A Fast, Small-Radius GPU Median Filter

- Jongtae Park -

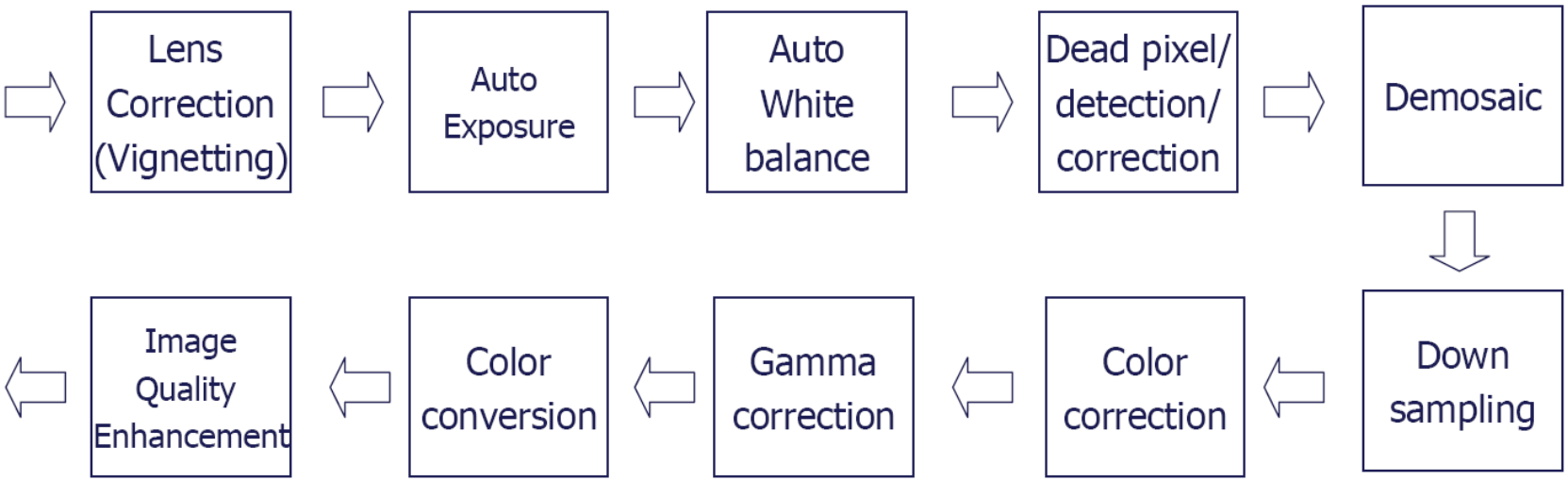
Embedded Systems Language & compiler lab

Organization

- Introduction of our work
- Demo "ISP Tool"
- A fast, Small-Radius GPU Median Filter
- Benchmark result

Our work

- The goal
  - To realize ISP on GPGPU by using stream pr ogramming model with novel scheduling tech niques such as software pipelining



What is the ISP

- ISP(Image Signal Processing)
  - Wikipedia says

    "In electrical engineering and computer scie nce, image processing is any form of signal processing for which the input is an image, such as a photograph or video frame"

What is the ISP(Cont')

- Example of ISP
  - Gamma Correction

What is the ISP(Cont')

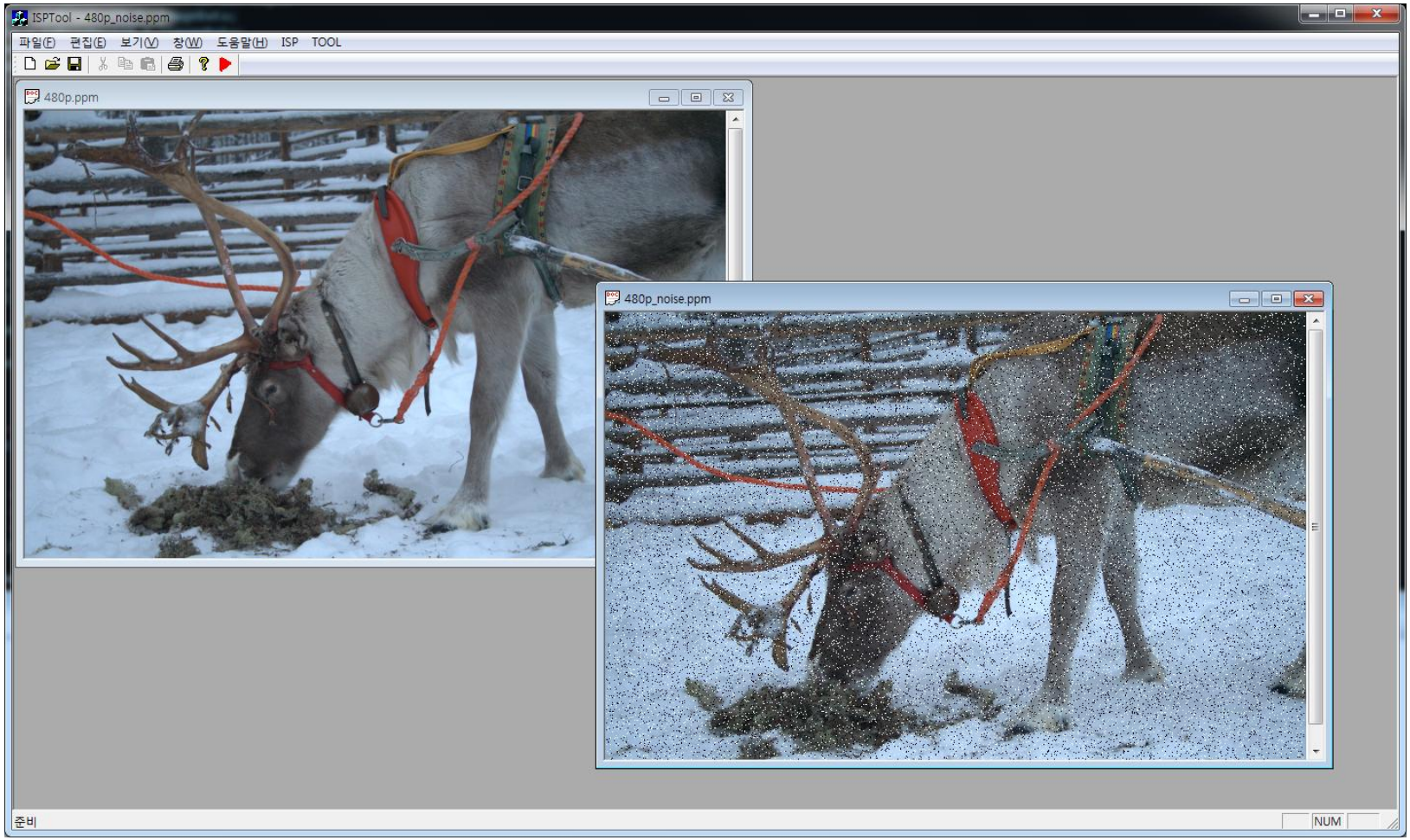
- Example of ISP
  - Motion deblurring

Short exposure time

Long exposure time

Motion Deblurred image
- PSNR 35.6

Demo of ISP Tool



A fast, Small-Radius GPU Median Filter

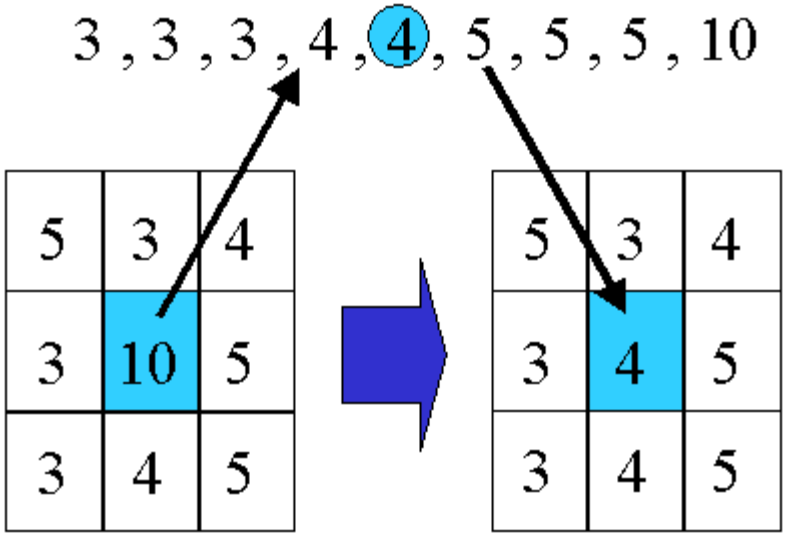
- Prof. Morgan McGuire, Williams College
- the chapter of book "ShaderX"
  - Describes
    - a very fast median filter for today's GPUs
    - how to port it to future GPUs and other data-parallel processors like DSPs and CPUs with vector instructions (e.g., MMX, SIMD)

Median Filters

- The median filter is a nonlinear digital filtering technique, often used to remove noise

- Main ideas is to run through the signal e ntry by entry, replacing each entry with the median of neighboring entries

Median Filters(Cont')




```
33  void InsertSort(uchar* d)
34  {
35      int i, j;
36      uchar temp;
37
38      for( i = 1 ; i < 9 ; i++ )
39      {
40          temp = d[i];
41          j = i - 1;
42
43          while( j > -1 && d[j] > temp )
44          {
45              d[j+1] = d[j];
46              j--;
47          }
48
49          d[j+1] = temp;
50      }
51  }
```

Proposed technique of Prof. Mcguire

- No use of "if" statement

```
#define MAX( a, b )      (( (a) > (b) ) ? (a) : (b))
#define MIN( a, b )      (( (a) > (b) ) ? (b) : (a))
#define s2(a, b)                 temp = a; a = MIN(a, b); b = MAX(temp, b);
#define mn3(a, b, c)             s2(a, b); s2(a, c);
#define mx3(a, b, c)             s2(b, c); s2(a, c);
#define mnmx3(a, b, c)           mx3(a, b, c); s2(a, b);
#define mnmx4(a, b, c, d)        s2(a, b); s2(c, d); s2(a, c); s2(b, d);
#define mnmx5(a, b, c, d, e)     s2(a, b); s2(c, d); mn3(a, c, e); mx3(b, d, e);
#define mnmx6(a, b, c, d, e, f) s2(a, d); s2(b, e); s2(c, f); mn3(a, b, c); mx3(d, e, f);
```

- starting with a subset of size 6, remove the min and max each time

Proposed technique of Prof. Mcguire

- starting with a subset of size 6, remove the min and max each time

```
iLocalPixOffset = (get_local_id(1) + 1) * iLocalPixPitch + get_local_id(0) + 1;
uchar window[6];
window[ 0 ] = ldata1[ iLocalPixOffset - (get_local_size(0)+2) - 1 ].x;              // left up
window[ 1 ] = ldata1[ iLocalPixOffset - (get_local_size(0)+2)     ].x;              // up
window[ 2 ] = ldata1[ iLocalPixOffset - (get_local_size(0)+2) + 1 ].x;              // right up
window[ 3 ] = ldata1[ iLocalPixOffset - 1                         ].x;              // left
window[ 4 ] = ldata1[ iLocalPixOffset                             ].x;              // center
window[ 5 ] = ldata1[ iLocalPixOffset + 1                         ].x;              // right


// Starting with a subset of size 6, remove the min and max each time
uchar temp;
mnmx6(window[0], window[1], window[2], window[3], window[4], window[5]);

window[5] = ldata1[ iLocalPixOffset + (get_local_size(0)+2) - 1 ].x;                // left down

mnmx5(window[1], window[2], window[3], window[4], window[5]);

window[5] = ldata1[ iLocalPixOffset + (get_local_size(0)+2)     ].x;                // down

mnmx4(window[2], window[3], window[4], window[5]);

window[5] = ldata1[ iLocalPixOffset + (get_local_size(0)+2) + 1 ].x;                // right down

mnmx3(window[3], window[4], window[5]);


uchar4  out = ldata1[ iLocalPixOffset ];
out.x       = window[ 4 ];
g_out[iDevGMEMOffset + get_global_size(0)] = out;
```

The number of branches(Geforce 8600)

- Inserting Sort

branch	divergent branch	instructions	warp serialize
2987955	534157	12113430	1775349

- Macguire's techniques

branch	divergent branch	instructions	warp serialize
495270	32580	6718850	5090112

The execution time(Geforce 8600)

- Inserting Sort
  - 100 iterations / total elasped time 1470 msec / average time 14 msec

- Macguire's techniques
  - 100 iterations / total elasped time 1035 msec / average time 10 msec

Thanks