Fast Median and Bilateral Filtering

Ben Weiss, Shell & Slate Software
Why Median?
Why Median?

- Fundamental
Why Median?

• Fundamental
  – Mean Minimizes $\sum |I_p - m|$
Why Median?

• Fundamental
  - Mean Minimizes $\sum |I_p - m|$  
  - Median Minimizes $\sum |I_p - m|$
Why Median?

- **Fundamental**
  - Mean Minimizes $\sum |I_p - m|$
  - Median Minimizes $\sum |I_p - m|$
Why Median?

• Fundamental
  - Mean Minimizes $\sum (I_p - m)$
  - Median Minimizes $\sum |I_p - m|$
Why Median?

- **Fundamental**
  - Mean Minimizes $\sum |I_p - m|$
  - Median Minimizes $\sum |I_p - m|$
Why Median?

• Fundamental
  – Mean Minimizes \[ \sum |I_p - m| \]
  – Median Minimizes \[ \sum |I_p - m| \]
Why Median?

- Fundamental
  - Mean Minimizes $\sum |I_p - m|$
  - Median Minimizes $\sum |I_p - m|$

Gaussian

Median

SIGGRAPH2006
Why Median?

• Fundamental
  – Mean Minimizes \( \sum |I_p - m| \)
  – Median Minimizes \( \sum |I_p - m| \)
Why Median?

• Fundamental
  – Mean Minimizes $\sum (I_p - m)$
  – Median Minimizes $\sum |I_p - m|$
Why Median?

• Fundamental
  – Mean Minimizes $\sum |I_p - m|$
  – Median Minimizes $\sum |I_p - m|$

• Challenging
Why Median?

• Fundamental
  – Mean Minimizes $\sum |I_p - m|$ 
  – Median Minimizes $\sum |I_p - m|$ 

• Challenging
  – Minimum $= 0^{th}$ Percentile $= Easy$
Why Median?

- **Fundamental**
  - Mean Minimizes $\sum |I_p - m|$
  - Median Minimizes $\sum |I_p - m|$

- **Challenging**
  - Minimum = 0th Percentile = Easy
  - Maximum = 100th Percentile = Easy
Why Median?

• Fundamental
  – Mean Minimizes $\left| \sum I_p - m \right|$
  – Median Minimizes $\sum |I_p - m|$

• Challenging
  – Minimum = 0\textsuperscript{th} Percentile = Easy
  – Maximum = 100\textsuperscript{th} Percentile = Easy
  – Median = 50\textsuperscript{th} Percentile = Hard!
Why Median?

- **Fundamental**
  - Mean Minimizes $\sum |I_p - m|$
  - Median Minimizes $\sum |I_p - m|$

- **Challenging**
  - Minimum = 0th Percentile = Easy
  - Maximum = 100th Percentile = Easy
  - Median = 50th Percentile = Hard!

- **Versatile**
Outline
Outline

- 8-bit Median
Outline

- 8-bit Median
- High-Precision Median
Outline

• 8-bit Median
• High-Precision Median
• Bilateral Filter
Outline

- 8-bit Median
- High-Precision Median
- Bilateral Filter
- Creative Applications
Standard 8-Bit $O(r)$ Algorithm
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
- 256-Element Histogram $H$
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
- 256-Element Histogram $H$
  - Initialize $H$ to Top Left
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
- 256-Element Histogram $H$
  - Initialize $H$ to Top Left
  - Extract Median from $H$
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
- 256-Element Histogram $H$
  - Initialize $H$ to Top Left
  - Extract Median from $H$
  - Slide Window, Update $H$
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
- 256-Element Histogram $H$
  - Initialize $H$ to Top Left
  - Extract Median from $H$
  - Slide Window, Update $H$
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
- 256-Element Histogram $H$
  - Initialize $H$ to Top Left
  - Extract Median from $H$
  - Slide Window, Update $H$
Standard 8-Bit $O(r)$ Algorithm

- One Column at a Time
- 256-Element Histogram $H$
  - Initialize $H$ to Top Left
  - Extract Median from $H$
  - Slide Window, Update $H$
- Redundant Computation
Parallel $O(r)$ Algorithm
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns ↔ 9 Windows
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns ↔ 9 Windows
- 9 Windows ↔ 9 Histograms
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\leftrightarrow$ 9 Windows
- 9 Windows $\leftrightarrow$ 9 Histograms
- Histograms are Distributive
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\leftrightarrow$ 9 Windows
- 9 Windows $\leftrightarrow$ 9 Histograms
- Histograms are **Distributive**
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\Leftrightarrow$ 9 Windows
- 9 Windows $\Leftrightarrow$ 9 Histograms
- Histograms are **Distributive**
  - $H_n =$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\leftrightarrow$ 9 Windows
- 9 Windows $\leftrightarrow$ 9 Histograms
- Histograms are **Distributive**
  - $H_n = H_c$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\Leftrightarrow$ 9 Windows
- 9 Windows $\Leftrightarrow$ 9 Histograms
- Histograms are Distributive
  - $H_n = H_c + (H_n - H_c)$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\Leftrightarrow$ 9 Windows
- 9 Windows $\Leftrightarrow$ 9 Histograms
- Histograms are **Distributive**
  - $H_n = H_c + (H_n - H_c)$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\leftrightarrow$ 9 Windows
- 9 Windows $\leftrightarrow$ 9 Histograms
- Histograms are **Distributive**
  - $H_n = H_c + (H_n - H_c)$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\iff$ 9 Windows
- 9 Windows $\iff$ 9 Histograms
- Histograms are **Distributive**
  - $H_n = H_c + (H_n - H_c)$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\Leftrightarrow$ 9 Windows
- 9 Windows $\Leftrightarrow$ 9 Histograms
- Histograms are **Distributive**
  \[-H_{A \cup B}[v] = H_A[v] + H_B[v]\]
  \[-H_n = H_c + (H_n - H_c)\]
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\Leftrightarrow$ 9 Windows
- 9 Windows $\Leftrightarrow$ 9 Histograms
- Histograms are Distributive
  - $H_n = H_c + (H_n - H_c)$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\leftrightarrow$ 9 Windows
- 9 Windows $\leftrightarrow$ 9 Histograms
- Histograms are **Distributive**
  
  $$H_n = H_c + (H_n - H_c)$$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\iff$ 9 Windows
- 9 Windows $\iff$ 9 Histograms
- Histograms are **Distributive**
  - $H_n = H_c + (H_n - H_c)$
Parallel $O(r)$ Algorithm

- E.g., Nine Columns at Once
- 9 Columns $\leftrightarrow$ 9 Windows
- 9 Windows $\leftrightarrow$ 9 Histograms
- Histograms are Distributive
  - $H_n = H_c + (H_n - H_c)$
- Approaches ninefold speedup
Adaptive $O(r^{1/2})$ Algorithm
Adaptive $O(r^{\frac{1}{2}})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
Adaptive $O(r^{1/2})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
Adaptive $O(r^{\frac{1}{2}})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
Adaptive $O(r^{1/2})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
Adaptive $O(r^{1/2})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
Adaptive $O(r^{1/2})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
- For Fixed $N$, Radius $r$ is Variable
Adaptive $O(r^{1/2})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
- For Fixed $N$, Radius $r$ is Variable
Adaptive $O(r^{1/2})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
- For Fixed $N$, Radius $r$ is Variable
Adaptive $O(r^{\frac{1}{2}})$ Algorithm

- For Fixed Radius $r$, $N$ is Variable
- For Fixed $N$, Radius $r$ is Variable
- $N \approx r^{\frac{1}{2}}$ yields $O(r^{\frac{1}{2}})$ Complexity
The $O(\log r)$ Algorithm
The $O(\log r)$ Algorithm

- Add Histogram *Tiers* with Radius
The $O(\log r)$ Algorithm

- Add Histogram **Tiers** with Radius
  - $H_n = H_c$
The $O(\log r)$ Algorithm

- Add Histogram Tiers with Radius
  \[ H_n = H_c + (H_d - H_c) \]
The $O(\log r)$ Algorithm

- Add Histogram Tiers with Radius
  
  \[ H_n = H_c + (H_d - H_c) + (H_n - H_d) \]
The $O(\log r)$ Algorithm

- Add Histogram Tiers with Radius

$$H_n = H_c + (H_d - H_c) + (H_n - H_d) + \ldots$$
The $O(\log r)$ Algorithm

- Add Histogram Tiers with Radius
  \[ H_n = H_c + (H_d - H_c) + (H_n - H_d) + \ldots \]
- Constant Radix yields $O(\log r)$ Algorithm
8-Bit Median - Performance

PowerMac G5 2.5GHz – Single Processor – 8-Megapixel RGB Image

- Photoshop® CS2 – $O(r)$ Algorithm
- Our $O(\log r)$ Algorithm

Processing Time (seconds)

Filter Radius (pixels)
8-Bit Median - Performance

Demo
8-Bit Algorithm  ⇒  16-Bit
8-Bit Algorithm ⇒ 16-Bit

- What’s a Few Extra Bits?
8-Bit Algorithm ⇒ 16-Bit

- What’s a Few Extra Bits?
- Histogram Size: 256 ⇒ 65536 Elements!
8-Bit Algorithm ⇒ 16-Bit

- What’s a Few Extra Bits?
- Histogram Size: 256 ⇒ 65536 Elements!
- But... Histogram becomes Sparse
8-Bit Algorithm ⇒ 16-Bit

- What’s a Few Extra Bits?
- Histogram Size: 256 ⇒ 65536 Elements!
- But... Histogram becomes Sparse
  - Many Elements will be 0 or 1
8-Bit Algorithm ⇒ 16-Bit

- What’s a Few Extra Bits?
- Histogram Size: 256 ⇒ 65536 Elements!
- But... Histogram becomes Sparse
  - Many Elements will be 0 or 1
  - Still Need to Handle All Cases
8-Bit Algorithm ⇒ 16-Bit

• What’s a Few Extra Bits?
• Histogram Size: 256 ⇒ 65536 Elements!
• But… Histogram becomes Sparse
  – Many Elements will be 0 or 1
  – Still Need to Handle All Cases
• Solution?
The Ordinal Transform

• Make Each Source Value Unique
The Ordinal Transform

- Make Each Source Value Unique
  - Enables Binary Histograms, 16x Smaller
The Ordinal Transform

- Make Each Source Value Unique
  - Enables Binary Histograms, 16x Smaller
- Technique: Cardinal ↔ Ordinal Mapping
The Ordinal Transform

- Make Each Source Value Unique
  - Enables Binary Histograms, 16x Smaller
- Technique: Cardinal ↔ Ordinal Mapping
The Ordinal Transform

- Make Each Source Value Unique
  - Enables Binary Histograms, 16x Smaller
- Technique: Cardinal ↔ Ordinal Mapping
- Median is Invariant under Ordinal Transform!
The Ordinal Transform

- Make Each Source Value Unique
  - Enables Binary Histograms, 16x Smaller
- Technique: Cardinal ↔ Ordinal Mapping
- Median is Invariant under Ordinal Transform!
- Apply Median to *Ordinal* Image
The Compound Histogram
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

• Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
- $H_c \Rightarrow H_n$ in Constant Time!
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
- $H_c \Rightarrow H_n$ in Constant Time!
- 8-Bit $H_c \leftrightarrow 128$ Columns
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
- $H_c \Rightarrow H_n$ in Constant Time!
- 8-Bit $H_c \leftrightarrow 128$ Columns
- Hybrid Algorithm:
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
- $H_c \Rightarrow H_n$ in Constant Time!
- 8-Bit $H_c \leftrightarrow 128$ Columns
- Hybrid Algorithm:
  - 8-Bit Algorithm on High Bits
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
- $H_c \Rightarrow H_n$ in Constant Time!
- 8-Bit $H_c \leftrightarrow 128$ Columns
- Hybrid Algorithm:
  - 8-Bit Algorithm on High Bits
  - $H_c$ for Fine Tuning
The Compound Histogram

- Redundancy in Binary $H_n$
- Store Offsets in a “Compound” Histogram: $H_c$
- $H_c \Rightarrow H_n$ in Constant Time!
- 8-Bit $H_c \leftrightarrow 128$ Columns
- Hybrid Algorithm:
  - 8-Bit Algorithm on High Bits
  - $H_c$ for Fine Tuning
  - Overall $O(\log^2 r)$ Complexity
16-Bit Median - Performance

PowerMac G5 2.5GHz – Single Processor – 8-Megapixel RGB Image

- Photoshop® CS2 – $O(r)$ Algorithm
- Our $O(\log^2 r)$ Algorithm
Bilateral Filtering

- Nonlinear Weighted Convolution
Bilateral Filtering

• Nonlinear Weighted Convolution
  – Bilateral = Blur that Favors “Similar” Values
Bilateral Filtering

- Nonlinear Weighted Convolution
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference
Bilateral Filtering

- Nonlinear Weighted Convolution
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference

\[
J_s = \frac{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p}{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)}
\]
Bilateral Filtering

- **Nonlinear Weighted Convolution**
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference

$$J_s = \sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p \div \sum_{p \in \Omega} f(p - s)g(I_p - I_s)$$
Bilateral Filtering

- **Nonlinear Weighted Convolution**
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference

\[ J_s = \sum_{p \in \Omega} f(p - s) g(I_p - I_s) I_p / \sum_{p \in \Omega} f(p - s) g(I_p - I_s) \]
Bilateral Filtering

• Nonlinear Weighted Convolution
  – Bilateral = Blur that Favors “Similar” Values
  – Weighted by Spatial and Intensity Difference

\[ J_s = \sum_{p \in \Omega} f(p - s) g(I_p - I_s) I_p \left/ \sum_{p \in \Omega} f(p - s) g(I_p - I_s) \right. \]
Bilateral Filtering

- Nonlinear Weighted Convolution
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference

\[
J_s = \frac{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p}{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)}
\]
Bilateral Filtering

- Nonlinear Weighted Convolution
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference

\[
J_s = \sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p \div \sum_{p \in \Omega} f(p - s)g(I_p - I_s)
\]

- \(f()\) and \(g()\) are typically Gaussian
Bilateral Filtering

- Nonlinear Weighted Convolution
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference

\[ J_s = \frac{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p}{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)} \]
  - \( f() \) and \( g() \) are typically Gaussian

- Special Case - Box Filter
Bilateral Filtering

- **Nonlinear Weighted Convolution**
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference
  - $J_s = \sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p \sum_{p \in \Omega} f(p - s)g(I_p - I_s)$
    - $f()$ and $g()$ are typically Gaussian

- **Special Case - Box Filter**
- **Relative Intensity**
Bilateral Filtering

- Nonlinear Weighted Convolution
  - Bilateral = Blur that Favors “Similar” Values
  - Weighted by Spatial and Intensity Difference

\[ J_s = \sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p \] \[ \sum_{p \in \Omega} f(p - s)g(I_p - I_s) \]

- \( f() \) and \( g() \) are typically Gaussian

- Special Case - Box Filter
- Relative Intensity
- Smooth Spatial Falloff
Special Case - Box Filter Bilateral
Special Case - Box Filter Bilateral

\[ J_s = \frac{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p}{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)} \]
Special Case - Box Filter Bilateral

- $J_s = \sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p \bigg/ \sum_{p \in \Omega} f(p - s)g(I_p - I_s)$
- Box Filter - Uniform Spatial Weighting
Special Case - Box Filter Bilateral

\[ J_s = \frac{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p}{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)} \]

- Box Filter - Uniform Spatial Weighting
Special Case - Box Filter Bilateral

- $J_s = \sum_{p \in \Omega} f(p-s)g(I_p - I_s)I_p / \sum_{p \in \Omega} f(p-s)g(I_p - I_s)$

- Box Filter - Uniform Spatial Weighting

- $J_s = \sum_{p \in W} g(I_p - I_s)I_p / \sum_{p \in W} g(I_p - I_s)$
Special Case - Box Filter Bilateral

\[ J_s = \frac{\sum_{p \in \Omega} f(p - s)(I_p - I_s)I_p}{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)} \]

- Box Filter - Uniform Spatial Weighting

\[ J_s = \frac{\sum_{p \in W} g(I_p - I_s)I_p}{\sum_{p \in W} g(I_p - I_s)} \]

- Can be Computed from Histograms
Special Case - Box Filter Bilateral

- \[ J_s = \sum_{p \in \Omega} \frac{f(p - s)g(I_p - I_s)I_p}{\sum_{p \in \Omega} f(p - s)g(I_p - I_s)} \]

- Box Filter - Uniform Spatial Weighting

- \[ J_s = \sum_{p \in W} \frac{g(I_p - I_s)I_p}{\sum_{p \in W} g(I_p - I_s)} \]

- Can be Computed from Histograms

- \[ J_s = \sum_{v=0}^{255} \frac{H[v]g(v - I_s)v}{\sum_{v=0}^{255} H[v]g(v - I_s)} \]
Special Case - Box Filter Bilateral

- \( J_s = \sum_{p \in \Omega} f(p - s)g(I_p - I_s)I_p \div \sum_{p \in \Omega} f(p - s)g(I_p - I_s) \)

- Box Filter - Uniform Spatial Weighting

- \( J_s = \sum_{p \in W} g(I_p - I_s)I_p \div \sum_{p \in W} g(I_p - I_s) \)

- Can be Computed from Histograms

- \( J_s = \sum_{v=0}^{255} H[v]g(v - I_s)v \div \sum_{v=0}^{255} H[v]g(v - I_s) \)

- Use 8-Bit Median to Generate Histograms
Special Case - Box Filter Bilateral

- \[ J_s = \sum_{p \in \Omega} f(p, s) g(I_p - I_s) I_p \bigg/ \sum_{p \in \Omega} f(p, s) g(I_p - I_s) \]

- Box Filter - Uniform Spatial Weighting

- \[ J_s = \sum_{p \in W} g(I_p - I_s) I_p \bigg/ \sum_{p \in W} g(I_p - I_s) \]

- Can be Computed from Histograms

- \[ J_s = \sum_{v=0}^{255} H[v] g(v - I_s) v \bigg/ \sum_{v=0}^{255} H[v] g(v - I_s) \]

- Use 8-Bit Median to Generate Histograms

- For higher-precision data, blend into 8 bit
Bilateral Filter - Performance

PowerMac G5 2.5GHz – Single Processor – 8-Megapixel RGB Image

- Photoshop® CS2 – $O(r)$ Algorithm
- Our $O(\log r)$ Algorithm

Processing Time (seconds)

Filter Radius (pixels)
Relative Intensity Bilateral
Relative Intensity Bilateral

- The Eye Perceives Relative Brightness
Relative Intensity Bilateral

- The Eye Perceives Relative Brightness
Relative Intensity Bilateral

- The Eye Perceives Relative Brightness
- Logarithmic Mapping
Relative Intensity Bilateral

- The Eye Perceives Relative Brightness
- Logarithmic Mapping
- Scaled Intensity Function
Relative Intensity Bilateral

• The Eye Perceives Relative Brightness
• Logarithmic Mapping
• Scaled Intensity Function
  – Width Proportional to Brightness
Relative Intensity Bilateral

- The Eye Perceives Relative Brightness
- Logarithmic Mapping
- Scaled Intensity Function
  - Width Proportional to Brightness
Bilateral Filter

Demo
String Theory
String Theory
String Theory
String Theory
String Theory
String Theory
String Theory
String Theory
String Theory
Conclusion
Conclusion

• Fast 2D Median Filtering Algorithms
Conclusion

- Fast 2D Median Filtering Algorithms
- Efficient Bilateral Filtering Algorithm
Conclusion

- Fast 2D Median Filtering Algorithms
- Efficient Bilateral Filtering Algorithm
- Future Directions
Conclusion

- Fast 2D Median Filtering Algorithms
- Efficient Bilateral Filtering Algorithm
- Future Directions
  - Theoretical Analysis
Conclusion

- Fast 2D Median Filtering Algorithms
- Efficient Bilateral Filtering Algorithm
- Future Directions
  - Theoretical Analysis
  - GPU Implementation
Conclusion

• Fast 2D Median Filtering Algorithms
• Efficient Bilateral Filtering Algorithm
• Future Directions
  – Theoretical Analysis
  – GPU Implementation
  – 3D Median, Variable Width, Circular, etc.
Conclusion

• Fast 2D Median Filtering Algorithms
• Efficient Bilateral Filtering Algorithm
• Future Directions
  – Theoretical Analysis
  – GPU Implementation
  – 3D Median, Variable Width, Circular, etc.

contact: ben@shellandslate.com
http://www.shellandslate.com/fastmedian.html