③OpenGL による動画像処理

床井浩平

こんなことを話します

- OpenCV による動画像の入力
- 非同期処理
- オフスクリーンでの画像処理
- 簡単な動画像処理
- 表示画像の保存

動画像の入力

ビデオキャプチャ

ビデオ入出力ライブラリの追加 (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)
#  pragma comment(lib, "opencv_videoio" CV_VERSION_STR CV_EXT_STR)
#endif
```

キャプチャデバイスの準備 (oglcv.cpp)

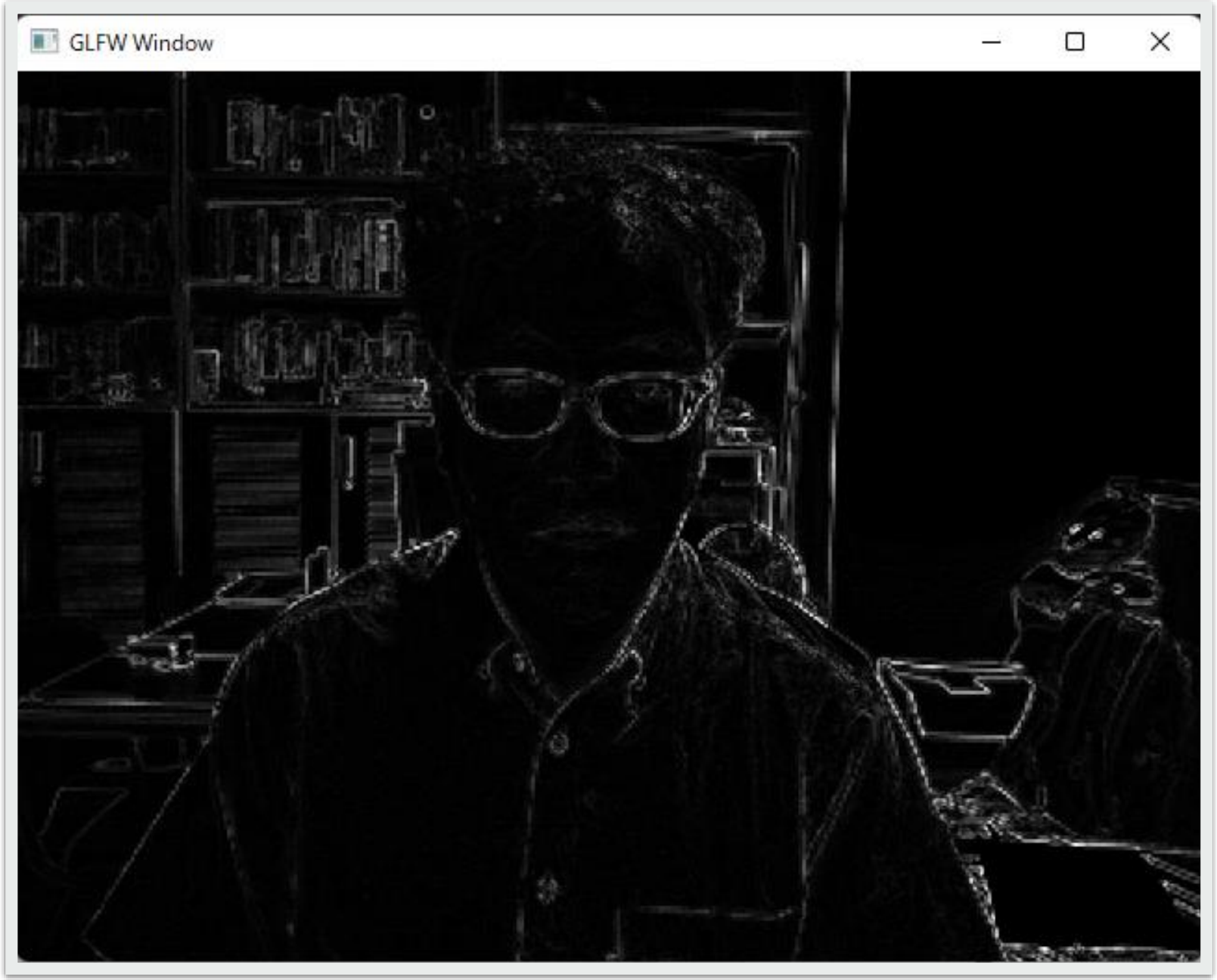
```
// uniform 変数の場所を調べる
const GLint aspectLoc{ glGetUniformLocation(program, "aspect") };
const GLint imageLoc{ glGetUniformLocation(program, "image") };

// キャプチャデバイスの準備
cv::VideoCapture camera;
if (!camera.open(0)) throw std::runtime_error("The capture device cannot be used.");

// キャプチャデバイスから 1 フレーム取り込む
cv::Mat image;
if (!camera.read(image)) throw std::runtime_error("No frames has been grabbed.");

// 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);
```

デフォルトのキャプチャデバイス以外を使うときは camera.open(0) の 0 を 1 以上にしてください

キャプチャデバイスの解像度を調べるために 1 フレーム読み込みます

動かない

フレームを更新していない



フレームの更新 (oglcv.cpp)

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

    // カメラのフレームが取り込めたら
    if (camera.read(image))
    {
      // テクスチャユニットを指定する
      glActiveTexture(GL_TEXTURE0);
      glBindTexture(GL_TEXTURE_2D, texture);

      // テクスチャに転送する
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, image.cols, image.rows,
        GL_BGR, GL_UNSIGNED_BYTE, image.data);
    }

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

テクスチャをバンドしてから画像を転送します

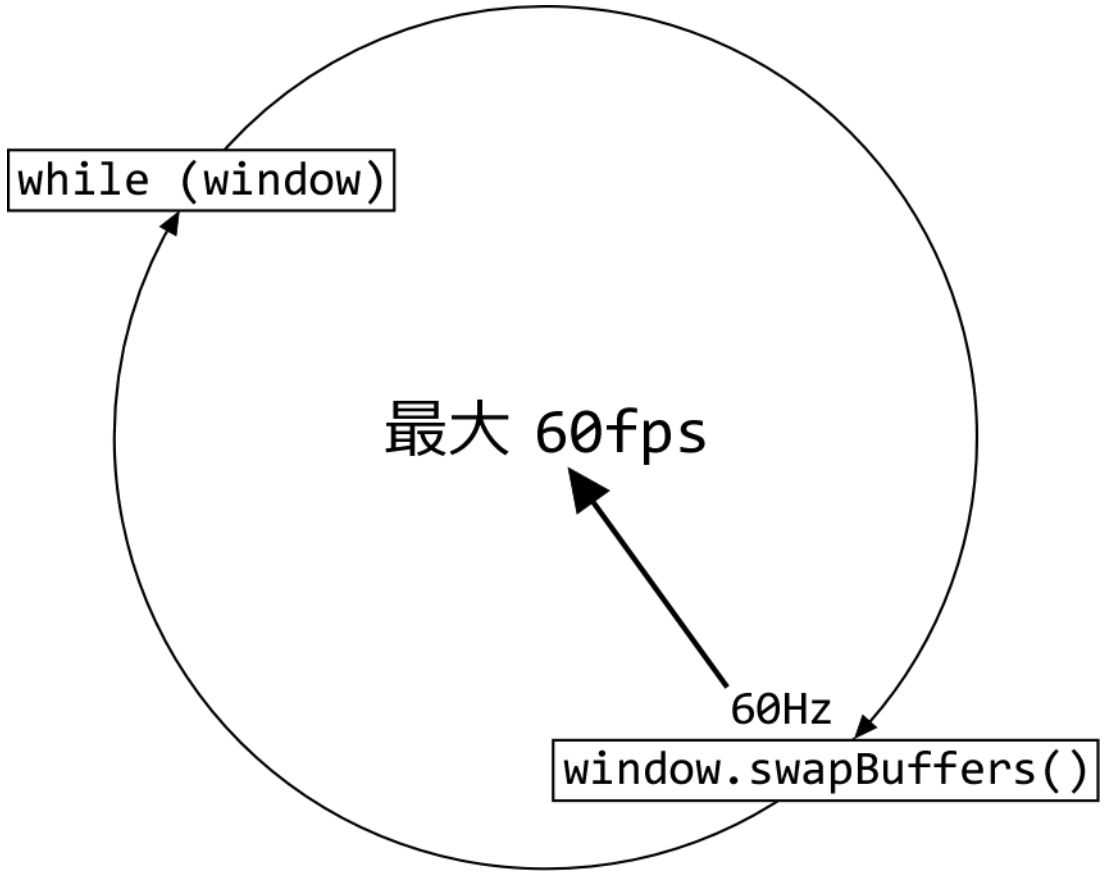
動く

非同期処理

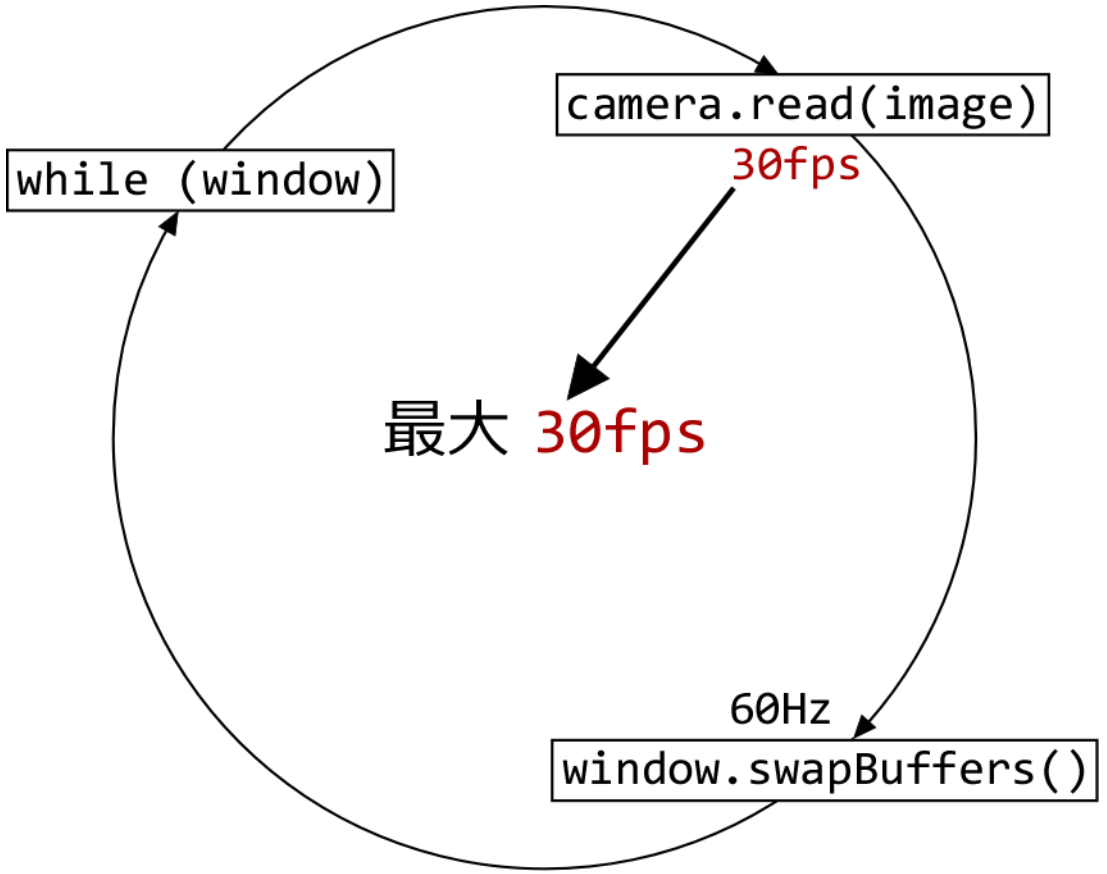
動画像の入力と図形表示

カメラとディスプレイはフレームレートが異なる

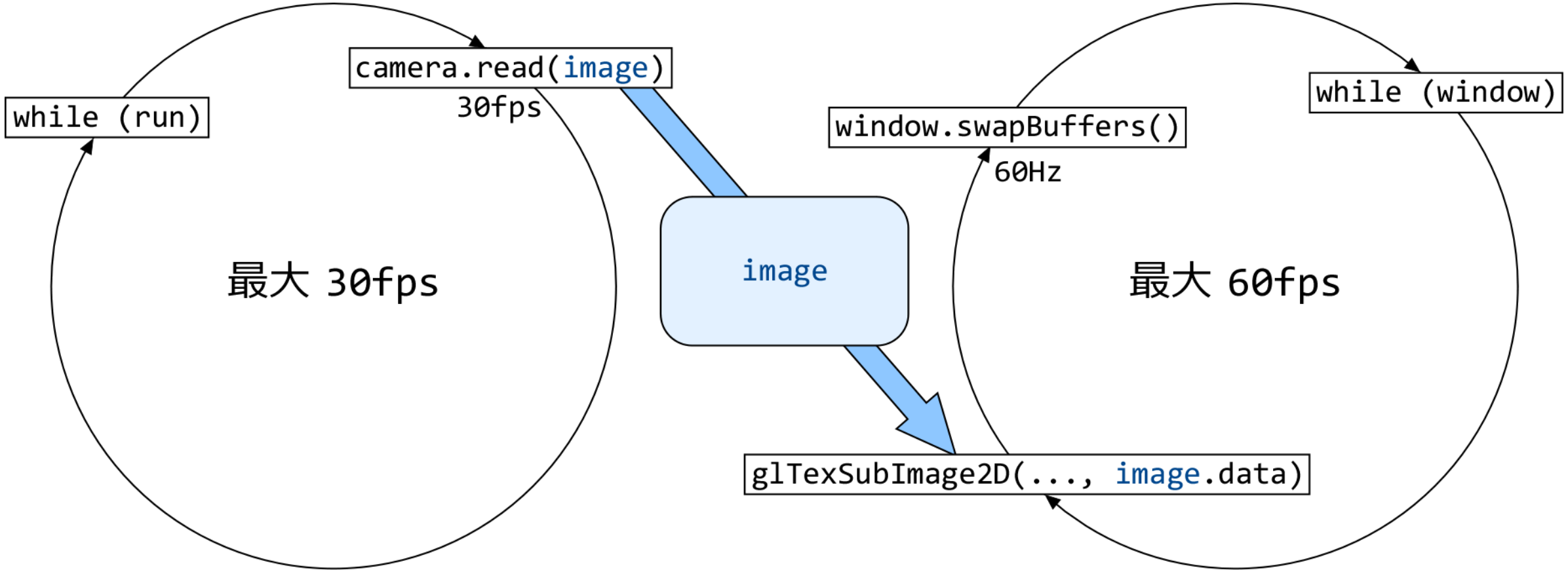
ディスプレイへの表示のみ行う



入力してから表示する



映像入力を表示処理と非同期で実行する



スレッドを使う (oglcv.cpp)

```
// 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)
#  pragma comment(lib, "opencv_videoio" CV_VERSION_STR CV_EXT_STR)
#endif

// スレッド
#include <thread>
```

キャプチャスレッドの起動 (oglcv.cpp)

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

  // キャプチャスレッド起動
  bool run{ true };
  bool update{ false };
  auto capture{ std::thread([&] {
    while (run) {
      update = camera.read(image);
    }
  }) };

  // 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);
```

キャプチャの完了を通知します

フレームをテクスチャに転送 (oglcv.cpp)

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

    // カメラのフレームが更新されたら
    if (update)
    {
      // テクスチャユニットを指定する
      glActiveTexture(GL_TEXTURE0);
      glBindTexture(GL_TEXTURE_2D, texture);

      // テクスチャに転送する
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, image.cols, image.rows,
        GL_BGR, GL_UNSIGNED_BYTE, image.data);

      // 転送完了を通知する
      update = false;
    }
```

終了時にキャプチャスレッドを停止 (oglcv.cpp)

```
    // 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();
  }

  // キャプチャスレッド停止
  run = false;
  capture.join();

  return EXIT_SUCCESS;
}
```

ここはウィンドウを閉じたときに実行されますが例外が発生したときは実行されません

アトミック変数を使った排他処理 (oglcv.cpp)

```
// 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)
#  pragma comment(lib, "opencv_videoio" CV_VERSION_STR CV_EXT_STR)
#endif

// スレッド
#include <thread>
#include <atomic>
```

アトミック変数でデータ取り出しをロック (oglcv.cpp)

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

// キャプチャスレッド起動
bool run{ true };
bool update{ false };
std::atomic<bool> lock{ false };
auto capture{ std::thread([&] {
  while (run) {
    update = camera.grab() && !lock.exchange(true) && camera.retrieve(image);
    lock.store(false);
  }
}) };

// ウィンドウが開いている間繰り返す
while (window)
{
```

camera.read(image) をカメラからのキャプチャ camera.grab() と image への格納 camera.retrieve(image) に分けます

image は描画処理でも参照しますが、camera.read(image) を使うとキャプチャが完了するまでこのスレッドで image が占有されてしまい、描画処理のときに image を参照する時間が確保できなくなります

フレームをテクスチャに転送 (oglcv.cpp)

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

    // カメラのフレームが更新されたら
    if (update && !lock.exchange(true))
    {
      // テクスチャユニットを指定する
      glActiveTexture(GL_TEXTURE0);
      glBindTexture(GL_TEXTURE_2D, texture);

      // テクスチャに転送する
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, image.cols, image.rows,
        GL_BGR, GL_UNSIGNED_BYTE, image.data);

      // 転送完了を通知する
      update = false;
      lock.store(false);
```

【参考】ミューテックスを使った排他処理 (oglcv.cpp)

```
// 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)
#  pragma comment(lib, "opencv_videoio" CV_VERSION_STR CV_EXT_STR)
#endif

// スレッド
#include <thread>
#include <mutex>
```

【参考】データ取り出しをロック (oglcv.cpp)

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

// キャプチャスレッド起動
bool run{ true };
bool update{ false };
std::mutex mtx;
auto capture{ std::thread([&] {
  while (run) {
    if (camera.grab()) {
      std::lock_guard<std::mutex> guard{ mtx };
      update = camera.retrieve(image);
    }
  }
}) };

// ウィンドウが開いている間繰り返す
while (window)
```

【参考】フレームをテクスチャに転送 (oglcv.cpp)

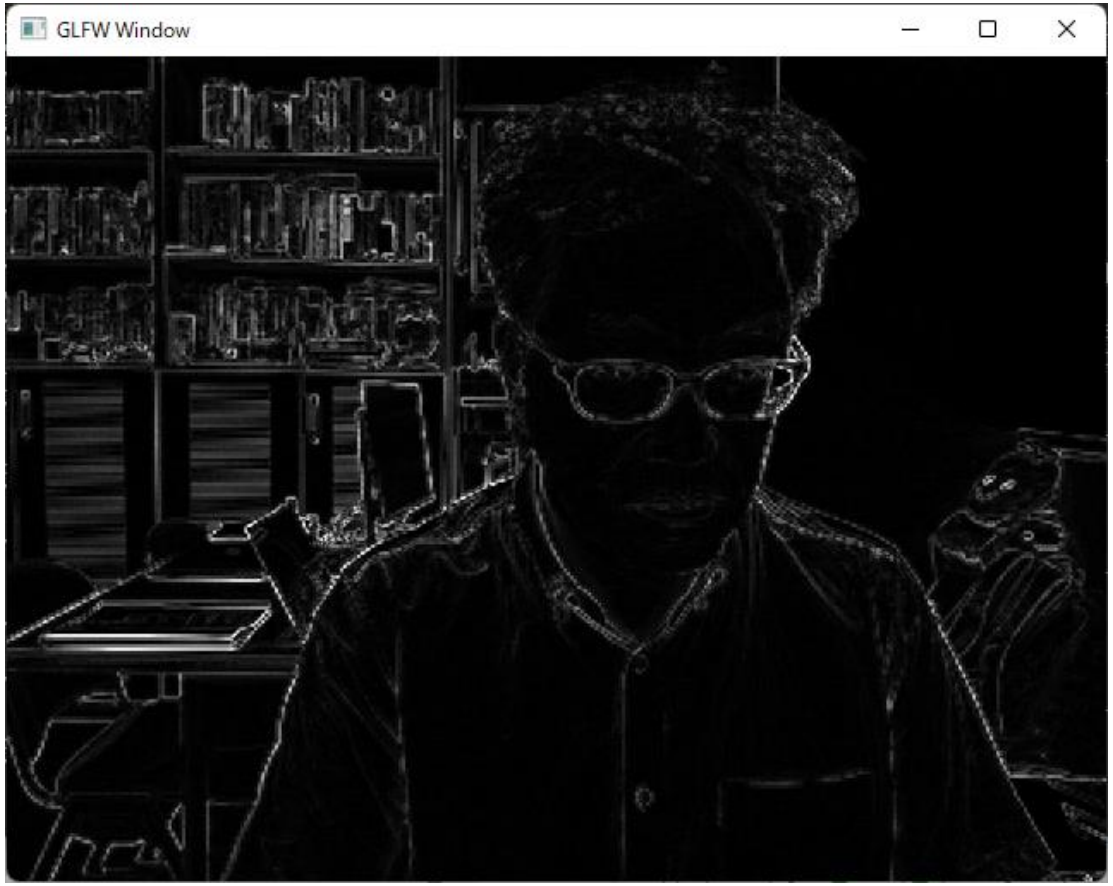
```
// uniform 変数に値を設定する
glUniform1f(aspectLoc, aspect);
glUniform1i(imageLoc, 0);

// カメラのフレームが更新されたら
if (update && mtx.try_lock())
{
  // テクスチャユニットを指定する
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, texture);

  // テクスチャに転送する
  glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, image.cols, image.rows,
    GL_BGR, GL_UNSIGNED_BYTE, image.data);

  // 転送完了を通知する
  update = false;
  mtx.unlock();
```

フレームバッファオブジェクト

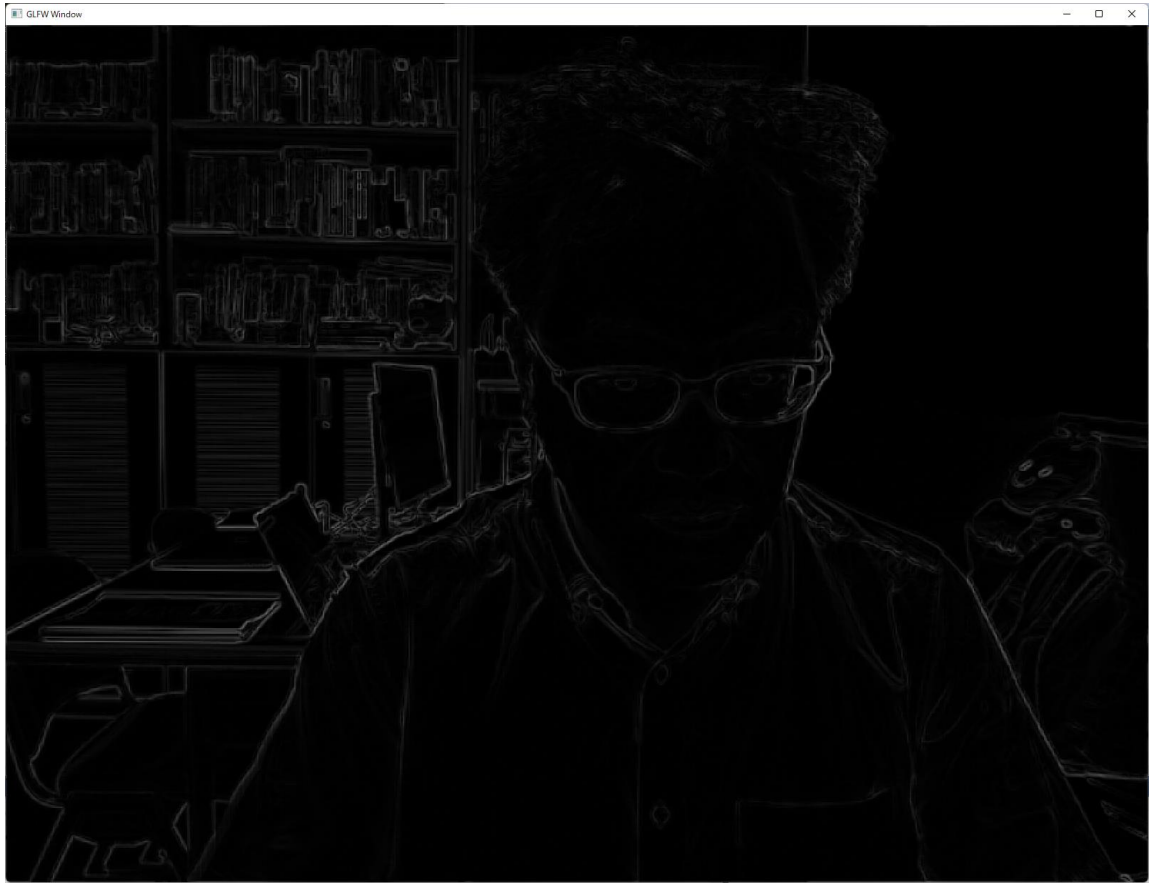
オフスクリーンの画像処理

処理解像度がウィンドウサイズに依存する

640 × 480



1600 × 1200



フレームバッファオブジェクトのサイズ (oglcv.cpp)

```
// 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);

// テクスチャをサンプリングする方法の指定
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// フレームバッファオブジェクトのサイズ
const GLsizei fboWidth{ image.cols * 4 };
const GLsizei fboHeight{ image.rows * 4 };
```

仮に入力画像の４倍くらいにしてみます

カラーバッファに使うテクスチャの準備 (oglcv.cpp)

```
// フレームバッファオブジェクトのサイズ
const GLsizei fboWidth{ image.cols * 4 };
const GLsizei fboHeight{ image.rows * 4 };

// フレームバッファオブジェクトのカラーバッファに使うテクスチャ
GLuint color;
glGenTextures(1, &color);
glBindTexture(GL_TEXTURE_2D, color);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, fboWidth, fboHeight, 0,
  GL_BGR, GL_UNSIGNED_BYTE, 0);

// テクスチャをサンプリングする方法の指定
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

フレームバッファオブジェクトの作成 (oglcv.cpp)

```
// フレームバッファオブジェクトの準備
GLuint fbo;
glGenFramebuffers(1, &fbo);
glBindFramebuffer(GL_FRAMEBUFFER, fbo);

// フレームバッファオブジェクトにカラーバッファに使うテクスチャを組み込む
glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, color, 0);

// レンダリングターゲット
const GLenum bufs[] = { GL_COLOR_ATTACHMENT0 };

// 標準のフレームバッファに戻す
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

フレームバッファオブジェクトへの描画 (oglcv.cpp)

```
  // ウィンドウが開いている間繰り返す
  while (window)
  {
      // フレームバッファオブジェクトへの描画
      glBindFramebuffer(GL_FRAMEBUFFER, fbo);

      // このフレームバッファオブジェクトにはデプスバッファが無い
      glDepthMask(GL_FALSE);

      // ビューポートをフレームバッファオブジェクトに合わせる
      glViewport(0, 0, fboWidth, fboHeight);

      // 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(fboWidth) / GLfloat(fboHeight) };
```

ディスプレイには表示されなくなります

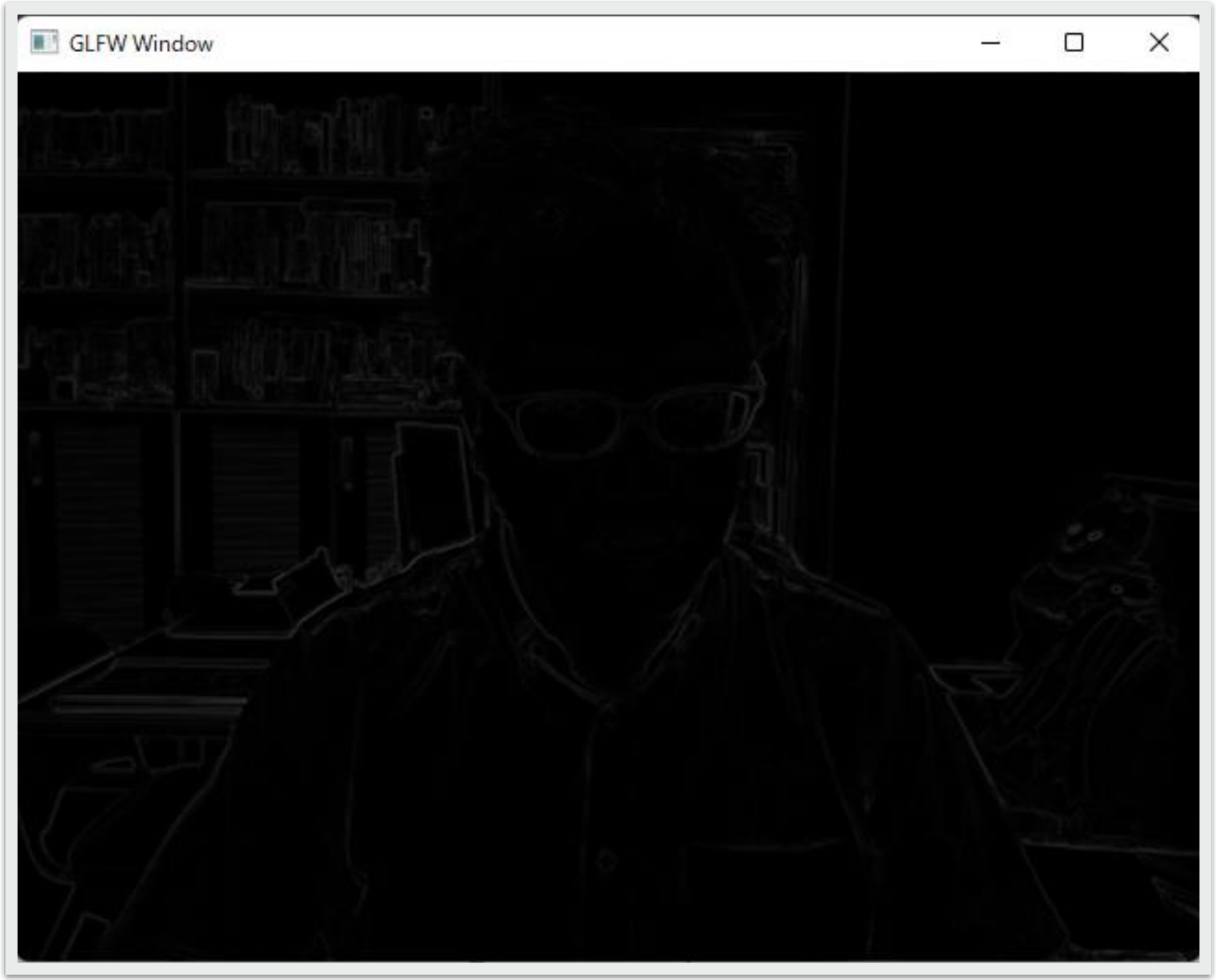
これ以降はフレームバッファオブジェクトに描画

フレームバッファオブジェクトのアスペクト比

描画結果をフレームバッファに転送 (oglcv.cpp)

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

    // レンダーターゲットの指定
    glDrawBuffers(std::size(bufs), bufs);

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

    // 標準のフレームバッファへの転送
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo);
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBlitFramebuffer(0, 0, fboWidth, fboHeight,
      0, 0, window.getWidth(), window.getHeight(), GL_COLOR_BUFFER_BIT, GL_LINEAR);

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

フレームバッファオブジェクトの内容



描画結果をウィンドウの中央に配置 (oglcv.cpp)

```
    // 標準のフレームバッファへの転送
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo);
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    GLsizei w{ window.getWidth() };
    GLsizei h{ window.getHeight() };
    GLsizei x{ w / 2 };
    GLsizei y{ h / 2 };
    float t{ h * aspect };
    if (h * aspect > w) h = w / aspect; else w = t;
    x -= w / 2;
    y -= h / 2;
    glBlitFramebuffer(0, 0, fboWidth, fboHeight,
      x, y, x + w, y + h, GL_COLOR_BUFFER_BIT, GL_LINEAR);
```

【参考】フィルタの例 (ortho.frag)

```
#version 410

//
// ラプラシアンフィルタ
//

// 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) * 4.0
        - textureOffset(image, texcoord, ivec2( 1,  0))
        - textureOffset(image, texcoord, ivec2(-1,  0))
        - textureOffset(image, texcoord, ivec2( 0,  1))
        - textureOffset(image, texcoord, ivec2( 0, -1));
}
```

```
#version 410

//
// 輪郭強調フィルタ
//

// 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) * 5.0
        - textureOffset(image, texcoord, ivec2( 1,  0))
        - textureOffset(image, texcoord, ivec2(-1,  0))
        - textureOffset(image, texcoord, ivec2( 0,  1))
        - textureOffset(image, texcoord, ivec2( 0, -1));
}
```

背景差分

前景を切出すための前処理

二つのテクスチャの差の絶対値を表示する (ortho.frag)

```
#version 410

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

// サンプラー
uniform sampler2D image;
uniform sampler2D past;

// テクスチャ座標
in vec2 texcoord;

void main()
{
  color = vec4(abs(texture(image, texcoord) - texture(past, texcoord)).rgb, 1.0);
}
```

二つ目のテクスチャのサンプラ

二つのテクスチャの画素の差の絶対値を出力します

二つ目のサンプラの uniform 変数の場所 (oglcv.cpp)

```
// 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") };
const GLint imageLoc{ glGetUniformLocation(program, "image") };
const GLint nextLoc{ glGetUniformLocation(program,  “past”) };

// キャプチャデバイスの準備
cv::VideoCapture camera;
if (!camera.open(0)) throw std::runtime_error("The capture device cannot be used.");

// キャプチャデバイスから 1 フレーム取り込む
cv::Mat image;
if (!camera.read(image)) throw std::runtime_error("No frames has been grabbed.");
```

テクスチャを二つ用意する (oglcv.cpp)

```
  // キャプチャデバイスから 1 フレーム取り込む
  cv::Mat image;
  if (!camera.read(image)) throw std::runtime_error("No frames has been grabbed.");

  // テクスチャの準備
  std::array<GLuint, 2> textures;
  glGenTextures(textures.size(), textures.data());
  for (const auto texture : textures)
  {
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image.cols, image.rows, 0,
      GL_BGR, GL_UNSIGNED_BYTE, image.data);

    // テクスチャをサンプリングする方法の指定
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  }
```

二つ目のテクスチャに画像を転送する (oglcv.cpp)

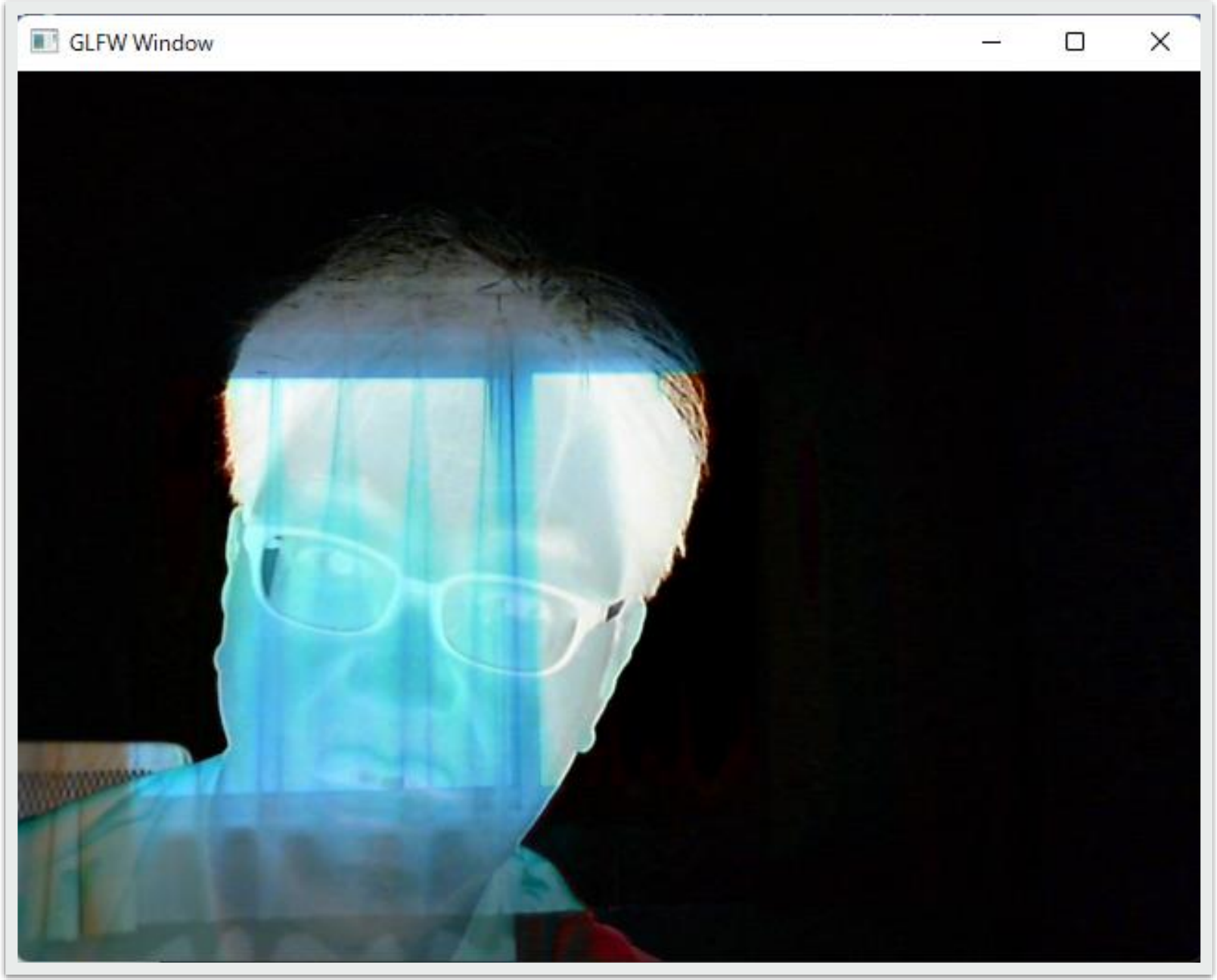
```
    // uniform 変数に値を設定する
    glUniform1f(aspectLoc, aspect);
    glUniform1i(imageLoc, 0);
    glUniform1i(nextLoc, 1);

    // カメラからフレームを取り込んでテクスチャに転送する
    if (update.load())
    {
      // テクスチャユニットを指定する
      glActiveTexture(GL_TEXTURE0);
      glBindTexture(GL_TEXTURE_2D, textures[0]);
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, image.cols, image.rows,
        GL_BGR, GL_UNSIGNED_BYTE, image.data);
      update.store(false);
    }

    // 二つ目のテクスチャユニットを指定する
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, textures[1]);
```

スペースキーで元画像を更新する (oglcv.cpp)

```
    // テクスチャユニット番号
    const int unit{ glfwGetKey(window.get(), GLFW_KEY_SPACE) == GLFW_RELEASE ? 1 : 0 };

    // カメラからフレームを取り込んでテクスチャに転送する
    if (update.load())
    {
      // テクスチャユニットを指定する
      glActiveTexture(GL_TEXTURE0 + unit);
      glBindTexture(GL_TEXTURE_2D, textures[unit]);
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, image.cols, image.rows,
        GL_BGR, GL_UNSIGNED_BYTE, image.data);
      update.store(false);
    }

    // 二つ目のテクスチャユニットを指定する
    glActiveTexture(GL_TEXTURE1 - unit);
    glBindTexture(GL_TEXTURE_2D, textures[1 - unit]);
```

背景フレームとの差分

前景の検出



フレーム間差分

動いているものを見つけるための前処理

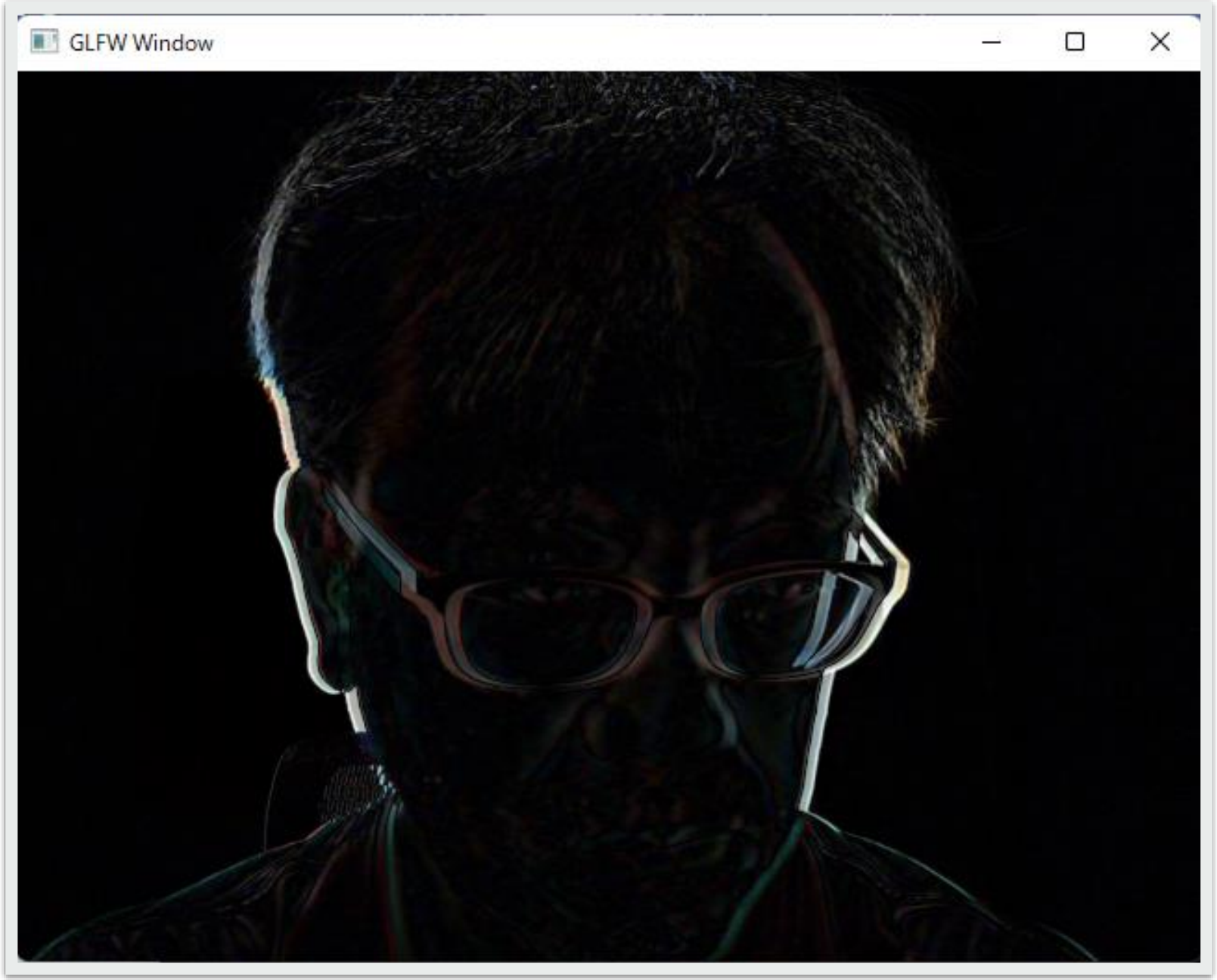
フレーム単位でテクスチャユニットを切替 (oglcv.cpp)

```
    // テクスチャユニット番号
    static int unit{ 0 };

    // カメラからフレームを取り込んでテクスチャに転送する
    if (update.load())
    {
      // テクスチャユニットを指定する
      glActiveTexture(GL_TEXTURE0 + unit);
      glBindTexture(GL_TEXTURE_2D, textures[unit]);
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, image.cols, image.rows,
        GL_BGR, GL_UNSIGNED_BYTE, image.data);
      update.store(false);

      // テクスチャユニットを入れ替える
      unit = 1 - unit;
    }

    // 二つ目のテクスチャユニットを指定する
    glActiveTexture(GL_TEXTURE0 + unit);
    glBindTexture(GL_TEXTURE_2D, textures[unit]);
```

無くても動きます

全フレームとの差分

動きの検出



表示画像の保存

フレームバッファオブジェクトの内容の読み出し

フレームバッファの内容を保存する (oglcv.cpp)

```
    // キャプチャスレッド停止
    run = false;
    capture.join();

    // フレームバッファオブジェクトからの読み出し
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo);
    cv::Mat result{ cv::Size2i(fboWidth, fboHeight), CV_8UC3 };
    glReadPixels(0, 0, fboWidth, fboHeight, GL_BGR, GL_UNSIGNED_BYTE, result.data);

    // 上下を反転して保存
    cv::flip(result, result, 0);
    cv::imwrite("result.jpg", result);

    return EXIT_SUCCESS;
}
```

0 を指定すればウィンドウの表示内容を保存できます

【参考】テクスチャの内容を保存する (oglcv.cpp)

```
  // キャプチャスレッド停止
  run = false;
  capture.join();

  // テクスチャからの読み出し
  glBindTexture(GL_TEXTURE_2D, color);
  cv::Mat result{ cv::Size2i(fboWidth, fboHeight), CV_8UC3 };
  glGetTexImage(GL_TEXTURE_2D, 0, GL_BGR, GL_UNSIGNED_BYTE, result.data);

  // 上下を反転して保存
  cv::flip(result, result, 0);
  cv::imwrite("result.jpg", result);

  return EXIT_SUCCESS;
}
```