②OpenCV と OpenGL の連携

床井浩平

雛形プログラム (oglcv.cpp)

```
// 補助プログラム
#include "GgApp.h"

// アプリケーション本体
int GgApp::main(int argc, const char* const* argv)
{
  // ウィンドウを作成する
  Window window;

  // ウィンドウが開いている間繰り返す
  while (window)
  {
    //
    // ここで図形を描く
    //

    // カラーバッファを入れ替えてイベントを取り出す
    window.swapBuffers();
  }

  return EXIT_SUCCESS;
}
```

こんなことを話します

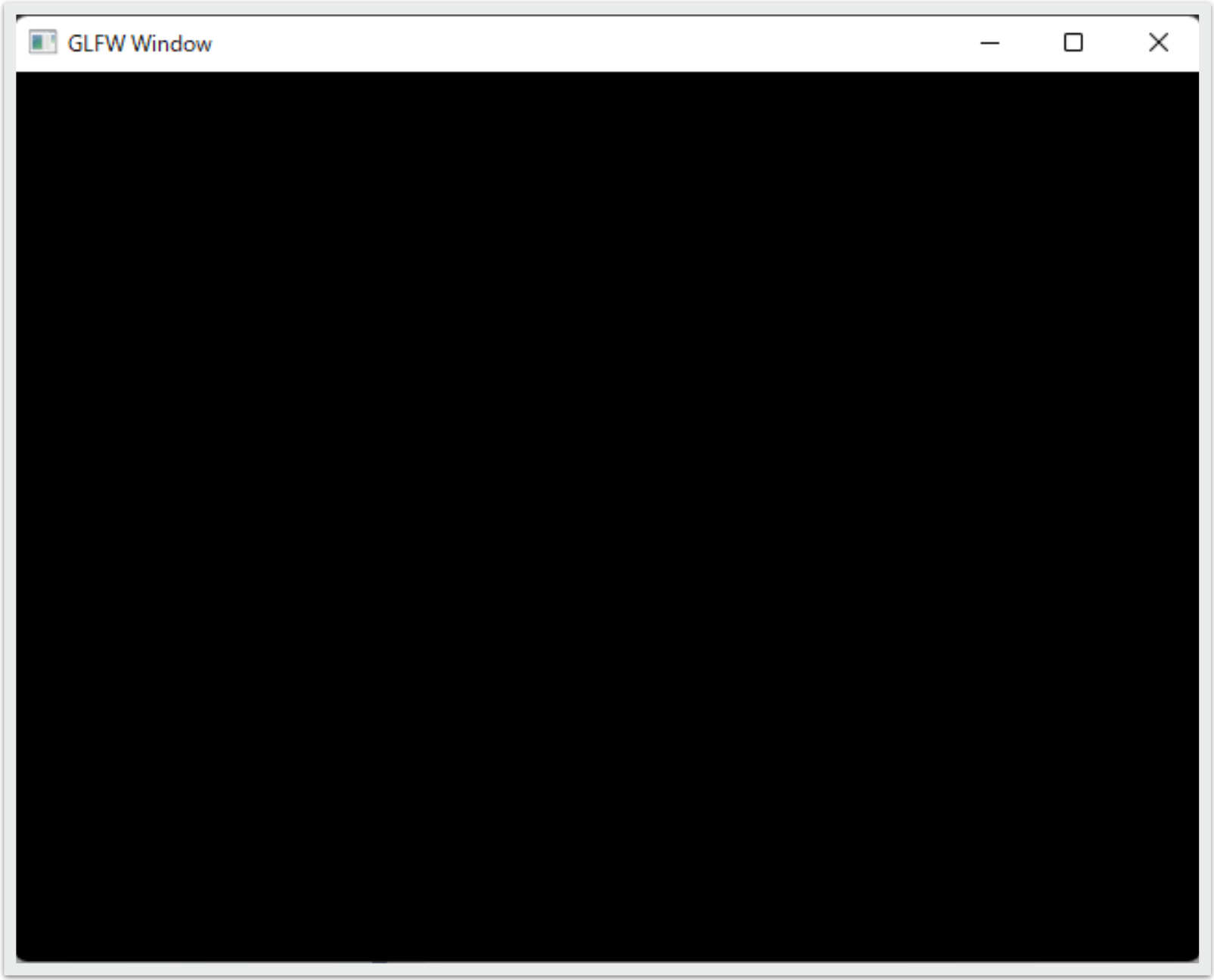
- OpenGL による図形の描画
- OpenCV による画像の読み込み
- テクスチャマッピング
- 簡単な画像処理

ひな形プログラムのビルドと実行

ウィンドウを開く

ウィンドウを開く

画面をクリアしていないので
何が表示されるかわからない



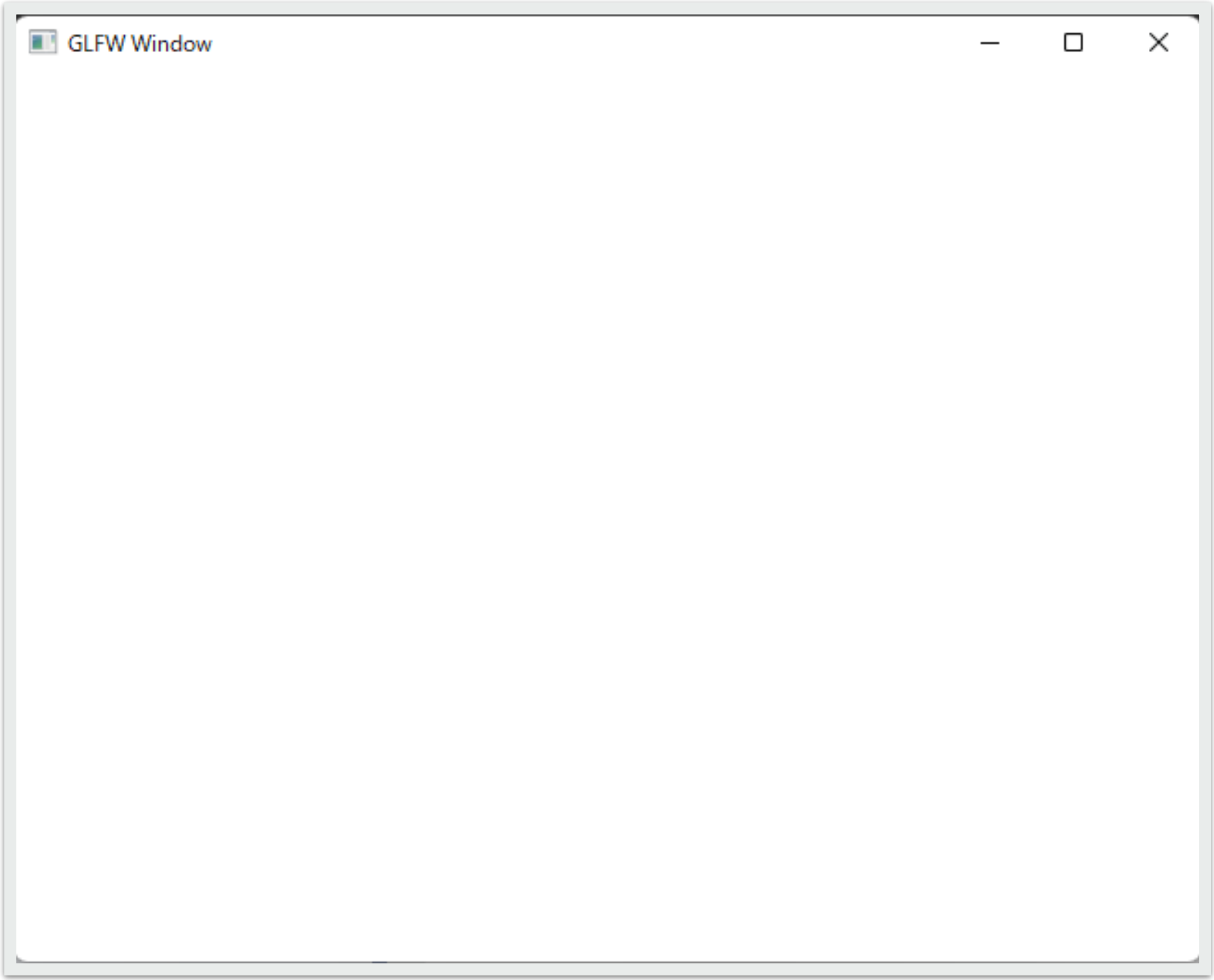
画面クリア (oglcv.cpp)

```
  // 背景色の設定
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

  // ウィンドウが開いている間繰り返す
  while (window)
  {
    // 画面クリア
    glClear(GL_COLOR_BUFFER_BIT);

    //
    // ここで図形を描く
    //

    // カラーバッファを入れ替えてイベントを取り出す
    window.swapBuffers();
  }
```

R, G, B, A を 0～1 で指定します

画面クリア

背景色で塗りつぶされる



図形の準備 (oglcv.cpp)

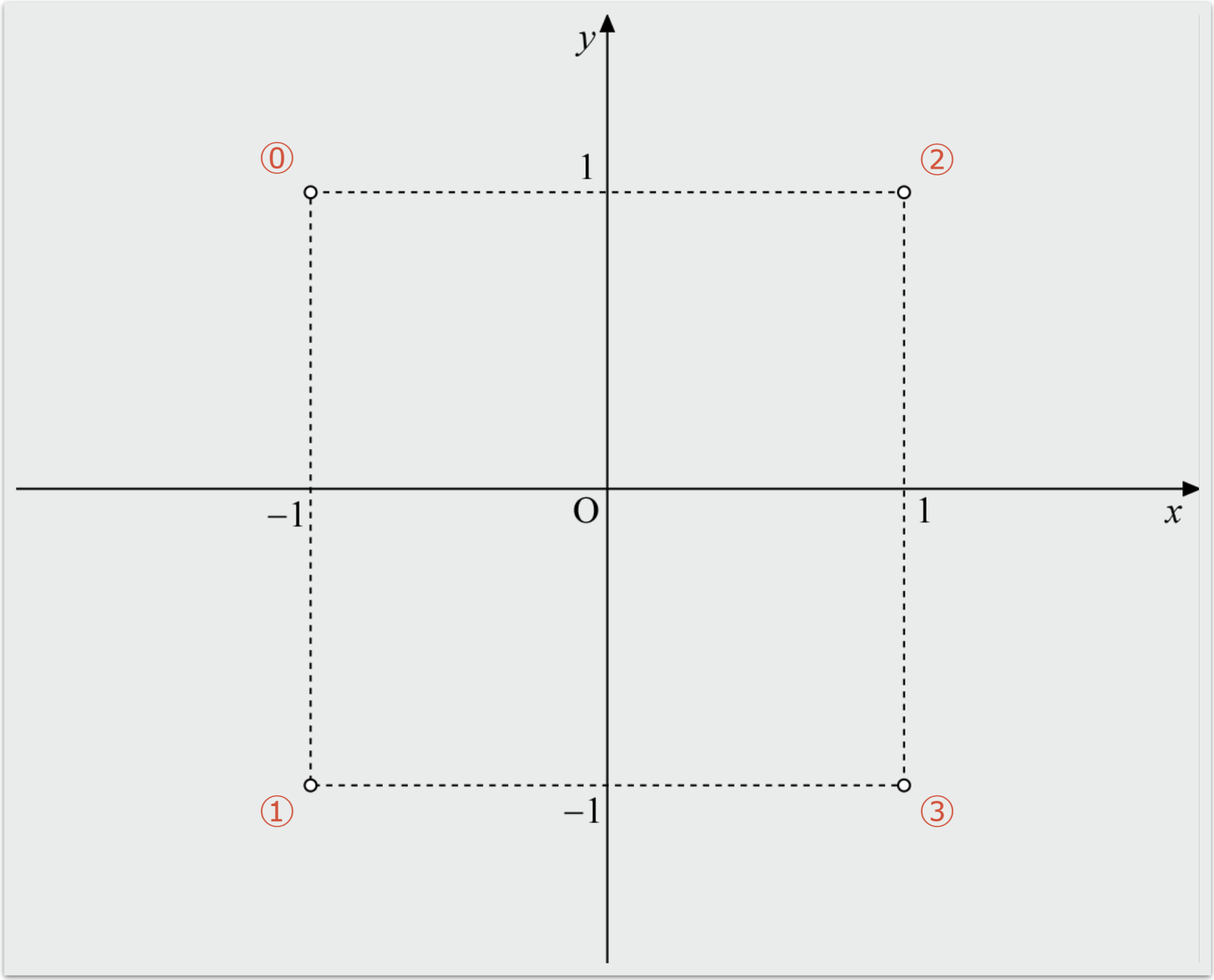
```
  // ウィンドウを作成する
  Window window;

  // 頂点配列オブジェクト
  GLuint vao;
  glGenVertexArrays(1, &vao);
  glBindVertexArray(vao);

  // 頂点バッファオブジェクト
  GLuint vbo;
  glGenBuffers(1, &vbo);
  glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

図形の描画

ウィンドウいっぱいに矩形を描く

頂点属性

頂点の位置

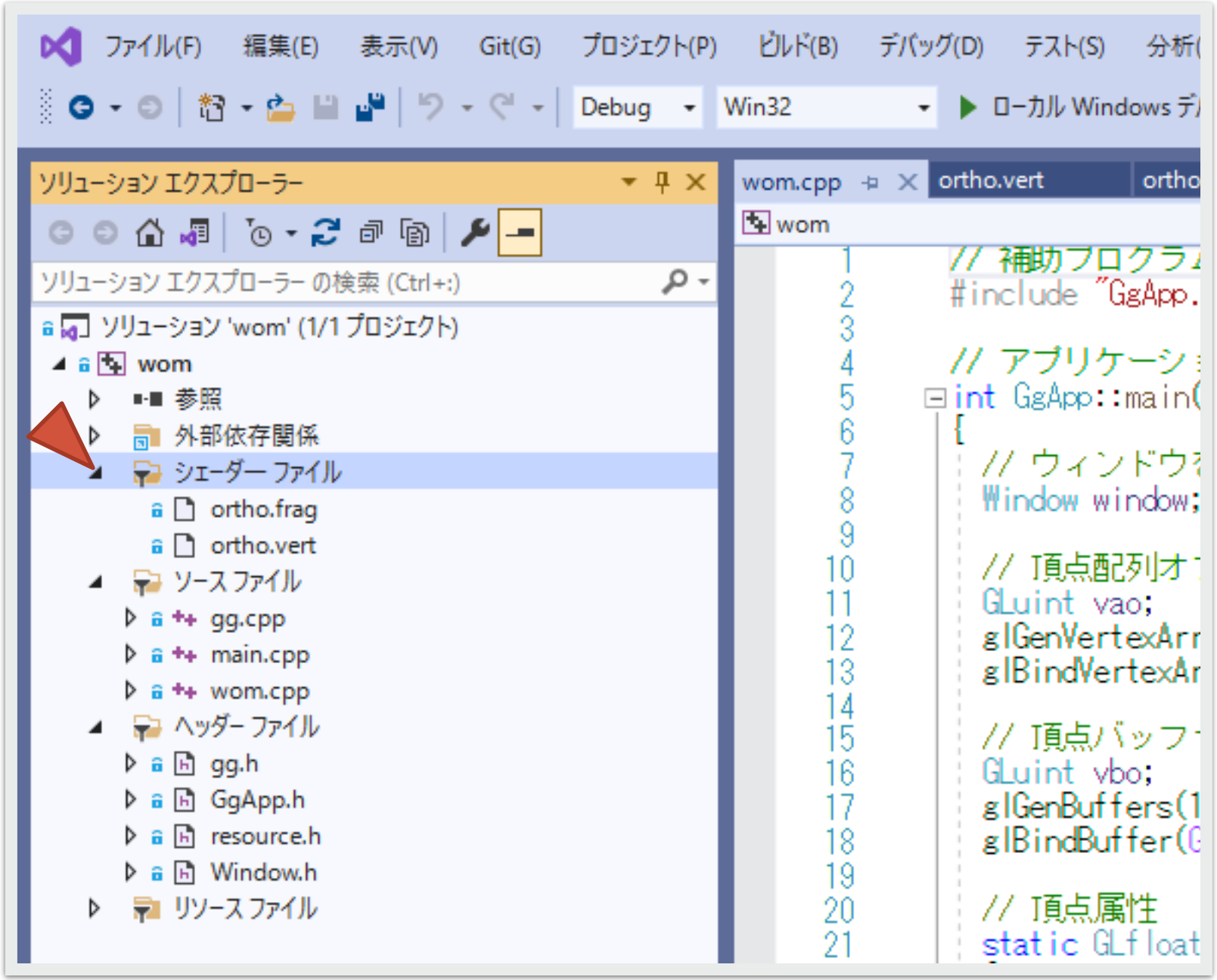


```
  // 頂点属性
  static GLfloat position[][2]
  {
      { -1.0f,  1.0f },   ⓪
      { -1.0f, -1.0f },   ①
      {  1.0f,  1.0f },   ②
      {  1.0f, -1.0f }    ③
  };

  // 頂点属性の転送
  glBufferData(GL_ARRAY_BUFFER, sizeof position, position, GL_STATIC_DRAW);

  // attribute 変数の設定
  glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(0);

  // 頂点配列オブジェクト完成
  glBindVertexArray(0);

  // 背景色の設定
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
```

シェーダのソースファイル

ortho.vert

ortho.frag



バーテックスシェーダのソースファイル (ortho.vert)

```
#version 410

// 頂点の位置
layout (location = 0) in vec4 position;

void main()
{
  gl_Position = position;
}
```

フラグメントシェーダのソースファイル (ortho.frag)

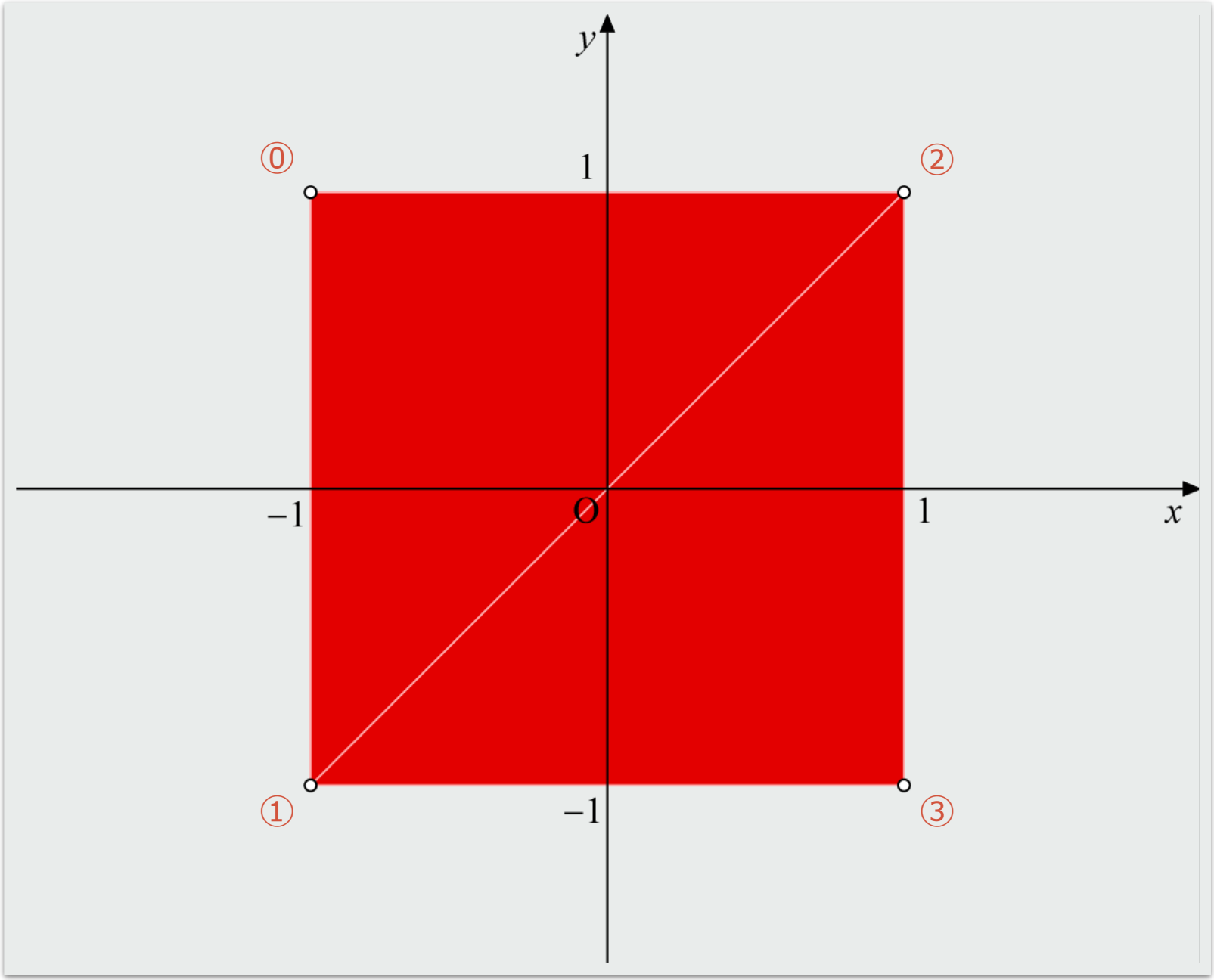
```
#version 410

// 画素の色
layout (location = 0) out vec4 color;

void main()
{
  color = vec4(1.0, 0.0, 0.0, 1.0);
}
```

プログラムオブジェクトの作成 (oglcv.cpp)

```
// attribute 変数の設定
 glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
 glEnableVertexAttribArray(0);

 // 頂点配列オブジェクト完成
 glBindVertexArray(0);

 // プログラムオブジェクトの作成
 const GLuint program{ ggLoadShader("ortho.vert", "ortho.frag") };

 // 背景色の設定
 glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

 // ウィンドウが開いている間繰り返す
 while (window)
 {
   // 画面クリア
   glClear(GL_COLOR_BUFFER_BIT);
```

描画するプリミティブ

GL_TRIANGLE_STRIP

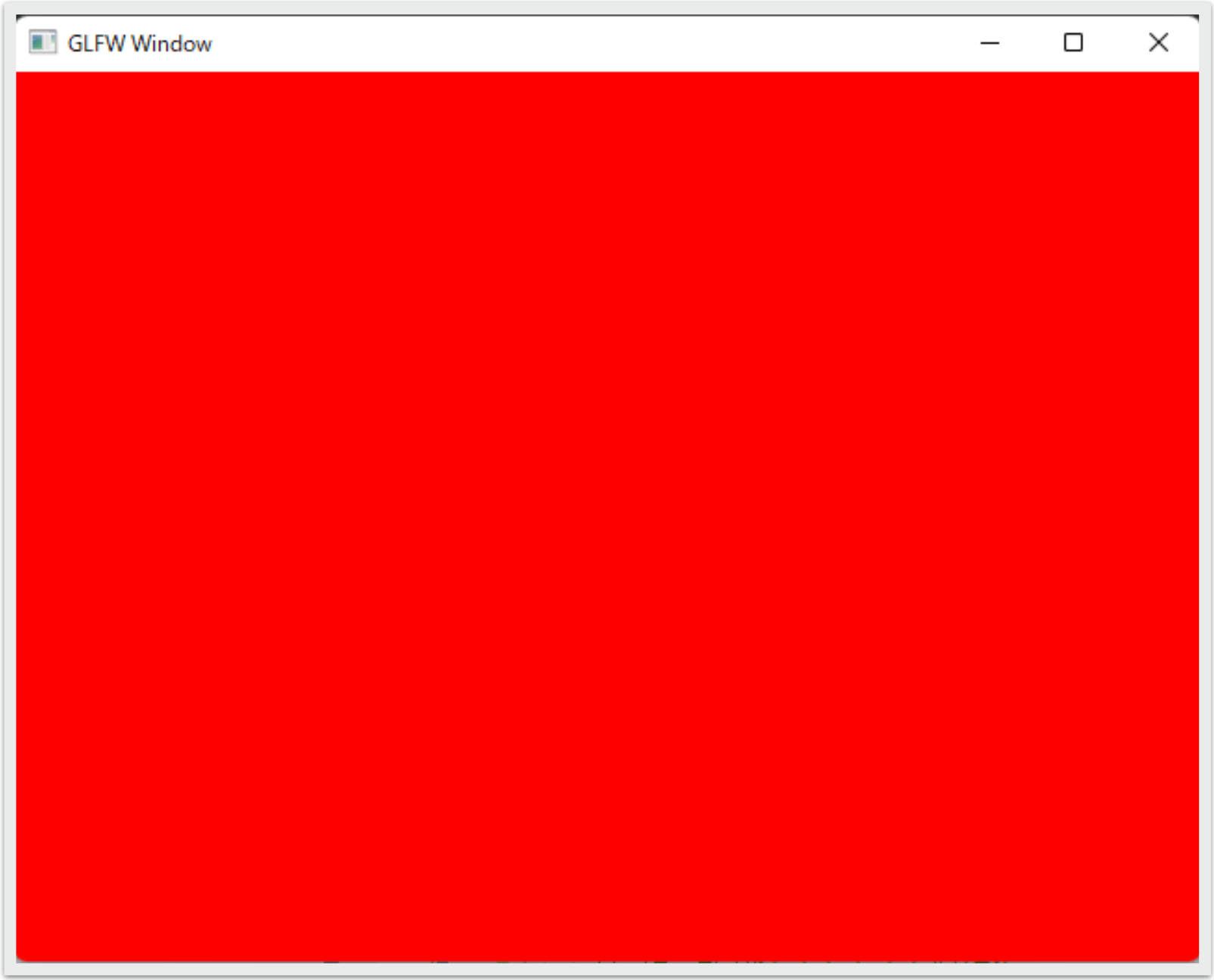


図形の描画 (oglcv.cpp)

```
  // ウィンドウが開いている間繰り返す
  while (window)
  {
    // 画面クリア
    glClear(GL_COLOR_BUFFER_BIT);

    // プログラムオブジェクトの指定
    glUseProgram(program);

    // 頂点配列オブジェクトの指定
    glBindVertexArray(vao);

    // 図形の描画
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

    // カラーバッファを入れ替えてイベントを取り出す
    window.swapBuffers();
  }
```

描画する頂点数なので std::size(position) としたいところですが、ここはあえて 4 のままで

図形の描画



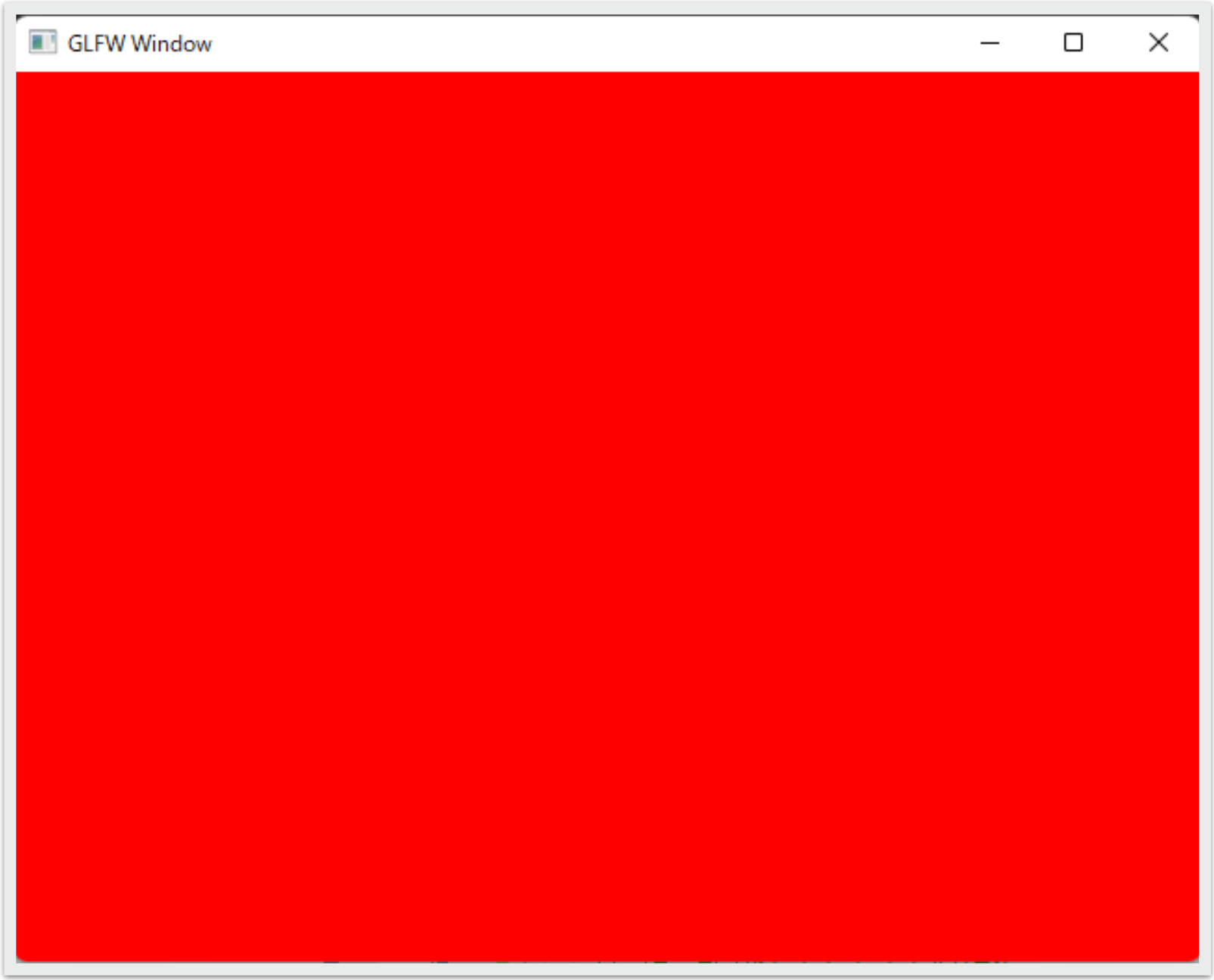
シェーダプログラミング

同次座標の取り扱いと２次元の座標変換

バーテックスシェーダでスケーリングする (ortho.vert)

```
#version 410

// 頂点の位置
layout (location = 0) in vec4 position;

void main()
{
  gl_Position = position * 0.5;
}
```

変わらない



3次元の座標値は同次座標で表される

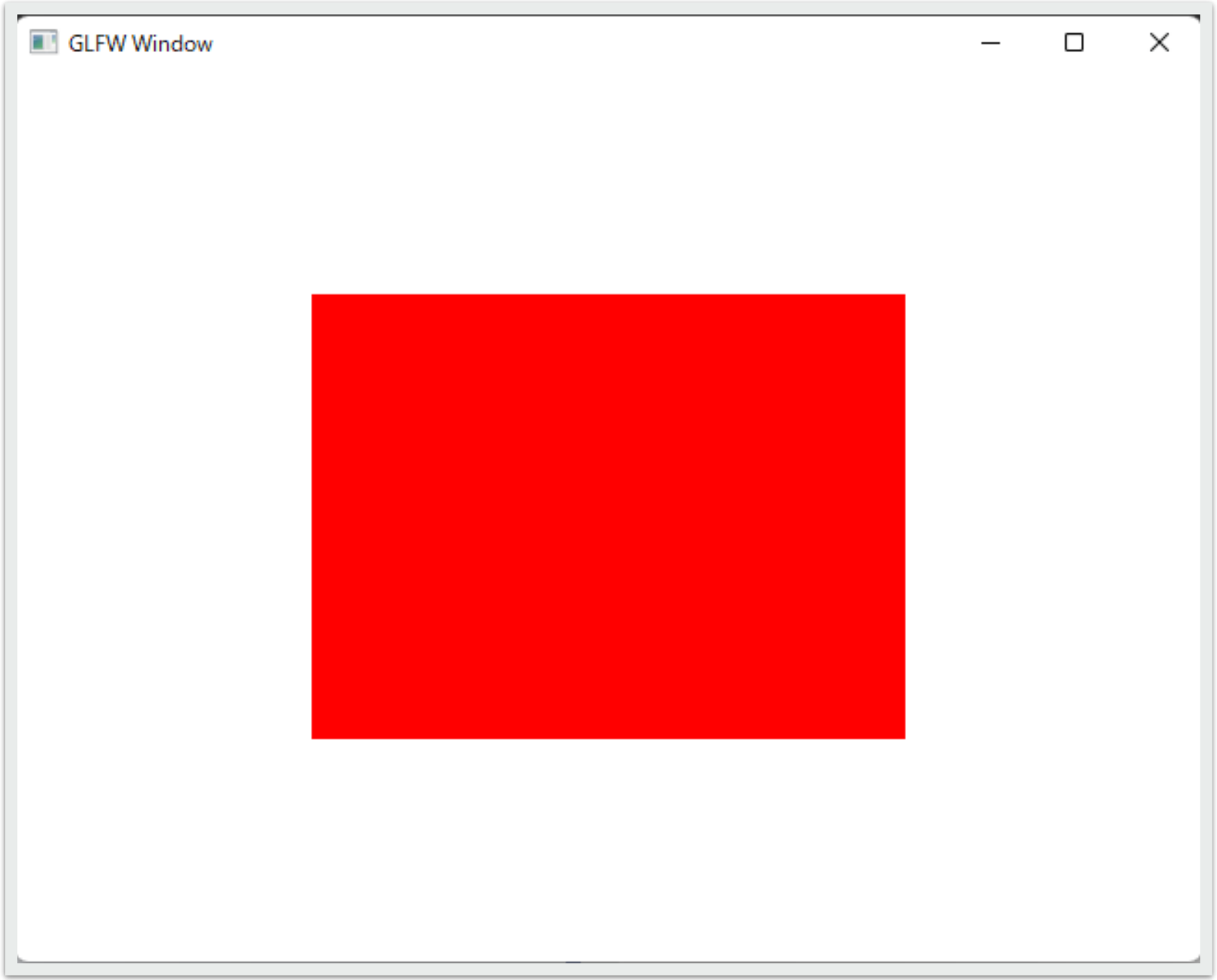
- 頂点属性の2次元の頂点位置は $z = 0$ の平面上の3次元の位置になる
  - $(x, y) \Rightarrow (x, y, 0)$
- 3次元の座標値は同次座標で表される
  - $(x, y, z) \Rightarrow (x, y, z, 1)$
  - position, gl_Position のデータ型は vec4
- 同次座標をスカラー倍しても標準座標は変わらない
  - 同次座標 $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$ → 標準座標 $\begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$ $\Rightarrow$ a $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} ax \\ ay \\ az \\ aw \end{pmatrix} \rightarrow \begin{pmatrix} ax/aw \\ ay/aw \\ az/aw \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$

バーテックスシェーダでスケーリングする (ortho.vert)

```
#version 410

// 頂点の位置
layout (location = 0) in vec4 position;

void main()
{
  gl_Position = position * vec4(0.5, 0.5, 0.5, 1.0);
}
```

縮小された

でも横に伸びている



ウィンドウのアスペクト比で補正する (ortho.vert)

```
#version 410

// 頂点の位置
layout (location = 0) in vec4 position;

// ウィンドウのアスペクト比
uniform float aspect;

void main()
{
  gl_Position = position * vec4(0.5 / aspect, 0.5, 0.5, 1.0);
}
```

uniform 変数の場所を調べる (oglcv.cpp)

```
  // 頂点配列オブジェクト完成
  glBindVertexArray(0);

  // プログラムオブジェクトの作成
  const GLuint program{ ggLoadShader("ortho.vert", "ortho.frag") };

  // uniform 変数の場所を調べる
  const GLint aspectLoc{ glGetUniformLocation(program, "aspect") };

  // 背景色の設定
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

  // ウィンドウが開いている間繰り返す
  while (window)
  {
    // 画面クリア
    glClear(GL_COLOR_BUFFER_BIT);
```

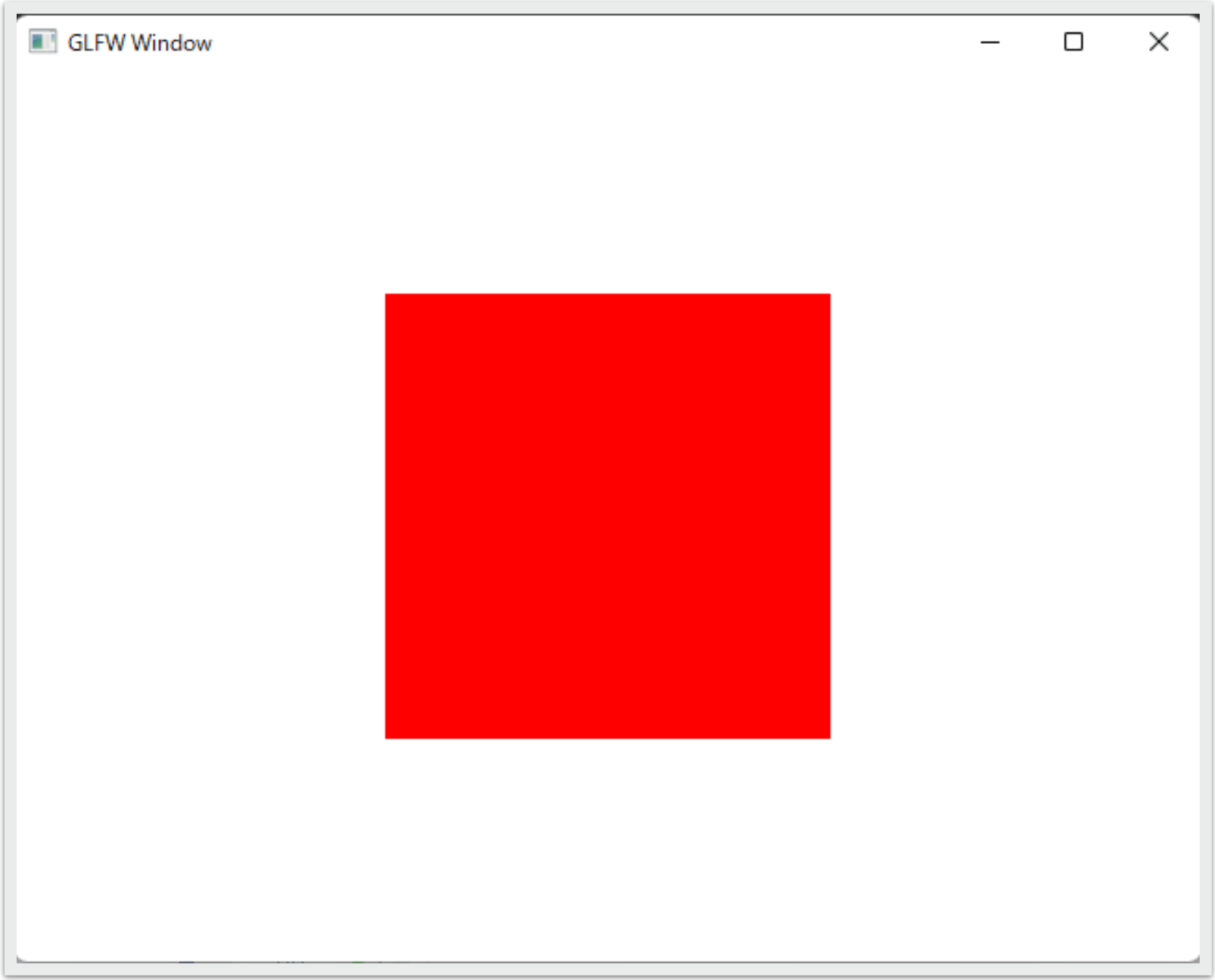
アスペクト比を uniform 変数に設定する (oglcv.cpp)

```
    // 画面クリア
    glClear(GL_COLOR_BUFFER_BIT);

    // プログラムオブジェクトの指定
    glUseProgram(program);

    // ウィンドウのアスペクト比
    const GLfloat aspect{ GLfloat(window.getWidth()) / GLfloat(window.getHeight()) };

    // uniform 変数に値を設定する
    glUniform1f(aspectLoc, aspect);

    // 頂点配列オブジェクトの指定
    glBindVertexArray(vao);

    // 図形の描画
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

window.getAspect() というメソッドも用意してあります

正方形になる

ウィンドウをリサイズしても
アスペクト比は維持される



画像の読み込みとテクスチャ

OpenCV の導入

OpenCV の準備 (oglcv.cpp)

```
// 補助プログラム
#include "GgApp.h"

// OpenCV
#include "opencv2/opencv.hpp"
#if defined(_MSC_VER)
#  define CV_VERSION_STR ¥
     CVAUX_STR(CV_MAJOR_VERSION) CVAUX_STR(CV_MINOR_VERSION) CVAUX_STR(CV_SUBMINOR_VERSION)
#  if defined(_DEBUG)
#    define CV_EXT_STR "d.lib"
#  else
#    define CV_EXT_STR ".lib"
#  endif
#  pragma comment(lib, "opencv_core" CV_VERSION_STR CV_EXT_STR)
#  pragma comment(lib, "opencv_imgcodecs" CV_VERSION_STR CV_EXT_STR)
#endif
```

PDF からコピペすると多分 ¥ がバックスラッシュ 0x5C にならず円記号 0xA5 になるので消して打ち直すか１行で書いてください

テクスチャの作成 (oglcv.cpp)

```
  // uniform 変数の場所を調べる
  const GLint aspectLoc{ glGetUniformLocation(program, "aspect") };
  const GLint imageLoc{ glGetUniformLocation(program, "image") };

  // 画像の読み込み
  cv::Mat image{ cv::imread("image.jpg") };
  if (image.empty()) throw std::runtime_error("Cannot open image file.");

  // テクスチャの準備
  GLuint texture;
  glGenTextures(1, &texture);
  glBindTexture(GL_TEXTURE_2D, texture);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image.cols, image.rows, 0,
    GL_BGR, GL_UNSIGNED_BYTE, image.data);

  // テクスチャを拡大・縮小する方法の指定
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

シェーダでテクスチャを参照するのに使うサンプラの変数

テクスチャユニットの指定 (oglcv.cpp)

```
    // ウィンドウのアスペクト比
    const GLfloat aspect{ GLfloat(window.getWidth()) / GLfloat(window.getHeight()) };

    // uniform 変数に値を設定する
    glUniform1f(aspectLoc, aspect);
    glUniform1i(imageLoc, 0);

    // テクスチャユニットを指定する
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);

    // 頂点配列オブジェクトの指定
    glBindVertexArray(vao);

    // 図形の描画
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

サンプラの uniform 変数にはテクスチャユニットの番号 (GL_TEXTURE0) を設定します

テクスチャ座標の生成 (ortho.vert)

```
#version 410

// 頂点の位置
layout (location = 0) in vec4 position;

// ウィンドウのアスペクト比
uniform float aspect;

// テクスチャ座標
out vec2 texcoord;

void main()
{
  gl_Position = vec4(0.5 / aspect, 0.5, 0.5, 1.0) * position;
  texcoord = position.xy * 0.5 + 0.5;
}
```

テクスチャ座標は 0～1 なのに対して position は ±1 なので 1/2 倍して 1/2 を足しています

テクスチャのサンプリング (ortho.frag)

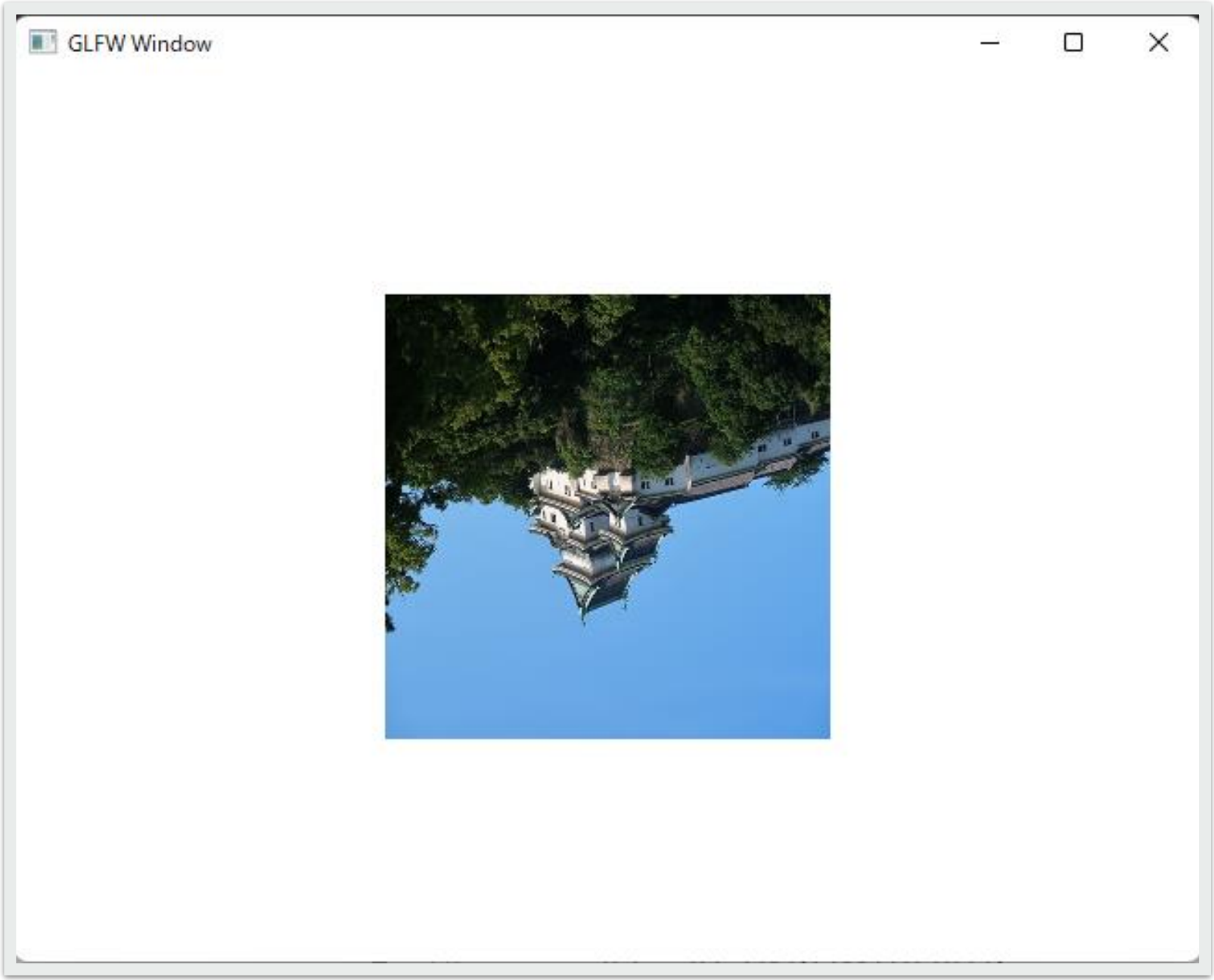
```
#version 410

// 画素の色
layout (location = 0) out vec4 color;

// サンプラー
uniform sampler2D image;

// テクスチャ座標
in vec2 texcoord;

void main()
{
  color = texture(image, texcoord);
}
```

テクスチャ座標の画素の位置における補間値

画像が貼り付けられる

上下が反転している

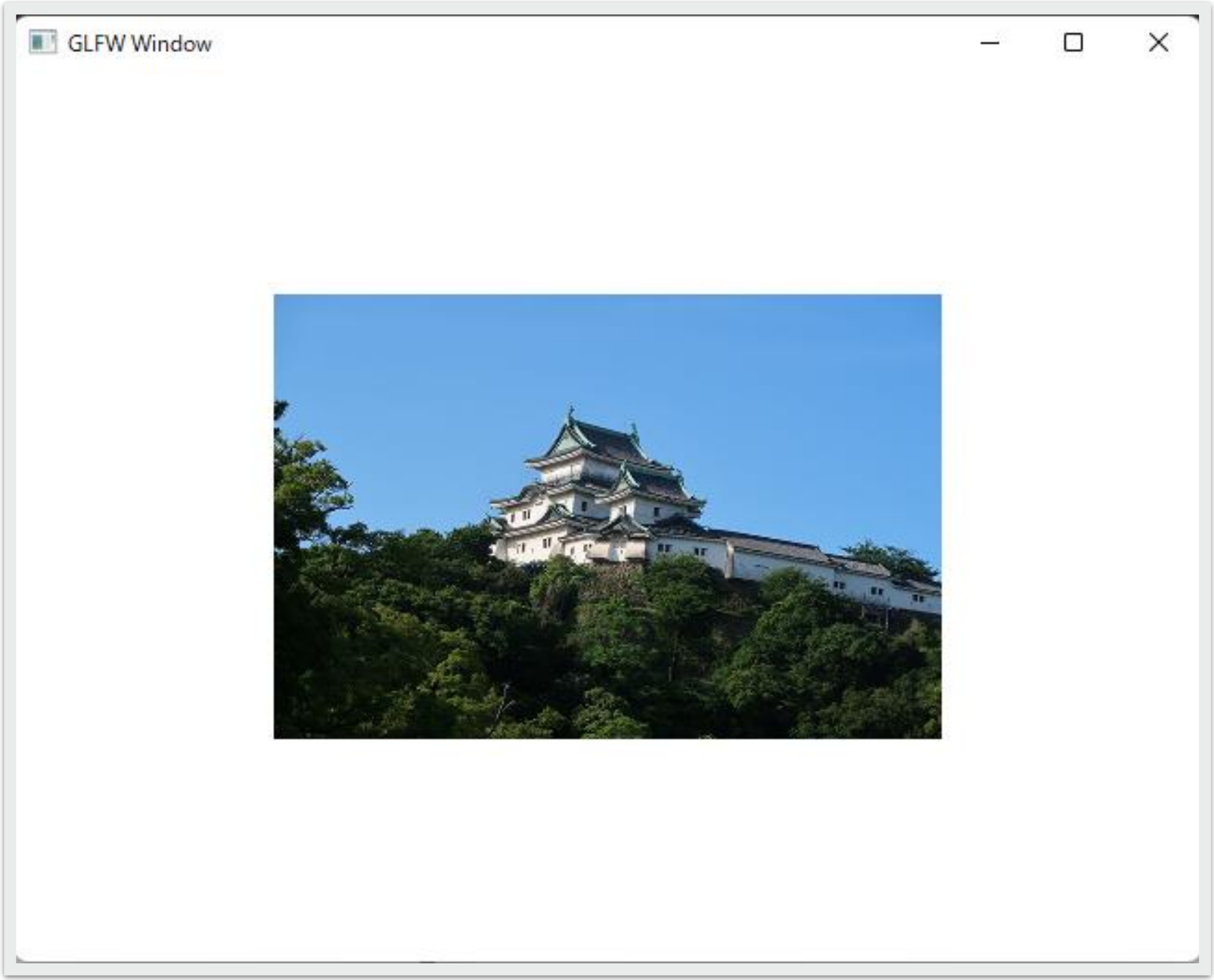
画像のアスペクト比が反映されていない



上下反転してアスペクト比を合わせる (ortho.vert)

```
#version 410

// 頂点の位置
layout (location = 0) in vec4 position;

// ウィンドウのアスペクト比
uniform float aspect;

// テクスチャのサンプラ
uniform sampler2D image;

// テクスチャ座標
out vec2 texcoord;

void main()
{
  vec2 size = vec2(textureSize(image, 0));
  gl_Position = position * vec4(size.x / size.y * 0.5 / aspect, 0.5, 0.5, 1.0);
  texcoord = position.xy * vec2(0.5, -0.5) + 0.5;
}
```

元の画像と同じ



cv::flip() で画像自体を反転してもよい (oglcv.cpp)

```
// 画像の読み込み
cv::Mat image{ cv::imread("image.jpg") };
if (image.empty()) throw std::runtime_error("Cannot open image file.");

// テクスチャの反転
cv::flip(image, image, 0);

// テクスチャの準備
GLuint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image.cols, image.rows, 0,
  GL_BGR, GL_UNSIGNED_BYTE, image.data);

// テクスチャを拡大・縮小する方法の指定
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

シェーダでテクスチャ座標を反転する代わりに画像自体を反転します

image.isContinuous() == false のときは image にギャップ存在するので、テクスチャとして正常に読み込まれません。ギャップを埋めるには copyTo() や メソッドflip() で深いコピーを作ってください。

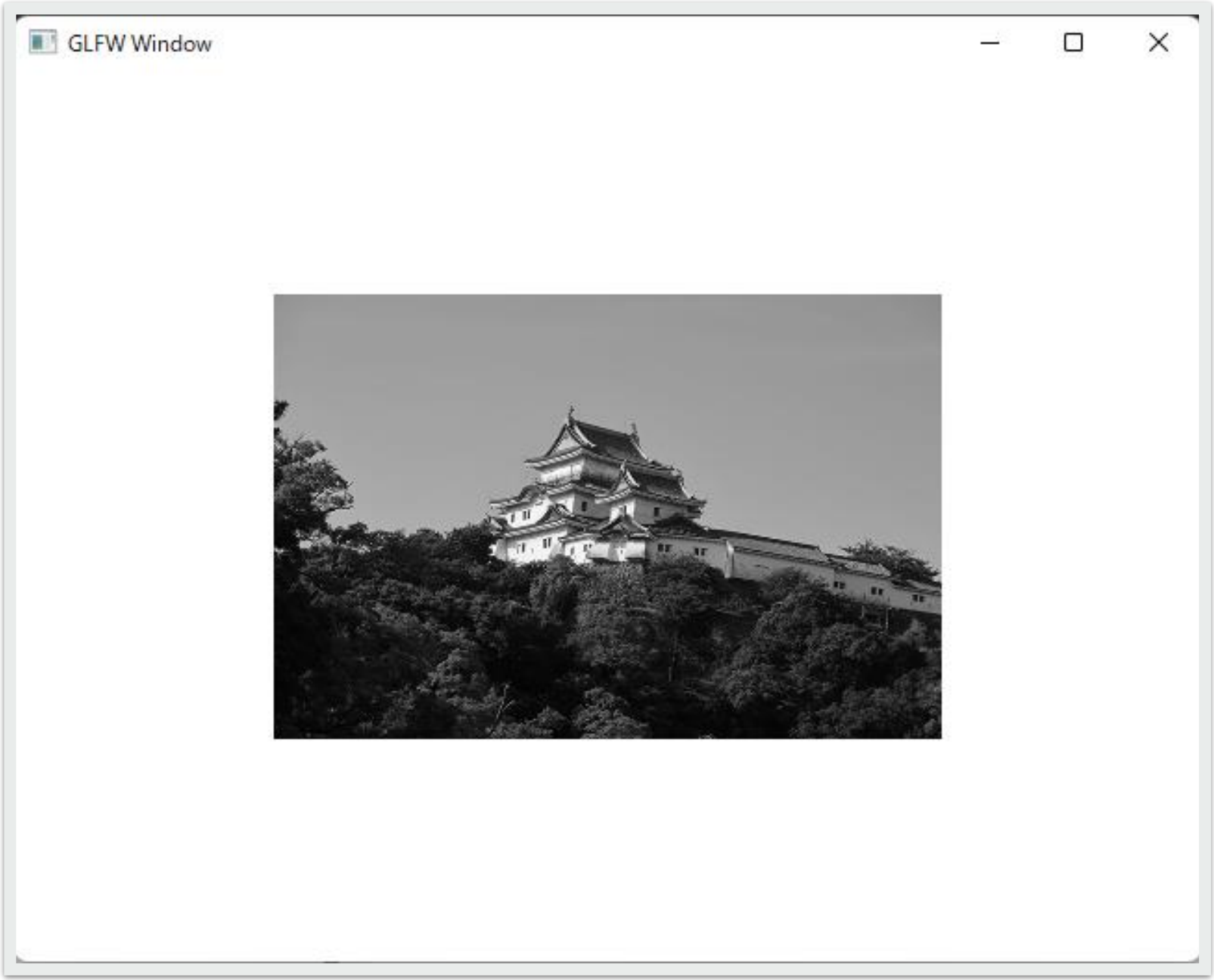
OpenGL による画像処理

表示画像を加工する

グレースケール化

```
#version 410

// 画素の色
layout (location = 0) out vec4 color;

// サンプラー
uniform sampler2D image;

// テクスチャ座標
in vec2 texcoord;

void main()
{
  // ITU-R BT.601
  float gray = dot(vec3(0.299, 0.587, 0.114), texture(image, texcoord).rgb);
  color = vec4(vec3(gray), 1.0);
}
```

グレースケール化



境界線抽出 (ortho.frag)

```
#version 410

// 画素の色
layout (location = 0) out vec4 color;

// サンプラー
uniform sampler2D image;

// テクスチャ座標
in vec2 texcoord;

void main()
{
  // ITU-R BT.601
  float gray = dot(vec3(0.299, 0.587, 0.114), texture(image, texcoord).rgb);
  color = vec4(vec3(fwidth(gray)), 1.0);
}
```

境界線抽出

ウィンドウの上下いっぱいに表示する (ortho.vert)

```
#version 410

// 頂点の位置
layout (location = 0) in vec4 position;

// ウィンドウのアスペクト比
uniform float aspect;

// テクスチャのサンプラ
uniform sampler2D image;

// テクスチャ座標
out vec2 texcoord;

void main()
{
  vec2 size = vec2(textureSize(image, 0));
  gl_Position = position * vec4(size.x / size.y / aspect, 1.0, 1.0, 1.0);
  texcoord = position.xy * vec2(0.5, -0.5) + 0.5;
}
```

ウィンドウの上下いっぱいに表示