⑤GPU メモリの利用

床井浩平

こんなことを話します

- ピクセルバッファオブジェクトによる GPU メモリの利用
  - Pixel Buffer Object (PBO)
- ArUco Maker による AR マーカ検出

ピクセルバッファオブジェクト (PBO)

- GPU 上に確保する汎用のメモリオブジェクト
  - GPU 上でのデータのやり取りを CPU を介さずに行う
  - GPU のメモリを CPU のメモリ空間にマップできる
    - GPU 上のデータが CPU の配列変数にあるように見せかけることができる
- フレームバッファオブジェクトやテクスチャは直接マップできない
  - 頂点バッファオブジェクトは可能
- ピクセルバッファオブジェクトにコピーすればマップして使える
  - フレームバッファオブジェクトやテクスチャとピクセルバッファオブジェクトのコピーは GPU 内で行われるので非常に高速
  - CPU のメモリとのデータ転送は DMA によるため高速かつ非同期に実行可能

PBO の作成 (oglcv.cpp)

```
// 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);

// フレームバッファの読み出しに使うピクセルバッファオブジェクトを作成する
GLuint pixel;
glGenBuffers(1, &pixel);

// フレームバッファと同じサイズのメモリを確保する
glBindBuffer(GL_PIXEL_PACK_BUFFER, pixel);
glBufferData(GL_PIXEL_PACK_BUFFER, fboWidth * fboHeight * 3, nullptr, GL_DYNAMIC_COPY);
glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);

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

データは送らず
メモリだけを確保します

GL_PIXEL_PACK_BUFFER　：コピー元→PBO
GL_PIXEL_UNPACK_BUFFER ：PBO→コピー先

FBO から PBO にデータをコピーする

```
    // 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]);

    // 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
    glDrawArraysInstanced(GL_TRIANGLE_STRIP, 0, 41 * 2 + 2, 31);

    // フレームバッファオブジェクトからピクセルバッファオブジェクトにコピー
    glBindBuffer(GL_PIXEL_PACK_BUFFER, pixel);
    glReadPixels(0, 0, fboWidth, fboHeight, GL_BGR, GL_UNSIGNED_BYTE, 0);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
```

PBO を GL_PIXEL_PACK_BUFFER で結合すれば glReadPixels() で読み出したデータの書き込み先が PBO になります

PBO の先頭に書き込みます

PBO のデータのコピー先を CPU のメモリ上に作る

```
// フレームバッファの読み出しに使うピクセルバッファオブジェクトを作成する
GLuint pixel;
glGenBuffers(1, &pixel);

// フレームバッファと同じサイズのメモリを確保する
glBindBuffer(GL_PIXEL_PACK_BUFFER, pixel);
glBufferData(GL_PIXEL_PACK_BUFFER, fboWidth * fboHeight * 3, nullptr, GL_DYNAMIC_COPY);
glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);

// ピクセルバッファオブジェクトのデータのコピー先
cv::Mat buffer{ cv::Size(fboWidth, fboHeight), CV_8UC3 };

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

PBO から CPU のメモリにデータをコピーする

```
    // 図形の描画
    glDrawArraysInstanced(GL_TRIANGLE_STRIP, 0, 41 * 2 + 2, 31);

    // フレームバッファオブジェクトからピクセルバッファオブジェクトにコピー
    glBindBuffer(GL_PIXEL_PACK_BUFFER, pixel);
    glReadPixels(0, 0, fboWidth, fboHeight, GL_BGR, GL_UNSIGNED_BYTE, 0);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);

    // ピクセルバッファオブジェクトから CPU のメモリにコピー
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pixel);
    glGetBufferSubData(GL_PIXEL_UNPACK_BUFFER, 0, fboWidth * fboHeight * 3, buffer.data);
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);

    // 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);
```

PBO を CPU から直接アクセスすることもできる

```
    // 図形の描画
    glDrawArraysInstanced(GL_TRIANGLE_STRIP, 0, 41 * 2 + 2, 31);

    // フレームバッファオブジェクトからピクセルバッファオブジェクトにコピー
    glBindBuffer(GL_PIXEL_PACK_BUFFER, pixel);
    glReadPixels(0, 0, fboWidth, fboHeight, GL_BGR, GL_UNSIGNED_BYTE, 0);

    // ピクセルバッファオブジェクトを CPU のメモリにマップ
    cv::Mat frame{ cv::Size(fboWidth, fboHeight), CV_8UC3,
      glMapBuffer(GL_PIXEL_PACK_BUFFER, GL_READ_WRITE) };

    // ピクセルバッファオブジェクトから CPU のメモリにコピー
    frame.copyTo(buffer);

    // ピクセルバッファオブジェクトを開放
    glUnmapBuffer(GL_PIXEL_PACK_BUFFER);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
```

frame は PBO (GPU 上のメモリ) ですが glMapBuffer() と glUnmapBuffer() の間では CPU 上のメモリと同様に扱えます

ArUco Maker ライブラリの組み込み

```
// 補助プログラム
#include "GgApp.h"

// OpenCV
#include "opencv2/opencv.hpp"
#include "opencv2/aruco.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)
#  pragma comment(lib, "opencv_aruco" CV_VERSION_STR CV_EXT_STR)
#endif
```

ArUco Marker の辞書・検出パラメータとID・位置

```
  // ピクセルバッファオブジェクトのデータのコピー先
  cv::Mat buffer{ cv::Size(fboWidth, fboHeight), CV_8UC3 };

  // ArUco Marker の辞書と検出パラメータ
  const cv::Ptr<cv::aruco::Dictionary>
    dictionary{ cv::aruco::getPredefinedDictionary(cv::aruco::DICT_6X6_1000) };
  const cv::Ptr<cv::aruco::DetectorParameters>
    parameters{ cv::aruco::DetectorParameters::create() };

  // 検出された ArUco Marker の ID と位置
  std::vector<int> ids;
  std::vector<std::vector<cv::Point2f>> corners, rejected;

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

ArUco Marker を検出する

```
    // 図形の描画
    glDrawArraysInstanced(GL_TRIANGLE_STRIP, 0, 41 * 2 + 2, 31);

    // フレームバッファオブジェクトからピクセルバッファオブジェクトにコピー
    glBindBuffer(GL_PIXEL_PACK_BUFFER, pixel);
    glReadPixels(0, 0, fboWidth, fboHeight, GL_BGR, GL_UNSIGNED_BYTE, 0);

    // ピクセルバッファオブジェクトを CPU のメモリにマップ
    cv::Mat frame{ cv::Size(fboWidth, fboHeight), CV_8UC3,
      glMapBuffer(GL_PIXEL_PACK_BUFFER, GL_READ_WRITE) };

    // マーカーを検出する
    cv::aruco::detectMarkers(frame, dictionary, corners, ids, parameters, rejected);

    // ピクセルバッファオブジェクトを開放
    glUnmapBuffer(GL_PIXEL_PACK_BUFFER);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
```

検出結果を画像に書き込む

```
    // ピクセルバッファオブジェクトを開放
    glUnmapBuffer(GL_PIXEL_PACK_BUFFER);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);

    // マーカーが見つかったら
    if (ids.size() > 0)
    {
        // 検出結果を表示に描き込む
        cv::aruco::drawDetectedMarkers(frame, corners, ids);
        glBindTexture(GL_TEXTURE_2D, color);
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pixel);
        glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, fboWidth, fboHeight, GL_BGR, GL_UNSIGNED_BYTE, 0);
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
        glBindTexture(GL_TEXTURE_2D, 0);
    }

    // 標準のフレームバッファへの転送
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo);
```

PBO を使用しているときは
0 で PBO の先頭から読み込みます

上下を反転してレンダリングする (ortho.vert)

```
#version 410

... （中略）

void main()
{
  int s = gl_VertexID & ~1;
  int t = (~gl_VertexID & 1) + gl_InstanceID << 1;
  vec2 position = vec2(s, t) / vec2(slices, stacks) - 1.0;

  // クリッピング空間いっぱいに描く
  gl_Position = vec4(position, 0.0, 1.0);

  // 大きさが aspect × 1.0 のスクリーン上の上下を反転した点の位置
  vec2 uv = position * vec2(aspect, -1.0);

  ... （中略）
}
```

テクスチャ座標の Y 座標を反転します

上下を反転してフレームバッファに転送する

```
    // 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 + h, x + w, y,
      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();
}
```

上下を反転せずに画像を保存する

```
  // 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);

  // 結果を保存
  cv::flip(result, result, 0);
  cv::imwrite("result.jpg", result);

  return EXIT_SUCCESS;
}
```


id=203
id=124
id=23
id=98
id=30