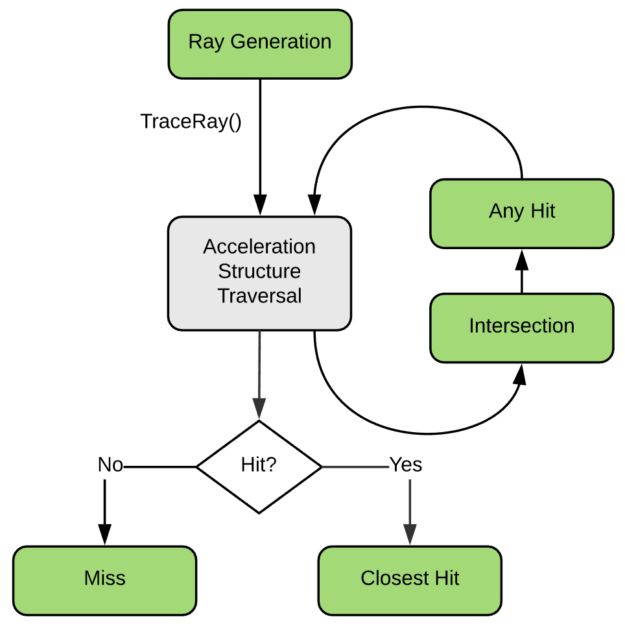
<https://devblogs.nvidia.com/introduction-nvidia-rtx-directx-ray-tracing/>



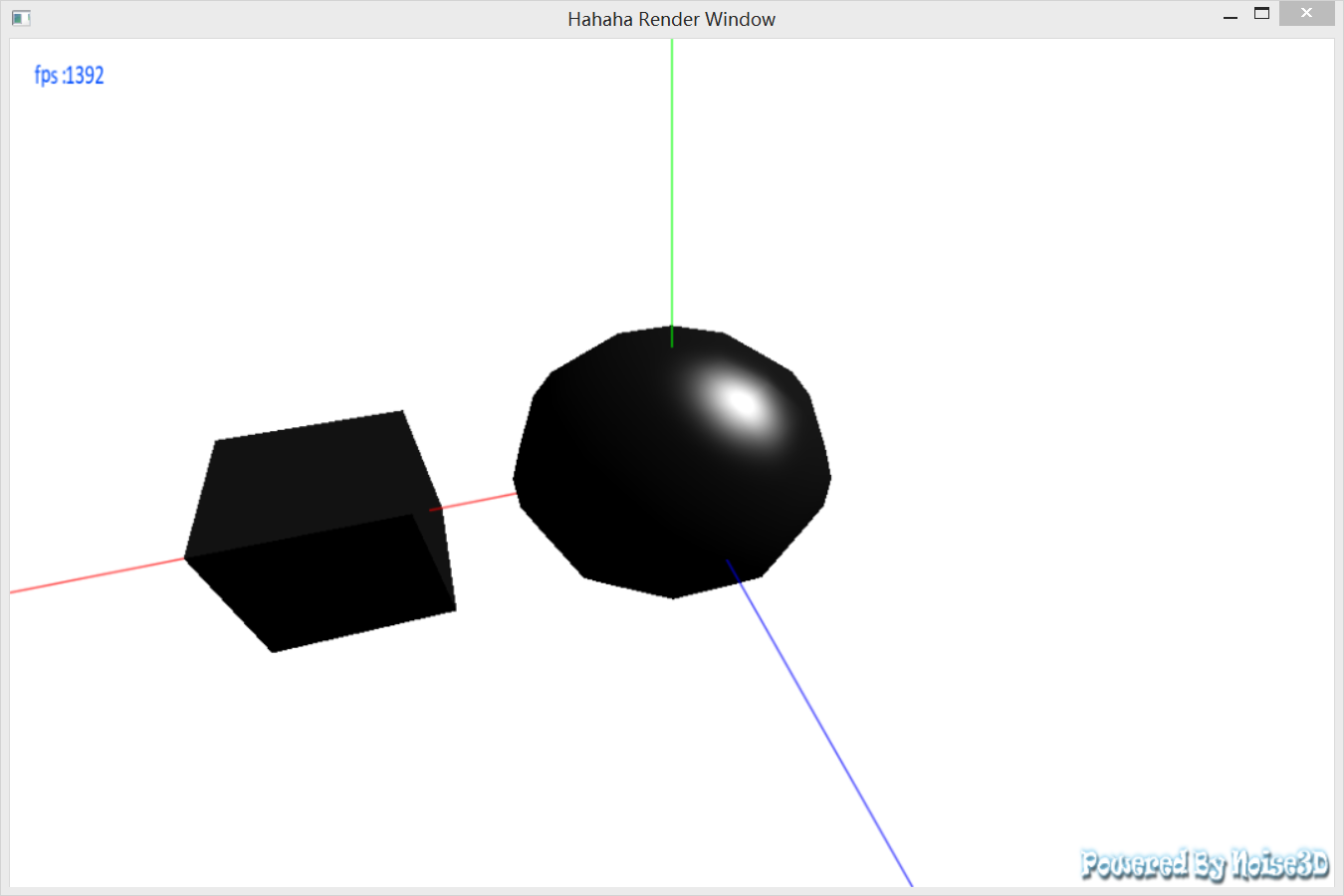
<https://developer.nvidia.com/rtx/raytracing/dxr/DX12-Raytracing-tutorial-Part-1>

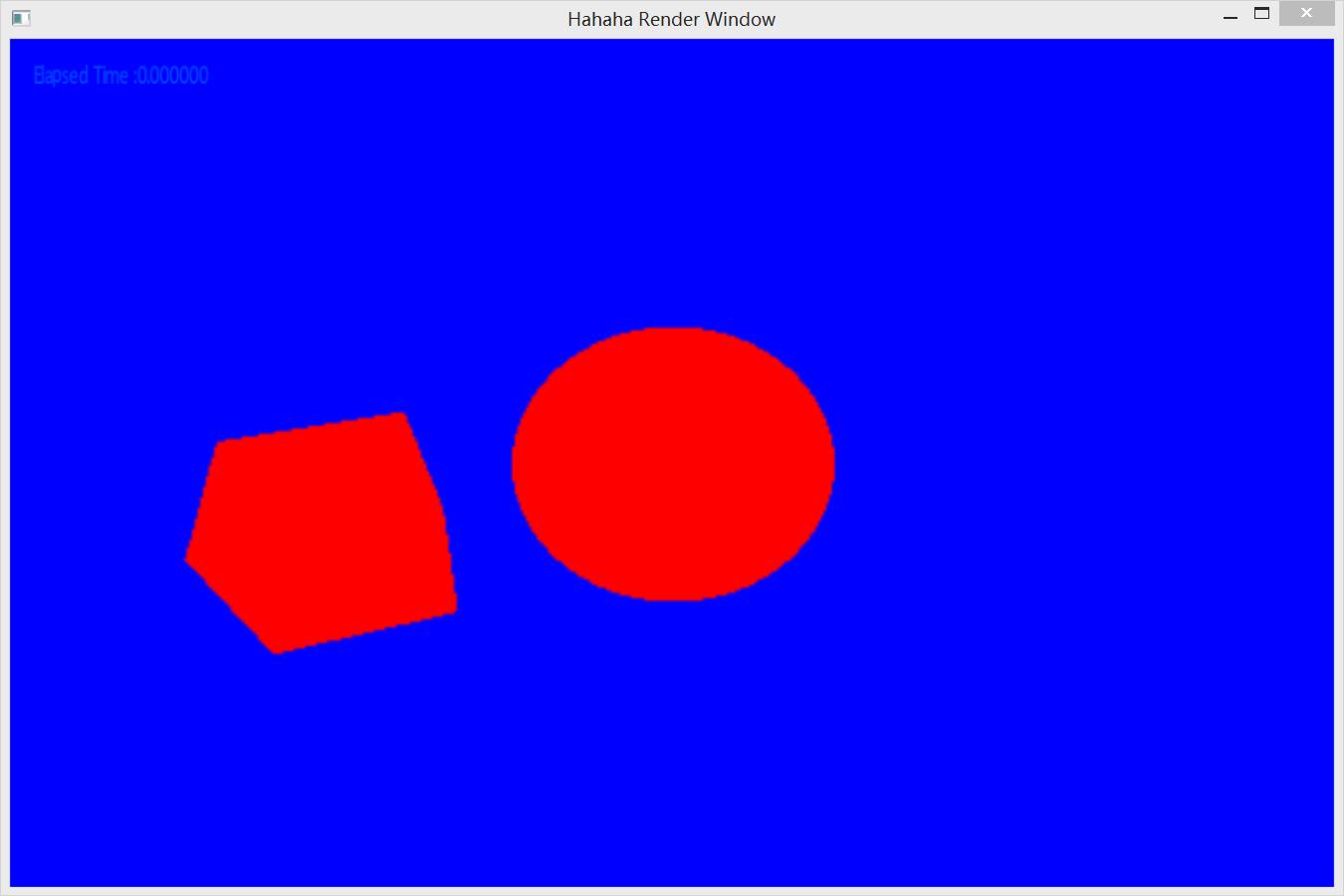
Render Tile：分块渲染，多线程并行，std::async/std::thread，具体看PathTracer.h/.cpp

SoftShaderInterface:

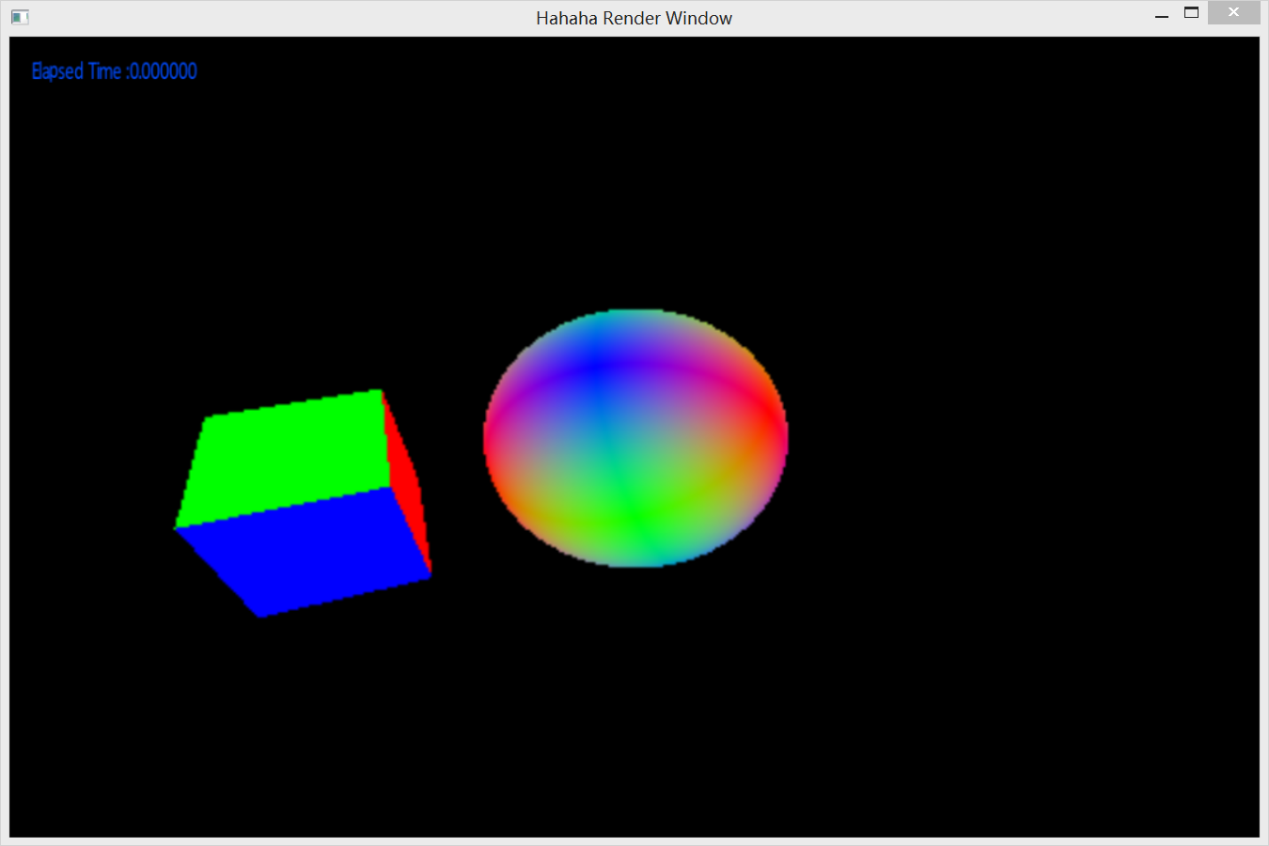
|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*                              GI: Soft shader interface          define some pure virtual function or something like that          to allow users to override the interface like programmable          graphics pipeline do. Then we can write c++ functions while          regarding them as 'soft shaders'.          (2019.3.31)in D3D12/DXR, there are 5 shaders in          ray tracing pipeline:          \*\*1. ray generation shader (impl in PathTracer class)          \*\*2. any hit shader (for alpha mask, but doesn't intend to impl)          \*\*3. intersection shader (impl in CollisionTestor)          \*\*4. closest hit shader          \*\*5. miss shader  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #pragma once  namespace Noise3D  {      namespace GI      {          class /\*\_declspec(dllexport)\*/ IPathTracerSoftShader          {          public:              IPathTracerSoftShader() :                  m\_pFatherPathTracer(nullptr) {}              virtual void ClosestHit(int bounces, float travelledDistance, const N\_Ray& ray, const N\_RayHitInfoForPathTracer& hitInfo, N\_TraceRayPayload& in\_out\_payload)=0;              //5. doesn't hit anything, might want to sample the skydome/skybox cubemap or sth              virtual void Miss(N\_Ray ray, N\_TraceRayPayload& in\_out\_payload)=0;          protected:              friend void PathTracer::Render(Noise3D::SceneNode\*, IPathTracerSoftShader\*);              void \_InitInfrastructure(PathTracer\* pt, CollisionTestor\* ct)              {                  m\_pFatherPathTracer = pt;                  m\_pCollisionTestor = ct;              }              void \_TraceRay(int bounces, float travelledDistance, const N\_Ray& ray, N\_TraceRayPayload& payload)              {                  //bounces and distance will be accumulated automatically in PathTracer::TraceRay()                  m\_pFatherPathTracer->TraceRay(bounces, travelledDistance, ray, payload);              }              uint32\_t \_MaxBounce()              {                  return m\_pFatherPathTracer->GetMaxBounces();              }              float \_MaxDistance()              {                  return m\_pFatherPathTracer->GetRayMaxTravelDist();              }          private:              PathTracer\* m\_pFatherPathTracer;              CollisionTestor\* m\_pCollisionTestor;          };      }  } |

**Noise3D Path Tracer Soft Shader Demo：**

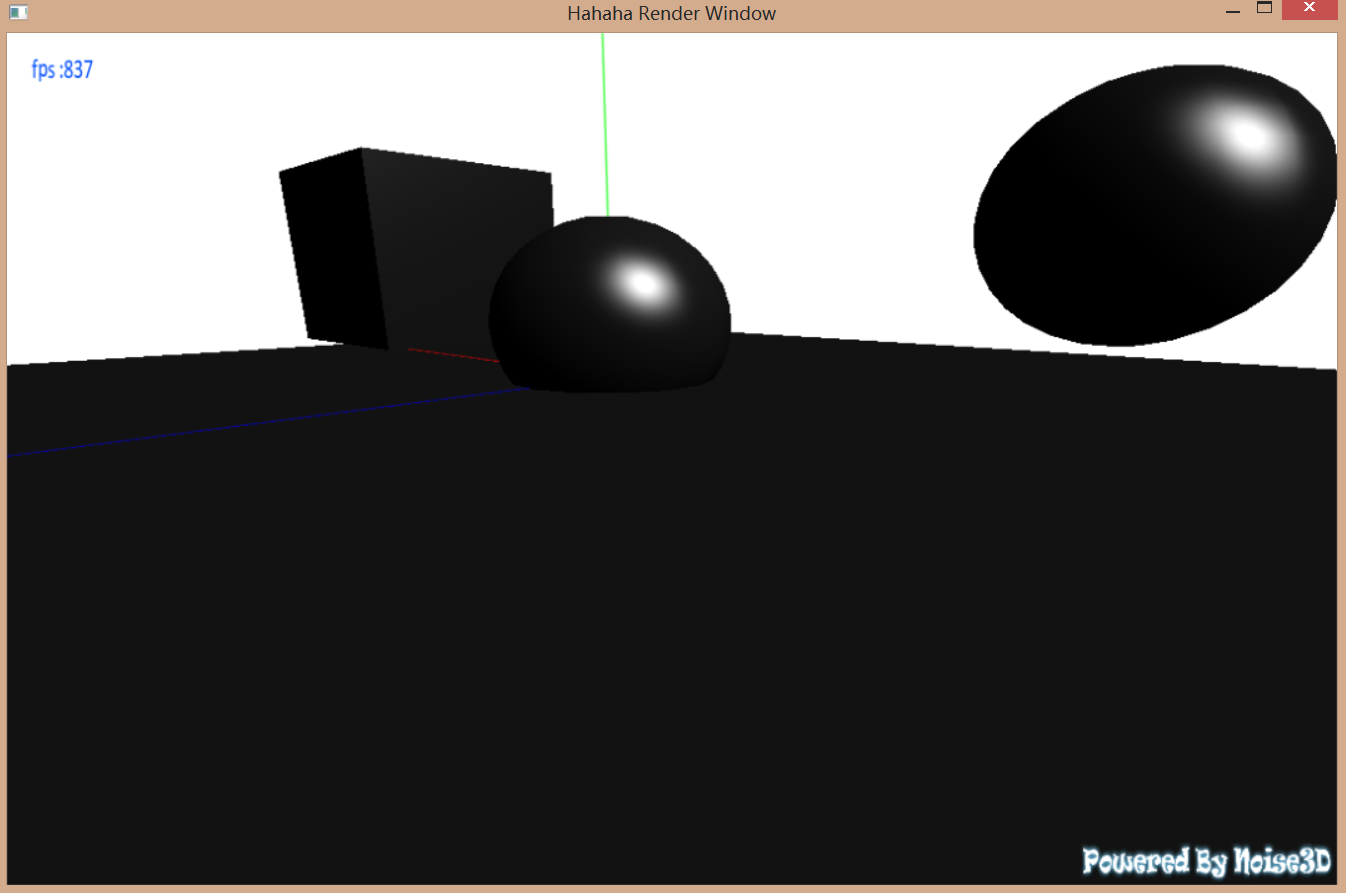




|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*                      Path Tracer Shader: Minimal              Noise3D::PathTracer's soft shader "hello world",              for demonstration and test usage  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "Noise3D.h"  using namespace Noise3D;  void Noise3D::GI::PathTracerShader\_Minimal::ClosestHit(N\_Ray ray, const N\_RayHitInfo & hitInfo, N\_TraceRayPayload & in\_out\_payload)  {      in\_out\_payload.radiance = GI::Radiance(1.0f, 0, 0);  }  void Noise3D::GI::PathTracerShader\_Minimal::Miss(N\_Ray ray, N\_TraceRayPayload & in\_out\_payload)  {      in\_out\_payload.radiance = GI::Radiance(0, 0, 1.0f);  } |



|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*                      Path Tracer Shader: Minimal              Noise3D::PathTracer's soft shader "hello world",              for demonstration and test usage  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "Noise3D.h"  using namespace Noise3D;  void Noise3D::GI::PathTracerShader\_Minimal::ClosestHit(N\_Ray ray, const N\_RayHitInfo & hitInfo, N\_TraceRayPayload & in\_out\_payload)  {      in\_out\_payload.radiance = GI::Radiance(abs(hitInfo.normal.x), abs(hitInfo.normal.y), abs(hitInfo.normal.z));  }  void Noise3D::GI::PathTracerShader\_Minimal::Miss(N\_Ray ray, N\_TraceRayPayload & in\_out\_payload)  {      in\_out\_payload.radiance = GI::Radiance(0, 0, 0);  } |



|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*                      Path Tracer Shader: Sky & Reflection              Noise3D::PathTracer's soft shader "hello world",              for demonstration and test usage  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "Noise3D.h"  using namespace Noise3D;  void Noise3D::GI::PathTracerShader\_Sky::SetSkyTexture(Texture2D \* pTex)  {      m\_pSkyDomeTexture = pTex;  }  void Noise3D::GI::PathTracerShader\_Sky::ClosestHit(int bounces, float travelledDistance, const N\_Ray& ray, const N\_RayHitInfoForPathTracer & hitInfo, N\_TraceRayPayload & in\_out\_payload)  {      Texture2dSampler\_Spherical sampler;      sampler.SetTexturePtr(m\_pSkyDomeTexture);      Vec3 reflectedDir = Vec3::Reflect(ray.dir, hitInfo.normal);      N\_Ray reflectedRay = N\_Ray(hitInfo.pos, reflectedDir);      //bounces and travelled distance is added automatically      N\_TraceRayPayload payload;      IPathTracerSoftShader::\_TraceRay(bounces, travelledDistance,reflectedRay, payload);      GI::Radiance reflectedColor = payload.radiance;      in\_out\_payload.radiance = reflectedColor;  }  void Noise3D::GI::PathTracerShader\_Sky::Miss(N\_Ray ray, N\_TraceRayPayload & in\_out\_payload)  {      Texture2dSampler\_Spherical sampler;      sampler.SetTexturePtr(m\_pSkyDomeTexture);      Color4f skyColor = sampler.Eval(ray.dir);      in\_out\_payload.radiance = GI::Radiance(skyColor.x, skyColor.y, skyColor.z);      //in\_out\_payload.radiance = GI::Radiance(0, 0, 0);  } |