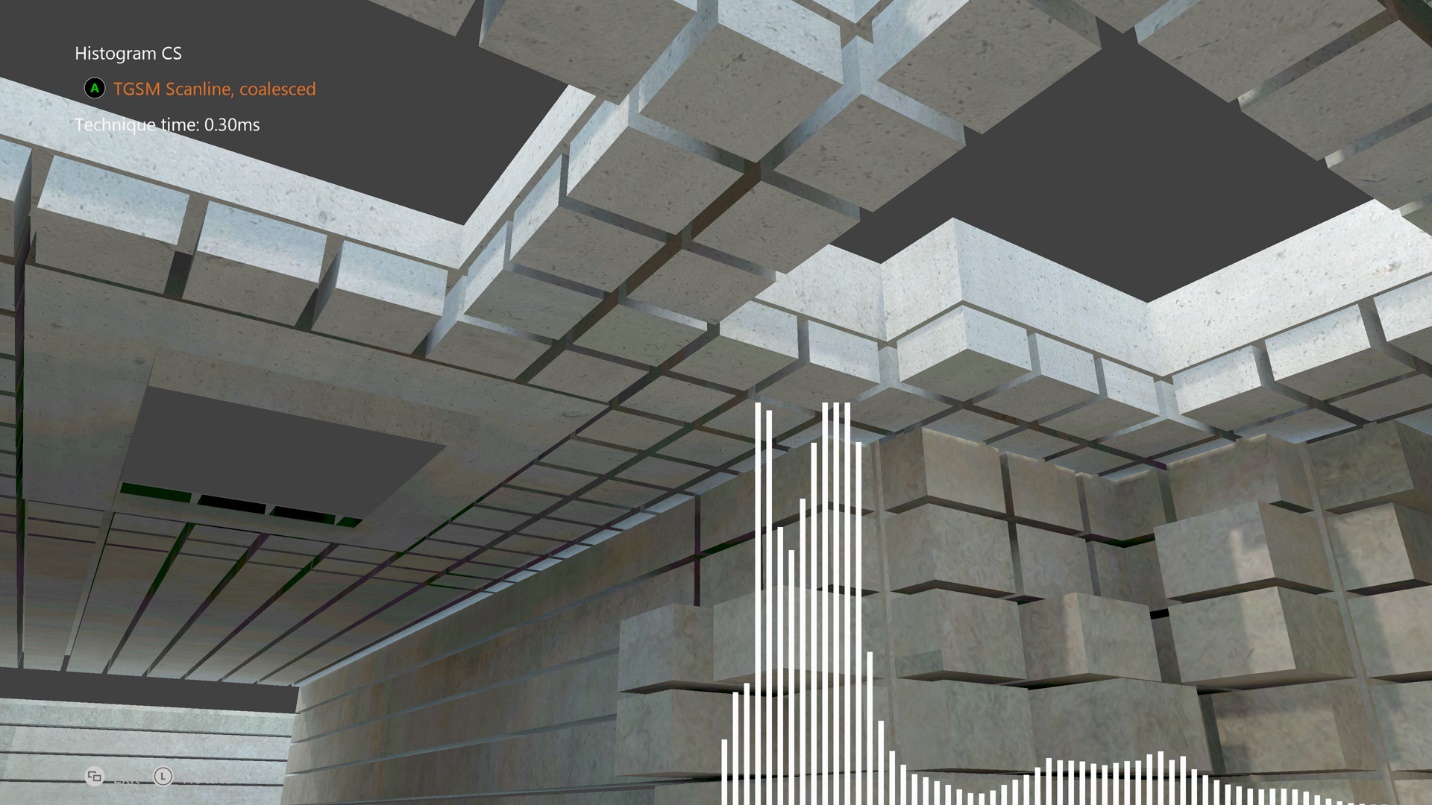


直方图 CS 示例

*此示例兼容于 Microsoft 游戏开发工具包（2020 年 8 月）*

# 说明

# 此示例演示了在 Xbox 上计算着色器的一些性能注意事项。本质上，它演示了一种计算场景强度的 64 存储桶直方图的最佳方法，这样做（使用最佳实现）只需不到 0.2 毫秒。



# 生成示例

如果使用 Xbox One 开发工具包，请将活动解决方案平台设置为 Gaming.Xbox.XboxOne.x64。

如果使用 Xbox Series X|S 开发工具包，请将活动解决方案平台设置为 Gaming.Xbox.Scarlett.x64。

*有关详细信息，请参阅 GDK 文档中的*运行示例*。*

# 使用示例

|  |  |
| --- | --- |
| 操作 | 游戏手柄 |
| 循环直方图技术 | A 按钮 |
| 旋转视图 | 左控制杆 |
| 重置视图 | 左控制杆（单击） |
| 退出 | “视图”按钮 |

# 实现说明

# 借助 CPU 编程，在跨多个核心分散工作时，我们通常会尝试让每个核心获取一个连续的数据块，以提高缓存利用率，从而提高性能。在 GPU 上实现算法时，很容易以相同的方式进行思考，因为在 HLSL 中，每个着色器程序似乎都在自己的空间中运行（类似于 CPU 上的线程）。但是，由于新式 GPU 的高度并行特性，情况远非如此。

# Xbox One GPU 实际上是在成为波的线程组上工作的，最宽可达 64。这意味着每个线程都是一个更大的并行线程组的一部分，这些线程在 lockstep 中运行，它们都执行完全相同的指令。可以将此视为类似于 CPU 上设置的 64 浮点宽 SIMD 指令。

# 结果是，让每个单独的线程提取相邻的数据最终并不是最佳的。。理想的情况是合并读取 - 即相邻的线程读取相邻的数据。实际上，对于循环的每次迭代，线程组中的每个线程读取第 64 项。从示例中可以看出，此操作会取得更好的结果。

# 但是，读取顺序并不是此示例中的唯一阻碍因素。直方图是通过对直方图的每个存储桶执行原子添加生成的（总共有 64 个存储桶）。Xbox One GPU 执行这些操作的速度非常快（它们实际上是在 L2 缓存中执行的），但它仍然构成了一个限制因素。为了加快此运行的速度，我们可以利用极快的通过 HLSL 中组共享内存公开的内部 LDS 缓存。通过让每个线程在整个扫描线上运行，在 LDS 中生成结果，然后在末尾以原子方式添加最终结果。与主内存（或 L2）相比，LDS 中的原子操作要快得多，因此会显著提高性能。

# 有关此技术和其他内容的详细信息，请务必查看 Xfest 2013 中的计算着色器讲座。

# 已知问题

无。

# 更新历史记录

示例的原始版本是使用基于 XSF 的框架编写的。2020 年 6 月，示例被改写为使用 ATG 示例模板。

# 隐私声明

在编译和运行示例时，将向 Microsoft 发送示例可执行文件的文件名以帮助跟踪示例使用情况。若要选择退出此数据收集，你可以删除 Main.cpp 中标记为“示例使用遥测”的代码块。

有关 Microsoft 的一般隐私策略的详细信息，请参阅 [Microsoft 隐私声明](https://privacy.microsoft.com/en-us/privacystatement/)。