阅读新闻

背景：

2D游戏图像数据传输的优化技巧

[日期：2006-09-13] 来源：云风工作室云风程序人生 作者：Analyst [字体：大 中 小]

2D游戏图像数据传输的优化技巧

目前，限制2D游戏速度的瓶颈主要是大量的在内存/显存中传输图像数据所造成的。一帧800*600*16bit的图像数据量差不多有1M，要不断的在内存/显存间传输，一秒钟要处理的数据量是相当大的。一般，我们为了使游戏更加丰富多彩，总希望游戏有更高的分辨率、颜色数以及更多的卷轴、精灵等等，这些无不是以增大图像数据传输量为代价的。单纯的靠提高位图传输速度来解决显然不是办法，因为传输速度是由硬件的传输带宽决定的，无论你怎么优化都不会有很大的提高。另外，尽管显存之间的传输带宽非常大，但是对于软件渲染引擎来说又用不上，也是行不通的。看来我们只有从算法和总体设计上另想办法。

从2D游戏画面产生的特点我们可以看出，无论什么2D游戏，相邻的两帧画面总有一定的连续性，换句话说相邻两帧画面的改变量并会很多。因此在生成每一帧游戏图像的时候，我们没有必要重画全部的画面，而只需要重画那些改变的部分就可以了，这样子就可以节省下大量的数据传输带宽了。另外，对于有多层画面的地方，下面被遮挡部分的画面完全没有必要绘制，我们也可以在这个方面想办法优化。

对于前者的优化，我们可以采用脏矩形(dirty rect)算法，即对每处画面将有改变（例如，一个精灵需要重绘）的地方定义一个脏矩形，每次绘制的时候只对脏矩形内的画面进行绘制。那么对于有地图卷轴的情况又该如何处理呢？这个时候我们需要定义一个比屏幕更大的帧缓冲区，只要改变帧缓冲与屏幕的对应关系，就可以实现地图卷轴，这样就不需要在每次地图卷动的时候重绘整个地图了。

对于后者的优化则并不容易，在3D中我们可以使用Z-Buffer避免重绘那些被遮挡的像素点，在2D使用Z-Buffer则显然得不偿失。我们还是回到使用矩形裁减的方法，对于一般的对话框这类正规矩形的界面元素直接定义裁减矩形就可以了，但问题是一般的精灵都不会是矩形的，该如何解决这个问题呢？ 我们可以使用程序在精灵内部计算生成出一系列裁减矩形列表，这些裁减矩形将把前面绘制的被遮挡部分裁减掉，使其不被绘制。但是要注意如果裁减矩形太小，花在裁减上的时间就可能超过绘制的时间，那就得不偿失了。因此，我们还应该规定一个裁减矩形的最小面积，小于这个面积的矩形将不被添加到列表中。当然，如果觉得生成列表太麻烦，也可以只保留最大的那个裁减矩形。另外，对于半透明的图面则不能生成裁减矩形。

好了，说了那么多，现在让我们来整理一下这个算法的具体过程。

第一步：生成所有图面的裁减矩形列表，这一步是程序预计算的。

第二步：在每帧游戏渲染的过程中，记录下所有游戏渲染的图面以及脏矩形，在更新屏幕时一次性渲染到帧缓冲并输出到屏幕。这就是异步渲染机制。

第三步：对每个脏矩形进行渲染。

第四步：每个脏矩形的渲染过程为：每个要渲染的图面将按画家算法从前到后渲染，将每个图面的裁减矩形列表也按这个绘制顺序排列生成一个总的裁减矩形列表，将裁减矩形列表作用于每个要渲染的图面上，将被遮挡部分裁减掉，每渲染完一个图面就将这个图面的裁减矩形在列表中删除，并进行下一个图面的渲染工作直到完成渲染。

使用这个算法还要注意矩形间的合并，另外还要针对游戏的具体情况来做权衡，不然很有可能会得不偿失。

录入：zhaoruibin

上一篇：[图文]真彩色转成高彩色快速算法
下一篇：使用Mobile 3D API进行高级游戏开发

相关新闻

本文评论 全部评论

发表评论

点评： 字数
姓名： 发 表

- 尊重网上道德，遵守中华人民共和国的各项有关法律法规
- 承担一切因您的行为而直接或间接导致的民事或刑事法律责任
- 本站管理人员有权保留或删除其管辖留言中的任意内容
- 本站有权在网站内转载或引用您的评论
- 参与本评论即表明您已经阅读并接受上述条款