

高级 ESRAM 示例

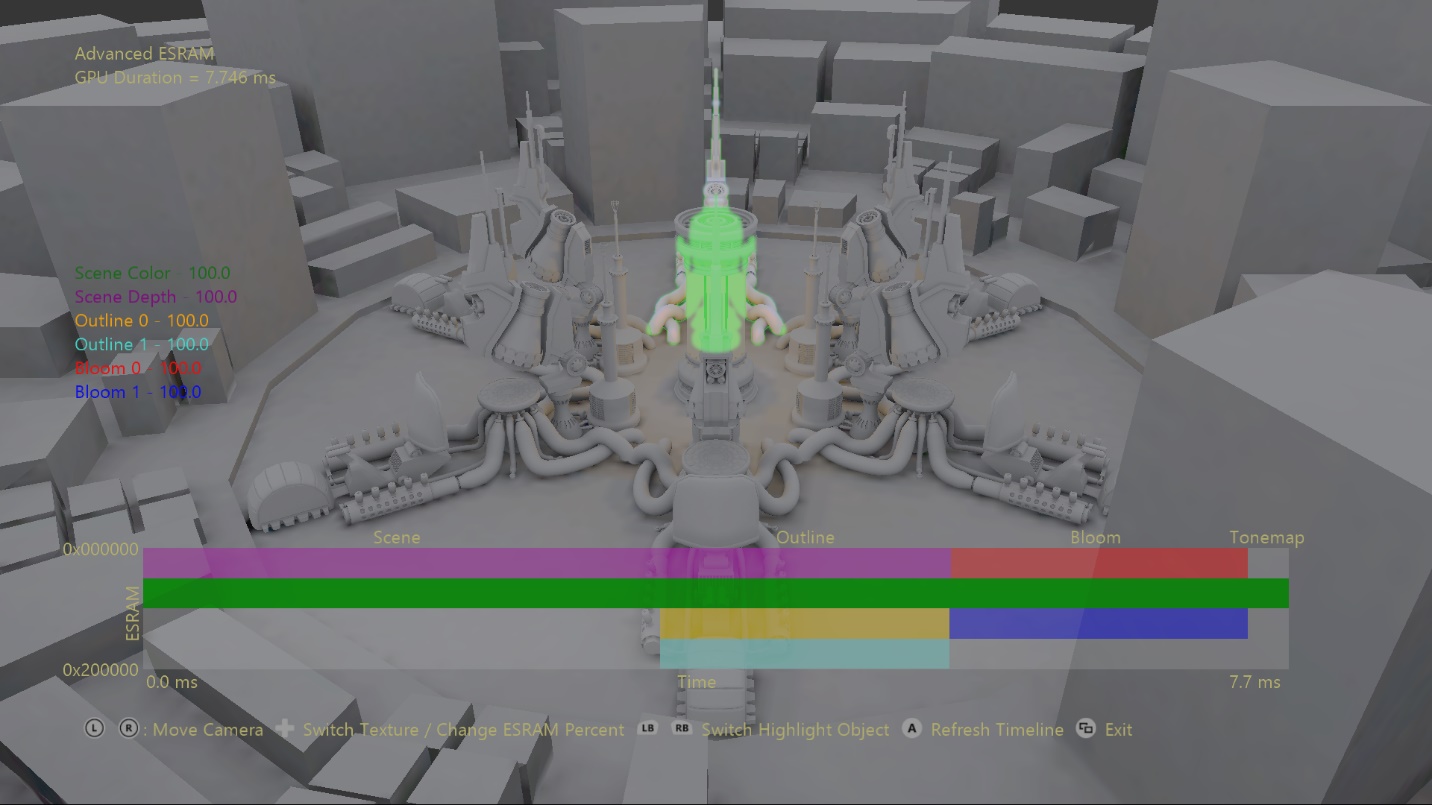
*\* \* 此示例与 2019 年 11 月 GXDK 兼容。*

# 说明

此示例演示如何使用高级 DirectX 12.x 内存功能来有效地别名化 D3D 资源的内存。此示例的核心 API 为 ID3D12CommandQueue::CopyPageMappingsX 和 ID3D12CommandQueue::CopyPageMappingsBatchX。这些函数支持将 CPU 分页表项复制到 GPU 时间线上的 GPU TLB，允许将虚拟 D3D 资源动态映射到内存页。

此示例利用此功能来实现瞬态资源分配器，该资源分配器以 64 KB 页面粒度映射各个 ESRAM 和 DRAM 块。这样可以使帧的 GPU 内存使用保持最佳状态，从而可以充分利用 ESRAM 的全部功能。其界面反映了 XG 内存库中 XGMemoryLayout 的页面映射功能的重要方面。

备注：Xbox One X 和 Scarlett 没有 ESRAM。在这些平台上，此示例将在禁用所有 ESRAM 选项和可视化效果的情况下简单地渲染场景。



# 使用示例

该示例的主要功能允许操纵瞬态纹理资源的分配位置。帧中使用的资源适用于场景颜色、场景深度的纹理，两个用于轮廓，两个用于泛光。资源的 ESRAM 组合以占资源内存的百分比形式显示在最左侧。ESRAM 布局的可视化效果可立即反馈每个纹理的 ESRAM和 DRAM 组合的变化。每个资源的 ESRAM 占用空间沿 Y 轴显示，而其生存期沿 X 轴显示。可以通过按下按钮来刷新沿时间轴使用的 GPU 计时。

## 控制

|  |  |
| --- | --- |
| 操作 | 游戏手柄 |
| 将相机移向原点/远离原点 | 左操纵杆向上/向下 |
| 沿轨道移动相机 | 右操纵杆 |
| 重置相机 | 右操纵杆（单击） |
| 循环瞬时纹理 | D-Pad 左/右 |
| 更改 ESRAM 百分比 | D-Pad 向上/向下 |
| 循环突出显示的对象 | 左/右缓冲键 |
| 刷新时间线 | A 按钮 |
| 退出 | “视图”按钮 |

# 实现说明

将创建一个较大虚拟地址空间，用于映射从 ESRAM 和 DRAM 分配的 64 KB 内存页（页面池）块。页面池使用 ID3D12Device::RegisterPagePoolX 在 DirectX12 中注册，不再使用时，将使用 ID3D12Device::UnregisterPagePoolX 注销。此映射用于暂存 CPU 分页表项，以便直接复制到 GPU 分页表。

ID3D12CommandQueue::CopyPageMappingsX 或 ID3D12CommandQueue::CopyPageMappingsBatchX 函数使这些页面池中的页面范围能够映射到 GPU 时间线上的指定 GPU 虚拟地址。这使虚拟 D3D 资源可以灵活地动态映射到 64 KB 物理页面。此功能使资源之间的内存别名化变得微不足道，并且乐趣无穷！

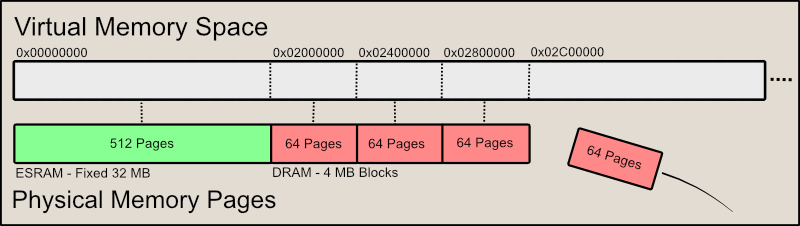


图 1：示例中使用的内存映射范例。ESRAM（如果有）被映射到虚拟地址空间的前 32 MB，同时根据需要附加 4 MB DRAM 页面池。

页面块的创建和管理由 PageAllocator 类执行。将提供一个虚拟地址范围，并且分配器会根据需要按顺序将页面池映射到此范围。然后使用“RegisterPagePoolX”在 DirectX12 中注册页面池。将完全跟踪分配器页面的使用情况 - 它从头到尾分配页面，并在页面释放时替换它们。

TransientCache 负责管理虚拟 D3D 资源。这些都是根据需要创建的，但会进行缓存以避免重新创建常见资源而产生不必要的开销。缓存这些资源的内存开销实际上为零，因为它们仅分配虚拟地址空间。对于每个帧，每个资源只能分配一次。

此 TransientAllocator 类使用页面分配器和瞬态缓存来向用户执行资源请求。当请求资源时，它将从 TransientCache 中获取一个实例。然后，它从 PageAllocators 分配所需的页数，分析令牌以确定是在页面级粒度使用 ESRAM 还是 DRAM。然后生成适当的结构，以便稍后提供给“CopyPageMappingsBatchX”，即 D3D12XBOX\_PAGE\_MAPPING\_BATCH 和 D3D12XBOX\_PAGE\_MAPPING\_RANGE 结构的矢量。

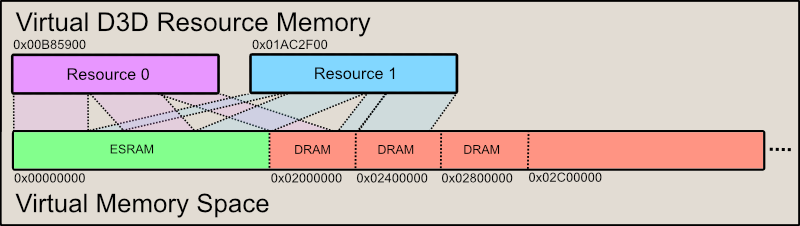


图 2：虚拟资源正被映射到页面池内的页面范围，以满足其内存需求。这些映射是 CopyPageMappingsX 和 CopyPageMappingsBatchX 调用的结果。为了保持视觉简洁，此可视化效果中仅使用了两个资源，并且未对内存进行别名化。但是，内存别名化是此技术的预期优势。

由于内存别名化，TransientAllocator 还负责在必要时执行着色器和缓存刷新。在 DirectX12 中，刷新将作为资源屏障的一部分插入。由于我们已规避此系统执行内存别名化，因此我们必须手动插入自己的刷新。TransientAllocator 通过检查资源的关联视图来确定应刷新哪些着色器阶段和缓存。

最后，在将使用分配的瞬态资源的命令列表提交到其命令队列之前，必须在 TransientAllocator 上调用“Finalize”以完成资源映射。此时，CopyPageMappingsBatchX 调用将被置于命令队列中，其设置要在后续命令列表中使用的资源内存映射。

# 更新历史记录

2018/8/6 - 创建示例。

2019/12/17 – 移植到 GXDK 和 Scarlett。

# 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，以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/)。