利用DirectX 12的底层特性大幅度提高渲染效率

曹家音 - 内容技术工程师
jecao@nvidia.com

NVIDIA

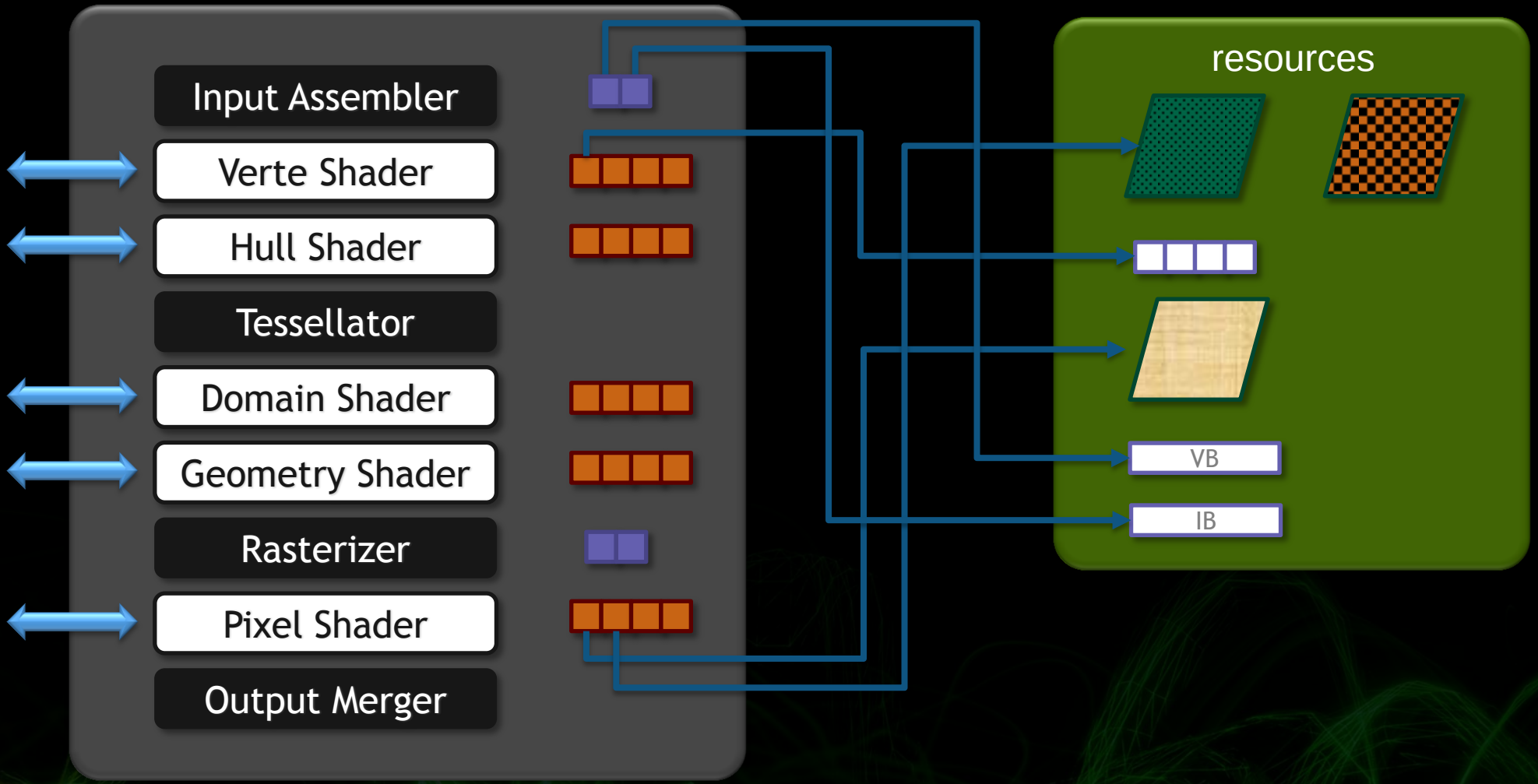
摘要

- D3D12简要介绍
- 显式的内存管理模型
- 减少CPU开销
  - CPU 效率优化
  - CPU 并行优化
- 提高GPU的性能
- D3D12与D3D11，OpenGL 4.x的性能对比
- 新的图形特性

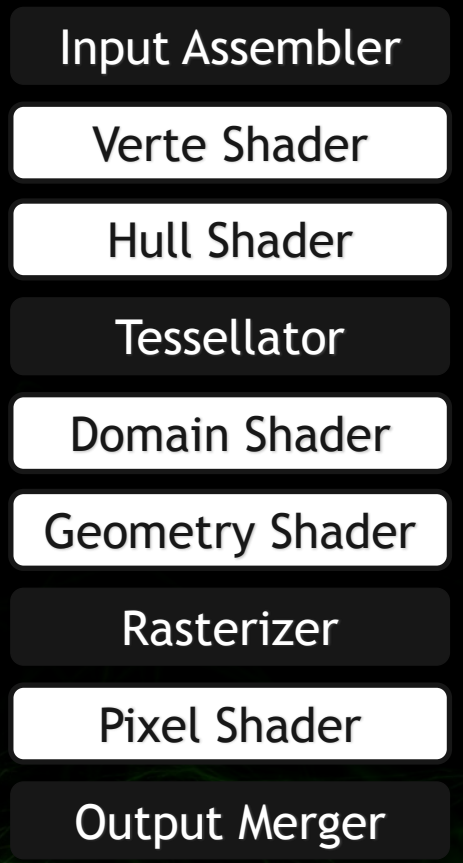
Direct3D12 简介

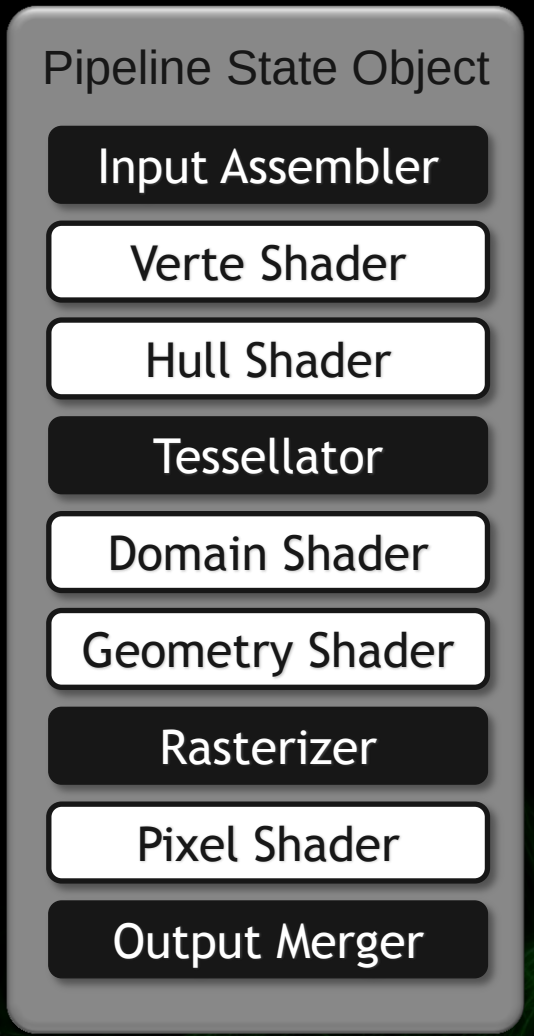
- 最新的高性能图形API
- 更加底层，直接
- 在微软的所有平台都可以运行

D3D11 图形管线


Input Assembler
Verte Shader
Hull Shader
Tessellator
Domain Shader
Geometry Shader
Rasterizer
Pixel Shader
Output Merger
resources
VB
IB

Pipeline State Object



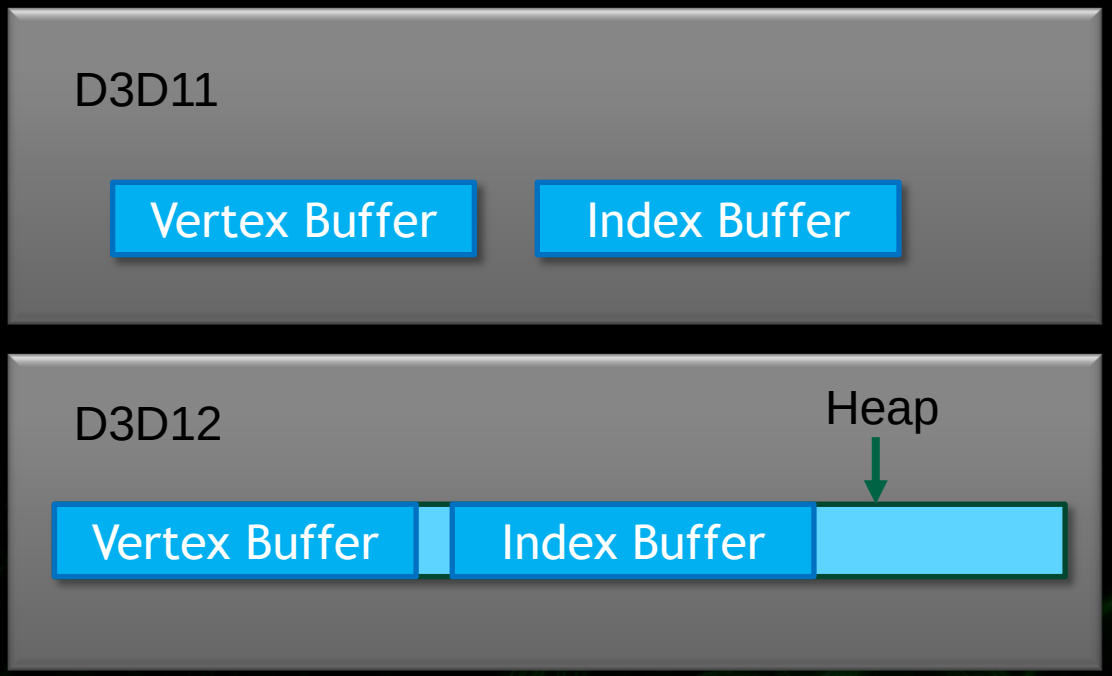


Pipeline State Object (续)

- 不会在渲染过程中有隐形的Shader编译与链接
- 在创建PSO的时候就已经生成大部分硬件指令
- PSO的Shader输入是二进制的，对于Shader Cache而言非常友好
- 需要开发者注意的事项:
  - 在单独的线程中创建PSO
  - 对于不在意的变量，尽量应用相同的默认值
  - 在连续的Draw Call中，尽量保证PSO状态相似

灵活的内存管理模型

- 基于堆的内存管理
  - 贴图
  - 缓冲 (VB/IB/CB)
  - Descriptors
  - 采样器



资源绑定模型

- 在D3D11中，有四种不同的View。在D3D12中，将会有更多种类的View
  - Constant Buffer View
  - Vertex Buffer View
  - Index Buffer View
  - ...
- 他们不再是D3D对象，开发者具有更多的内存管理权限

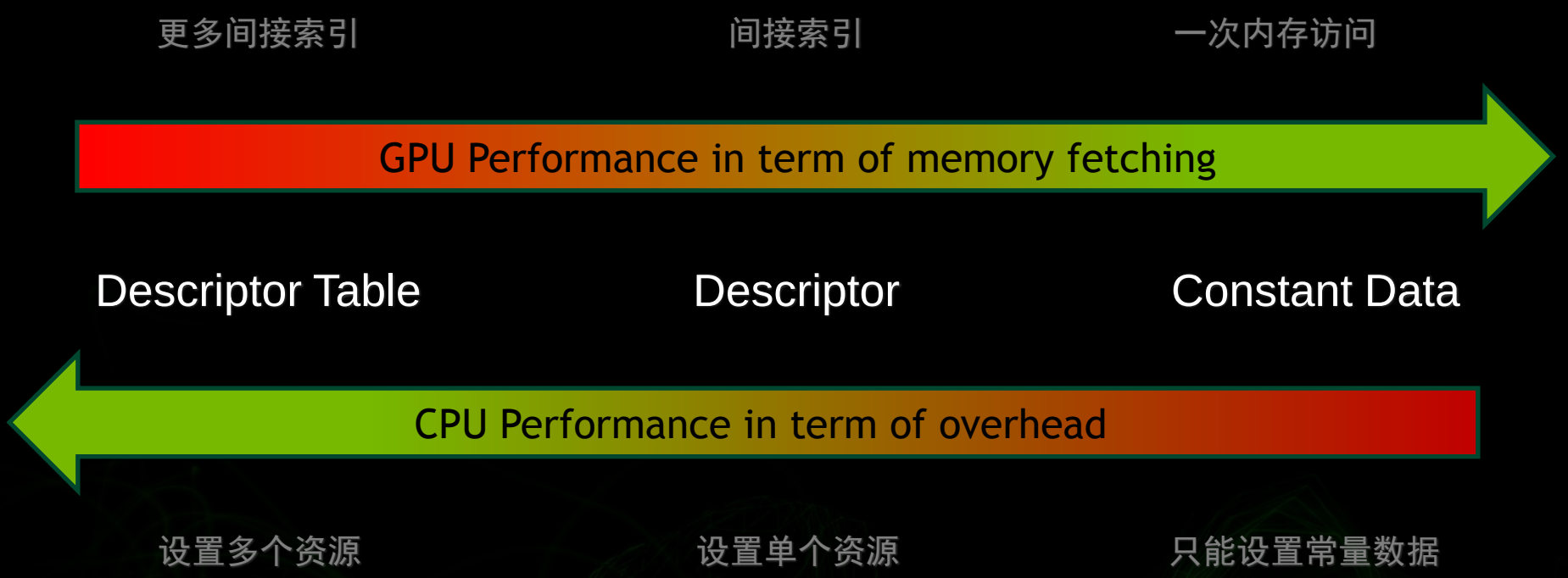
新的资源绑定模型

- 下面的资源绑定方式与D3D11相似：
  - 渲染目标
  - 顶点/索引缓冲（通过view，而不是资源句柄）
  - 视口/裁剪矩形
- 对于下面的资源，绑定方式有很大变化：
  - 贴图
  - 常量数据
  - 采样器
- 相比D3D11，D3D12需要设置另外一些数据：
  - PSO
  - Root Signature
  - Heap

新的资源绑定模型（续）

- D3D12 引入了一个全新的对象类型 “RootSignature”.
  - 它是为Shader设置资源的唯一“窗口”
- 可以以三种不同的形式设置资源和数据:
  - Descriptor table
  - Descriptor
  - Constant Data

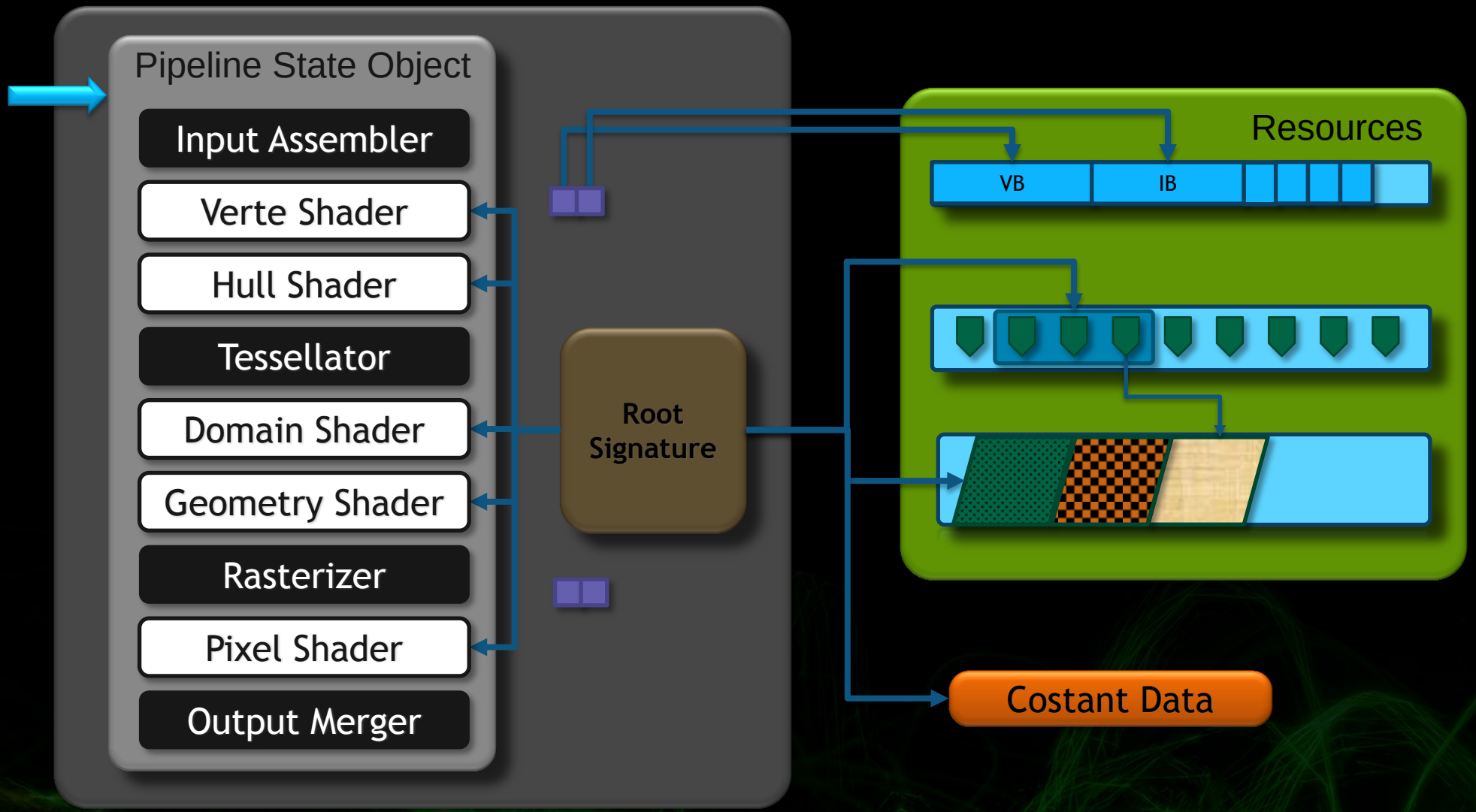
根据实际情况具体权衡


更多间接索引
间接索引
一次内存访问
GPU Performance in term of memory fetching
Descriptor Table
Descriptor
Constant Data
CPU Performance in term of overhead
设置多个资源
设置单个资源
只能设置常量数据

谨慎使用RootSignature

- 尽可能控制RootSignature大小
- 有效控制Shader可见范围
- 只有在必要的时候才更新RootSignature数据

全新的D3D12图形渲染管线



资源管理中的注意事项

- D3D管线中，几乎所有任务都是延迟处理的，确认不要更改仍然在处理队列中的数据

- 开发者需要自己处理如下问题
  - 资源生命周期管理
  - 资源存储管理
  - 避免资源冲突

防止资源冲突

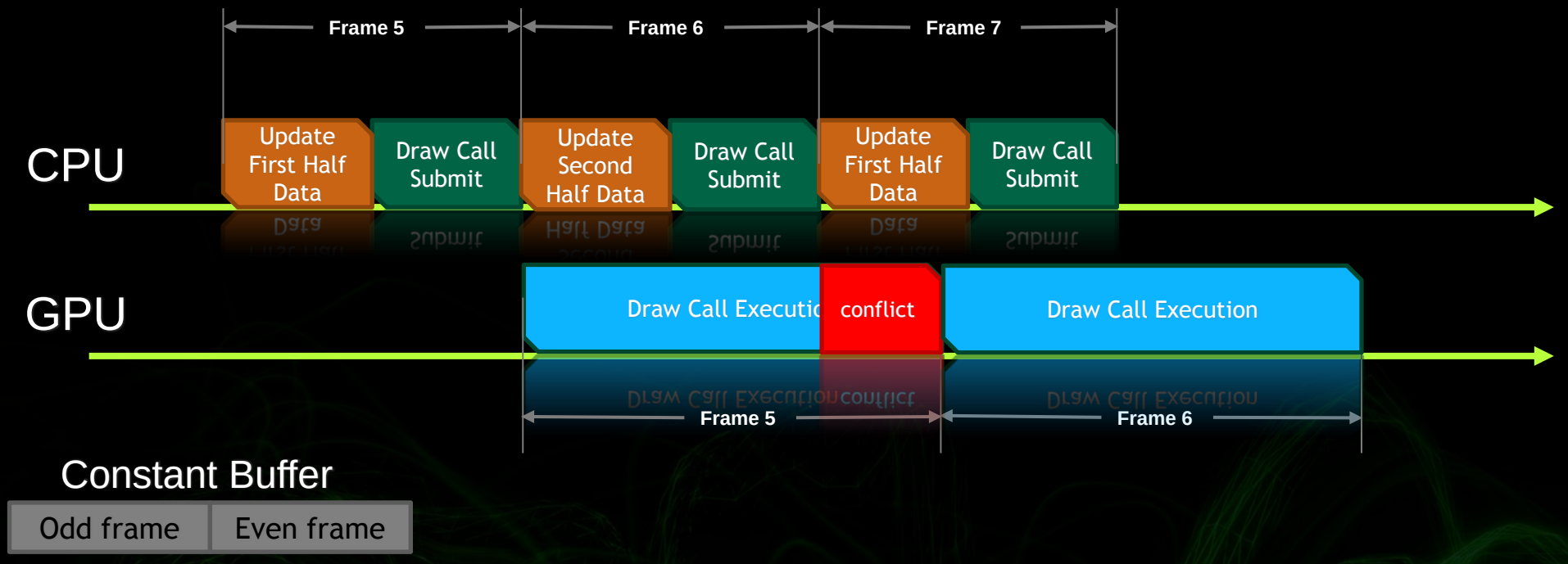
- D3D11的资源状态切换是隐式的
- 在D3D12中，开发者需要自己正确维护资源的状态切换
  - ResourceBarrier

防止资源冲突

- D3D11的资源状态切换是隐式的
- 在D3D12中，开发者需要自己正确维护资源的状态切换
  - ResourceBarrier

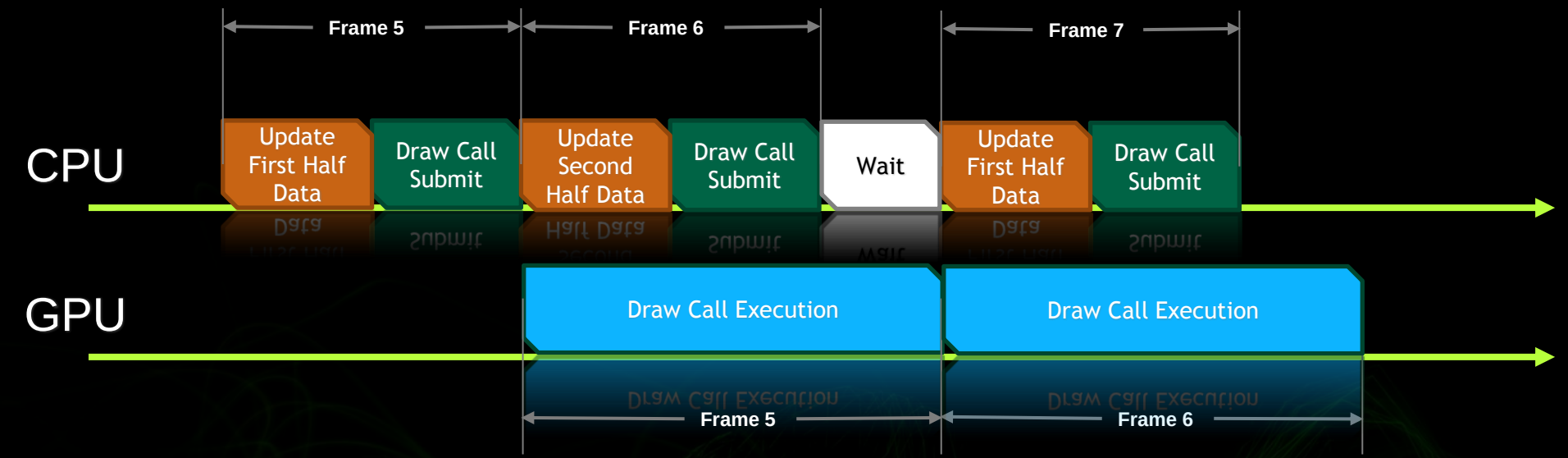
另一个冲突的例子

- 注意不要更改GPU正在读取的数据



另一个冲突的例子

- 注意不要更改GPU正在读取的数据

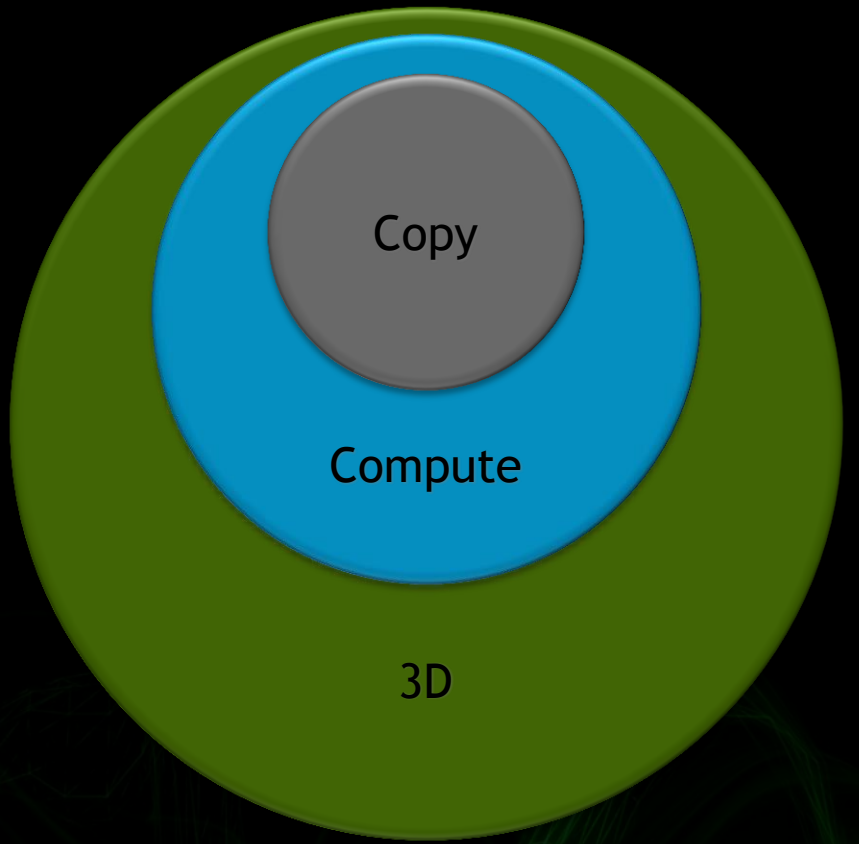


典型的资源冲突场景

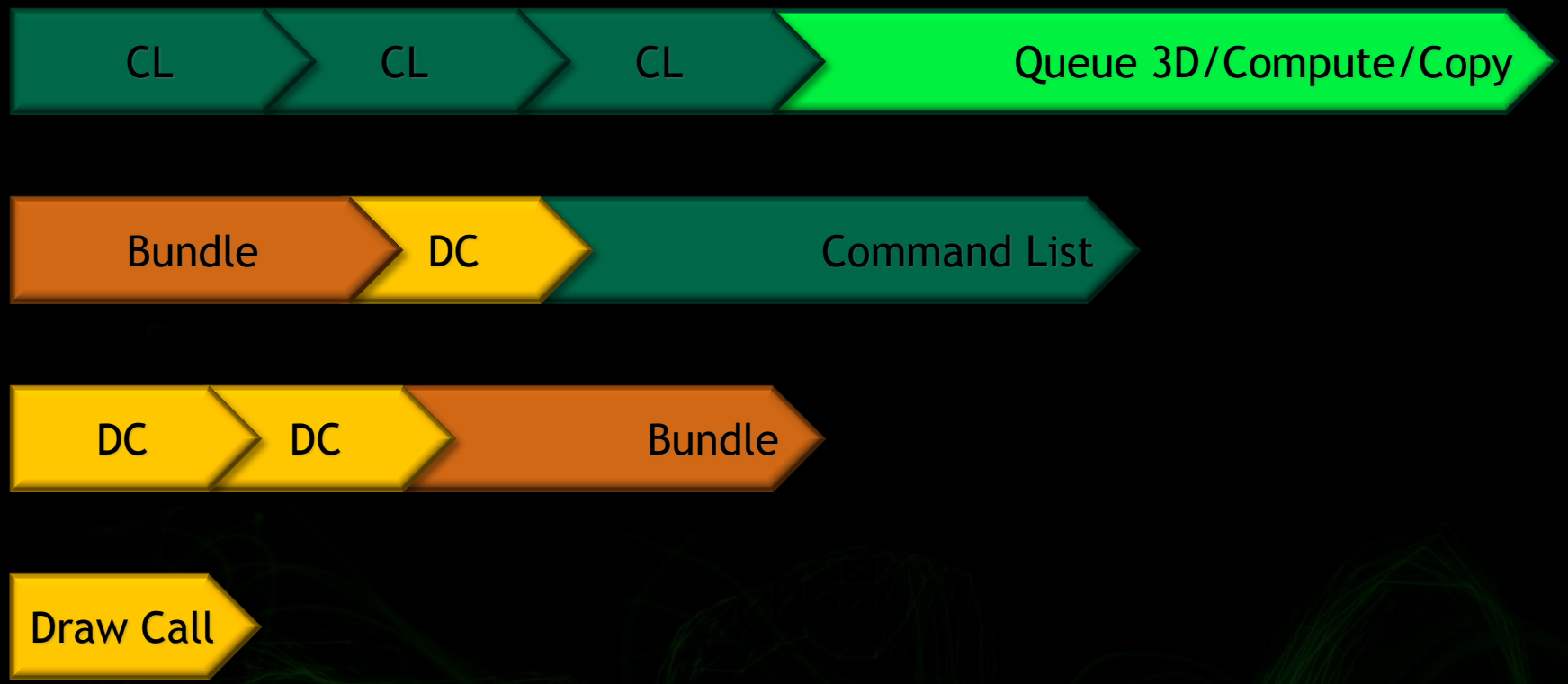
- Shadow map
- 延迟光照/渲染
- 实时的折射和反射
- …
- 任何应用渲染目标作为后续渲染中贴图的情况

Draw/dispatch调用的全新模型

- 命令队列
  - 3D队列
  - 计算队列
  - 拷贝队列
- 命令列表
- Bundle



Draw/Dispatch call调用模型



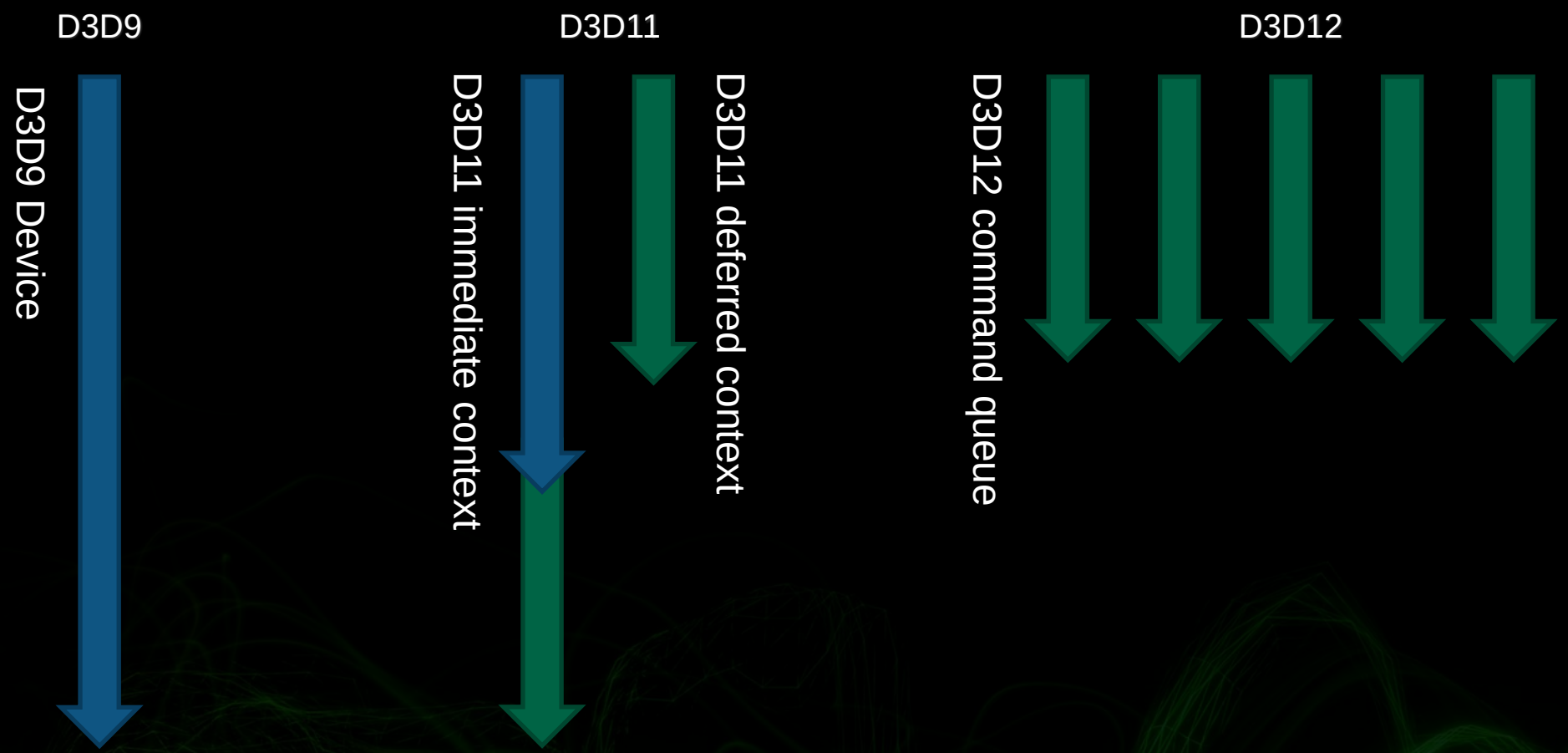
提交Draw call的具体步骤

- 不再有立即模式
- 为了提交一个Draw call
  1. 创建一个或多个3D/计算队列
  2. 创建一个或多个命令列表
  3. 为每个命令列表记录Draw call
  4. 执行命令列表

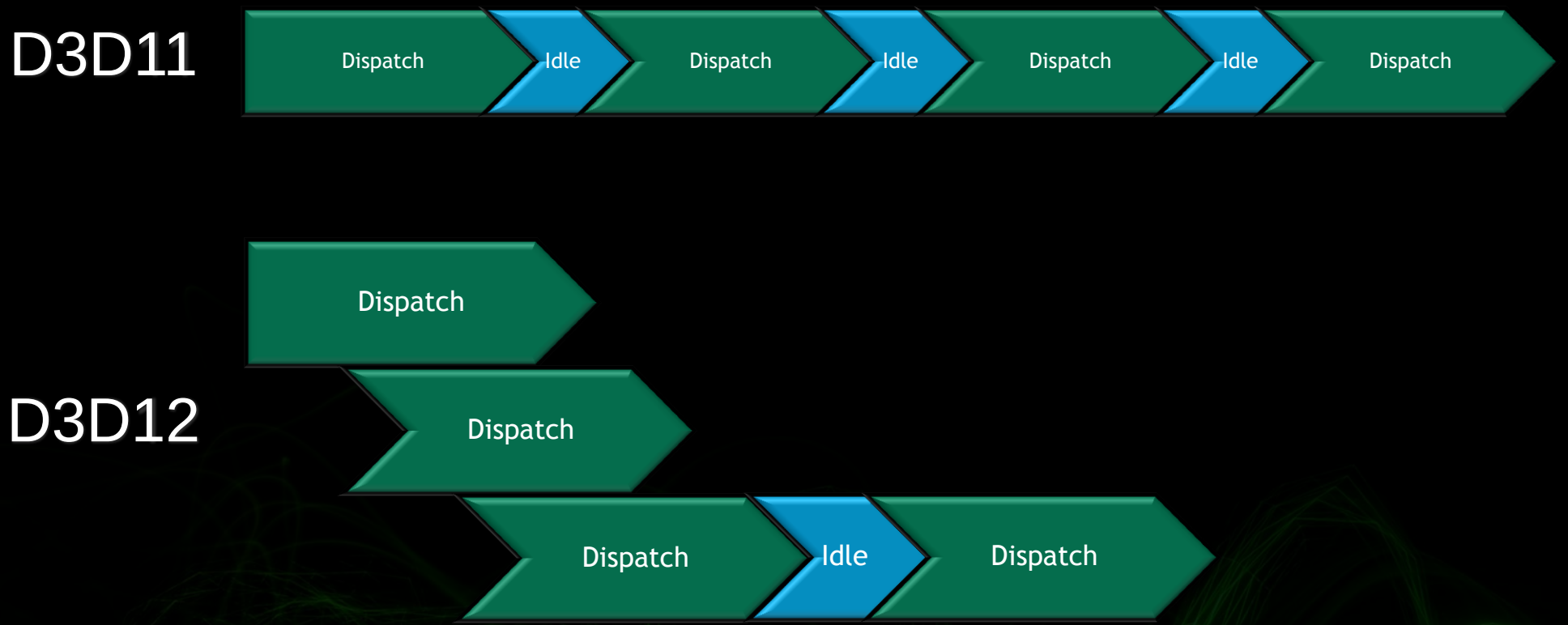
多线程渲染

- 之前的多线程渲染模型
  - 一个单独的渲染线程用来提交Draw/Dispatch调用
  - 其他一个或多个线程处理游戏逻辑
- 新的多线程模型
  - 每个线程都可以提交Draw/Dispatch调用

多线程渲染（续）


D3D9
D3D9 Device
D3D11
D3D11 immediate context
D3D11 deferred context
D3D12
D3D12 command queue

更好的GPU效率


D3D11
Dispatch
Idle
Dispatch
Idle
Dispatch
Idle
Dispatch
D3D12
Dispatch
Dispatch
Dispatch
Idle
Dispatch

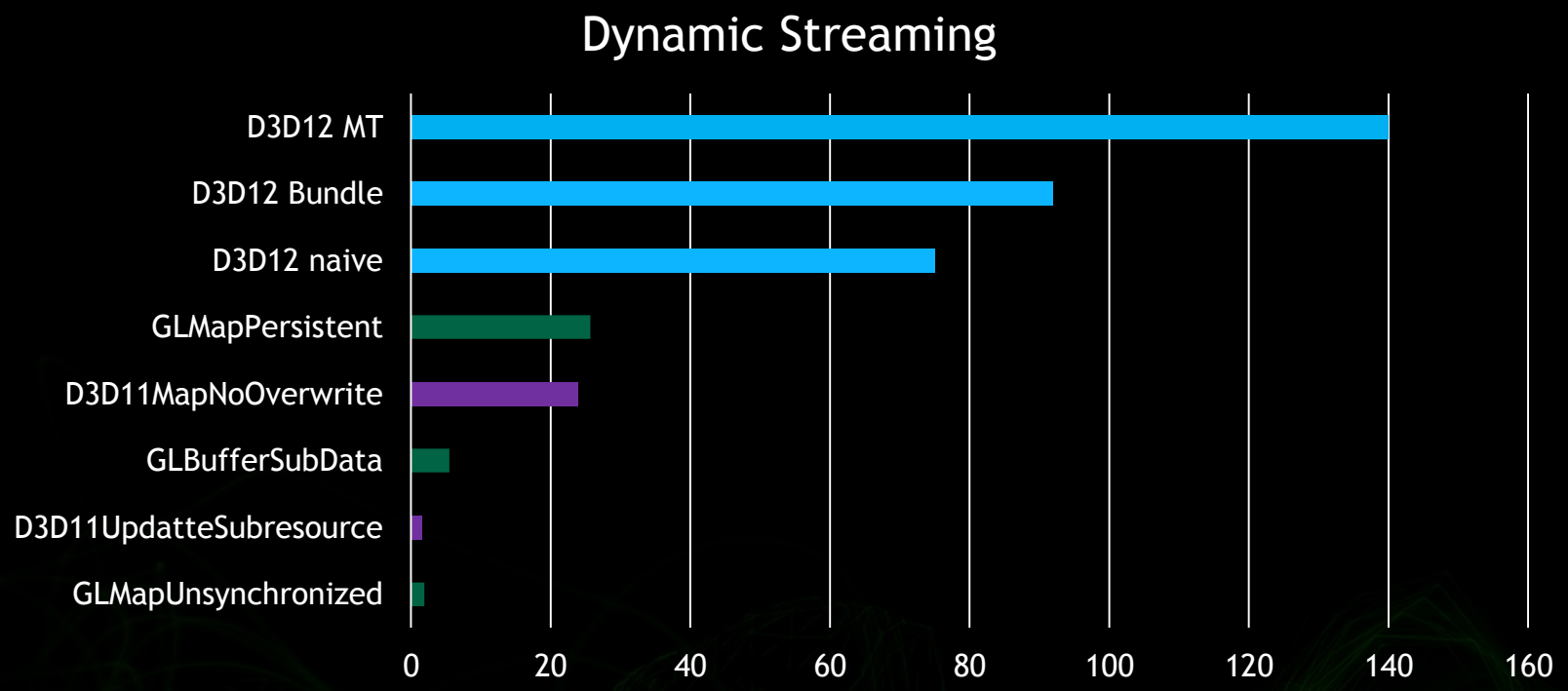
从D3D11移植到D3D12

- 简单快捷的解决方案：D3D11on12
- 只需要在D3D11程序的基础上进行简单的改动：
  - 创建D3D12设备
  - 创建D3D11形式的资源来包装BackBuffer
  - 显式地管理BackBuffer
  - 在Present前提交所有指令
  - 为你的渲染增加Fence
- 完整的D3D12移植往往是非常必要的，D3D11on12不会带来特别多的性能提升.

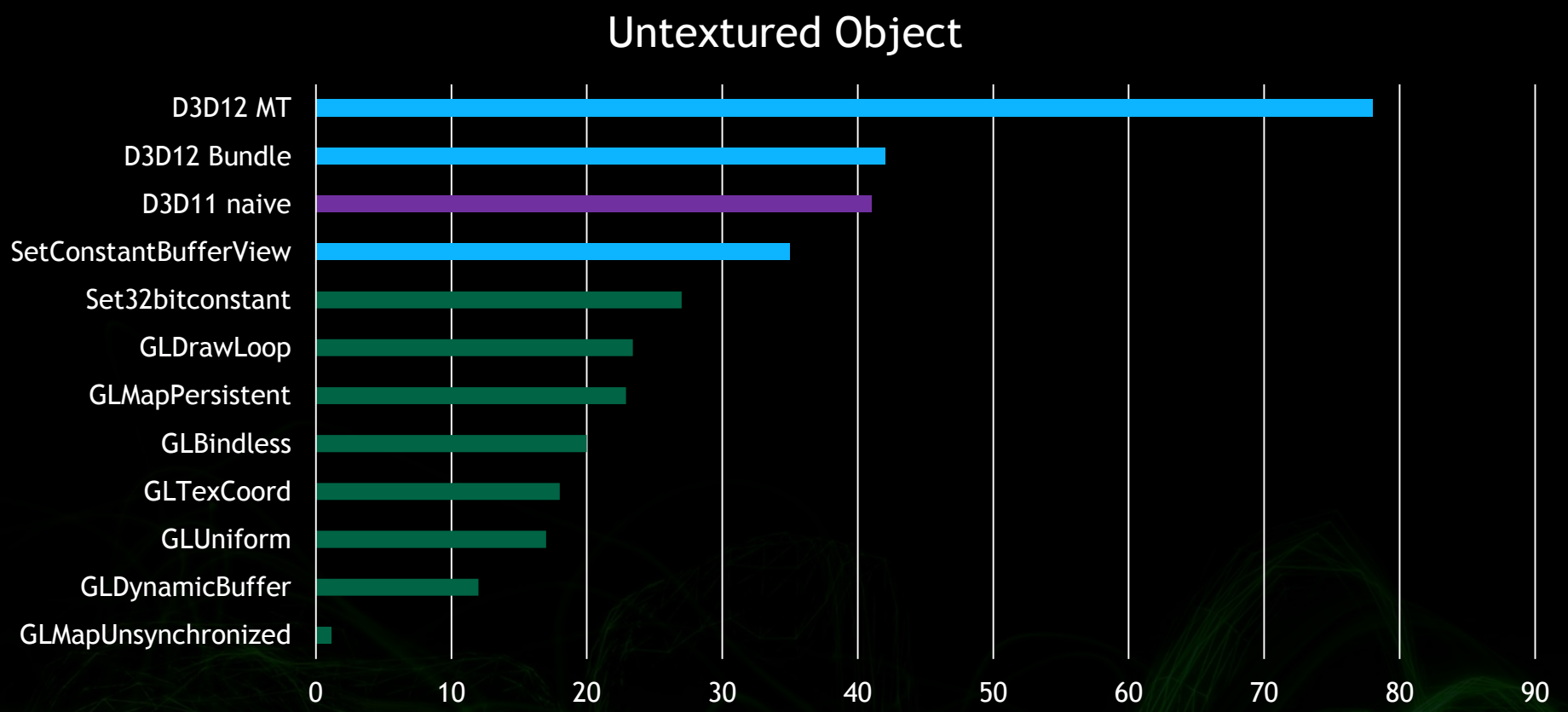
API test (扩展版本)

- API test 是一个用于测试API性能的Benchmark.
- 其中有四个渲染情景:
  - 清除Back Buffer
  - 动态Streaming，250000个粒子，每个粒子都有不同的顶点缓冲数据
  - 无贴图渲染，64x64x64个方块，每个方块有单独的常量数据
  - 有贴图渲染，160000个正方形，每个正方形有单独的贴图
- 可以在Github上下载到源代码:
  - https://github.com/JerryCao1985/apitest

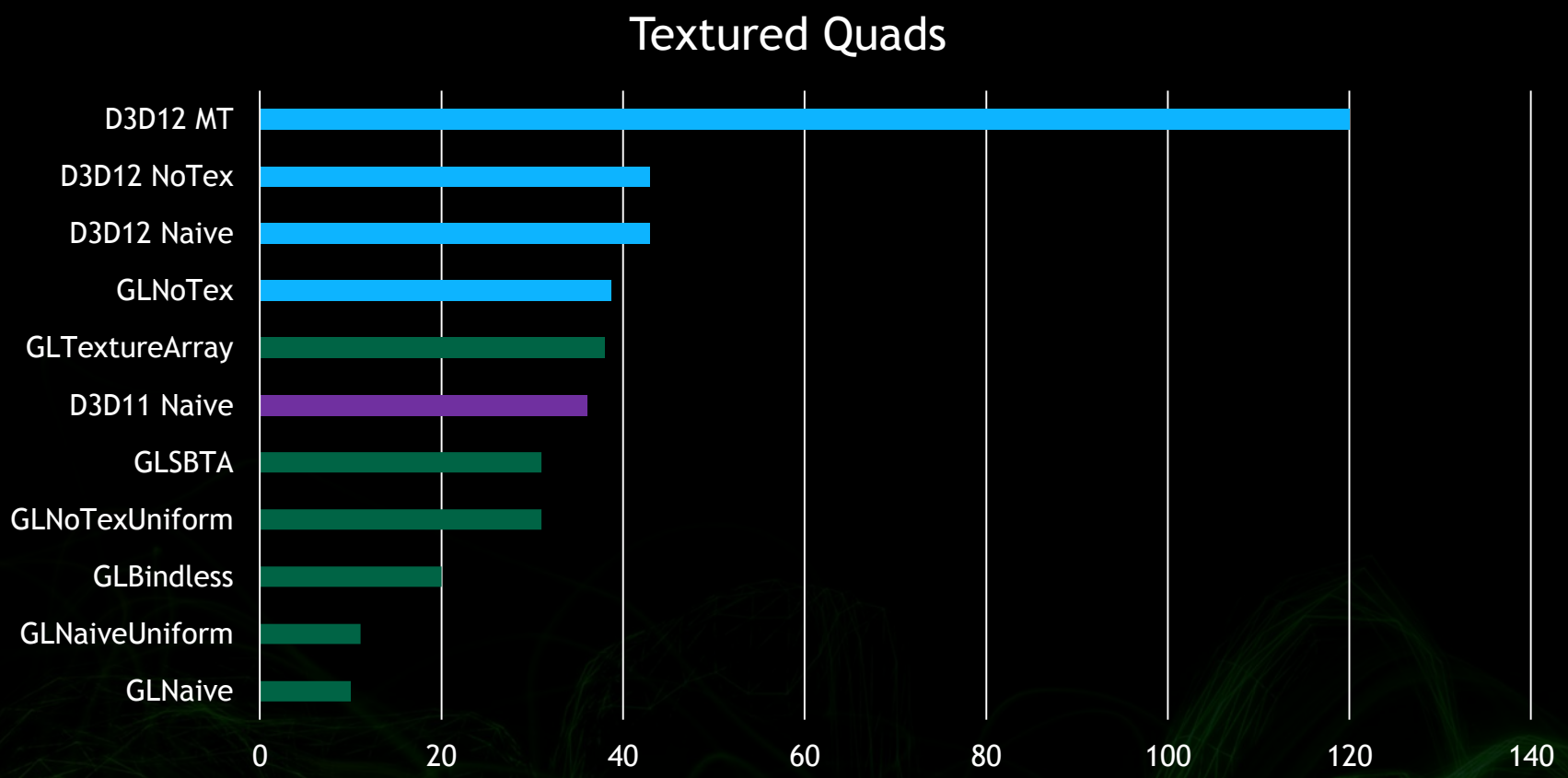
性能


Dynamic Streaming
D3D12 MT
D3D12 Bundle
D3D12 naive
GLMapPersistent
D3D11MapNoOverwrite
GLBufferSubData
D3D11UpdatteSubresource
GLMapUnsynchronized
0
20
40
60
80
100
120
140
160

性能


Untextured Object
D3D12 MT
D3D12 Bundle
D3D11 naive
SetConstantBufferView
Set32bitconstant
GLDrawLoop
GLMapPersistent
GLBindless
GLTexCoord
GLUniform
GLDynamicBuffer
GLMapUnsynchronized
0
10
20
30
40
50
60
70
80
90

性能

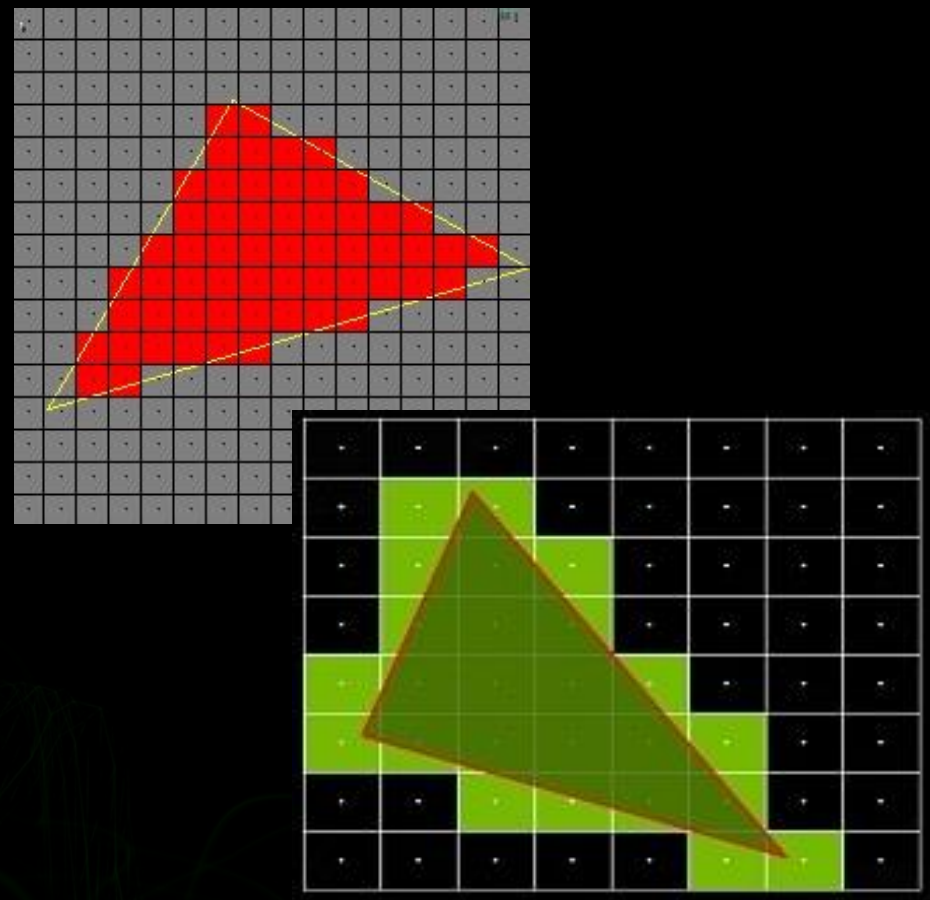

Textured Quads
D3D12 MT
D3D12 NoTex
D3D12 Naive
GLNoTex
GLTextureArray
D3D11 Naive
GLSBTA
GLNoTexUniform
GLBindless
GLNaiveUniform
GLNaive
0
20
40
60
80
100
120
140

新的图形特性

- 保守光栅化
- 光栅化顺序查看
- 区块化资源（三维贴图）
- Typed UAV Load
- 像素着色器制定的模板参考值

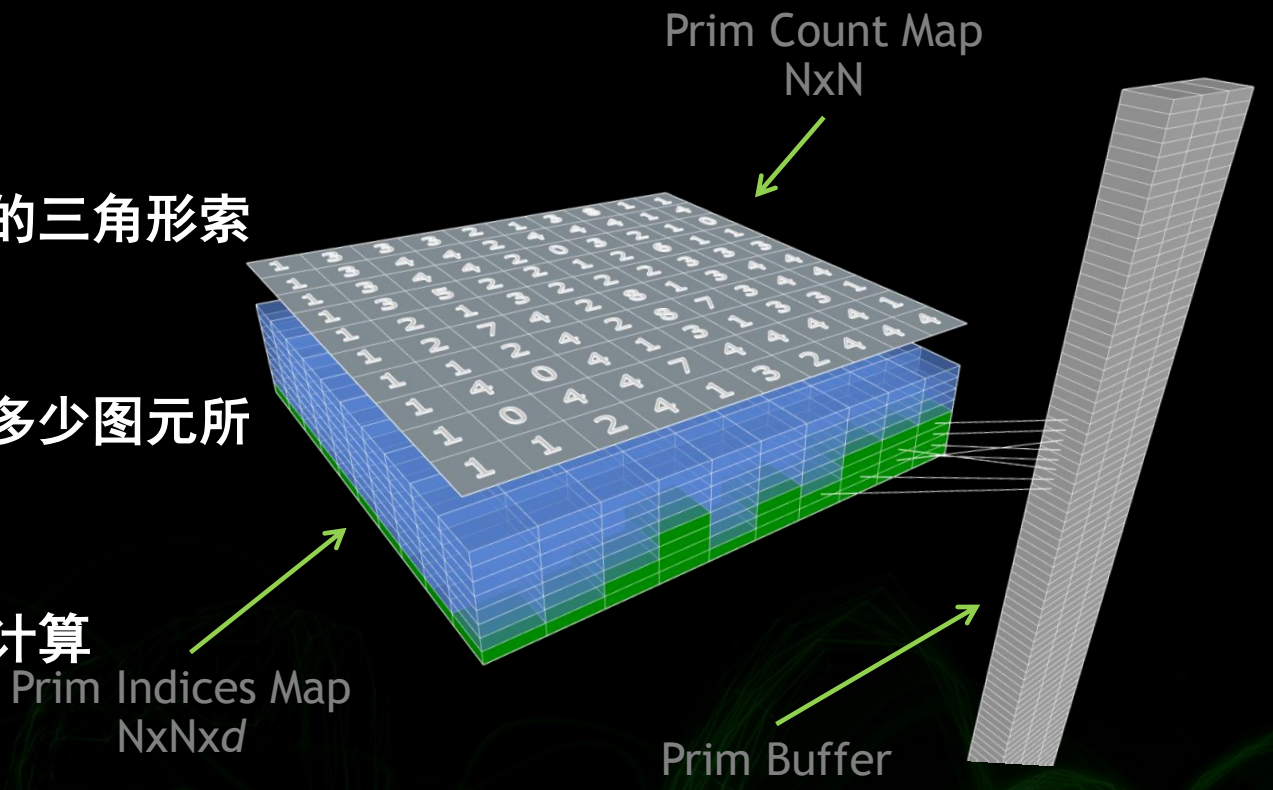
保守光栅化

- 所有被三角形覆盖的像素都会被着色，而不是应用传统的top-left标准
- 可以用GS来模拟，但是相对较慢
  - See J. Hasselgren et. Al, “Conservative Rasterization“, GPU 精粹 2
- 我们可以用保守光栅化实现更多的图形特效

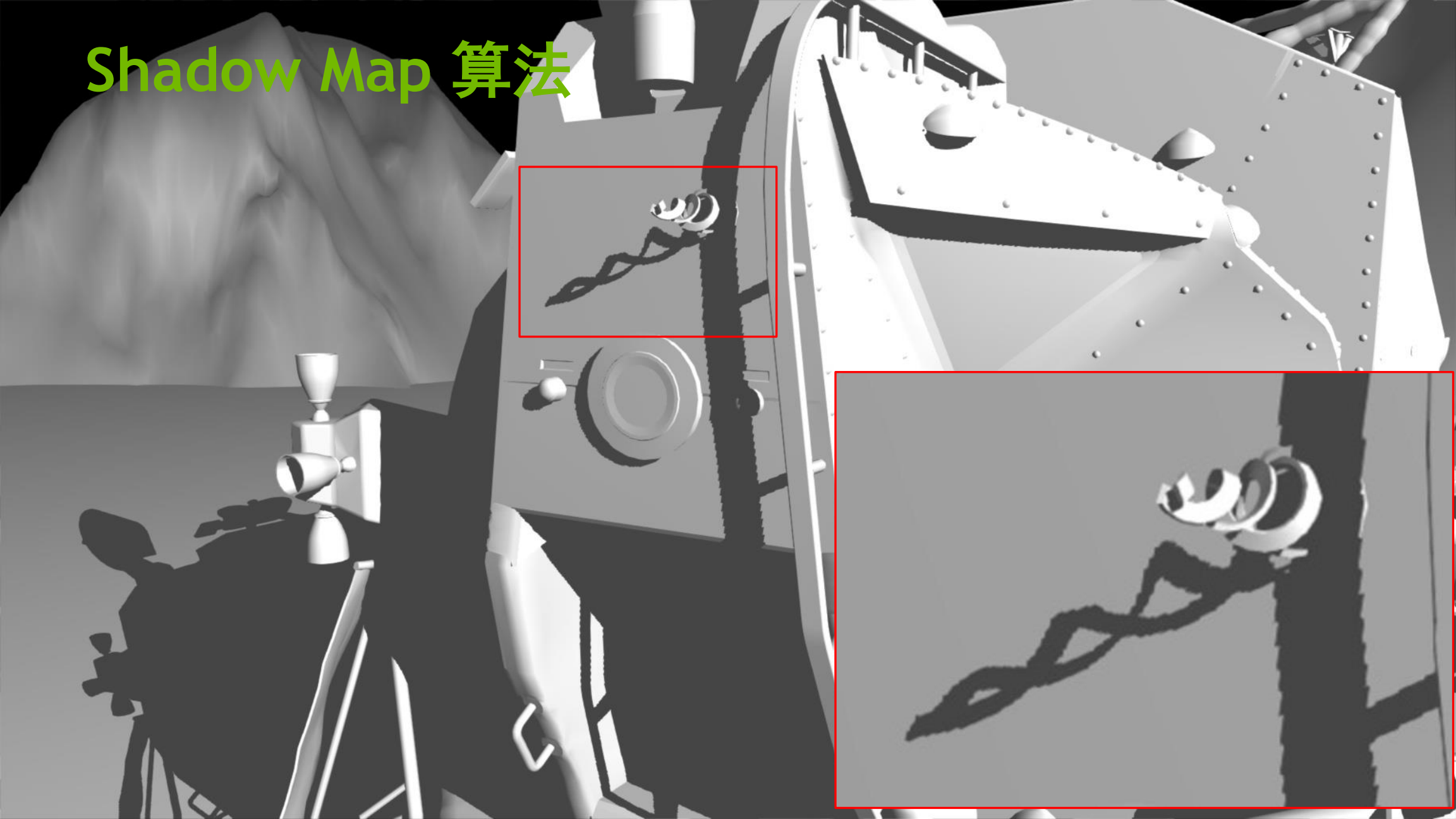


混合式光线跟踪阴影

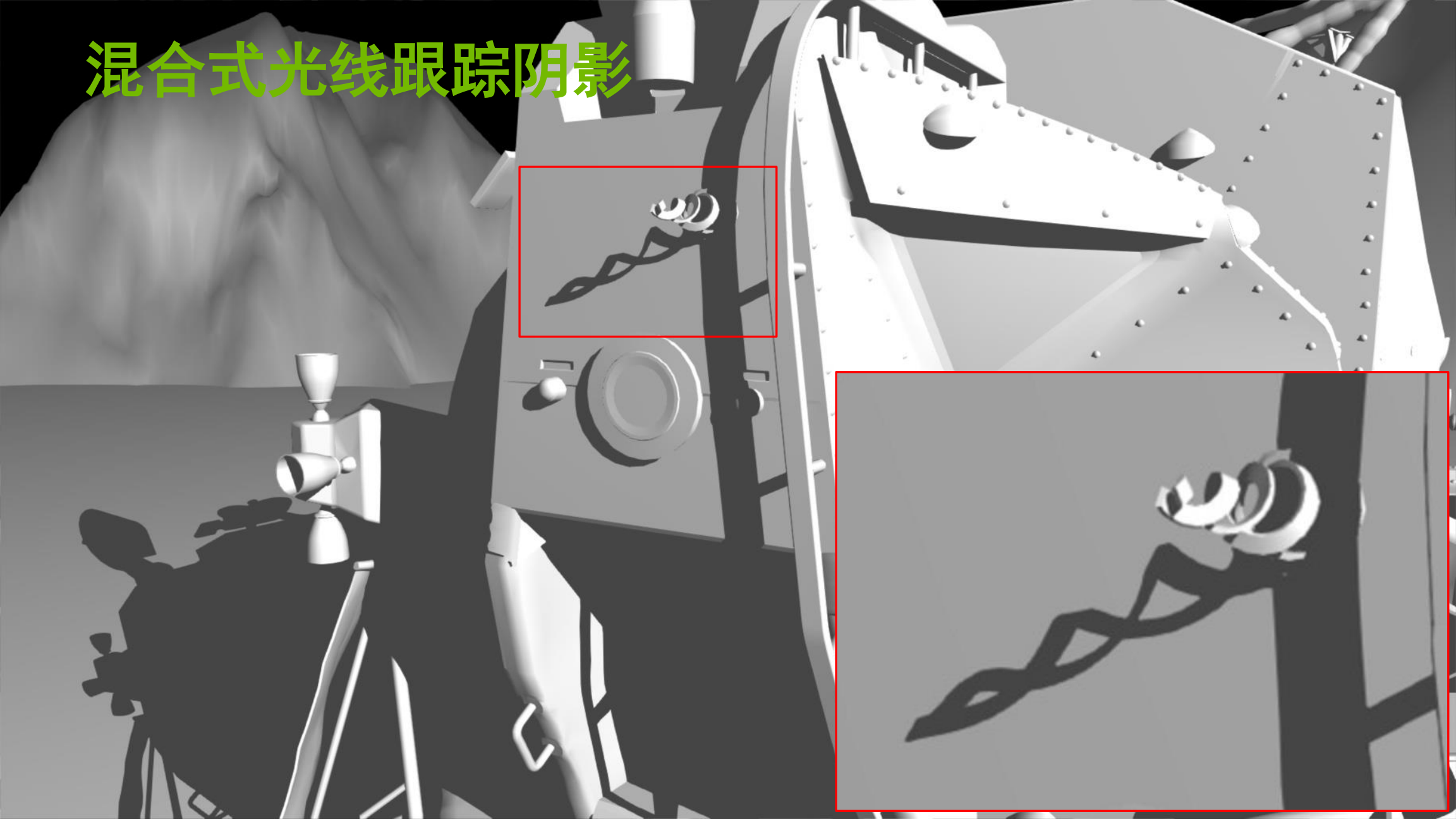
- 图元缓冲 – 三角形顶点数据
- 图元索引缓冲 – 图元缓冲中的三角形索引
- 图元数量缓冲 – 每个像素被多少图元所覆盖
- 在后续的Pass中进行光线跟踪计算



Shadow Map 算法



混合式光线跟踪阴影



总结

- D3D12提供的更好的性能
  - 管线变化
  - 内存管理模型
  - Draw/Dispatch提交模型
  - 减少多余的驱动等待
- D3D12和其他API的性能对比
- D3D12中新的图形特性
  - 混合式光线跟踪阴影

问答