GLES 2.0 Part 3 ポリゴン描画とテクスチャ利用

OpenGLの描画

すべてポリゴンを描画することで行う。

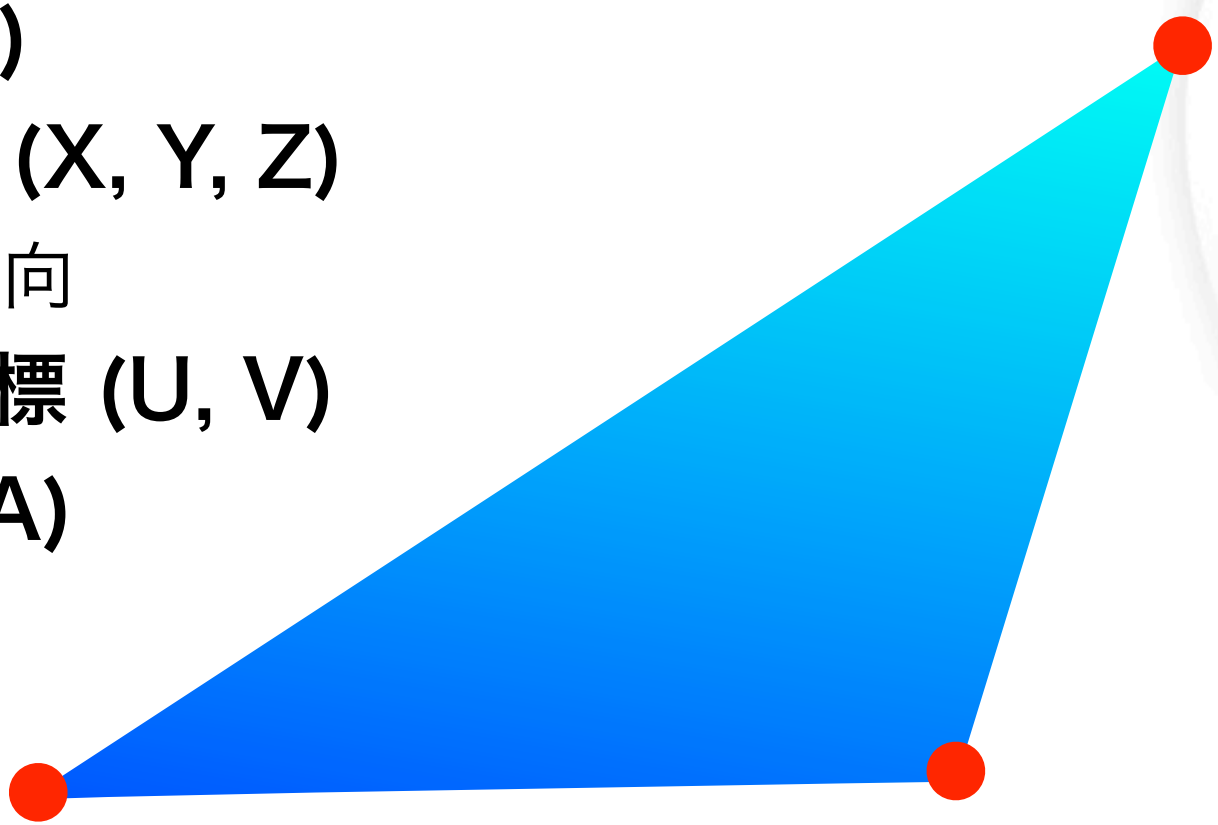
ポリゴン

三角形で表される平面。

各頂点には属性がセットできる。

- **座標 (X, Y, Z)**
- **法線ベクトル (X, Y, Z)**

  光線の反射方向
- **テクスチャ座標 (U, V)**
- **色 (R, G, B, A)**



ポリゴンの描画

- **glBegin()〜glEnd()は過去の遺物**
  GL / GLES の別は問わず。
- **頂点バッファまたは頂点インデックスバッファによる描画が基本。**
  さらに効率化するために VBO と VAO を併用する。

VBO / VAO

- **頂点バッファオブジェクト (VBO)**
  頂点のデータのメモリ位置を指定する。
- **頂点配列オブジェクト (VAO)**
  VBOにメモリ内の情報構成を指定する。
    例. 座標3要素(float)→法線ベクトル3要素(float)
      座標2要素(short)→色4要素(ubyte)

VBO単体で使うこともできるが、基本的にVAOと組み合わせて使う。

作成の手順：VAOの作成→VBOの作成
利用の手順：VAOを選択するだけ

ポリゴンの描画

3Dに必要なセットアップ

viewDidLoad メソッドにて

- **view.drawableDepthFormat =**
  GLKViewDrawableDepthFormat24;
  None/16bit/24bit が選択可能。

setupGL メソッドにて

- **glEnable(GL_DEPTH_TEST);**
  どの頂点が手前に来るかをチェックする。

ポリゴンデータの用意

立方体のデータ＝

6面 × 2個の三角形 × 3頂点 = 36頂点

各頂点ごとに、座標＋法線ベクトル＋テクスチャ座標

```
GLfloat gCubeVertexData[8*36] =
{
            座標                    法線ベクトル          テクスチャ座標
    0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f,  0.0f, 0.0f,
    0.5f, 0.5f, -0.5f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f,
    0.5f, -0.5f, 0.5f,  1.0f, 0.0f, 0.0f,  0.0f, 1.0f,
    ...
};
```

手順1. VAOとVBOの作成


```
// VAOの作成と選択
GLuint _vertexArray;
glGenVertexArraysOES(1, &_vertexArray);
glBindVertexArrayOES(_vertexArray);

// VBOの作成と選択（VAOに対してセットされる）
GLuint _vertexBuffer;
glGenBuffers(1, &_vertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, _vertexBuffer);
```

glBindBuffer()の第１引数：

頂点の場合は GL_ARRAY_BUFFER

頂点インデックスの場合は GL_ELEMENT_ARRAY_BUFFER

手順2. VBOのデータセット


```
glBufferData(
      GL_ARRAY_BUFFER,
      sizeof(gCubeVertexData),
      gCubeVertexData,
      GL_STATIC_DRAW);
```

第１引数：

頂点の場合は GL_ARRAY_BUFFER

頂点インデックスの場合は GL_ELEMENT_ARRAY_BUFFER

第４引数：

以後のデータ変更なし GL_STATIC_DRAW

以後のデータ変更あり GL_DYNAMIC_DRAW

手順3. VAOの各種データサイズ設定


```
// 座標データの利用
glEnableVertexAttribArray(GLKVertexAttribPosition);
glVertexAttribPointer(
    GLKVertexAttribPosition,
    3, GL_FLOAT, // 3個のfloat
    GL_FALSE,    // 常にGL_FALSE
    32,          // 1頂点分のデータサイズ
                 (float 4byte*8個)
    BUFFER_OFFSET(0));
                 // これより前のデータのサイズ
                 (なし＝0)
```

手順3. VAOの各種データサイズ設定


```
// 法線ベクトルデータの利用
glEnableVertexAttribArray(GLKVertexAttribNormal);
glVertexAttribPointer(
    GLKVertexAttribNormal,
    3, GL_FLOAT, // 3個のfloat
    GL_FALSE,    // 常にGL_FALSE
    32,          // 1頂点分のデータサイズ
                 (float 4bytes*8個)
    BUFFER_OFFSET(12));
                 // これより前のデータのサイズ
                 (座標データ＝12bytes)
```

手順3. VAOの各種データサイズ設定


```
// テクスチャ座標データの利用
glEnableVertexAttribArray(GLKVertexAttribTexCoord0);
glVertexAttribPointer(
    GLKVertexAttribTexCoord0,
    3, GL_FLOAT, // 3個のfloat
    GL_FALSE,    // 常にGL_FALSE
    32,          // 1頂点分のデータサイズ
                 (float 4bytes*8個)
    BUFFER_OFFSET(24));
                 // これより前のデータのサイズ
                 (座標データ＋法線データ＝24bytes)
```

手順4. 描画の実行


```
// VAOの選択
glBindVertexArrayOES(_vertexArray);

// 頂点配列の描画
glDrawArrays(GL_TRIANGLES, 0, 36);
```

頂点インデックス配列の場合は、
描画に glDrawElements() を使う。

テクスチャの利用

テクスチャの設定


```
// テクスチャの読み込み
NSURL *imageURL = [[NSBundle mainBundle]
    URLForResource:@"tex" withExtension:@"png"];
GLKTextureInfo *texInfo = [GLKTextureLoader
    textureWithContentsOfURL:imageURL
    options:nil error:NULL];

// テクスチャのセット
self.effect.texture2d0.enabled = GL_TRUE;
self.effect.texture2d0.name = texInfo.name;
self.effect.texture2d0.target = texInfo.target;

[self.effect prepareToDraw];
```

テクスチャサイズの制限

- **GLES2.0 では GL_TEXTURE_2D で NPOT 対応**
  - NPOT: Non-Power-Of-Two
    - 2の乗数ではないサイズが利用できる。
    - Mac OS X 10.6.8 以降でも同様。
- **従来の GL_TEXTURE_2D は POT のみ**
  - 32, 64, 128, 256, ... など2の乗数のみ
- **縦横ともMAX1024ピクセル**
  - iPad 2などではもう少し大きい。