# 基于 OGRE 的汉字渲染方案①

彭四伟 蔡 明 (北京化工大学 信息科学与技术学院 北京 100029)

**摘 要：** OGRE（object-oriented graphic render engine）作为一种通用的图形渲染引擎并不提供专门针对汉字的渲染方案。汉字字符与英文字符存在很大差别，不能简单套用英文字符的渲染方法，分析了汉字在 OGRE 的场景中的 3 种不同的渲染方案。并在讨论和比较的基础上，深入的研究并实现了在现阶段比较有价值的方案，为进一步研究积累了经验。

**关键词：** OGRE；汉字渲染；3D

## A Chinese Characters' Rendering Method on OGRE

PENG Si-Wei, CAI Ming

(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

**Abstract:** OGRE(object-oriented graphic render engine) is a common graphic render engine. It does not support any Chinese characters' rendering method. Chinese characters are very different from English ones so one cannot simply copy the method rendering English Characters. This paper analyses 3 different Chinese characters' rendering methods based on OGRE and implements a valuable one, which could also give way to further studies.

**Keywords:** ogre; Chinese characters' rendering; 3D

## 1 引言

### 1.1 背景

伴随着计算机硬件的不断发展，3D 软件越来越多的从概念变为现实。在虚拟社区，产品仿真,演示等领域 3D 软件能够提供身临其境的逼真感觉，更加直观，更易理解。OGRE 作为开源的专业图形图像引擎，充分利用了成功的面向对象技术，使 3D 软件的实现更加方便，已有广泛应用。3D 场景中的文字起着说明的作用，在许多应用中是不可缺少的。所以，如何在 OGRE 的 3D 场景下实现汉字的渲染，并保证运行效率至关重要。

非常遗憾的是，由于 OGRE 只是作为一个 3D 渲染引擎被设计出来，并且那是它唯一的应用。它不能提供对文字渲染的直接支持[1]，只是留出了一定的程序接口。本文考虑了汉字在 3D 场景中的三类渲染解决方案。第一类方法，通过字体信息在 3D 场景中生成字形的 3D 模型，然后用适当的材料覆盖在表面。这样产生的汉字有立体感，有 3D 效果。但是，根据字体信息生成 3D 模型非常困难，目前没有好的解决方案。第二类方法，是利用 2D 的渲染方法，把文字渲染到在 3D 层之上的 2D 层上。这样就能方便的利用 2D 已有的技术处理好汉字在 3D 场景的显示问题。但是，这样会使文字固定在 2D 层上，不能随着镜头的移动而有近大远小的变化。第三类方法，在场景中通过对图片的渲染，显示文字。虽不及第一类实现有立体感，不过，因为是在场景之中，也有近大远小的效果，所以比第二类方法更逼真。本文实现的汉字显示算法就是基于第三类解决方法。

### 1.2 目标

鉴于 OGRE 并不提供任何在场景中渲染汉字的解决方案，本文的目标是提供一种在基于 OGRE 的场景中汉字渲染的算法。

① 收稿时间:2010-02-01;收到修改稿时间:2010-03-14

### 1.3 待解决的问题

作为一种主流的开源 3D 通用引擎，OGRE 并不是专为某一类或某一种应用而设计的。出于对大多数项目支持的需要，OGRE 只是专注于对 3D 图形和场景的渲染。所以，它并不特别的支持对于文字的渲染，这就给许多应用 OGRE 的中文项目造成了很大的困难。

### 1.4 基本思路

从 OGRE 的角度看来，场景中的可见的实体都是可渲染对象(Renderable Object)，它对这些各式各样的可渲染对象拥有同一个渲染接口。客观上，可以通过把汉字对象加以改进，让它满足 OGRE 可渲染对象的接口方法，利用 OGRE 自身渲染“可渲染对象”的机制来实现 OGRE 对汉字渲染的支持。

## 2 需求及基本方案

### 2.1 需求

在实际应用中，需要能够和场景内其他物体融为一体的有立体感的汉字；需要有用于用户界面的汉字；需要有用于说明的文字。

### 2.2 三种可能的方案

第一种可能的方案是在场景中使用 OGRE 模型。所谓 OGRE 模型，是指在 OGRE 的场景中，用 OGRE 支持的数据格式存储的用于描述场景中的实体的数据结构[2]。使用 OGRE 模型在 OGRE 场景中渲染文字，首先从字库中读取字体信息，然后根据得到的这些字体信息，使用指定的材料，生成字体模型。

第二种可能的方案是使用 2D 技术实现在 3D 的场景之上的文字渲染。在 2D 的环境下显示文字的主要步骤是从字体库文件中读取字体的信息。然后在内存中建立缓冲区，存储根据字体信息生成的文字点阵数据。最后把文字点阵数据“画”到屏幕上，就完成了文字的显示。

第三种可能的方案是建立在前两种方案的基础上的。首先从字体库文件中读出有关字体的信息，在内存中建立缓冲区，把生成的文字图片存储在缓冲区中。当需要显示的时候，读取相关位置的文字图片信息，同时在场景中创建指示板，将文字图片贴到指示板上显示。

### 2.3 三种方案的对比

第一种方案能提供一种很好的有立体感的汉字。它利用的是 OGRE 中渲染实体的接口，可以把渲染文字的技术无缝的链接到 OGRE 的场景中，并保证和 OGRE 的其他部分协同工作[3]。因为是基于渲染实体的接口来实现的文字渲染，实体所具有的特效，如光照，近大远小，平移，旋转，放缩等，也自动获得了支持。但是，其中关键的步骤，通过字体信息自动的生成模型信息，并没有突破[4]。这种方案实行起来，难度很大。而且生成和维护这样的汉字也是需要很大的时间和空间开销，这种方案只能满足对少量的有立体感的汉字的需求。

第二种方案是在 OGRE 的 2D 场景中用 2D 的技术渲染文字。举例来说，CEGUI 作为通用的 UI 库，就是通过 OGRE 中描述的 2D 的层的接口，嵌入到 OGRE 中的。因此，完全可以在 3D 场景的层上面，再多渲染一个 2D 的层，用来渲染文字。这种方案简单易行。可是，所显示的文字不能随着摄像机(camera)的移动而进行放缩，即不会有近大远小的效果。一些现行的解决方案中，有对这一缺陷的修正，比如出现了根据摄像机镜头的推进和倒退来修正文字的大小。可是结果并不理想，一部分原因是因为在 OGRE 外部加入的算法，不能很好的与 OGRE 协同的工作；另一部分原因是，要对摄像机的运动进行一定的限制，比如运动的范围和速度。这种方案只能满足对用户界面的需求。

第三种方案舍弃了建立模型的步骤，改为使用生成图片的做法。可以实现在 3D 场景中随着摄像机镜头的移动而具有近大远小的效果。缺点是，不能使文字有一个立体的效果。因为从本质上来说，文字的显示放在了 OGRE 的 3D 场景之中，并不是关于文字的模型，只是放置在指示板上的图片。这种方案虽然没有立体效果，不过可以在 3D 的场景中渲染，可以部分的满足有立体感的汉字的需求；这种方案也可以满足对用户界面和说明文字的需求。

所以在应用的需求，效率和渲染效果的综合衡量中，第三种方案是比较有价值，应该重点突破，其中的一些关键技术对最终突破第一种方案是一种有益的尝试。

## 3 设计

### 3.1 字体读取方式

一般有两种方式来读取字体信息。一种是一次加载字体库中所有的信息，另一种是在需要指定的字符的时候才加载。在英文的渲染中多采用前者，但是这

并不适合汉字的渲染。

表 1　汉字字符和英文字符特点的比较

| 语言 | 基本字符数量 | 一次加载需求内存 | 在段落中的重码率 |
|---|---|---|---|
| 中文 | >5000 | 多 | 低 |
| 英文 | <512 | 少 | 高 |

如表 1 所示，英文的渲染之所以可以使用一次性加载字体库中所有信息的方法，是因为英文的字符数量比较少，完全加载的代价非常小；在段落中的重码率很高，生成的图片几乎都会用到。如果对英文采用“即用即加载”的策略，就会有大量的字符需要重复载入，浪费大量的 CPU 时间，损失了效率。所以，一次性载入是适合英文字符的算法。

而中文则恰恰相反，如表 1 所示，虽然中文的字符在不同的标准下有不同的数量，但其数量都远远超过英文。如果使用一次性的加载算法，会比英文的加载多出很多内存容量和 CPU 的加载时间；由于重码率低，这些加载的字符只有很少一部分能被使用，造成了巨大的资源浪费。如果使用随用随加载的算法的话，就可以避免大量的内存和 CPU 时间的浪费；由于重码率低，重复加载的概率也大大降低。所以，对于汉字的加载，使用“即用即加载”的策略更为适合。

### 3.2 矢量字库或点阵字库

汉字的字形信息通常是以矢量字库或者点阵字库的形式存储在外存上。矢量字库中存储的是不同字体文字外部形态的矢量信息，用自动抽取轮廓的方法对点阵信息抽边，形成高离散的轮廓描述。渲染的时候，用拟合的方法对离散轮廓做逼真，形成轮廓的矢量描述图，对初始轮廓做修正[3]。使用矢量的好处是，可以任意的放大汉字，而不用担心汉字会变形，或是变模糊。但是，由于从矢量字库中读取的是字体的笔画信息，因此需要一部分 CPU 时间来生成用于显示的文字图片，并且汉字的缩小会产生问题。汉字字号的缩小要先进行汉字轮廓的缩小，在进行填充，而这势必导致对原来像素点的抽取和合并。常用的汉字有 7000 个，结构复杂不同于西方的字母，矢量字库无法判断抽取和合并的像素点的重要性，进行的是随机的抽取和合并，使那些能够体现汉字信息的关键像素点丢失，在字号较小的情况下，得到的新字号汉字失真将非常严重[4](如图 1 所示)。

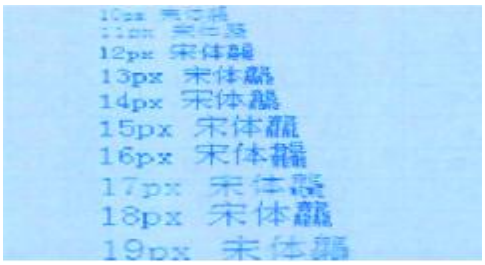


图 1　矢量字体在缩小时的失真[3]

点阵字库是在一张或几张图片上划分许多小的区域，直接通过位置索引文字信息。它的好处是，无须耗费 CPU 时间来生成图片。但是，它需要大量的内存来存储不同大小的文字点阵图片。当图片需要改变大小时可能会出现锯齿或笔画丢失，文字变形等问[4](如图 2 所示)。

图 2　(a)小 5 号宋体 (b)4 号宋体 (c)3 号字体[4]

具体采用矢量字库，还是点阵字库要视具体情况而定。

### 3.3 OGRE 的“可渲染对象”

可渲染对象(Renderable Object)接口是 OGRE 用以渲染实体的唯一接口。它继承自子实体(Sub Entity)，而实体又可以包含一个或多个子实体[1]。这就是说，可渲染对象所描述的，可以是整个实体，也可以是实体的一部分。一个实体就是通过相应的渲染对象来分别描述其各个部分的渲染信息的(如图 3 所示)。

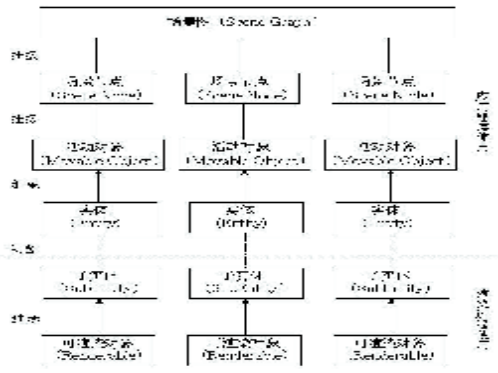

图 3　渲染对象与实体，以及实体和场景的关系

利用可渲染对象的接口，可以实现汉字的渲染。把和汉字渲染相关的信息，用继承自可渲染对象接口的子类封装起来，就可以无缝的连结到 OGRE 的场景之中，挂接到场景节点上了。这样就会使汉字的渲染成为 OGRE 的一个组成部分一样。

### 3.4 汉字的载入

前文已经讨论，汉字的渲染比较适合使用“即用即加载”的策略。每一次载入汉字信息，都只会生成一张图片，这部分原因是由于和 OGRE 无缝连接造成的，生成图片的内存大小由 OGRE 决定。这样，重复加载汉字就会产生多张图片，会浪费内存。因为整张图片的大小已经固定了，由于汉字数量少，会存在大量的内存位置空白。重复加载，加载的数量也可能只是一两个汉字(这是“即用即加载”策略的结果)，也要重新生成一张图片的内存，那么就会产生新的空白。

要解决这个问题就要设法利用空白的内存。由于 OGRE 并不是为渲染汉字而设计，所以在内存中的图片是不整齐的。在默认的情况下，小图片的大小是不同的。由于要显示的汉字的大小和要显示的图片的大小都不相同。在重新载入的时候无法准确的确定在哪个位置可以放入新载入的图片，必须重新载入，这样就浪费了 CPU 的时间。

在一般情况下，由于内存浪费十分巨大，重新载入的过程又是分散的，对 CPU 的占用也是分散的，所以基本可以接受重新载入的方案。只有在 CPU 的时间非常宝贵且内存的浪费可以完全不计较时，才应该放弃“即用即加载”的策略。

### 3.5 可渲染对象子类的封装

继承可渲染对象接口，并在子类中实现文字渲染的细节。首先，无论是从 TTF 文件读取的汉字信息，还是根据现有的文字图片读取的文字信息，最后总要在内存中建立要渲染的汉字的图片信息，对子类的封装正以此作为基础。

确定字体大小，字体颜色等信息，并生成汉字图片。图片被存为 OGRE 下的纹理(texture)。纹理是 OGRE 用来渲染的物体时的“皮肤”，即贴在物体表面的“皮肤”。由于 OGRE 的渲染是通过层进行的，所以应根据需要来确定，是否要将汉字放在首层进行渲染(即文字是否将永远位于其他物体之前)。并确定灯光渲染的属性，以实现汉字在不同的灯光下的不同效果。

在使用的时候，将子类实例化，并将生成的实例挂载到场景中，设定 SetVisible 方法为 true，汉字就可以被渲染了(如图 4 所示)。


图 4　楷体 GB_2312 字在 OGRE 场景中的渲染效果

## 4　总结

本文所讨论的这三种汉字渲染方案各有利弊，如前文所述，第一种方案效果最好，可能的应用很多，开销最大，技术不成熟，需要继续不断摸索，也是汉字在 OGRE 中的渲染这一技术的前进方向。使用 2D 技术的第二种方案，因其技术成熟，已经大量使用。本文重点研究的第三种方案可以在很多的应用中缓解现有的利用 2D 技术渲染汉字的尴尬局面，其中的关键技术，汉字载入技术及其在内存中的优化，索引，和在 OGRE 中的封装对进一步丰富和研究汉字在 OGRE 下的渲染方案有重要的参考价值，为进一步研究积累了经验。

### 参考文献

1 Junker G. Pro OGRE 3D Programming. 901 Grayson Street Suite 204 Berkely, CA USA; Apress，2006.

2 Rodrigues M, Robinson A, Alboul L, Brink W. 3D modelling and recognition. Proc. of the 6th WSEAS International Conference. Windsor, Ontario, Canada：World Scientific and Engineering Academy and Society, 2006:72－76.

3 Pollefeys M, Van Gool L. From images to 3D models. ACM Press, 2002,45(7):50－55.

4 毕晓君,韩仲年.汉字字形及汉字库存储技术研究.计算机应用, 2007,26(12):95－98.