结语

原文	**Final thoughts (https://learnopengl.com/#!In-Practice/2D-Game/Final-thoughts)**
作者	JoeydeVries
翻译	包纸 (https://github.com/ShirokoSama)
校对	暂无

> **Note**
>
> 本节暂未进行完全的重写，错误可能会很多。如果可能的话，请对照原文进行阅读。如果有报告本节的错误，将会延迟至重写之后进行处理。

与仅仅是用OpenGL创建一个技术演示相比，这一整章的教程给我们了一次体验在此之上的更多内容的机会。我们从零开始制作了一个2D游戏，并学习了如何对特定的底层图形学概念进行抽象、使用基础的碰撞检测技术、创建粒子、展示基于正射投影矩阵的场景。所有的这些都使用了之前教程中讨论过的概念。我们并没有真正地学习和使用OpenGL中新的、令人兴奋的图形技术，更多的是在将所有知识整合至一个更大的整体中。

Breakout这样的一个简单游戏的制作可以被数千种方法完成，而我们的做法也只是其中之一。随着游戏越来越庞大，你开始应用的抽象思想与设计模式就会越多。如果希望进行更深入的学习与阅读，你可以在game programming patterns (http://gameprogrammingpatterns.com/)找到大部分的抽象思想与设计模式。（译注：《游戏编程模式》一书国内已有中文翻译版，GPP翻译组译，人民邮电出版社）

请记住，编写出一个有着非常干净、考虑周全的代码的游戏是一件很困难的任务（几乎不可能）。你只需要在编写游戏时使用在当时你认为正确的方法。随着你对视频游戏开发的实践越来越多，你学习的新的、更好地解决问题的方法就越多。不必因为编写“完美”代码的困难感到挫败，坚持编程吧！

优化

这些教程的内容和目前已完成的游戏代码的关注点都在于如何尽可能简单地阐述概念，而没有深入地优化细节。因此，很多性能相关的考虑都被忽略了。为了在游戏的帧率开始下降时可以提高性能，我们将列出一些现代的2D OpenGL游戏中常见的改进方案。

- **渲染精灵表单/纹理图谱(Sprite sheet / Texture atlas)**：代替使用单个渲染精灵渲染单个纹理的渲染方式，我们将所有需要用到的纹理组合到单个大纹理中（如同位图字体），并用纹理坐标来选择合适的精灵与纹理。切换纹理状态是非常昂贵的操作，而使用这种方法让我们几乎可以不用在纹理间进行切换。除此之外，这样做还可以让GPU更有效率地缓存纹理，获得更快的查找速度。（译注：cache的局部性原理）
- **实例化渲染**：代替一次只渲染一个四边形的渲染方式，我们可以将想要渲染的所有四边形批量化，并使用实例化渲染 (../../../04 Advanced OpenGL/10 Instancing/)在一次<>`draw call`中成批地渲染四边形。这很容易实现，因为每个精灵都由相同的顶点组成，不同之处只有一个模型矩阵(Model Matrix)，我们可以很容易地将其包含在一个实例化数组中。这样可以使OpenGL每帧渲染更多的精灵。实例化渲染也可以用来渲染粒子和字符字形。
- **三角形带(Triangle Strips)**：代替每次渲染两个三角形的渲染方式，我们可以用OpenGL的`TRIANGLE_STRIP`渲染图元渲染它们，只需4个顶点而非6个。这节约了三分之一需要传递给GPU的数据量。
- **空间划分(Space partition)算法**：当检查可能发生的碰撞时，我们将小球与当前关卡中的每一个砖块进行比较，这么做有些浪费CPU资源，因为我们可以很容易得知在这一帧中，大多数砖块都不会与小球很接近。使用BSP，八叉树(Octress)或k-d(imension)树等空间划分算法，我们可以将可见的空间划分成许多较小的区域，并判断小球是否在这个区域中，从而为我们省去大量的碰撞检查。对于Breakout这样的简单游戏来说，这可能是过度的，但对于有着更复杂的碰撞检测算法的复杂游戏，这些算法可以显著地提高性能。
- **最小化状态间的转换**：状态间的变化（如绑定纹理或切换着色器）在OpenGL中非常昂贵，因此你需要避免大量的状态变化。一种最小化状态间变化的方法是创建自己的状态管理器来存储OpenGL状态的当前值（比如绑定了哪个纹理），并且只在需要改变时进行切换，这可以避免不必要的状态变化。另外一种方式是基于状态切换对所有需要渲染的物体进行排序。首先渲染使用着色器A的所有对象，然后渲染使用着色器B的所有对象，以此类推。当然这可以扩展到着色器、纹理绑定、帧缓冲切换等。

这些应该可以给你一些关于，我们可以用什么样的的高级技巧进一步提高2D游戏性能地提示。这也让你感受到了OpenGL的强大功能。通过亲手完成大部分的渲染，我们对整个渲染过程有了完整的掌握，从而可以实现对过程的优化。如果你对Breakout的性能并不满意，你可以把这些当做练习。

开始创作！

你已经看到了如何在OpenGL中创建一个简单的游戏，现在轮到你来创作属于自己的渲染/游戏程序了。到目前为止我们讨论的许多技术都可以应用于大部分2D游戏中，如渲染精灵、基础的碰撞检测、后期处理、文本渲染和粒子系统。现在你可以将这些技术以你认为合理的方式进行组合与修改，并开发你自己的手制

游戏吧！