Lecture 13

Image filters

OpenCL

GPU computing with GLSL

OpenGL Compute shaders

Lecture questions

1) What kind of devices will OpenCL run on?

2) What does an OpenCL work group correspond to in CUDA?

3) What geometry is typically used for shader-based GPU computing?

4) Are scatter or gather operations preferrable? Why?

Lab 5

- **Image filtering with shared memory**
- **Low-pass filters**
- **Median filter**

Intended as continuation of previous image filtering lab.

Lab 6

- OpenCL
- Reduction
- Sorting using bitonic sort

Image filters (lab 3 and 5)

Linear filters: Convolution

Box filter

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

/25

Gaussian approximation

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

/256

And others

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Sobel (gradient)

0	-1	0
-1	4	-1
0	-1	0

Laplace

Separable filters

$$\begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix}/5 \oplus \begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix}/5 = \begin{bmatrix}1 & 1 & 1 & 1 & 1\\1 & 1 & 1 & 1 & 1\\1 & 1 & 1 & 1 & 1\\1 & 1 & 1 & 1 & 1\\1 & 1 & 1 & 1 & 1\end{bmatrix}/25$$

$$\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix}/16 \oplus \begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}/16 = \begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & 36 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}/256$$

Repeated box filters converge to Gaussian!

$$\begin{pmatrix}1\\4\\6\\4\\1\end{pmatrix}/16 = \begin{pmatrix}1\\2\\1\end{pmatrix}/4 \oplus \begin{pmatrix}1\\2\\1\end{pmatrix}/4 = \begin{pmatrix}1\\1\end{pmatrix}/2 \oplus \begin{pmatrix}1\\1\end{pmatrix}/2 \oplus \begin{pmatrix}1\\1\end{pmatrix}/2 \oplus \begin{pmatrix}1\\1\end{pmatrix}/2$$

Central limit theorem

Compare to dice!

Non-linear filters

Median filter

Outputs median of neighborhood.

Requires some method to find the median.

Important application noise suppression.
Preserves edges!

Separable only as approximation.

Median filters

Example

1	2	2
1	9	3
1	7	10

1 1 1 2 2 3 7 9 10

Average: 4
Median: 2

Not separable

1	2	2	2
1	9	3	3
1	7	10	7
1	7	3	

Rows first or columns first both end up 3 for this case

Works as approximation

How to filter edges

The filter kernel reaches outside the image!

My answer: clamp!

Solved for you in the Lab 5 code.

Why? Avoid branching!

```
if (x < imagesizex && y < imagesizey)
{
// Filter kernel (simple box filter)
 sumx=0;sumy=0;sumz=0;
 for(dy=-kernelsizey;dy<=kernelsizey;dy++)
  for(dx=-kernelsizex;dx<=kernelsizex;dx++)
  {
   // Use max and min to avoid branching!
   int yy = min(max(y+dy, 0), imagesizey-1);
   int xx = min(max(x+dx, 0), imagesizex-1);

   sumx += image[((yy)*imagesizex+(xx))*3+0];
   sumy += image[((yy)*imagesizex+(xx))*3+1];
   sumz += image[((yy)*imagesizex+(xx))*3+2];
  }
 out[(y*imagesizex+x)*3+0] = sumx/divby;
 out[(y*imagesizex+x)*3+1] = sumy/divby;
 out[(y*imagesizex+x)*3+2] = sumz/divby;
}
```

Remember texture memory?

Clamp and repeat

This is perfect and automatic!

You are used to this

ERROR	ERROR	ERROR	ERROR
ERROR	1	2	ERROR
ERROR	3	4	ERROR
ERROR	ERROR	ERROR	ERROR

Now you can get this

4	3	4	3
2	1	2	1
4	3	4	3
2	1	2	1

or this

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

In the lab

1. Shared memory
Use shared memory to reduce global memory access.
Major part of the lab!

2. Separable filters
Easy if step 1 is done right.

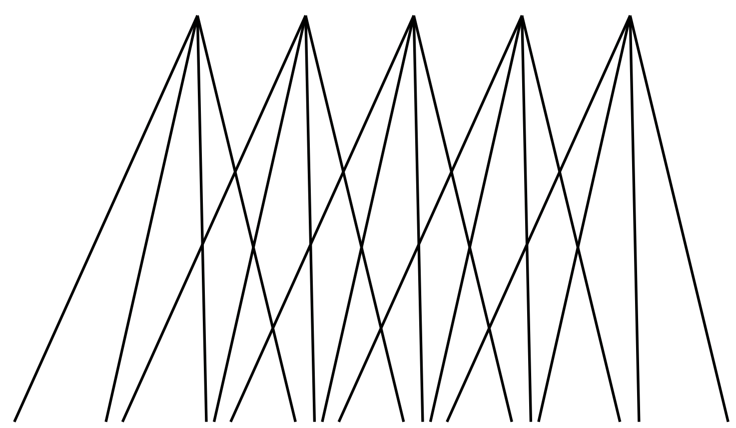
3. Weighted kernels
One size is enough.

4. Median filter
Variable size, modest demands.

Trivial filter

Just loop over kernel

Inefficient! Multiple reads from global memory!

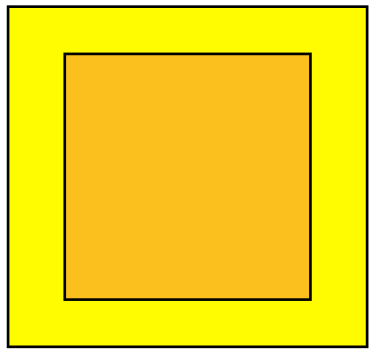


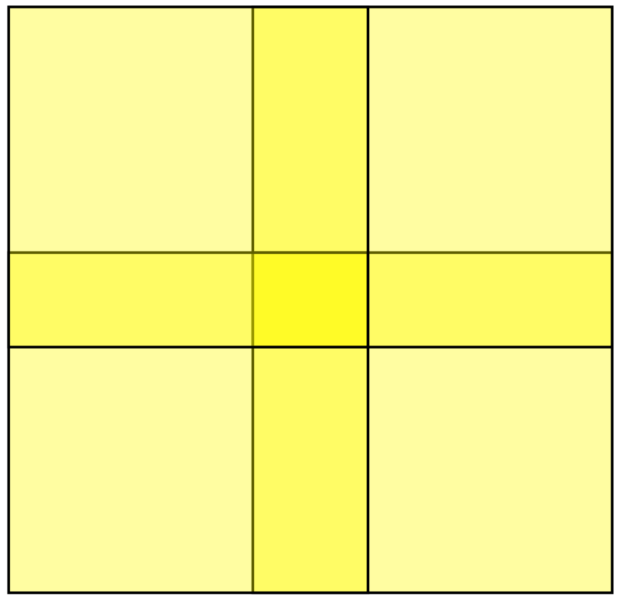
Better filter

Use shared memory!

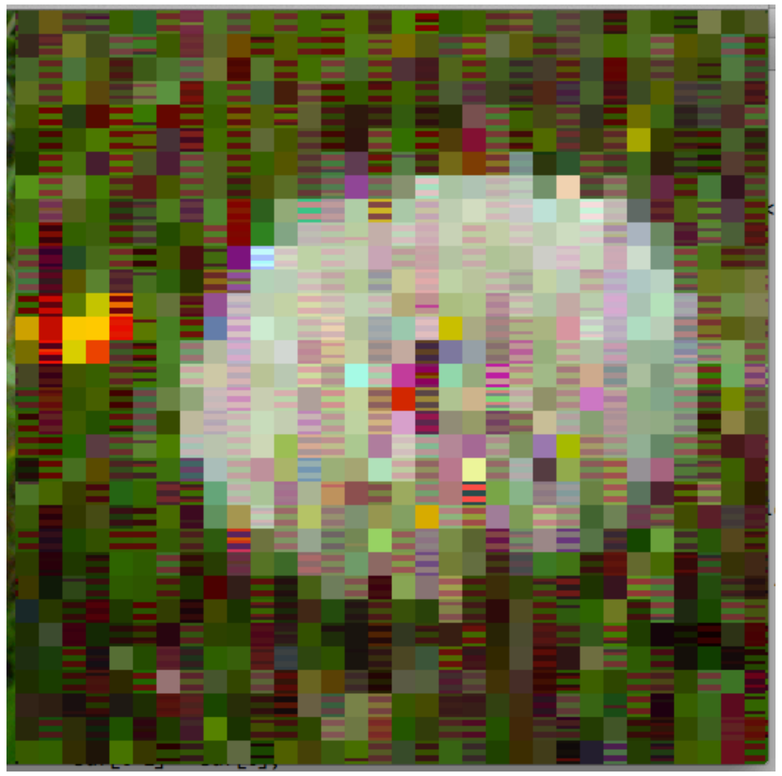
How can you minimize global reads,
or at least significantly reduce them?

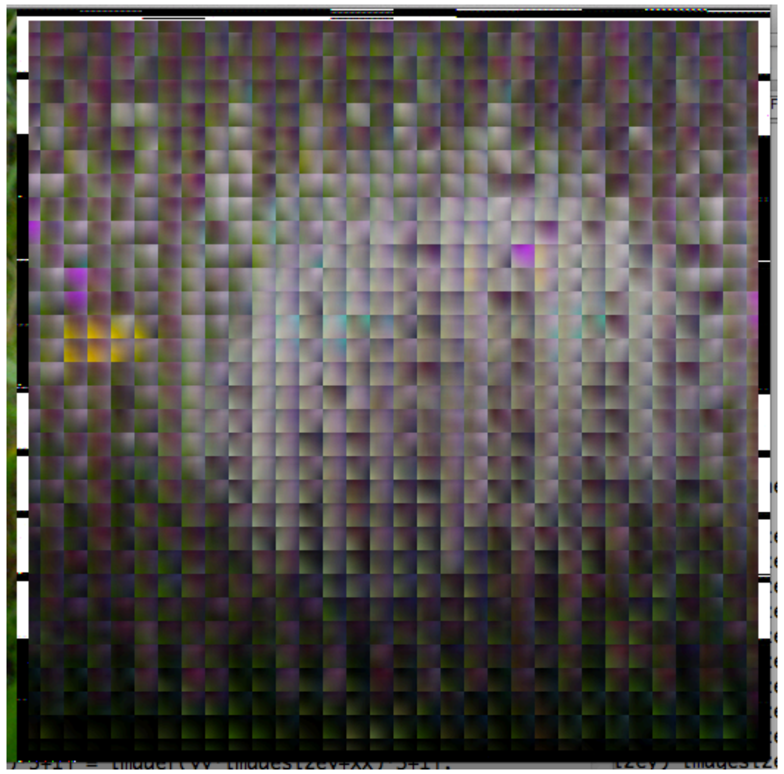
Note the edges of the patch
computed by each block!





Bonus: Unintentional fun!





Coding filters in CUDA is like a box of chocolate...