WebGL 基礎 .04

深度テスト・カリング・ブレンド・テクスチャ

Chapter.01

深度テスト

前後関係を正しく処理する

WebGL の描画は、基本的に、頂点を次々にスクリーン上に「上書きモード」で描画していく。

頂点が描かれる順番は、単純に頂点を定義した順番や、インデックスバッファの内容によって決まるが、完全な上書きモードで頂点が描画されていくと、本来なら後ろあって隠れるはずだった頂点が、最後のほうになって描画される可能性も出てくる。

完全な上書きモードでは、頂点が描画される順序によっては、上記のように前後関係を無視した描画が行われてしまう可能性があることがわかる。

上書きモードをどう制御するか

この頂点を強制的に上書きモードで描いていく方法を、根本から変える方法はない（たとえば頂点が自動的に並び替えられて奥のほうから勝手に描画されるとか、そういう機能はない）。

それではどうやって前後関係を正しく制御しているのかというと、これには「深度」というキーワードが大きなカギを握っている。

WebGL では、色を書き込むためのカラーバッファの他に、深度（つまり奥行き）だけを書き込むための専用のバッファが存在する。頂点の前後関係を直感的に正しく描画するためには、この深度バッファを用いた処理を有効化しなければならない。

深度バッファ

サンプル.w07 では、最初から深度バッファを使った処理を有効化している。

これを無効化すると、どのようなことが起こるのだろうか。

やってみよう

サンプル内の gl.enable メソッドで、DEPTH_TEST という引数を使っている箇所をコメントアウトして、そのままサンプルを実行したらどうなるか、試してみよう。

深度テスト

深度バッファには、なにかしらの頂点を描画するたびに、その頂点の深度に関する情報が自動的に書き込まれるようになっている。

そして、その深度バッファの状態を確認しながら、描こうとする頂点が手前にあるのか奥にあるのか、描画すべきかそうでないのか、これをチェックするための機構が「深度テスト」である。

これは単に有効化するだけで利用できるが、テストの方法を複数から選択することができる。

深度テスト

深度テストは、デフォルトでは無効にされている。

有効化すると、テストの方法として gl.LEQUAL がデフォルトで指定されている。この指定だと、「奥にあるものが隠される」という普通の挙動になる。

逆に言えば、テストの指定の方法を変えれば、手前にあるものが隠れるようにしたりすることもできる。

深度テスト

深度テストのテスト方法を変更するとどのような変化が起こるのだろうか。

やってみよう

深度テストの方法にはどんな選択肢があるかを調べて、違うものを指定したらどうなるか確かめてみよう。

Chapter.02

カリング

カリング

サンプル.w07 をよく観察してみると、深度テストと同じように gl.enable メソッドを使って有効化している機能がもうひとつあることがわかる。

```
gl.enable(gl.CULL_FACE); ← これ！
```

これはカリングと呼ばれる機能を有効化しているコードで、gl.enable メソッドを呼び出すことで有効化できる点は、深度テストなどと同様である。

カリング

カリングは、ポリゴンの向きによって描画すべきかどうかをチェックし、場合によりポリゴンの描画をしないように設定する機能。

カリングを有効化すると、一見して見た目にはまったく変化がないように見えるが、たとえば月の裏側を地表から見ることができないのと同じで、裏側の見えていない面は、本来描画処理を行う必要性がない。

こういった裏側になり結果的にレンダリングされない面は、カリングの有効化によって描画されなくなり、内部的には負荷が下がる。

カリング

カリングは特殊な処理をする場合には意図的に消去する面を反転させることもできる。

つまり「本来なら隠れてしまう面」だけを描き、「本来なら描画すべき面」をあえて描画しないようにしてしまうのである。

こういった処理がどのようなところで必要になるのか、なかなかイメージしにくいかもしれないが、恐らく使うことになるときがいずれくるので、覚えておくこと。

ポリゴンの表と裏

カリングを用いれば裏になる面を描画しないようにできる。しかし、面が表向きなのかそれとも裏向きなのか、WebGL の内部では何を指標に判断しているのだろうか。

実は、ポリゴンの裏表を決めるのは、頂点を結ぶ順序である。

カメラから見て、反時計回りの軌道で頂点が結ばれる面が表。その逆が裏である。カリングの設定を変更することで、裏と表の判定を逆転させたりすることも可能。

カリング面の反転

カリングの効果を確かめるために、カリング面を反転させて描画してみる。

やってみよう

gl.cullFace メソッドを利用すると、カリングする面を切り替えることができる。
デフォルトは gl.BACK で、裏面がカリングされる。これを gl.FRONT にすることで、表面をカリングすることができる。

Chapter.03

ブレンド

ブレンド

WebGL や OpenGL では、既定ではいくつかの機能（深度テストやカリング）が無効化されているが、この「ブレンド」もそんな無効化されている機能のひとつである。

ブレンドを有効化し適切に設定を行うと、アルファブレンディングや色のスクリーン合成など、フォトレタッチソフトに見られるような色の合成に関する機能を利用することができる。

逆に言うと、きちんと設定を行わないと色の合成を行うことはできず、ブレンドを有効化したとしても絶えず色が強制的に上書きされていくだけの、無効化時と変わらない挙動のままになる。

ブレンド

ブレンドは、WebGL コンテキストそのものに対して設定が保持される。

頂点や、利用しているシェーダの種類などとは無関係で、あくまでも WebGL コンテキストそのものがブレンドに関する設定を持っている。

ブレンドを有効化したあとは、必要に応じて設定を行っていく。

ブレンドファクター

ブレンドを有効化したあとの設定には、大きくわけて合成方式と、その計算方式のふたつがある。

合成方式は、たとえば色を混ぜ合わせる際にカラーやアルファ値をどう扱うのかを指定する。計算方式というのは、その混ぜあわせた色を、加算するのか減算するのかといったことを指定する。

これらの設定のことをブレンドファクターなどと呼ぶことがある。

色の合成方式

色の合成方式は、たとえば以下のように指定する。

```
gl.blendFuncSeparate(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA, gl.ONE, gl.ONE);
```

このようにすると、いわゆる「アルファブレンディング」の設定を行ったのに等しい。最初はかなり意味不明な設定方法に見えるかもしれないが、落ち着いて考えてみると、意外にも単純な方法で色の合成は計算されている。

gl.blendFuncSeparate の各引数の意味は次のようになる。

blendFuncSeparate

- 第一引数：これから描く「色」に乗算する要素
- 第二引数：すでに描かれている「色」に対して乗算する要素
- 第三引数：これから描く「アルファ」に乗算する要素
- 第四引数：すでに描かれている「アルファ」に対して乗算する要素

```
gl.blendFuncSeparate(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA, gl.ONE, gl.ONE);
```

つまり上記のような指定の場合、これから描く色は、描こうとしているアルファ成分によって補正されるため、アルファが 0.7 だとすると、70%の濃さの色となる。下地の色は逆に、1.0 マイナスアルファなので 0.3 となり、30%の濃さの色になる。

blendEquationSeparate

gl.blendFuncSeparate では色の合成方法について指定が可能だが、最終的に SRC（つまりこれから描く色）と DST（つまりすでに描かれている色）をどのように演算するのかは、Equation 系のメソッドを使って指定する。

```
gl.blendEquationSeparate(gl.FUNC_ADD, gl.FUNC_ADD);
```

大抵の場合、このメソッドは gl.blendFuncSeparate と一緒に利用される。

引数の意味は次のようになる。

blendEquationSeparate

- 第一引数：RGB 成分の計算方式
- 第二引数：アルファ成分の計算方式

```
gl.blendEquationSeparate(gl.FUNC_ADD, gl.FUNC_ADD);
```

こちらは色の計算方法を指定している。

FUNC_ADD は最終的にブレンドファクターが適用されたあとの色を加算で処理する。減算合成などの特殊な合成処理を行うことも設定しだいで可能。

ブレンドファクター

ブレンドファクターは最初は直感的にイメージするのが難しいジャンル。

オンラインのサンプルを使って、最終的な出力結果をイメージできるようになっておくとよい。

http://wgld.org/s/sample_018/

やってみよう

参考 URL のサンプルで、様々なブレンドファクターを試してみよう。

Chapter.04

テクスチャの生成と準備

テクスチャ

WebGL や OpenGL では、画像データやイメージデータを扱う場合、テクスチャと呼ばれる専用のオブジェクトを利用する。

テクスチャに利用できるソースとして、WebGL では JPEG や PNG などの、ブラウザでの閲覧が可能な画像フォーマットに加え、Canvas2D を使ってプロシージャルに描画したものをそのまま利用したり、Video タグによって再生されている動画をテクスチャに利用したり、といったことが可能。

テクスチャ

テクスチャ周りの処理は、まずはじめは画像データを読み込んで利用する形から覚えるのが無難だが、WebGL の場合はこれが少々独特な実装方法になる。

これはウェブ上では、画像データの取得は通信の関係でタイミングがずれたりすることがあるため、それを考慮した実装を行わなくてはならないからである。

具体的には、画像をロードするための入れ物として、JavaScript の Image オブジェクトを用意し、画像の参照（src 属性の指定）を始める前にまずコールバック関数を定義しておく。そうすることで、画像の読み込みが終わったら、自動的に実行される処理を事前に登録しておくことができる。

画像のロードをイベントで検知する

画像の読み込みが完了したと同時に、自動的に実行されるような処理を行いたい場合は Image オブジェクトの onload イベントを活用する。

```
var img = new Image();

img.onload = function(){

    ロード完了後の処理;

};

img.src = '画像ファイルのパス';
```

※ addEventListener('load', ...); でも、モチロン可

テクスチャオブジェクト

VBO などのバッファオブジェクトの初期化の際に、まずはオブジェクトを生成し、それをバインドしてデータをセットする、という一連の流れがあった。それと同じように、テクスチャもまずはオブジェクトとして生成するところから始まる。

```
var texture = gl.createTexture(); // テクスチャオブジェクトの生成

gl.bindTexture(gl.TEXTURE_2D, texture);
```

以降、バインドされているテクスチャオブジェクトに対して、あらゆる操作や設定が適応されていく。

テクスチャオブジェクト

テクスチャオブジェクトを生成してバインドしたら、続いてテクスチャに画素のデータを割り当てる。

これには、gl.texImage2D メソッドを用いる。引数が非常に多いが、普通に画像を読み込むだけであれば、ほとんどの引数は固定で問題ない。

```
// 最後の引数に「読み込みが完了している  Image オブジェクト」を渡す

gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, img);
```

テクスチャとミップマップ

テクスチャにはミップマップと呼ばれる仕組みがある。

これは、非常に小さくしか描画されないオブジェクトにテクスチャが貼り付けられる際に、縮小時の色の補間を効率化するための仕組み。

あらかじめ、本来のテクスチャの大きさよりも小さな縮小版をメモリ上に作っておき、レンダリング時のスクリーン上でのサイズに応じて、適切なサイズのテクスチャを自動的に選択して使ってくれる機能。WebGL には、このミップマップを自動生成することができるメソッドがある。

```
gl.generateMipmap(gl.TEXTURE_2D);

// バインドしているテクスチャのミップマップが自動生成される
```

テクスチャサイズの制限

WebGL では、テクスチャの元となる画像のサイズに制限がある。

具体的には、テクスチャのサイズは常に「2の累乗」になっていなければならない。
1, 2, 4, 8, 16……256, 512, 1024……というように、2の累乗の大きさになる画像を準備する。

これ以外のサイズの画像を使ってテクスチャを初期化すると、その時点ではエラーが起こることはないものの、実際の描画の段階になってイメージが正しく反映されななどの不具合が起こるので注意が必要。

テクスチャパラメータ

テクスチャには、様々な設定項目を付加することができる。

テクスチャに対する設定は、その時点でバインドされているテクスチャに対して適用される。

複数のテクスチャオブジェクトを使っている場合は、一括でまとめて設定する方法などはないので、多少手間でもひとつひとつバインドするテクスチャを切り替えて設定していく。

テクスチャパラメータ

テクスチャパラメータの設定では「何に対して設定するか」と、「何を設定するか」というふたつの情報を与えつつ、gl.texParameteri を呼び出す。

```
// テクスチャの補間に関する設定

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);

// テクスチャの範囲外を参照した場合の設定

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
```

Chapter.05

テクスチャの利用

テクスチャユニット

GLSL でテクスチャを使うためには、テクスチャを適切にバインドしてやる必要がある。

WebGL にはテクスチャを格納できるユニットという概念があり、規定ではユニットの0番目がアクティブになっている。アクティブなユニットを他の番号に切り替えて再度バインドを行えば、同時に複数のユニットに異なるテクスチャをバインドすることができる。何番目のユニットまで同時に使うことができるのかは、ハードウェアの性能によって異なる。

```
gl.activeTexture(gl.TEXTURE0);

// gl.TEXTURE0 から gl.TEXTURE31 まで定数が用意されている
```

JavaScript からユニット番号をプッシュ

GLSL 側では、sampler2D という型の変数を uniform 変数として宣言しておき、これを使ってテクスチャを参照する関数を呼び出す。

JavaScript 側からは、利用したいユニット番号を gl.uniform1i メソッドを使ってシェーダへプッシュする。GLSL 側でどのユニットのテクスチャを参照するかは、ここでプッシュされてきた uniform 変数の番号によって決まる。

```
uniform sampler2D texture;

// ↑ GLSL    ↓ JavaScript

gl.uniform1i(uniLocation, 0); // この場合はユニット 0 を指定していることになる
```

GLSL でテクスチャの情報を取得

シェーダ内でテクスチャから画素の情報を読み出すには、texture2D 関数を利用する。

第一引数には uniform として渡ってきたユニット番号を、第二引数には、テクスチャ座標を渡す。

```
uniform sampler2D texture;

vec4 smpColor = texture2D(texture, texCoord.st);

// 上記のようにすると、変数 smpColor に読み出した色が格納される
```

テクスチャ座標

テクスチャ座標は、原則として頂点に持たせる頂点属性のひとつ。

0.0 から 1.0 の範囲で指定し、対象の頂点がテクスチャのどの座標を参照すればいいのかを指定する。

頂点に個別に割り振られるものなので、普通は attribute としてシェーダに送られるようにする。また、シェーダ内では慣例として XY ではなく ST を用いて座標を表現する場合が多い。

テクスチャ座標のイメージ

テクスチャ画像

テクスチャの元画像には、サイズの制限があった。

それを踏まえつつ、サンプルファイルを改造して、別の画像がテクスチャとして貼り付けられるようにしてみる。

やってみよう

w.08 ではテクスチャを用いた描画を行っているので、サンプルの画像を他のものに切り替えてみよう。

テクスチャへの画像データ割り当て時の注意

テクスチャに Image オブジェクトを割り当てる gl.texImage2D メソッドを呼び出す際は、Image オブジェクトの画像データのロードが完全に完了している必要がある。

texImage2D は、あくまでもその呼びだされた瞬間の状態をテクスチャに格納しようとする。つまり、画像のロードが完了していない状態で呼び出してしまうと、引数の指定などが一切間違っていなくても、真っ暗なテクスチャになってしまうなどの不具合が起こる。

完全に画像がロードされてから初期化処理を行うようにすること。

Chapter.06

WebGL のパラメータ調査

ハードウェアパラメータのチェック

WebGL に限らず、3D の分野ではハードウェアの性能により実現できる処理の範囲が大きく左右される。そのような理由から、ハードウェアの性能を調べるための、パラメータチェックの仕組みが API に備わっている。

このパラメータのチェックには多数の項目があるが、代表的なところでは、たとえばテクスチャユニットの最大数、読み込めるテクスチャの大きさの最大ピクセル数、シェーダへ送信することができるベクトルの最大個数などがある。

パラメータのチェック

WebGL コンテキスト初期化後、基本的にどんなタイミングでもチェック用のメソッドを呼ぶことはできる。

実現したい処理が性能に左右される可能性がある場合は、事前にチェックしなければならない。

```
gl.getParameter(gl.MAX_TEXTURE_SIZE); // テクスチャの最大サイズ
```

付属サンプル

- w08（テクスチャを使った基本的な処理）