Falcor
And How It Helps Our Rendering Research

Nir Benty, Senior Graphics Software Engineer
Invent New Rendering Techniques
Our Measure of Success

How Many of Our Inventions End Up In Products

Hardware Features/Games/Professional Graphics Tools
Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder

CHAKRAPANYI A. ALIA CHAITANYA, Massey University, University of Montreal and McGill University
ANTON S. KAPLANIAN, NVIDIA

GameWorks Ray Tracing Overview

Intersecting GameWorks for Ray Tracing

To allow game developers to create new worlds, NVIDIA has announced the NVIDIA GameWorks SDK that adds ray tracing capabilities. This suite of tools and resources for developers will dramatically increase realism and shorten shader development cycles. Developed using the Unreal Engine and NVIDIA RTX, these capabilities enable developers to create highly detailed scenes with realistic materials, physics, and lighting. With this new toolkit, developers can create complex high-quality reflections and scenes, allowing them to develop new lighting and shading techniques faster than ever. These capabilities are available on an industry-standard platform (like Fortnite), allowing developers to create games that have access to these tools.

Figure 1: Our Classroom scene with eye fixation at the yellow area. (Left) Our perceptually validated-shader prespes. We performed rendering under different lighting and shadows. Precomputing lighting data at 70% of the pixels and closely matches the frequency density of our validation data. (Right) A traditional shader that requires temporal anti-aliasing to improve temporal stability by an order of magnitude (spreading samples similar to a temporally-smartlighting. The original version of the classroom scene is courtesy of Christopher Svirad.

HFTS: Hybrid Frustum-Trace Shadows in “The Division”

Infiltrating GameWorks for Ray Tracing

With these capabilities, developers can create realistic high-quality reflections and scenes. NVIDIA RTX Shadow, which features dynamic lighting, and shadow maps, enables developers to create games with more realistic lighting and shadows. NVIDIA RTX Shadow is available for free in the NVIDIA GeForce Experience. Infiltrating GameWorks for Ray Tracing

Ashish Goel, NVIDIA, Stanford

Figure 1: Blurred alpha results in unsharp parts of geometry far from the source. We show a blur of static and animated objects using traditional alpha testing. (a) a) traditional, hardware-driven, and (b) standard blurring. The original version of the scene is courtesy of Christopher Svirad.

ACM Transactions on Graphics, 37(4), Aug 2018
Proceedings of ACM SIGGRAPH 2018

NVIDIA
Falcon

What Researchers Asked For

A tool to rapidly build and modify renderers

Code at the level of abstraction they think in

Visualize and debug everything

Support for the latest APIs and technologies
5 Things That Greatly Improve Our Research

- API-abstraction is essential
- Good raw-API interop is a must
- Crank shader-reflection up to 11
- Visual quality matters
- You really should be using a render-graph system
API-Abstraction Is Essential
Raw DX12

```cpp
#include "stdafx.h"
#include "D3D12HelloTriangle.h"

D3D12HelloTriangle::D3D12HelloTriangle(UINT width, UINT height, std::wstring name) :
  m_frameIndex(0),
  m_viewport(),
  m_scissorRect(),
  m_rtvDescriptorSize(0)
{
  m_viewport.Width = static_cast<float>(width);
  m_viewport.Height = static_cast<float>(height);
  m_viewport.MaxDepth = 1.0f;

  m_scissorRect.right = static_cast<long>(width);
  m_scissorRect.bottom = static_cast<long>(height);
}

void D3D12HelloTriangle::OnInit()
{
  LoadPipeline();
  LoadAssets();
}

void D3D12HelloTriangle::LoadPipeline()
{
  #if defined(_DEBUG)
  // Enable the D3D12 debug layer.

  ComPtr<ID3D12Debug> debugController;
  if (SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(debugController))))
  {
    debugController->EnableDebugLayer();
  }
  ```
341 lines of code later...
Why Abstraction?

- DX12 and Vulkan are error-prone APIs
- Faster adoption of new APIs
- Focus on your algorithm, not the API
- Abstraction layer contains many non-obvious optimizations
#include "RenderScene.h"

void RenderScene::onLoad(SampleCallbacks* pSample, RenderContext::SharedPtr pRenderContext)
{
    mpSceneRenderer = SceneRenderer::create(Scene::loadFromFile("SunTemple/SunTemple.fscene"));
    mpProgram = GraphicsProgram::createFromFile("RenderScene.ps.hls1", "", "main");

    mpGraphicsState = GraphicsState::create();
    mpGraphicsState->setProgram(mpProgram);
    mpProgramVars = GraphicsVars::create(mpProgram->getReflector());
}

void RenderScene::onFrameRender(SampleCallbacks* pSample, RenderContext::SharedPtr pRenderContext, Fbo::SharedPtr pTargetFbo)
{
    const glm::vec4 clearColor(0.38f, 0.52f, 0.10f, 1);
    pRenderContext->clearFbo(pTargetFbo.get(), clearColor, 1.0f, 0, FboAttachmentType::All);

    mpGraphicsState->setFbo(pTargetFbo);
    pRenderContext->setGraphicsState(mpGraphicsState);
    pRenderContext->setGraphicsVars(mpProgramVars);

    mpSceneRenderer->renderScene(pRenderContext.get());
}

int WINAPI WinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_ LPSTR lpCmdLine, _In_ int nShowCmd)
{
    RenderScene::UniquePtr pRenderer = std::make_unique<RenderScene>();
    SampleConfig config;
    config.windowDesc.title = "Falcor Model Viewer";
    Sample::run(config, pRenderer);
    return 0;
}
UE4’s Sun-Temple
(with very basic lighting)
#include "RenderScene.h"

void RenderScene::onLoad(SampleCallbacks* pSample, RenderContext::Shared pRenderContext)
{
    mpSceneRenderer = SceneRenderer::create(Scene::loadFromFile("SunTemple/SunTemple.fscene"));
    mpProgram = GraphicsProgram::createFromFile("RenderScene.ps.hlsl", ",", "main");
    mpGraphicsState = GraphicsState::create();
    mpGraphicsState->setProgram(mpProgram);
    mpProgramVars = GraphicsVars::create(mpProgram->getReflector());
}

void RenderScene::onFrameRender(SampleCallbacks* pSample, RenderContext::Shared pRenderContext, Fbo::Shared pTargetFbo)
{
    const glm::vec4 clearColor(0.38f, 0.52f, 0.10f, 1);
    pRenderContext->clearFbo(pTargetFbo.get(), clearColor, 1.0f, 0, FboAttachmentType::All);
    mpGraphicsState->setFbo(pTargetFbo);
    pRenderContext->setGraphicsState(mpGraphicsState);
    pRenderContext->setGraphicsVars(mpProgramVars);
    pSceneRenderer->renderScene(pRenderContext.get());
}

int WINAPI WinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_ LPSTR lpCmdLine, _In_ int nShowCmd)
{
    RenderScene::Unique pRenderer = std::make_unique<RenderScene>();
    SampleConfig config;
    config.windowDesc.title = "Falcon Model Viewer";
    Sample::run(config, pRenderer);
    return 0;
}
```cpp
#import Shading;
#import DefaultVS;

float4 main(VertexOut vOut) : SV_Target
{
    ShadingData sd = prepareShadingData(vOut, gMaterial, gCamera.posW);
    float3 color = 0;

    for (uint l = 0; l < gLightsCount; l++)
    {
        color += evalMaterial(sd, gLights[l], 1).color.rgb;
    }

    return float4(color, 1);
}
```
Good Raw-API Interop Is a Must
Days since last bug in motion vectors: 0
What Will Anjul Do?

1. Wait for Nir to come back from Whistler

2. Change Falcors’s internal code

3. Use Falcors’s raw-API interop constructs to call the library from his application
Falcor-GameWorks SDK Interop

```
// Get the API handles and prepare the resources
Texture::SharedPtr pSrc, pDst;
ID3D12ResourcePtr pSrcResource = pSrc->getApiHandle();
ID3D12ResourcePtr pDstResource = pDst->getApiHandle();
pContext->resourceBarrier(pDst, Resource::State::UnorderedAccess);

// Store the state
pContext->pushGraphicsState(pState);

// Call the library
ID3D12CommandListPtr pCmdList = pContext->getLowLevelData()->getCommandList();
GFSDK_DoSomething(pCmdList, pSrc, pDst);

// Restore the state
pContext->popGraphicsState();
pContext->invalidateGraphicsRootSignature();
pContext->bindDescriptorHeaps();
```
Good Raw-API Interop

- Getting it right is tricky

- Keep API-interop in mind if you don’t think you need it

- You can use new libraries without modifying the framework
Crank Shader-Reflection Up To 11
Let’s Start With Constant-Buffers
(And The Different Ways To Assign Them)
Method #1 - The Common Header

```c
#ifdef WIN32
    #define CBUFFER struct // CPP
#else
    #define CBUFFER cbuffer   // HLSL
#endif

CBUFFER PerFrame
{
    float4x4 matrix;
    float foo[4]; // This doesn't do what you think it does
};
```
Method #2 - The Offsets Reflection

```c
char* cbData = (char*)pCB->map();
int offset = getVariableOffset("foo");
(float*)(cbData + offset) = 3.0f;
pCB->unmap();
```

// This works, but:
// 1. No type checking on assignments
// 2. Cumbersome, many getVariableOffset() calls
// 3. A common optimization is to cache the offsets somewhere
// 4. This doesn't handle new/removed CB fields efficiently
// 5. Still need to manually allocate descriptor-sets and create root-signatures

// It's a good baseline which we can improve on
Method #3 - The Falcor Way

```cpp
void ModelViewer::onLoad(SampleCallbacks* pSample, RenderContext::SharedPtr pContext)
{
    mpProgram = GraphicsProgram::createFromFile("ModelViewer.ps.hls1", ",", "ps");
    mpProgramVars = GraphicsVars::create(mpProgram->getReflector());
}

void ModelViewer::onFrameRender(...) {
    mpProgramVars["PerFrameCB"["viewProjMat"] = mpCamera->getViewProjMatrix();
    mpProgramVars->setTexture("gMaterial.specTex", mpMaterial->getSpecularTexture());
    pContext->setGraphicsVars(mpProgramVars);
    // Bind the rest of the state and draw
}```
mpProgramVars["PerFrameCB"]->renderUI();
Visual Quality Matters
Getting Your Research From A Framework...
Into a Real Game Engine
Start with a basic asset
Test in on PBR Shader Balls
Scale it up
Falcon’s Modular Visual Quality Features

- Modern physically-based shading system
- Point-, spot-, directional- and area-lights
- Light-maps
- Light-probes
- High-quality shadows
- Skinned animations
- Cameras, meshes and lights motion-paths
- Post-processing techniques
- TAA, FXAA
- SSAO
- Particles system
- Integration with NVIDIA’s ShadowLib and HBAO+
Renderer Pixel Shader

```cpp
#include Shading;
#include DefaultVS;

float4 ps(VertexOut vOut, float4 pixelCrd : SV_POSITION) : SV_TARGET
{
    PsOut psOut;
    ShadingData sd = prepareShadingData(vOut, gMaterial, gCamera.posW);

    float3 color = 0;
    for (uint l = 0; l < gLightsCount; l++)
    {
        float shadowFactor = 1;
        shadowFactor = gVisibilityBuffer[l].Load(int3(vOut.posH.xy, 0)).r;  // Per-light shadow-buffer
        color.rgb += evalMaterial(sd, gLights[l], shadowFactor).color.rgb;    // Evaluate direct illumination
    }

    color.rgb += sd.emissive;                                                // Add the emissive component
    color.rgb += evalMaterial(sd, gLightProbe).color.rgb;                    // Eval light-probe
    color.rgb += sd.lightMap.rgb;                                            // Add light-map
    return float4(color, sd.opacity);
}
```
You Really Should Be Using A Render-Graph System
Our Falcor/DXR Based Framework. Ray traced and denoised glossy/diffuse (high quality settings).
Hybrid Raster/RT Renderer

Actual app has around 50(!) different passes to choose from
Think You Can Manage 50 Passes?

- Passes are constantly added/remove
- Data- and control-flow constantly changing
- Complex resource allocation and execution flow code
Forward-Renderer Anti-Aliasing Render-Targets Allocation Logic

```cpp
void ForwardRenderer::applyAAMode(ImageSampleCallbacks* pSample)
{
    if (mLightingProcess.pProgram == nullptr) return;

    mMAAmode = AAMode::MSAA; if (mMSASTargetCount > 1) mMAAmode = AAMode::IAA;

    GLuint a = pSample->getCurrentFramebuffer()->getWidth();
    GLuint b = pSample->getCurrentFramebuffer()->getHeight();

    // Common SRD desc (3 color outputs - color and normal)
    FboDesc fboDesc;
    fboDesc.setColorTarget(0, ResourceFormat::RGB4AlphaFloat);
    fboDesc.setColorTarget(1, ResourceFormat::RGB4AlphaNorm);
    fboDesc.setDepthStencilTarget(ResourceFormat::D32F);

    // Release the IAA fbo
    mMAA.resetFbo();

    if (mMAAmode == AAMode::IAA)
        fboDesc.setColorTarget(2, ResourceFormat::RS32F);

    taFboDesc.taaFboDesc;
    taFboDesc.setColorTarget(0, ResourceFormat::RGB4AlphaNorm);
    mMia.createFramebuffer(w, b, taaFboDesc);

    applyLightingProgramCurrent(SuperSampling);
    fboDesc.setSampleCount(mMAAmode == AAMode::MSAA ? mMSASTargetCount : 1);

    if (mMAAmode == AAMode::MSAA)
    {
        FboDesc resolveFbo;
        resolveFbo.setColorTarget(0, ResourceFormat::RGB4AlphaFloat);
        resolveFbo.setColorTarget(1, ResourceFormat::RGB4AlphaNorm);
        mpResolveFbo = FboHelper::create2D(w, h, resolveFbo);
    }
    
    else if (mMAAmode == AAMode::FXAA)
    {
        FboDesc resolveFbo;
        resolveFbo.setColorTarget(0, pSample->getcurrentFramebuffer()->getColorTexture()->getFormat());
        mpResolveFbo = FboHelper::create2D(w, h, resolveFbo);
    }

    mpMainFbo = FboHelper::create2D(w, h, fboDesc);
    mpDepthPassFbo = FboHelper::create2D();
    mpDepthPassFbo->setDepthYield(true);
    mpDepthPassFbo->setDepthStencilTarget(mpMainFbo->getDepthYieldTarget());

    if (mMAAmode != AAMode::MSAA)
    {
        mpResolveFbo = mpMainFbo;
    }
```
Render-Graph
There's a movie here.
Didn’t You Build an Engine?
A Library Of Rendering Lego Blocks

- Work in whatever abstraction level suits your needs
- Mix and match features however you like
- Access to the source code
- Targets researchers, not artists and level-designers
5 Things That Greatly Improve Our Research

- API-abstraction is essential
- Good raw-API interop is a must
- Crank shader-reflection up to 11
- Visual quality matters
- You really should be using a render-graph system
https://github.com/nvidiagameworks/falcor

https://developer.nvidia.com/orca

@falcor3d