模型已经是最简三维白模型，无高模或者进一步粗糙化的可能。

#### 算法

结合层级数和相机和对象距离：





#### 阈值确定

阈值与对象体积挂钩——》外接球的半径乘以一定比率(2.5或者3为佳)设为阈值

### 延迟加载策略

#### 前提

1. 用户操作幅度太大，场景多次渲染，不断进行不同层级的模型交换，导致卡顿和性能压力。
2. 操作速度快，事业模糊，无需模型渲染

#### 解决

1. 使用画面比例（不同层级模型对同一个场景单位速度数值敏感度不同）变化速度做为依据设定阈值，高于这个速度则停止渲染
2. 每次缩放需要重新计算阈值即标准速度