High-performance image processing routines for video and film processing

Hannes Fassold
2018-03-28
Our research group

- GPU-accelerated algorithms / applications @ CCM
  - Connected Computing research group, DIGITAL – Institute for Information and Communication Technologies, JOANNEUM RESEARCH, Graz, Austria
  - Content-based film and video quality analysis
    - http://vidicert.com
  - Digital film restoration
    - http://www.hs-art.com
  - Real-time video analysis & brand monitoring
    - https://recap-project.com
    - http://www.branddetector.at
  - Surveillance / traffic video analysis
  - GPU activities since 2007

http://vidicert.com
http://www.hs-art.com
https://recap-project.com
http://www.branddetector.at
Presentation overview

- High-performance image processing routines
  - Motivation & Design principles
  - Simple example kernel – code walkthrough
  - Morphological / generalized convolution operators

- Applications
  - Film and video restoration
  - 360° video tools
    - Automatic quality assessment
    - Automatic camera path

Brainweb dataset [Cocosco1997]), Denoising result (9 % Riccian noise)
Motivation / Goals

Motivation

Basic image processing routines (arithmetic operators, convolutions, morphological ops, …) are at the core of important high-level computer vision algorithms

- Feature point tracking, Interest point detection (SIFT), …

Existing libraries are not a good fit for us due to certain deficiencies

- NPP (for Toolkit 7.0): No border handling, performance problems for some important routines
- ArrayFire: Difficult to integrate (has own memory manager), no 16-bit floats, …
- OpenCV: Enjoy building 😊 (Huge framework, lot of dependencies, huge DLL size, no 16-bit floats, …)

Goals for development of our own basic GPU image processing routines

- Broad coverage (different # of channels, different datatypes, …)
- Reasonable development time, easy maintainable code
- Performance!
Design principles

- Design principles of GPU implementation
  - Based on principles mentioned in [Iandola2013]
  - „Register blocking“ (employed also on CPU e.g. for high performance GEMM)
  - Load directly into register via „texture path“
  - Computation of multiple outputs per thread (parameter „grainsize“)
  - Make it easy for compiler to unroll the innermost convolution loop
    (by making e.g. convolution filter radius a template parameter)

- Example
  - Brief code walkthrough on a simple kernel for pixel-wise addition of two one-channel images

Multiple outputs per thread (Image courtesy of [Iandola2013]).
Code walkthrough

Input / output images

- All input images are bound to a texture object
  - Provides automatic caching (& partial coalescing) via texture cache
  - Accessing pixels outside of image borders is allowed (via several border modes)
    - Makes code for convolution / morphological operators much more compact and readable!

- All output images are simple pitch-linear memory buffers

- Datatype and grain size are template parameters

```cpp
template <int iplDepth, int grainSize>
global
void add2Kernel(Cusu::Dev::TexObjectLin2D a, Cusu::Dev::TexObjectLin2D b, int height, int width,
                Cusu::Dev::ImageBufferC1<typename Jrs::Ipl::TypeInfo<iplDepth>::RepTypeGPU> dst)
{
  // e.g. for 'half', the underlying binary type is 'uint16' and the 'calculation type' is 'float'
typedef typename Jrs::Ipl::TypeInfo<iplDepth>::RepTypeGPU BinaryT;
typedef typename Jrs::Ipl::TypeInfo<iplDepth>::CalcType CalcT;
```
Main part of kernel (load into register tile – process tile – write tile)

```cpp
// 'x0' and 'y0' are the upper-left coordinate of the 'grainSize x grainSize' region which will be handled by current thread
int x0 = grainSize * blockIdx.x % blockDim.x + grainSize * threadIdx.x;
int y0 = grainSize * blockIdx.y % blockDim.y + grainSize * threadIdx.y;
if (y0 < height && x0 < width) {
    // load the 'grainSize x grainSize' tile (for a, b) from texture memory into register arrays
    BinaryT aTile[grainSize][grainSize];
    RegisterLoad::load<BinaryT, grainSize>(aTile, y0, x0);
    BinaryT bTile[grainSize][grainSize];
    RegisterLoad::load<BinaryT, grainSize>(bTile, y0, x0);

    // now add all tiles together in registers, and write result into tile 'dstTile'.
    BinaryT dstTile[grainSize][grainSize];
    for (int dy = 0; dy < grainSize; ++dy) {
        for (int dx = 0; dx < grainSize; ++dx) {
            dstTile[dy][dx] = wrap<iplDepth>(unwrap<iplDepth>(aTile[dy][dx]) + unwrap<iplDepth>(bTile[dy][dx]));
        }
    }

    // write back register tile 'dstTile' to global memory
    RegisterStore::store<BinaryT, grainSize>(dstTile, dst, y0, x0, height, width);
}
```
Morphological operators & generalized convolution

- Binary morphological filters (dilation / erosion)
  - Equivalent to convolution + thresholding
  - So we can reuse our super-optimized box filter 😊

- „Generalized convolution“ operator (GCO)
  - $L_{p,w}(x) = \frac{\sum_{k=1}^{n} w_k \cdot x_k^p}{\sum_{k=1}^{n} w_k \cdot x_k^{p-1}}$
  - = weighted Lehmer mean [Beliakov2016], counter-harmonic mean [Masci2012]
  - Is able to „morph“ smoothly between a (approximate) morphological operator and standard convolution via parameter $p$
  - In „deep learning speak“: A GCO layer is a generalization / unification of max pooling layers and standard convolution layers
  - $P$ can be treated as weight parameter which is optimized during training of the network (see [Masci2012])

Learning a top-hat transform (Image courtesy of [Masci2012])
Film and video restoration

- Automatic digital restoration of film & video
  - Detection and repair of common film and video defects like
    - Dust, dirt, blotches, line / block dropouts
    - Film grain, electronic noise
    - Flicker, Stain, Mold
    - Instability

- Available locally or as cloud-ready service
  - Locally via DIAMANT (Film/Video) restoration suite
    [http://www.hs-art.com](http://www.hs-art.com)
  - Via AVEROS whitelabel service for the cloud
    [https://www.automatic-restoration.com](https://www.automatic-restoration.com)

Restoration result for IR video from a FLIR camera. Denoising algorithm from [Fassold2015].
360° video tools

Video quality analysis

- Hyper360 (EU H2020 research project)
  - Aims to build a complete end-to-end production toolset for enriching 360° (omnidirectional) video with 3D storytelling and personalisation elements
  - [http://www.hyper360.eu](http://www.hyper360.eu)

- Video quality check for 360° video
  - Does content-based quality check for defects occurring in the stitched video
  - Quality checks for noise, blurriness, macroblocking, dropouts, …
Automatic camera path calculation

Goal: Provide a “lean-back” experience (without requiring user interaction) for consuming 360° video

Calculates most pleasing / most interesting camera path based on several cues

- Video saliency / motion cues
- Person / object detection
- Result of quality analysis
- …
Interested in our **technologies** and/or **applications**?

- Contact me
  (hannes.fassold@joanneum.at)
- Or contact Georg Thallinger
  (head of Smart Media Services)
  georg.thallinger@joanneum.at

GPU-accelerated inpainting for LIDAR depth maps & images [Rosner2009]
Depth maps courtesy of Karlsruhe Institute of Technology
References

Acknowledgments

Thanks for Karlsruhe Institute of Technology for providing the LIDAR depth maps.

Thanks to NVIDIA for the support and the provided GPUs.

The research leading to these results has received partial funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 761934, “Hyper360 - Enriching 360 media with 3D storytelling and personalisation elements”.

http://www.hyper360.eu/
Hannes Fassold
hannes.fassold@joanneum.at