

基于 OpenGL 实例化技术的文字标牌高性能渲染方法

王盛朋¹ 杨柳静¹

WANG Shengpeng YANG Liujiang

摘要

针对海上目标持续存在且数量众多的特点,传统的 OpenGL 文字渲染方式在利用 GPU 加速二维场景中标牌的绘制过程中,遭遇了由 CPU 与 GPU 间数据传输限制导致的性能瓶颈,难以高效处理大规模文字标牌的实时更新需求。对此,结合船舶信息系统应用的实际需求,创新性地设计了一种基于 OpenGL 实例化技术的大规模文字标牌渲染方案。通过集中的字符批次绘制策略,结合 OpenGL 实例化渲染技术,显著提升了大量文字渲染的效率。实验验证表明,在模拟 10 000 个典型海上目标标牌、每秒更新 1 次的高负载场景下,新方法能够实现 70 帧/s 的流畅渲染效果,成功突破了原有 OpenGL 文本渲染的技术局限,为二维文字信息的高效可视化提供了强有力的支持。

关键词

OpenGL 实例化; CPU-GPU 传输瓶颈; 实时更新; 海量文字渲染; 高性能

doi: 10.3969/j.issn.1672-9528.2024.09.006

0 引言

在船舶信息系统中,目标长期存在,故面向海上的船舶信息系统需要显示大量目标信息。目前,船舶信息系统常用的 CPU 绘制方法难以应对较大量的文字标牌绘制,容易造成程序运行卡顿、用户体验下降。为支撑船舶信息系统流畅运行、为用户提供优秀的使用体验,需研发使用基于 OpenGL 的 GPU 进行绘制的高效目标标牌绘制方法。同时,尽管拥有强大渲染能力,但 OpenGL 本身没有提供任何文字处理能力,而主流的 OpenGL 文字渲染方式受限于 CPU-GPU 交流速度瓶颈,无法实现大量文字的流畅渲染。

因此,本文提出一种基于 OpenGL 3.3 及以上版本实例化技术实现二维场景下渲染大量文字标牌的方法,充分利用显卡渲染性能优势,最终实现了 1 万项典型目标标牌的流畅渲染。

1 现有技术分析

OpenGL 不包含文字处理功能,为能够在 OpenGL 场景中进行文字渲染,有如下几种常见渲染方法。

1.1 位图字体方法(bitmap font)

位图字体方法在纹理的预定义区域中预先生成需要使用的所有字符^[1]。当需要在场景中渲染一个字符时,只要通过渲染字符纹理中一块特定的位图字体区域到一个 OpenGL 场景中的二维四边形上即可。该方法的优点是相对来说容易实现,且位图字体已预光栅化,因此渲染效率也很高^[2]。该方

法的缺点在于方法缺少必要的灵活性,例如,当需使用不同的字体或者大小不同的文字时,就需要重新编制一套全新的位图字体纹理。更严重的问题是,这种方式几乎仅能支持 ASCII 字符集,当需要使用更大的字符集时就很难。例如,编制一套包含 3000 个常用汉字的字符集纹理,其所消耗资源较高。故该方法不适用于中文文字的渲染。

1.2 FreeType 渲染方法

使用 FreeType 渲染方法时,通过调用 FreeType 库加载字体,获得每个字符的尺寸,用于计算每个字符的位置生成二维四边形,再调用 FreeType 库生成合适尺寸的字符纹理贴在 OpenGL 场景中二维四边形上^[3]。该方法解决了位图字体方法的灵活性问题,可用来渲染任意尺寸的任何语言文字标牌。但因该方法对每一个字符都生成一个四边形对象,故很容易因 CPU 与 GPU 频繁交流而遇到性能瓶颈。实验显示,在 OpenGL 场景中,直接使用该方法,当字符(示例)数量达到 2000 个时,性能就已经低至 0.7 帧/s,且此时 GPU 利用率很低。故该方法不适于直接渲染大量文字标牌。

1.3 其他方法

除了以上两种方法外,陈飞等人^[4]、周小军等人^[5]描述了一种利用 SDL 或者 GDI 绘制线程生成文字纹理并利用 OpenGL 输出的方法,但这些方法本质依然是利用 CPU 绘制文字,大量文字的绘制时依然会遇到 CPU 绘制效率的性能瓶颈。芮小平等人^[6]描述了使用 OpenGL 渲染三维字符的方法,但该方法着重描述三维字体的渲染,对于效率方面并未重点关注。

1. 中国电子科技集团公司第二十八研究所 江苏南京 210000

2 利用实例化技术加速字符渲染思路

OpenGL 实例化方法是一种只创建一个实例、只调用一次渲染函数，就能渲染很多相同或类似实例的技术^[7]。配合实例化数组缓冲以及精心设计的顶点着色器、片段着色器的 GLSL 程序，实例化技术可以控制实例渲染的数量、位置、大小和颜色^[8]。

考虑船舶信息系统的实际情况，典型的标牌内容由唯一标识、更新时间和名称（呼号）等部分组成。其中字唯一标识和时间是由 0～9 十个数字字符和“:”（冒号）共 11 种字符组成；绝大部分船舶名称（呼号）由数字、字母（大小写共 52 个）和少量符号（典型如“/”“:”）组成，可知共计使用不超过 94 种字符（参考 ASCII 基础字符集）。使用中文字符的情况较少，且由于行业规定，中文字符往往是省简称和船舶用途的结合，命名方式统一、字符重复较多，如“浙某渔 123”。通过“船讯网”^[9]随机挑选统计 10 000 个标牌，汉字数量约为 400 个。综上，初步认为，在船舶信息系统实际使用中，渲染 10 000 个文字标牌所需的字符种类数量不超过 500，即结合实例化技术，可以使用不超过 500 条渲染指令进行 10 000 个标牌的渲染，远未触及性能瓶颈，使用 FreeType 文字渲染方法结合实例化技术进行目标标牌渲染具备可行性。

具体来说，传统 FreeType 渲染方法中，对于标牌中的文字，需要知道标牌中每个字符的位置、颜色、大小等参数，并逐字符调用渲染指令提交给 GPU 进行渲染，造成大量 CPU-GPU 交互。根据实验^[10]，在一个渲染循环中，如果存在 2000 次 GPU 渲染指令调用，帧率将下降到 0.7 帧/s 左右。

应用实例化技术，在字符串内容有多重重复文字的场景中，可先分析统计得出全部字符串中使用的所有字符，再将计算获得的字符参数（位置、大小、颜色等）按照字符分类，统一拼装成多个渲染批次，最后调用实例化渲染命令进

行 GPU 渲染。如此，可大幅减少 GPU 指令调用次数，规避 CPU-GPU 交互的性能瓶颈。如图 1 所示，渲染同样的字符串，常规渲染方法需要 6 条渲染指令，而实例化渲染则需要 2 条指令，可有效减少 CPU-GPU 交互。

3 详细实现

上述思路的关键在于统计所有需要绘制标牌的字符，并为每个字符构建包含字符属性的实例化缓冲数组。本文后续着重描述字符的位置属性、字符的大小和颜色等其他属性，可使用相同的处理方式。

通过如下步骤实现实例化字符绘制。

第一步，将标牌操作转化为针对单个字符的操作。为利用实例化绘制技术，需将标牌分解为字符的组合，按照字符进行归类，并将每个标牌的操作分解、生成对应的每个字符操作指令。具体来说，对应标牌的增（Create）、删（Delete）、更新（Update）操作，转化生成成为字符的增、删、更新操作指令。下文对这些指令简称为“CUD 指令”。

第二步，构建字符缓冲数组，用来保存字符的绘制信息。如字符在标牌中的序号、字符屏幕位置、字符其他属性如大小颜色等。

第三步，用第一步生成的字符 CUD 操作指令，更新第二步构建出的字符缓冲数组信息，并调用实例化绘制指令绘制字符缓冲数组中的内容，最终实现标牌的各类操作。下文描述各个步骤的实现方法。

3.1 将标牌操作转化为字符 CUD 指令

在实际应用中，程序通过调用标牌操作接口进行文字标牌的增加、更新和删除。本方法中，为利用实例化技术，需要将标牌内容字符串拆分成独立字符，并以字符为单位使用 OpenGL 进行实例化渲染，因此需要将标牌接口生成标牌字符的字符 CUD 指令。

文字标牌转换到 CUD 指令步骤复杂，利用面向对象编程设计思想，设计实现 GL 标牌类（CusGLLabel）、GL 字符类（CusGLChar）和 GL 字符的 CUD 指令类（CusCUDInstruction），用于封装转换标牌到字符、更新动作到 CUD 指令所需的全部数据和算法，如图 2 所示。其中，“GL 标牌”用于为记录应用程序需要渲染的标牌内容，主要包含的数据有：标牌内容（字符串）、标牌屏幕位置、拆分内容获得的“GL 字符”集合，以及标牌中字符的“CUD 指令”集合。字符集合使用哈希表结构，键为字符的 UTF-8 编码，值为字符信息列表；CUD 指令集合使用哈希表结构，键为字符的 UTF-8 编码，值为字符指令列表。

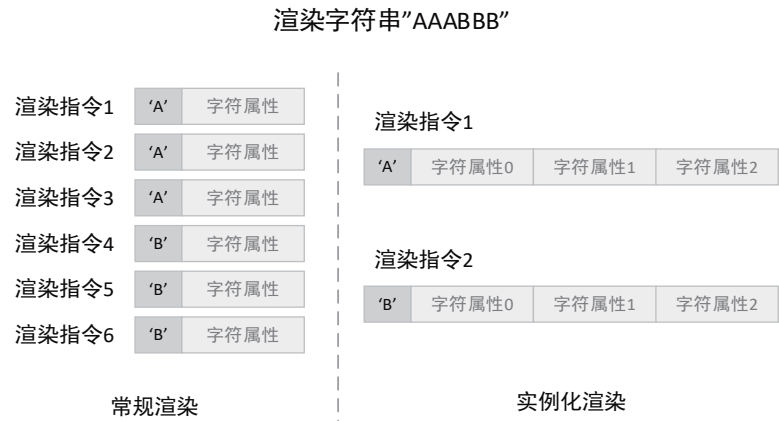


图 1 传统 OpenGL 渲染方法与实例化渲染方法对比

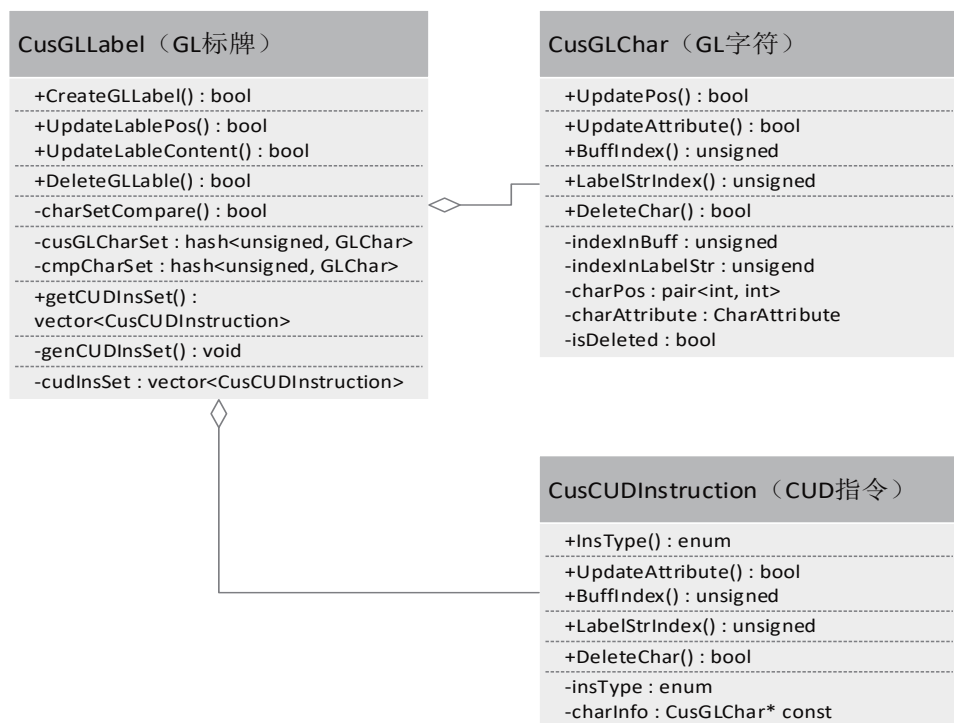


图2 关键类图

生成字符串“14:45:20”的CUD指令集



图3 创建文字标牌时构建的字符集与指令集

“GL 字符”用于记录标牌中的每个字符的信息，主要包含字符在标牌字符串中的相对位置、字符在屏幕上的坐标和字符大小颜色等其他属性信息。其中，记录字符在字符串中的相对位置，是为了应对在一个字符在标牌字符串中重复出现时，区分每个出现的字符。

字符“CUD 指令”是本文渲染方法的核心数据结构，用于直接指示渲染操作和渲染内容，主要包含标牌的唯一标识、修改类型（增、改或者删）和需要修改的“GL 字符”的引用。实际上，本文中所有其它数据结构及其运用，如 GL 标牌、GL 字符，最终都是为了生成字符 CUD 指令的中间产物。

在完成构建上述类后，整个生成 CUD 指令的过程，可以简要描述成如下步骤。

对每一个需要绘制的文字标牌创建对应的 GL 标牌，GL 标牌保存标牌绘制信息，并将标牌字符内容按照字符进行分类，形成 GL 字符集合。

当发生新建、删除 GL 标牌时，GL 标牌可根据操作直接生成可执行的字符创建和删除指令；当标牌发生更新时，典型如标牌位置更新、内容更新，GL 标牌生成新的 GL 字符集，并通过新旧 GL 字符集比较，生成 CUD 指令。

下面具体描述使用 GL 标牌、GL 字符，将标牌的创建、删除和修改操作转化为字符 CUD 指令的过程。

创建标牌时，同步创建 GL 标牌数据结构，并遍历字符串中每个字符，根据字符编码创建 GL 字符集；遍历字符的过程中同步生成“字符 C（创建）指令”，并添加到标牌的字符指令集中。例如，创建对于标牌文字内容“14:45:20”，创建字符集和字符指令集结果如图3所示。生成的字符 CUD 指令携带 GL 字符的关键属性信息，典型如字符的屏幕位置。

删除标牌时，遍历字符集中的每一个字符，生成字符删除指令，添加到标牌的字符指令集中。过程与“创建标牌”的过程类似，不同之处在于字符指令的类型为“D”（删除）。完成删除字符的 CUD 指令后，再销毁对应的 GL 标牌。

更新标牌包含两种方式，分别为位置更新和内容更新。位置更新操作相对简单，仅需遍历字符串中每一个字符，根据新的位置生成字符更新指令，添加到标牌的字符指令集中即可，过程类似于“创建标牌”，不同之处在于指令类型为“U”（更新）。

另一种较为复杂的更新方式为标牌内容发生变化。此时，需要通过 GL 标牌中字符串内容的比较得出字符的 CUD 指令集。具体做法是，首先根据新的标牌内容生成一份临时 GL 字符集，接着依次比较原有字符集与临时字符集中的每一个 GL 字符信息。在对比过程中，对于 GL 字符信息不一致的部分，生成相应的字符 CUD 指令。例如，对于文字标牌内容从“14:45:20”变化到“14:52:23”时，原 GL 字符集、临时 GL 字符集、对比生成的字符 CUD 指令结果如图4所示。

字符串“14:45:20”变换为“14:52:23”
生成字的CUD指令集



图4 修改文字内容时构建字符指令集示意图

3.2 将标牌操作转化为字符 CUD 指令

实例化缓冲数组是分配在显存中连续的数据存储空间，通过 OpenGL 提供的 API 进行分配和删除。创建缓冲数组时，API 将返回缓冲数组的地址，通过该地址操作缓冲数组中的内容。

实例化缓冲数组用来保存每个字符的属性集合，典型如字符在屏幕中的位置信息。将字符 UTF-8 编码作为 Key，与字符信息数据结构组成 Key-Value 对，如图 5 所示。

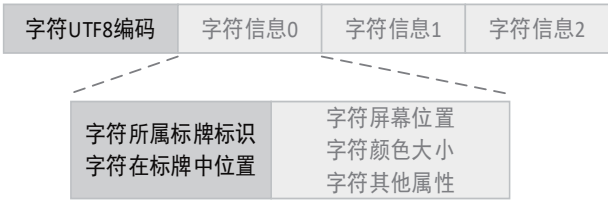


图5 实例化缓冲数组构建示意图

需要修改缓冲内容时，根据字符 UTF-8 编码查询到对应字符的缓冲地址，通过 OpenGL 内存映射方法操作对应缓冲地址内容。而在 GPU 进行文字渲染时，依次渲染每个字符的缓冲数组中字符属性信息即可。

3.3 将标牌操作转化为字符 CUD 指令

执行字符 CUD 指令分为两个步骤：第一步，遍历所有的“GL 标牌”收集字符 CUD 指令，并按照指令类型将指令集分类并存储为“创建”“删除”和“修改”三个指令表；第二步，依次执行“U”（更新）、“D”（删除）、“C”（创建）三个指令表，根据表中指令，在显存中更新对应字符的实例化缓冲数组。下面描述具体的执行过程。

执行“更新”指令时，遍历每个字符的每一条修改指令，根据指令中实例化缓冲数组下标的指示，用指令中的新字符位置信息覆盖缓冲数组中旧的位置信息即可。

执行“删除”指令时，遍历每个字符的每一条删除指令，根据指令中实例化数组下标的指示，用实例化数组末尾的位置信息替换下标出的位置信息，并缩减实例化数组的长度。这样的方式可避免删除操作造成数组内容的大量移动。

执行“创建”指令时，遍历每个字符的每一条创建指令，在字符集对应的实例化数组末尾追加位置信息。若该字符从未出现过，则需要通过 FreeType 库创建字符纹理，并在显存中新建对应的字符实例化缓冲数组。

3.4 实现小结

通过组合上述步骤，能够根据标牌操作调用生成字符 CUD 指令，再通过执行 CUD 指令操作字符实例化缓冲数组，最后调用 OpenGL 实例化渲染命令进行显存中实例化缓冲数组内容的输出，从而最终实现标牌文字的展现与各类操作。该过程的流程图如图 6 所示。

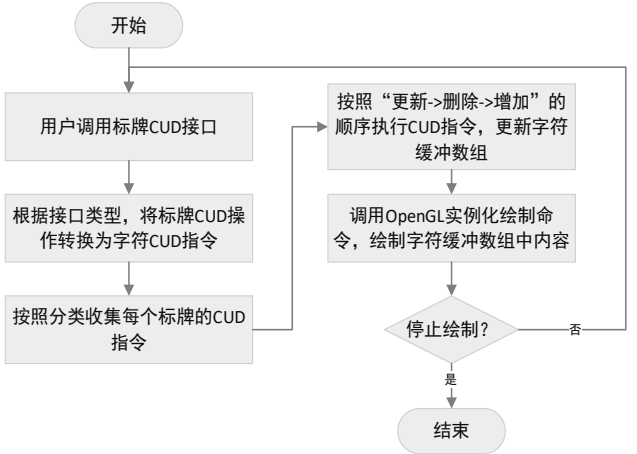


图6 实例化绘制方法流程图

4 实验及结果

为验证文中方法的实际效果，通过构建测试程序渲染典型目标标牌内容（包含目标编号、目标名称、时间），来验证本方法的渲染性能；作为对比，使用 CPU 绘制同样内容并记录绘制耗时。测试中使用的硬件配置与开发软件见表 1。

表 1 试验环境

CPU	Xeon E5-1620v3@3.50 GHz 4 核 8 线程
GPU	GeForce GTX 980
内存	16 GB@1600 MHz
操作系统	Windows 7
开发工具	Visual Studio 2010 Qt4.8.7 OpenGL 3.3

此外，实验使用的对话框分辨率为 1600×1200，标牌内容如图 7 所示，使用定时器进行每秒 1 次的内容更新（修改标牌中的时间），持续运行 15 min。其中，在 CPU 标绘绘制测试中，使用 Qt 界面框架中 QPainter 类提供的绘制文字接口进行单线程绘制。GPU 绘制和 CPU 绘制程序在编译时，均使用 Win32 Relase 模式的默认优化参数。测试结果见表 2。



图 7 试验使用的标牌内容

表 2 试验结果数据

	CPU 标准绘制 (单线程) /ms	OpenGL 实例化 标牌渲染 /ms
10 个标牌 (平均 / 最大)	< 1/16	< 1/11
1000 标牌 (平均 / 最大)	71/141	3/17
10000 标牌 (平均 / 最大)	733/951	14/130

测试使用本文的方法渲染 10 000 个标牌的运行截图如图 8 所示（标牌大量重叠）。

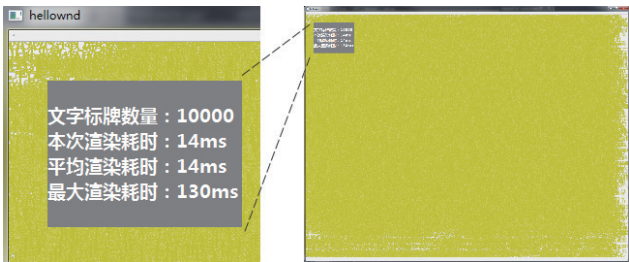


图 8 绘制运行截图

测试结果显示，本文论述的 OpenGL 实例化技术的大量标牌渲染方法，在典型标牌渲染测试中达到了预期目标，可实现 10 000 个典型目标标牌的流畅渲染。在 10 000 个典型标牌绘制场景下，相比于 CPU 绘制，使用 OpenGL 实例

化的渲染方法让文本标牌绘制性能得到了 50 倍左右的提升，收益显著。

5 结语

本文提出了一种基于 OpenGL 实例化的大量二维文字标牌渲染方法，并较详细地描述了实现的具体步骤。实验结果显示，该方法实现了 10 000 个典型文字标牌流程渲染的能力。目前，该方法已初步结合信息系统进行测试实验，性能上达到预期目标。未来，将基于该方法完善具体功能，以期真正具备实装条件，让信息系统获得更加优秀的用户体验。

参考文献：

[1] 李彦龙, 刘建, 孟凡. 基于 OpenGL 事件驱动的雷达目标仿真系统 [J]. 雷达与对抗, 2016,36(3):32-35.

[2] 吴仁彪, 吴海宁, 屈景怡, 等. 一种高效 OpenGL 多线程文本绘制方法 [J]. 中国民航大学学报, 2013,31(4):23-26.

[3] 柳佳佳, 栾晓岩, 边淑莉. 基于 OpenGL 的二维矢量地图可视化技术研究 [J]. 测绘科学, 2013,38(5):88-90+93.

[4] 陈飞, 黄海明, 杨猛, 等. 基于 SDL 和 OpenGL 实时绘制中文字体 [J]. 计算机工程与设计, 2011,32 (8): 2764-2767, 2783.

[5] 周小军, 夏青, 蒋秉川. OpenGL 多线程电子地图显示研究 [J]. 测绘科学, 2010, 35(6):105-106+113.

[6] 芮小平, 张彦敏, 杨崇俊. OpenGL 中文字的几种常用绘制方法 [J]. 计算机工程与应用, 2002,12(3):128-129.

[7] 张晔嘉, 朱贤平, 孙翌晨, 等. 高效海量空间目标可视化方法 [J]. 指挥信息系统与技术, 2020,11(1): 55-61.

[8] 占伟伟, 李坪泽, 王辉, 等. 自主可控环境下三维海量态势显示优化方法 [J]. 指挥信息系统与技术, 2019,10(2): 80-84.

[9] 船讯网 [EB/OL]. <https://www.shipxy.com/Ship/Index>.

[10] JOEY D V. Instancing [EB/OL]. <https://learnopengl.com/Advanced-OpenGL/Instancing>.

【作者简介】

王盛朋（1988—），男，安徽合肥人，硕士，工程师，研究方向：计算机可视化。

杨柳静（1987—），女，河南驻马店人，硕士，高级工程师，研究方向：计算机可视化。

（收稿日期：2024-06-12）