Evolution of Image Filters on Graphics Processor Units Using Cartesian Genetic Programming

Simon Harding
Memorial University, St John's, NL, Canada

http://www.tgdaily.com/content/view/37611/140/

Evolving image filters

- Evolving programs to manipulate images
  - Noise removal
  - Arbitrary filters (current and future work).
- The real novelty here is the use of the GPU
  - But, we do make some additions to the technique that become feasible with additional computational resources.

Image filters

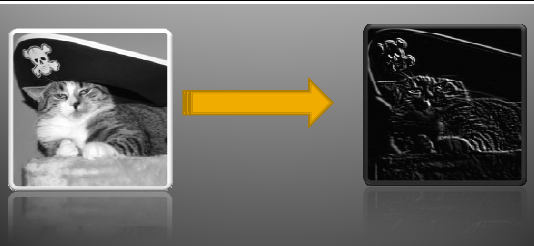
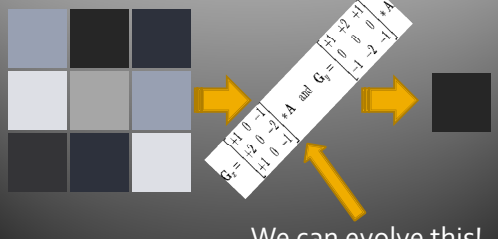


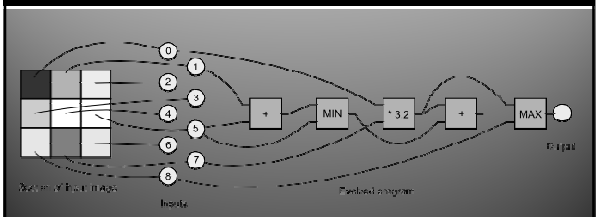
Image filters



Evolving image filters

- We will evolve a program/expression that will take a neighbourhood of pixels, apply a convolution and output a new value for the centre pixel.

GP as an image filter



Cartesian Genetic Programming

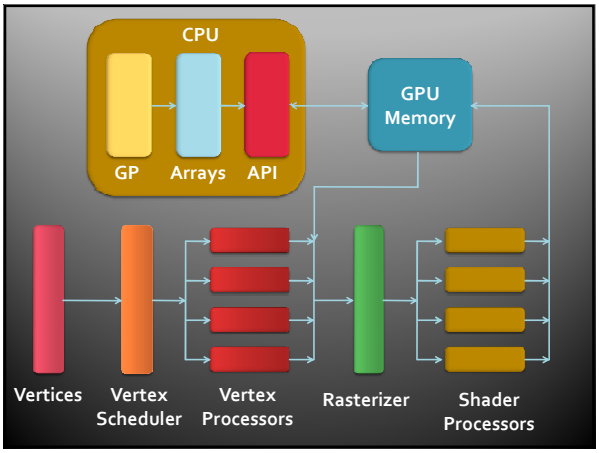
- Form of GP based on acyclic directed graphs
  - Implicit re-use of nodes in the graph
- Fixed length genotype
  - List of integers encoding the nodes in the graph
- Bounded variable length phenotype
  - Not all nodes in the graph are connected
- A form of neutrality present in CGP
  - Shown to be beneficial to the evolutionary process

Function set

ADD	Add the two inputs
SUB	Subtract the second input from the first
MULT	Multiply the two inputs
DIV	Divide the first input by the second
ADD CONST	Adds a constant (the node's parameter) to the first input
MULT CONST	Multiplies the first input by a constant (the node's parameter)
SUB CONST	Subtracts a constant (the node's parameter) to the first input
DIV CONST	Divides the first input by a constant (the node's parameter)
SQRT	Returns the square root of the first input
POW	Raises the first input to the power of the second input
COS	Returns the cosine of the first input
SIN	Returns the sin of the first input
NOP	No operation - returns the first input
CONST	Returns a constant (the node's parameter)
ABS	Returns the absolute value of the first input
MIN	Returns the smaller of the two inputs
MAX	Returns the larger of the two inputs
CEILING	Rounds up the first input
FLOOR	Rounds down the first input
FRACTION	Returns the fractional part of dividing the first input by the second
LOG2	Log (base 2) of the first input
RECIPRICAL	Returns 1/[first input]
RSQRT	Returns 1/sqrt[first input]

The evolutionary algorithm

- Population of size 50
- 5% of genes mutated
- No crossover
- 5 best individuals promoted to next generation
- Graph size of 50 nodes
- Maximum of 50,000 evaluations



MS Accelerator

- We chose to use Microsoft Research's Accelerator
- http://research.microsoft.com/act/

MS Accelerator

ADVANTAGES

- Simple to use
- Highly abstracted
- Lazy evaluator
- CPU mode

DISADVANTAGES

- Windows only
- May be too highly abstracted

MS Accelerator – example code

```
ParallelArrays.InitGPU();

float[,] CPU_Array1 = new float[4096,4096];
float[,] CPU_Array2 = new float[4096, 4096];

//Populate CPU arrays here

FloatParallelArray GPU_Array1 = new DisposableFloatParallelArray(CPU_Array1);
FloatParallelArray GPU_Array2 = new DisposableFloatParallelArray(CPU_Array2);

FloatParallelArray GPU_Array3 = ParallelArrays.Add(GPU_Array2, GPU_Array2);

FloatParallelArray GPU_Array4 = ParallelArrays.Divide(0.1234f, GPU_Array3);

float[,] CPU_Result = new float[4096, 4096];

ParallelArrays.ToArray(GPU_Array4, out CPU_Result);

// Process CPU_Result array here

ParallelArrays.UnInit();
```

Available functions

- GPUs have a full range of mathematical operations
- Some are specific to vector manipulation
- Others are more useful in parallel uses

Available functions

MATHEMATICAL

- Abs, add, ceiling, cos, divide, floor, log2, multiply, multiply add, pow, reciprical, rsqrt, sqrt, subtract

LOGICAL

- ==, <=, >=, !=, <, >

BINARY

- And, or, not,

VECTOR

- Sum, rotate, add / drop dimension, gather, inner product, product, stretch, select, section

Evolving image filters

- Computational problem:
  - We need to apply this evolved program to every pixel in an image.
  - We also need to compute some sort of fitness score.
  - We need to do this for every individual in our population in every generation.

Evolving image filters

- There is actually relatively little research on evolving filters using GP.
- Why? It takes too long on a CPU to test an individual.
- A lot of the research looks into evolving hardware (e.g. FPGA) implementations.

Evolving image filters

- Most research looks at applying filters to a single 256x256 pixel image.
- But how do we know if this evolved program would work on other images, with different characteristics?

Evolving image filters

- Ideally, we need to train on a bunch of pictures.
- Better still, we should be able to validate these filters on another set of images.
  - In the WCCI paper I did not do this, but will later briefly describe it as part of our ongoing work.

Fitness function

- Find the average error of each pixel
- Where the error is the difference between the output of the evolved program and the target image.
- This is also computationally expensive, but can be processed in parallel on the GPU.

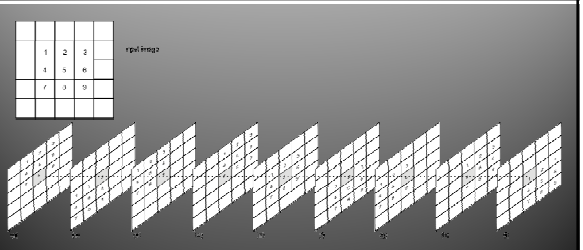
Noise removal

- Salt and pepper
  - 5% of pixels set to either black or white
- Random noise
  - 5% of pixels set to random grey value

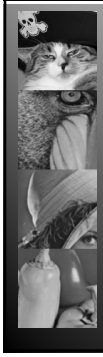
Evolving image filters on the GPU

- We need to map to the SIMD architecture and the MS Accelerator way of working.
- Accelerator is purely element wise.
  - Simple API leads to some minor issues.
- But how do we get the neighbourhoods of pixels?
  - Shifting!
  - Would be much better to do some sort of direct access – but this is not available in this API.

Evolving image filters on the GPU



Multiple Image



- Here we train on 4 different images (to help prevent over fitting) .
- We also tried compared using 4 copies of each image.
- We corrupt the images with noise, and then find a filter that produces an image that is close to the original image.

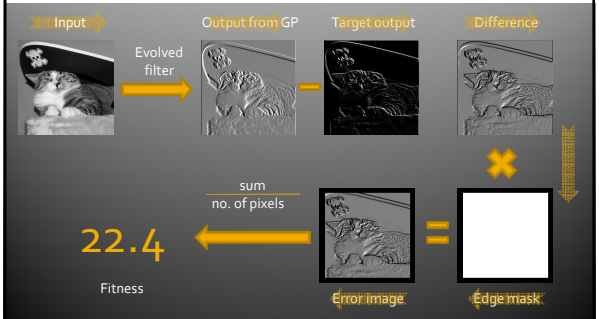
Presenting multiple images at the same time

- We present multiple images simultaneously.
- In effect, we process one large image.
- However, we must be careful of the edges where different images touch.

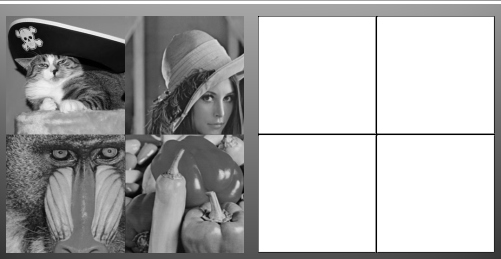
Presenting multiple images at the same time

- We apply a mask to the fitness function
- The mask is another image consisting of pixels of either 0 or 1.
- When we find the difference between two images, we then multiply this by the mask.

Fitness function With Mask



Fitness Function With Mask



Results

Noise Type	No. of test images	Average Fitness	Best Fitness
Random	4	1.3	0.55
Random	16	1.06	0.53
Salt and Pepper	4	0.76	0.39
Salt and Pepper	16	1.04	0.3

Noise Type	Median filter fitness	Unfiltered image fitness
Random	3.54	3.77
Salt and Pepper	3.62	6.30

Results

Noise Type	No. of test images	Avg. evals.	Min evals.
Random	4	42539	14712
Random	16	43296	252
Salt and Pepper	4	42852	8732
Salt and Pepper	16	39795	19403

Removing salt and pepper noise

Original image | Corrupted with 5% Salt and Pepper noise | Evolved filter | Conventional filter

Removing random noise

Original image | Corrupted with 5% Salt and Pepper noise | Evolved filter | Conventional filter

GPU Speed Up

- MS 'broke' the CPU side version of Accelerator in their last update!
- But, you can switch it to use the reference driver (CPU based).
- This has unknown overheads and is unclear what the performance penalty may be.

GPU Speed Up

Processor	Image Size	Million GP Op/s
GPU	512x512	300
GPU	1024x1024	620
CPU	1024x1024	1.82

- nVidia GeForce 7300, Intel 6400, Windows XP

Current and Future Work

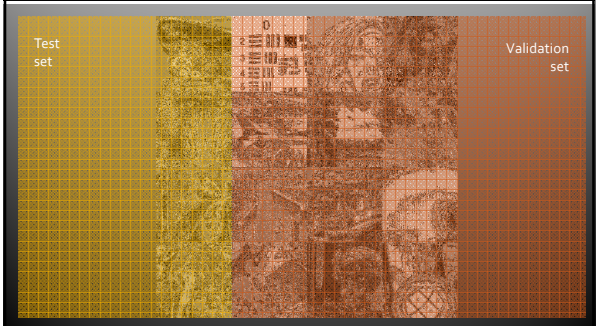
Reverse engineering filters

- We can use the same approach to reverse engineer an existing filter.
- i.e. We take an image, apply a filter in an image manipulation package (Photoshop, GIMP) and then evolve a program that can reproduce the effect.

Reverse engineering filters

- 16 images
- 256 x 256 pixels per image

Reverse engineering filters

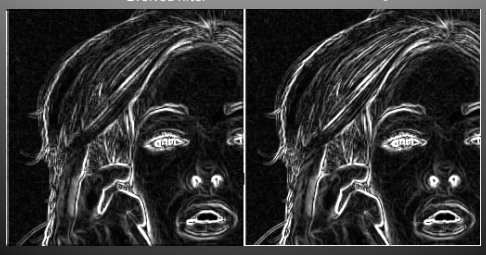


Evolution of a filter

Input | Best evolved individual | Target

Examples of evolved filters

- Sobel

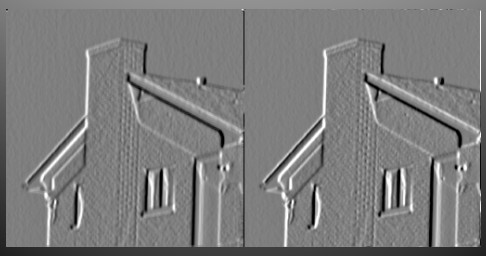


Examples of evolved filters

- Erode

Examples of evolved filters

- Emboss



GPU Speed

- We measured the Millions of Genetic Programming Operations Per Second (Nvidia 8800)

Filter	Peak	Average
dilate	116	62
dilate2	281	133
emboss	254	129
erode	230	79
erode2	307	195
motion	280	177
neon	266	185
sobel	292	194
sobel2	280	166
unsharp	324	139

GPU Speed

- Using the CPU bound reference driver, we found we could achieve an average of 1.2 million GPOps.
- The CPU appears to be 100 times slower.
- The GPU has 128 processors….

Conclusion

- GPUs are effective platforms to use for image processing operations. We can exploit this computational power when researching GP applications involving image sets.
- We can feasibly tackle much harder (more interesting?) problems.
- We can also increase the reliability of our algorithms by greater testing during evolution.

Conclusion

- We have inconsistent results in timing Operations Per Second.
  - The reason is unclear. Perhaps it is due to the MS API being 'efficient'?
- We do however see big increase in performance over running on the CPU reference driver
  - we have already measured true CPU performance on other problems.

Thank you!

- www.gpgpgpu.com

GPU Programming

Thoughts & opinions on the practical aspects

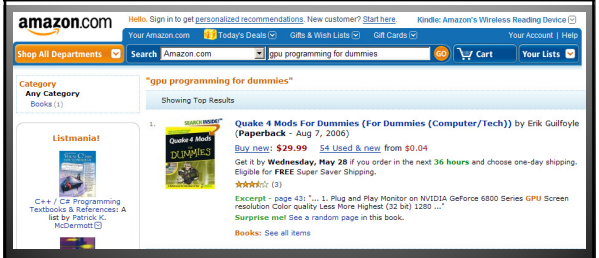
Timeline

- MS Accelerator
  - Early 2007
- RapidMind
  - Mid 2007 …
- CUDA
  - Early 2008
- HLSL
  - Two weeks ago

API Hell

- Numerous APIs exist
  - Each with their own benefits and drawbacks.
- It is still unclear which is the best approach

Documentation



High Level Programming

- C or C++
  - RapidMind
  - CUDA
  - Cg, Etc
- Java
  - Via the JNI to the C ones?
- .net
  - Accelerator
  - XNA/HLSL
  - RapidMind fails to link due to CRT issues

Debugging

- GPGPU development tools are immature.
- Debugger support is weak
  - Unable to inspect memory on GPU during execution
  - Unable to step through shader programs
  - (Conditional)Breakpoints

Future

- Will my favourite API still be useful next month? Will I be forced to change platform when a new GPU technology comes out (e.g. Multi-GPU)?
- Several APIs have already died
  - Sh, PyGPU
- Several are 'underground'
  - MS Accelerator, Peakstream