PopSift

AliceVision

Oct 05, 2020

INSTALL

PopSift is an open-source implementation of the SIFT algorithm in CUDA [GCH18]. PopSift tries to stick as closely as possible to David Lowe's famous paper [Low04], while extracting features from an image in real-time at least on an NVidia GTX 980 Ti GPU.

CHAPTER

ONE

REQUIREMENTS

1.1 Hardware

PopSift is a GPU implementation that requires an NVIDIA GPU card with a CUDA compute capability >= 3.0 (including, e.g. the GT 650M). The code is originally developed with the compute capability 5.2 card GTX 980 Ti in mind.

You can check your NVIDIA GPU card CC support here or on the NVIDIA dev page. If you do not have a NVIDIA card you will still able to compile and use the CPU version.

Here are the minimum hardware requirements for PopSift:

Minimum requirements	
Operating systems	Windows x64, Linux, macOS
CPU	Recent Intel or AMD cpus
RAM Memory	8 GB
Hard Drive	No particular requirements
GPU	NVIDIA CUDA-enabled GPU (compute capability >= 3.5)

1.2 Software

The core library depends only on Cuda >= 7.0

The library includes a few sample applications that show how to use the library. They require

- Boost >= 1.55 (required components atomic, chrono, date-time, system, thread)
- [optionally] DevIL (libdevil-dev) can be used to load a broader range of image formats, otherwise only pgm is supported.

CHAPTER

TWO

VCPKG

vcpkg is a cross-platform (Windows, Linux and MacOS), open-source package manager created by Microsoft.

We are planning to release a port of the library so that it can be easily built using the package manager on all supported platforms. Stay tuned!

CHAPTER

THREE

BUILDING THE LIBRARY

3.1 Building tools

Required tools:

- CMake >= 3.14 to build the code
- Git
- C/C++ compiler supporting the C++11 standard (gcc >= 4.6 or visual studio or clang)
- CUDA >= 7.0

3.2 Dependencies

3.2.1 vcpkg

vcpkg can be used to install all the dependencies on all the supported platforms. This is particularly useful on Windows. To install the dependencies:

```
vcpkg install cuda devil boost-system boost-program-options boost-thread boost-
↪filesystem
```

You can add the flag `--triplet` to specify the architecture and the version you want to build. For example:

- `--triplet x64-windows` will build the dynamic version for Windows 64 bit
- `--triplet x64-windows-static` will build the static version for Windows 64 bit
- `--triplet x64-linux-dynamic` will build the dynamic version for Linux 64 bit

and so on. More information can be found here

3.2.2 Linux

On Linux you can install from the package manager:

For Ubuntu/Debian package system:

```
sudo apt-get install g++ git-all libboost-all-dev libdevil-dev
```

For CentOS package system:

```
sudo yum install gcc-c++ git boost-devel devil
```

3.2.3 MacOS

On MacOs using Homebrew install the following packages:

```
brew install git boost devil
```

3.3 Getting the sources

```
git clone https://github.com/alicevision/PopSift.git
```

3.4 CMake configuration

From PopSift root folder you can run cmake:

```
mkdir build && cd build
cmake ..
make -j `nproc`
```

On Windows add `-G "Visual Studio 16 2019" -A x64` to generate the Visual Studio solution according to your VS version (see CMake documentation).

If you are using the dependencies built with VCPKG you need to pass `-DCMAKE_TOOLCHAIN_FILE=path/to/vcpkg/scripts/buildsystems/vcpkg.cmake` at cmake step to let it know where to find the dependencies.

3.4.1 CMake options

CMake configuration can be controlled by changing the values of the following variables (here with their default value)

- `BUILD_SHARED_LIBS:BOOL=ON` to enable/disable the building shared libraries
- `PopSift_BUILD_EXAMPLES:BOOL=ON` to enable/disable the building of applications
- `PopSift_BUILD_DOC:BOOL=OFF` to enable/disable building this documentation and the Doxygen one.

For example, if you do not want to build the applications, you have to pass `-DPopSift_BUILD_EXAMPLES:BOOL=OFF` and so on.

CHAPTER FOUR

POPSIFT AS THIRD PARTY

When you install PopSift a file `PopSiftConfig.cmake` is installed in <install_prefix>/lib/cmake/PopSift/ that allows you to import the library in your CMake project. In your `CMakeLists.txt` file you can add the dependency in this way:

```
# Find the package from the PopSiftConfig.cmake
# in <prefix>/lib/cmake/PopSift/. Under the namespace PopSift::
# it exposes the target PopSift that allows you to compile
# and link with the library
find_package(PopSift CONFIG REQUIRED)
...
# suppose you want to try it out in a executable
add_executable(popsiftTest yourfile.cpp)
# add link to the library
target_link_libraries(popsiftTest PUBLIC PopSift::PopSift)
```

Then, in order to build just pass the location of `PopSiftConfig.cmake` from the cmake command line:

```
cmake .. -DPopSift_DIR=<install_prefix>/lib/cmake/PopSift/
```

CHAPTER

FIVE

DOCKER IMAGE

A docker image can be built using the Ubuntu based `Dockerfile`, which is based on nvidia/cuda image (https://hub.docker.com/r/nvidia/cuda/)

5.1 Building the dependency image

We provide a `Dockerfile_deps` containing a cuda image with all the necessary PopSift dependencies installed.

A parameter `CUDA_TAG` can be passed when building the image to select the cuda version. Similarly, `OS_TAG` can be passed to select the Ubuntu version. By default, `CUDA_TAG=10.2` and `OS_TAG=18.04`

For example to create the dependency image based on ubuntu 18.04 with cuda 8.0 for development, use

```
docker build --build-arg CUDA_TAG=8.0 --tag alicevision/popsift-deps:cuda8.0-ubuntu18.
↪04 -f Dockerfile_deps .
```

The complete list of available tags can be found on the nvidia [dockerhub page](https://hub.docker.com/r/nvidia/cuda/)

5.2 Building the PopSift image

Once you built the dependency image, you can build the popsift image in the same manner using `Dockerfile`:

```
docker build --tag alicevision/popsift:cuda8.0-ubuntu18.04 .
```

5.3 Running the PopSift image

In order to run the image nvidia docker is needed: see the installation instruction. Once installed, the docker can be run, e.g., in interactive mode with

```
docker run -it --runtime=nvidia alicevision/popsift:cuda8.0-ubuntu18.04
```

5.4 Official images on DockeHub

Check the docker hub PopSift repository for the available images.

CHAPTER

SIX

LIBRARY USAGE

6.1 Detection

CHAPTER

SEVEN

API REFERENCES

7.1 Main Classes

class SiftJob

Public Functions

SiftJob (int *w*, int *h*, **const** unsigned char **imageData*)
Constructor for byte images, value range 0..255.

Parameters

- `[in] w`: the width in pixel of the image
- `[in] h`: the height in pixel of the image
- `[in] imageData`: the image buffer

SiftJob (int *w*, int *h*, **const** float **imageData*)
Constructor for float images, value range [0..1[.

Parameters

- `[in] w`: the width in pixel of the image
- `[in] h`: the height in pixel of the image
- `[in] imageData`: the image buffer

~SiftJob ()
Destructor releases all the resources.

popsift::FeaturesHost ***get** ()

See *getHost()*

popsift::FeaturesHost ***getHost** ()

Return

void **setFeatures** (popsift::FeaturesBase **f*)
fulfill the promise

class PopSift

Public Types

enum ImageMode
Image modes.

Values:

enumerator ByteImages
byte image, value range 0..255

enumerator FloatImages
float images, value range [0..1[

enum AllocTest
Results for the allocation test.

Values:

enumerator Ok
the image dimensions are supported by this device's CUDA texture engine.

enumerator ImageExceedsLinearTextureLimit
the input image size exceeds the dimensions of the CUDA Texture used for loading.

enumerator ImageExceedsLayeredSurfaceLimit
the scaled input image exceeds the dimensions of the CUDA Surface used for the image pyramid.

Public Functions

PopSift (*ImageMode imode* = *ByteImages*)
We support more than 1 streams, but we support only one sigma and one level parameters.

PopSift (**const** popsift::*Config* *&config*, popsift::*Config*::*ProcessingMode* *mode* = popsift::*Config*::*ExtractingMode*, *ImageMode* *imode* = *ByteImages*)

Parameters

- `config`:
- `mode`:
- `imode`:

~PopSift ()
Release all the resources.

bool **configure** (**const** popsift::*Config* *&config*, bool *force* = false)
Provide the configuration if you used the *PopSift* default constructor.

void **uninit** ()
Release the resources.

AllocTest **testTextureFit** (int *width*, int *height*)
Check whether the current CUDA device can support the image resolution (width,height) with the current configuration based on the card's texture engine. The function does not check if there is sufficient available memory.

The first part of the test depends on the parameters width and height. It checks whether the image size is supported by CUDA 2D linear textures on this card. This is used to load the image into the first level of the first octave. For the second part of the tst, two value of the configuration are important: "downsampling", because it determines the required texture size after loading. The CUDA 2D layered texture must support the scaled width and height. “levels”, because it determines the number of levels in each octave. The CUDA 2D layered texture must support enough depth for each level.

Return AllocTest::Ok if the image dimensions are supported by this device’s CUDA texture engine, AllocTest::ImageExceedsLinearTextureLimit if the input image size exceeds the dimensions of the CUDA Texture used for loading. The input image must be scaled. AllocTest::ImageExceedsLayeredSurfaceLimit if the scaled input image exceeds the dimensions of the CUDA Surface used for the image pyramid. The scaling factor must be changes to fit in.

Remark * If you want to call *configure()* before extracting features, you should call *configure()* before textTextureFit().

Remark * The current CUDA device is determined by a call to cudaGetDevice(), card properties are only read once.

See *AllocTest*

Parameters

- `[in] width`: The width of the input image
- `[in] height`: The height of the input image

std::string **testTextureFitErrorString** (*AllocTest* *err*, int *w*, int *h*)
Create a warning string for an AllocTest error code.

SiftJob ***enqueue** (int *w*, int *h*, **const** unsigned char **imageData*)
Enqueue a byte image, value range [0,255].

Return the associated job

See *SiftJob*

Parameters

- `[in] w`: the width of the image.
- `[in] h`: the height of the image.
- `[in] imageData`: the image buffer.

SiftJob ***enqueue** (int *w*, int *h*, **const** float **imageData*)
Enqueue a float image, value range [0,1].

Return the associated job

See *SiftJob*

Parameters

- `[in] w`: the width of the image.
- `[in] h`: the height of the image.
- `[in] imageData`: the image buffer.

void **uninit** (int)

bool **init** (int, int *w*, int *h*)

popsift::FeaturesBase ***execute** (int, **const** unsigned char **imageData*)

struct popsift::**Config**
Struct containing the parameters that control the extraction algorithm.

Public Types

enum GaussMode

The way the gaussian mode is compute.

Each setting allows to mimic and reproduce the behaviour of other Sift implementations.

Values:

enumerator VLFeat_Compute

enumerator VLFeat_Relative

enumerator VLFeat_Relative_All

enumerator OpenCV_Compute

enumerator Fixed9

enumerator Fixed15

enum SiftMode

General setting to reproduce the results of other Sift implementations.

Values:

enumerator PopSift

Popsift implementation.

enumerator OpenCV

OpenCV implementation.

enumerator VLFeat

VLFeat implementation.

enumerator Default

Default implementation is *PopSift*.

enum LogMode

The logging mode.

Values:

enumerator None

enumerator All

enum ScalingMode

The scaling mode.

Values:

enumerator ScaleDirect

enumerator ScaleDefault

Indirect - only working method.

enum DescMode

Modes for descriptor extraction.

Values:

enumerator Loop

scan horizontal, extract valid points

enumerator ILoop

scan horizontal, extract valid points, interpolate with tex engine

enumerator Grid
scan in rotated mode, round pixel address

enumerator IGrid
scan in rotated mode, interpolate with tex engine

enumerator NoTile
variant of IGrid, no duplicate gradient fetching

enum NormMode
Type of norm to use for matching.

Values:

enumerator RootSift
The L1-inspired norm, gives better matching results ("RootSift")

enumerator Classic
The L2-inspired norm, all descriptors on a hypersphere ("classic")

enum GridFilterMode
Filtering strategy.

To reduce time used in descriptor extraction, some extrema can be filtered immediately after finding them. It is possible to keep those with the largest scale (LargestScaleFirst), smallest scale (SmallestScaleFirst), or a random selection. Note that largest and smallest give a stable result, random does not.

Values:

enumerator RandomScale
keep a random selection

enumerator LargestScaleFirst
keep those with the largest scale

enumerator SmallestScaleFirst
keep those with the smallest scale

enum ProcessingMode
Processing mode.

Determines which data is kept in the Job data structure after processing, which one is downloaded to the host, which one is invalidated.

Values:

enumerator ExtractingMode

enumerator MatchingMode

Public Functions

void **setGaussMode**(**const** std::string &*m*)
Set the Gaussian mode from string.

See *GaussMode*

Parameters

- `[in] m`: The string version of the GaussMode

void **setGaussMode**(*GaussMode m*)
Set the Gaussian mode.

Parameters

- `[in] m`: The Gaussian mode to use.

void **`setMode`**(*SiftMode m*)
Set the Sift mode.

See *SiftMode*

Parameters

- `[in] m`: The Sift mode

void **`setLogMode`**(*LogMode mode = All*)
Set the log mode.

See *LogMode*

Parameters

- `mode`: The log mode.

void **`setVerbose`**(bool *on* = true)
Enable/desable verbose mode.

Parameters

- `[in] on`: Whether to display additional information .

void **`setDescMode`**(**const** std::string &*byname*)
Set the descriptor mode by string.

See *DescMode*

Parameters

- `[in] byname`: The string containing the descriptor mode.

void **`setDescMode`**(*DescMode mode = Loop*)
Set the descriptor mode.

See *DescMode*

Parameters

- `[in] mode`: The descriptor mode.

float **`getPeakThreshold`**() **const**
computes the actual peak threshold depending on the threshold parameter and the non-augmented number of levels

bool **`ifPrintGaussTables`**() **const**
print Gauss spans and tables?

GaussMode **`getGaussMode`**() **const**
What Gauss filter scan is desired?

SiftMode **`getSiftMode`**() **const**
Get the SIFT mode for more detailed sub-modes.

Return The SiftMode

See *SiftMode*

LogMode **getLogMode**() **const**
find out if we should print logging info or not

void **setNormMode**(*NormMode m*)
Functions related to descriptor normalization: L2-like or RootSift

DEPRECATED(void *setUseRootSift*) bool on
Set the normalization mode.

See *NormMode*

Parameters

- `[in] on`: Use RootSift (`true`) or the L2-norm (`false`).

int **getNormalizationMultiplier**() **const**
Functions related to descriptor normalization: multiply with a power of 2.

float **getUpscaleFactor**() **const**
The input image is stretched by 2^upscale_factor before processing. The factor 1 is default.

bool **getCanFilterExtrema**() **const**
Have we enabled filtering? This is a compile time decision. The reason is that we use Thrust, which increases compile considerably and can be deactivated at the CMake level when you work on something else.

int **getFilterMaxExtrema**() **const**
Set the approximate number of extrema whose orientation and descriptor should be computed. Default is -1, which sets the hard limit defined by "number of octaves * getMaxExtrema()".

int **getFilterGridSize**() **const**
Get the grid size for filtering.

To avoid that grid filtering happens only in a tiny piece of an image, the image is split into *getFilterGridSize()* X *getFilterGridSize()* tiles and we allow *getFilterMaxExtrema()* / *getFilterGridSize()* extrema in each tile.

GridFilterMode **getFilterSorting**() **const**
Get the filtering mode.

Return the filtering mode.

See *GridFilterMode*

ScalingMode **getScalingMode**() **const**
Get the scaling mode.

Return the descriptor extraction mode.

See *ScalingMode*

DescMode **getDescMode**() **const**
Get the descriptor extraction mode.

Return the descriptor extraction mode

See *DescMode*

Public Members

int **octaves**
The number of octaves is chosen freely. If not specified, it is: log_2(min(x,y)) - 3 - start_sampling

int **levels**
The number of levels per octave. This is actually the number of inner DoG levels where we can search for feature points. The number of …

This is the non-augmented number of levels, meaning the this is not the number of gauss-filtered picture layers (which is levels+3), but the number of DoG layers in which we can search for extrema.

float **_edge_limit**
default edge_limit 16.0f from Celebrandil default edge_limit 10.0f from Bemap

Public Static Functions

GaussMode **getGaussModeDefault**()
Call this from the constructor.

const char ***getGaussModeUsage**()
Get a message with the strings to use for setting the values of `GaussMode`.

Return A message with the list of strings

7.2 Functions

7.3 Utility Classes

CHAPTER

EIGHT

ABOUT

8.1 License

PopSift is licensed under MPLv2 license.

More info about the license and what you can do with the code can be found at tldrlegal website

SIFT was patented in the United States from 1999-03-08 to 2020-03-28. See the patent link for more information. PopSift license only concerns the PopSift source code and does not release users of this code from any requirements that may arise from patents.

8.2 Contact us

You can contact us on the public mailing list at alicevision@googlegroups.com

You can also contact us privately at alicevision-team@googlegroups.com

8.3 Cite us

If you want to cite this work in your publication, please use the following

```
@inproceedings{Griwodz2018Popsift,
    author = {Griwodz, Carsten and Calvet, Lilian and Halvorsen, P{\aa}l},
    title = {Popsift: A Faithful SIFT Implementation for Real-time Applications},
    booktitle = {Proceedings of the 9th {ACM} Multimedia Systems Conference},
    series = {MMSys '18},
    year = {2018},
    isbn = {978-1-4503-5192-8},
    location = {Amsterdam, Netherlands},
    pages = {415--420},
    numpages = {6},
    doi = {10.1145/3204949.3208136},
    acmid = {3208136},
    publisher = {ACM},
    address = {New York, NY, USA},
}
```

8.4 Acknowledgements

This has been developed in the context of the European project POPART founded by European Union's Horizon 2020 research and innovation programme under grant agreement No 644874.

CHAPTER

NINE

BIBLIOGRAPHY

BIBLIOGRAPHY

[GCH18] Carsten Griwodz, Lilian Calvet, and Pål Halvorsen. Popsift: a faithful sift implementation for real-time applications. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, 415–420. New York, NY, USA, 2018. ACM. doi:10.1145/3204949.3208136.

[Low04] DG Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, pages 1–29, 2004. doi:10.1023/B:VISI.0000029664.99615.94.

INDEX

P

S