Grimoire.jsハンズオン

レンダラー周りをいじってみよう

Grimoire.js

ToC

- 外部のマテリアルを使ってみる
- レンダラーの設定をしてみよう
- マテリアルを書いてみよう

外部マテリアルを使ってみよう

- Session4/01.sortをダウンロードする
- `<import-material>`タグでマテリアルを読み込み
- `<mesh>`のmaterial属性に`new(typeName)`という形式で指定

Grimoire.jsマテリアル形式(.sort)

・シェーダーと描画設定などを固めたファイル

・シェーダーという単位ではなく、マテリアル
　という単位で配布、利用ができる

・扱うのに一切jsを書く必要がない

・分業が超容易

rendererノード

- canvas内の特定の領域の描画をするタグ
- 最初は実は省略されている。

省略した状態

```
<goml>
  <scene>
    <camera position="0,0,5"/>
    <mesh geometry="cube" color="red">
    </mesh>
  </scene>
</goml>
```

省略していない状態

```
<goml>
  <scene>
    <camera position="0,0,5"/>
    <mesh geometry="cube" color="red">
    </mesh>
  </scene>
  <renderer>
    <render-scene/>
  </renderer>
</goml>
```

render-XXXノード

- render-sceneノード
  - ビューポート全体を指定されたカメラから描画する。
- render-quadノード
  - ビューポート全体を一枚のquadで指定したマテリアルで描画する。

実際にマテリアルを書いてみる

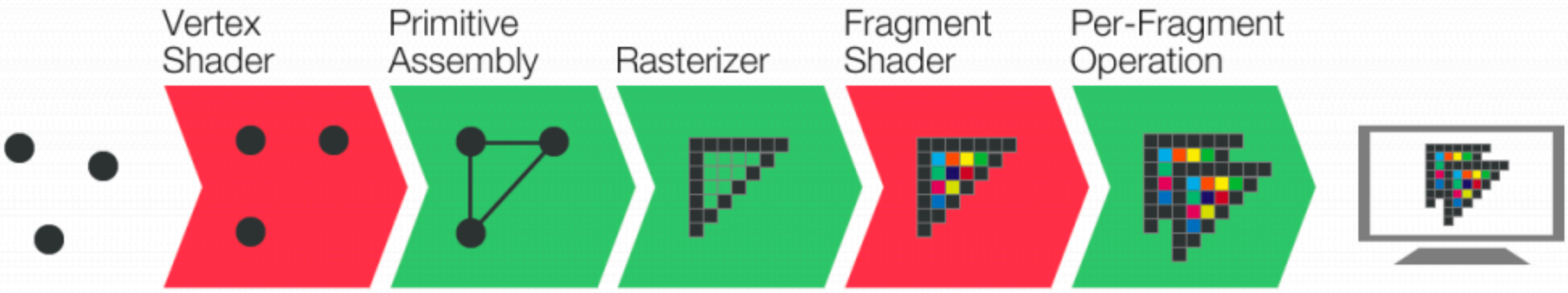
WebGL

- WebGL: Web上でGPUを用いるためのAPI

- GPU上で動作するシェーダー言語を用いると様々な描画処理を高速に行える。

- 3DCGはこれを3Dのシミュレーションに生かしているに過ぎない。

どうして絵が出るのか?



ここからはsortの中でGLSLを扱いますが、
その文法についてはあまり深く踏み込みません。
なんとなく。ぐらいで見ていきましょう。

マテリアルファイルの構造

```
@Pass{
  FS_PREC(mediump,float)
  #ifdef VS                                    頂点シェーダー

    attribute vec3 position;

    void main(){
      gl_Position = vec4(position,1);
    }
  #endif

  #ifdef FS                                    フラグメントシェーダー

    @{type:"color",default:"yellow"}
    uniform vec4 color;

    void main(){
      gl_FragColor = color;
    }
  #endif
}
```

マテリアル内の変数

セマンティクス　　属性

```
@USER_VALUE{type:"color",default:"yellow"}
uniform vec4 color;
```

セマンティクス・・・どのようにこの変数が渡されるか。

例:USER_VALUE・・・ノードの値から決める

TIME・・・現在の時間

セマンティクスによって、様々な属性を受け取り得る。

その他の良い機能

- マクロの値のノードへの露出
  (ループ回数などの定数を変更可能)
- blendFuncやcullFaceなどのglステートの変更
- マルチパス
- include
- などなど