Iterative Disparity Voting Based Stereo Matching Algorithm and Its Hardware Implementation

Zhi Hu, Yibo Fan, Xiaoyang Zeng
School of Microelectronics
Fudan University, Shanghai, China

Outline

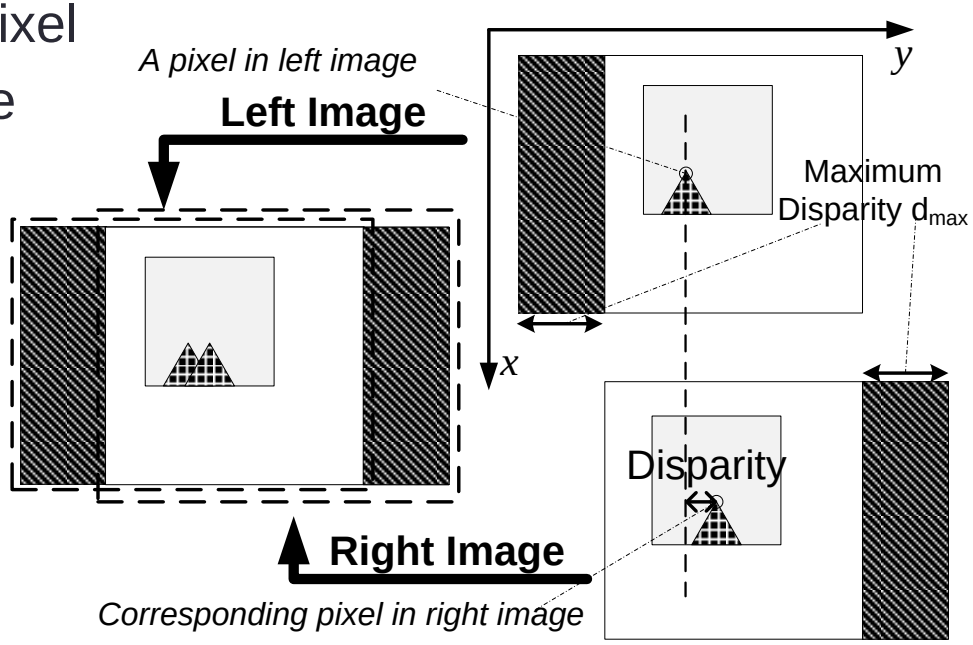
- **Introduction**
  - **Background**
  - **Motivation**
- Proposed Algorithm
- Proposed Hardware Architecture
- Experimental Results
- Conclusion

Stereo Matching – Problem Definition

- **Extract disparity information from binocular images**
- **Steps in a Nutshell**
  1. Find the corresponding pixel
  2. Use definition to calculate

$$d = y_{p_r} - \left(y_{p_l} - d_{max} \right)$$

- **Key Issue**
  - How to accurately find the corresponding pixel?



Stereo Matching – Vistas and Barriers

- **Prospective Applications**
  - Free View Point TV
  - Robotic Vision
  - Automotive Control
- **The problem we are facing**
  - Software approach: HIGH accuracy + LOW speed
  - Hardware approach: HIGH speed + LOW accuracy

Existing Approaches – An Overview

- **Software-Oriented Approaches: Global Optimization**
  - Graph Cut
  - Belief Propagation
  - Dynamic Programming
- **What Hardware Prefers**
  - Data access with spatial locality
  - Parallelizable computing
  - Can be efficiently mapped to a high-throughput pipeline
- **Hardware-Oriented Approaches: Local Refinement**
  - Coarse Estimation + Efficient & Effective Refinement
  - Locality ⊃ Efficiency? Typically, Yes!
  - Locality ⊃ Quality? **Our goal!**

Steps of a Typical Hardware Approach

1. **Coarse Disparity Map Generation**
   - Compute matching cost → Establish correspondence
     - Census Transform — Not **Robust** enough
     - Pixel-Based SAD — Not **accurate** enough
     - Region-Adaptive SAD — Not **hardware-friendly**
2. **Disparity Map Refinement**
   - Make disparity distribution more aligned to the outline of the objects
     - Bilateral Filter — Need **floating-point** computation for high quality
     - Common Disparity Voting — Improvement is **minor**
3. **Occlusion Detection and Filling**
   - Extrapolate the disparities of occluded areas

Our Contribution

1. **Coarse Disparity Map Generation**
   - Hardware-friendly cost function
   - Hardware architecture for parallel calculation
   - Trade-off Efficiency & Implementation Cost > Accuracy
     - We can leave further enhancement to the subsequent refinement
2. **Disparity Map Refinement**
   - Iterative disparity voting **cascading** + **interleaving**
   - Weighted voting for **error suppression**
   - Memory consumption reduction for **HD** processing
3. **Occlusion Detection and Filling**
   - Pipelined occlusion handling

Outline

- Introduction
- **Proposed Algorithm**
  - **Coarse map generation**
  - **Map refinement**
  - **Occlusion handling**
- Proposed Hardware Architecture
- Experimental Results
- Conclusion

Coarse Map Generation

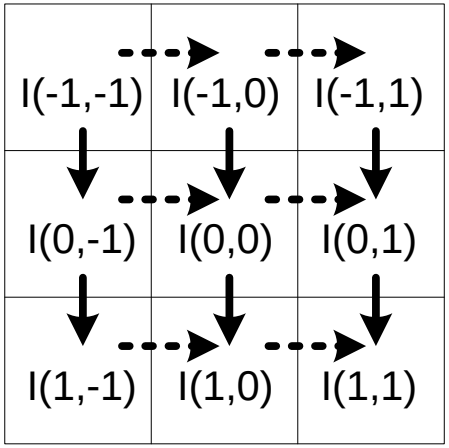
- **Cost computation (use luminance)**
  - Note that result of the horizontally adjacent pixel can be partially used

$$\mathrm{cost}(p_l, p_r) =$$

$$\sum_{-1\le i,j\le 1} \left| I_l(x_{p_l}+i, y_{p_l}+j) - I_r(x_{p_r}+i, y_{p_r}+j) \right| +$$

$$\sum_{-1\le i,j\le 1} \left| \frac{\Delta I_l(x_{p_l}+i, y_{p_l}+j)}{\Delta x} - \frac{\Delta I_r(x_{p_r}+i, y_{p_r}+j)}{\Delta x} \right| +$$

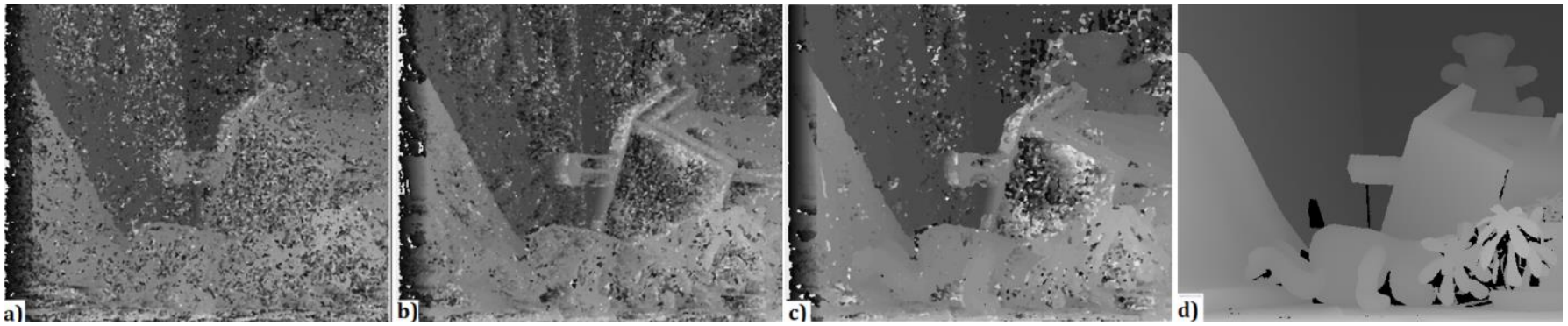
$$\sum_{-1\le i,j\le 1} \left| \frac{\Delta I_l(x_{p_l}+i, y_{p_l}+j)}{\Delta y} - \frac{\Delta I_r(x_{p_r}+i, y_{p_r}+j)}{\Delta y} \right|$$



- **Get the disparity for every pixel (in the left image by default)**
  - Less cost ⊃ Higher probability of being the correct match

$$d_{p_l} = \arg\min_i \mathrm{cost}\left(p_l, p_{r_i}\right)$$

Comparison of Coarse Map Quality



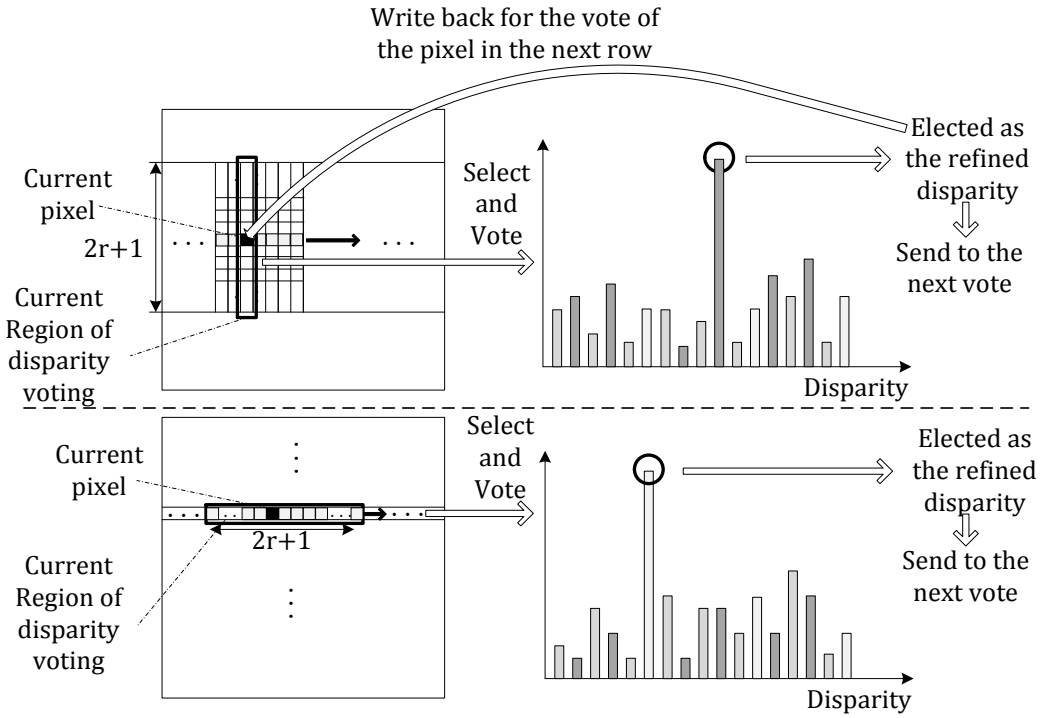
a) Census Transform
b) Pixel-Based SAD
c) Proposed Method
d) Ground Truth

- Less bad estimation
- More friendly for the subsequent refinement

- Denote the coarse disparity map as $D^{(0)}$.

Map Refinement

- **Iterative Weighted Disparity Map Voting**
  - Use neighboring pixels of similar disparity to amend wrong estimation region
- **6 rounds of vertical + horizontal voting**
  - $V_1 \rightarrow H_1 \rightarrow V_2 \rightarrow H_2 \rightarrow V_3 \rightarrow H_3$ After 6 rounds, the result tends to converge
  - Denote the resulting refined map of each stage as $D^{(1)}$, $D^{(2)}$, …, $D^{(6)}$
- **How do the "voting" processes proceed?**



Details of Voting (to be continued)

- **Select candidates for voting (use truncated RGB)**
  - **Assumption**
    - Correlation between luminance difference & disparity difference
  - **Criteria**
    - Vertical
    - Horizontal

$$\forall \text{channel}_C \left(| I_C(x_0+i, y_0) - I_C(x_0, y_0) | \le \tau \right) \quad -r \le i \le r$$
$$\forall \text{channel}_C \left(| I_C(x_0, y_0+j) - I_C(x_0, y_0) | \le \tau \right) \quad -r \le j \le r$$

- **Vote use weight function**
  - **Function**
    - Vertical
    - Horizontal
  - **Why so?**
    - ***i*** term: amend areas where wrong disparities agglomerate
      - Emphasize peripheral pixels
      - Trivial negative impact on small voting areas with correct disparities
        - disparities are virtually the same

$$w_{D^{(k)}(x_0+i, y_0)} = \begin{cases} i/2 + D^{(k)}(x_0+i, y_0)/8 & i \ge 0 \\ -i + D^{(k)}(x_0+i, y_0)/8 & i < 0 \end{cases}$$

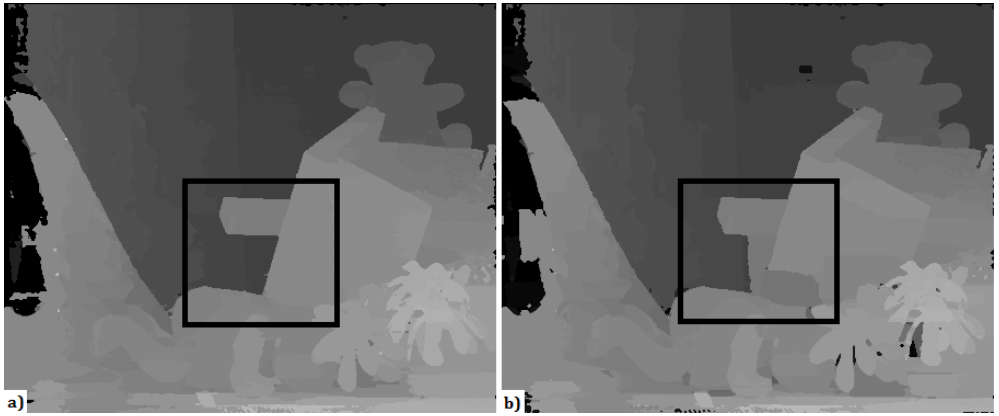
$$w_{D^{(k)}(x_0, y_0+j)} = |i| + D^{(k)}(x_0, y_0+j)/8$$

Details of Voting (continue)

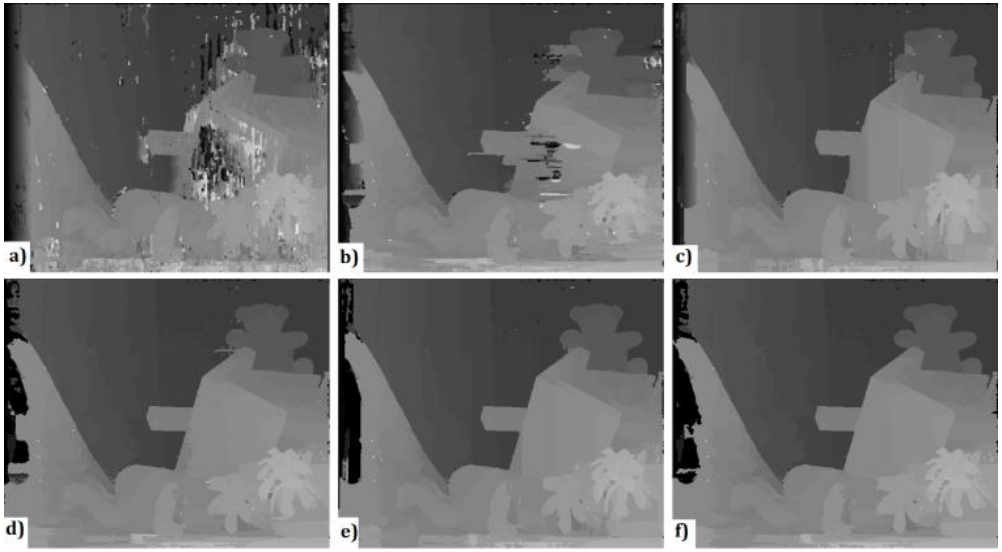
- **Vote use weight function**
  - **Function**
    - Vertical $$w_{D^{(k)}(x_0+i,y_0)} = \begin{cases} i/2 + D^{(k)}(x_0+i,y_0)/8 & i \geq 0 \\ -i + D^{(k)}(x_0+i,y_0)/8 & i < 0 \end{cases}$$
    - Horizontal $$w_{D^{(k)}(x_0,y_0+j)} = |i| + D^{(k)}(x_0,y_0+j)/8$$
  - **Why so?**
    - ***i*** term
      - Vertical ***i* / 2** and ***–i*** : write-back strategy (pixels above are more accurate)
      - Horizontal ***|i|***: write-back not used (timing)
    - $D^{(k)}$ term: reduce the effect of pixels with extremely low disparity
      - Empirically, wrongly-estimated disparities could be small
      - Background area with low disparity and smooth disparity transition is not affected by the term

Some Intermediate Results

- **With and without weight function**
  - Major differences are highlighted by black squares



- **Generated refined disparity map after each vote**
  - a) → f):
    - $D^{(1)}$, $D^{(2)}$, $D^{(3)}$, $D^{(4)}$, $D^{(5)}$, $D^{(6)}$
  - Refine → Converge



Occlusion Handling

- **Method:** Left-Right Check
- **Is this pixel Occluded?**
  - We Define
    - $D^{(6)}_l$: output map of the final voting of the left image
    - $D^{(6)}_r$: output map of the final voting of the right image
  - Criterion

$$\left| D_l^{(6)}(x,y) - D_r^{(6)}\left(x, y - D_l^{(6)}(x,y)\right) \right| \le \tau$$

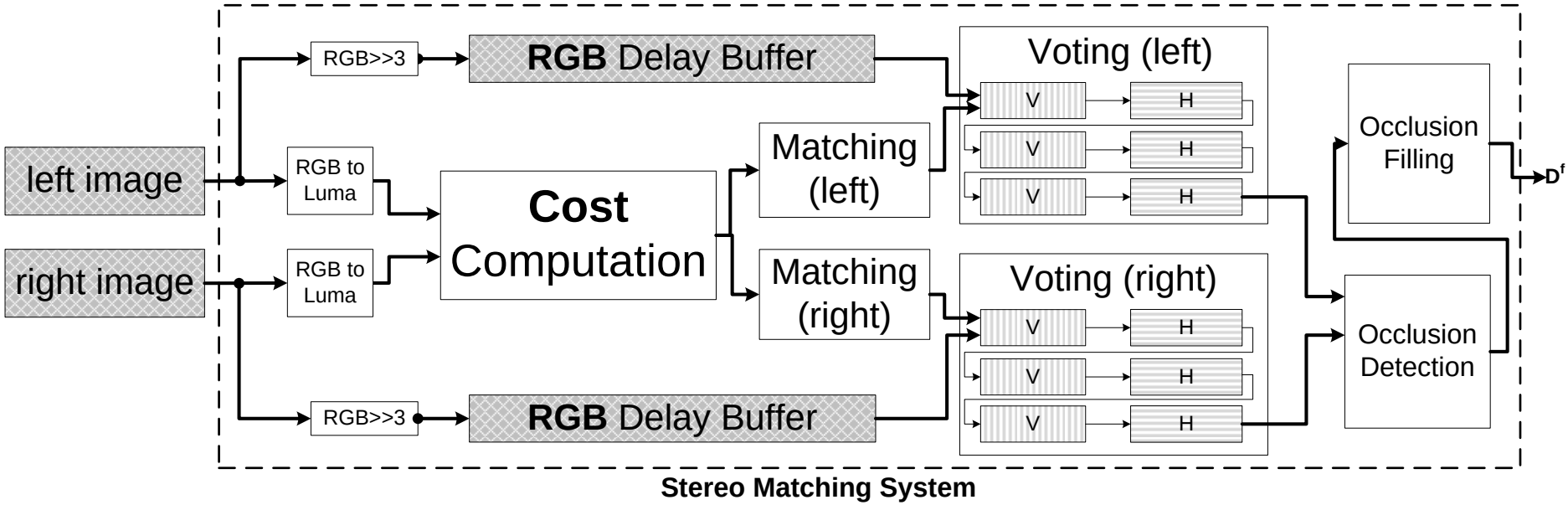
    - τ = 1
- **How to extrapolate?**
  - Two-pass scan & flood-fill
    - From left → right: occluded? Most immediate non-occluded pixel on the left
    - From right → left: occluded? Most immediate non-occluded pixel on the right

Outline

- Introduction
- Proposed Algorithm
- **Proposed Hardware Architecture**
  - **Overview**
  - **Coarse map generation**
  - **Map refinement**
  - **Occlusion handling**
- Experimental Results
- Conclusion

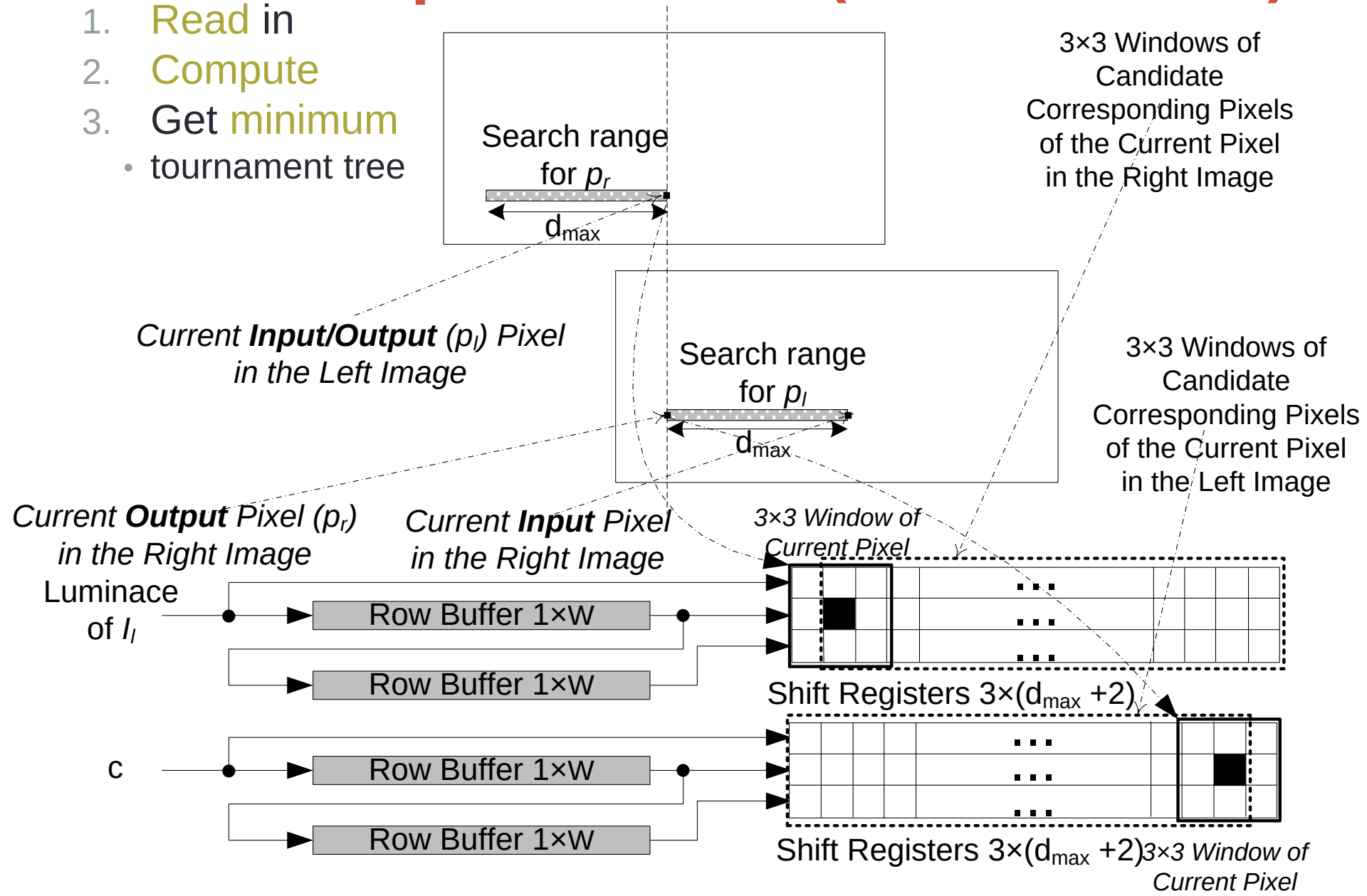
Overall Architecture

- Left and right image computed in parallel
- Fully-Pipelined
- Note the truncation (reduce buffer size)



Coarse Map Generation (to be continued)

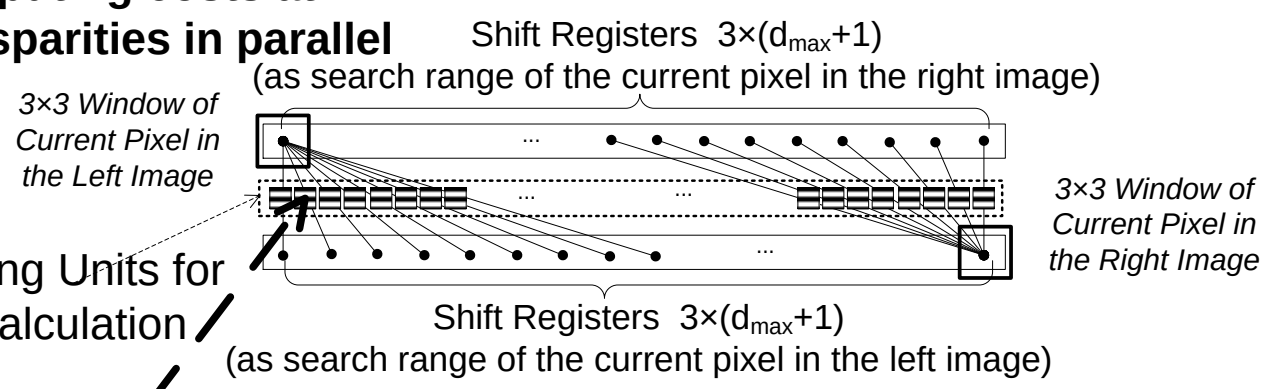
1. Read in
2. Compute
3. Get minimum
   - tournament tree



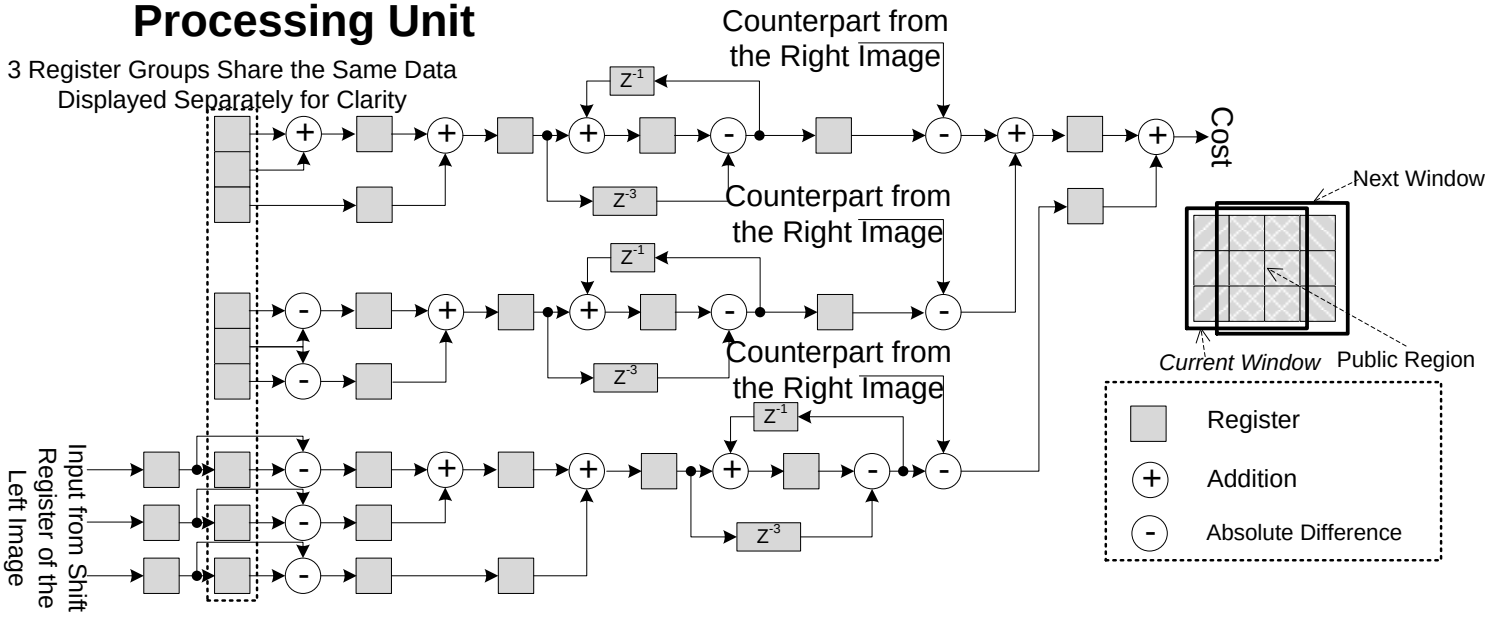
• Cost Computation

- Compute (note the public region whereby results can be shared)

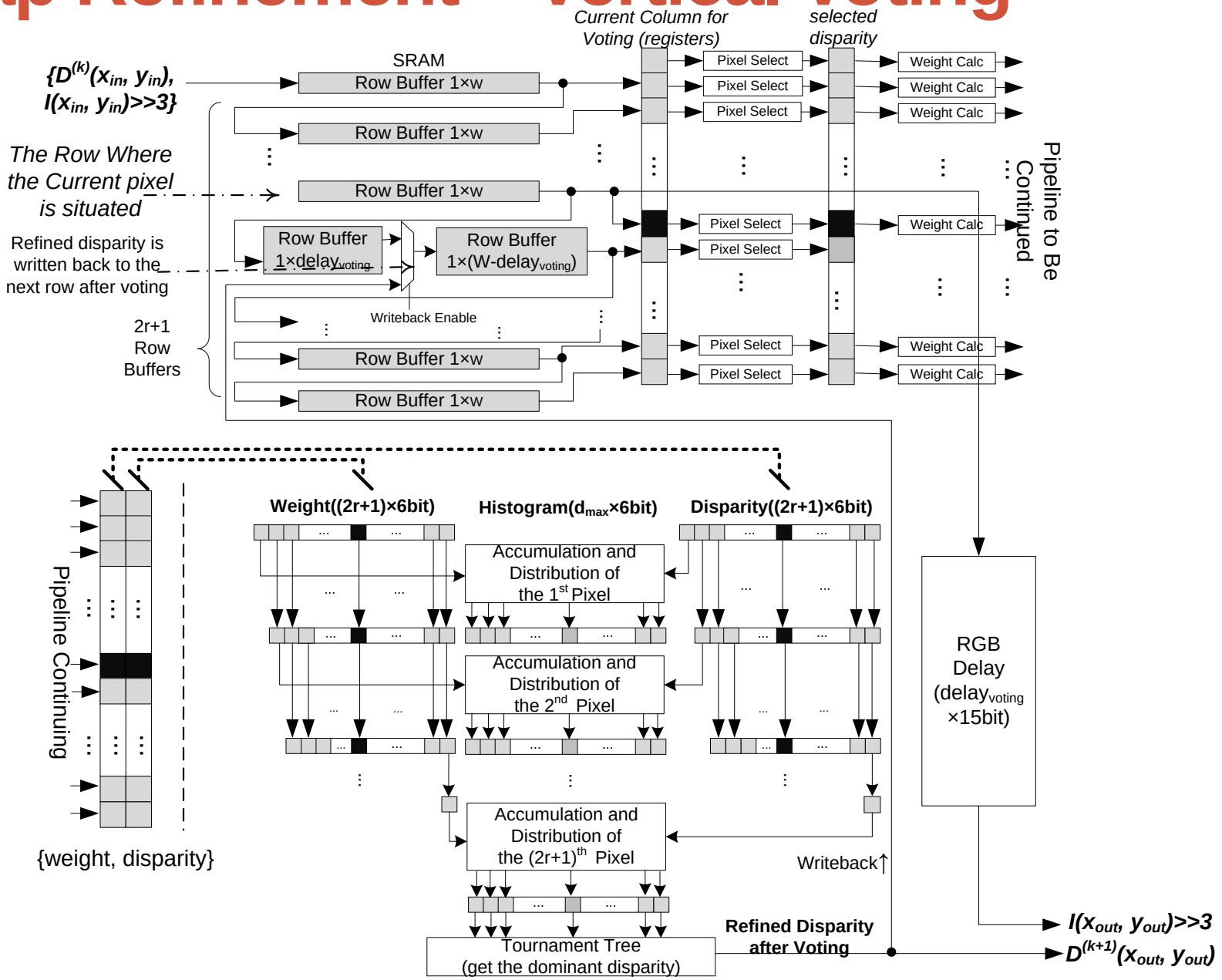
$2d_{max}$ pairs of 3×3 Windows for computing costs at different disparities in parallel



Implementation Detail of Each Processing Unit

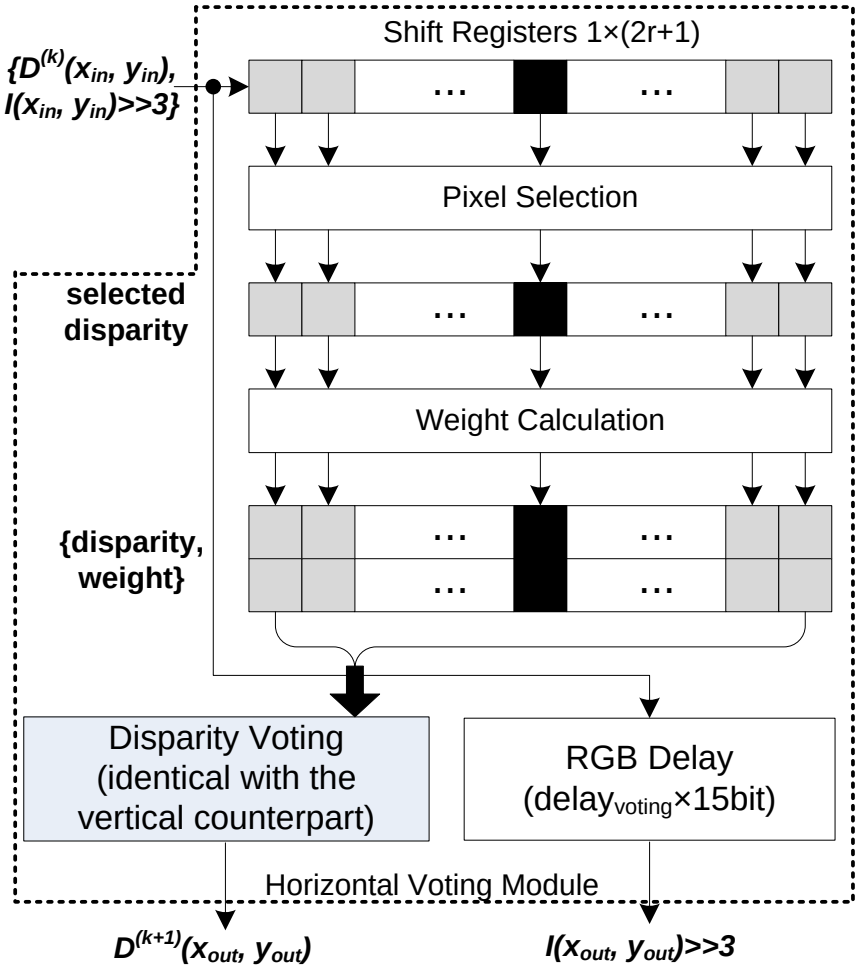


Map Refinement – Vertical Voting



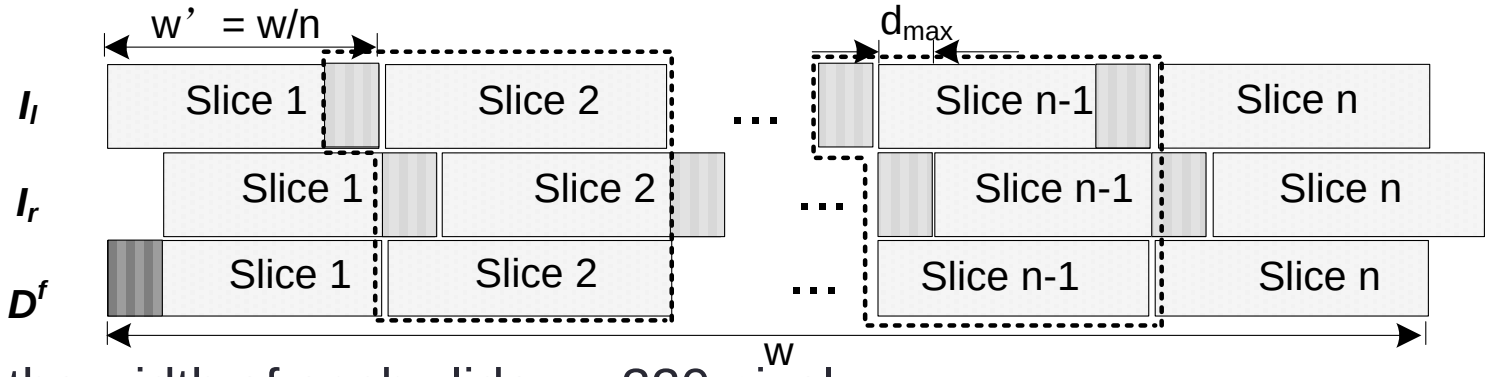
Architecture Observations

- **Vertical Voting Architecture (previous slide)**
  - Row buffer
  - Delay buffer
  - Write back
- **Horizontal Voting Architecture**
  - Generally the same
  - No write back
    - TIMING!



Memory Reduction of Vertical Voting

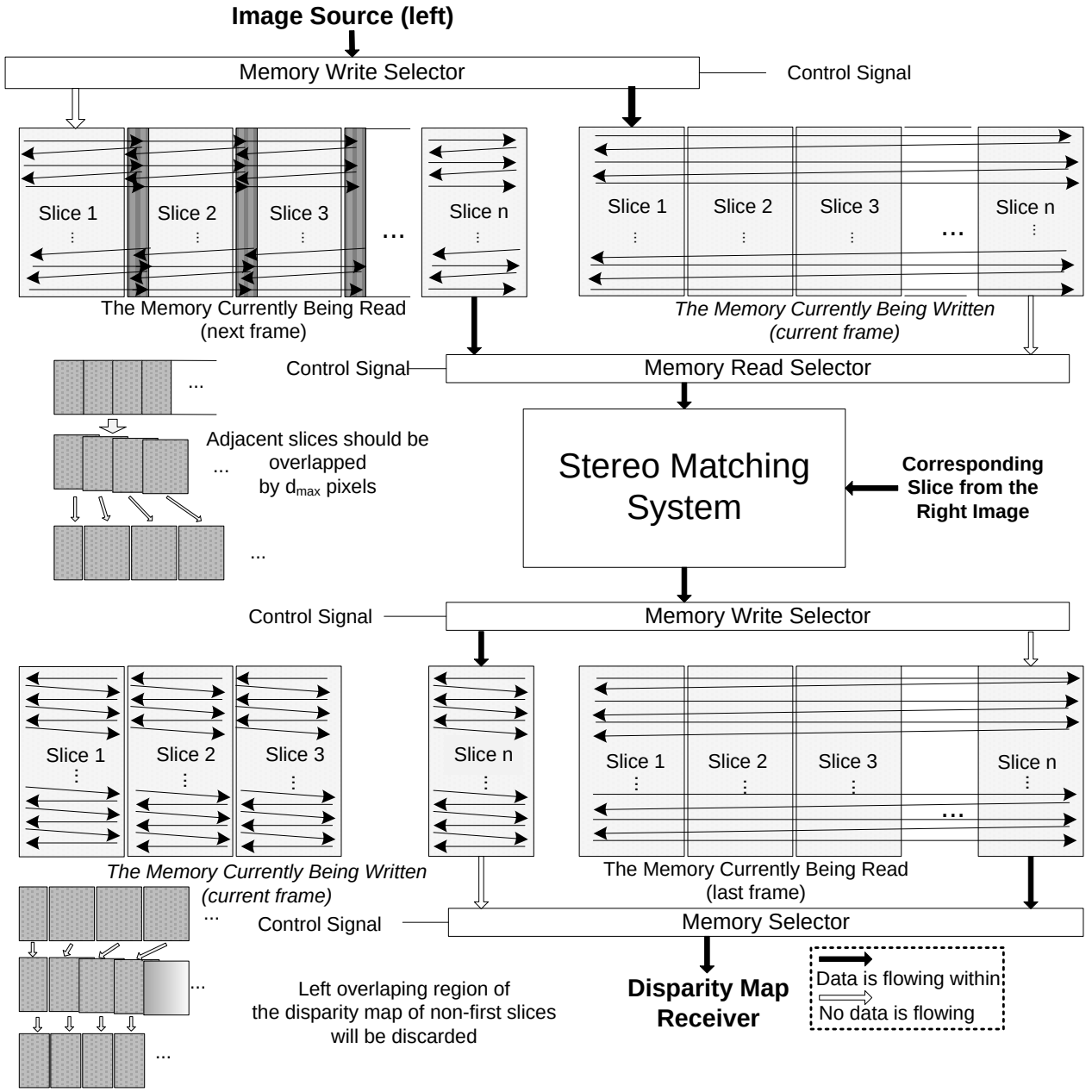
- **Rough Estimation**
  - For an 1920×1080 image with 64 disparity level
    - Vertical voting row buffer size: 620KB!!
- **Solution**
  - Slice-Based Processing
  - Correctness? Extend slices into blind zones.



  - Set the width of each slides = 320 pixels
    - Vertical voting row buffer size: 124KB
  - But slight increase in computation time due to the extensions

Raster → Slice-Based scan-order

Ping-pong buffers



Ping-pong buffers

Occlusion Handling

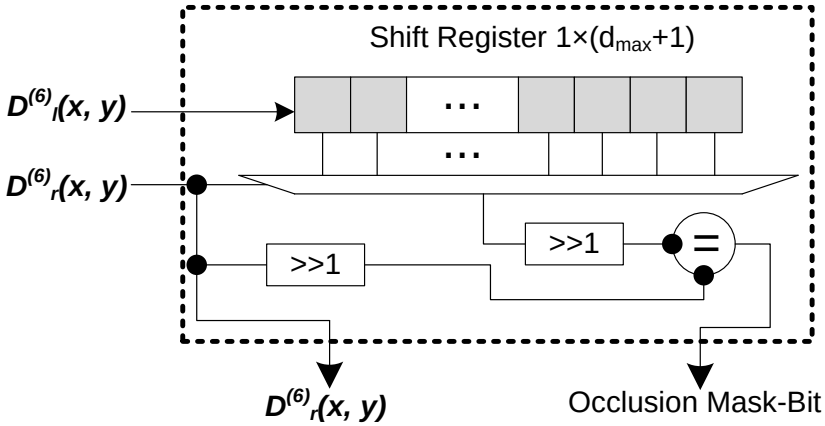
- **Occlusion Detection**
  - Criterion given before

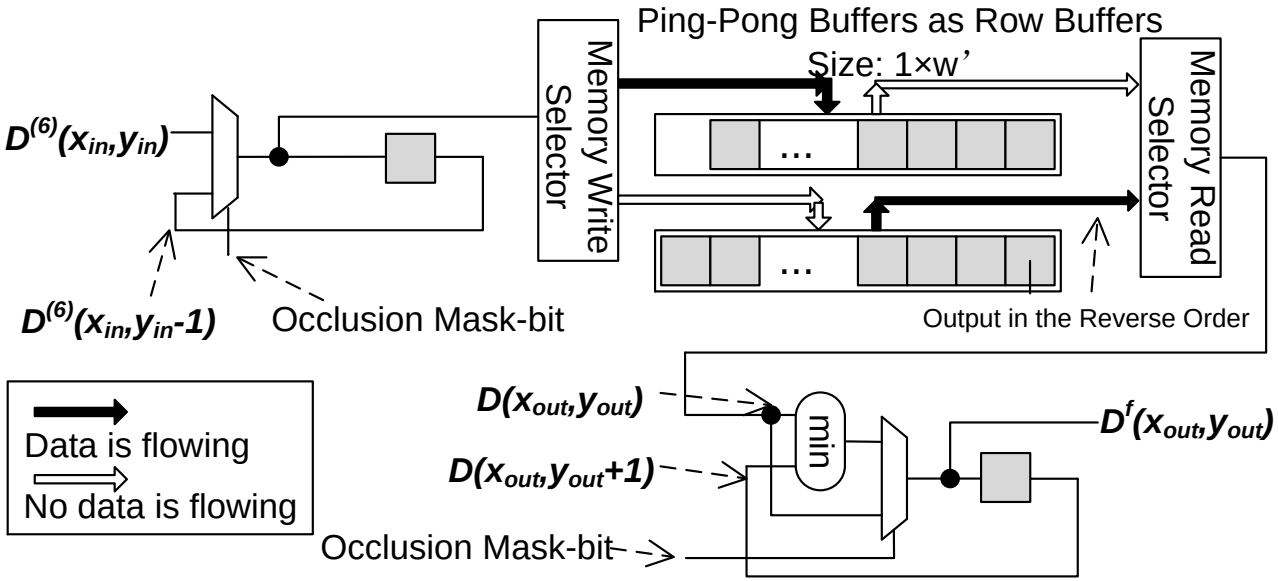
$$\left| D_l^{(6)}(x,y) - D_r^{(6)}\left(x, y - D_l^{(6)}(x,y)\right) \right| \le 1$$

  - In practice, judge
    - $D^{(6)}_l(x, y) / 2 = D^{(6)}_r(x, y - D^{(6)}_l(x, y)) ?$
- **Occluded Pixel Extrapolation**
  - Ping-pong buffer

    to avoid stalls

    in pipeline





Outline

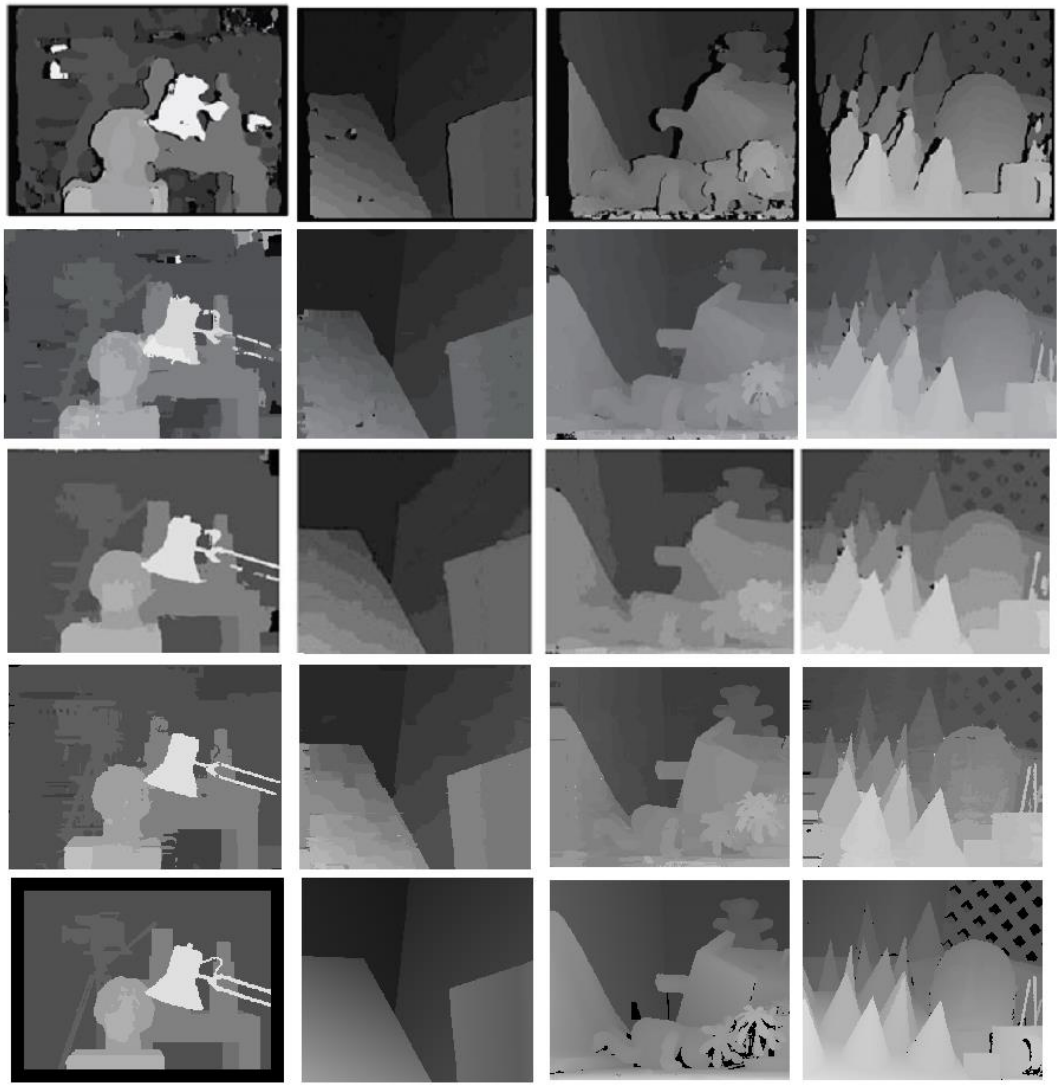
- Introduction
- Proposed Algorithm
- Proposed Hardware Architecture
- **Experimental Results**
  - **Hardware Resource Consumption and Performance**
  - **Quality of Output Results**
- Conclusion

Overhead and Performance

- Synthesized on Altera Stratix-IV EP4S40G2
- Problem scale: 1920x1080 images with 64 disparity levels
  - $\mathbf{f_{max}}$ = 121.76MHz
    - 1920x1080-image @ 48fps
    - MDES(millions of disparities estimated/second) 6370M
  - **Memory** 1,040kbit/14,283kbit (7.3%)
  - **Registers** 96,398/182,400(52.8%)
  - **Combinational ALUTs** 104,632/182,400(57.3%)

Quality of Output Results

- **Test images: Middlebury**
  - Tsukuba, Venus, Teddy, Cones
- **1st to 3rd row**
  - Selected existing works
- **4th row**
  - Our work
- **5th row**
  - Ground Truth
- High-Quality Results
  - But not very well in the Venus test case
    - Involves large-area diagonal disparity gradient
    - Could be improved in the future



Conclusion

- Proposed an iterative weighted disparity voting based local stereo matching algorithm
- Proposed corresponding hardware architecture
- Synthesis predicts system's capability in processing 1920×1080-image @ 48fps with low memory cost
- Test results reflect high quality

Thank you very much!

Any questions?