Optix 7 shader intro

Empty Optix Skeleton

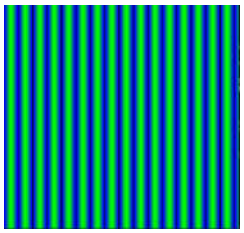
```
#include <optix.h>
#include "LaunchParams2.h" // our launch params
#include <vec_math.h> // NVIDIAs math utils

extern "C" {

	__constant__ LaunchParams optixLaunchParams;

	// closest hit program
	extern "C" __global__ void __closesthit__radiance() {

	}

	// anyhit hit program
	extern "C" __global__ void __anyhit__radiance() {

	}

	// miss program
	extern "C" __global__ void __miss__radiance() {

	}


	// ray gen program
	extern "C" __global__ void __raygen__renderFrame()  {

	}

} // end of extern
```

Struct with launch params

These functions must be defined for all ray types.
The prefix __function_kind__ is mandatory (optix).

Main function responsible for launching primary rays

Demo 0 – just draw something



```
extern "C" __global__ void __raygen__renderFrame()  {

    const uint3 index = optixGetLaunchIndex();

    // compute a test pattern based on pixel ID

    uint3 aux = index - optixGetLaunchDimensions() * 0.5;
    float2 pixelCenteredCoord = make_float2(aux.x, aux.y);
    float pixelIntensity = 0.5 + 0.5 * cos((pixelCenteredCoord.x + optixLaunchParams.
frame.frame) * 0.1f);

    const int r = 0;
    const int g = (pixelIntensity * 255);
    const int b = (1 - pixelIntensity * 255);

    // convert to 32-bit rgba value - explicitly set alpha to 0xff
    const uint32_t rgba = 0xff000000  | (r<<0) | (g<<8) | (b<<16);

    const unsigned int fbIndex = index.x + (index.y * optixGetLaunchDimensions().x);

    if (optixLaunchParams.frame.frame == 0 && index.x == 0 && index.y == 0) {

        // print info to console
        printf("=============================================\n");
        printf("Nau Ray-Tracing Hello World\n");
        printf("Lunch size: %i x %i\n", index.x, index.y);
        printf("Camera Direction: %f %f %f\n",
                optixLaunchParams.camera.direction.x,
                optixLaunchParams.camera.direction.y,
                optixLaunchParams.camera.direction.z
            );
        printf("=============================================\n");
    }

    optixLaunchParams.frame.colorBuffer[fbIndex] = rgba;
}
```

LaunchIndex indica qual a instância que está a correr (uint3)

LaunchDimensions indica qual o tamanho do buffer (uint3)

Cálculo de intensidade baseado na posição do pixel e num coseno. A imagem é animada devido ao uso do número da frame

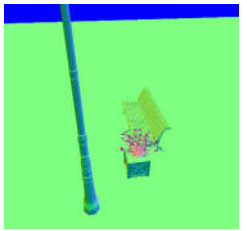
Cálculo da cor (baseada na intensidade) e conversão para uint32

Cálculo do index do buffer (corresponde ao index do pixel que queremos escrever)

Pode-se usar printf dentro de um shader, útil para debug. Atenção à guarda no if, para evitar escrever mais que uma vez

Escrita no output buffer (accessivel a partir do OpenGL como sendo uma textura

Demo 1 – Render geometry (1)

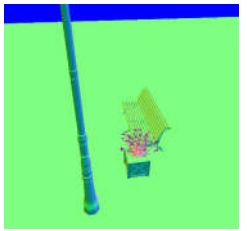


```
// ray gen program - responsible for launching primary rays
extern "C" __global__ void __raygen__renderFrame() {

    // compute a test pattern based on pixel ID
    const int ix = optixGetLaunchIndex().x;
    const int iy = optixGetLaunchIndex().y;
    const auto &camera = optixLaunchParams.camera;

    // ray payload
    float3 pixelColorPRD = make_float3(1.f);
    uint32_t u0, u1;
    packPointer( &pixelColorPRD, u0, u1 );

    // compute ray direction
    // normalized screen plane position, in [-1, 1]^2
    const float2 screen(make_float2(ix+.5f,iy+.5f)
                / make_float2(optixGetLaunchDimensions().x, optixGetLaunchDimensions().y) * 2.0 - 1.0);

    // note: nau already takes into account the field of view when computing
    // camera horizontal and vertival
    float3 rayDir = normalize(camera.direction
                        + screen.x  * camera.horizontal
                        + screen.y * camera.vertical);


     . . .
```

Preparar payload

Calcular direção do raio primário

Demo 1 – Render geometry (2)

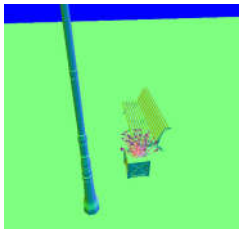


```
    . . .
    // trace primary ray
    optixTrace(optixLaunchParams.traversable,
             camera.position,
             rayDir,
             0.f,    // tmin
             1e20f,  // tmax
             0.0f,   // rayTime
             OptixVisibilityMask( 255 ),
             OPTIX_RAY_FLAG_DISABLE_ANYHIT,//OPTIX_RAY_FLAG_NONE,
             SURFACE_RAY_TYPE,             // SBT offset
             RAY_TYPE_COUNT,               // SBT stride
             SURFACE_RAY_TYPE,             // missSBTIndex
             u0, u1 );

    //convert float (0-1) to int (0-255)
    const int r = int(255.0f*pixelColorPRD.x);
    const int g = int(255.0f*pixelColorPRD.y);
    const int b = int(255.0f*pixelColorPRD.z);

    // convert to 32-bit rgba value
    const uint32_t rgba = 0xff000000
      | (r<<0) | (g<<8) | (b<<16);
    // compute index
    const uint32_t fbIndex = ix+iy*optixGetLaunchDimensions().x;
    // write to output buffer
    optixLaunchParams.frame.colorBuffer[fbIndex] = rgba;
}
```

Lançar raio primário

Resultado vem no payload, ou seja: pixelColor

Demo 1 – Render geometry (3)



```
extern "C" __global__ void __closesthit__radiance() {

    const TriangleMeshSBTData &sbtData
      = *(const TriangleMeshSBTData*)optixGetSbtDataPointer();

    // compute triangle normal:
    const int   primID = optixGetPrimitiveIndex();
    const uint3 index  = sbtData.index[primID];
    const float3 &A     = make_float3(sbtData.vertexD.position[index.x]);
    const float3 &B     = make_float3(sbtData.vertexD.position[index.y]);
    const float3 &C     = make_float3(sbtData.vertexD.position[index.z]);
    const float3 Ng     = normalize(cross(B-A,C-A)) * 0.5 + 0.5;

    float3 &prd = *(float3*)getPRD<float3>();
    prd = Ng;
}
```

Aceder aos dados dos vertices e materiais da mesh (dados passados pela Nau)

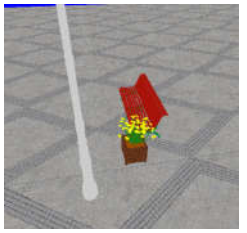
Aceder ao índice da primitive intersectada

Extrair os três vertices do triângulo

Cálcular a normal e comprimir para [0,1]^3

Preparar o payload de retorno

Demo 2 – Cores e Texturas(1)



```
extern "C" __global__ void __closesthit__radiance()
{
    const TriangleMeshSBTData &sbtData
      = *(const TriangleMeshSBTData*)optixGetSbtDataPointer();
    // compute triangle normal:
    const int   primID = optixGetPrimitiveIndex();

    float3 &prd = *(float3*)getPRD<float3>();

    if (sbtData.hasTexture && sbtData.vertexD.texCoord0) {
        // get barycentric coordinates
        const float u = optixGetTriangleBarycentrics().x;
        const float v = optixGetTriangleBarycentrics().y;
        // compute pixel texture coordinate
        const float4 tc
          = (1.f-u-v) * sbtData.vertexD.texCoord0[index.x]
          +         u * sbtData.vertexD.texCoord0[index.y]
          +         v * sbtData.vertexD.texCoord0[index.z];
        // fetch texture value
        float4 fromTexture = tex2D<float4>(sbtData.texture,tc.x,tc.y);
        prd= make_float3(fromTexture);
    }
    else
        prd = sbtData.color;
}
```

Verificar se o material tem textura e coordenadas de textura

Obter coordenadas baricêntricas do ponto de intersecção

Calcular coordenada de textura

Aceder à textura

Compiling CUDA files

- Using nvcc (NVIDIA C compiler)

- `nvcc.exe -O3 -use_fast_math -arch=compute_30 -code=sm_30 -I "PATH_TO_NVIDIA_OPTIX_INCLUDE_DIR" -I "PATH_TO_VS_C_INCLUDE_DIR" -I "." -m 64 -ptx -ccbin "PATH_TO_VS_VC_64_COMPILER_CL.EXE" input.cu -o output.ptx`

- Example for standard VS2019 and Optix 7 folders

- `nvcc.exe -O3 -use_fast_math -arch=compute_30 -code=sm_30 -I "C:\ProgramData\NVIDIA Corporation\OptiX SDK 7.0.0\include" -I "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.23.28105\include" -I "." -m 64 -ptx -ccbin "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.23.28105\bin\Hostx64\x64" optixTriangle.cu -o optixTriangle.ptx`

Launch Params

Estrutura de dados passada ao Optix todas as frames:

- Neste momento não é configurável.
- Output do Optix é uma textura
  - Não testado: múltiplos render targets?

```
struct LaunchParams
{
  struct {
    int frame; // frame count
    uint32_t *colorBuffer; // color buffers
    int2     size; // dimensão do buffer
    int raysPerPixel; //
  } frame;

  struct { // camera data
    float3 position;
    float3 direction;
    float3 horizontal;
    float3 vertical;
  } camera;
  // optix opaque struct
  OptixTraversableHandle traversable;
};
```

Programs Data

```
struct vertexData {
    float4* position;
    float4* normal;
    float4* texCoord0;
    float4* tangent;
    float4* bitangent;
};

struct TriangleMeshSBTData {
    uint3 *index;
    vertexData vertexD;
    int hasTexture;
    cudaTextureObject_t texture;
    float3  color;
};
```

```
struct RayGenData {
  int3 color;
};
```

Dados passados ao raygen
(só para fins de teste)

Dados passados aos programas “any hit” e “closest hit”

Projecto Nau – definição de um passo RT

```
<pass class="rt" name="pass1">
    <scenes>
        <scene name="MainScene" />
    </scenes>
    <camera name="MainCamera" />
    <renderTarget name="test" fromLibrary="Optix Ray Tracer Render Target" />
    <lights>
        <light name="Sun" />
    </lights>

    <rtRayTypes>
        <rayType name="Phong"/>
    </rtRayTypes>

    <rtVertexAttributes>
        <attribute name="position"/>
        <attribute name="normal"/>
        <attribute name="texCoord0"/>
    </rtVertexAttributes>

    <rtEntryPoint>
        <rayGen file="optix/testOptix2.ptx" proc="__raygen__renderFrame"/>
    </rtEntryPoint>
    <rtDefaultMaterial>
        <rayType name="Phong">
            <rtProgram type="ANY_HIT" file="optix/testOptix2.ptx" proc="__anyhit__radiance"/>
            <rtProgram type="CLOSEST_HIT" file="optix/testOptix2.ptx" proc="__closesthit__radiance"/>
            <rtProgram type="MISS"       file="optix/testOptix2.ptx" proc="__miss__radiance"/>
        </rayType>
    </rtDefaultMaterial>
</pass>
```

Links

- How to get started with Optix7
  - https://devblogs.nvidia.com/how-to-get-started-with-optix-7/
- Ingo Wald Optix7 course
  - https://gitlab.com/ingowald/optix7course
- Optix API documentation
  - https://raytracing-docs.nvidia.com/optix7/api/html/index.html
- Cuda Toolkit documentation
  - https://docs.nvidia.com/cuda/cuda-runtime-api/index.html