BAUM : Library for Recognizing Blur-Resistant Markers

by Naoki Shibata at Nara Institute of Science and Technology

Download

Latest version is **baum-1.10.tar.gz**

You can browse the source codes at GitHub.

Overview

Sometimes we want to use visual markers and cameras to track movement of objects or the camera itself. However, motion blur can be a problem in detecting markers when the camera or the object is moving. BAUM is a software library for drawing and recognizing newly designed circular barcodes that are relatively tolerant of linear motion blur. With this library, circular barcodes can be recognized in real time from Full HD videos utilizing a desktop GPU. With BAUM library, **all recognition task is executed by an OpenCL device**, and thus CPU load is very low if a GPU is used for recognition.

An example of BAUM circular barcode is shown in Fig. 1. With linear motion blur in the direction of top left to bottom right, this barcode can be captured by a camera as shown in Fig. 2. The part inside the red rectangle can be read as a 1-dimensional bar code.

Fig. 1 BAUM circular barcode

Fig. 2 Barcode with linear motion blur. The part inside red rectangle can be read as a 1-dimensional bar code.

Getting Started

It would be easier to understand by running a demo program. In the archive, precompiled executables for Windows OS (64 bit) are included. Please prepare a decent web camera (like Logitech C920) and an OpenCL-capable GPU to test them. nVidia GPUs work best, and Intel GPUs are not supported. If you don't have a supported OpenCL-capable GPU device, you can still try the software by installing a CPU OpenCL driver from here. Please print out exampleMarkers.pdf in the archive, or just open it on the display. Then, please execute runDemo.bat. It will first automatically make kernel execution plans for all available OpenCL devices, and then start the demo program. After the program starts, please select your camera from "Choose capture" list box, and your GPU from "Choose device" list box. Then, point your camera at the printed circular barcodes. The program will show the detected barcodes, as shown in Fig. 3. Shake the camera and see what happens. Please note that the barcodes and the recognition algorithm are tolerant of linear motion blur, but it is not very good at handling warping of video frames caused by rolling shutter.

If you cannot run the demo program, please watch our **YouTube video**.

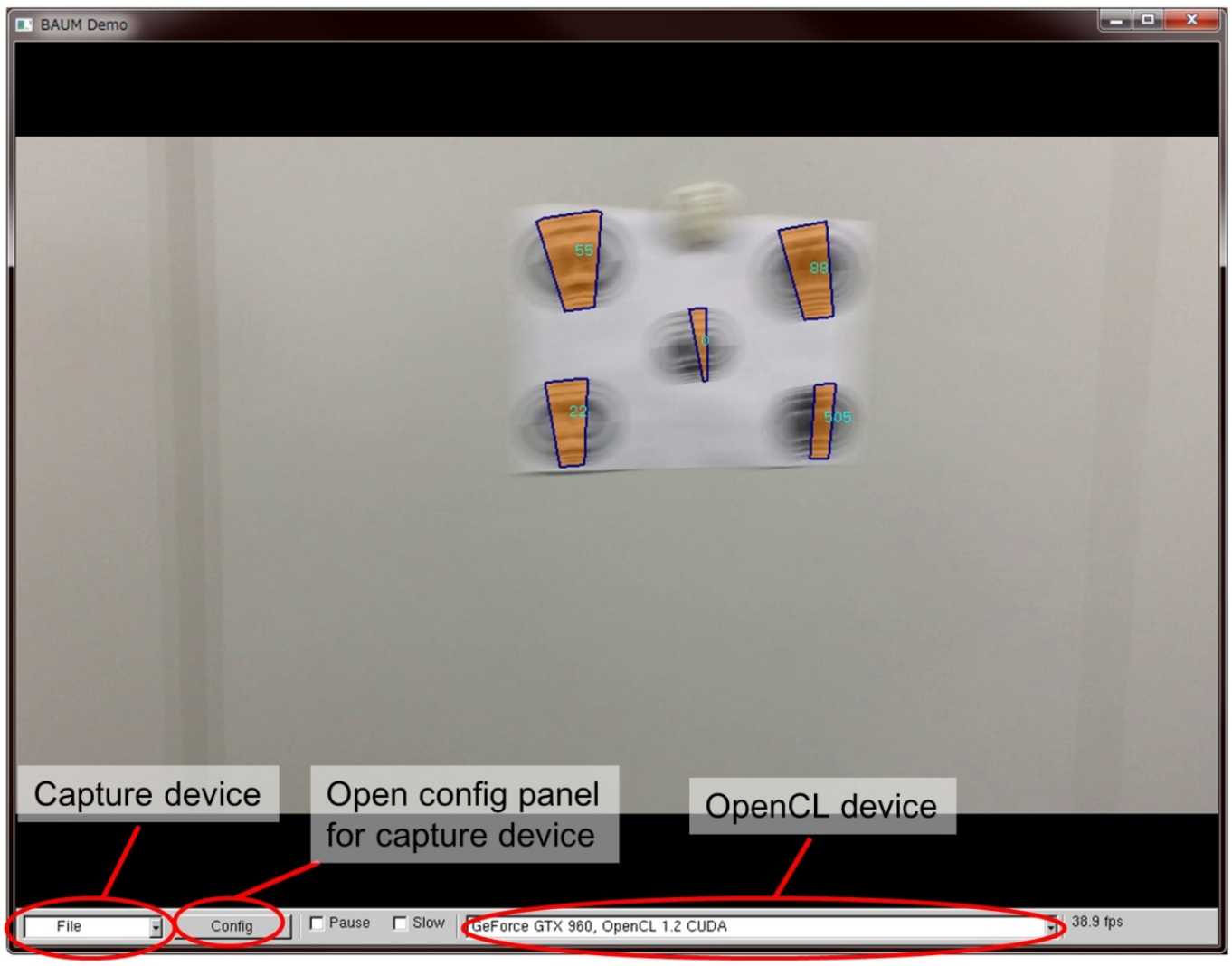

Fig. 3 Screen shot

You can also choose a video file as a video source. In order to decode a video, the program needs FFmpeg DLL file which is not included in the archive, but included in an OpenCV package. Try opening a video file, and it will complain that the DLL is missing. It also displays which version of DLL is required. Please put opencv_ffmpegXXXX_64.dll in bin folder, where XXXX is the version number. With the DLL, it can decode many video formats including MOV video files recorded with an iPhone.

If the video is played/processed too fast, you can slow it down or stop it by checking "Slow" or "Pause" check boxes.

You can also choose a still image file as a video source. In this case, it decodes the image file only once when it is loaded, but the recognition task is repeatedly run. You can check how much CPU and GPU load it requires.

Each barcode can store 10 bits of data, which means that there are 500 different barcodes. 10 barcode images in PNG format are placed in markers folder in the archive. You can make any barcode image in SVG format with baummarker.exe in bin folder, or in PNG format with DrawMarkerPNG java program in java folder.

Compiling on Ubuntu Linux

It should be fairly easy to compile the source code on Ubuntu OS. First, you need to download GLUI Interface Library. Please put glui-2.XX.zip under baum-1.XX directory, and change the version number of GLUI in Makefile accordingly. Then, please run apt-get to install the required packages as shown below.

```
sudo apt-get install g++ ocl-icd-opencl-dev freeglut3-dev libxmu-dev libopencv-dev libgtk2.0-dev
```

You can now run make and all binaries should be built. Before running the program, please check if your GPU is recognized as an OpenCL device by the OS, by running clinfo.

Compiling on Windows with Visual Studio

Below is a summary of the method for compiling the source code with Microsoft Visual Studio.

First thing to consider is which runtime library we should use. If you use the OpenCL.lib included in nVidia GPU Computing Toolkit, it must be LIBCMT.lib (/MT option), since this OpenCL.lib seems to be compiled with that option. Now, we need to build the other libraries with /MT option.

You can use the static pre-built libraries in the OpenCV package, which are compiled with /MT option, though I suggest rebuilding them yourself with appropriate options. You need to uncheck "BUILD_SHARED_LIBS" and check "BUILD_WITH_STATIC_CRT" on CMake GUI.

For freeglut, runtime library options can be changed with CMake GUI, **by checking Advanced check box**. Please replace all /MD by /MT.

For GLUI, please do the following.

1. Open the project file under msvc folder.
2. Change solution configurations to "Release x64".
3. Change the runtime library option from Project → Properties → C/C++ → Code Generation, from /MD to /MT.
4. Add freeglut include directory from Project → Properties → C/C++ → General → Additional Include Directories.
5. Add preprocessor definition FREEGLUT_STATIC from Project → Properties → C/C++ -> Preprocessor.
6. Build the project.
7. If your build fails with Error C2252, it can be eliminated by commenting out the corresponding two lines in glui.h.

GTK+ is not linked if the library is compiled for Windows. It is used to show a file chooser dialog and a message box on Linux.

In order to build Java JNI DLL files, you need to install JDK. Please also set JAVA_HOME environment variable according to the installation directory.

Please edit Makefile.vc under src folder to reflect the installation directories of the above libraries. In order to compile the source code, you need mingw32-make. Cygwin's make does not work. The actual command may depend on the configuration of your computer, but it would be like the following. Please execute the following commands on cmd.exe.

```
"C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\amd64\vcvars64.bat"
cd baum-1.00\src
C:\MinGW\bin\mingw32-make.exe -f Makefile.vc exe dll
```

Programs

baummarker <data> [<radius>] [<X position>] [<Y position>]

This program writes HTML and SVG files containing a BAUM circular barcode image in SVG format to output.html and output.svg. The barcode carries an integer *data*. *data* is a value between 0 and 500. The size and position of the barcode can be specified by *radius*, *posx* and *posy*.

baumcreateplan <device number> <file name>

This program creates an optimal execution plan for the recognition routines. An execution plan is a collection of local workgroup sizes for OpenCL kernels. The optimal execution plan is found by executing all kernels with various local workgroup sizes and measure the execution time of each kernel.

It uses a still image file specified by *file name* as an input. An OpenCL device can be specified by *device number*.

baumtest <device number> <file name>

baumtest is a sample program for explaining how to use BAUM library. It recognizes circular barcodes from an image specified by *file name* using an OpenCL device specified by *device number*, and output the result to output.png. *device number* is an integer ID for conveniently specifying an OpenCL device available for the computer. Number 0 should specify the first device, 1 specifies the second device, and so on. If less than two arguments are given, it prints available OpenCL devices and corresponding numbers, and exits.

glbaum <device number> [<file name>]

glbaum is a sample program for using BAUM library to recognize circular barcodes from video, and display it using OpenGL. You can specify a video file or a still image file as the second argument. If no file is specified, the first camera

is selected as the video source. You need to put ffmpeg DLL file on the same directory as the executable file to decode videos.

glbaum is a demo program of BAUM library.

glbaumui

glbaumui is a demo program of BAUM library.

Java Programs

java **DrawMarkerPNG** <data> [<magninifation>]

DrawMarkerPNG is a Java program that writes a PNG file containing a BAUM circular barcode image to output.png. The barcode carries an integer *data*. The size of the barcode can be specified by *magnification*.

java **BaumTest** <device id> <image file>

BaumTest is a sample Java program for explaining how to use BAUM library from Java. It recognizes circular barcodes from an image specified by *file name* using an OpenCL device specified by *device number*, and output the result to output.png. You need a BAUM JNI DLL file to execute this program. The path of the DLL file can be specified by **org.naokishibata.baum.jnilibpath** property.

java **CamTest** [<input video file>] [<output video file>]

CamTest is a sample Java program for capturing a video from a camera, decoding a video from a file, displaying it and writing the video to a file. If no argument is given, it just displays the captured video. If an input video file is given, it decodes and displays that video. If an output video file is given, it encodes the displayed video to the specified MPEG file. You need OpenCV JNI DLL file to execute this program. The path of the DLL file can be specified by **org.naokishibata.ocvimgio.jnilibpath** property. You also need FFmpeg DLL file included in a OpenCV package to decode or encode a video.

java **BaumCamTest** <device id> [<input video file>] [<output video file>]

BaumCamTest is a sample Java program for capturing a video from a camera, decoding a video from a file, recognize circular barcodes from the video, displaying the video and recognized barcodes and writing the video to a file.

API Reference

BAUM consists of a collection of functions for recognizing and creating BAUM circular barcodes, which are declared in baum.h. All of the functions can be called from C and C++ programs by including "baum.h" header.

baum_t ***baum_init**(int did, int maxiw, int maxih)

A *baum_t* object is constructed for given device number and image size. A *baum_t* object contains an OpenCL command queue, kernels, memory objects, etc.

did is an integer ID for conveniently specifying an OpenCL device available for the computer. Number 0 should specify the first device, 1 specifies the second device, and so on. If the specified OpenCL device is available and no error occurs upon resource allocation, *baum_init* returns a pointer to a created *baum_t* object. If -1 is specified as *did*, *baum_init* prints information for available OpenCL devices and corresponding numbers to stdout, and returns NULL. If the OpenCL device specified by *did* is not available, *baum_init* returns NULL.

maxiw and *maxih* specify the maximum width and height of images for recognition. Subsequent API calls specifying the returned baum_t object can handle image sizes smaller or equal to the specified size.

```
baum_t *baum_init2(cl_device_id device, cl_context context, int maxiw, int maxih)
```

A *baum_t* object is constructed for given OpenCL device ID and context.

device and *context* are OpenCL device ID and context for which a *baum_t* object is constructed. An OpenCL context that is associated with an OpenGL context can be specified as a parameter.

```
void baum_dispose(baum_t *thiz)
```

A *baum_t* object is destructed, and all resources allocated upon construction of given *baum_t* object are released.

thiz is the pointer of a *baum_t* object to destruct. Subsequent API calls specifying a destructed *baum_t* object is undefined.

```
char *baum_getDeviceName(baum_t *thiz)
```

A string for identifying the OpenCL device associated with the *baum_t* object specified by *thiz* is returned. The returned string is concatenation of the vendor name string, ", " and version string of the device. The returned string is newly allocated with malloc and it has to be freed.

```
int baum_enqueueTask(baum_t *thiz, void *resultPtr, size_t bufSize, const void *imgPtr, const int iw, const int ih, const int ws)
```

A series of commands for executing recognition task with given image is enqueued to the OpenCL command queue associated to given *baum_t* object.

iw and *ih* are pixel width and height of the given image. *ws* is the size of image row in bytes. *imgPtr* is the pointer of image data. The data format must be BGR, where each pixel is represented by 3 unsigned bytes.

resultPtr and *bufSize* are the pointer and size of a buffer for receiving a result of recognition task. The format of the received data is shown below. *K* can be larger than *bufSize*/32. In this case, the first *bufSize* of data are store to the buffer specified by *resultPtr*. Each row indicates that a one-dimensional barcode is detected at 2D coordinates (x_0, y_0)-(x_1, y_1).

Table 1 Data structure in the received data

Index (bytes)	Data format					
0	*K* = (1 + Number of decoded barcodes) int, 4 bytes	Reserved 28 bytes				
32	x_0 for barcode 1 float, 4 bytes	y_0 for barcode 1 float, 4 bytes	x_1 for barcode 1 float, 4 bytes	y_1 for barcode 1 float, 4 bytes	Data for barcode 1 int, 4 bytes	Reserved 12 bytes
64	x_0 for barcode 2 float, 4 bytes	y_0 for barcode 2 float, 4 bytes	x_1 for barcode 2 float, 4 bytes	y_1 for barcode 2 float, 4 bytes	Data for barcode 2 int, 4 bytes	Reserved 12 bytes
. . .						
32 * (*K*-1)	x_0 for barcode (*K*-1) float, 4 bytes	y_0 for barcode (*K*-1) float, 4 bytes	x_1 for barcode (*K*-1) float, 4 bytes	y_1 for barcode (*K*-1) float, 4 bytes	Data for barcode (*K*-1) int, 4 bytes	Reserved 12 bytes

baum_enqueueTask just enqueues a task, and the enqueued task is executed in the background by the OpenCL device after returning from *baum_enqueueTask*. The input image data specified by *imgPtr* are transferred to the device before returning from *baum_enqueueTask*, and users can modify the image data afterwards. The buffer specified by *resultPtr* is not filled until the queued task finishes.

baum_enqueueTask returns 0 upon successful completion.

```
int baum_poll(baum_t *thiz, int waitFlag)
```

baum_poll checks enqueued tasks, and returns 1 if there is a finished task. If *waitFlag* is not zero, *baum_poll* blocks until a task is finished. If *waitFlag* is zero and there is no finished tasks, *baum_poll* immediately returns 0. Enqueued tasks are processed in order.

int **baum_queueLen**(baum_t *thiz)

baum_queueLen returns the number of tasks in the queue, including unfinished tasks.

void **baum_createPlan**(baum_t *thiz, const char *path, void *resultPtr, size_t bufSize, const void *imgPtr, const int iw, const int ih, const int ws)

baum_createPlan finds optimal local workgroup sizes for kernels, and save the sizes to a file specified by *path*.

int **baum_loadPlan**(baum_t *thiz, const char *path)

baum_loadPlan loads local workgroup sizes for kernels from a file specified by *path*. Subsequent tasks are executed with the loaded workgroup sizes. *baum_loadPlan* returns 0 upon successful completion.

void ***baum_malloc**(baum_t *thiz, size_t size)

baum_malloc allocates *size* bytes of pinned memory, and returns a pointer to the allocated memory. Pinned memory is a special memory region that can be quickly accessed from GPU. Specifying memory regions allocated by *baum_malloc* as *resultPtr* and *imgPtr* could improve performance and reduce CPU usage when calling *baum_enqueueTask*. (More specifically, data transfer between a CPU and a GPU is a blocking operation if non-pinned memory is specified as a data buffer.) Pinned memory is a scarce resource and allocating too much pinned memory could degrade the overall system performance.

void **baum_free**(baum_t *thiz, void *ptr)

baum_free frees the pinned memory region allocated by *baum_malloc*.

void **baum_fprintMarkerSVG**(FILE *fp, int data, double radius, double posx, double posy)

baum_fprintMarkerSVG prints a circular barcode recognizable by this software to a file specified by *fp*, in SVG format. *data* is a value between 0 and 500. The size and position of the barcode can be specified by *radius*, *posx* and *posy*.

License

BAUM is in public domain. You can use and modify this code for any purpose without any obligation.

The archive contains the following software. Please refer to the distribution license of each software.

- **OpenCV** library is statically linked to the windows binary files under bin and java folder.
- **freeglut** library is statically linked to the windows binary files under bin folder.
- **GLUI User Interface Library** is statically linked to the windows binary files under bin folder.
- **glext.h** is in include directory under src folder.

Contact

Naoki Shibata at Nara Institute of Science and Technology
e-mail : ***n-sibata@is.naist.jp***