 DOI: https://doi.org/10.12349/iser.v7i5.10518

A High-Performance and Visually Continuous Gradient Rendering Method for UI Text—Based on Unity Engine

Zihao Jiang

Department of Information and Artificial Intelligence, Wuhu Vocational Technical University, Wuhu, Anhui, 241006, China

Abstract

Addressing persistent challenges in UI text gradient rendering-namely performance bottlenecks and visible color discontinuities at character boundaries-this study introduces a lightweight optimization framework centered on dynamic vertex color manipulation. Implemented within the Unity engine, and linear interpolation is directly applied to vertex colors to ensure seamless gradient transitions across individual glyphs and multi-character sequences. Critically, the method modifies only vertex color attributes while preserving the original mesh topology and material references, thereby maintaining UGUI batching integrity. No custom shaders are required; processing time for 100-character text consistently remains below 0.01 ms. Gradient parameters (direction, color stops, interpolation mode) support real-time adjustment, and the solution operates natively across Unity's Standard, URP, and HDRP pipelines. Validated in industrial AR/VR interfaces under Anhui Provincial Key Project (2023AH052383), this approach delivers a practical, engineer-friendly solution for high-fidelity UI text rendering-with demonstrated adaptability to non-Unity engines through minimal structural refactoring.

Keywords

UI font rendering; gradient algorithm; vertex color modification; Unity UGUI; batching optimization; visual continuity

面向高性能与视觉连续性的UI字体渐变渲染方法研究——以Unity引擎UI为例

江自昊

芜湖职业技术大学信息与人工智能学院，中国·安徽 芜湖 241006

摘要

本文针对UI系统中字体渐变渲染存在的性能瓶颈与视觉断裂问题，提出一种基于顶点颜色的动态修改的高性能渐变渲染优化框架。该方法依托Unity引擎，实现单字符及多字符的平滑过渡渐变。该方案在不改变原始网格拓扑与材质的前提下修改顶点颜色属性，因此不会影响到UGUI合批机制，且无需自定义Shader，且使得百字文本处理耗时稳定低于0.01ms；同时支持运行时动态调整,可兼容各种SRP渲染管线.为AR/VR及高性能交互应用中的UI字体渲染提供了可落地的技术路径,该方法可适用于其他渲染引擎。

关键词

UI字体渲染；渐变算法；顶点色修改；Unity UGUI；合批优化；视觉连续性

1 引言

在当代交互式应用与ARVR开发中，用户界面（UI）的视觉表现力已成为用户体验的核心维度之一[1]。文本作为UI的重要部分，在设计上也有了各种要求，渐变字体渲染作为一种高阶视觉特效,能够有效构建视觉层次,引导用户注意力,并赋予界面独特的艺术风格与品牌识别度.然而，在使用各种渲染引擎（包含2D和3D）开发时，如何实现兼具高性能,视觉连续性与运行时灵活性的文本渐变效果，长期面临技术实现的挑战。

以Unity的UGUI渲染框架为例，渐变字体渲染传统的实现路径有多Text叠加、渐变贴图配合Mask蒙板、预烘焙纹理法及材质（Shader）硬编码法等方法，但是这些方法

【基金项目】安徽省自然科学研究重点项目“焦化工业AI数字孪生仿真系统的研发”（项目编号：2023AH052383）2024年安徽省高校理工科教师赴企业挂职实践计划(项目编号：2024jsqygz166）
2024年安徽省“科技副总”计划。

【作者简介】江自昊（1984–），男，中国安徽芜湖人，硕士，副教授，从事计算机应用、ARVR应用、AI应用研究。

它们存在不少缺陷。如多Text叠加法本质为将文本拆解为单字符[2]，严重破坏UGUI合批机制，在动态长文本场景下引发帧率骤降，且颜色过渡呈阶梯状断裂[3]，无法实现光学级平
滑渐变。因此文字本体颜色恒定，无法动态修改；且Mask边缘易出现锯齿,半透明穿帮或对齐偏差,视觉真实性与专业度不足.预烘焙纹理法需离线生成含渐变效果的字体图集[4]，因此也有无法动态修改的弊端,在多语言切换,主题切换等场景下无法适用。Shader硬编码法虽本质通过渲染计算，却因UGUI Text顶点坐标缺乏信息，直接导致合批机制失效，渲染性能反低于其他优化方案[5]。

针对上述痛点,本文提出一种基于顶点色动态修改的渐变渲染优化框架.并在本人背景项目中尝试将该方法用于Unity引擎。通过在BaseMeshEffect组件的ModifyMesh阶段对文本网格顶点颜色进行精准调控[6]，确保渐变在跨字符,多行文本中“丝滑”衔接,同时严格复用原始顶点结构，避免顶点数量膨胀[7]。该方案具备三重优势：其一，渲染效率卓越——仅修改顶点颜色属性,材质与纹理保持不变，完美融入UGUI合批流水线，百字文本处理耗时稳定低于0.01 ms;其二,动态适应性强——支持运行时实时调整颜色序列、全方向角度（0°~360°）、插值模式（线性/平滑)等参数,可适用应用主题频繁切换,数据频繁切换等需求；其三，开发友好度高——无需编写自定义Shader,天然兼容各种SRP（渲染管线策略）的UI渲染路径，显著降低美术与程序协作门槛[8]。

2 原理

本方法的核心思想在于将文本视为连续的几何整体,通过空间映射与颜色插值实现视觉无缝的渐变效果.其原理流程可概括为四个关键环节：

2.1 空间边界界定

系统首先识别文本整体的几何轮廓,通过“绑定盒”动态计算其最小外接矩形边界（Axis-Aligned Bounding Box, AABB）。这样，该边界就可自适应文本内容、字体样式及多行排版变化，为渐变提供统一的空间参考系。

2.2 位置归一化映射

将每个顶点在边界内的物理坐标线性压缩至标准化区间[0,1]。公式为：

其中 t 表示顶点在渐变轴上的相对进度（t=0 为起点，t=1 为终点）。对于斜向渐变，可通过坐标旋转处理后将顶点投影至指定方向轴后再归一化，确保方向灵活性。

$$t = \frac{P_{\text{local}} - P_{\min}}{P_{\max} - P_{\min}}$$

2.3 颜色连续插值

依据预设的颜色序列（如起始色[C0,C1]、[C1,C2]）按归一化值 t 进行线性插值：

若 t∈[tk,tk+1]，则插值颜色为：

$$\mathbf{C}(t) = \mathbf{C}_k + (t - t_k) \cdot \frac{\mathbf{C}_{k+1} - \mathbf{C}_k}{t_{k+1} - t_k}$$

多色节点场景下,自动在相邻色段间分段插值,生成平滑过渡的色彩梯度。

2.4 属性融合

根据原始顶点的空间坐标,连接拓扑及纹理映射关系计算渲染管线在渲染阶段依据顶点色进行的片元插值,最后将计算所得颜色赋值给对应顶点,最终合成视觉连续的渐变文本。

原理如图 1所示，该方法有全文本全局和单个字符归一化两种策略,不管是跨字符还是单个字符,都可以建立起相应的坐标系映射.实现“视觉连续性”与“渲染高性能”的有机统一.该原理具备高度普适性,可适配任意字体,字号、文本内容及复杂排版需求。



图 1 原理图

3 具体实现

本方案以Unity UGUI渲染框架为实施载体，其技术路径严格遵循图形学通用原理且方法具备强跨引擎适应性,以下结合Unity实现细节展开阐述。

3.1 组件选择及思路

继承Unity官方BaseMeshEffect基类，通过[RequireComponent(typeof(Text))]建立组件约束。通过BaseMeshEffect“渲染前回调”的机制，实现顶点渲染处理及回调。

3.2 全局几何边界动态计算

在ModifyMesh入口调用自定义GetBounds方法，遍历VertexHelper提供的顶点流（vh.GetUIVertexStream(verts)），实时计算全文本最小外接矩形（AABB）。该过程可以自动融合单、多行文本字符的几何分布，生成统一空间参考系。同时,为规避每帧冗余计算,引入内容哈希比对机制:仅当文本内容,字体尺寸或锚点变更时触发重计算,其余帧复用缓存边

界值。关键边界提取逻辑：

Rect bounds=GetBounds(verts); //verts为全文本顶点列表

3.3 顶点归一化与方向自适应投影

基于上一步骤所得边界,将每个顶点局部坐标映射至[0,1]标准化空间,并依据方向生成单位投影向量.此步骤将任意斜向渐变统一转化为一维参数计算,可消除硬编码依赖.投影值经处理后能作为颜色采样依据,确保跨字符顶点在全局坐标系下连续过渡：

Vector2 normalized =

bounds.xMin)/bounds.width,(v.position.y -

new((v.position.x - bounds.yMin)/bounds.height);

float t = Mathf.Clamp01(Vector2.Dot(normalized, dir));

3.4 顶点色精准插值与网格无损更新

调用Unity内置Gradient.Evaluate(t)完成颜色插值：根据顶点的分布,位置计算颜色的渐变值,然后修改顶点颜色属性。这样可保留字符原坐标,UV,法线及三角拓扑结构,使UGUI BatchBuilder将渐变文本与普通文本合并至同一批次渲染.更新后通过vh.Clear()与vh.AddUIVertexTriangleStr-eam(verts)安全回写网格：

v.color = gradient.Evaluate(t);

verts[i] =v; // 无几何结构变更

3.5 渲染管线兼容性封装

作者在项目开发中有使用多种渲染管线的需求,由于该方案完全复用UI Shader的顶点色通道，且任何渲染管线字符UI的顶点通道规则相同，因此这种方式一般无需根据不同的可开发渲染管线而发生改变,规避了不同渲染管线需要自定义材质实例的必要性。Frame Debugger实测验证：百字文本Draw Call恒为1，证明该合批保护机制具备管线无关性。

3.6 运行时性能优化策略

在方案应用项目后的压力测试中（版本：Unity 2021.3.15f1, 试验环境：iphone 13、web），暴露出两类性能瓶颈：其一，每帧强制重建List<UIVertex>导致持续性GC Alloc（百字文本单帧约0.8KB），在移动端引发周期性卡顿（GC停顿峰值达8ms）；其二，静态文本（如界面标题、固定说明)在内容未变更时仍重复执行边界计算与颜色插值,百字文本单帧CPU耗时高达0.12ms，难以满足AR/VR等60fps+高帧率场景需求.此类问题同样会存在于其他未对此优化的引擎中,若未实施针对性优化,同样会导致感觉上的“跳帧”。

针对上述问题，本方案实施实施了三项优化策略：

1. 顶点缓冲区对象池复用：预分配固定容量顶点缓存，规避每帧堆内存分配。参考代码：

vertexCache ??= new List<UIVertex>(500);

vertexCache.Clear(); // 复用缓冲区

2. 内容哈希脏标记检测:计算文本内容,字体尺寸,渐变参数的联合哈希值,仅当哈希变更时触发重计算,“智能跳过”未变化的字符，使其不用重复渲染，参考代码：

if(cachedHash==ComputeCurrentHash())return; // 静态文本零计算

3. 结构体原生操作规避装箱：全程使用UIVertex结构体进行顶点修改，避免引用类型转换开销，此细节对IL2CPP及AOT编译环境的性能提升尤为关键。

化效果经Unity Profiler严格验证(测试环境:Unity2021.3.15f1,iPhone 13,Web），数据对比如下：

表 1 优化性能对比

评估维度	优化前（百字文本）	优化后（百字文本）	提升效果
单帧CPU耗时	0.12 ms	0.05 ms	58.3%↓
每帧GC Alloc	0.8 KB	0.1 KB	87.5%↓
静态文本帧耗时	0.12 ms	<0.01 ms	>90%↓
1000字符帧稳定性	帧率波动±8fps	帧率波动±1fps	显著平滑

优化后方案在动态场景(如血量数字实时渐变)与静态场景（如界面标题）均保持亚毫秒级开销，且Draw Call始终稳定。

4 总结

4.1 结论

本文提出一种以“全文本全局空间归一化”与“顶点属性最小化修改”为核心的UI字体渐变渲染方法。在Unity UGUI中验证表明：该方案以最小计算代价实现视觉连续性增强,且具有夸引擎通用性和处理的灵活性,为该项应用场景解决方案提供新思路。

4.2 应用场景

针对性地，本方案系统性解决了三大行业痛点：

1.性能瓶颈——根治合批断裂与资源膨胀；

2.视觉缺陷——消除跨字符渐变断裂与边缘瑕疵；

3.维护成本——简化工作流，支持Inspector实时预览与参数热更。

4.3 改进方向展望

未来工作将：

扩展至TMP,FairyGUI等主流文本组件,构建统一渐变中间层；

提炼核心算法为跨引擎SDK（抽象顶点流操作接口），降低Unreal/Cocos等平台迁移成本；

探索与AIGC结合的智能参数推荐（如根据文本情感自动匹配渐变色系），推动UI设计智能化演进。

本研究不仅提供可落地的工程方案,更为“效率与美学

协同优化”的UI特效设计新范式提供了技术支持。

参考文献

[1] 彭群生,金小刚,万华. 计算机图形学应用基础[M]. 北京:科学出版社,2009(2):32-36.

[2] 邢伟佳,徐邦联. 基于扩散模型的纹理生成研究进展综述[J/OL]. 智能计算机与应用.https://doi.org/10.20169/j.issn.2095-2163.25082701.

[3] 张亚平,文圣丰,于翔.基于FreeType的双缓存文字快速渲染方法[J].长江信息通信,2023,36(01):90-92.

[4] 严莉,彭波,李强,等.基于Mamba的图像分割应用综述[J/OL].计算机应用研究,1-12[2026-03-26].https://doi.org/10.19734/j.issn.1001-3695.2025.11.0483.

[5] 张福峰,王敏,金会赏.基于Unity3DUGUI动态材质修改的实现[J].电脑知识与技术,2020,16(25):218-219.DOI:10.14004/j.cnki.ckt.2020.3040.

[6] 赵志豪.基于Unity3D和Lua的脚本化游戏UI框架的设计与实现[D].南京大学,2016.DOI:10.27235/d.cnki.gnjiu.2016.001059.

[7] 张忆楠,严正,姚莉.实时渲染引擎架构[J].中兴通讯技术,2013,19(03):48-53.

[8] 盛斌,孟维亮,孙汉秋,等.MCGIM-Based Model Streaming for Realtime Progressive Rendering[J].Journal of Computer Science&Technology,2011,26(01):166-175.