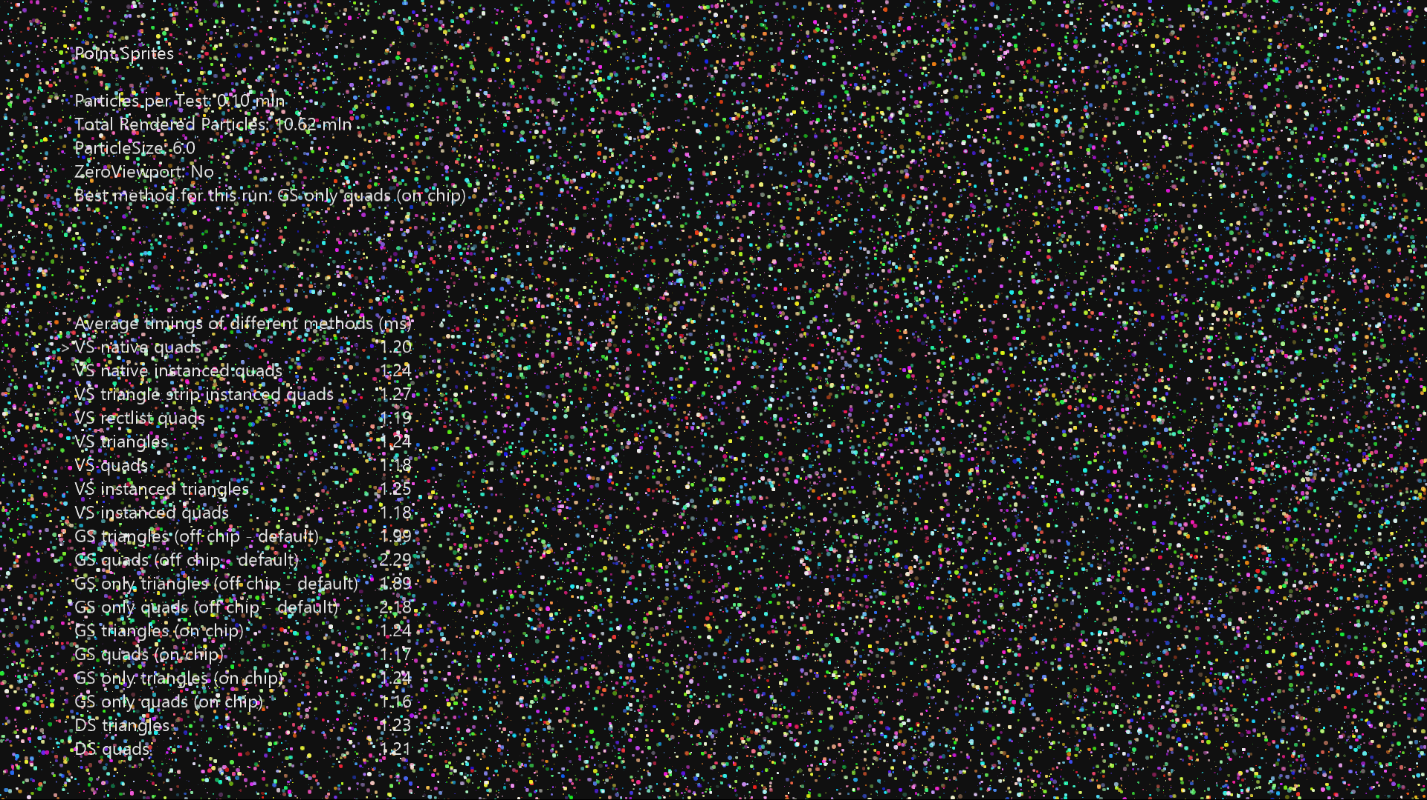


点精灵

此示例与 Microsoft 游戏开发工具包预览版（2019 年 11 月）兼容

# 说明

这个示例演示了在 DirectX 12 中呈现点精灵的十种方法。



# 构建示例

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。

有关详细信息，请参阅 GDK 文档中的“运行示例”。

# 使用示例

此示例使用以下控制。

|  |  |
| --- | --- |
| 操作 | 游戏手柄 |
| 退出示例。 | “视图”按钮 |
| 增加精灵大小 | A 按钮 |
| 减少精灵大小 | B 按钮 |
| 切换 Null 视区 | Y 按钮 |
| 更改所选测试 | 向上/向下方向键 |
| 切换突出显示的测试 | X 按钮 |

# 实现说明

随着图形管道的发展，已取消对点精灵提供一流支持，转而使用完全灵活且可自定义的阶段，如几何着色器。实际上，点精灵可以在 GPU 的每个与几何相关的阶段（顶点、几何和分割着色器）中实现。虽然该方法的理想选择应该基于测量逐个案例的性能，但似乎使用几何着色器来扩展点精灵并不是呈现简单点精灵的最有效方法。

## 三角形与四边形

根据呈现管道遇到瓶颈的位置，三角形或四边形可能具有不同的性能。在填充速率受限的情况下，四边形比三角形有性能优势，因为它们不需要在透明像素上运行像素着色器。当呈现绑定到几何阶段时，输出三角形（而非四边形）很有意义。在许多复杂的呈现情况下，三角形的呈现速度可能比四边形更快。

该示例演示了将三角形和四边形进行对比的每种方法，以显示不同大小的点精灵的差异。

## 方法 1 - 顶点 + 几何着色器

初始化管道，以便顶点着色器正常读取顶点，然后几何着色器为每个输入顶点输出一个四边形或一个三角形。这些测试还会比较芯片内部和芯片外部几何着色器的分配。编译器默认使用芯片外部内存，该内存用于存储顶点和几何阶段之间的阶段间数据。

优势

* 知名且简单。

劣势

* 需要进行最多设置。
* 在所有情况下，速度都不是最快的。

## 方法 2 – 仅几何着色器

此方法使用空顶点着色器。为了加载顶点数据，几何着色器使用 SV\_PrimitiveID 索引在顶点缓冲区的原始字节视图上执行缓冲区加载（对于点列表，几何着色器中的 SV\_PrimitiveID 与顶点着色器中的 SV\_VertexID 相同）。加载顶点之后，按照方法 1 执行点精灵展开。此处对芯片内部与芯片外部 GS 性能进行了类似的测试。

优势

* 无需输入布局，因为 VS 是 no-op。
* 与方法 1 相比，它的执行速度更快，因为 VS 和 GS 之间没有内部流量。

劣势

* 手动顶点加载必须为每个顶点布局提供不同的着色器变体。对于高度优化的代码，这不是问题。
* 这种方法有点难以理解。

## 方法 3 – 仅顶点着色器

在 DX11 中，并不一定要求只使用几何着色器将一个点展开成一个四边形或三角形，可以用顶点着色器来实现此目的。在 DX11 中，顶点着色器阶段可以读取原始字节 UAV，所以有了 SV\_VertexID 和顶点缓冲区的原始字节视图，就可以手动读取顶点了。

因此，若要把顶点展开成三角形或四边形，只需在绘图调用中呈现 3 倍或 6 倍以上的顶点，在着色器中执行除以 3 或 6 的除法以获取顶点的索引，使用此索引加载顶点，然后根据除法的余数展开它来获取精灵的角。

优势

* 到目前为止，这是呈现点精灵的最快方法。对于较小的精灵（受几何限制的情况），这种方法比任何几何着色器方法都要快 3 倍以上，在较大的精灵上，此方法的速度仍快 50% 左右。
* VS 不需要输入布局，因为它只消耗系统生成的 SV\_VertexID 值。

劣势

* 顶点加载是手动操作，所以可能需要不同的着色器来加载不同的顶点布局。如果想要使用绝对最快的方法，这不是问题。

## 方法 4 – 顶点着色器实例化

可以用实例化的方式，让 GPU 为我们加载点精灵顶点，加载 SV\_VertexID 来确定精灵的角，以便于扩展。这种方法比前一种方法稍慢，但仍比其他任何方法都快。

优势

* 速度第二快的方法，性能与方法 3 非常相似。
* 无需使用除法，因此着色器的 ALU 更少。
* 使用输入布局进行顶点加载，因此可以在不修改着色器的情况下加载不同的顶点布局。
* 似乎是超大尺寸的点精灵的获胜方法。

劣势

* 无

## 方法 5 – 分隔阶段

它可以使用分割阶段从单个输入顶点生成三角形和四边形。这种方法的性能与基于几何着色器的方法不相上下，但它具有更强的灵活性，因为可以实现更多的精灵形状。例如，通过在四边形域中进行分隔，可以输出圆形而不是四边形，如果像素着色器的速度非常慢，且精灵是圆形的话，此方法可能更高效。

优势

* 可以输出几乎任意几何图形。

劣势

* 与几何着色器的方法相同。

## 结论：

以上所有的方法在不同的 GPU 负载下都会有不同的表现，因此，请选择最适合自己的方法。

# 已知问题

此示例目前不会通过 Gaming.Xbox.Scarlett.x64 平台进行生成。

# 更新历史记录

2019 年 4 月 12 日 – 从 Xbox 示例框架移植到 DX12。

# 隐私声明

在编译和运行示例时，示例可执行文件的文件名将发送给 Microsoft，用于帮助跟踪示例使用情况。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/)》。