**Chapter 1 DirectX11**

**1.1 シェーダーモデル5**

DirectX9のシェーダーステージは頂点シェーダーとピクセルシェーダーのみでした。しかし最新のシェーダーモデル5ではジオメトリシェーダー、ハルシェーダー、ドメインシェーダー、コンピュートシェーダーの４つのシェーダーステージが追加されています。

**1.2 Windows8SDKへの統合とD3DXの廃止**

DirectXはWindows8からWindows8SDKへ統合され、DirectXSDKは廃止されました。また、この統合の際にD3DXは廃止され、D3DXから始まるAPIはすべてなくなっています。(つまりD3DXVECTOR3、D3DXMATRIX、D3DXVec3Lengthなどの関数はなくなっています！！！)

**Chapter 2 GPGPU**

**2.1 GPGPUの始まり**

GPGPUとはGeneral-purpose computing on graphics processing unitsの略でGPUに汎用計算を行わせることを指しています。汎用計算とは画像処理以外の処理のことを指します。例えばコリジョン処理など。

本来GPUは画像処理を行うために進化してきましたが、プログラマブルシェーダーの登場により、GPUでそれなりに複雑なロジックを実行することができるようになりました。また、CPUはGPUとは異なる進化を遂げてきたため、単純な計算であればCPUとは比べ物にならない速度で処理することができます。そのため、GPUに画像処理だけさせるのは勿体ない、もっと色々な処理をGPUに実行させよう！という考えが生まれます。これがGPGPUの始まりです。

**2.2 コンピュートシェーダー**

DirectX9の時代はGPUで行った計算の結果はテクスチャに対してしか出力することができません。また、両隣の計算結果を参照して処理を行うということも１パスでは行うことができません。このようにDirectX9でGPGPUを行う場合は多くの制限を受けていました。そこでDirectX10ではより高度なGPGPUを行うことができるように、コンピュートシェーダーという新しいシェーダーステージが追加されました。

コンピュートシェーダーはhlslで記述することができ、より柔軟に計算結果を出力することができるようになっています。

**2.3 データの入力と出力**

　多くの処理ではいろいろな計算を行うためにはデータを入力する必要があります。そしてその入力に対する計算結果を出力する必要もあるでしょう。クラスの平均点を求める関数であれば、下記のような実装になると思います。

**int** CalcAvarage( **int numStudent, int\* scoreArray** )

{

出力

入力

int totalScore = 0;

for( int I = 0; I < numStudent; i++ ){

totalScore += scoreArray[i];

}

return totalScore / numStudent;

}

このクラスの平均点を求める計算をGPUに行わせる場合も同様にGPUに対して生徒数と点数の配列を送ってやる必要があります。そして、GPUで行った計算結果を出力する必要があります。

**2.4 メモリ**

前節で示したように、GPGPUを行うためにはGPUへのデータを入力と、GPUからのデータの出力を行う必要があります。ではどのようにしてデータの入力、出力を行うのでしょうか？これを理解するためにはGPUとCPUとメモリについて学んでいく必要があります。

**2.4.1 VRAM**

少しPCに詳しい人であればVRAMという言葉を聞いたことがあると思います。VRAMとはGPUがアクセスできるメモリです。今まで君たちがC++で扱ってきたメモリはCPUがアクセスできるメモリ(メインメモリ)です。GPUがアクセスできるメモリはVRAMだけなので、テクスチャ、頂点バッファ、インデックスバッファ、そして定数レジスタなどのデータはすべてVRAMに載っている必要があります。今までDirextX9のAPIを使用してテクスチャや頂点バッファなどの作成を行っていたと思いますが、実はこれらのメモリはVRAM上で確保されていました。なので、GPGPUを行うための入力データはメインメモリからVRAMに転送する必要があります。DirectX9時代の定数レジスタをイメージするとわかりやすいでしょう。

　では出力データはどうなるのでしょうか？GPUから書き込みができるメモリはVRAMだけです。GPUで行った計算結果をCPUで使用するためには、VRAMからメインメモリに転送する必要があります。

**2.4.2 ユニファイドメモリ**

　さて、前節ではGPUがアクセスできるメモリはVRAMでCPUがアクセスできるメモリはメインメモリだと言いました。しかし、ユニファイドメモリを採用しているアーキテクチャではGPUとCPUが同じメモリ空間にアクセスすることができます。PS4はユニファイドメモリです。ユニファイドメモリのアーキテクチャではVRAMにデータを転送する必要はなく、直接メモリアドレスを指定するだけでよくなるため、実装がシンプルになります。

**2.4.3 DirectX11でのメモリの扱い**

　DirectX11でユニファイドメモリの場合とそうでない場合の両方のコードを記述することができます。しかし、ユニファイドメモリ前提のコードを記述した場合、そうではない場合に意図しない挙動をすることになるので、PCのようなユーザーによって多様なアーキテクチャが存在する環境で開発を行うのであれば、ユニファイドメモリではない場合のコードを記述するのが無難です。ですので、これ以降はメモリ空間が別になっているものとして話を進めていきます。

2**.5 GPUで平均点を計算するサンプルプログラム。**

ではGPUで平均点を計算するサンプルプログラムを見ていきましょう。

Tutorial01.cppのCComputeTest::Start関数で初期化を行っています。まず、コンピュートシェーダーをロードします。

|  |
| --- |
| //コンピュートシェーダーをロード。  m\_csShader.Load("Assets/shader/BasicCompute11.fx", "CSMain", CShader::EnType::CS); |

VRAM上に生徒の平均点を転送するためのメモリを確保します。

|  |
| --- |
| D3D11\_BUFFER\_DESC desc;  ZeroMemory(&desc, sizeof(desc));  //SRVとしてバインド可能。  desc.BindFlags = D3D11\_BIND\_SHADER\_RESOURCE; //SRVとしてバインド可能。  desc.ByteWidth = NUM\_STUDENT \* sizeof(int);  desc.MiscFlags = D3D11\_RESOURCE\_MISC\_BUFFER\_STRUCTURED;  desc.StructureByteStride = sizeof(int);  //StructuredBufferを作成。VRAM上にメモリを確保して入力データを転送。  m\_inputBuffer.Create(m\_score, desc); |

VRAM上に出力用のメモリを確保します。

|  |
| --- |
| D3D11\_BUFFER\_DESC desc;  ZeroMemory(&desc, sizeof(desc));  desc.BindFlags = D3D11\_BIND\_UNORDERED\_ACCESS; //UAVとしてバインド可能。  desc.ByteWidth = sizeof(int);  desc.MiscFlags = D3D11\_RESOURCE\_MISC\_BUFFER\_STRUCTURED;  desc.StructureByteStride = sizeof(int);  //StructuredBufferを作成。VRAM上に出力用のメモリを確保する。  m\_outputBuffer.Create(NULL, desc); |

出力結果をCPUで見るためにメインメモリ上にメモリを確保します。

|  |
| --- |
| D3D11\_BUFFER\_DESC desc;  ZeroMemory(&desc, sizeof(desc));  desc.CPUAccessFlags = D3D11\_CPU\_ACCESS\_READ; //CPUから読み込み可能に設定する。  desc.Usage = D3D11\_USAGE\_STAGING; //GPUからCPUへのデータコピーをサポートする。  desc.BindFlags = 0; //どこにもバインドしない。  desc.MiscFlags = 0;  desc.ByteWidth = sizeof(int);  desc.StructureByteStride = sizeof(int);  desc.MiscFlags = D3D11\_RESOURCE\_MISC\_BUFFER\_STRUCTURED;  //StructuredBufferを作成。メインメモリ上に作成する。  m\_outputBufferCPU.Create(NULL, desc); |

続いて、コンピュートシェーダーの実行を見ていきましょう。CComputeTest::Render関数で毎フレームコンピュートシェーダーを実行しています。

|  |
| --- |
| //コンピュートシェーダーを設定。  renderContext.CSSetShader(m\_csShader);  //入力SRVを設定。  renderContext.CSSetShaderResource(0, m\_inputSRV\_0);  //出力UAVを設定。  renderContext.CSSetUnorderedAccessView(0, m\_outputUAV);  //コンピュートシェーダーを実行。  renderContext.Dispatch(1, 1, 1);  //CPUからアクセスできるバッファにコピー。  renderContext.CopyResource(m\_outputBufferCPU, m\_outputBuffer);  //コンピュートシェーダーの結果を取得。  CMapper<CStructuredBuffer> mapper(renderContext, m\_outputBufferCPU);  int\* p = (int\*)mapper.GetData();  if (p) {  TK\_LOG("平均点 %d ", \*p);  } |

ではコンピュートシェーダーを見てみましょう。Assets/shader/BasicCompute11.fxを開いてください。

|  |
| --- |
| //生徒の数。  #define NUM\_STUDENT 30  //入力データ。  StructuredBuffer<int> g\_score : register(t0);  //出力データ。  RWStructuredBuffer<int> g\_avarage : register(u0);  //コンピュートシェーダのエントリ関数。  [numthreads(1, 1, 1)]  void CSMain( uint3 DTid : SV\_DispatchThreadID )  {  //まず合計点を求める。  int totalScore = 0;  for(int i = 0; i < NUM\_STUDENT; i++ ){  totalScore += g\_score[i];  }  //平均点を出力。  g\_avarage[0] = totalScore / NUM\_STUDENT;  } |

g\_scoreが点数のリスト、g\_avarageが平均点の出力先です。