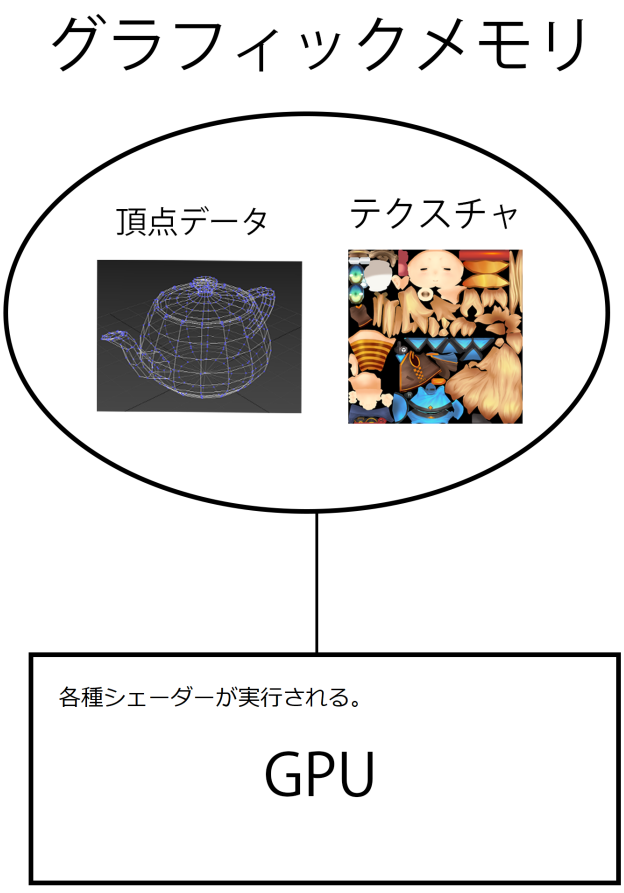
Chapter1 DirectX12のリソースバインディング

　このチャプターではDirectX12におけるリソースバインディングの仕組みについて見てきます。

1.1 リソースバインディングとは？

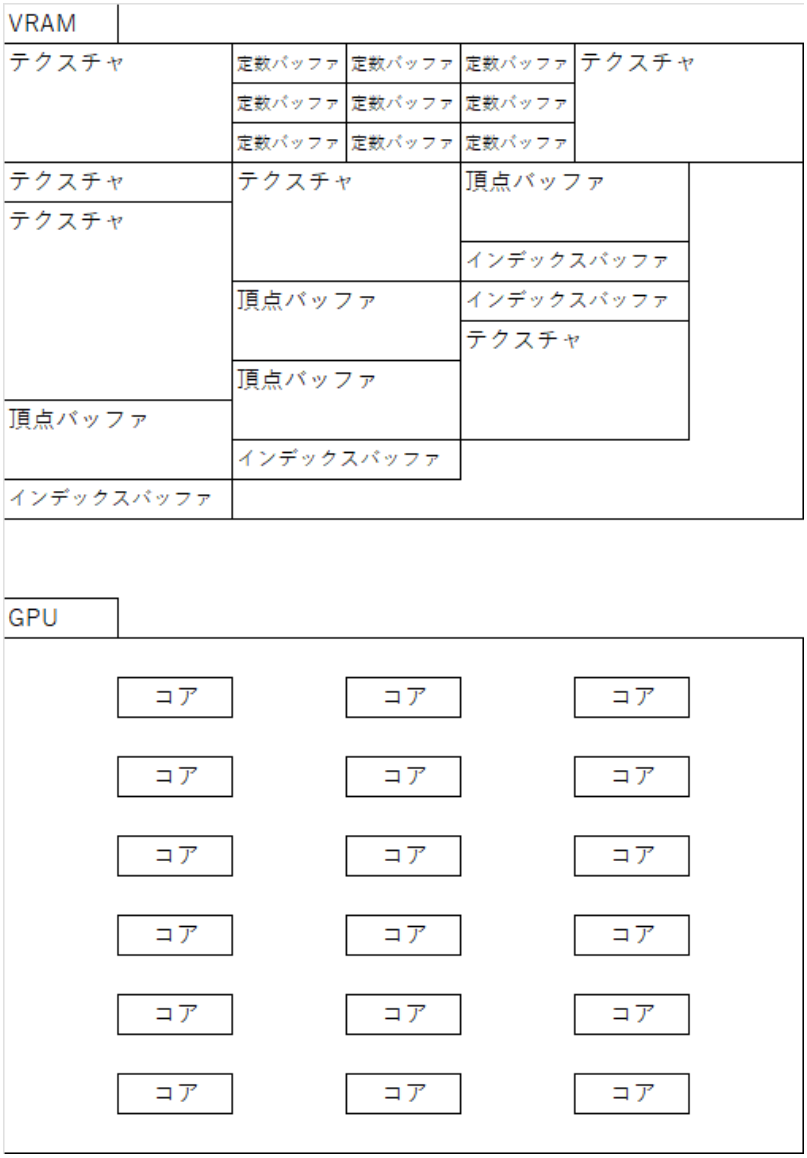
　では、リソースバインディングについて解説をしたいのですが、これを解説するには昨今のGPUのアーキテクチャについて多少の前提知識が必要になるので、それについて勉強していきましょう。　昨今の高品質なゲームを快適に遊ぶためにはグラフィックカードが必須となります。現在のグラフィックカードは主にNVIDIA製のものとAMD製のモノがあります。(IntelのCPUに内蔵されているUHDグラフィックスなどのインテル製のチップもありますが、このチップはあくまでも一般ユーザー向けのチップとなっており、高品質なゲームを遊びたいゲームユーザー向けではないため、割愛します。)　これらのグラフィックカードの構成を大雑把に説明すると、演算を行うGPU、テクスチャなどのデータを記憶するVRAMに分類することができます(図1.1)。

[図1.1 GPUとVRAM～その1]



　さて、グラフィックカードを大雑把に見てみるとGPUとVRAMがあることが分かりました。では、もう少し詳細にGPUのアーキテクチャについて見ていきましょう。図1.2を見てください。

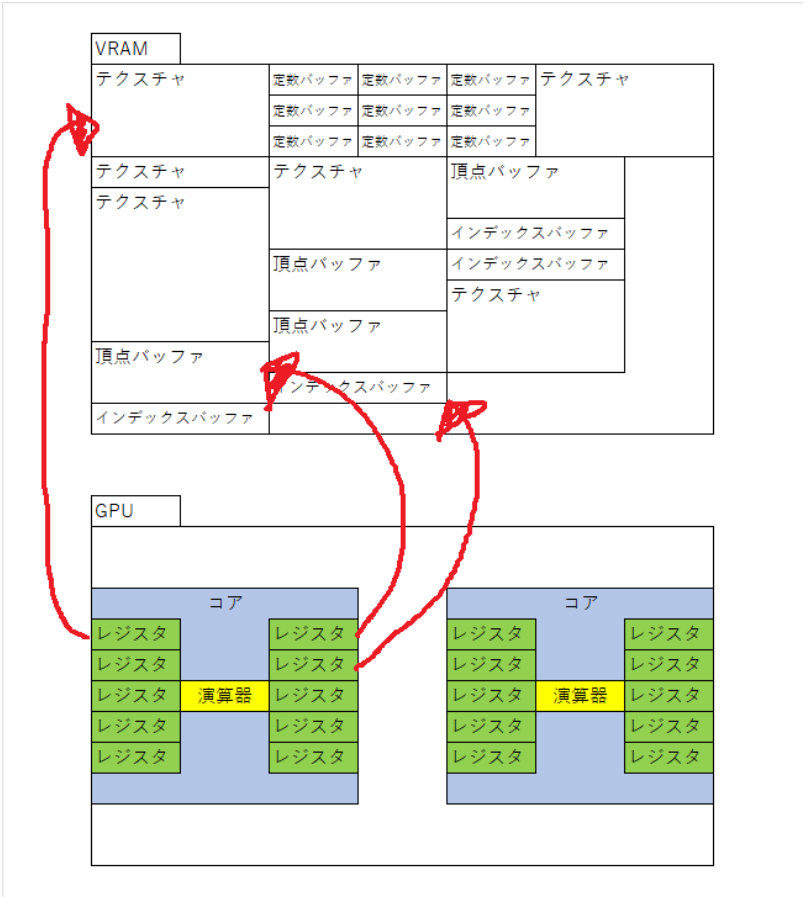
[図1.2 GPUとVRAM～その2]



GPUの内部には多数のコアと呼ばれる演算基が含まれており、前述したシェーダープログラムの実行はこのコアを使って並列に実行されます。例えば、100万頂点のモデルの描画コールを実行した場合、このコアで分担して100万頂点の頂点シェーダーを実行していくことになります。また、VRAMにはシーンを描画するために必要な各種リソースが乗っています。さて、そろそろリソースバインディングに関する話に近づいているのですが、もう少しだけグラフィックカードのアーキテクチャを詳細に見ていきましょう。図1.3を見てください。

[図1.3 GPUとVRAM～その3]

図1.3ではコアの内部情報を詳細に記載しています。コアの内部には数値演算を行うための演算器と高速なメモリのレジスタがあります。この演算器でシェーダープログラムを実行していると考えてください。ここで重要なのはレジスタです。シェーダープログラムが直接アクセスできるメモリはこのレジスタに乗っているデータになります。さて、ここで少しGPUの動きを考えてみましょう。アプリケーション側からユニティちゃんを描画するためのドローコールが実行されるとGPUはユニティちゃんを描画するためにレンダリングパイプラインを実行していきます。このパイプライン上に頂点処理を行う頂点シェーダーやピクセル処理を行うピクセルシェーダーが実行されていくわけです。この時、頂点シェーダーでは当然VRAMに乗っているユニティちゃんの頂点バッファやインデックスバッファにアクセスする必要があります。ピクセルシェーダーではユニティちゃんのテクスチャにアクセスる必要があります。しかし、VRAMにはその他のオブジェクトの頂点バッファやテクスチャといったグラフィックリソースもVRAMに乗っているのです。では、GPUはどのようにして多数あるVRAM上のリソースから、Unityちゃんのグラフィックリソースを選べばいいのでしょうか。答えを言うと、GPUが自動的にユニティちゃんのグラフィックリソースを選ぶということはできません。プログラマが明示的にプログラムを記載して、ユニティちゃんのグラフィックリソースを指定する必要があります。もう少し具体的に言うと、GPUの各種レジスタに使用するグラフィックリソースのアドレスを設定するのです(図1.4)。 [図1.4 GPUとVRAM～その4]



イメージとしてはC++のポインタをイメージしてみてください。レジスタに使用するリソースのアドレスを設定して、そのアドレスを使ってリソースをロードしていきます。そしてこのレジスタと使用するリソースを関連付けすることをリソースバインディングといいます。DirectX12を利用して絵を描画するためには、後述するディスクリプタ、ディスクリプタヒープ、そしてルートシグネチャを活用して、描画したい絵に必要なリソースとレジスタの関連付けを行う必要があります。

1.1 ディスクリプタ

1.2 ディスクリプタヒープ

1.3 ルートシグネチャとディスクリプタテーブル