WebGL 基礎 .05

フレームバッファ・予備知識

Chapter.01

フレームバッファとは

フレームバッファ

WebGL にはフレームバッファと呼ばれる特殊なオブジェクトが存在する。

これは「描画の対象となるバッファの集まり」のようなオブジェクトで、フレームバッファの中に、色を格納するためのカラーバッファや、深度値を格納するための深度バッファなどがまとめて入っている。

フレームバッファ

WebGL コンテキストを初期化した時点で、このフレームバッファが暗黙でひとつ生成される。

わざわざフレームバッファを生成するような処理を行わなくてもこれまで描画を行うことができていたのは、この暗黙のフレームバッファを既定値として利用してきたからである。

フレームバッファは、独自にあとから生成して追加することもでき、この場合はメモリ上に複数の描画対象領域を持つことができるため、様々な意味で実装の幅が広がる。

フレームバッファ

現代の 3DCG シーンにおいては、フレームバッファは非常に重要かつポピュラーなものになっている。

一度、予備のバッファに対して描画を行い、これに対してさらに処理を行ったり、あるいは一度裏で描画したシーンを参照しながら最終シーンを再構築したり、応用範囲はかなり広い。

フレームバッファの中身

フレームバッファはただ生成しただけでは役に立たず、フレームバッファの中に、それぞれに役割の異なる別のバッファを格納していくことで、はじめて利用することが可能となる。

フレームバッファには最低でも、色を書き込む領域である「カラーバッファ」と、深度値を書き込む領域である「深度バッファ」を格納するのが一般的。

Chapter.02

フレームバッファの初期化

フレームバッファの生成

フレームバッファを生成するには、次のようにメソッドをひとつ呼ぶだけでよい。フレームバッファは、その中身にさらにバッファを格納する外枠のような役割のオブジェクトなので、生成した直後には中身が空っぽの状態になっている。

テクスチャや VBO などの他のバッファと同様に、処理を行うためには生成したフレームバッファはバインドしなければならないので注意。

```
var fb = gl.createFramebuffer(); // フレームバッファオブジェクトの生成

gl.bindFramebuffer(gl.FRAMEBUFFER, fb); // フレームバッファのバインド
```

フレームバッファに格納するバッファ

フレームバッファには、カラーバッファと深度バッファを格納してやる。ただし、カラーバッファにはテクスチャオブジェクトを代用することができるようになっており、かつ、テクスチャに色情報を書き出すことができれば、別のフレームでそれを読み出すこともでき都合が良い。

このような理由により、フレームバッファに格納する（アタッチするという言い方をする）カラーバッファにはテクスチャを利用する場合が多い。

アタッチ用のテクスチャを生成する

フレームバッファにアタッチするテクスチャは、画像などを読み込んでテクスチャとして利用する場合と、生成過程はほとんど同じ。

ただし、ソースとなる元画像があるわけではないので、中身が空になるように初期化する。

```
var texture = gl.createTexture();

gl.texImage2D(gl.TEXTURE_2D, 0 gl.RGBA, width, height, 0, gl.RGBA,
gl.UNSIGNED_BYTE, null);

// 最終引数を null にすることで中身のないテクスチャを準備する
```

フレームバッファにテクスチャをアタッチ

中身のない空のテクスチャが生成できたら、これをハブとなるフレームバッファにアタッチする。

これで、このフレームバッファを選択して描画すると、その内容がここでアタッチしたテクスチャに反映されるようになる。

```
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
```

レンダーバッファと深度バッファ

続いては深度バッファをフレームバッファにアタッチする。

深度バッファは、明確なオブジェクトとして深度バッファというオブジェクトがあるのではなく、様々な用途に利用できるように設計されている汎用的なバッファである「レンダーバッファ」を使って用意する。

```
var db = gl.createRenderbuffer(); // レンダーバッファを生成

gl.bindRenderbuffer(gl.RENDERBUFFER, db); // レンダーバッファをバインドする
```

深度バッファとしてアタッチする

レンダーバッファは汎用的なバッファなので、バッファをバインドしたあと、これをどのように利用するのかを明示する。

深度バッファとして利用することを指定できたら、フレームバッファにアタッチするという点はテクスチャの場合と同じ。

```
gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16, width, height);

gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
gl.RENDERBUFFER, db);
```

Chapter.03

フレームバッファへの描画

フレームバッファへの描画

フレームバッファは、初期状態ではデフォルトの「暗黙のフレームバッファ」がバインドされた状態になっているので、これを生成したフレームバッファで置き換えてやると、レンダリング結果がフレームバッファにアタッチしたテクスチャへと焼きこまれるようになる。

バインドを解除すると、再び規定のフレームバッファが有効になり、スクリーン上に描画結果が反映されるようになる。

```
gl.bindFramebuffer(gl.FRAMEBUFFER, fb); // 自前のフレームバッファのバインド

gl.bindFramebuffer(gl.FRAMEBUFFER, null); // バインド解除し規定のバッファを有効化
```

フレームバッファへの描画

自前のフレームバッファをバインドしレンダリングを行ったとしても、本当に正しくレンダリングされているのかどうかは、テクスチャとして値を読み出し、なにかしらの方法で画面に出力してみるまではわからない。

まずは、テクスチャの内容をそのまま画面に投影する方法から考える。

ビューポートの設定

フレームバッファは、テクスチャをアタッチする都合上、大きさが 2 の累乗サイズになっている必要がある。

このことから、最終的にシーンを映し出すスクリーンの大きさと、バックバッファとなるフレームバッファの大きさは多くの場合一致しない。

WebGL では描画領域の大きさを考慮して勝手に描画をコントロールしてくれたりはしない（むしろされたら困りそう）ので、そこはこちらでしっかりと設定を行ってやる必要がある。

ビューポートの設定

描画する領域はビューポートの設定によって決まるので、フレームバッファへの描画を行う際にはフレームバッファの大きさを、ビューポート設定で反映させる。

フレームバッファへのレンダリングが完了し、規定のスクリーンに対して描画を行う場合も、やはりビューポートの設定しなおしが必要なので注意。

```
gl.viewport(0, 0, framebufferWidth, framebufferHeight); // フレームバッファ

gl.viewport(0, 0, canvas.width, canvas.height); // スクリーンサイズ
```

サンプル.w09

フレームバッファを用いた実際の描画がどのような流れで行われるのか、確認しておく。

このサンプルはシェーダを二組使っているので、こういったマルチシェーダな実装について慣れておくとよい。

やってみよう

どうしてマルチシェーダな実装になるのか、理由を説明できるでしょうか。

また、ふたつ目のシェーダを改造して簡単なポストエフェクト（モノクロ化など）を実装してみよう。

Chapter.04

予備知識

WebGL において便利な予備知識

ここからは、すぐに必須となるようなものではないが、知識として持っているだけで有益な概念や情報についていくつか見ていく。

無理に覚える必要はないが、いずれは使わなければならない場面や、知っていることで素早くバグに対処できたりする場面が必ず出てくる。

拡張機能

拡張機能とは、既定では使えない機能や、次のバージョンで利用可能になる予定の機能などを、一部先行して使うことができる仕組みである。

代表的なところでは、FLOAT で値を格納できる FLOAT テクスチャや、効率よく大量のオブジェクトを描画できる Instanced Array などがある。

特に FLOAT テクスチャはとても重要で活躍する場面も多いので、使えるようになっておくとよい。

拡張機能

拡張機能は、ハードウェアの性能によって使える場合と使えない場合がある（だからこそ拡張機能扱いになっているとも言える）ので、拡張機能を有効化することが可能かどうか、事前にチェックしておく必要がある。

拡張機能は、WebGL コンテキストの取得時と同じように、引数に文字列を渡すタイプのメソッドである gl.getExtension を使って初期化する。もし gl.getExtension の戻り値としてオブジェクトを取得できれば、その拡張機能が利用できる環境ということになる。

```
var ext = gl.getExtention('有効化したい拡張機能名');

// 変数 ext に何かしらのオブジェクトが取得できていれば拡張機能は有効になっている
```

環境マッピング

環境マッピングを用いると、完全な鏡面反射を再現することができる。

いわゆる金属的な、周囲の環境を映し込むような質感を再現するために使われる。

WebGL ではキューブ環境マッピングがサポートされているので、専用のテクスチャなどを用意することさえできれば、比較的簡単に実装が可能。

クォータニオン

クォータニオンは、行列と並び、3D 数学には欠かせない概念。

誤解を恐れずものすごくざっくりいうと、回転を扱うことに非常に長けた概念なので、行列を使った座標変換と比較して、様々な点で回転の制御に優れる。

また、データの量も少なくて済み、ジンバルロックと呼ばれるオイラー角による制御で発生する問題も起こらないなど、ある程度複雑な回転を行う上では欠かせない。

ステンシルバッファ

ステンシルバッファは、フレームバッファに格納されるタイプの、つまりカラーバッファや深度バッファと同じように、レンダリング時に自動的に更新されていくタイプの特殊な描画領域。

カラーバッファには色が、深度バッファには深度が、ステンシルバッファにはステンシルマスクの情報が格納される。

WebGL では、ステンシルバッファは既定では無効化されているため、そのままでは利用することができない。

ステンシルバッファ

ステンシルバッファを有効化するには、WebGL コンテキストを初期化する際に、フラグを立てて初期化を行えばいよい。

canvas の getContext メソッドで WebGL コンテキストを初期化する際に、オブジェクトを引数で渡してやる。

```
canvas.getContext('webgl', {stencil: true});
```

WebGL コンテキストの初期化オプション

ステンシルバッファの例のように、コンテキストの初期化時にオプションを付加することで、初めて有効化される機能がいくつかある。

代表的なところでは、ステンシルバッファのほか、アンチエイリアスのオンオフや、スクリーンに描いた内容を自動的にクリアするかどうかなどが設定可能。

ポイントスプライト

周知の通り、WebGL ではポリゴン以外にも、点、あるいは線でのレンダリングを行うことが可能。

この、点を利用したレンダリングを行うと、頂点の数自体を少なく抑えることができる。しかも点の大きさは環境にもよるがある程度まで大きくできるうえに、そこにテクスチャを貼ることもできる。これがポイントスプライトと呼ばれるもの。

必ずしもポイントスプライトがベターとは限らないが、そういうやり方もあるということは知っておくとよいかもしれない。

付属サンプル

- w09（フレームバッファを使った処理）