Web 端图形引擎架构设计与三维场景展示性能提升研究

黄昱嘉

四川电力设计咨询有限责任公司 四川 成都 610041

【摘　要】：Web 端三维场景展示受浏览器线程、GPU 接口、网络加载和终端算力共同约束，图形引擎架构决定资源能否按层入队、渲染任务能否按帧预算执行、画质能否随运行状态调节。本文分析 Web 端图形引擎在资源模块化、浏览器渲染链路、跨终端承载、接口演进和大规模场景瓶颈中的架构要求，设计资源层、引擎层和展示层协同的性能控制链，并从管线重构、空间分层、压缩缓存和帧率监测四个环节提出三维场景展示性能控制方法。分层架构可把帧耗、对象规模、GPU 内存和 LOD 距离纳入统一判断，减少单点优化失效。

【关键词】：Web 端图形引擎；三维场景展示；渲染管线；资源调度；性能提升

DOI:10.12417/2811-0528.26.20.010

引言

Web 端三维展示已进入城市数字孪生、工业运维、在线展陈和工程模型浏览等场景，浏览器受到主线程任务、GPU 命令提交、资源下载和移动端内存限制。WebGL 兼容成熟，WebGPU 提高现代 GPU 访问上限，但接口升级不能替代引擎层的资源组织、状态缓存和质量调节。复杂场景在首屏、漫游、拾取和告警定位之间切换，若模型解析、纹理上传和绘制提交缺少统一调度，帧率下降会表现为卡顿、空白或画质突变。围绕架构设计讨论性能控制，可把三维展示从经验调参转为可定位、可分级的工程链条。

1 Web 端图形引擎架构设计的性能意义

1.1 支撑三维场景资源模块化与分层呈现

三维场景资源由网格、纹理、材质、动画、节点层级和业务属性共同组成，同一页面可能承载建筑构件、设备模型、地形底图和动态标注。Web 端图形引擎若只把对象作为静态文件加载，纹理尺寸、材质数量和节点命名难以统一。资源层独立后，可在导入阶段完成网格合并、纹理图集、材质复用和属性索引，使渲染层面对可分批、可裁剪、可缓存的标准对象[1]。地形、建筑外壳、设备内部件和实时数据标牌具有不同可见距离与刷新频率，资源层按空间层级、语义类别和精度等级建立元数据后，展示层可依据视点距离和交互状态调用对应层级。对象包围盒、材质引用、纹理尺寸和业务类别被统一保存后，模型来源差异被压缩在资源层，展示层面对的是稳定对象集合。

1.2 推动浏览器渲染链路实时响应

浏览器三维渲染链路包含脚本更新、场景遍历、状态排序、GPU 命令提交和画面合成，任何同步解析、频繁状态切换或主线程长任务都会占用单帧时间。架构设计把场景更新、资源上传和绘制提交拆成可监测阶段，避免模型加载、材质创建和交互计算在同一帧集中发生。实时响应还要求渲染循环与浏览器调度协同，动画和交互围绕 requestAnimationFrame 组织，异步加载、解码和缓存写入避开关键帧路径[2]。图形引擎预留任务队列、优先级和取消机制后，可让视角变化、拾取反馈和热点更新优先于低优先级资源补全。单帧加权耗时把脚本更新、场景遍历、命令提交、GPU 渲染和合成等待纳入同一阈值判断，用于区分主线程拥塞和 GPU 侧填充压力。单帧加权耗时评价见式（1）。

$$T_f = \sqrt{w_{js}T_{js}^2 + w_sT_s^2 + w_dT_d^2 + w_gT_g^2 + w_cT_c^2} \quad (1)$$

式（1）中，Tf 为单帧加权耗时，单位为毫秒；Tjs 为脚本更新耗时，单位为毫秒；Ts 为场景遍历与状态排序耗时，单位为毫秒；Td 为绘制命令提交耗时，单位为毫秒；Tg 为 GPU 渲染耗时，单位为毫秒；Tc 为合成与等待耗时，单位为毫秒；wjs、ws、wd、wg、wc 为各阶段权重，单位为无量纲。该关系把脚本、场景、提交、GPU 和合成等待纳入同一加权耗时，可识别超过目标帧预算的主导阶段，并把 Tf 用于浏览器渲染链路的阈值判断。

1.3 形成跨终端适配与统一展示承载

Web 端三维展示面对桌面浏览器、移动浏览器、内嵌 WebView 和大屏终端等环境，GPU 能力、纹理上限、内存余量和网络带宽差异明显。统一承载并不意味着所有终端使用同一画质，而是同一图形引擎能够识别设备能力并选择合适后端、精度和资源等级。架构设计应把能力探测、特性开关和降级策略放在初始化阶段，使 WebGL、WebGL2 或 WebGPU 后端可在统一场景描述下工作。工程设备模型在桌面端可显示完整管线和动态传感标牌，在移动端保留主要构件和关键交互热点；两端共享对象 ID、空间坐标和业务属性，能降低多端版

本分裂成本。

2 Web 端图形引擎架构设计基础与性能瓶颈

2.1 引擎分层架构的基础组成

Web 端图形引擎通常由资源层、场景层、渲染层、交互层和监测层共同工作。资源层负责模型解析、纹理压缩、材质合并和缓存索引；场景层维护节点层级、空间包围盒、可见性状态和业务属性；渲染层把可见对象转换为绘制批次、着色器参数和 GPU 命令；交互层处理拾取、漫游、测量和标注；监测层记录帧耗、draw call 数量、显存占用和加载队列长度[3]。分层边界清晰时，性能问题可以定位到具体环节。资源工程师围绕 glTF、纹理格式和压缩参数调优输入对象，渲染开发人员调整批处理、着色器和状态缓存，业务开发人员通过统一 API 绑定传感数据或交互事件，系统仍能保留可测量指标。

2.2 浏览器图形接口能力演进

WebGL 以 OpenGL ES 风格向浏览器暴露图形能力，成熟生态使 Three.js、Babylon.js 等框架能够快速构建三维场景。其优势在于兼容性强、运行边界清晰，但状态机式接口容易在大量材质、纹理和几何切换时产生额外开销。WebGPU 面向现代 GPU 接口设计，支持更明确的管线对象、绑定组和计算能力，在复杂渲染和并行计算场景中具备更高上限[4]。实际架构中，WebGL 与 WebGPU 更适合形成双后端适配：旧设备承担基础展示，新终端处理实例化绘制、后处理和可见性计算。引擎抽象层保持场景对象、材质参数和渲染任务的后端无关描述，性能差异才可归因到管线实现，而不是数据口径变化。

2.3 场景资源加载状态约束

三维场景首屏性能受资源加载状态制约。大体量模型经过网络下载、解压、解析、纹理上传和 GPU 缓冲创建，多个步骤跨越网络线程、主线程和 GPU 资源管理。若引擎缺少状态约束，用户视角进入目标区域时，关键模型可能仍处于解析或上传阶段；若一次性加载全量资源，又会造成首屏等待过长和内存峰值过高[5]。资源状态应被显式建模为排队、下载、解码、待上传、可渲染和可回收等阶段。弱网络重复请求、移动端后台切换、大纹理集中上传都会造成异常，引擎应为资源设置优先级、超时、重试和回收策略，并把状态变化暴露给展示层。

2.4 大规模三维展示的性能瓶颈

大规模三维展示的瓶颈不是单一指标造成的。拾取任务以 4000 个可见对象和 45m 的 LOD 切换距离保护近景精度，远景浏览允许 18000 个可见对象和 180m 切换距离；4000 个低于 18000 个，45m 低于 180m，说明近景拾取和远景概览面对不同可见范围。首屏概览目标帧耗为 20ms，远景浏览为 24ms，漫游交互与构件拾取均按 16.67ms 控制；16.67ms 低于 20ms，20ms 低于 24ms，说明首屏等待、连续交互和远景概览对应不同帧预算。GPU 内存预警值从 640MB 调整为 896MB 分档，设备巡检和首屏概览均为 768MB，告警定位为 704MB，构件拾取和漫游交互为 640MB；这些阈值把显存压力、对象数量和可见范围同时纳入瓶颈判断。几何顶点过高会增加顶点处理和内存传输压力，材质和纹理数量过多会打断绘制批次，透明对象排序会增加场景遍历成本，实时标注和拾取又会占用主线程计算时间；引擎先识别瓶颈发生在资源、渲染、交互还是展示策略环节，再选择压缩、合批、裁剪或降级动作。六类任务参数组合见表 1。

表 1 Web 端三维场景性能控制参数

展示任务	目标帧耗(ms)	可见对象上限(个)	纹理等级	GPU 内存预警(MB)	LOD 切换距离(m)
首屏概览	20	12000	中	768	120
漫游交互	16.67	8000	中低	640	90
设备巡检	18	6000	高	768	60
构件拾取	16.67	4000	高	640	45
远景浏览	24	18000	低	896	180
告警聚焦	18	5000	高	704	50

表 1 中，首屏、漫游、巡检、拾取、远景和告警的目标帧耗、对象上限、GPU 内存预警和 LOD 距离并不相同；远景浏览对象上限最高而帧耗预算最宽，构件拾取对象上限最低且帧耗预算最紧，说明性能控制应按展示任务分档。

3 三维场景展示性能提升的架构优化措施

3.1 基于渲染管线重构降低帧耗

渲染管线重构首先要减少无效绘制。场景遍历阶段依据包围盒、视锥和遮挡关系筛选可见对象，再按材质、纹理和着色器状态排序，使相同状态的对象尽量合批提交。大量重复构件可采用实例化绘制或几何合并降低 draw call 数量，尺寸小但数量多的标注和图标可使用纹理图集减少纹理切换。管线重构的目标不是盲目合并所有对象，而是在可拾取、可更新和可维护之间保留必要粒度。管线重构后，目标帧耗不再作为单一全局常量，而是按展示任务分档；同一引擎在漫游和拾取中优先压低帧耗，在远景浏览中保留更大可见范围，以免一种预算同

时约束所有场景。任务帧耗预算折线见图 1。

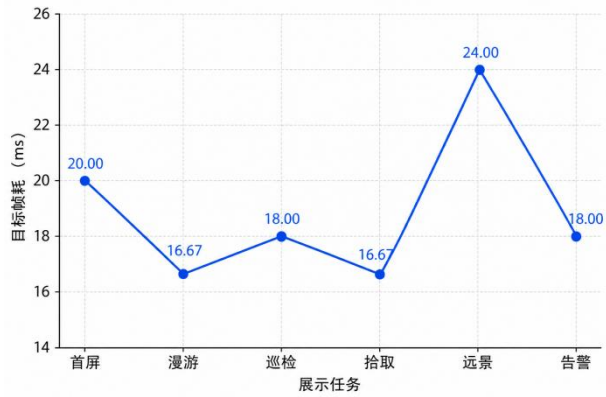

图 1 不同展示任务的目标帧耗设计关系

图 1 中，漫游交互和构件拾取的目标帧耗为 16.67ms，远景浏览放宽至 24ms，首屏概览为 20ms，设备巡检与告警定位均为 18ms，说明交互密集任务优先控制响应，远景任务可用较高帧耗换取更大可见范围。

3.2 以空间分层重塑加载顺序与可见范围

空间分层调度把三维场景划分为视点附近、当前视域、邻近预取和远景概览等层级，加载顺序随视角移动和交互任务动态变化。视点附近对象承担拾取和细节查看，当前视域对象承担主要画面，邻近区域保留低精度预取，远景区域以轮廓、瓦片或简化网格维持整体空间感。可见范围控制应与相机参数、对象尺寸和业务优先级共同决定；普通 LOD 只按距离切换，容易在重要小构件接近视点边缘时过早降级。Ld 作为 LOD 切换距离输出，可把对象屏幕面积和相机移动速度同时转化为可见范围边界，使加载、解码和 GPU 上传分散到多个帧周期。LOD 切换距离修正关系见式（2）。

$$L_d = L_0\sqrt{\frac{S_{ref}}{S_o}}\exp\left(-\gamma\frac{v}{v_{ref}}\right) \qquad （2）$$

式（2）中，Ld 为 LOD 切换距离，单位为米；L0 为基准切换距离，单位为米；Sref 为参考对象屏幕面积，单位为像素平方；So 为当前对象屏幕面积，单位为像素平方；v 为相机移动速度，单位为米每秒；vref 为参考速度，单位为米每秒；γ为速度衰减系数，单位为无量纲。该关系把对象屏幕面积和相机移动速度共同纳入 LOD 切换距离，避免远景对象过早占用高精度资源，并把 Ld 用于可见范围分级。

3.3 完善模型资源压缩与缓存策略

模型资源压缩应区分几何、纹理和材质三个对象。几何压缩可以减少网络传输和解析体量，但过度压缩会增加解码时间；纹理压缩可以降低显存占用和采样带宽，但需匹配浏览器与设备支持格式；材质复用可以减少着色器变体和状态切换，却不能抹去语义差异。引擎架构应记录每类资源的压缩比例、解码成本和显示等级，把压缩收益与运行成本一起评估。浏览器缓存、IndexedDB、Service Worker 和内存缓存承担不同层级，首屏加载必要骨架，交互稳定阶段补齐细节，内存压力升高时回收远景和低优先级纹理。业务热点对象的缓存键应包含对象 ID、LOD 等级和材质版本，防止低精度资源覆盖高精度查看任务。

4 结语

Web 端图形引擎架构设计与三维场景展示性能改善具有连续技术关系：资源层决定模型、纹理和材质能否复用与分级，引擎层决定渲染状态、绘制批次和任务调度能否稳定收敛，展示层决定可见范围、画面质量和交互响应能否按终端能力动态调整。性能改善不能只依赖单点压缩或接口更换，而应围绕帧耗预算建立资源入队、渲染输出和质量调节链条。后续系统建设应强化双后端适配、资源标准化、运行监测和异常降级，使浏览器三维展示在大规模对象、弱网络和移动终端条件下保持清晰边界。

参考文献：

[1] 刘秋燕,孙赛,杨礼国.一种 Web 端图形引擎中自定义空间多边形面积的计算方法[J].计算机时代,2022,(4):55-57.

[2] 邱大佐,范繁荣.沉浸式森林防灭火教学系统三维场景的设计与优化探究[J].科技资讯,2026,24(4):222-225.

[3] 张小兰,伍杰华,蔡雪莲,等.基于 Web 的三维场景重建系统架构设计及关键技术研究[J].电子元器件与信息技术,2023,7(2):109-113.

[4] 郭泽,谭衢霖,戴泽宇,等.基于 WebGIS 的铁路线路三维场景构建[J].铁路计算机应用,2022,31(5):22-27.

[5] 林业胜.基于虚拟现实技术的数字三维场景设计与开发[J].黑河学院学报,2024,15(3):68-70+107.