

使用英特尔 GPA 优化《轩辕传奇》游戏的性能

英特尔：卢卷彬 Kiefer Kuah 腾讯：丛越

介绍

随着集成显卡功能和性能的日益增强以及移动平台的不断普及，集成显卡已经成了游戏开发者不可忽视的重要对象。

英特尔的图形性能分析器（GPA）是一套非常优秀专业的性能分析工具，它可以极方便的帮助游戏开发人员分析，评估和改进游戏在英特尔集成显卡上的性能。GPA 工具的图形性能分析部分主要由 HUD 系统分析器和帧分析器两个工具组成，HUD 系统分析器从平台系统的角度提供实时监控，帧分析器从单帧渲染的角度提供更详细全面的离线分析，可以让使用者十分方便的分析出性能开销和瓶颈。

本文介绍了使用 GPA 在英特尔集成显卡上分析和优化《轩辕传奇》网络游戏的案例。《轩辕传奇》是由腾讯研发团队打造的一款 MMORPG 网游，是腾讯的首款史诗战争网游。该游戏在引擎技术、美术、服务器等诸多方面都力求精益求精，达到了国内顶级网游水准。为了满足更多玩家的机器配置，我们特别针对集成显卡进行了测试和优化。

性能分析开始

本文档的分析和优化基于 Intel HD Graphics 3000 平台，代号 SandyBridge(简称 SNB)。SNB 是 Intel 在 2011 年推出的一款性能非常出色的处理器集成显卡。它的性能可以媲美一些独立显卡。作为一款网络游戏，《轩辕传奇》的开发者希望能够在更多的平台上面流畅运行游戏，所以尽管游戏在 SNB 上的性能已经很流畅了，我们还是要尽量的对游戏性能进行优化。以下是我们这次性能分析和优化所选择的目标场景：



图 1. 《轩辕传奇》性能分析和优化的目标场景

系统分析

GPA HUD 能够实时的显示游戏运行的时候 CPU, DX runtime, 以及 GPU 上的性能数据。同时支持多种 D3D 流水线上的 override 模式, 帮助游戏开发者进一步定位游戏的瓶颈所在。

通过 GPA HUD 的分析, 我们发现:

1. 游戏的大多数计算都是在主线程中完成, 多核心利用率比较低。那么在玩家角色多的时候, 因为动画计算比较多, CPU 将会成为瓶颈。可以考虑使用多线程来消除这种场景下的瓶颈。
2. 游戏中每帧的 State Changes 数量不算很多, 大概每帧 6000 个, 平均每个 DP call 7.5 次。能够做一些优化减少 State Changes 数量的话, 对性能是很有好处的。
3. 通过 Override 模式测试, Null Driver, Null Hardware 都可以使游戏的 fps 达到 100 左右, 100fps 是游戏的限帧。所以在该场景下游戏的瓶颈是在 GPU 上。我们还在玩家角色比较多的场景进行了类似测试, 发现 Null Hardware 对 fps 提升不大, 此时瓶颈在 CPU 上。

帧分析

GPA 帧分析器能够详细的分析游戏的一帧的所有 draw call 的性能数据以及它们所使用的纹理, d3d 状态, shader 等等。图 2 显示了我们使用 GPA 帧分析器打开我们在目标场景抓取的一帧数据。

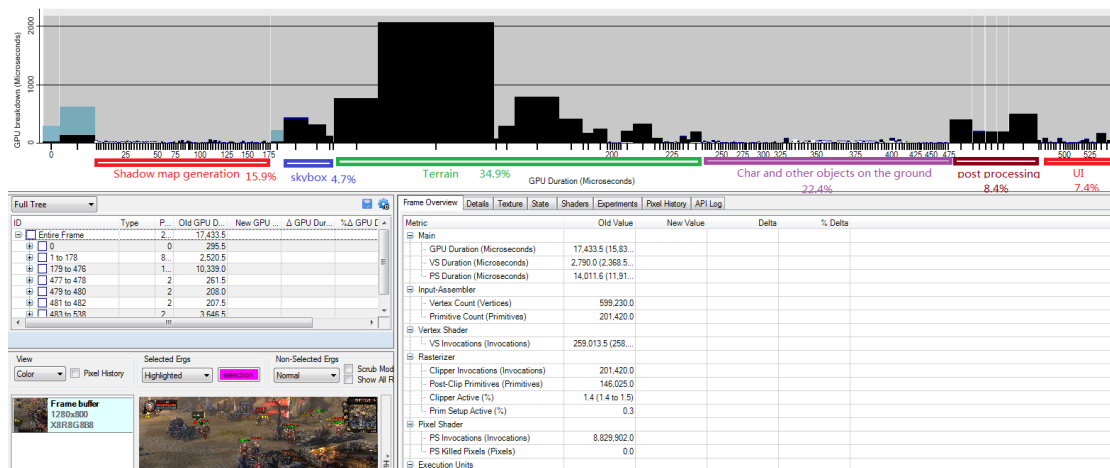


图 2. 使用 GPA 帧分析器打开抓取的一帧数据

通过帧分析器分析, 我们得到整个一帧的时间分布为:

1. 生成阴影贴图: ~170 个 DP Calls, 15.9% frame time。场景中所有的物体都计算了实时阴影。我们可以使用静态阴影来替代其中的一部分, 比如某些距离远的, 固定的物体。
2. 天空盒: 3 个 DP Calls, 4.7% frame time。实际上在该场景中看不到天空, 我们可以在渲染天空盒之前进行可见性检测。如果不可见, 则不渲染。
3. Terrain: 50 个 DP Calls, 34.9% frame time。和大多数网络游戏一样, 地形总是最耗时的部分, 是优化的重点。
4. 人物和其他地上物件: ~240 个 DP Calls, 22.4% frame time。没有发现明显的可以优化的地方。
5. 后处理: 8 个 DP Calls, 8.4% frame time。没有发现明显的可以优化的地方。
6. UI: 55 DP Calls, 7.4% frame time。有许多 UI 元素过小, 可以合并一下再渲染。UI 并不是每帧都变化的, 可以重复利用上一帧的结果。

优化策略和结果:

1. 阴影贴图生成

游戏花费了~170 个 draw calls 来生成所有物体的阴影贴图，但实际上，场景中某些固定的物体或者远处的物体，是没有必要使用实时计算的阴影的，使用提前生成的静态阴影即可。这可以节省 draw calls 的数量，提高性能。图 3 是优化前后的对比，右边是优化后的阴影贴图生成，减少了使用实时阴影的物体数量，可以看到 draw calls 数量减少到了 45 个，整个阴影地图生成花费的时间从 2.97ms 减少到了 1.03ms。

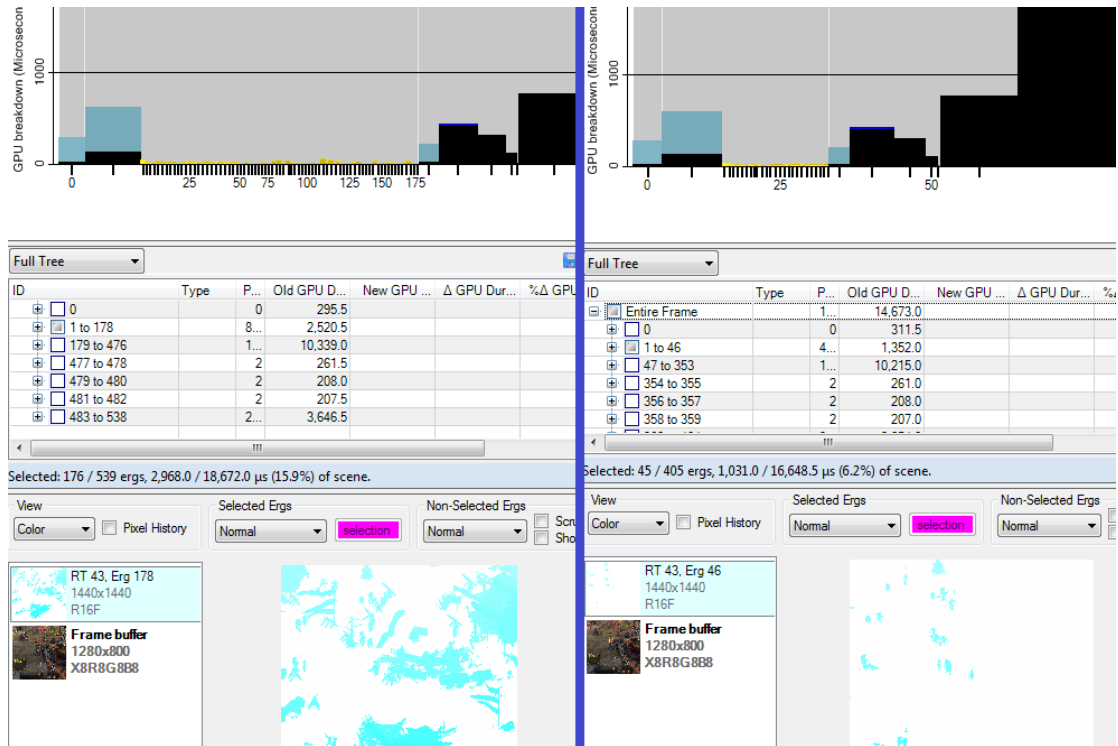


图 3. 优化前和优化后的阴影贴图计算

2. 天空盒

天空盒在这个场景中是不可见的，但是帧分析器显示游戏依然渲染了它。我们可以在渲染之前检测天空盒的可见性，如果不可见，则不渲染。这可以节省 0.88ms。而且，天空盒最好是放到场景中不透明物体渲染完之后再渲染，因为天空盒的大部分都是被遮挡的。

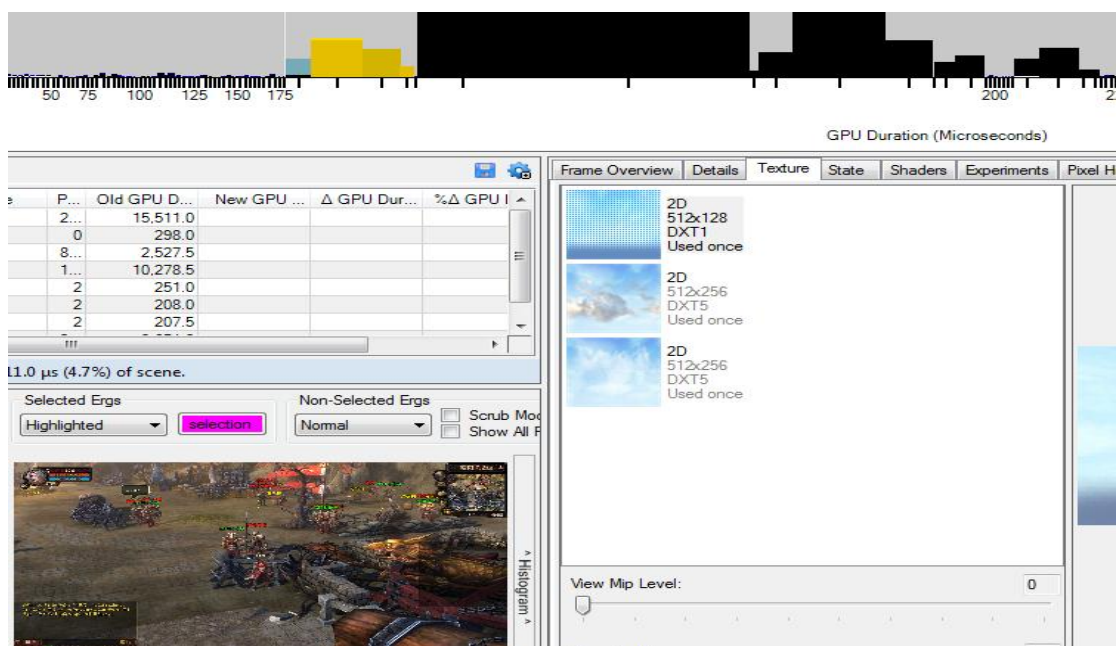


图 4. 游戏渲染了不可见的天空盒

3. 地形

地形部分的渲染占用了最多的时间，网络游戏一般都是这样。

其中一部分主要的地形，占用了 6.5ms 中的 5.1ms，它们使用的同一段 Pixel Shader，以下是这部分地形的截屏：

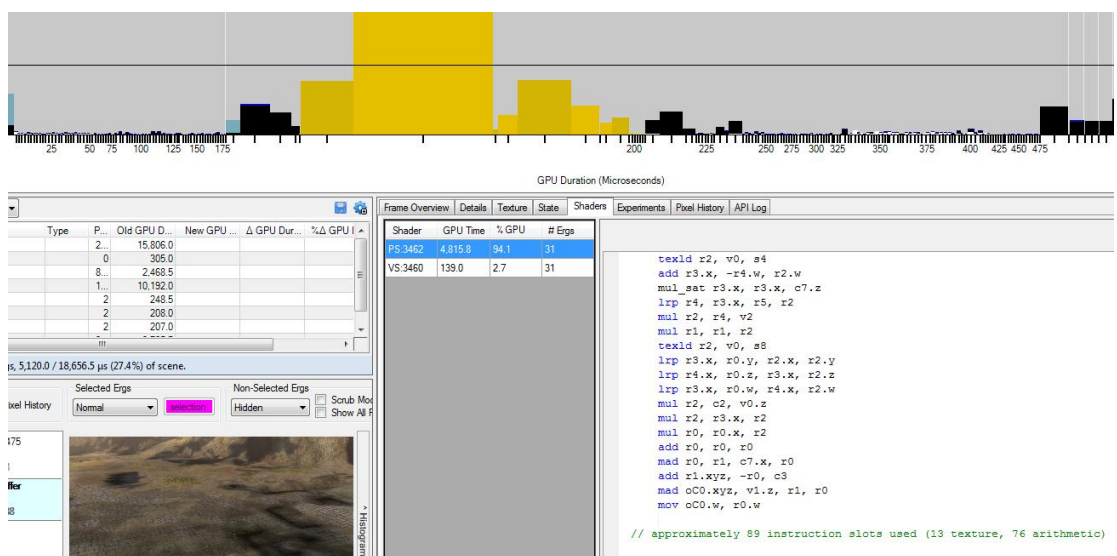


图 5. 部分地形使用同样的 Shader，包含 89 个指令，其中 13 个纹理 load，76 个算术指令

通过帧分析器我们可以看到，这部分地形的渲染，PS duration 占用了 90%的时间，在帧分析器中我们可以在 Shader Tab 查看这些渲染所使用的 shader code，如图 5 所示，这部分地形的 shader 包含了 89 条指令，其中 13 个纹理 load，76 个数学计算，比较复杂。

我们把 shader 做了一定的简化，去掉了高光，normal map，AO map 等。图 6 是修改了 pixel shader 后的截图，修改后 pixel shader 包含 31 条指令，其中 8 个纹理 load，23 个算术指令。这部分地形渲染的时间从 5.1ms 减少到了 3.4ms。

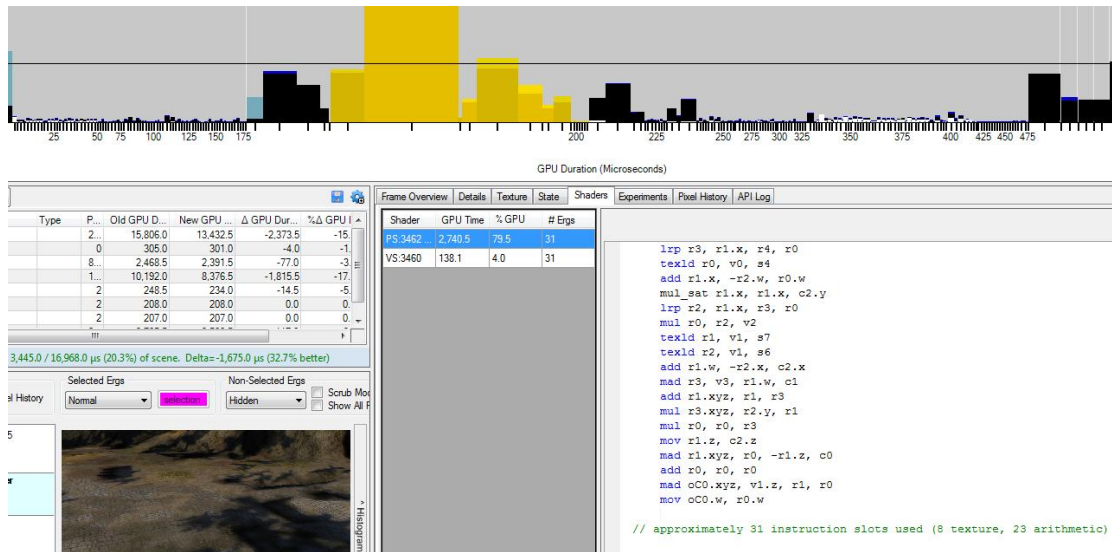


图 6. 优化 shader 代码后，这部分地形所花费的时间减少到了 3.4ms

我们还为特别低端的平台准备了更加简单的，只需要 2 个纹理的 shader 版本，时间进一步减少到了 0.93ms，见图 7 所示：

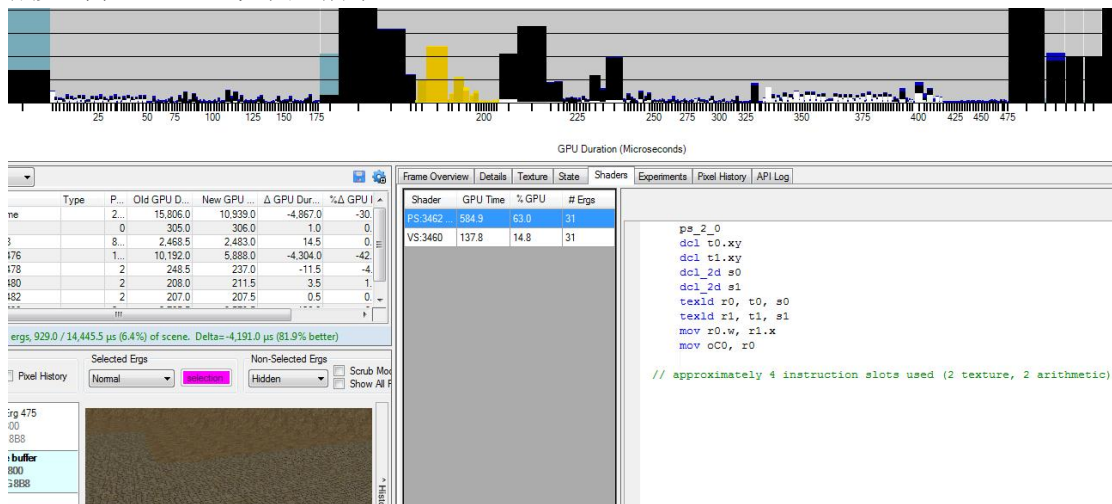


图 7. 在更低端平台上进一步简化地形的 pixel shader

当然，使用 2 个纹理后，地形画面质量肯定有所下降。在 GPA 帧分析器我们能够实时的看到修改 shader code 后对画面的影响。

4. UI

UI 部分共占用 7.4%的时间，图 8 是游戏中左上角的人物 UI 截屏。



图 8. 游戏中左上角的角色 UI

通过帧分析器，我们发现为了画游戏左上角的人物头像以及血槽等部分，游戏使用了 9 个 draw calls 来完成（图 9），而且是每一帧都会花费 9 个 draw calls 来做这件事情。但是对网络游戏来说，99%的时间，这部分是不会变化的，那么每帧都使用 9 个 draw call 来画这部分，是没有必要的。我们可以把这个头像部分先渲染到一个 image buffer，然后使用一个 draw call 把这个 image 画到屏幕上，如果下一帧这个头像部分没有更新，则继续使用上次的这个 image buffer。这样一来，只有在头像部分需要更新的时候，才去重新做这 9 次 draw calls，其他时候，只需要一个 draw call 就完成了。

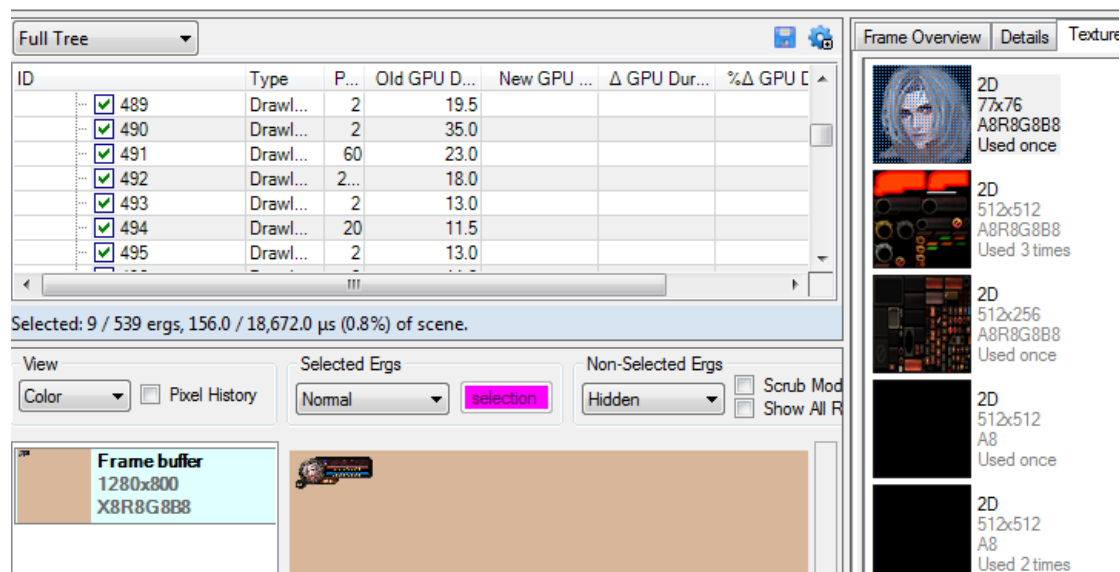


图 9. 游戏使用了 9 个 draw calls 来完成这部分 UI 的渲染

图 10 显示了优化后的截图，这一帧重利用了 image buffer，只使用了一个 draw calls 就完成了这部分 UI 的渲染。我们对其它的 UI 也进行了类似的优化。优化后，渲染 UI 所花费的时间从 1.38ms 减少到了 1.17ms。

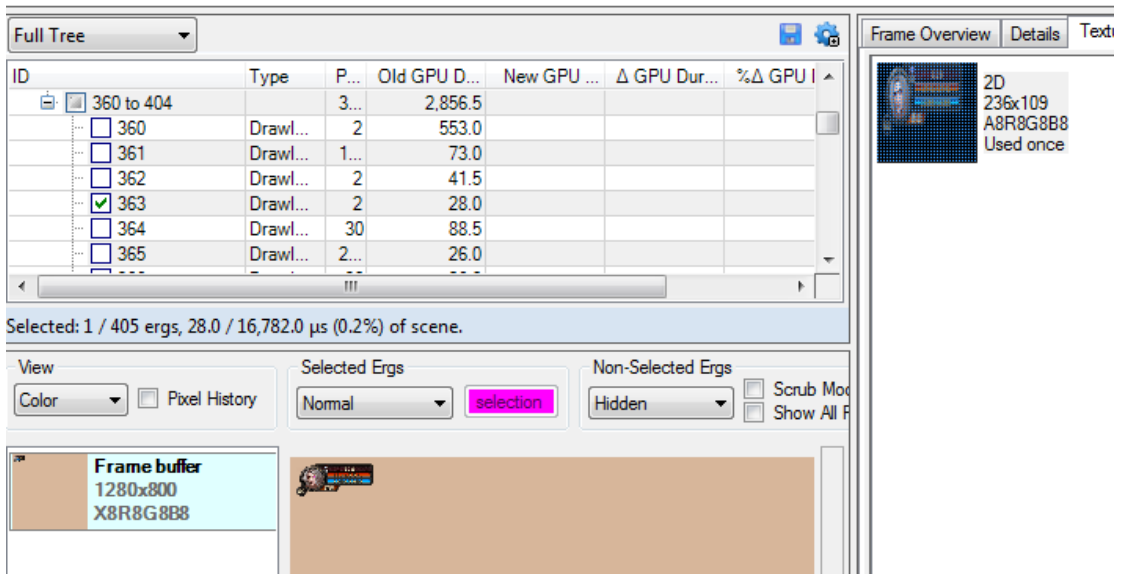


图 10. 使用 image buffer 来存储这部分 UI 以便重复使用

5. 减少 State Changes 数量

State Changes 过多会导致游戏程序, d3d runtime, 显卡驱动和显卡硬件的负载增加。减少 State Changes 的数量能够提高游戏性能。通过帧分析器对这一帧的详细分析, 我们发现有很多的 draw calls, 都会使用很小的纹理, 16*16, 32*32 或者 64*64 大小, 大量的使用小纹理, 不是一种有效率的做法。图 11 显示了两个相邻的 draw calls 使用了非常小的纹理:

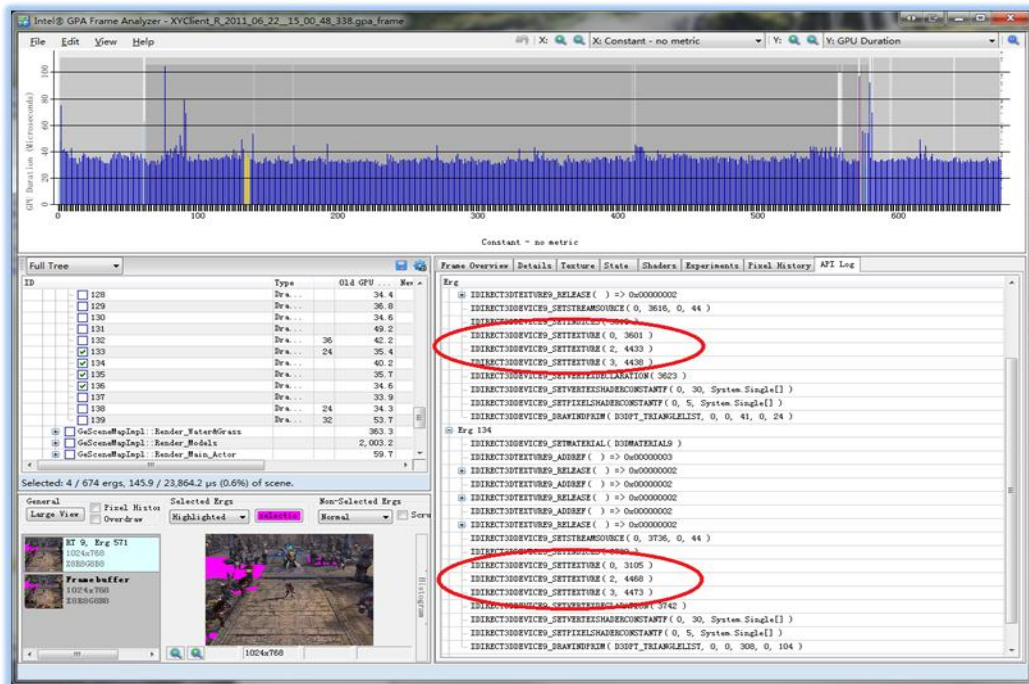


图 11. 两个相邻的 draw calls, 都使用了非常小的贴图

从上面图 11 中我们看到, 临近的 2 个 draw calls, 分别调用了 3 个 SETTEXTURE API, 其中第二和第三个 texture, 都是很小的纹理 (见图 12)。我们可以合并这些小的纹理, 然后只需要设置一次合并后的纹理, 接下来的几个 draw calls 就可以直接使用, 而不用再设置了。这可

以减少 State Changes 的数量。当然，使用合并纹理的时候要注意设置正确的 UV 值。

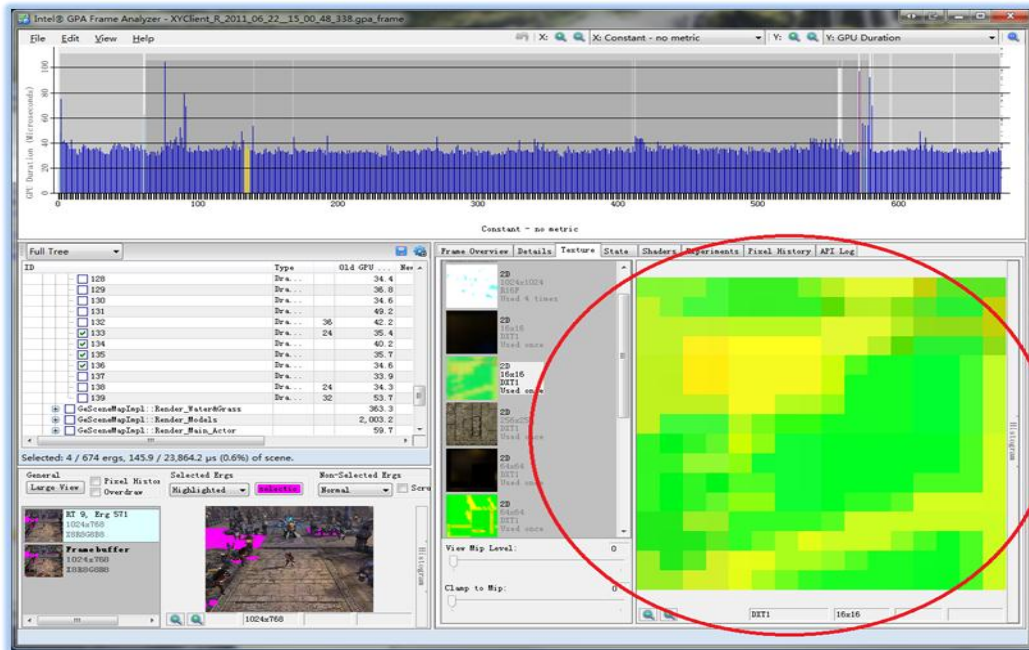


图 12. 这些 draw calls 所使用的小尺寸纹理

图 13 和 14 是优化后的截屏，我们合并了 3 个 draw calls 所使用的小纹理，SETTEXTURE API 调用的次数从 9 次减少到了 4 次。整个一帧，我们减少了~500 次 State Changes。

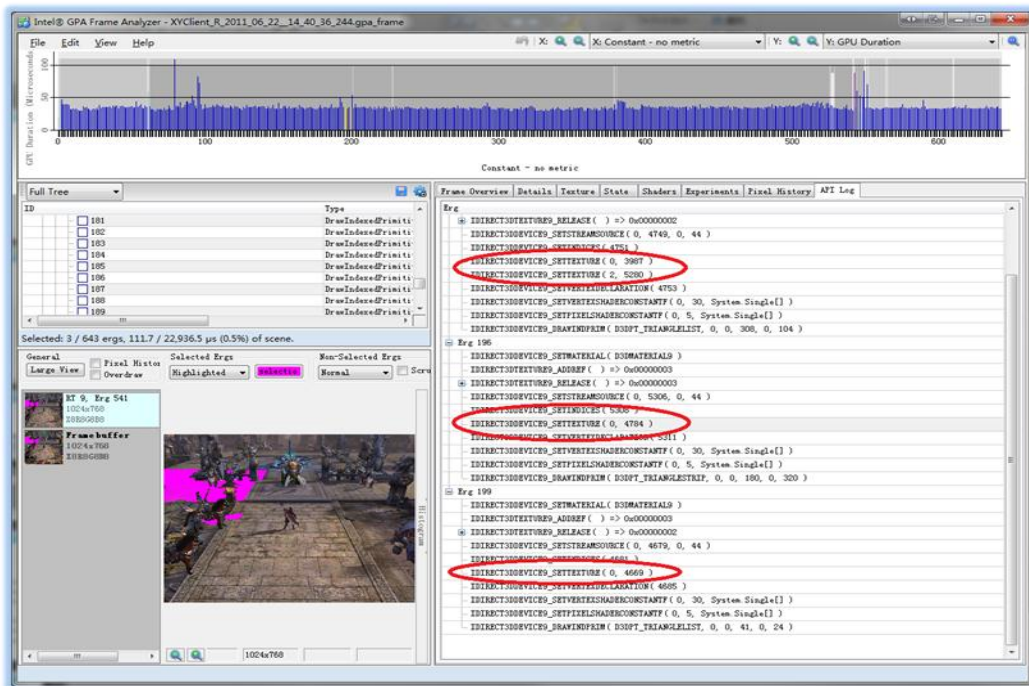


图 13. 合并纹理后，3 个 draw calls 所调用的 SETTEXTURE API 从 9 次减少到了 4 次

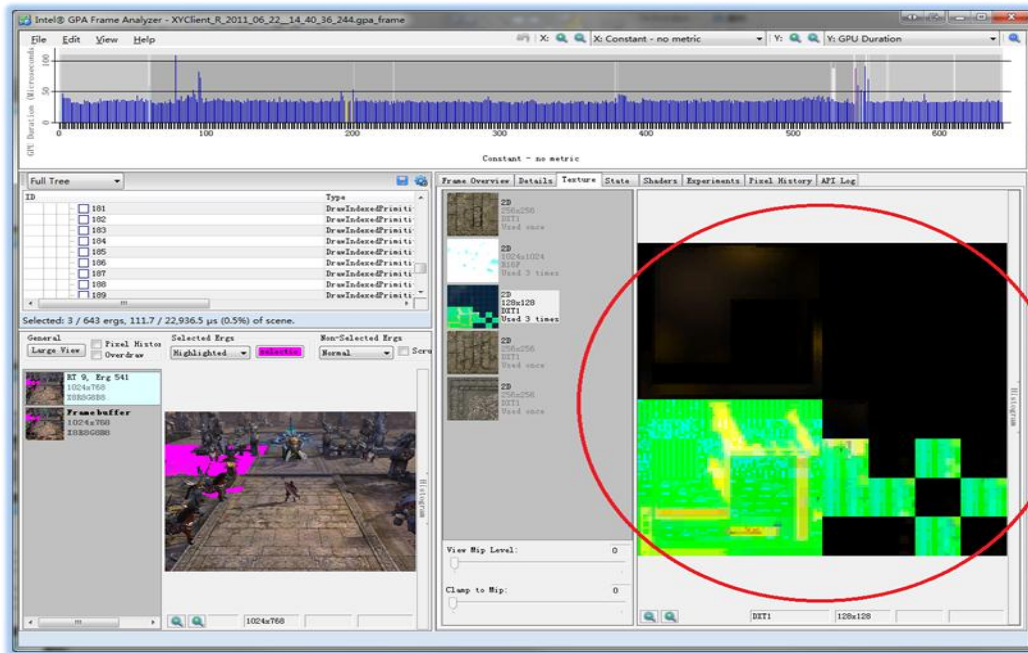


图 14. 将小纹理合并后得到的大纹理

优化总结：

我们介绍了在英特尔集成显卡平台上通过 GPA 对游戏性能进行分析和优化的成功案例。GPA 工具帮助我们找到了游戏的瓶颈所在。我们使用它对一帧进行了深入的分析，对 shader 进行了实时修改，并找到了减少 State changes 数量的办法。表 1 是我们所完成的优化的总结。经过优化，游戏性能得到了很大提升，能让更多玩家流畅体验轩辕传奇这款游戏。其实我们还使用 GPA 对其他不同场景，不同状态（聊天，骑乘，战斗，换装，技能特效等），不同平台（各个厂商的独立显卡，集成显卡以及不同型号的 CPU），不同系统进行了各种测试，限于篇幅这里只挑出了一个有代表性的例子。

优化项目	优化方法	优化前	优化后	提升
阴影贴图生成	对某些不重要或者远处的物体采用静态阴影	2.97ms	1.03ms	1.94ms
天空盒	如果 skybox 不可见，则不渲染	0.88ms	0.0ms	0.88ms
地形	将纹理数量从 13 减少到 8，并简化 pixel shader	5.1ms	3.4ms	1.7ms
UI	合并 UI，重利用	1.38ms	1.175ms	0.205ms
State changes	合并小纹理	~6000	~5500	减少 ~500 state changes

作者简介:

卢卷彬 是英特尔公司的应用工程师，他和国内几家大的游戏公司有着多年的合作，帮助他们在英特尔平台上优化游戏客户端的性能。

丛越 有 8 年以上的游戏开发经验，他参加了中国最早期的 3D MMORPG 的开发，有着丰富的经验。丛越现在就职于腾讯，主要负责游戏和引擎的开发。

Kiefer Kuah 是英特尔的软件工程师，他主要负责游戏在英特尔平台上的优化工作。