Chapter 1

プログラマブルシェーダーとは

1.1 固定機能

シェーダーが生まれる前、DirectX9 までは固定機能パイプラインというものが存在していました。この固定機能パイプラインは DirectX10 で削除され、それ以降固定機能パイプラインは用意されなくなっています。これは OpenGL、OpenGLES、Sony や任天堂などが提供する専用 SDK(PS4、PS3、WiiU などで使用できる DirectX のようなもの)でも同じで固定機能はグラフィックプログラミングの世界では過去のものとなっています。では固定機能と呼ばれるものがどのようなものか見ていきましょう。下記は固定機能を使って 3D ポリゴンを表示しているコードです。

```
//-----------------------------------------------------------------------------
// ワールド*ビュー*プロジェクション行列を設定。
//-----------------------------------------------------------------------------
void SetupMatrices()
{
    // ワールド行列を設定。
    D3DXMATRIXA16 matWorld;
    D3DXMatrixIdentity( &matWorld );
    D3DXMatrixRotationX( &matWorld, timeGetTime() / 500.0f );
    g_pd3dDevice->SetTransform( D3DTS_WORLD, &matWorld );

    // ビュー行列を設定。
    D3DXVECTOR3 vEyePt( 0.0f, 3.0f,-5.0f );
    D3DXVECTOR3 vLookatPt( 0.0f, 0.0f, 0.0f );
    D3DXVECTOR3 vUpVec( 0.0f, 1.0f, 0.0f );
    D3DXMATRIXA16 matView;
    D3DXMatrixLookAtLH( &matView, &vEyePt, &vLookatPt, &vUpVec );
    g_pd3dDevice->SetTransform( D3DTS_VIEW, &matView );

    // プロジェクション行列を設定。
    D3DXMATRIXA16 matProj;
    D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI / 4, 1.0f, 1.0f, 100.0f );
    g_pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
}
//-----------------------------------------------------------------------------
// ライトを設定。
//-----------------------------------------------------------------------------
void SetupLights()
{
        //マテリアルを設定。
    D3DMATERIAL9 mtrl;
    ZeroMemory( &mtrl, sizeof( D3DMATERIAL9 ) );
    mtrl.Diffuse.r = mtrl.Ambient.r = 1.0f;
    mtrl.Diffuse.g = mtrl.Ambient.g = 1.0f;
    mtrl.Diffuse.b = mtrl.Ambient.b = 0.0f;
    mtrl.Diffuse.a = mtrl.Ambient.a = 1.0f;
    g_pd3dDevice->SetMaterial( &mtrl );

    // ディフューズライトの向きとカラーを設定。
    D3DXVECTOR3 vecDir;
    D3DLIGHT9 light;
```

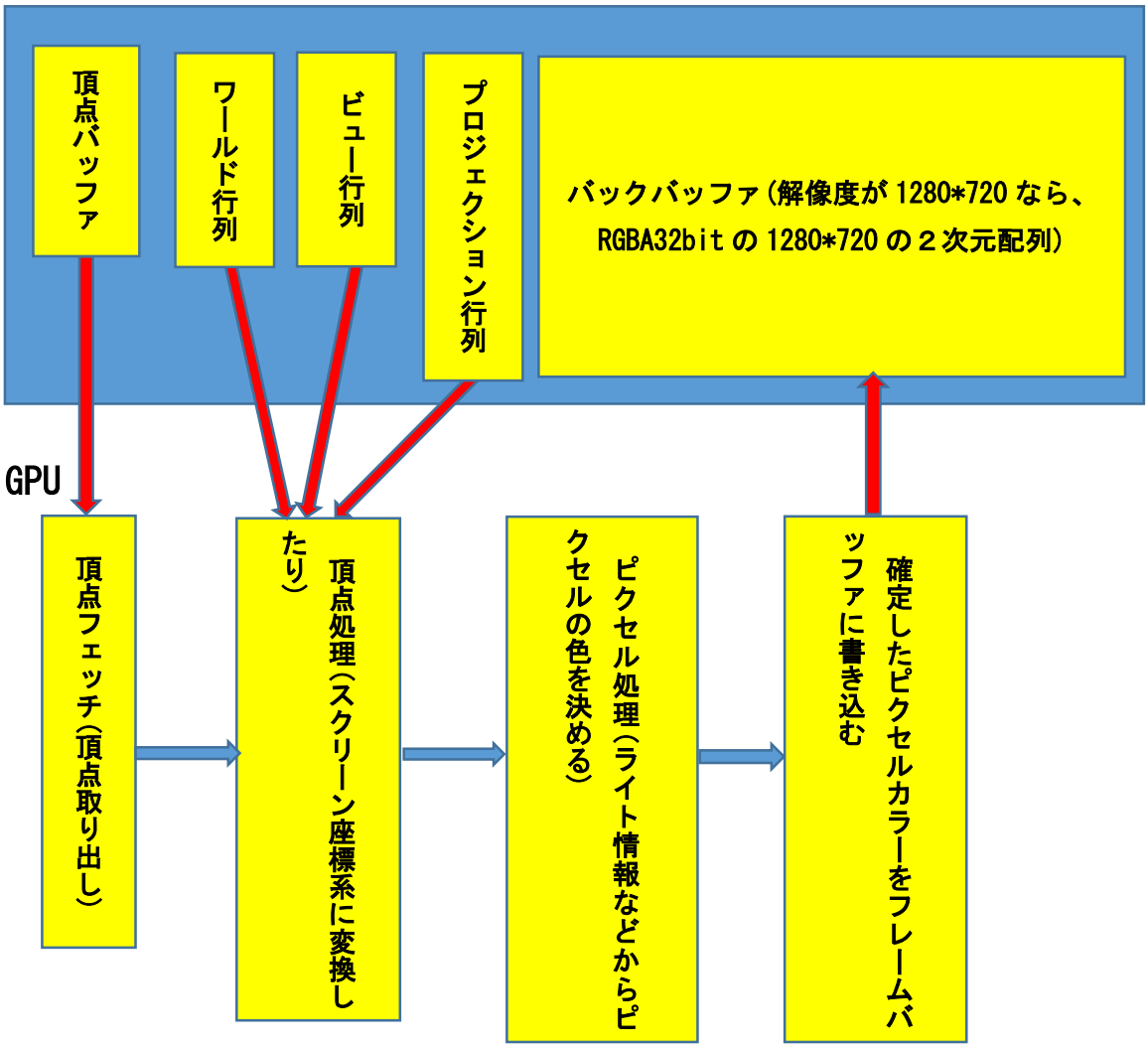
```
    ZeroMemory( &light, sizeof( D3DLIGHT9 ) );
    light.Type = D3DLIGHT_DIRECTIONAL;
    light.Diffuse.r = 1.0f;
    light.Diffuse.g = 1.0f;
    light.Diffuse.b = 1.0f;
    vecDir = D3DXVECTOR3( cosf( timeGetTime() / 350.0f ),
                          1.0f,
                          sinf( timeGetTime() / 350.0f ) );
    D3DXVec3Normalize( ( D3DXVECTOR3* )&light.Direction, &vecDir );
    light.Range = 1000.0f;
    g_pd3dDevice->SetLight( 0, &light );
    g_pd3dDevice->LightEnable( 0, TRUE );
    g_pd3dDevice->SetRenderState( D3DRS_LIGHTING, TRUE );
    // アンビエントライトを設定。
    g_pd3dDevice->SetRenderState( D3DRS_AMBIENT, 0x00202020 );
}
//-----------------------------------------------------------------------------
// 描画
//-----------------------------------------------------------------------------
VOID Render()
{
    // バックバッファと Z バッファをクリア
    g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
                         D3DCOLOR_XRGB( 0, 0, 255 ), 1.0f, 0 );

    // 描画開始。
    if( SUCCEEDED( g_pd3dDevice->BeginScene() ) )
    {
        // ライトとマテリアルを設定。
        SetupLights();
        // ワールドビュープロジェクション行列を設定。
        SetupMatrices();
        // 頂点バッファを設定。
        g_pd3dDevice->SetStreamSource( 0, g_pVB, 0, sizeof( CUSTOMVERTEX ) );
        // 頂点のフォーマットを指定。
        g_pd3dDevice->SetFVF( D3DFVF_CUSTOMVERTEX );
        // 描画。
        g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLESTRIP, 0, 2 * 50 - 2 );
        // 描画終了。
        g_pd3dDevice->EndScene();
    }

    // バックバッファの内容を表示。
    g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
}
```

さて、上記のコードでどの部分が固定機能と呼ばれるものか分かりますか？このコードですと、ワールド行列、ビュー行列、プロジェクション行列、マテリアル、ライトの設定が固定機能になります。
では GPU で実行される処理を考えながら、固定機能とはなんなのかを考えていきましょう。

1.1.1 グラフィックスパイプライン

CPU から Draw 命令送られてくると、下記のようにセットされた頂点バッファから頂点をフェッチ(取り出して)して、その頂点に陰影処理を行いスクリーン座標系に変換して対応するピクセルの色を決定します。

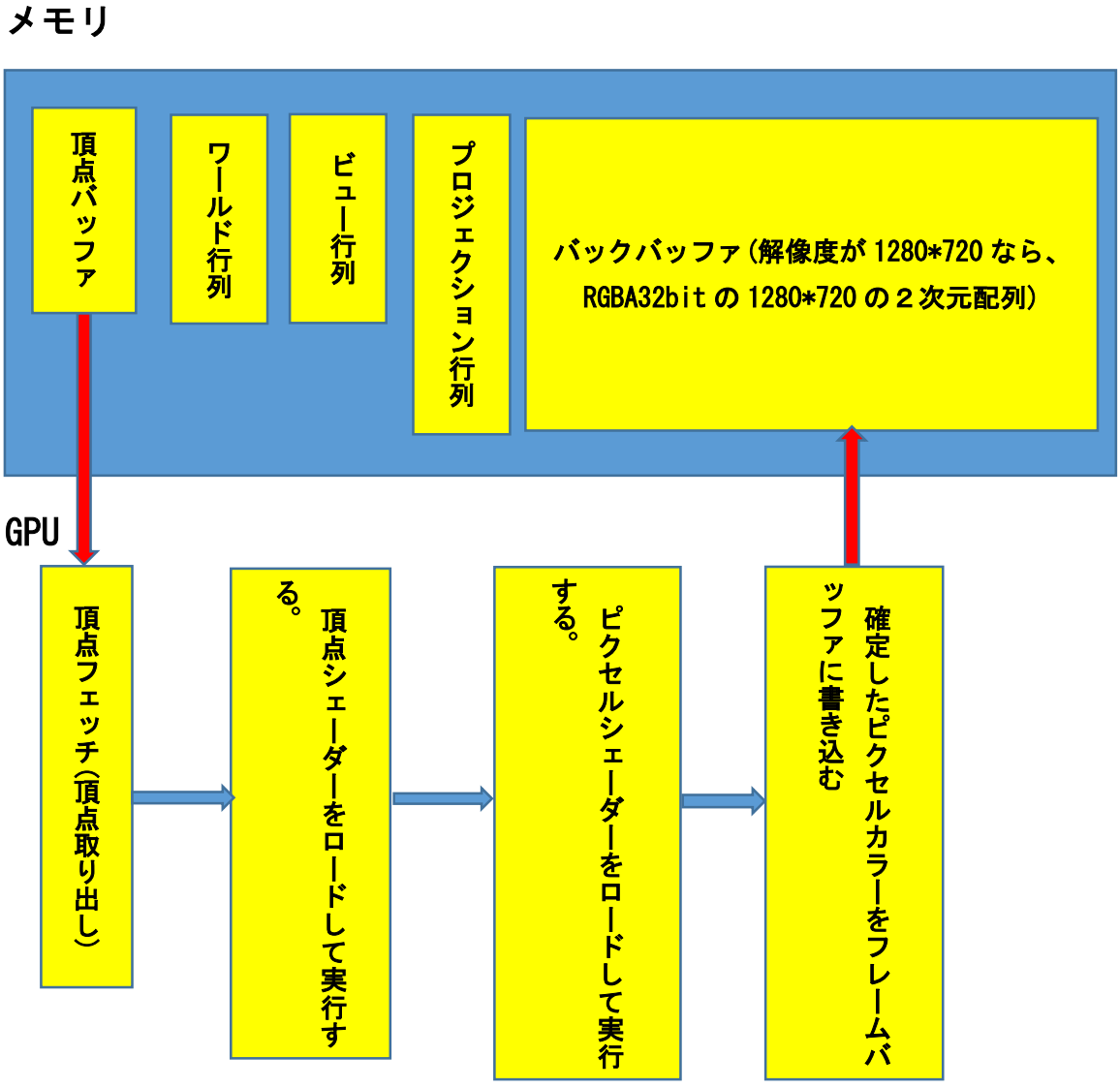
メモリ



このように３次元データから２次元画像を作り出すまでの処理をグラフィックスパイプラインといいます。現在失われた固定機能とはこの図では頂点処理、ピクセル処理がそれにあたります。つまり、現在の GPU は頂点をスクリーン座標に変換する機能とピクセルカラーの決定を自動的に行う機能を用意していないのです！それではどのようにして絵を出しているのでしょうか？頂点をスクリーン座標系に変換して、ポリゴンがスクリーン座標系でどこに位置するのかを決めて、ピクセルカラーを決めないと絵は描けません？そこでシェーダーが登場します。

1.2 シェーダー

シェーダーの導入で先ほど紹介したグラフィックパイプラインの頂点処理とピクセル処理を自分でプログラミングして、自由に頂点処理やピクセル処理を実装することができるようになりました。つまり、自分で頂点をスクリーン座標系へ変換したり、ピクセルカラーを決定するプログラムを書くことになります。つまり DirectX10 以降ではシェーダーを書かないと絵は表示できなくなりました。



この図のように、頂点処理とピクセル処理がシェーダーをロードして実行するという内容に変わっています。ではなぜ固定機能が削除されてシェーダーが登場したのでしょ

うか？せっかく用意されていたものがなくなって、同じものを作らないと絵を出せなくなったなんて面倒だと思いませんか？

1.3 シェーダーが生まれた経緯

固定機能しか存在していなかった DirectX7 まではマイクロソフトが用意したグラフィック表現しか行うことができませんでした。先ほどピクセルカラーを決める方法が固定されているといった話を思い出してみてください。DirectX9 ではせいぜいディフューズライト、スポットライトライト、ポイントライト、アンビエントライトくらいでしょうか？ではこれらの機能を使って下記のようなアニメ調のグラフィック表現が実現できるでしょうか？

アニメ調のグラフィックを実現するためには、特殊なライティングアルゴリズムを実装する必要があります。しかしシェーダーが生まれる前は新しいグラフィック表現を実現するためにはマイクロソフトがその処理を実装するまで待つ必要がありました。また、多数のゲーム開発者の要望に全て答えようとすると DirectX の API がどんどん膨らんで行くことにもなります。(ゲーム開発者というのは他とはことなるユニークな表現を行いたがるものなのです)その要望に答えるためにマイクロソフトは自分たちで処理を実装することを止めて、頂点処理、ピクセル処理を自由にプログラミングできるようにしました。これによりグラフィック表現の幅は大きく広がり、現在の高品質なフォトリアルな表現や、ナリティメットストームのようなノンフォトリアル表現まで多様な表現が実現できるようになったのです。

実はこのアニメ調の表現はプログラマブルシェーダーを書かなくても実現できます。CPU で頂点をロックすれば頂点を自由に加工することができますよね？また、ピクセルカラーも単なる 2 次元

配列に 32bit のピクセルカラーを描き込んでいるだけなので、こちらも CPU でプログラミング可能です。このように CPU でグラフィック処理を行うことをソフトウェアレンダリングといいます。ではなぜわざわざ GPU でプログラミングをするのか？その答えは浮動小数点計算において GPU は CPU に比べて圧倒的に高速に動作するからです。もし興味があれば、一度頂点のスクリーン座標変換を CPU と GPU の両方で実装してみて、速度を比較してみるといいでしょう。CPU の方は目も当てられないような動作速度になるはずです。

Chapter 2

ShaderTutorial_01(最もシンプルなシェーダープログラム)

では実際に簡単なサンプルプログラムを見てみて、シェーダーがどのようなものか見ていきましょう。下記のパスにプログラムを上げていますので Github からコードを pull してください。

https://github.com/KawaharaKiyohara/DirectXLesson/ShaderTutorial_01