

Обработка видео

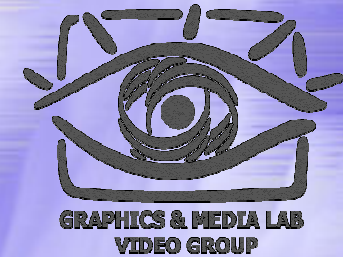
Denoising и Deblocking

Дмитрий Ватолин

*Московский Государственный Университет
CS MSU Graphics&Media Lab*

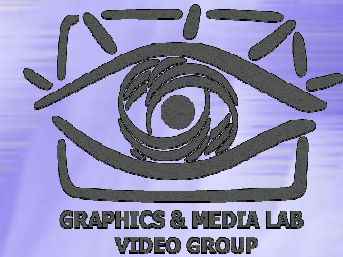
Version 1.0

Благодарности



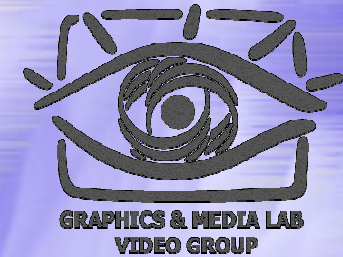
Автор выражает признательность
Сергею Гришину и Дарье
Калинкиной за помощь в подготовке
этих слайдов

Denoising: Содержание



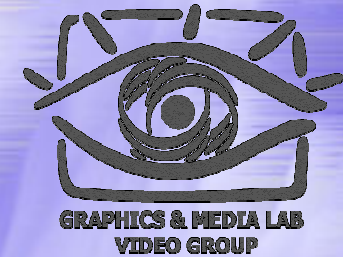
- ◆ **Виды шума и их источники**
- ◆ **Методы подавления шума**
 - Методы, работающие в пространственной области
 - Методы, работающие во временной области (как использующие, так и не использующие компенсацию движения)
 - Методы удаления вертикальных царапин со старых киноплёнок
- ◆ **MSU Noise Remover**
- ◆ **Методы оценки качества шумоподавления**

Виды шума



- ◆ Самый распространенный – **белый гауссовский шум** (описывается следующей функцией плотности распределения: $p(d) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{d^2}{2\sigma^2}}$)
- ◆ **Биение пикселов** (изолированные точки на изображении, значение которых значительно отличается от значений окружающих их точек)
- ◆ **Вертикальные царапины** (характерны для старых черно-белых видеозаписей)

Происхождение шума

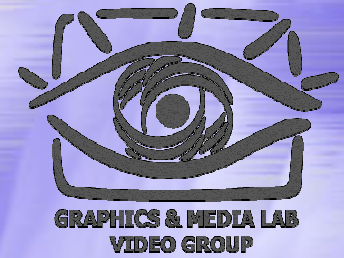


- ◆ Источниками белого и импульсного шума могут быть:
 - неидеальное оборудование для захвата изображения (видеокамера и.т.п.)
 - помехи при передаче по аналоговым каналам
 - неточности при выделении яркостного и цветоразностных сигналов из аналогового композитного сигнала
 - и. т. д.
- ◆ Вертикальные царапины возникают при механическом повреждении эмульсии на пленке.

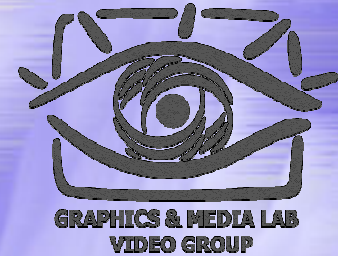
Шум в телевидении



Шум на старых фильмах

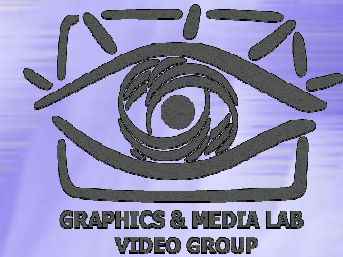


Зачем нужно подавлять шум



- ◆ Улучшается визуальное качество изображения
- ◆ Подавление шума очень важно при сжатии видео:
 - Уменьшается межкадровая разница, и, как следствие, увеличивается степень сжатия
 - Лучше работают алгоритмы компенсации движения

Типы алгоритмов шумоподавления



Все алгоритмы шумоподавления можно разделить на два типа:

- ◆ **Пространственные (spatial)**

Основная идея : усреднение значений пикселей на каждом отдельном кадре.

Проблема: могут пострадать мелкие детали, соизмеримые по размеру с шумом и четкость краев предметов.

- ◆ **Временные (temporal)**

Основная идея: усреднение значений пикселей между кадрами.

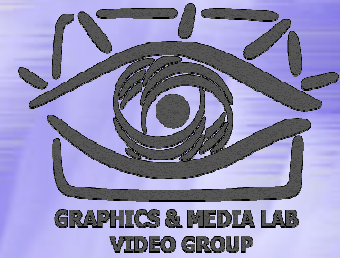
Проблема: при сильном движении появляются такие артефакты, как motion blur и ghosting.

Denoising: Содержание



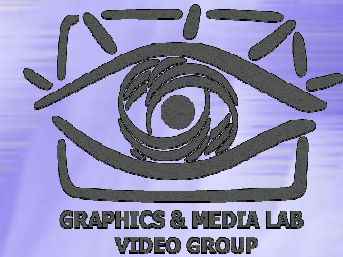
- ◆ **Виды шума и их источники**
- ◆ **Методы подавления шума**
 - **Методы, работающие в пространственной области**
 - Методы, работающие во временной области (как использующие, так и не использующие компенсацию движения)
 - Методы удаления вертикальных царапин со старых киноплёнок
- ◆ **MSU Noise Remover**
- ◆ **Методы оценки качества шумоподавления**

Методы пространственного шумоподавления

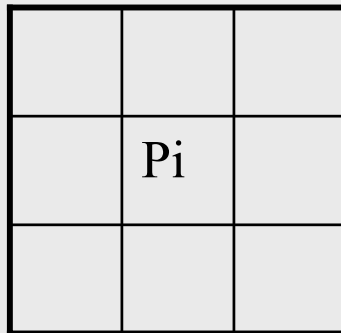


- ◆ **Линейное усреднение пикселов по соседям**
- ◆ Вейвлет-преобразование
- ◆ Медианная фильтрация
- ◆ Гауссовское размытие
- ◆ Математическая морфология

2D Cleaner



```
for (each pixel of the current video frame){
    GetRGB (source_pixel, r, g, b);
    tot_red = tot_green = tot_blue = 0;
    count_red = count_green = count_blue = 0;
    for (each pixel in the specified radius){
        GetRGB (neighbour_pixel, r1, g1, b1);
        if (abs(r1-r) < Threshold) tot_red += r1; count_red ++;
        if (abs(g1-g) < Threshold) tot_green += g1; count_green ++;
        if (abs(b1-b) < Threshold) tot_blue += b1; count_blue ++;
    }
    destination_pixel = RGB (tot_red/count_red,
                             tot_green / count_green ,
                             tot_blue / count_blue );
}
```

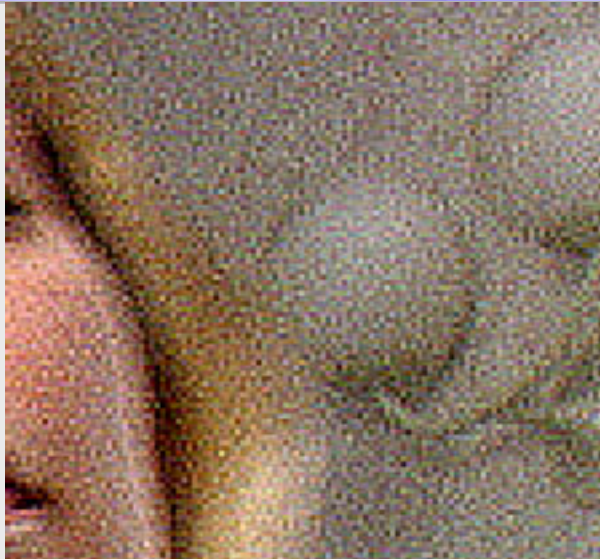
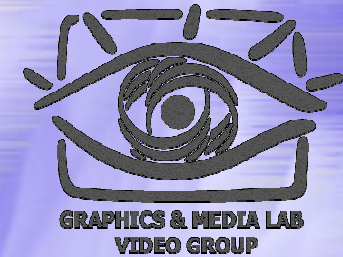


Параметры:

Threshold

Radius

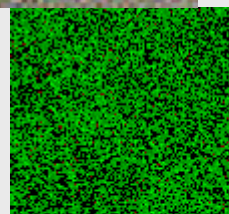
2D Cleaner: Results



Source image



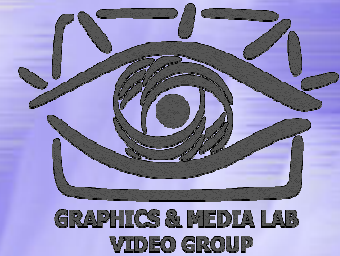
Processed image



1-st image: X:\Report\input.bmp
2-nd image: X:\Report\2d cleaner.bmp
MSE: 0,839 NMSE: 0,029 Max Diff RGB: 7,1
SNR: 15,352 PSNR: 40,761 Max Diff LUV: 4,2
Black: 32,8% Green: 66% Red: 1%

Comparison of the source and processed images using LUV Metric

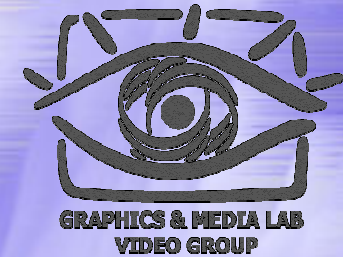
Spatial Smoother



```
for (i=0; i<=511; ++i) square_tab[i] = (i-255) ^ 2;

for (each pixel of the current video frame){
    GetRGB (source_pixel, r, g, b);
    tot_red = tot_green = tot_blue = 0;
    count = 0;
    r_tab = square_tab[255 - r];
    g_tab = square_tab[255 - g];
    b_tab = square_tab[255 - b];
    ...
}
```

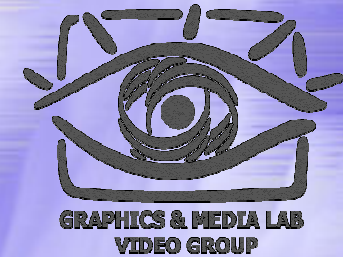

Spatial Smoother



```
for (each pixel in the specified radius){
  GetRGB (neighbour_pixel, r1, g1, b1);
  square_error = (r_tab[r1] + g_tab[g1] + b_tab[b1])
                >> Strength;
  if (square_error > 16) square_error = 16;
  square_error = 16 - square_error ;
  tot_red += r1* square_error;
  tot_green += g1* square_error;
  tot_blue += b1* square_error;
  count += square_error;
}
destination_pixel = RGB (tot_red/count,
                        tot_green / count,
                        tot_blue / count);
}
```

Параметры:
Diameter
Strength

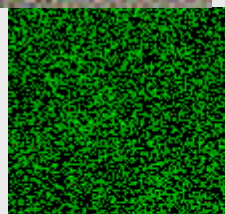
Spatial Smoother: Results



Source image



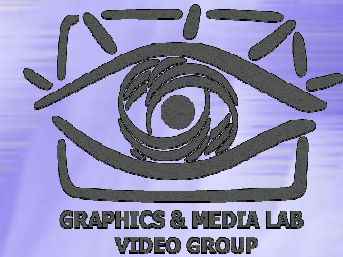
Processed image



1-st image: X:\Report\input.bmp
2-nd image: X:\Report\spatial smoother.bmp
MSE: 0,363 NMSE: 0,012 Max Diff RGB: 4,6
SNR: 18,981 PSNR: 44,389 Max Diff LUV: 3,4
Black: 50,1% Green: 49,7% Red: 0,1%

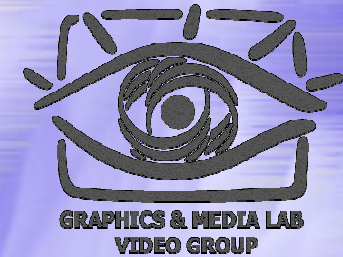
Comparison of the source and processed images using LUV Metric

2-D Spatial Noise Filter (G. Наан)



Это усредняющий по соседям фильтром с двумя порогами, рекурсивный по-вертикали (*он создавался для телевизором с небольшим объемом памяти, а элементы вертикальной задержки являются наиболее дорогими с этой точки зрения*).

2-D Spatial Noise Filter



Результирующее значение пикселя определяется по следующей формуле:

$$F_F(\underline{x}, t) = G(\underline{x}, t) \cdot \left(\sum_{\underline{n} \in N_1} K_1(\underline{x}, \underline{n}, t) \cdot F(\underline{x} + \underline{n}, t) + \sum_{\underline{n} \in N_2} K_2(\underline{x}, \underline{n}, t) \cdot F_F(\underline{x} + \underline{n}, t) \right)$$

$$K_1(\underline{x}, \underline{n}, t) = f(|F(\underline{x}, t) - F(\underline{x} + \underline{n}, t)|)$$

$$K_2(\underline{x}, \underline{n}, t) = f(|F(\underline{x}, t) - F_F(\underline{x} + \underline{n}, t)|)$$

- весовые коэффициенты,
где

$$f(a) = \begin{cases} 1, & a \leq Th \\ 0.25, & Th < a \leq 4Th \\ 0, & a > 4Th \end{cases}$$

- монотонно убывающая
функция
(Th – параметр, зависящий
от уровня шума)

2-D Spatial Noise Filter

$$F_F(\underline{x}, t) = G(\underline{x}, t) \cdot \left(\sum_{\underline{n} \in N_1} K_1(\underline{x}, \underline{n}, t) \cdot F(\underline{x} + \underline{n}, t) + \sum_{\underline{n} \in N_2} K_2(\underline{x}, \underline{n}, t) \cdot F_F(\underline{x} + \underline{n}, t) \right)$$

$$\frac{1}{G(\underline{x}, t)} = \sum_{\underline{n} \in N_1} K_1(\underline{x}, \underline{n}, t) + \sum_{\underline{n} \in N_2} K_2(\underline{x}, \underline{n}, t) \quad - \text{нормализующий коэффициент}$$

Пиксель смешивается с соседями, определяемыми следующими векторами (w последовательно принимает значения $-1, 0, 1, 0$):

$$N_1 = \left\{ \begin{pmatrix} -2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix} \right\}$$

$$N_2 = \left\{ \begin{pmatrix} -8+w \\ -1 \end{pmatrix}, \begin{pmatrix} -4+w \\ -1 \end{pmatrix}, \begin{pmatrix} w \\ -1 \end{pmatrix}, \begin{pmatrix} 4+w \\ -1 \end{pmatrix}, \begin{pmatrix} 8+w \\ -1 \end{pmatrix} \right\}$$

2-D Spatial Noise Filter

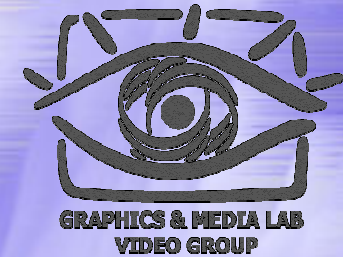
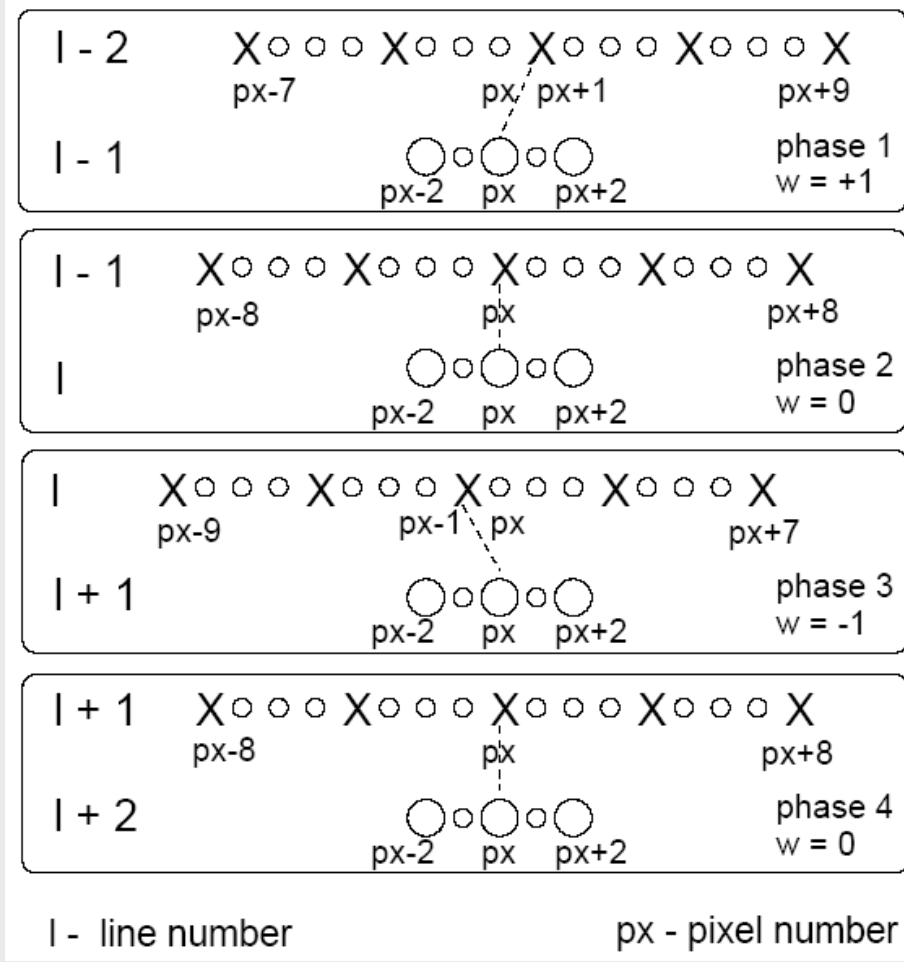
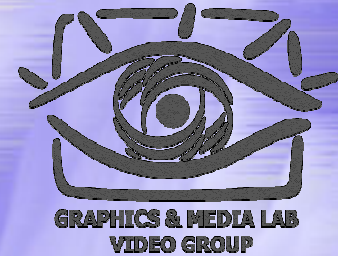


Иллюстрация работы фильтра на 4-х последовательных строках.

Вектор N_2 изменяется от строки к строке, таким образом фильтр работает циклически.



2-D Spatial Noise Filter



Особенность данного фильтра:

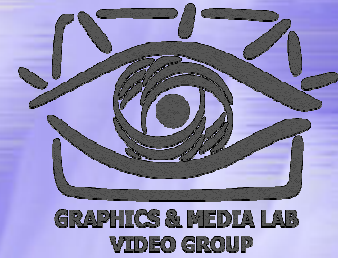
- ◆ В большинстве других пространственных фильтрах пиксели смешиваются со своими непосредственными соседями и в симметричном окне.

Аргументация: при такой обработке шумовые различия между пикселями в окне сглаживаются в наибольшей степени.

- ◆ В данном фильтре смешиваются пиксели, находящиеся на некотором расстоянии друг от друга.

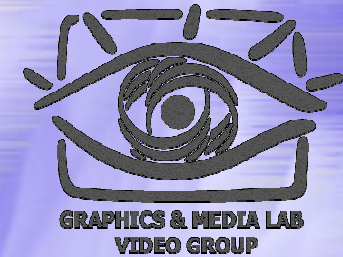
Аргументация: таким способом удастся подавить низкочастотный шум, который более заметен, чем высокочастотный, и практически не затронуть наименее заметные высокие диагональные частоты в 2D спектре благодаря периодической смене смешиваемых пикселей.

Методы пространственного шумоподавления



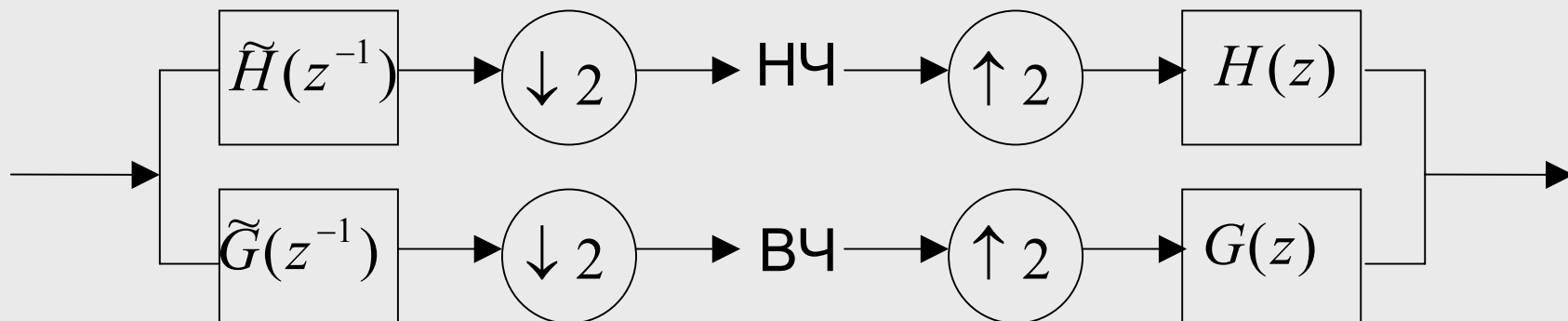
- ◆ Линейное усреднение пикселов по соседям
- ◆ Вейвлет-преобразование
- ◆ Медианная фильтрация
- ◆ Гауссовское размытие
- ◆ Математическая морфология

Вейвлеты

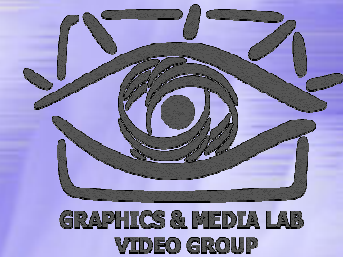


Дискретное вейвлет-преобразование (DWT):

$$\begin{array}{cccc} \tilde{h} & \tilde{g} & h & g \\ \tilde{H}(z) & \tilde{G}(z) & H(z) & G(z) \end{array}$$



DWT (на примере фильтра Хаара)



$$\tilde{\mathbf{h}} = [1/2, 1/2]$$

$$\tilde{\mathbf{g}} = [-1/2, 1/2]$$

$$\mathbf{h} = [1, 1]$$

$$\mathbf{g} = [1, -1]$$

<u>Resolution</u>	<u>Averages</u>	<u>Detail coefficients</u>
3	[2, 2, 0, 2, 3, 5, 4, 4]	
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[1.5, 4]	[0.5, 0]
0	[2.75]	[-1.25]

Haar wavelet decomposition: [2.75, -1.25, 0.5, 0, 0, -1, -1, 0]

2D DWT

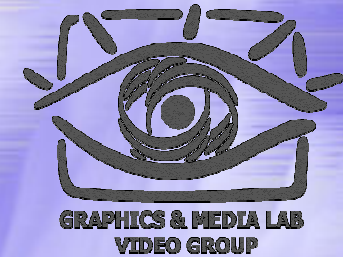


2D DWT:

DWT применяется сначала к строкам, потом к столбцам изображения.

LL	HL
LH	HH

DWT и шумоподавление



Информация о шуме содержится в ВЧ-составляющих (вейвлет-коэффициентах) DWT.

Если на изображении $\mathbf{f} = [\mathbf{f}_1, \dots, \mathbf{f}_n]$ присутствует белый шум с нулевым математическим ожиданием и дисперсией σ^2 , то в силу линейности DWT для вейвлет-коэффициентов будет выполняться:

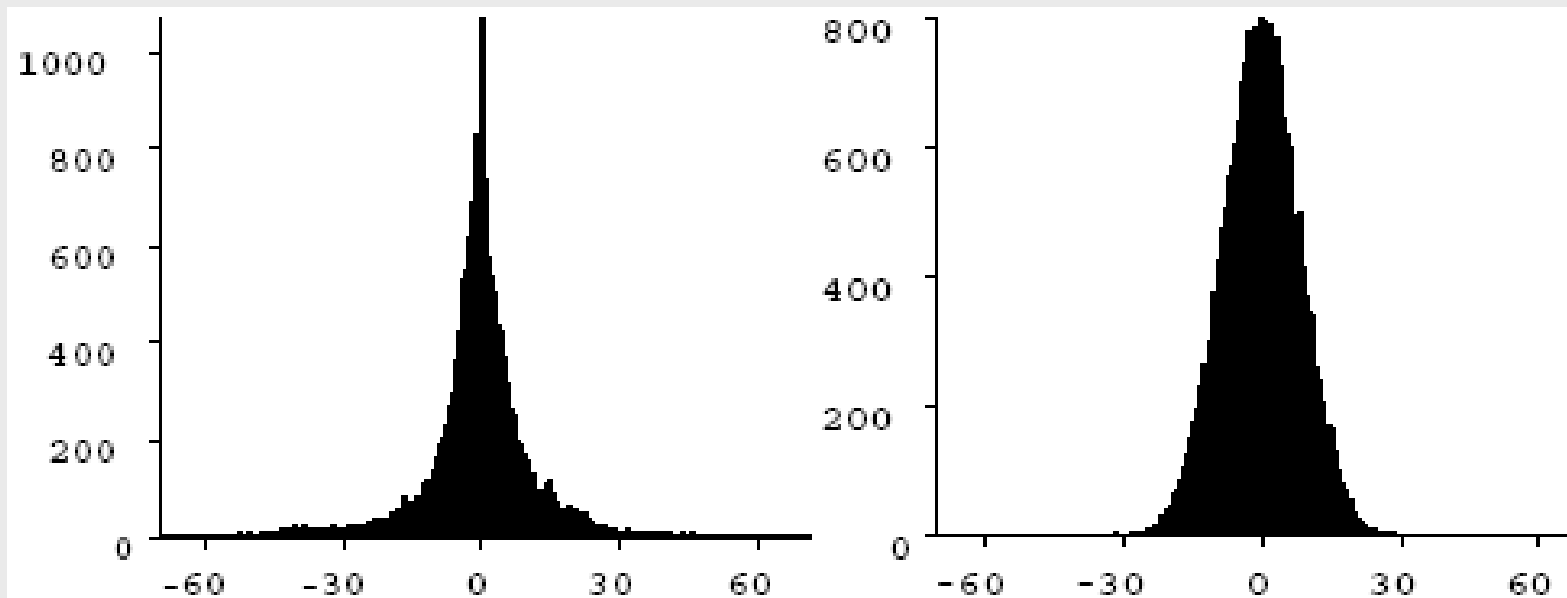
$$w_i = y_i + \epsilon_i, \quad i = 1, \dots, n$$

где y_i - вейвлет-коэффициенты без шума,
а $\epsilon_i \sim N(0, \sigma^2)$ - шум

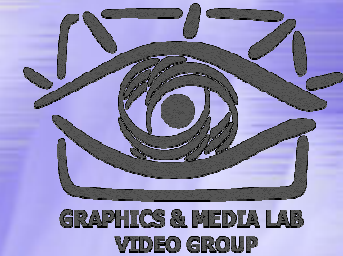
DWT и шумоподавление



Слева – гистограмма вейвлет-коэффициентов у чистого изображения, справа – у изображения с шумом.

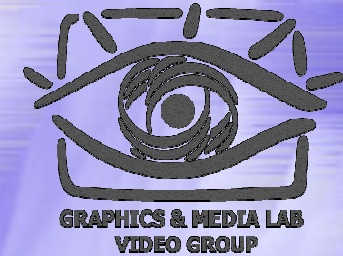


DWT и шумоподавление



- ◆ При шумоподавлении обычно используется вейвлет-преобразование без прореживания, то есть примененное к каждому пикселу.
- ◆ Шумоподавление выполняется путем уменьшения значений вейвлет-коэффициентов в зависимости от уровня шума и вероятности того, что данный коэффициент представляет собой шум.
- ◆ Проблема – как отличить шум от деталей изображения.

DWT и шумоподавление



Можно выделить два основных подхода:

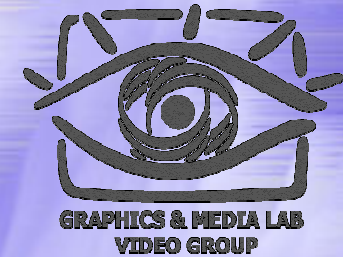
◆ **Thresholding**

- hard-thresholding :
$$\tilde{w}_i = \begin{cases} 0, & |w_i| \leq T \\ w_i, & |w_i| > T \end{cases}$$

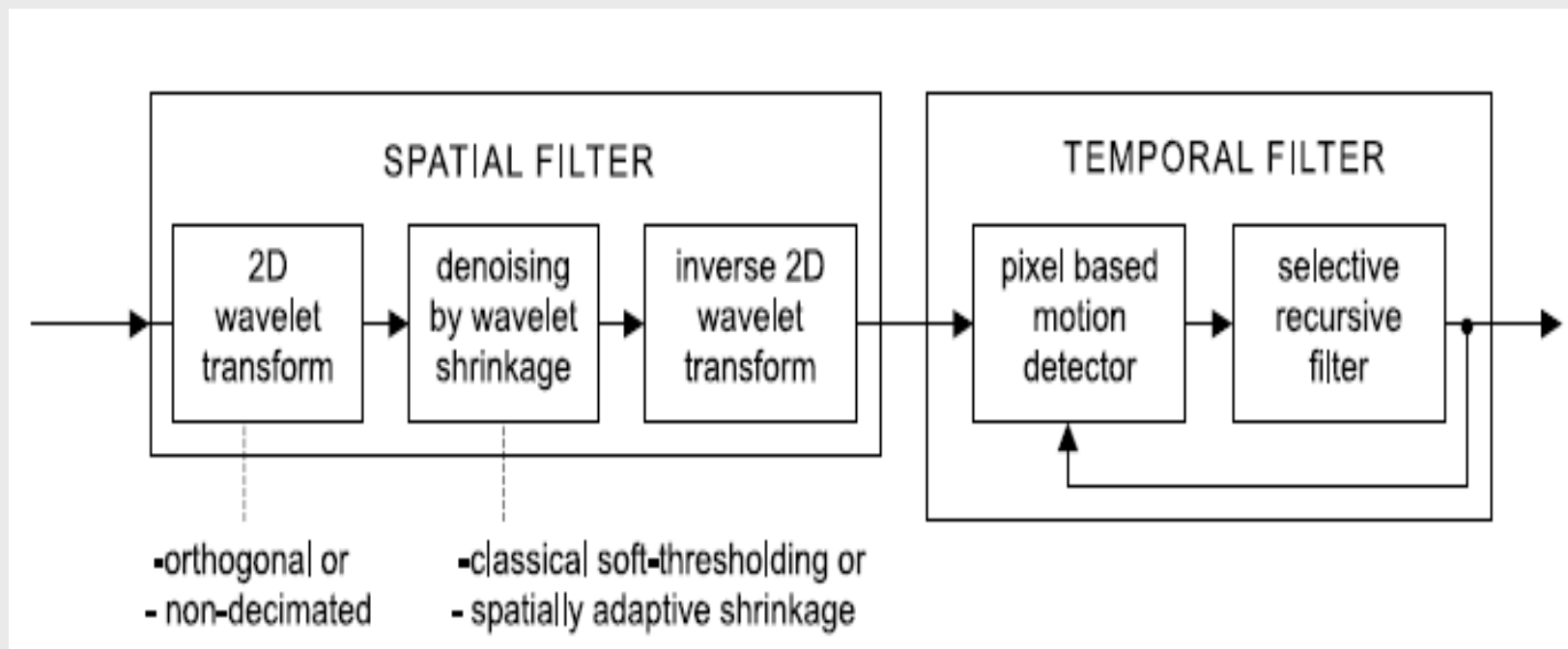
- soft-thresholding:
$$\tilde{w}_i = \begin{cases} 0, & |w_i| \leq T \\ \text{sgn}(w_i)(|w_i| - T), & |w_i| > T \end{cases}$$

◆ **Spatially adaptive shrinkage**

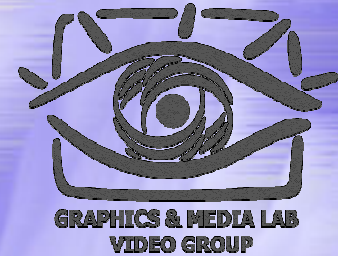
Общая схема шумоподавления с использованием вейвлетов



Обычно объединяют пространственный вейвлет-фильтр с некоторым временным фильтром.



Combined Wavelet Domain and Temporal Video Denoising



Состоит из двух фильтров, выполняющихся последовательно:

- ◆ Spatially Adaptive Noise Filter in the Wavelet Domain
- ◆ Selective Recursive Temporal Filter in the Signal Domain

Рассмотрим подробнее адаптивный алгоритм уменьшения вейвлет-коэффициентов, представленный в пространственном фильтре.

Wavelet shrinkage estimator



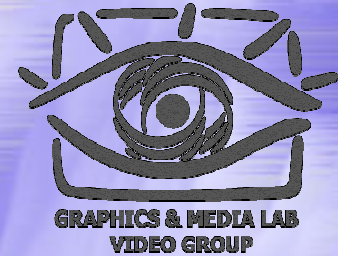
Вероятность того, что данный вейвлет-коэффициент представляет собой шум, оценивается исходя из следующих данных:

- значения самого вейвлет-коэффициента w_l
- среднего значения вейвлет-коэффициентов в небольшом окне $\delta(l)$ вокруг этого коэффициента:

$$z_l = \sum_{k \in \delta(l)} |w_k|$$

- глобального распределения коэффициентов на данном уровне

Wavelet shrinkage estimator



Пусть гипотеза H_1 означает, что шум отсутствует ($|y| \geq \tau$), а H_0 – что присутствует ($|y| < \tau$).

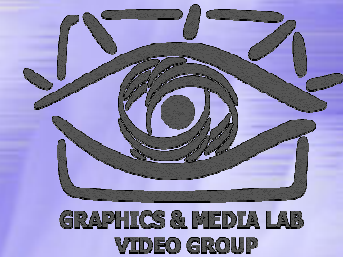
Тогда новые вейвлет коэффициенты будут определяться следующей формулой на основе 3-х пунктов, перечисленных выше:

$$\hat{y}_l = \frac{\rho \xi_l \eta_l}{1 + \rho \xi_l \eta_l} w_l,$$

where

$$\rho = \frac{P(H_1)}{P(H_0)}, \quad \xi_l = \frac{p(w_l|H_1)}{p(w_l|H_0)} \quad \text{and} \quad \eta_l = \frac{p(z_l|H_1)}{p(z_l|H_0)}$$

Wavelet shrinkage estimator



$$\hat{y}_l = \frac{\rho \xi_l \eta_l}{1 + \rho \xi_l \eta_l} w_l,$$

where

$$\rho = \frac{P(H_1)}{P(H_0)}, \quad \xi_l = \frac{p(w_l|H_1)}{p(w_l|H_0)} \quad \text{and} \quad \eta_l = \frac{p(z_l|H_1)}{p(z_l|H_0)}$$

$p(x|H_0)$ – вероятностная функция того, что коэффициент x – шум; $p(x|H_1)$ – вероятностная функция того, что коэффициент x – не является шумом. Они определяются следующим образом:

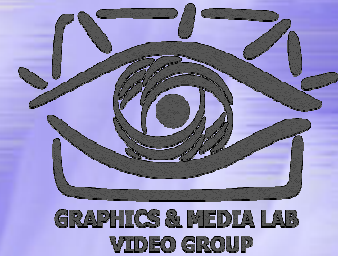
$$p(y|H_0) = \begin{cases} A_0 \exp(-\lambda|y|^\nu) & \text{if } y \leq T, \\ 0 & \text{if } y > T, \end{cases}$$

$$p(y|H_1) = \begin{cases} 0 & \text{if } y \leq T, \\ A_1 \exp(-\lambda|y|^\nu) & \text{if } y > T. \end{cases}$$

$$A_0 = (\lambda/2)e^{\lambda T} / (e^{\lambda T} - 1) \quad \text{and} \quad A_1 = (\lambda/2)e^{\lambda T}$$

T – некий порог, зависящий от уровня шума

Wavelet shrinkage estimator



Первый коэффициент - ρ - зависит от глобального распределения коэффициентов:

$$\rho = \frac{P(H_1)}{P(H_0)} = \frac{\exp(-\lambda T)}{1 - \exp(-\lambda T)}.$$

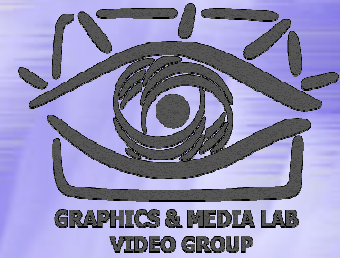
$$\lambda = [0.5(\sigma_w^2 - \sigma^2)]^{-1/2} \quad v = 1$$

σ_w – дисперсия сигнала с шумом

σ – дисперсия сигнала без шума

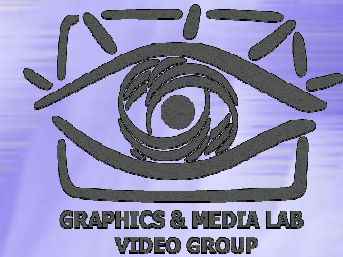
Лучшие результаты были получены при $T = \sigma$.

Методы пространственного шумоподавления



- ◆ Линейное усреднение пикселов по соседям
- ◆ Вейвлет-преобразование
- ◆ Медианная фильтрация
- ◆ Гауссовское размытие
- ◆ Математическая морфология

Медианная фильтрация

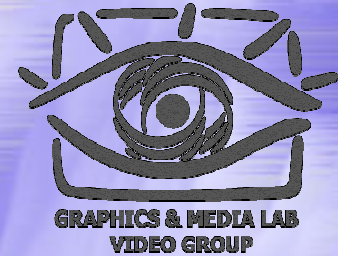


- ◆ Медианная фильтрация – это стандартный способ подавления импульсного шума.
- ◆ Для каждого пиксела вычисляется среднее значение в некотором его окружении (окне) и присваивается этому пикселу:

$$med = \operatorname{argmin}_{f_i \in W} \sum_j |f_i - f_j|$$

где W – окно размера $(2n+1) \times (2n+1)$

Vector Median Filter (VMF)



Для цветных изображений используется векторный медианный фильтр (VMF):

$$med = \arg \min_{f_i \in W} \sum_{j=0}^{N-1} d(F_i, F_j)$$

где d – произвольная метрика
(Euclidean or city-block)

Однако чистый медианный фильтр размывает края, поэтому на практике практически не используется.

Fast Modified Vector Median Filter (FMVMF)

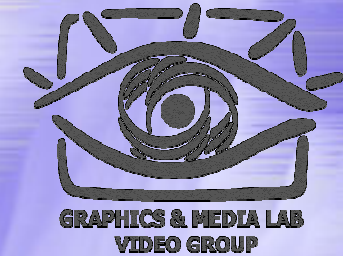


Его модификация – быстрый векторный медианный фильтр (FMVMF).

В FMVMF используется окно с 4-мя соседями:
(F_0 – изменяемый пиксел)

	F_1	
F_4	F_0	F_2
	F_3	

FMVMF

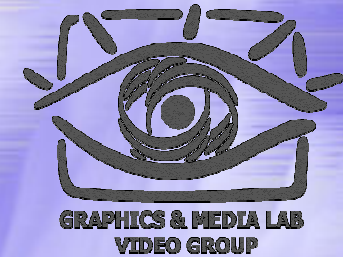


Рассматриваемый центральный пиксел F_0 заменяется на F_k , где:

$$k = \begin{cases} 0, & \text{if } -\beta + \sum_{j=1}^{N-1} d(F_0, F_j) \leq \min_{F_i \in W \setminus F_0} \sum_{j=1}^{N-1} d(F_i, F_j) \\ i_{opt} := \arg \min_{F_i \in W \setminus F_0} \sum_{j=1}^{N-1} d(F_i, F_j), & \\ \text{if } \min_{F_i \in W \setminus F_0} \sum_{j=1}^{N-1} d(F_i, F_j) < -\beta + \sum_{j=1}^{N-1} d(F_0, F_j) & \end{cases}$$

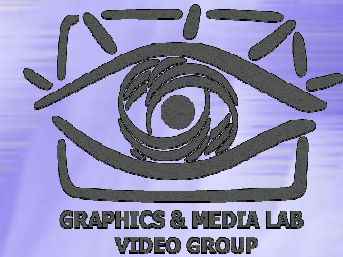
β - параметр (экспериментально равен 0.75, если цветовые компоненты нормализованы в интервале $[0, 1]$)

Primum Non Nocere Vector Median Filter (PNN-VMF)



- ◆ За основу взят FMVMF.
- ◆ Обход ведется по-горизонтали слева направо сверху вниз и в качестве F_1 и F_4 берутся уже ранее обработанные значения (предположительно, без шума).
- ◆ В FMVMF F_0 изменяется, если условие выполнялось хотя бы для одного из соседей, а в PNN-VMF - только если оно выполнено для ВСЕХ его соседей.

Two-pass Median-like Filter for impulse noise removal



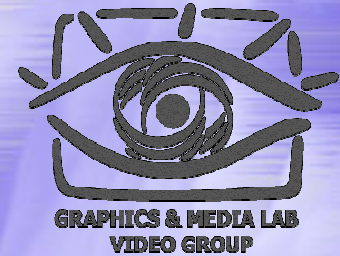
- ◆ Первый шаг – PNN-VMF.
- ◆ На втором шаге выполняется одна из модификаций PNN-VMF - N2, N3, N4 – на основе того, сколько пикселей было изменено на первом шаге.

condition	triggered 2 nd pass action
$FMP \leq T_1$	no action, terminate
$(FMP > T_1) \wedge (FMP \leq T_2)$	N4
$(FMP > T_2) \wedge (FMP \leq T_3)$	N3
$FMP > T_3$	N2

N_i – это PNN-VMF, у которого для изменения центрального пиксела достаточно выполнения условия для N_i соседей из 4-х.

Для T_1 , T_2 и T_3 экспериментально были получены значения 0.04, 0.08 и 0.13 соответственно.

Методы пространственного шумоподавления



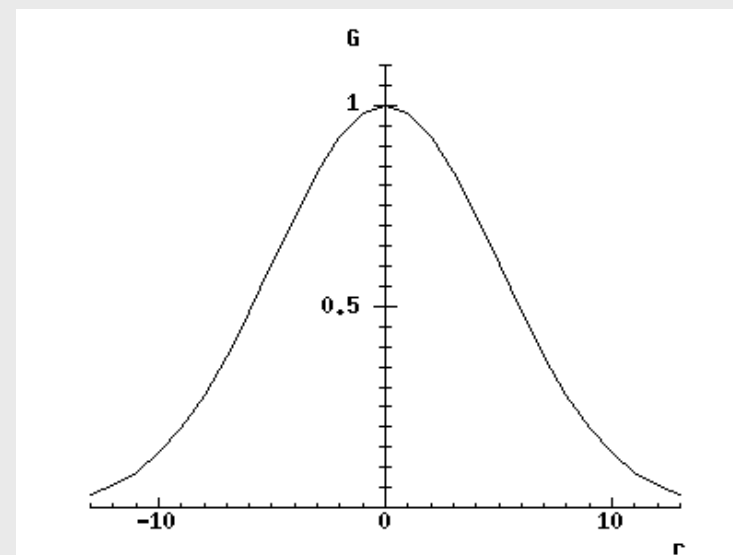
- ◆ Линейное усреднение пикселов по соседям
- ◆ Вейвлет-преобразование
- ◆ Медианная фильтрация
- ◆ **Гауссовское размытие**
- ◆ Математическая морфология

Гауссовское размытие

Это свертка по функции:

$$g(x, y) = A e^{-\frac{x^2 + y^2}{\sigma^2}}$$

Параметр σ задает степень размытия.

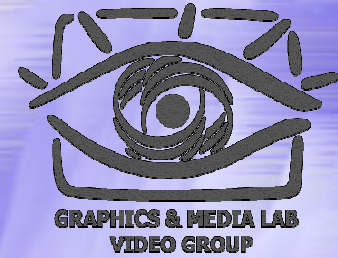


$$I'(i, j) = \sum_{x=-n}^n \sum_{y=-m}^m I(i-x)(j-y) \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{d^2}{2\sigma^2}}$$

$$d = \sqrt{x^2 + y^2}$$

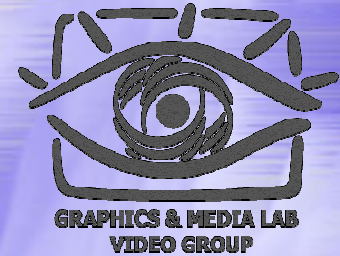
$$\begin{bmatrix} \frac{1}{4} & 1 & \frac{1}{4} \\ \frac{1}{4} & 5 & \frac{1}{4} \\ \frac{1}{4} & 1 & \frac{1}{4} \end{bmatrix}$$

Методы пространственного шумоподавления



- ◆ Линейное усреднение пикселов по соседям
- ◆ Вейвлет-преобразование
- ◆ Медианная фильтрация
- ◆ Гауссовское размытие
- ◆ Математическая морфология

Математическая морфология



Морфологические фильтры:

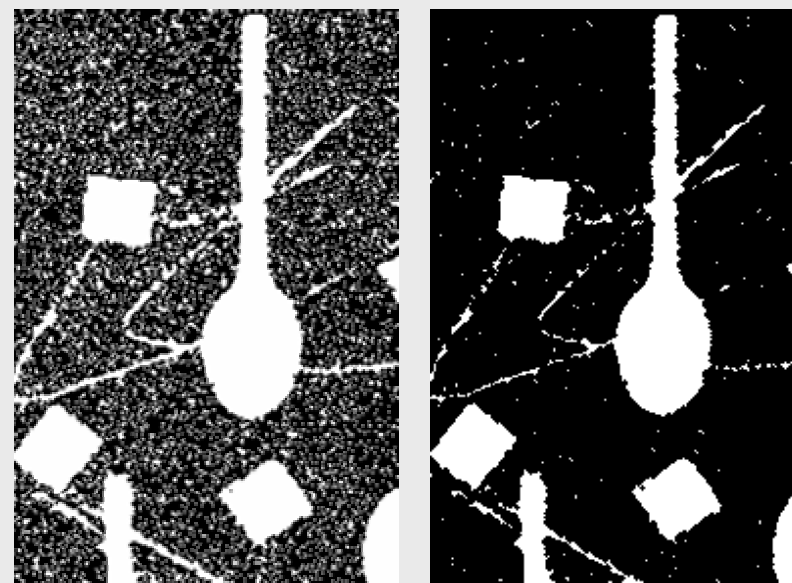
- ◆ Сужение (Erosion)
- ◆ Расширение (Dilation)
- ◆ Закрытие (Closing)
- ◆ Раскрытие (Opening)

Сужение, Расширение

Рассмотрим на примере бинарного изображения:

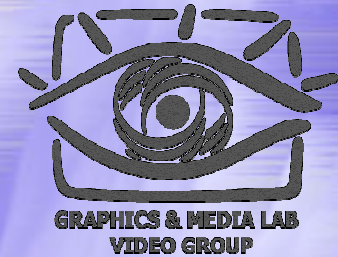
Сужение: если хотя бы 1 сосед черный, ставим черную точку, иначе – белую.

Расширение: если все соседи черные, ставим черную точку, иначе – белую.



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ - маска (структурный элемент)}$$

Раскрытие, Заккрытие



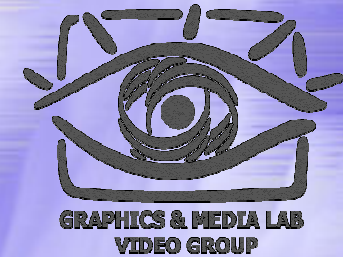
Раскрытие: сначала сужение, потом расширение.

Убирает выступы на границах объектов

Заккрытие: сначала расширение, потом сужение.

Заполняет отверстия внутри и на границах

Morphological Image Cleaning Algorithm (MIC)



Пусть I – изображение, I_Z - открытие этого изображения с помощью структурного элемента Z , а I^Z – его закрытие с использованием того же структурного элемента Z .

Определим фильтр $OCCO(I; Z)$ следующим образом:

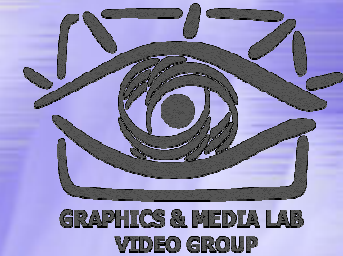
$$OCCO(I; Z) = \frac{1}{2}(I_Z)^Z + \frac{1}{2}(I^Z)_Z$$

Общая схема алгоритма:

Пусть I – черно-белое изображение с шумом, а $S = OCCO(I; Z)$ – сглаженное изображение без шума.

Тогда изображение, равное их разнице, $D = I - S$ будет содержать весь шум исходного изображения I . Так же оно будет содержать мелкие все детали исходного изображения, размер которых меньше структурного элемента.

MIC



Если шум в I изменялся в меньших пределах, чем мелкие детали, то в D большая амплитуда будет у деталей, чем у шума.

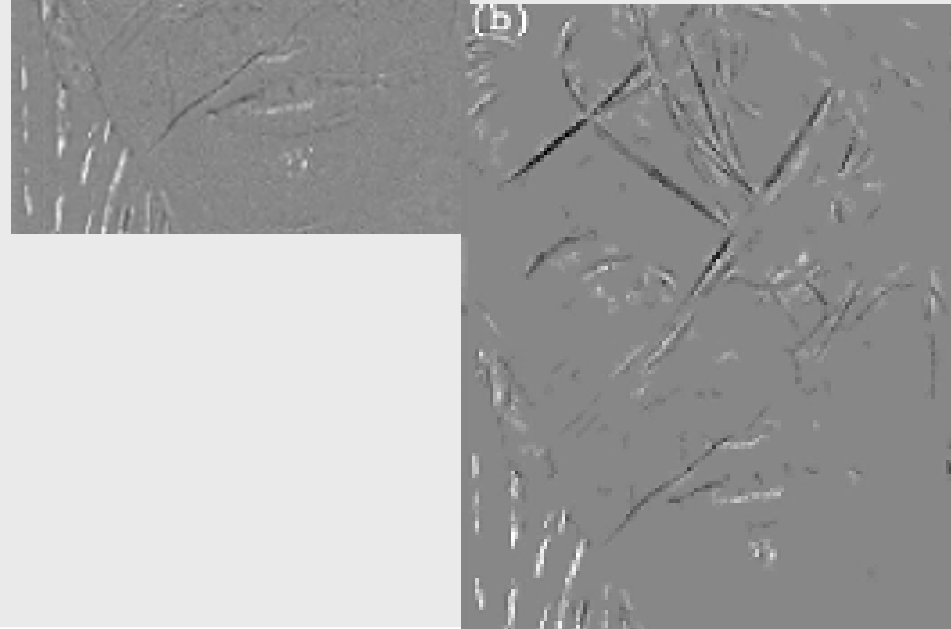
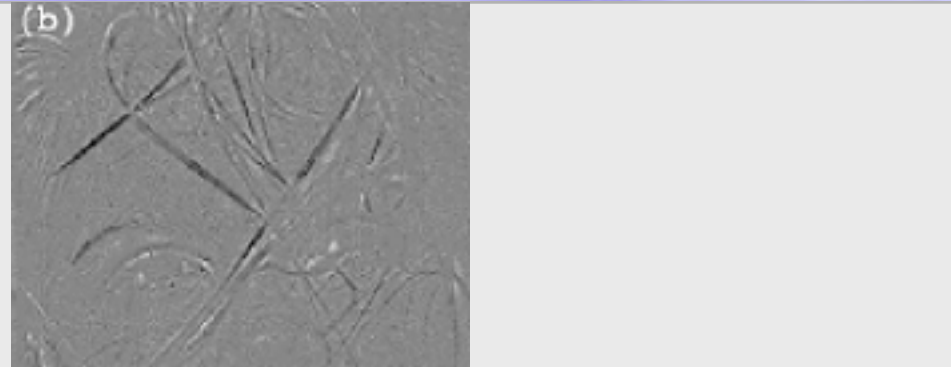
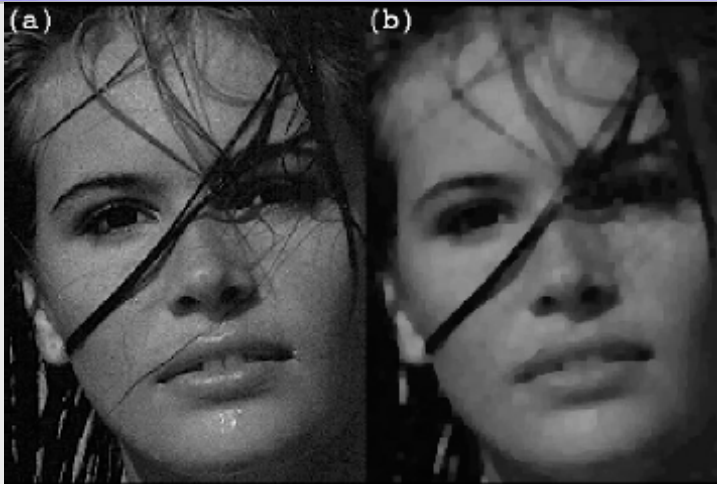
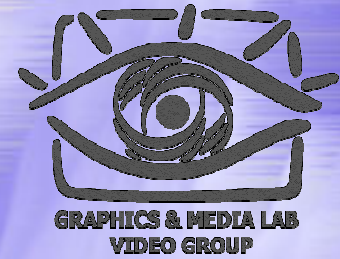
Таким образом, маска D' , полученная следующим образом:

$$D'(x,y) = D(x,y), \text{ если } |D(x,y)| > T, \text{ иначе } D'(x,y) = 0$$

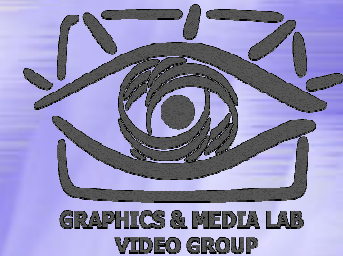
будет содержать все детали изображения.

Далее по полученной D' маске восстанавливаются мелкие детали, края и тонкие линии в сглаженном изображении S .

MIC: Results

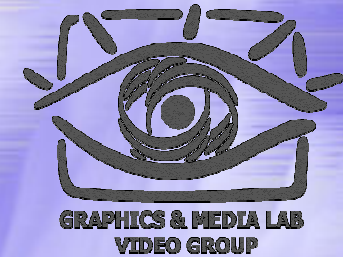


Denoising: Содержание



- ◆ **Виды шума и их источники**
- ◆ **Методы подавления шума**
 - Методы, работающие в пространственной области
 - **Методы, работающие во временной области**
(как использующие, так и не использующие компенсацию движения)
 - Методы удаления вертикальных царапин со старых киноплёнок
- ◆ **MSU Noise Remover**
- ◆ **Методы оценки качества шумоподавления**

Методы временного шумоподавления

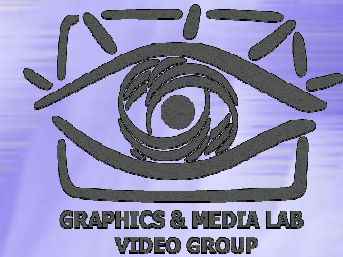


Общая схема временного шумоподавления:

$$F_F(\mathbf{x}, t) = k F(\mathbf{x}, t) + (1 - k) F_F(\mathbf{x}, t - T)$$

где $F(\mathbf{x}, t)$ – исходное значение пиксела на текущем фрейме, а $F_F(\mathbf{x}, t)$ - результирующее

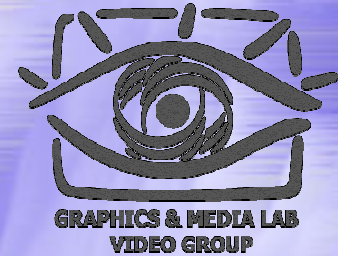
Методы временного шумоподавления



Делятся на алгоритмы:

- ◆ не использующие Motion Estimation
(проблемы – motion blur и ghosting)
- ◆ использующие Motion Estimation
 - простое детектирование движения: пиксели в движущихся блоках не изменяются
 - построение скомпенсированного кадра и смешивание текущего с ним (МЕ должно быть выполнено качественно)

Алгоритмы, не использующие МЕ: Chroma Noise Reduction



Работает в пространстве YUV, обрабатывает только цветностные компоненты U и V.

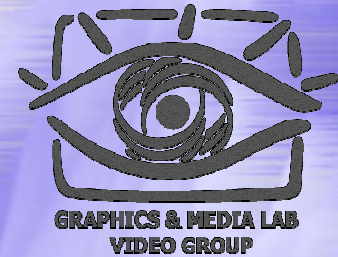
$U_{cur} =$

$$q(Y) * q(U) * U_{prev} + (1 - q(Y)) * (1 - q(U)) * U_{cur}$$

$V_{cur} =$

$$q(Y) * q(V) * V_{prev} + (1 - q(Y)) * (1 - q(V)) * V_{cur}$$

Chroma Noise Reduction



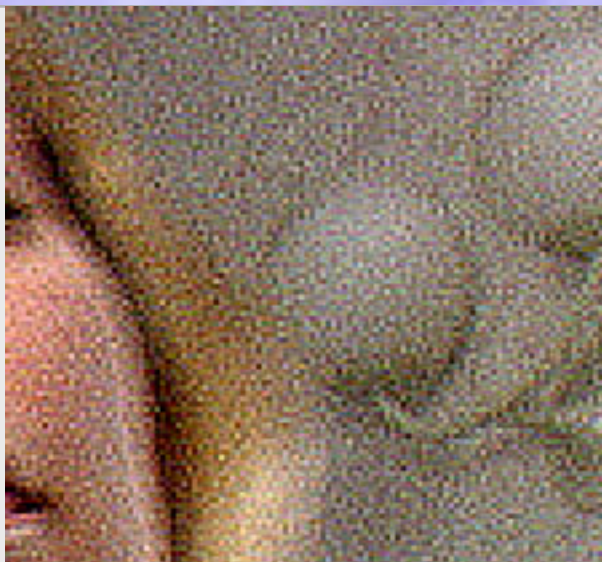
$Y_tab[i] = 0$, if $i < -Y_ymax$, $i > Y_ymax$

$Y_tab[i] = Y_xmax/2 * (1 + \cos(\pi * i / Y_ymax))$, if
 $-Y_ymax < i < Y_ymax$

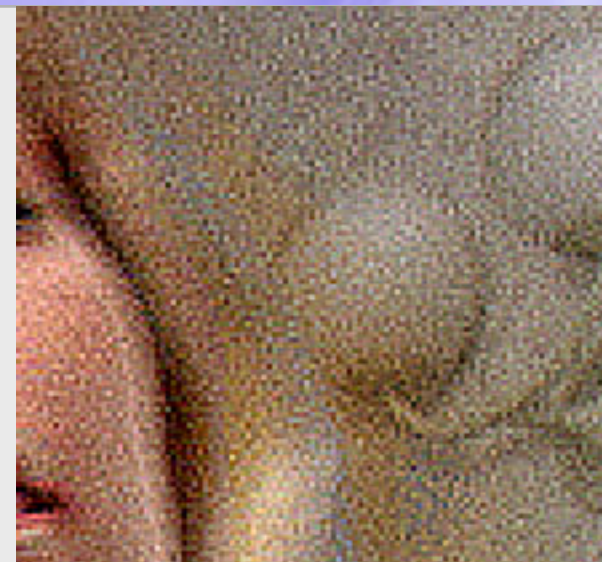
(U_tab, V_tab определяются аналогично)

```
for (each pixel of the current video frame){
    (RGB) -> (YUV)
    Y_diff = Y - Y_prev;
    U_diff = U - U_prev;
    V_diff = V - V_prev;
    U_new = (Y_tab[Y_diff]*U_tab[U_diff] * U_prev +
            (1 - Y_tab[Y_diff])*(1 - U_tab[U_diff]) * U);
    V_new = (Y_tab[Y_diff]*V_tab[V_diff] * V_prev +
            (1 - Y_tab[Y_diff])*(1 - V_tab[V_diff]) * V);
    (YUV) -> (RGB)
}
```

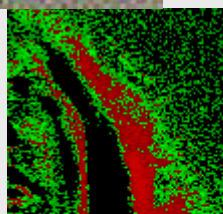

Chroma Noise Reduction: Results



Source image



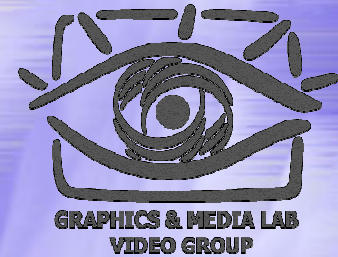
Processed image



1-st image: X:\Report\input.bmp
2-nd image: X:\Report\CNR.bmp
MSE: 0,029 NMSE: 0,001 Max Diff RGB: 0,8
SNR: 29,834 PSNR: 55,242 Max Diff LUV: 8
Black: 44,7% Green: 37,1% Red: 18%

Comparison of the source and processed images using LUV Metric

Алгоритмы, не использующие ME: Dynamic Noise Reduction

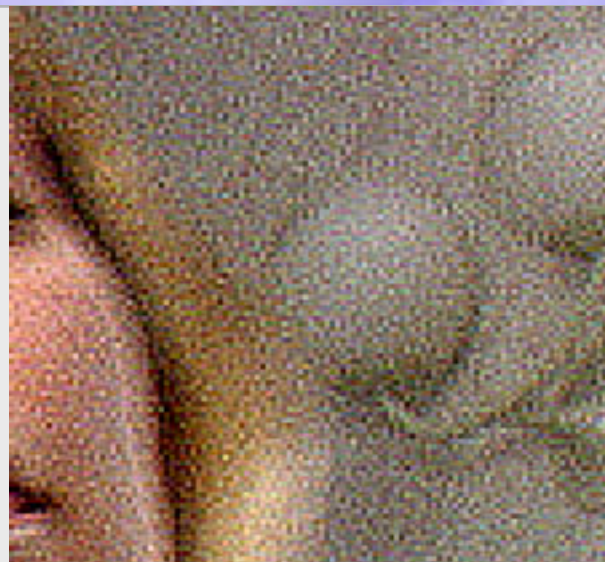


```
for (each pixel of the current video frame){
    GetRGB(source_pixel, r, g, b);
    r_diff = abs(r - r_prev);
    g_diff = abs(g - g_prev);
    b_diff = abs(b - b_prev);
    if (r_diff < Level)
        if (r_diff > (Level >> 1))
            r = 2/3*r + 1/3*r_prev;
            else r = r_prev;
    ... (the same for each color component)
    destination_pixel = RGB (r, g, b);
}
```

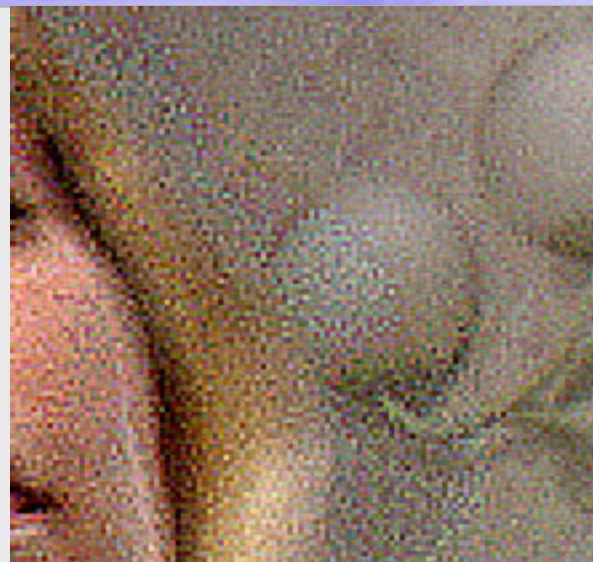
Параметр:

Level

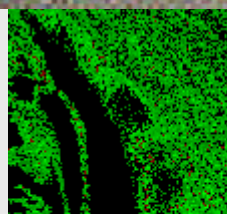
Dynamic Noise Reduction: Results



Source image



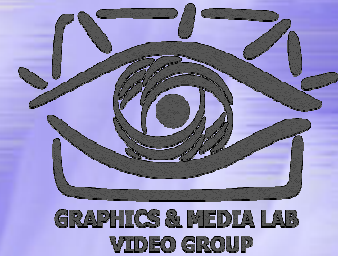
Processed image



1-st image: X:\Report\input.bmp
2-nd image: X:\Report\DNR.bmp
MSE: 0,564 NMSE: 0,019 Max Diff RGB: 5
SNR: 17,074 PSNR: 42,483 Max Diff LUV: 4,3
Black: 46,2% Green: 52,4% Red: 1,2%

Comparison of the source and processed images using LUV Metric

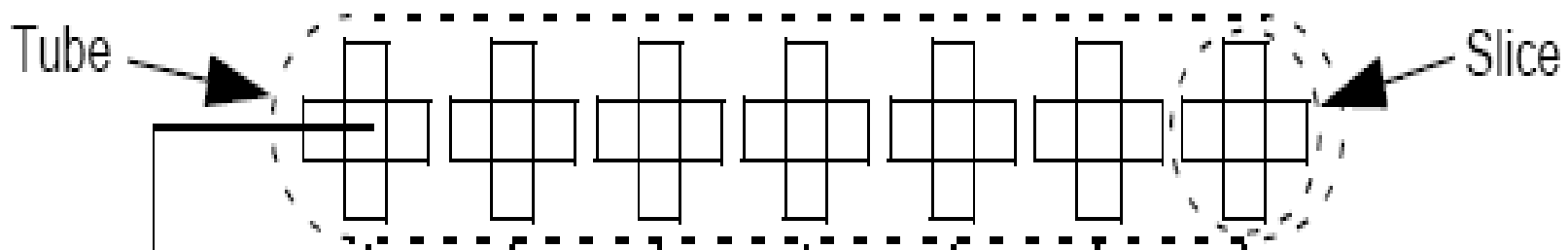
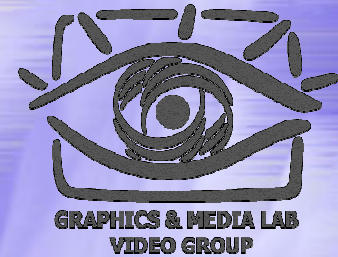
Spatio-Temporal Noise Reduction ASIC for Real-Time Video Processing



Этот метод уже учитывает движение и не изменяет изображение в тех местах, где оно было.

Обработка текущего кадра производится на основе 7 кадров: его самого, трех предшествующих и трех последующих.

Spatio-Temporal Noise Reduction ASIC for Real-Time Video Processing



Центральный пиксель и 4 его соседа на одном кадре образуют сечение, а соответствующие сечения на всех 7ми кадрах - цилиндр.

$$\text{Frame: } F(n) = \{R(n), G(n), B(n)\}$$

$$\text{Pixel: } P(i, j, n) = \{R(i, j, n), G(i, j, n), B(i, j, n)\} \in F(n)$$

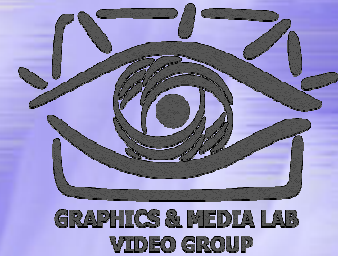
$$\text{Slice: } S(i_0, j_0, n_0) \subset F(n_0)$$

$$\text{Tube: } T(i_0, j_0, n_0) = \{S(i_0, j_0, n) \mid (n_0 - d) \leq n \leq (n_0 + d)\}$$

$$\text{Number of frames: } 2 \cdot d + 1$$

The centre frame is surrounded by d frames in time.

Spatio-Temporal Noise Reduction ASIC for Real-Time Video Processing



Индикатором движения является функция:

$$I(n, n-1) = \begin{cases} 1 & \text{if } ((\Delta Y > T_1) \vee (\Delta B > T_2)) \\ 0 & \text{otherwise} \end{cases}$$

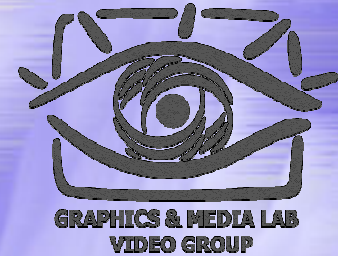
где ΔY и ΔB - модули разности соответственно яркости и цветности между двумя соседними сечениями:

$$\Delta Y = |\bar{Y}(n) - \bar{Y}(n-1)|$$

$$\Delta B = |B(i_0, j_0, n) - B(i_0, j_0, n-1)|$$

($\bar{Y}(n)$ - среднее значение яркости на n -м сечении)

Spatio-Temporal Noise Reduction ASIC for Real-Time Video Processing



Пусть n_0 – текущий обрабатываемый кадр, всего кадров $2d+1$.

1. Считаем $\bar{Y}(n)$ для $(n_0 - d) \leq n \leq (n_0 + d)$.
2. Определяем правый временной промежуток, на котором не было движения δ_R :

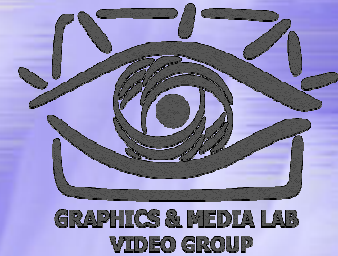
$k := d$

`while (I(k, k-1) = 1 && (k ≠ 1)) k := k-1`

`if ((k ≠ 1) || I(k, k-1) = 0) then $\delta_R = k$
else $\delta_R = 0$`

3. Аналогично определяем левый временной промежуток, на котором не было движения - δ_L .
4. Получили временной интервал $[n_0 - \delta_L, n_0 + \delta_R]$, который переопределяем следующим образом:
`if ($\delta_R = \delta_L = 0$) then $\delta_R = \delta_L = 1$`

Spatio-Temporal Noise Reduction ASIC for Real-Time Video Processing



5. Определяем функцию:

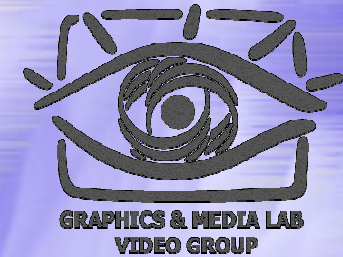
$$Y(i_0, j_0, n) = \begin{cases} Y(i_0, j_0, n) & \text{if } n \in [n_0 - \delta_L, n_0 + \delta_R] \\ \text{Alt}\{y_{\max} y_{\min}\} & \text{otherwise} \end{cases}$$

6. Считаем результирующее значение пикселя как результат действия медианного фильтра на эту функцию:

$$Y(i_0, j_0, \tilde{n}) = \text{Median}\{Y(i_0, j_0, n) \mid n = n_0 - d, \dots, n_0 + d\}$$

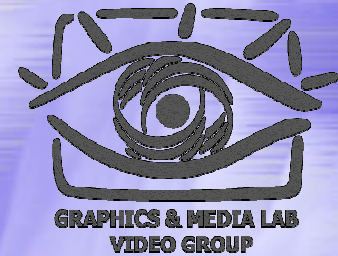
$$\tilde{P}(i_0, j_0, n_0) = P(i_0, j_0, \tilde{n})$$

Denoising: Содержание



- ◆ **Виды шума и их источники**
- ◆ **Методы подавления шума**
 - Методы, работающие в пространственной области
 - Методы, работающие во временной области (как использующие, так и не использующие компенсацию движения)
 - **Методы удаления вертикальных царапин со старых киноплёнок**
- ◆ **MSU Noise Remover**
- ◆ **Методы оценки качества шумоподавления**

Методы удаления царапин



Царапины появляются при повреждении эмульсии на пленке.

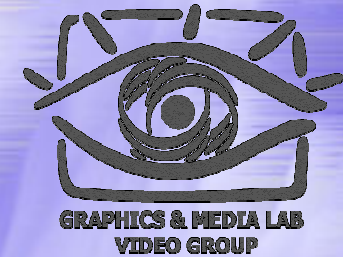
Свойства царапин:

- ◆ Располагаются обычно вертикально
- ◆ Содержатся в нескольких подряд идущих кадрах
- ◆ Имеют примерно пять пикселей в ширину

Проблема: как отличить их от вертикальных деталей на изображении?

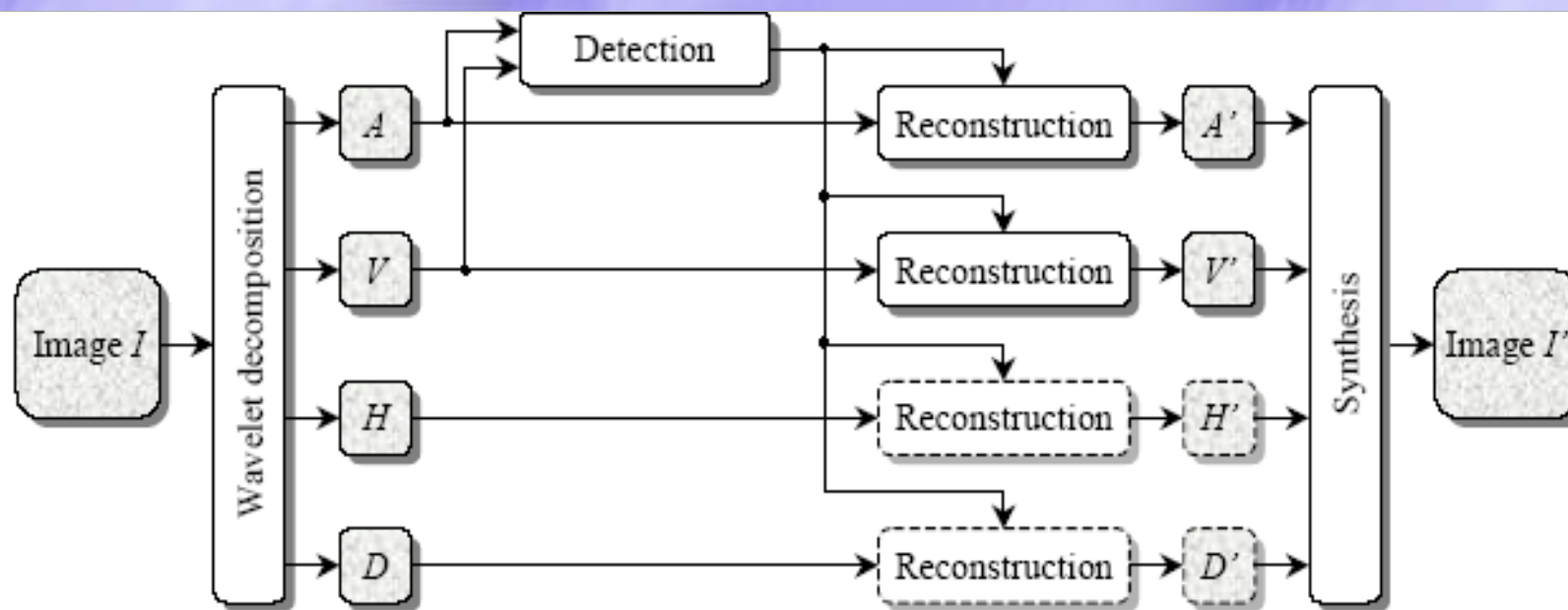


Удаление царапин с помощью DWT



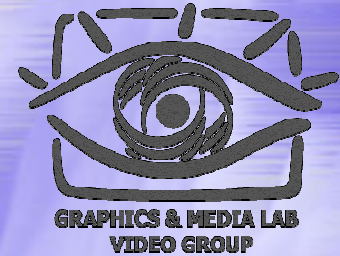
- ◆ Один из методов удаления царапин заключается в анализе коэффициентов, получающихся в результате DWT.
- ◆ Его можно применять как в черно-белых, так и цветных изображениях (отдельно применяя к каждой компоненте цвета).

Общая схема



Детектирование царапин производится на основе НЧ-составляющей и одной из ВЧ-составляющих, отвечающей за вертикальные детали. Восстановление остальных ВЧ-составляющих необязательно.

Алгоритм обнаружения царапин

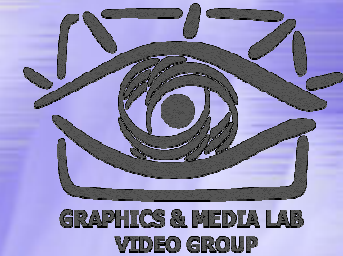


Первый шаг:

Считаются суммы значений коэффициентов по столбцам ВЧ-составляющей V и затем по некоторому порогу из этих сумм выбираются локальные максимумы.

Это сделано того, чтобы далее рассматривать не все изображение, а отдельные области, на которых предположительно может находиться царапина.

Алгоритм обнаружения царапин

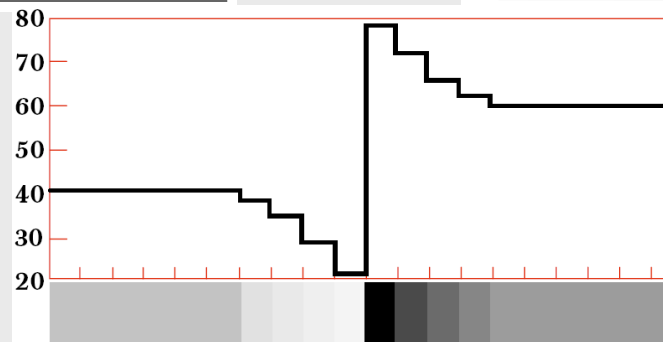
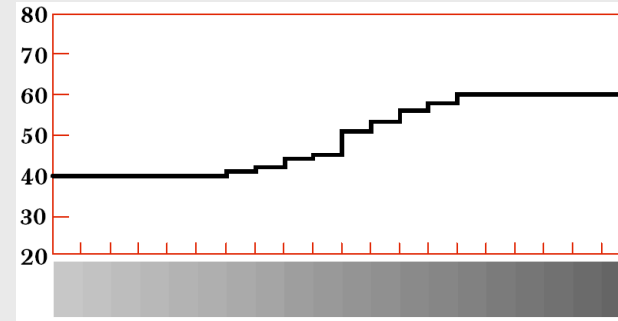
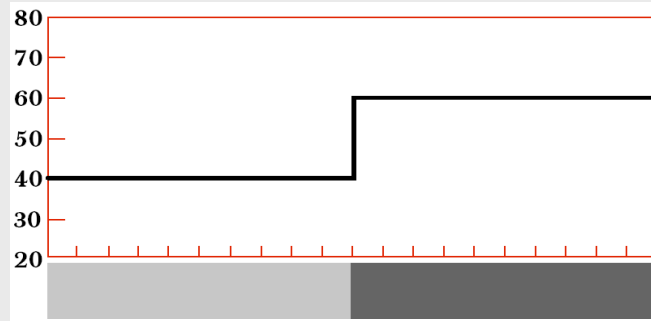
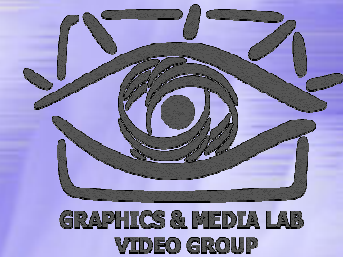


Второй шаг:

1. К НЧ-составляющей A применяется вертикальный unsharp masking (окно фильтра должно быть больше ширины царапины).
2. Из полученного A_u получают бинарное изображение B следующим образом:
$$\text{if } A_u(x,y) > 0 \quad B(x,y) = 1 \quad \text{else } B(x,y) = 0$$

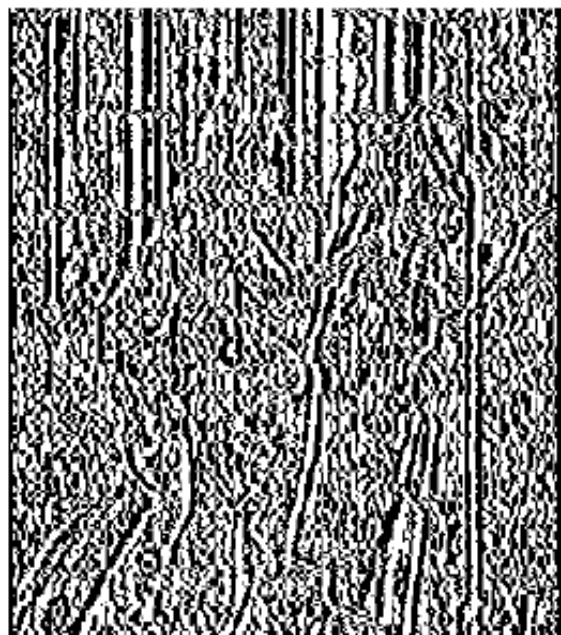
(это для темных царапин; для светлых – наоборот)
3. По столбцам B считаются суммы коэффициентов и среди них ищется глобальный максимум. Его x -координата и будет определять положение царапины.

Unsharp masking

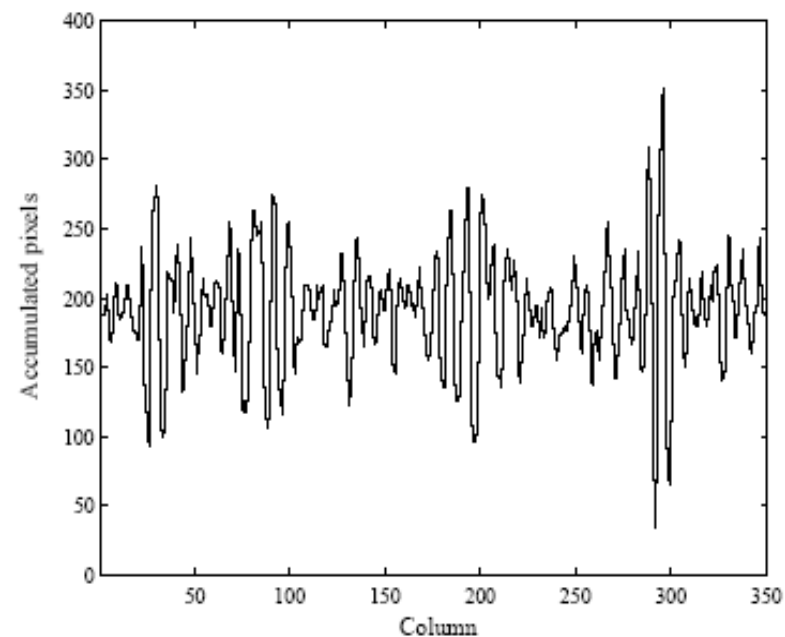


Если пиксель на исходном изображении светлее, чем на размытом, он делается светлее на значение, равное их разнице, иначе он делается темнее на то же значение.

Алгоритм обнаружения царапин



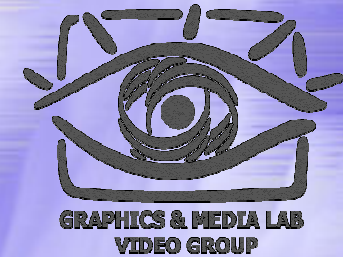
(a)



(b)

Figure 4: Unsharp masking for scratch detection: (a) binary image B of Figure 1(b), (b) projection of B with a maximum at the x-position of the scratch.

Удаление царапин



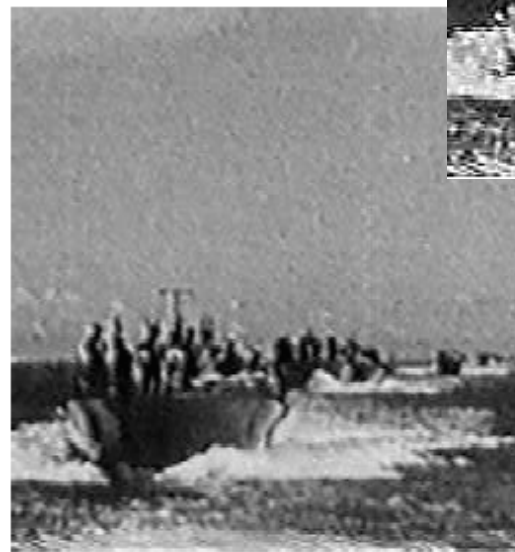
Удаление царапины производится отдельно в A и V .

Вот ряд методов для их удаления:

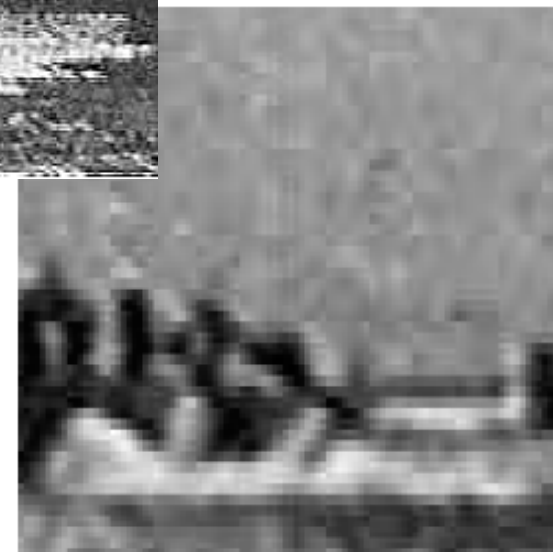
- ◆ Медианный фильтр
- ◆ Кубическая интерполяция сплайнами
- ◆ Интерполяция полиномами третьей степени

Для V лучшие результаты показал медианный фильтр.

Results



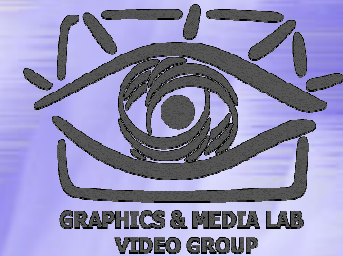
(a)



(b)

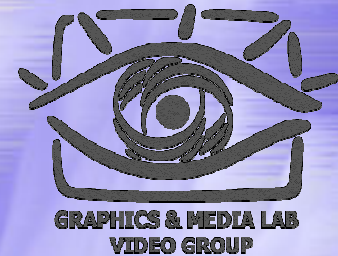
Figure 5: Scratch removal in real imagery: (a) reconstruction result for Figure 1(a),
(b) enlarged section of Figure 5(a).

Denoising: Содержание



- ◆ **Виды шума и их источники**
- ◆ **Методы подавления шума**
 - Методы, работающие в пространственной области
 - Методы, работающие во временной области
(как использующие, так и не использующие компенсацию движения)
 - Методы удаления вертикальных царапин со старых киноплёнок
- ◆ **MSU Noise Remover**
- ◆ **Методы оценки качества шумоподавления**

MSU Noise Remover

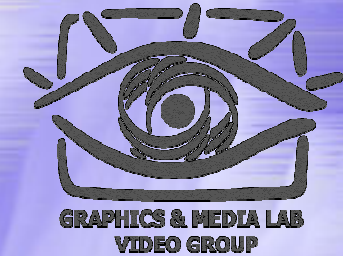


Фильтр состоит из 3-х частей:

1. Измерение уровня шума во временной и пространственно областях
2. Временное шумоподавление с использованием ME
3. Пространственное шумоподавление

MSU Noise Remover:

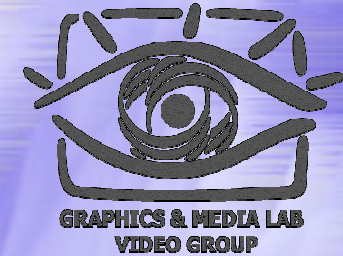
Измерение уровня шума



По нескольким выборочным блокам - однородным и не слишком темным/светлым – считается дисперсия, которая и является показателем зашумленности. Для пространственного шумоподавления она считается по кадру, для временного - между кадрами.

Замер уровня шума производится каждые n кадров и позволяет фильтру автоматически адаптироваться к уровню шума.

MSU Noise Remover: Временное шумоподавление



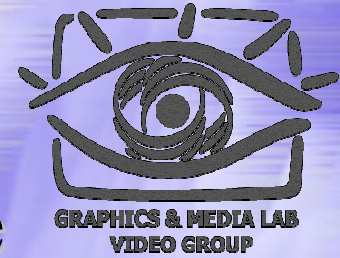
Пикселы текущего кадра смешиваются с пикселами из предыдущего скомпенсированного кадра с некоторым коэффициентом, зависящим от:

- ◆ Уровня временного шума
- ◆ Качества нахождения скомпенсированного блока
- ◆ Разницы между соотв. пикселями и его соседями

$$p_{cur} = k * p_{prev}^{MC} + (1 - k) * p_{cur}, \quad 0 \leq k \leq 0.5$$

MSU Noise Remover:

Пространственное шумоподавление



К различным областям кадра применяются гауссовские матрицы различного радиуса и силы размытия в зависимости от:

- ◆ Уровня пространственного шума
- ◆ Дисперсии в данной области

MSU Noise Remover: Results (sharpened)

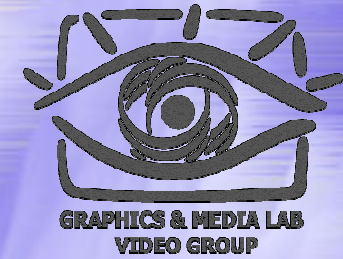


Source image

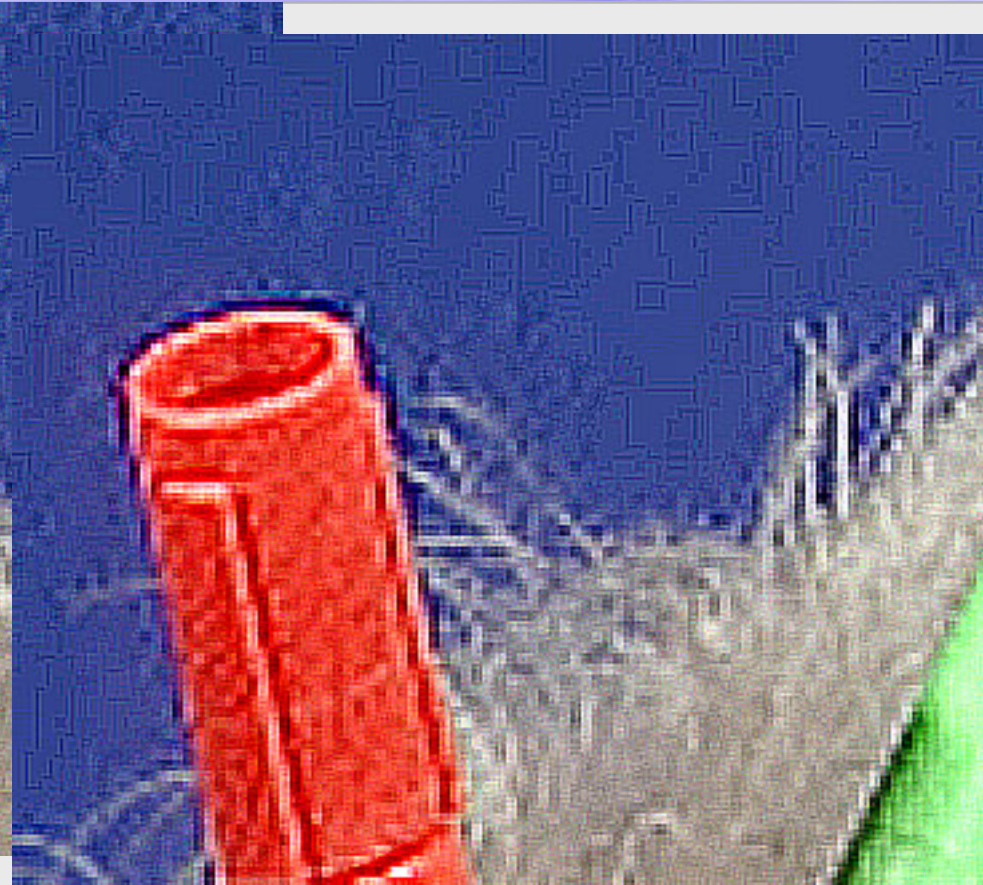


Processed image

MSU Noise Remover: Results (sharpened)

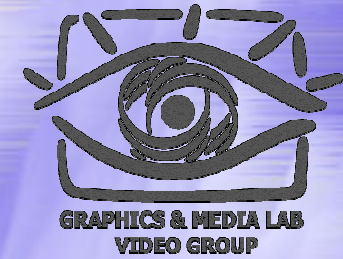


Source image



Processed image

MSU Noise Remover: Results (after DivX)

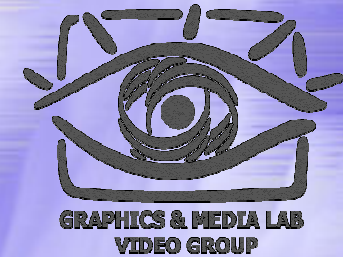


Source image



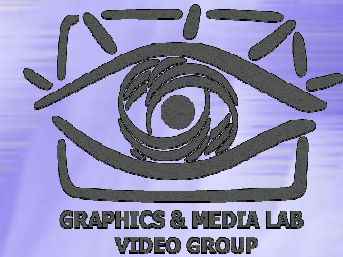
Processed image

Denoising: Содержание



- ◆ **Виды шума и их источники**
- ◆ **Методы подавления шума**
 - Методы, работающие в пространственной области
 - Методы, работающие во временной области
(как использующие, так и не использующие компенсацию движения)
 - Методы удаления вертикальных царапин со старых киноплёнок
- ◆ **MSU Noise Remover**
- ◆ **Методы оценки качества шумоподавления**

Методы оценки качества шумоподавления



- ◆ **PSNR** - мера отношения сигнала к шуму (peak-to-peak signal-to-noise ratio):

$$d(x, y) = 10 * \log_{10} \frac{255^2 * n^2}{\sum_{i=1, j=1}^{n, n} (x_{ij} - y_{ij})^2}$$

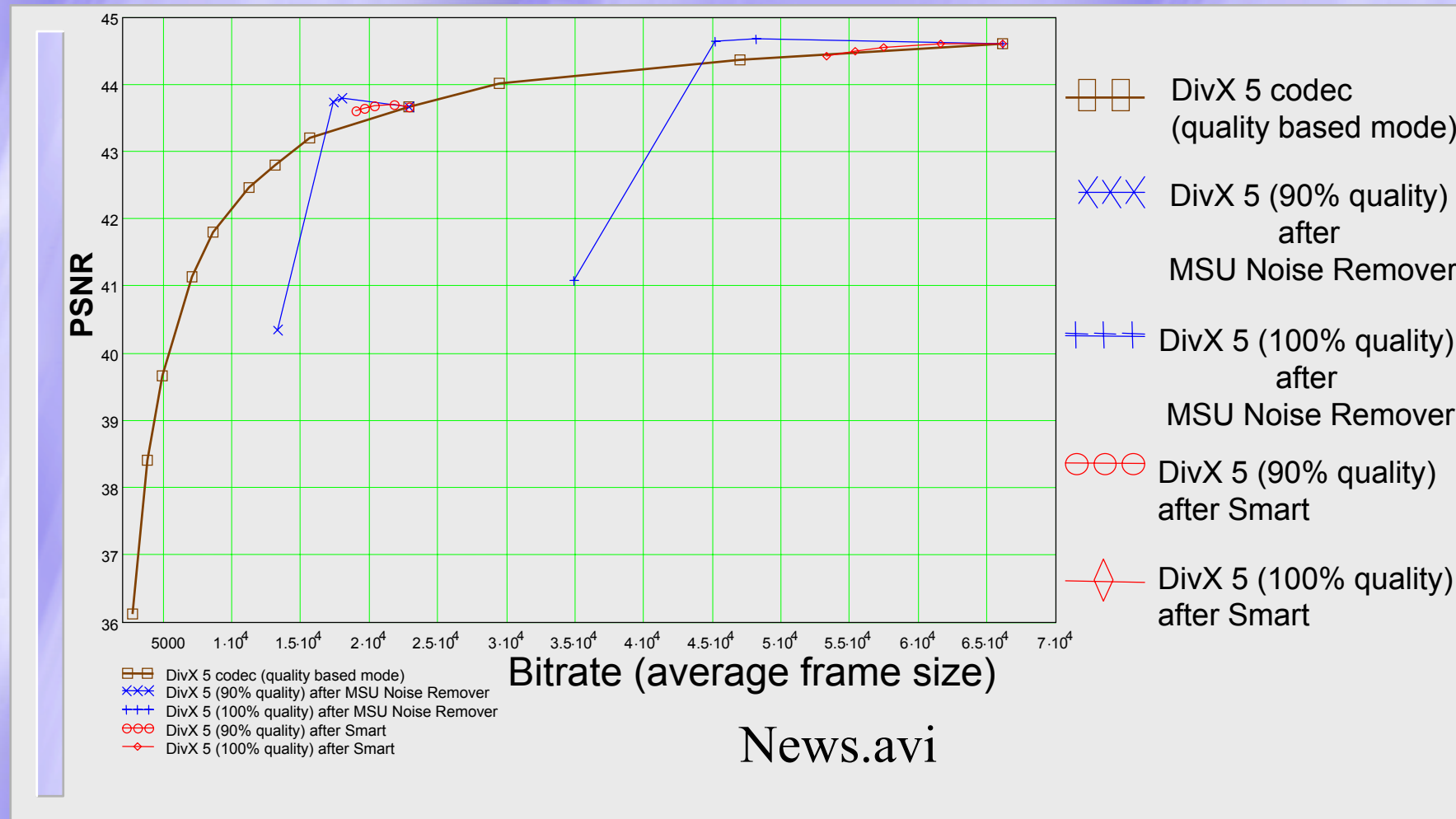
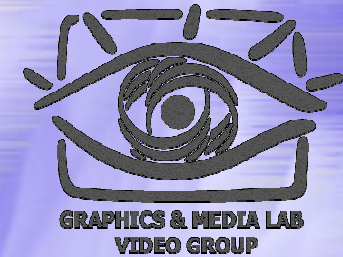
- ◆ **NMSE** - normalized mean square error:

$$NMSE = \frac{\sum_{x=1}^X \sum_{y=1}^Y (f(x, y) - f_e(x, y))^2}{\sum_{x=1}^X \sum_{y=1}^Y f(x, y)^2}$$

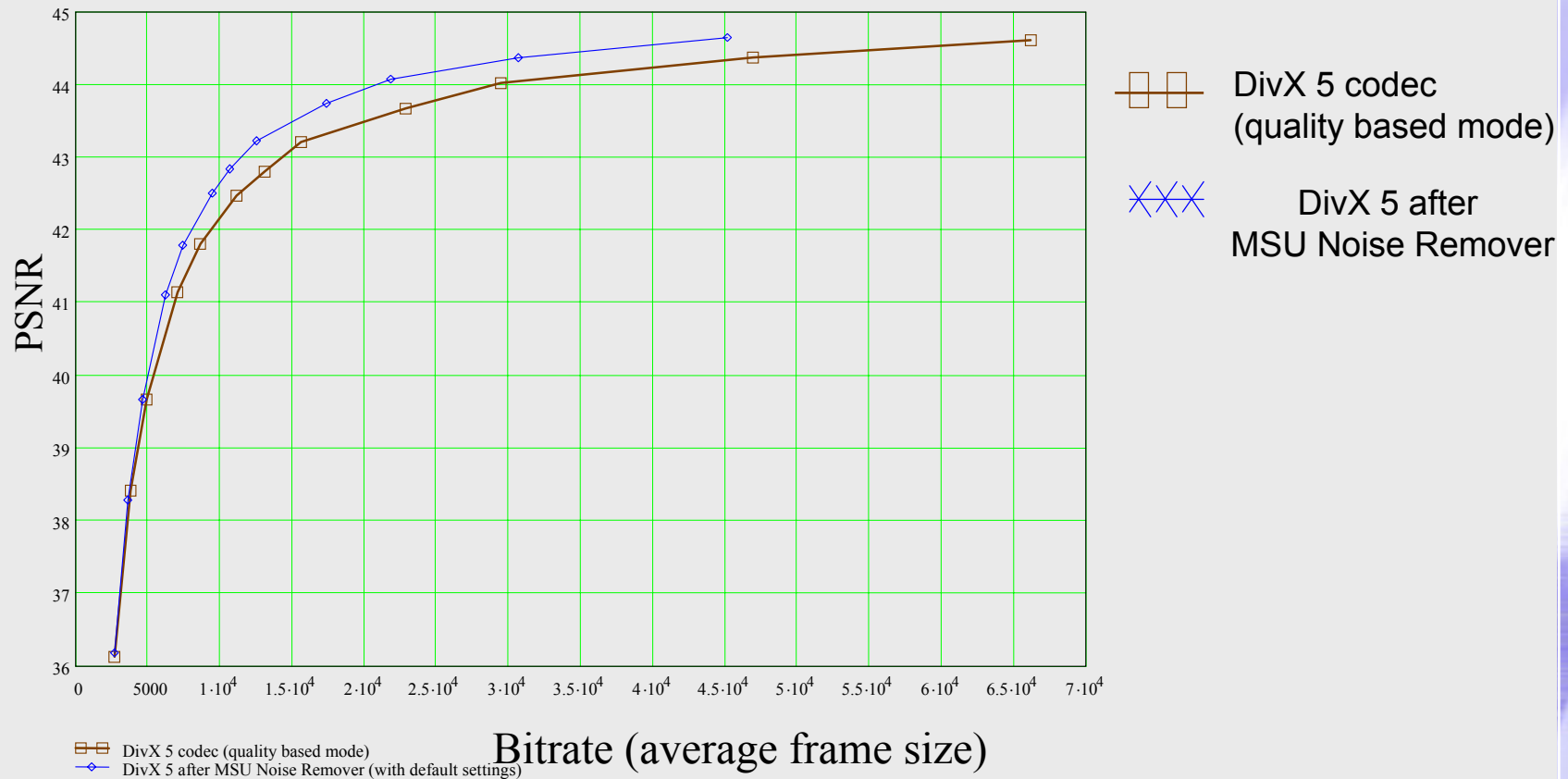
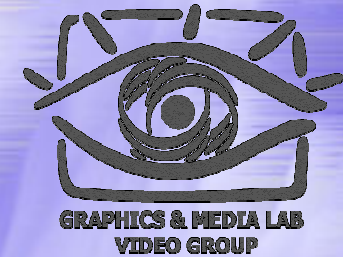
где $f()$ – исходное изображение
 $f_e()$ – обработанное изображение

- ◆ Графики **PSNR/Bitrate**

PSNR/Bitrate Diagram

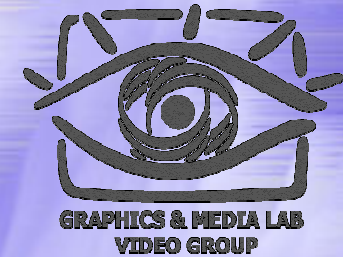


PSNR/Bitrate Diagram



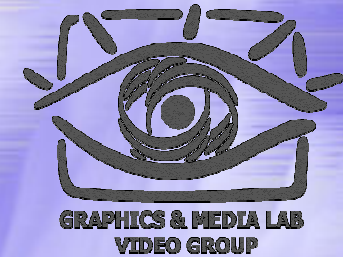
News.avi

Шумоподавление и интерлейсинг



- ◆ Рассмотрим временное шумоподавление при интерлейсинге
- ◆ Проблема: если применять усреднение с предыдущим полем той же четности, то имеем слишком большую разницу во времени (объекты могли сильно сдвинуться). Если же применять усреднение с ближайшим полем другой четности, то теряем вертикальные делала (если только у нас не суперкачественный деинтерлейсинг).
- ◆ Решение: разбиваем изображение на 2 частотные полосы и к ним применяются 2 разных способа...

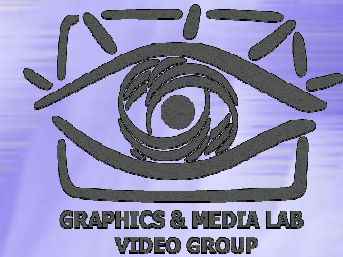
Шумоподавление и интерлейсинг



- ◆ Для низких частот: усреднение с использованием предыдущего поля
 - Теряем вертикальное разрешение, но для НЧ это не критично
 - Получаем хорошее разрешение по времени, минимизируем площадь необработанных областей

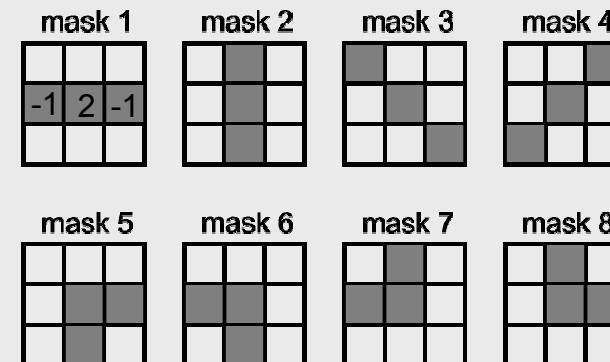
- ◆ Для высоких частот: усреднение с использованием поля той же четности
 - Получаем хорошее разрешение по вертикали → четкое изображение
 - Ухудшается подавление шума, НО: оно и не важно, т.к. глаз практически не чувствителен к шуму на высоких частотах

Edge-directional filtering



- ◆ Пространственное шумоподавление: как повысить эффективность размытия?

8 возможных
конфигураций фильтров

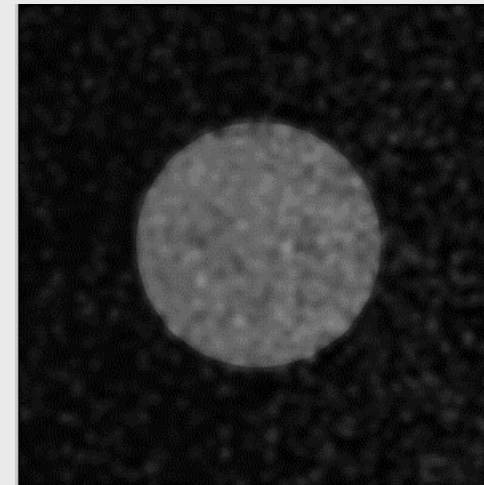
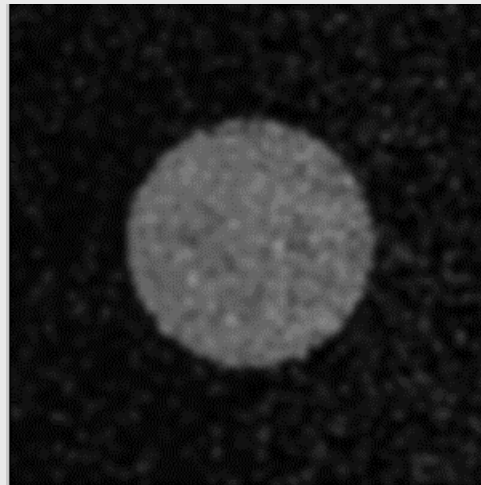
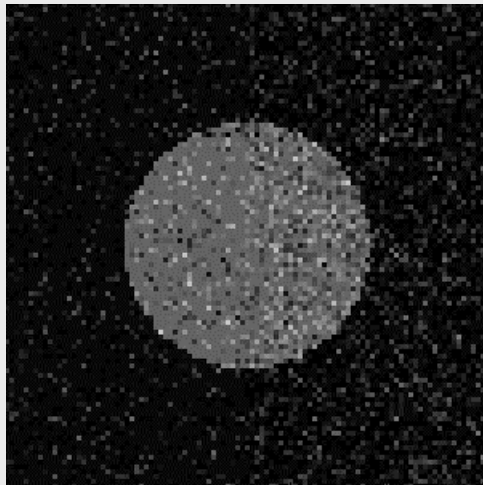


- ◆ Размытие вдоль границ изображения
 1. Находим наилучшее направление размытия, вычисляя «производную по направлению»
 2. Проводим фильтрацию, регулируя силу эффекта исходя из наличия границы (величина производной)

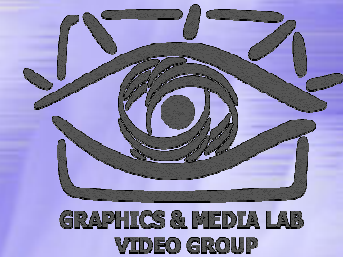
Edge kernels



- ◆ Адаптация ядра свертки к краям изображения



Анизотропная диффузия



- ◆ Карта интенсивности как карта температуры. Моделирование процесса теплопереноса

$$I_t = c \cdot \Delta I$$

- ◆ Решение ур-я теплопроводности

$$I(x, y, t) = \iint I(x', y', 0)G(x, x', y, y', t)dx'dy'$$

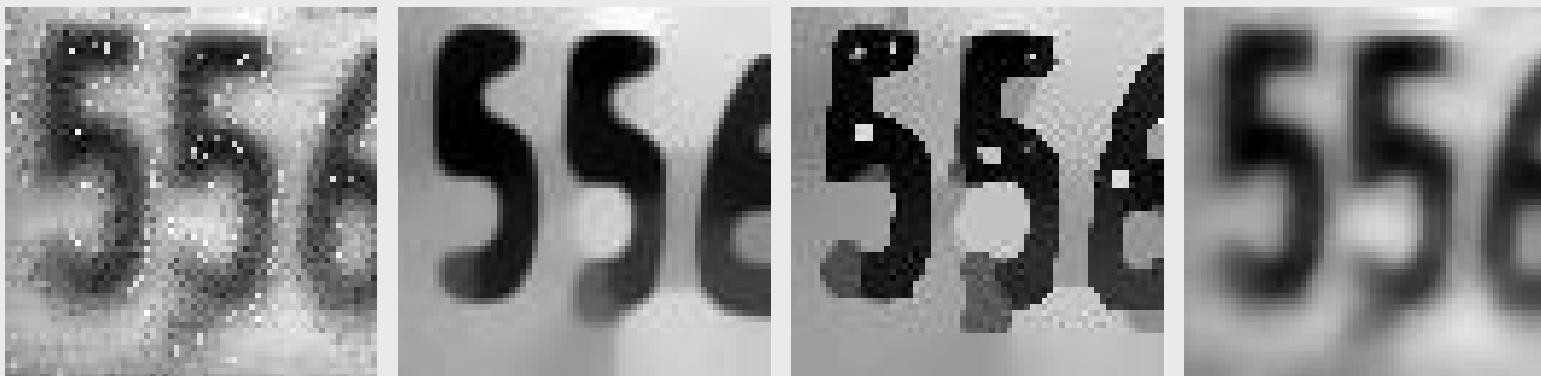
- ◆ Пусть коэффициент теплопроводности зависит от градиента интенсивности изображения

$$c(\nabla I) = \exp\left(-\left(\frac{|\nabla I|}{k}\right)^2\right)$$

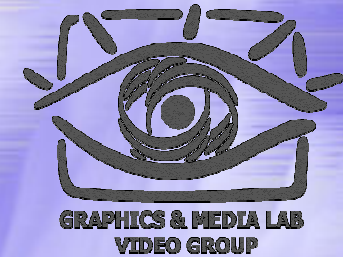
Анизотропная диффузия



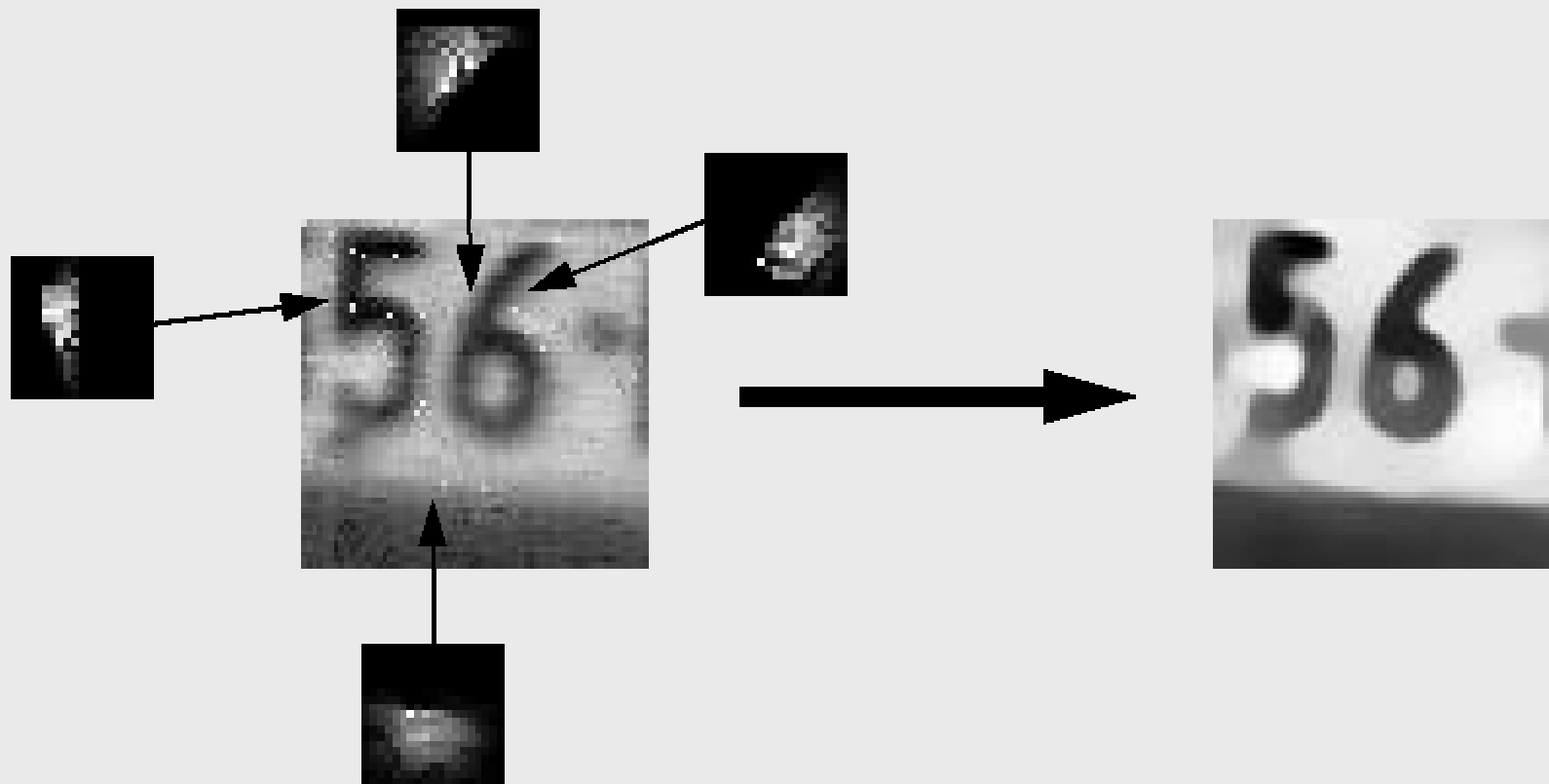
- ◆ Проблемы: решение дифференциального уравнения неизвестно → нужно интегрировать численно → медленный итерационный метод



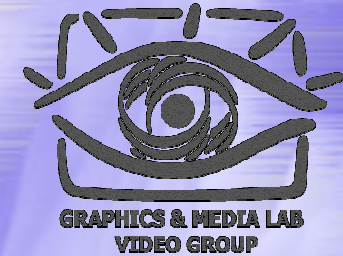
Анизотропная диффузия



◆ Эквивалентные ядра свертки

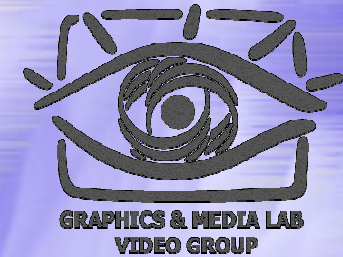


Список литературы



1. "Combined Wavelet Domain and Temporal Video Denoising" Aleksandra Pizurica, Vladimir Zlokolica and Wilfried Philips
2. "Noise Reduction in Video Sequences Using Wavelet-Domain and Temporal Filtering" Aleksandra Pizurica, Vladimir Zlokolica and Wilfried Philips
3. "Television noise reduction ic" G. de Haan, T.G. Kwaaitaal-Spassova, M.M. Larragy, O.A. Ojo, and R.J. Schutten
4. "Noise filtering for television receivers with reduced memory" R. J. Schutten, G. de Haan and A. H. M. van Roermund
5. "Automatic 2-D and 3-D noise filtering for high-quality television receivers" G. de Haan, T. G. Kwaaitaal-Spassova and O.A. Ojo
6. "A two-pass median-like filter for impulse noise removal in multi-channel images" Szymon Grabowski, Wojciech Bieniecki
7. "Removal of Vertical Scratches in Digitised Historical Film Sequences Using Wavelet Decomposition" T. Bretschneider, O. Kao, P.J. Bones
8. "A New Algorithm for Image Noise Reduction using Mathematical Morphology" Richard Alan Peters, IEEE Transactions on Image Processing, May 1995
9. "Spatio-Temporal Noise Reduction ASIC for Real-Time Video Processing" Hakan Norell, Bengt Oelmann and Youshi Xu

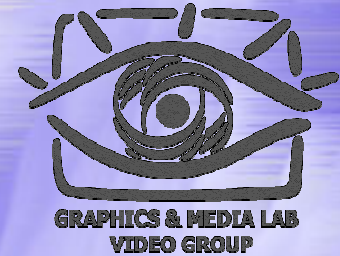
Deblocking: Содержание



- ◆ Понятие блокинга и возможные стратегии деблокинга
- ◆ Алгоритмы деблокинга
 - 4-х пиксельный
 - 6-ти пиксельный метод
 - Алгоритм фильтра Ffdshow
 - Modeled Low Pass Filter
 - Adaptive Low Pass Filter
- ◆ Алгоритм MSU Media Lab
- ◆ Сравнение описанных алгоритмов

Понятие блокинга

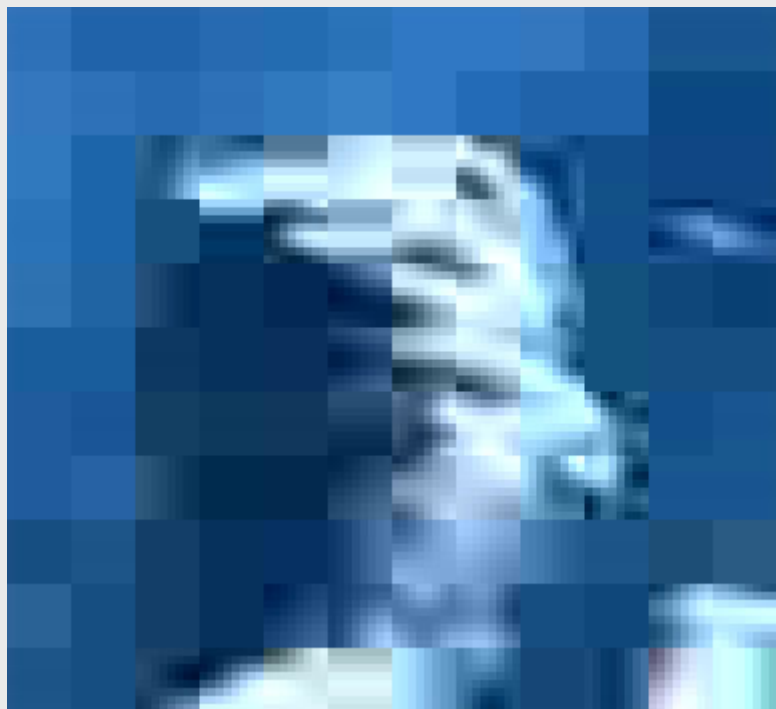
Причины блочности



- ◆ Возникает при кодировании на низких битрейтах при использовании алгоритмов, основанных на DCT
- ◆ Деблокинг still images и video – два абсолютно разных направления
- ◆ Особенности деблокинга видео:
 - Критическое значение имеет скорость алгоритма
 - Сила блочности меняется во времени

Понятие блокинга

Примеры



Battle.avi

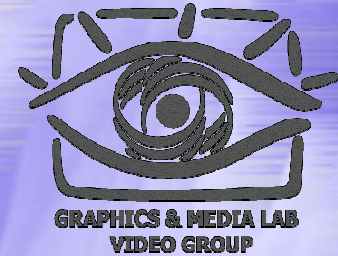


IceAge.avi

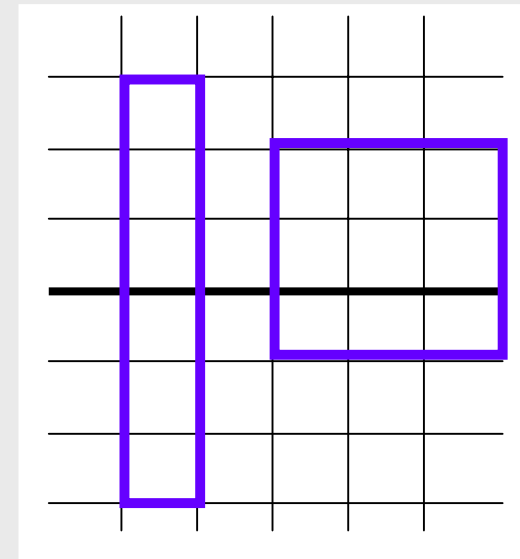
Blocked frames

Понятие блокинга

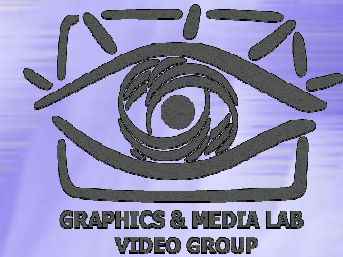
Возможные стратегии деблокинга



- ◆ Обработка одной строки(столбца) на границе для каждого пикселя
- ◆ Использование шаблона
- ◆ Оценка параметров кодирования и ее использование при обработке границы блоков
- ◆ Использование Вейвлет преобразования



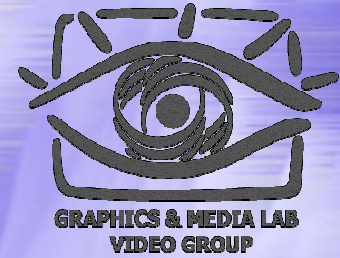
Deblocking: Outline



- ◆ Понятие блокинга и возможные стратегии деблокинга
- ◆ **Алгоритмы деблокинга**
 - 4-х пиксельный
 - 6-ти пиксельный метод
 - Алгоритм фильтра Ffdshow
 - Modeled Low Pass Filter
 - Adaptive Low Pass Filter
- ◆ Алгоритм MSU Media Lab
- ◆ Сравнение описанных алгоритмов

Алгоритмы деблокинга

Основные требования

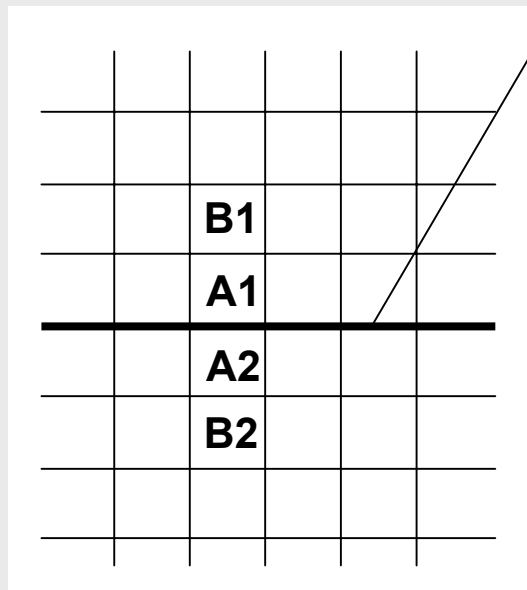
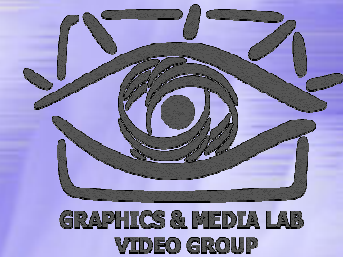


Алгоритм деблокинга должен удовлетворять следующим требованиям:

- Сглаживать блочность
- Оставлять неизменными края объектов

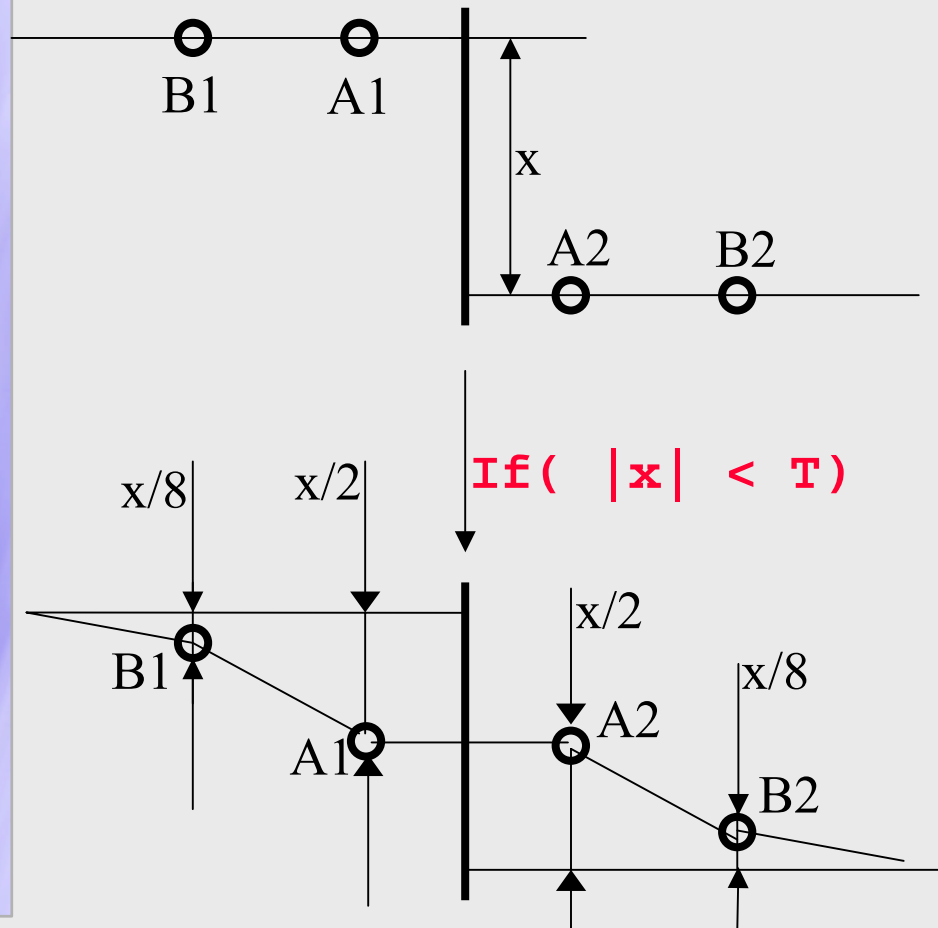
Очевидно, что требования - противоречивые

4-х пиксельный метод



```
X=A1-A2
If( |x| < T )
{
    B1 -= x/8
    A1 -= x/2
    A2 += x/2
    B2 += x/8
}
```

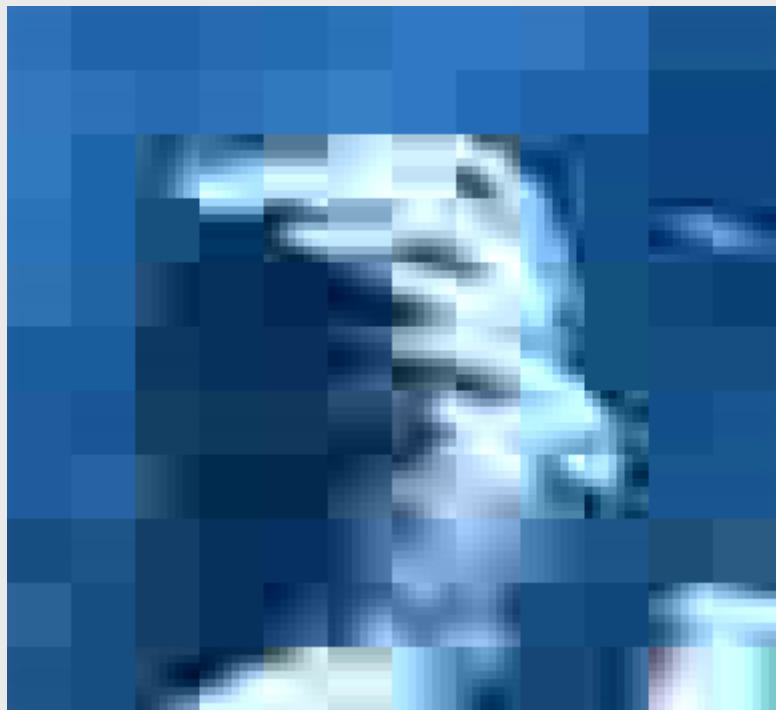
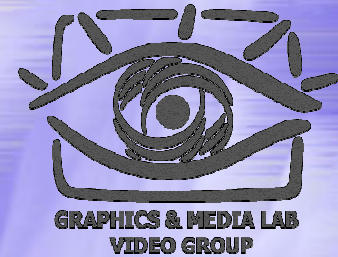

4-х пиксельный метод (2)



Недостатки:

- ✓ Артефакты
- ✓ Пространственная локализованность
- ✓ Отсутствие адаптивности

4-х пиксельный метод (3)

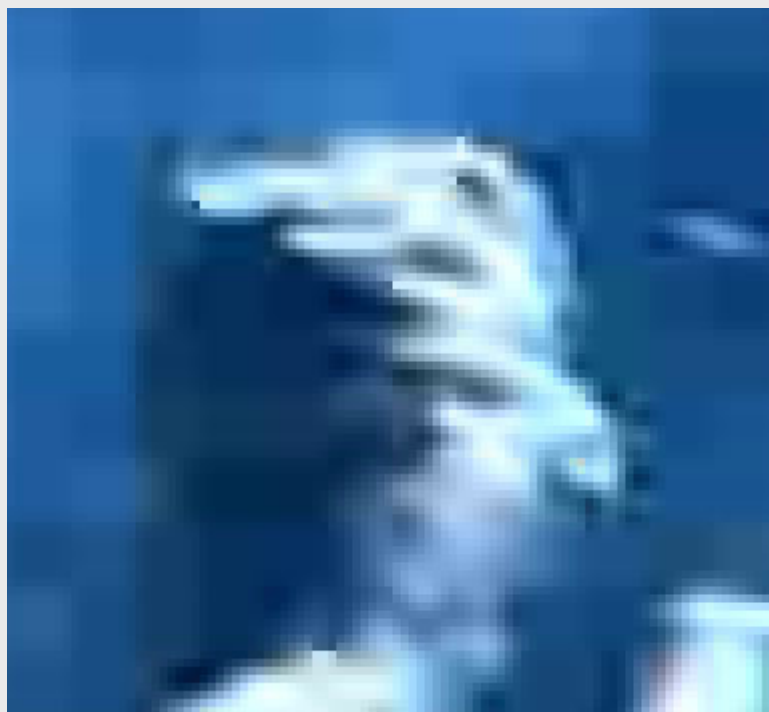


Battle.avi

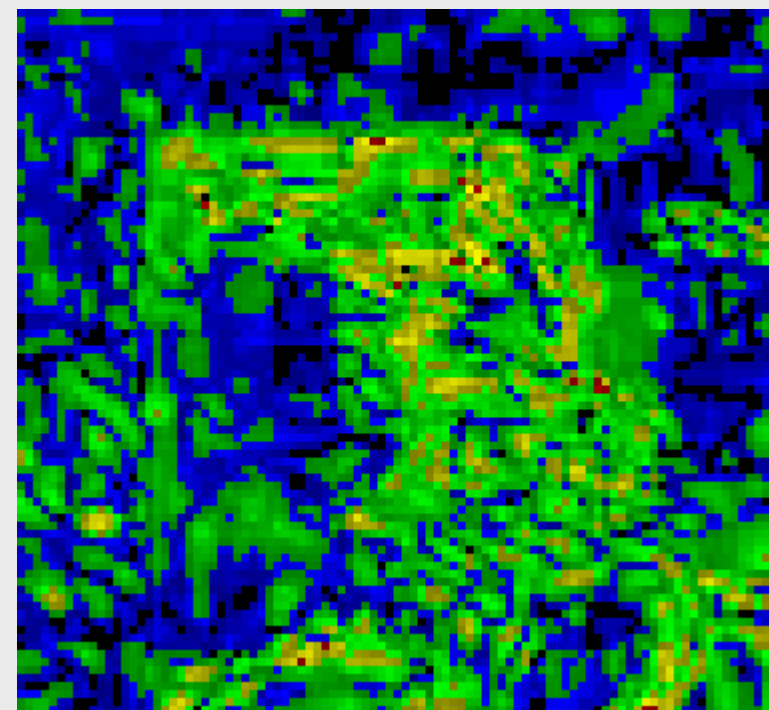


T = 80

4-х пиксельный метод (4)

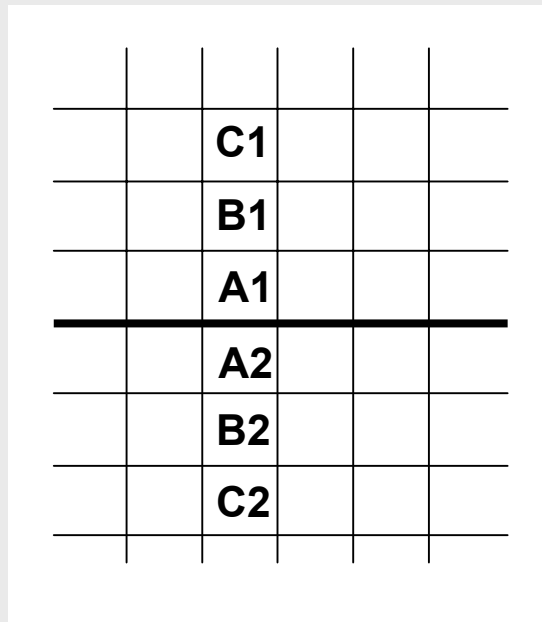
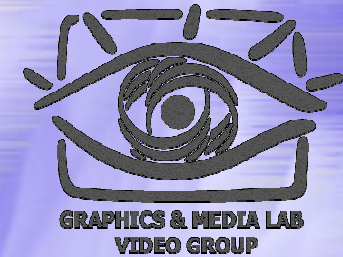


T = 80



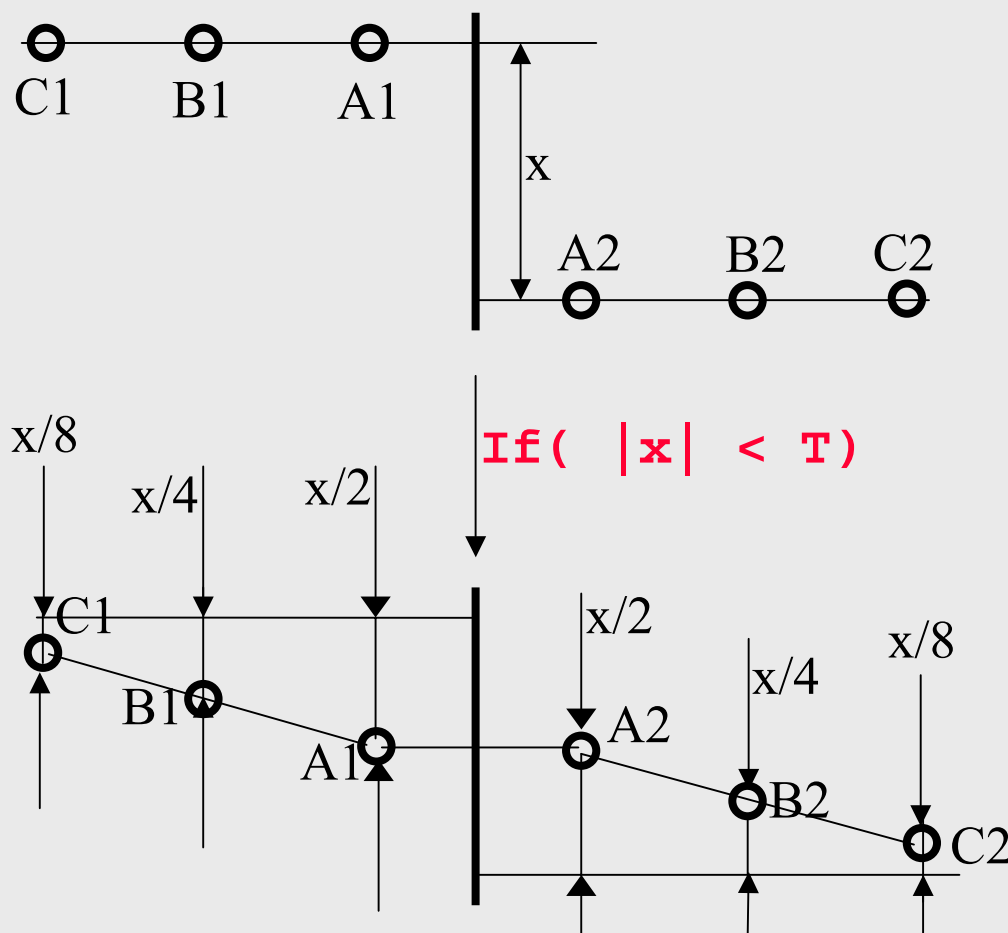
Y-PSNR = 32.03 db

6-ти пиксельный метод



```
x = A1 - A2
If( |x| < T )
{
    C1 -= x/8
    B1 -= x/4
    A1 -= x/2
    A2 += x/2
    B2 += x/4
    C2 += x/8
}
```

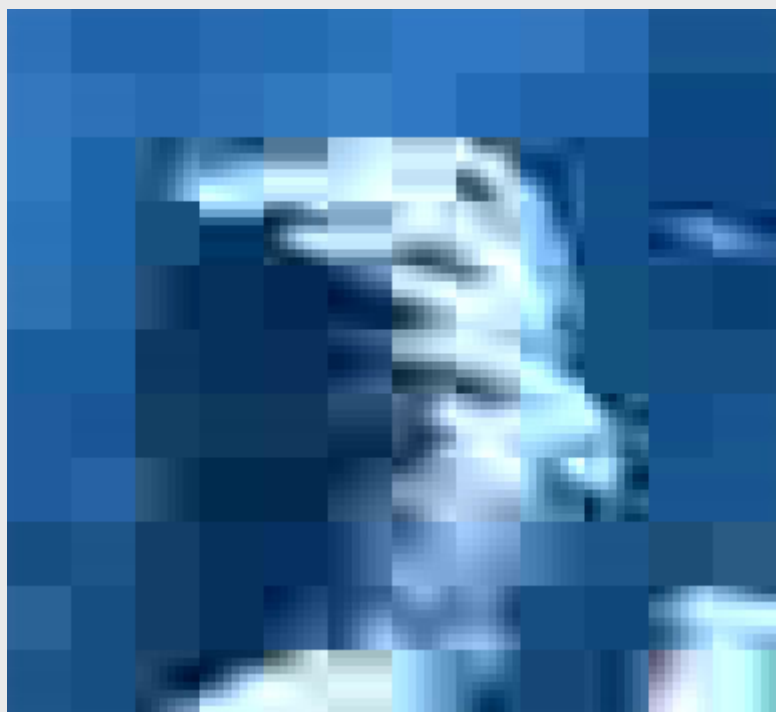
6-ти пиксельный метод (2)



Краткое описание:

- ✓ Те же недостатки, что и у 4-х пиксельного
- ✓ Применяется в случае более сильной блочности

6-ти пиксельный метод (3)



Battle.avi



6-Pixel, T = 40

6-ти пиксельный метод (4)

Сравнение с 4-х пиксельным алгоритмом



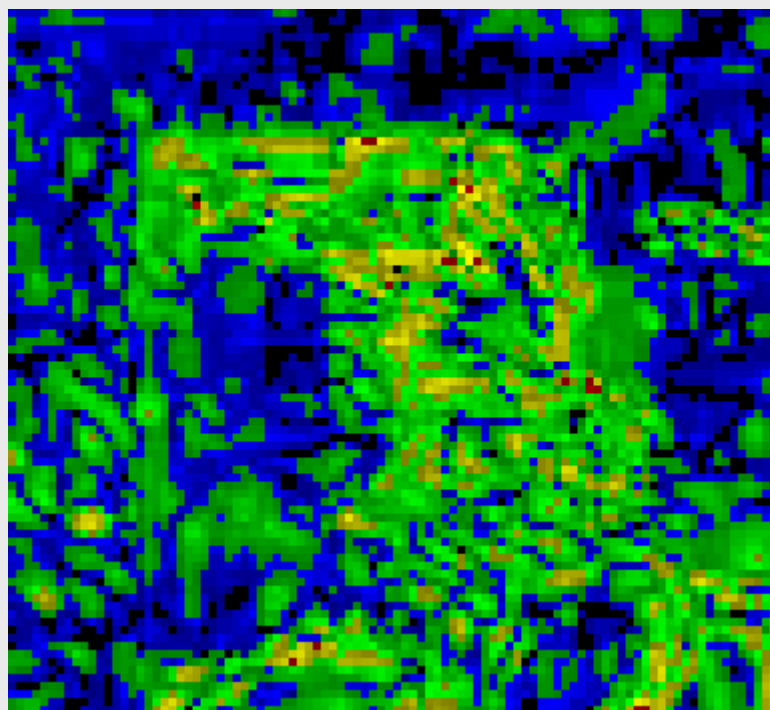
4-Pixel, T = 80



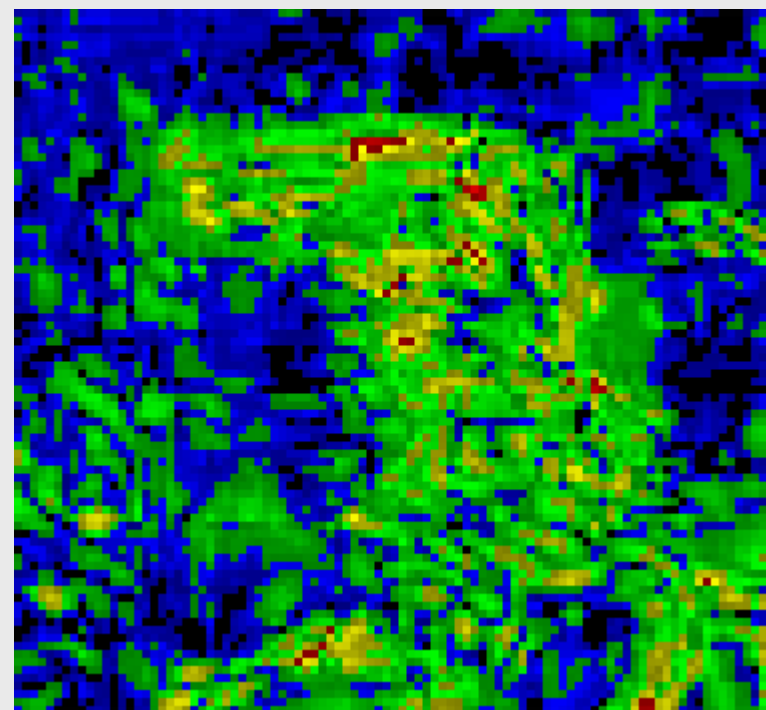
6-Pixel, T = 40

6-ТИ ПИКСЕЛЬНЫЙ МЕТОД (5)

Сравнение с 4-х пиксельным алгоритмом

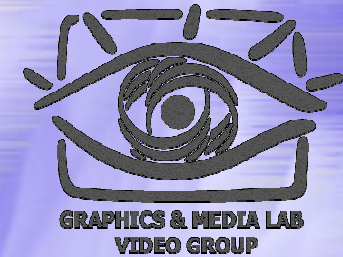


4-Pixel: Y-PSNR = 32.03



6-Pixel: Y-PSNR = 32.06

Метод фильтра Ffdshow

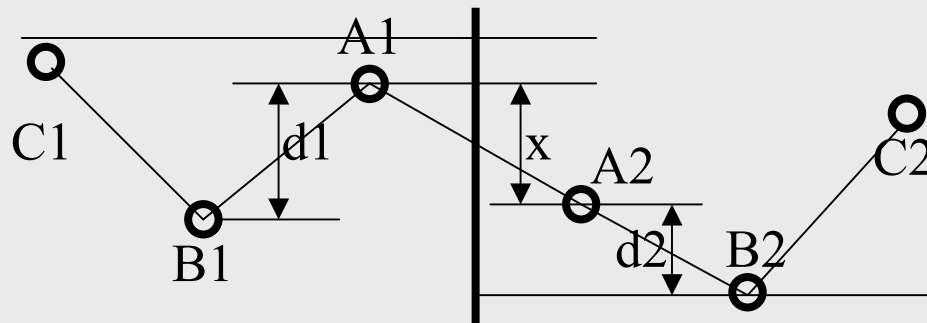
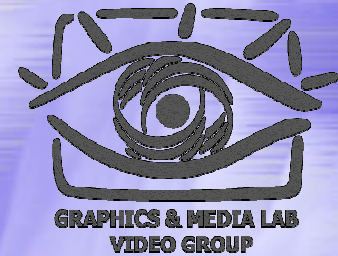


Описание алгоритма:

1. Оценка силы блочности
2. Выбор длины фильтра в зависимости от силы блочности(4-х или 6-пиксельный алгоритм)
3. Анализ присутствия детали
4. Если детали нет, то обработка границы выбранным алгоритмом

“A Simple and Efficient Deblocking Algorithm for Low Bit-Rate Video Coding”, K. Ramkishor, Pravin Karandokar

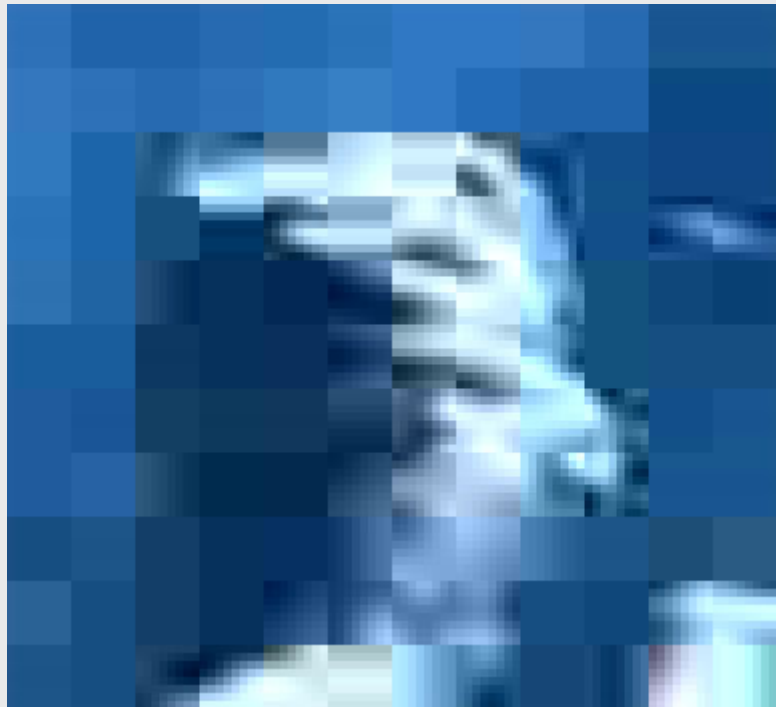
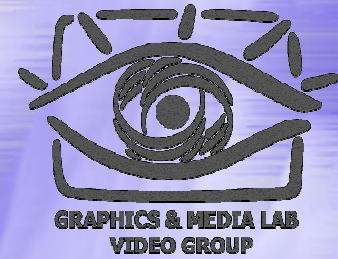
Метод фильтра Ffdshow(2)



```
d1=A1-B1  
d2=A2-B2  
If( |d1| + |d2| < 5 )  
    6Pixel( 2*T )  
Else 4Pixel( 1.5*T )
```

- ✓ Оценка силы блочности:
 $|d1| + |d2| < 5$
- ✓ Анализ присутствия детали:
пороги в базовых алгоритмах

Метод фильтра Ffdshow(3)



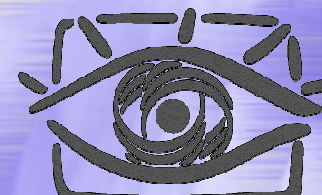
Battle.avi



Ffdshow: T = 60

Метод фильтра Ffdshow(4)

Сравнение с 6-ти пиксельным алгоритмом



GRAPHICS & MEDIA LAB
VIDEO GROUP



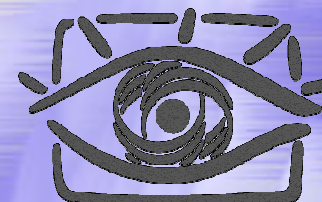
6-Pixel, T = 40



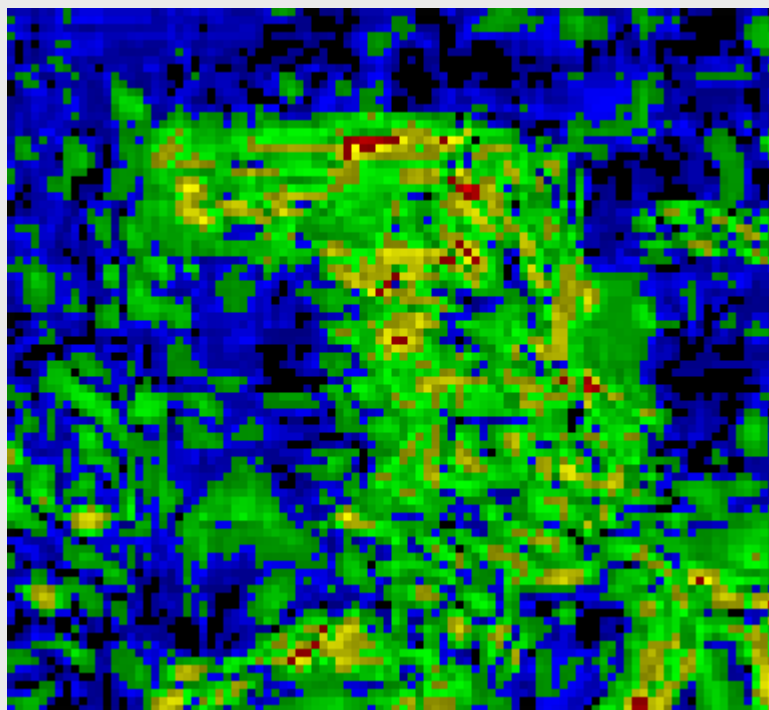
Ffdshow: T = 60

Метод фильтра Ffdshow(5)

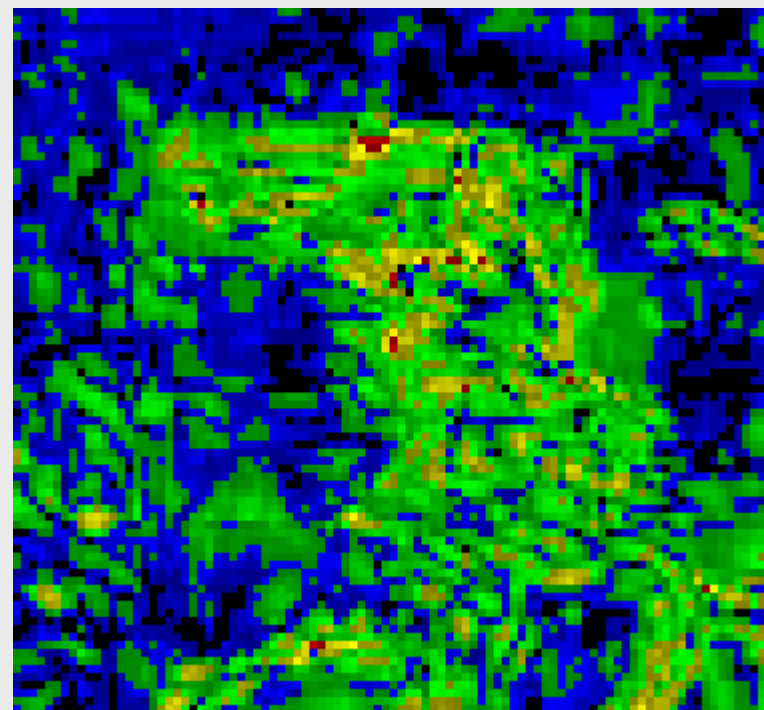
Сравнение с 6-ти пиксельным алгоритмом



GRAPHICS & MEDIA LAB
VIDEO GROUP

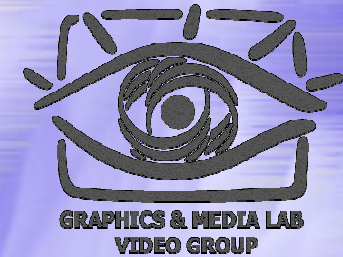


6-Pixel: Y-PSNR = 32.06



FFdshow: Y-PSNR = 32.15

Modeled Lowpass Filter



Общая идея алгоритма:

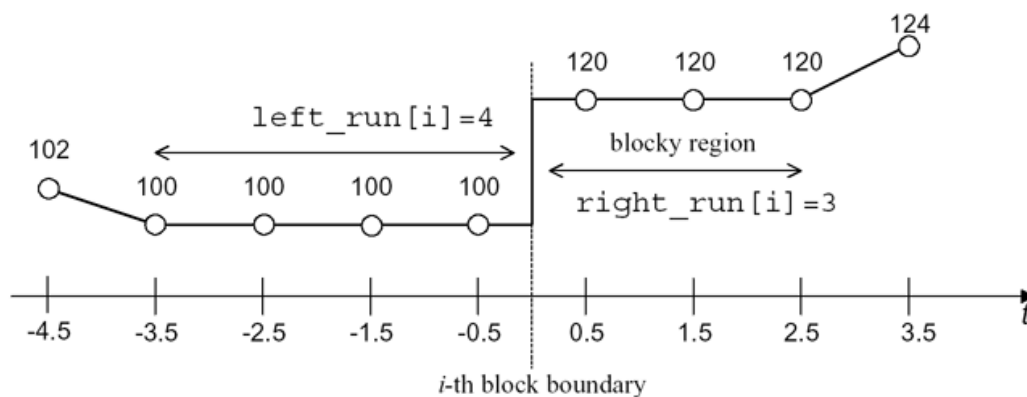
1. С каждой стороны границы определяется число пикселей, подлежащих обработке
2. Выход фильтра зависит от выбранной модели (модель – функция, используемая для фильтрации)

Пример модели:

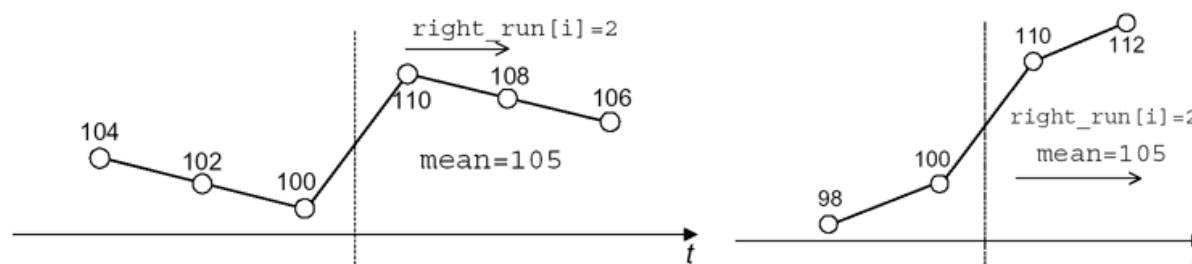
$$y(t) = A(1 - e^{-\lambda t}).$$

Modeled Lowpass Filter(2)

Типы границ



(a)



(b)

(c)

(a) - Flat
boundary

(b), (c) - NonFlat
boundary

Modeled Lowpass Filter(4)

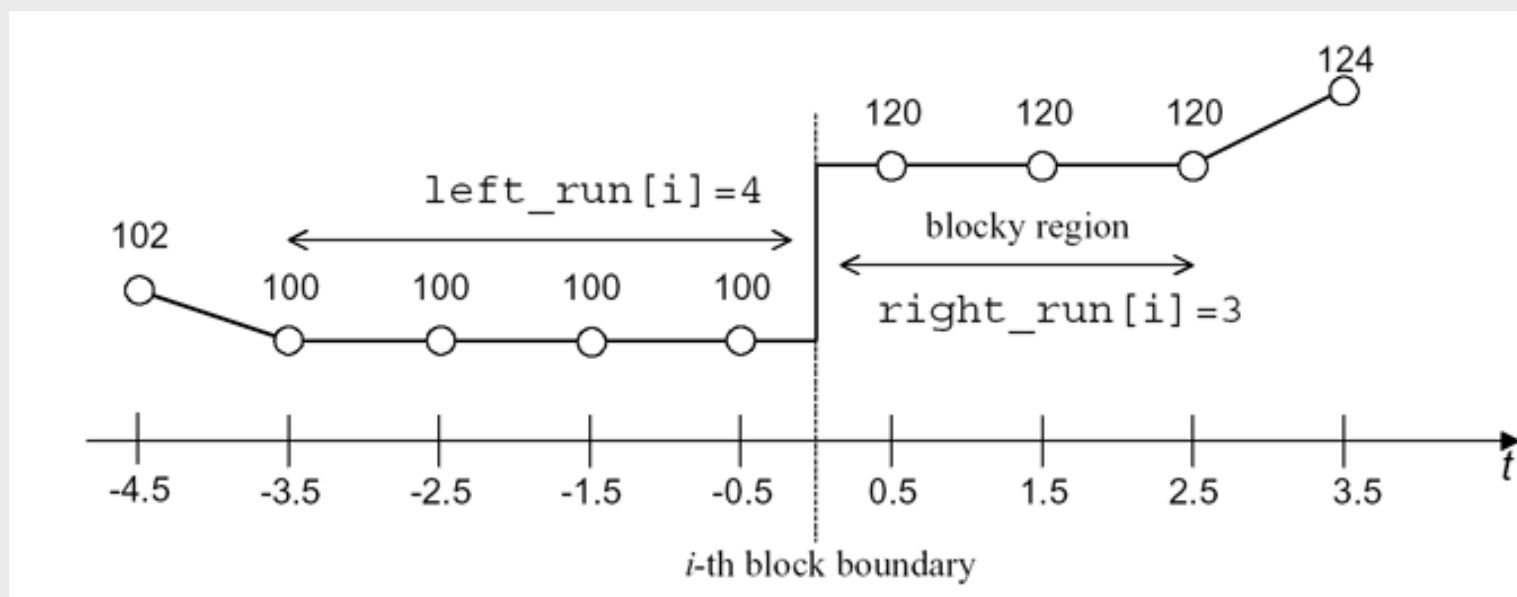
Длина области с блочностью



- ◆ Flat boundary – количество одинаковых пикселей в каждом блоке
- ◆ NonFlat boundary – если разница между первыми двумя пикселями в к-л. блоке вдвое меньше разницы между пикселями на границе блока, то кол-во обрабатываемых пикселей – 2, иначе - 1

Modeled Lowpass Filter(3)

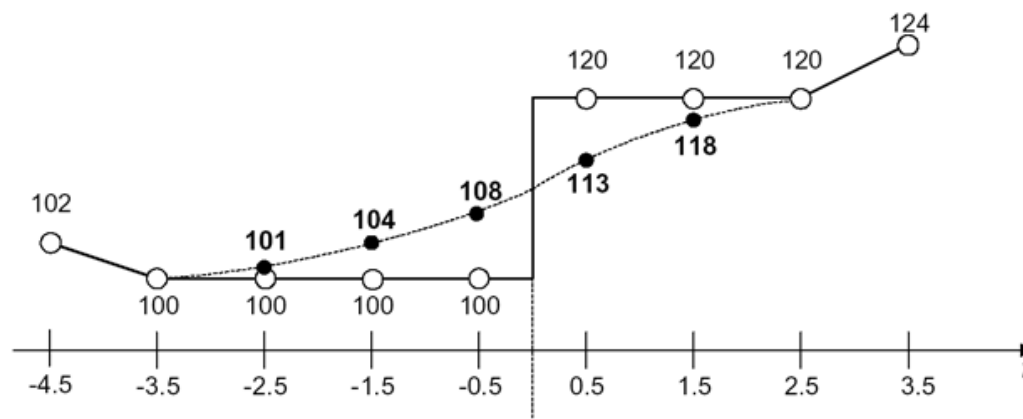
Пример модели



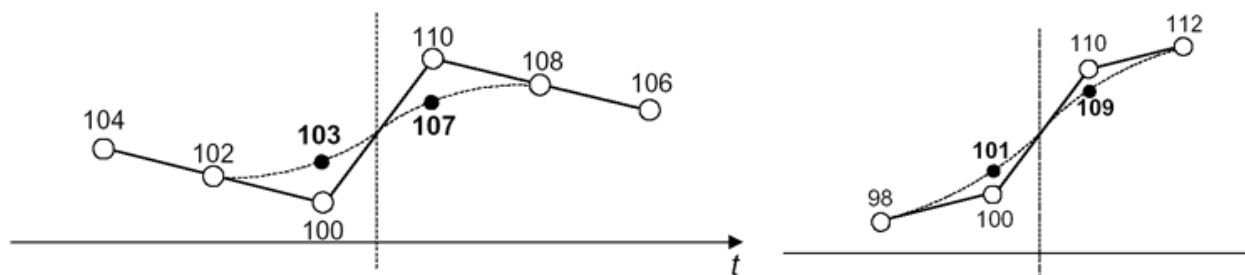
$$y(t) = \begin{cases} 110 + 10 \sin \left(\frac{\pi t}{2(\text{right_run} - 0.5)} \right), & t \geq 0 \\ 110 - 10 \sin \left(\frac{-\pi t}{2(\text{left_run} - 0.5)} \right), & t < 0 \end{cases}$$

Modeled Lowpass Filter(5)

Пример работы



(a)



(b)

(c)

Modeled Lowpass Filter(6)

Пример работы



Blocked frame



Deblocked

Modeled Lowpass Filter(7)

Сравнение

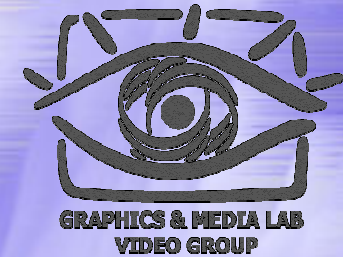


Crouse



Proposed

Adaptive LowPassFilter



Общая идея алгоритма:

1. Фильтрация в диагональном направлении
2. Фильтрация по горизонтали и вертикали

Adaptive LowPassFilter(2)

Фильтрация по диагонали

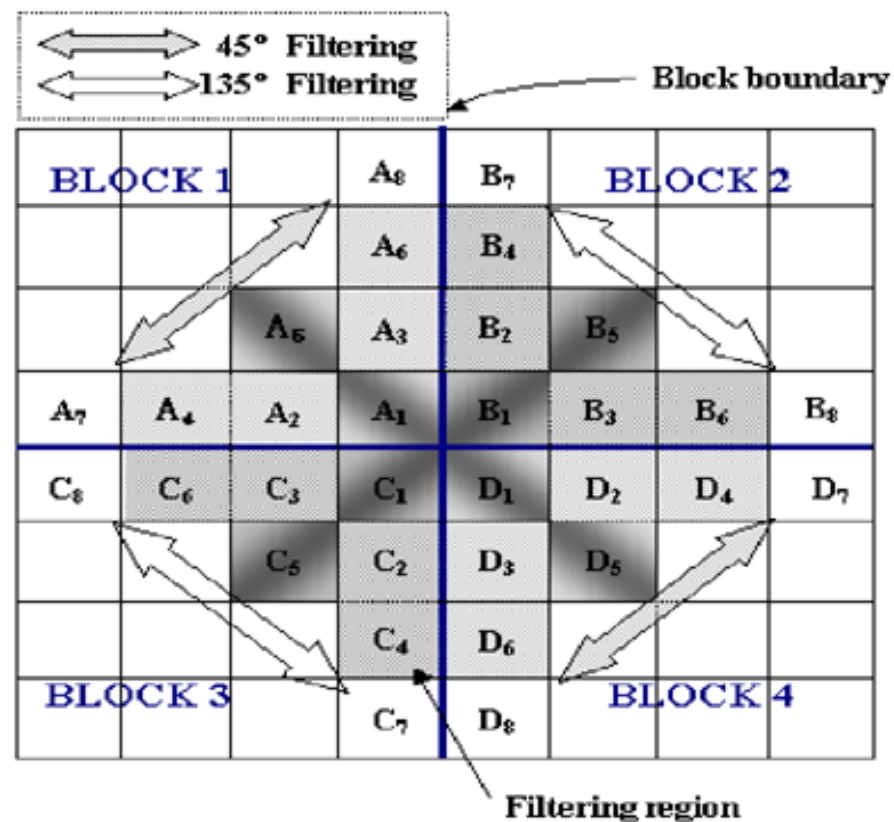
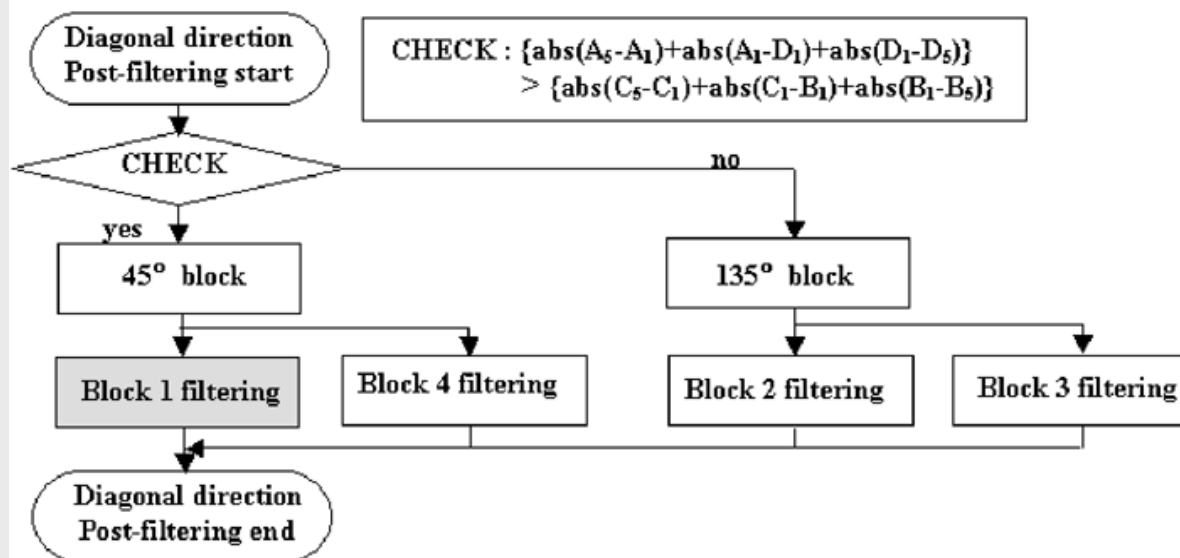


Figure 1. The filtering region and pixel difference in the diagonal direction

Adaptive LowPassFilter(3)

Фильтрация по диагонали



(a) The filtering procedure for diagonal direction.

If (CHECK)

$$= \begin{cases} \text{The filtering for } 135^\circ \text{ direction, when } CHECK = 1 \\ \text{The filtering for } 45^\circ \text{ direction, when } CHECK = 0 \end{cases} \quad (1)$$

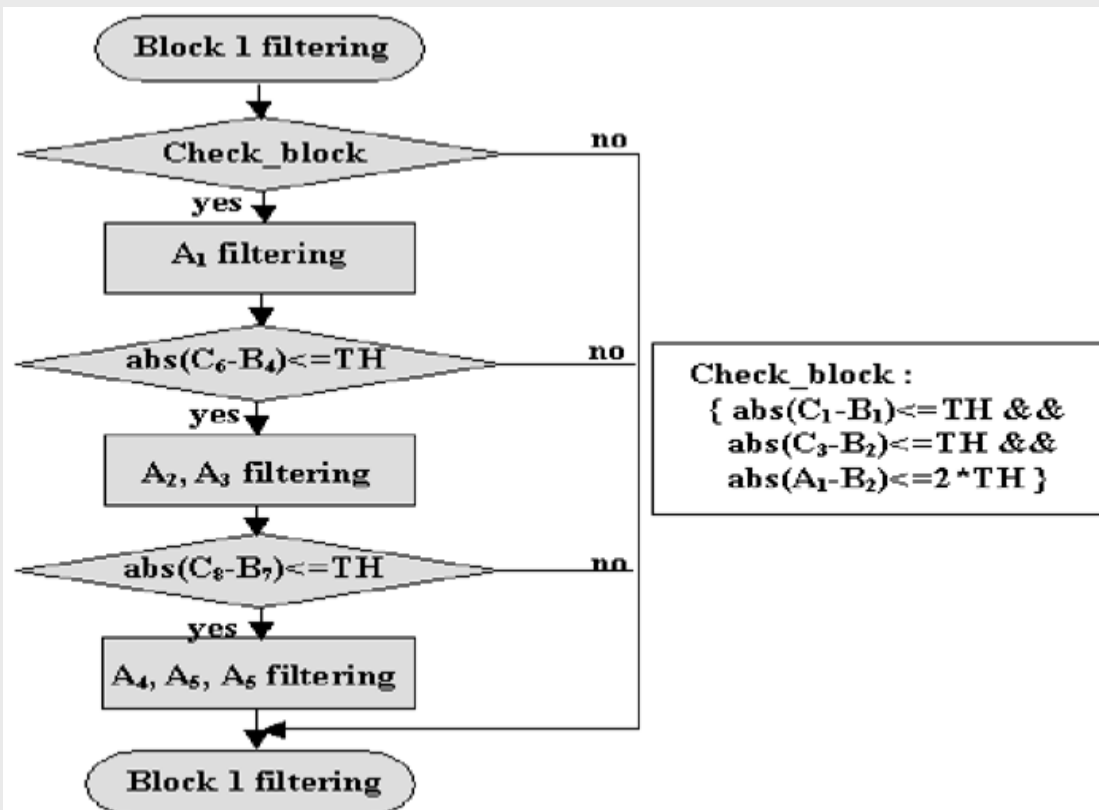
$$CHECK : \{abs(A_5 - A_1) + abs(A_1 - D_1) + abs(D_1 - D_5)\} > \{abs(C_5 - C_1) + abs(C_1 - B_1) + abs(B_1 - B_5)\}$$

Adaptive LowPassFilter(4)

Фильтрация по диагонали



BLOCK 1			A_8	B_7
			A_6	B_4
		A_5	A_3	B_2
A_7	A_4	A_2	A_1	B_1
C_8	C_6	C_3	C_1	D_1



(b) The filtering procedure for *BLOCK1*

Adaptive LowPassFilter(4)

Фильтрация по диагонали

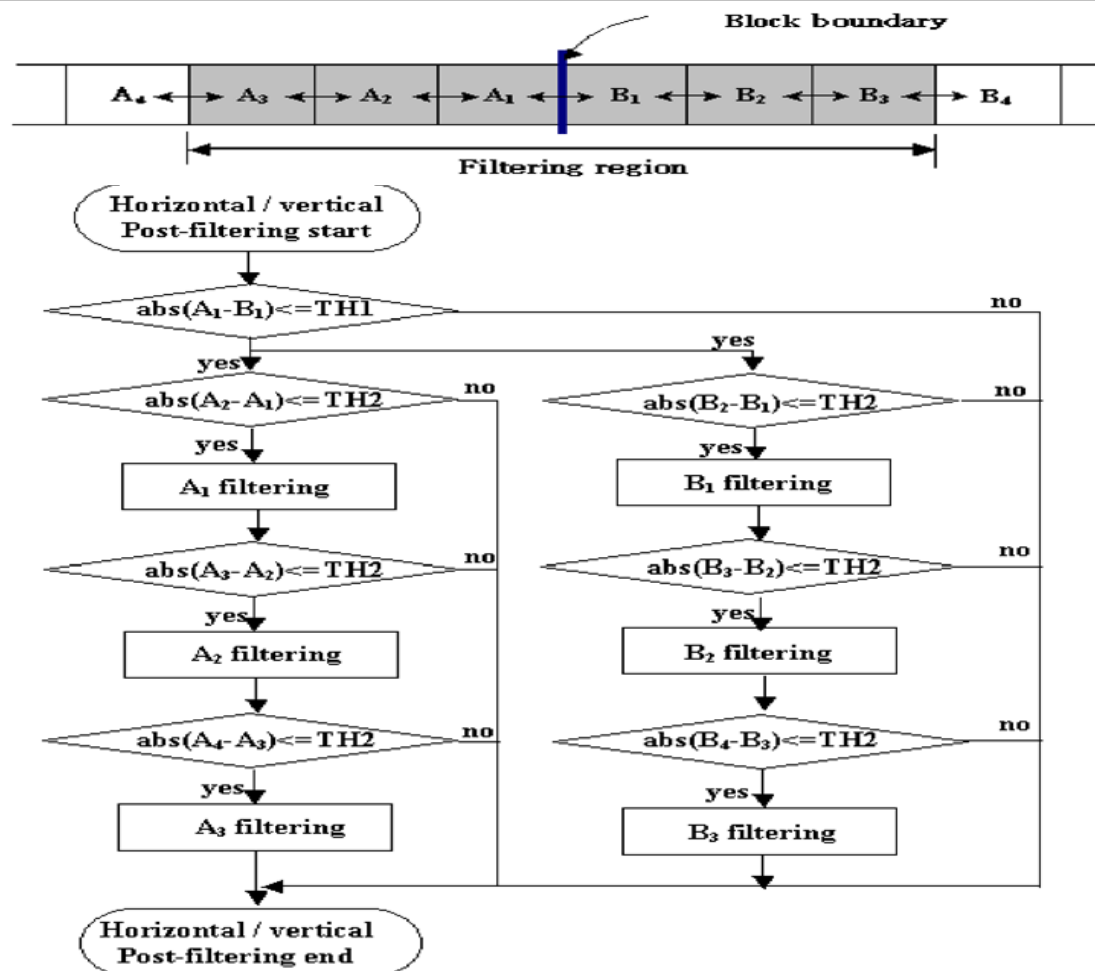


BLOCK 1			A_8	B_7
			A_6	B_4
		A_5	A_3	B_2
A_7	A_4	A_2	A_1	B_1
C_8	C_6	C_3	C_1	D_1

$$\left\{ \begin{array}{l} A_1 \text{ filtering} : A'_1 = [C_3 \ A_1 \ B_2] \cdot [1 \ 2 \ 1]^T // 4 \\ A_2 \text{ filtering} : A'_2 = [C_6 \ A_2 \ A_3] \cdot [1 \ 2 \ 1]^T // 4 \\ A_3 \text{ filtering} : A'_3 = [A'_2 \ A_3 \ B_4] \cdot [1 \ 2 \ 1]^T // 4 \\ A_4 \text{ filtering} : A'_4 = [C_8 \ A_4 \ A_5] \cdot [1 \ 2 \ 1]^T // 4 \\ A_5 \text{ filtering} : A'_5 = [A'_4 \ A_5 \ A'_6] \cdot [1 \ 2 \ 1]^T // 4 \\ A_6 \text{ filtering} : A'_6 = [A_5 \ A_6 \ B_7] \cdot [1 \ 2 \ 1]^T // 4 \end{array} \right.$$

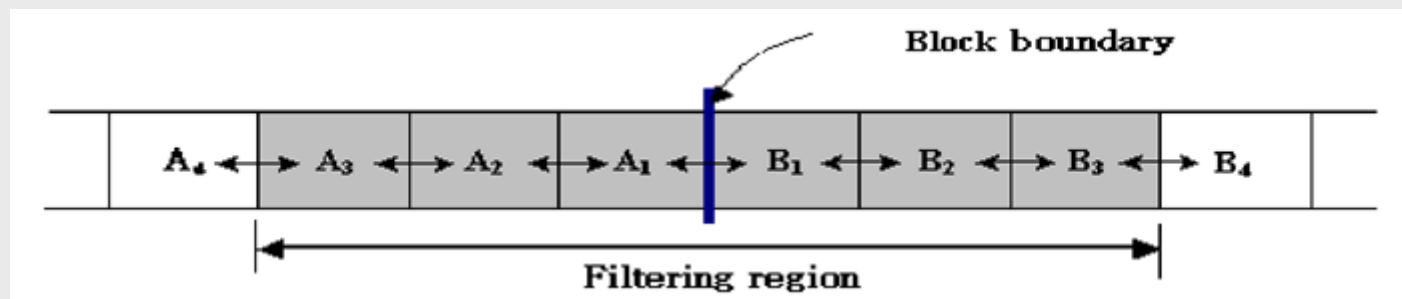
Adaptive LowPassFilter(5)

Фильтрация по горизонтали и вертикали



Adaptive LowPassFilter(6)

Фильтрация по горизонтали и вертикали



$$\begin{cases} A_1 \text{ filtering} : A'_1 = [A_2 \ A_1 \ B_1] \cdot [1 \ 2 \ 1]^T // 4 \\ A_2 \text{ filtering} : A'_2 = [A_3 \ A_2 \ A'_1] \cdot [1 \ 2 \ 1]^T // 4 \\ A_3 \text{ filtering} : A'_3 = [A_4 \ A_3 \ A'_2] \cdot [1 \ 2 \ 1]^T // 4 \end{cases}$$

Adaptive LowPassFilter(7)

Рекомендуемые пороги



For intra-frame:

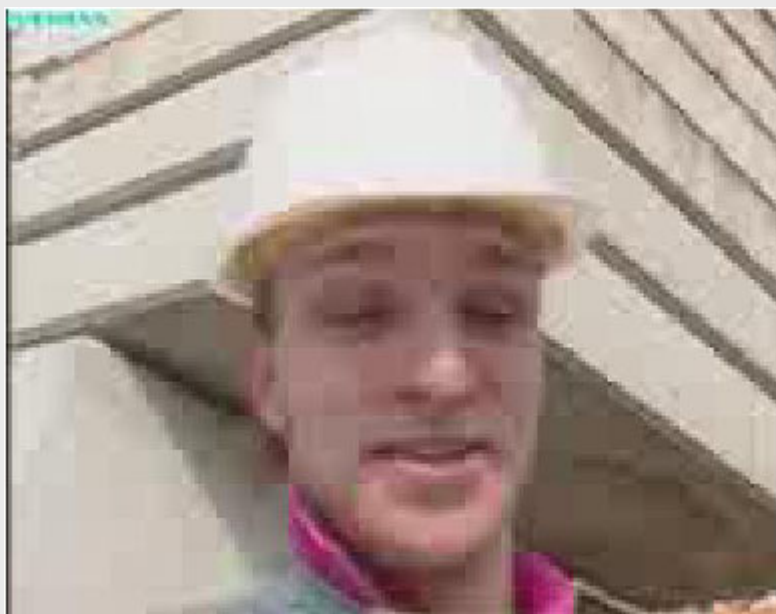
$$\left\{ \begin{array}{l} \text{Horizontal, vertical direction} \left\{ \begin{array}{l} TH\ 1 = \lfloor 2.5 \times QP \rfloor \\ TH\ 2 = \lfloor 1.5 \times QP \rfloor \end{array} \right. \\ \text{diagonal direction : } TH = 2 \times QP \end{array} \right.$$

For inter-frame:

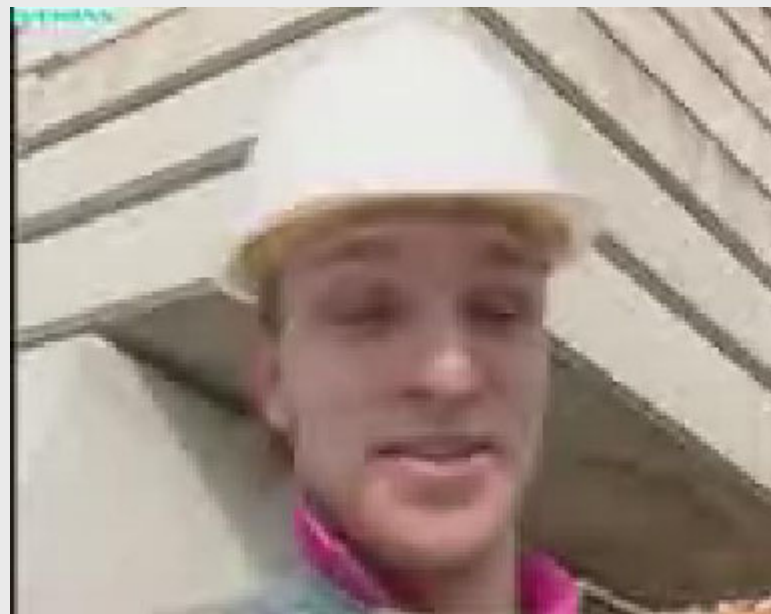
$$\left\{ \begin{array}{l} \text{Horizontal, vertical direction} \left\{ \begin{array}{l} TH\ 1 = 2 \times QP \\ TH\ 2 = 1 \times QP \end{array} \right. \\ \text{diagonal direction : } TH = 2 \times QP \end{array} \right.$$

Adaptive LowPassFilter(8)

Обработка intra-frame'ов



Intra-frame



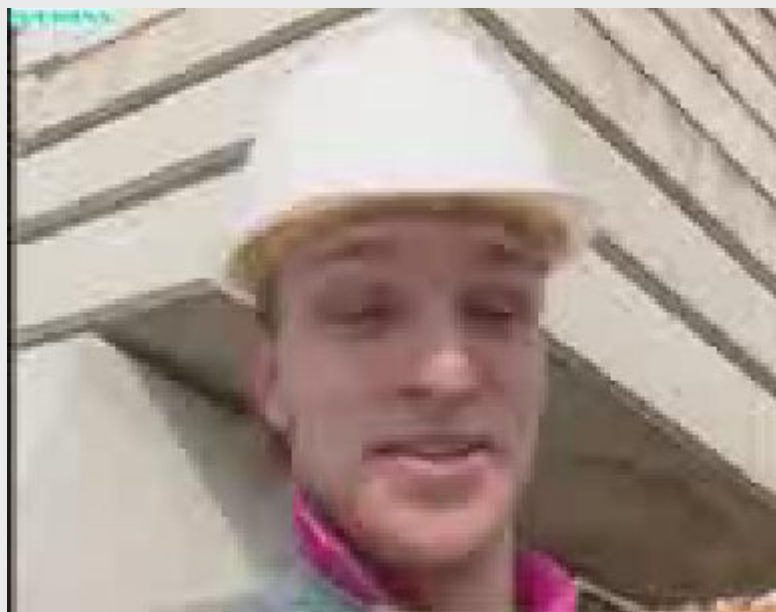
Deblocked frame

Adaptive LowPassFilter(9)

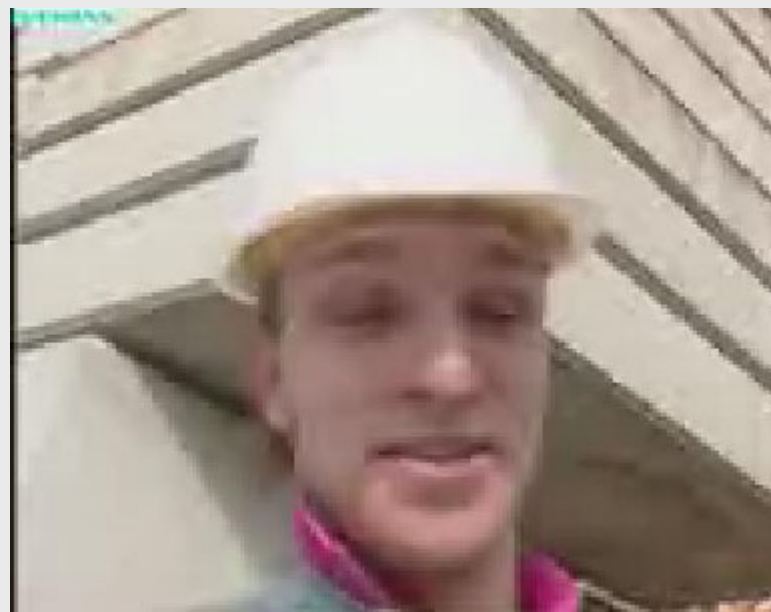
Сравнение с H.263 post-processing



Intra-frame



H.263 post-processing



Proposed

Adaptive LowPassFilter(10)

Обработка inter-frame'ов



Inter-frame



Deblocked frame

Adaptive LowPassFilter(11)

Сравнение с H.263 post-processing



Inter-frames

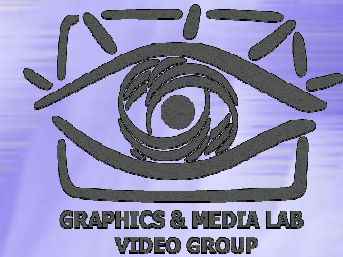


H.263 post-processing



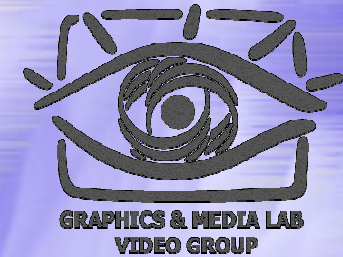
Proposed

Deblocking: Outline



- ◆ Понятие блокинга и возможные стратегии деблокинга
- ◆ Алгоритмы деблокинга
 - 4-х пиксельный
 - 6-ти пиксельный метод
 - Алгоритм фильтра Ffdshow
 - Modeled Low Pass Filter
 - Adaptive Low Pass Filter
- ◆ Алгоритм MSU Media Lab
- ◆ Сравнение описанных алгоритмов

MSU Deblocking

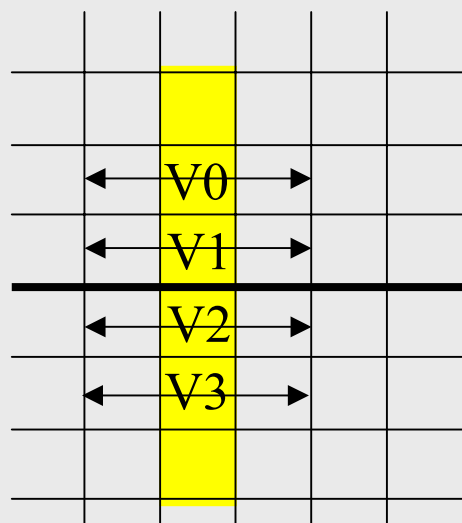


Идея алгоритма:

- Использование метрики блочности
- Для некоторых кадров вычисляется среднее значение метрики
- В зависимости от вычисленного среднего значения выбирается алгоритм обработки текущего кадра
- Каждый пиксель обрабатывается согласно значению метрики в данной точке

MSU Deblocking (2)

Алгоритм вычисления метрики



V0...V3 - векторы

$$A = V0 - V1$$

$$C = V1 - V2$$

$$B = V2 - V3$$

$$D = C - (A+B) / 2$$

$$M = \text{abs}(D[0] + D[1] + D[2])$$

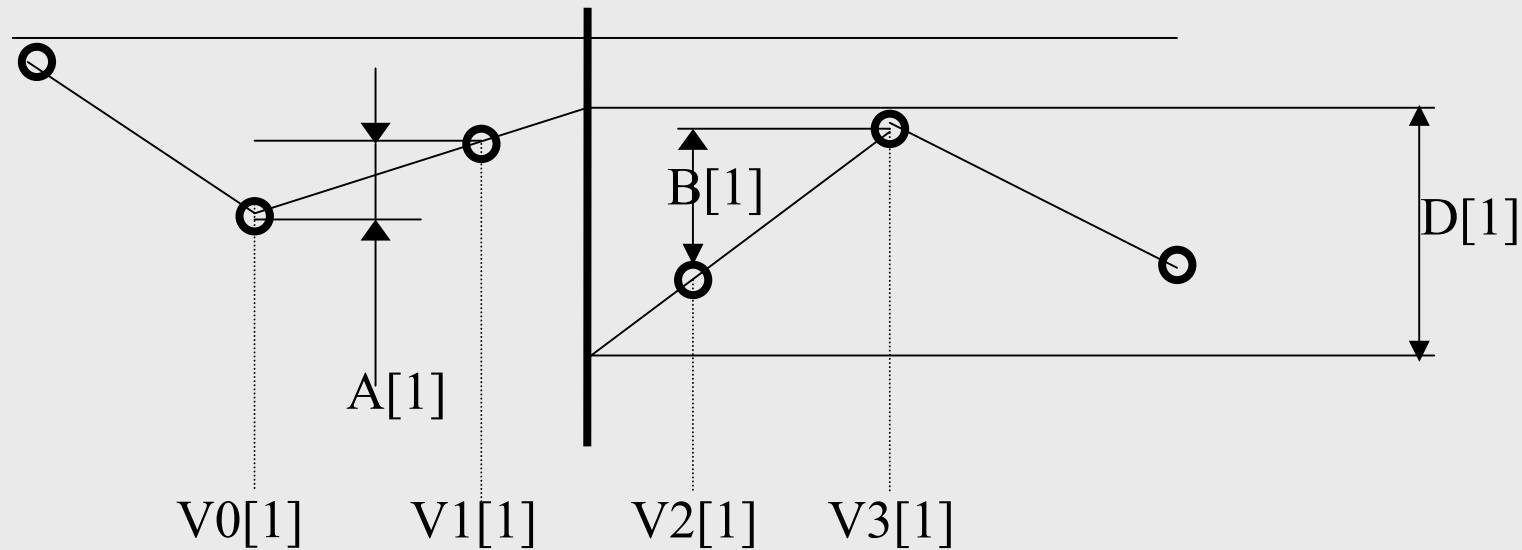
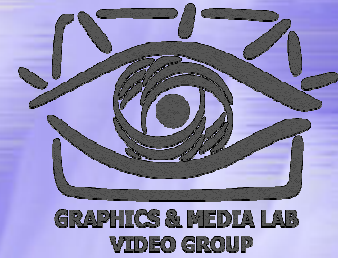
$$W1 = W(\text{abs}(A[0]) + \text{abs}(B[0]))$$

$$W2 = W(\text{abs}(A[1]) + \text{abs}(B[1]))$$

$$W3 = W(\text{abs}(A[2]) + \text{abs}(B[2]))$$

$$\text{Metric} = M * (W1 + W3) * W2$$

MSU Deblocking (3)



$$A = V0 - V1$$

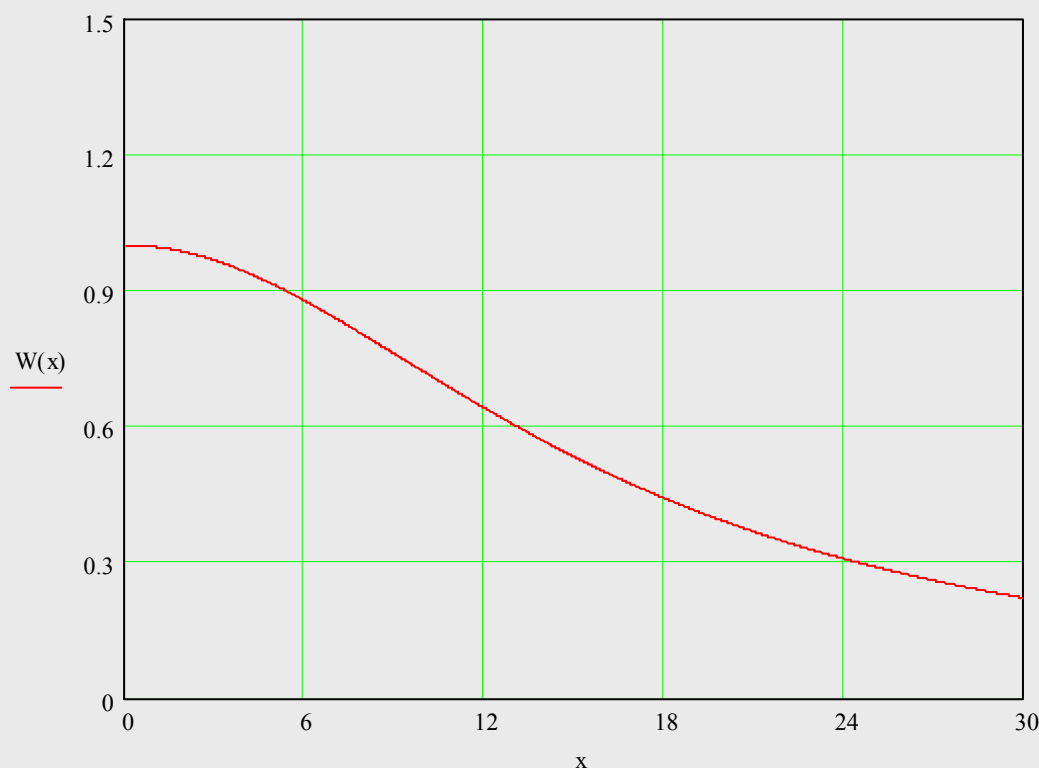
$$C = V1 - V2$$

$$B = V2 - V3$$

$$D = C - (A+B) / 2$$

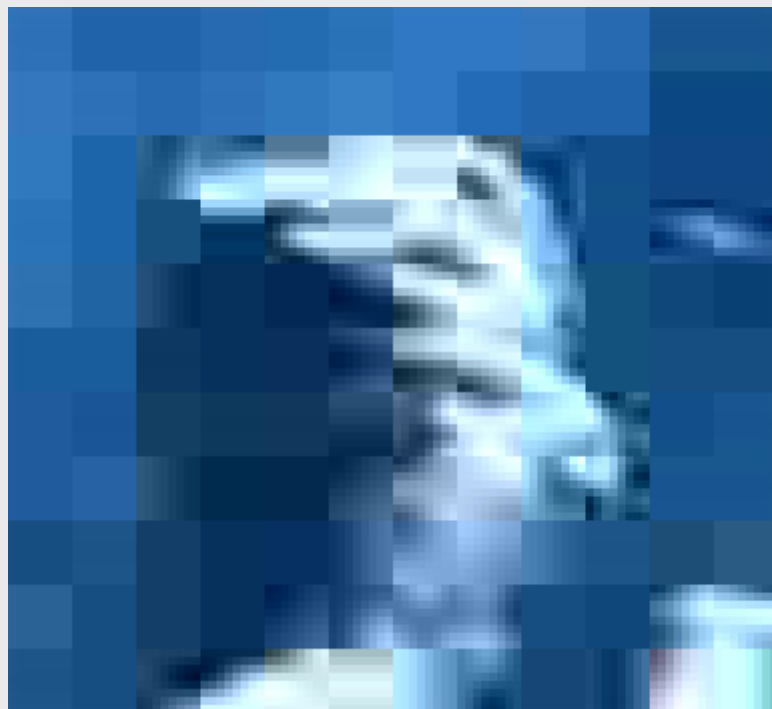
MSU Deblocking (4)

Функция для вычисления веса



- ✓ При малых x убывает медленно
- ✓ При достаточно больших x стремится к нулю

MSU Deblocking (5)



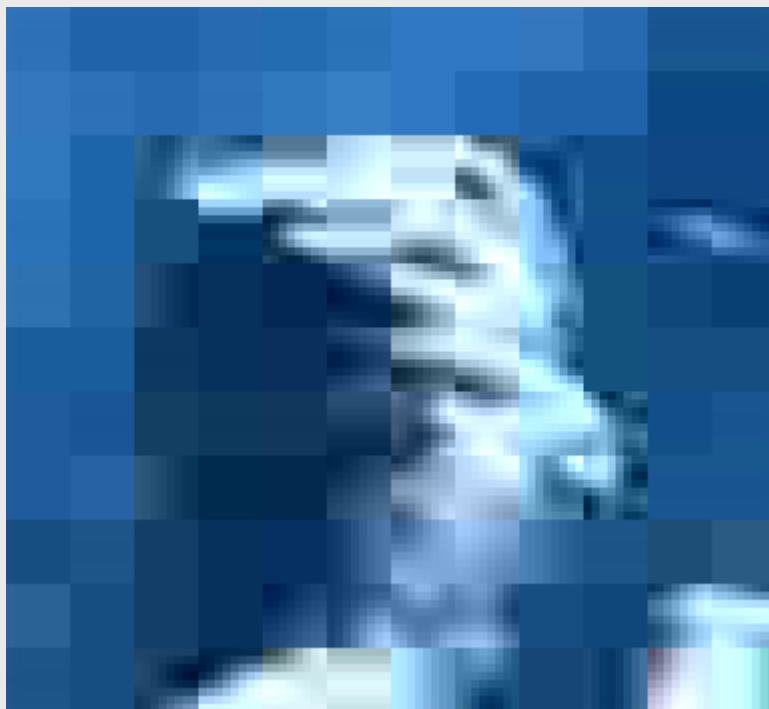
Battle.avi



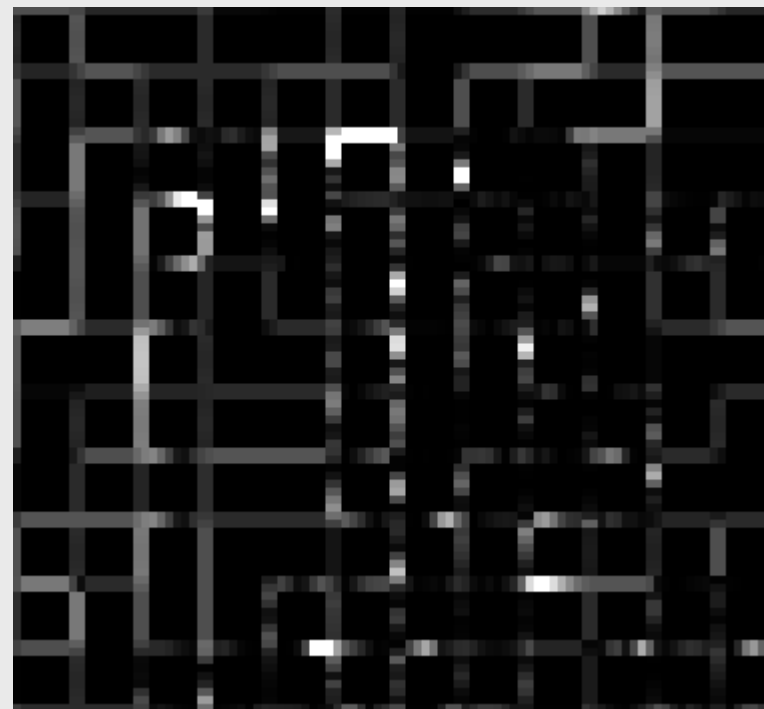
MSU

MSU Deblocking (6)

Метрика



Battle.avi



MSU-metric

MSU Deblocking (7)

Сравнение с методом из ffdshow



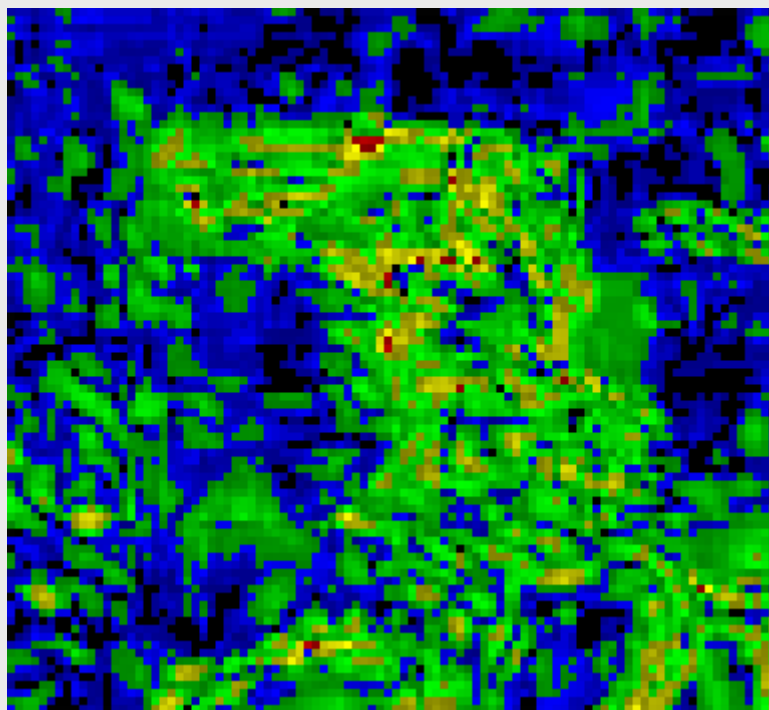
MSU



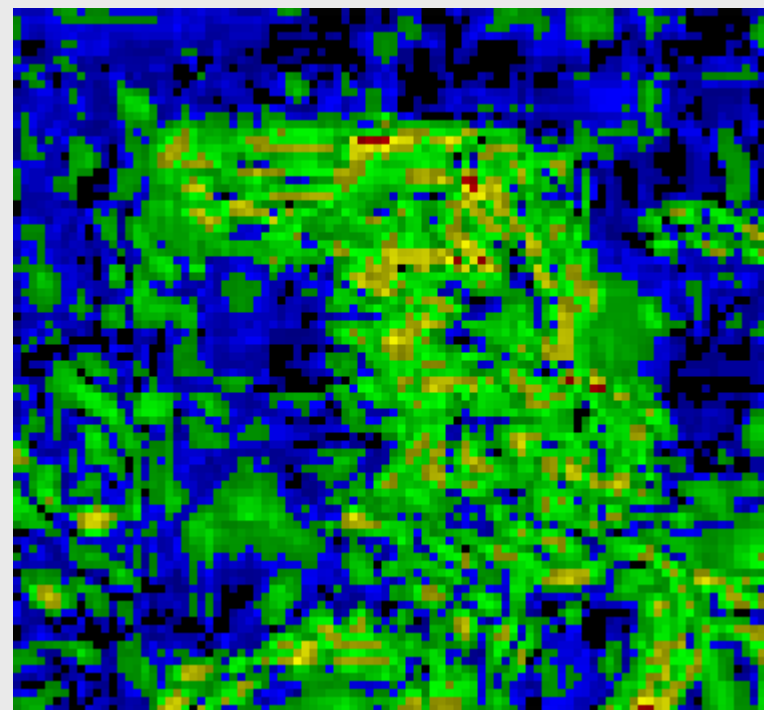
Ffdshow: T = 60

MSU Deblocking (8)

Сравнение с методом из ffdshow

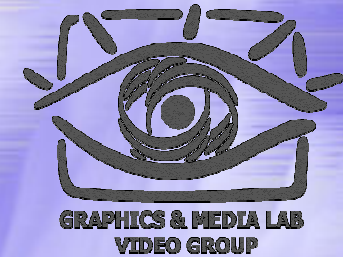


FFdshow: Y-PSNR = 32.15



MSU: Y-PSNR = 32.17 db

Deblocking: Outline



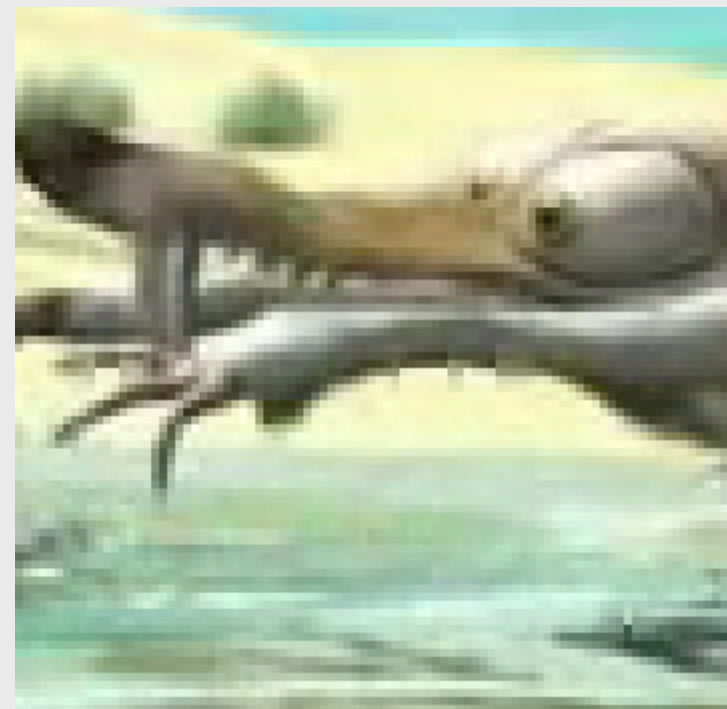
- ◆ Понятие блокинга и возможные стратегии деблокинга
- ◆ Алгоритмы деблокинга
 - 4-х пиксельный
 - 6-ти пиксельный метод
 - Алгоритм фильтра Ffdshow
 - Modeled Low Pass Filter
 - Adaptive Low Pass Filter
- ◆ Алгоритм MSU Media Lab
- ◆ Сравнение описанных алгоритмов

Сравнение алгоритмов

4-х и 6-ти пиксельные методы



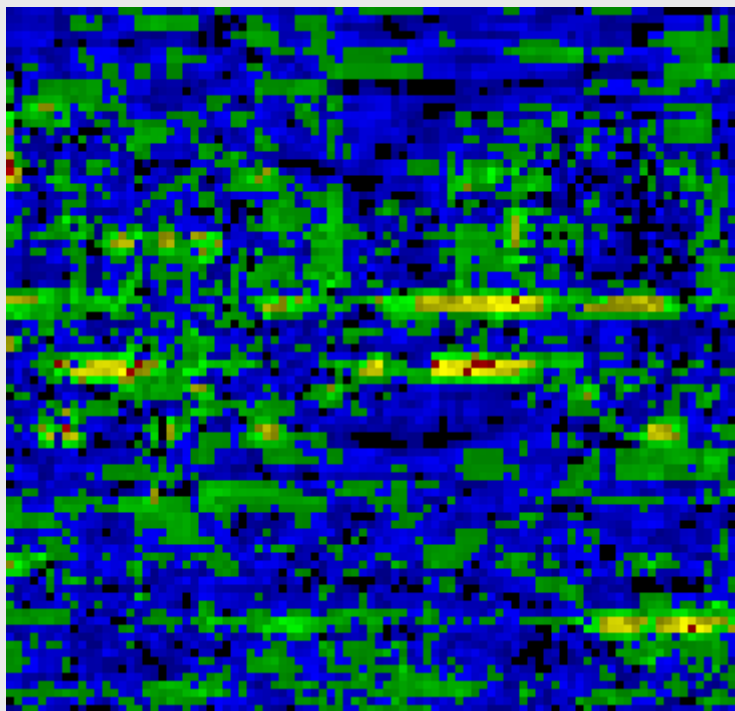
4-Pixel: T = 80



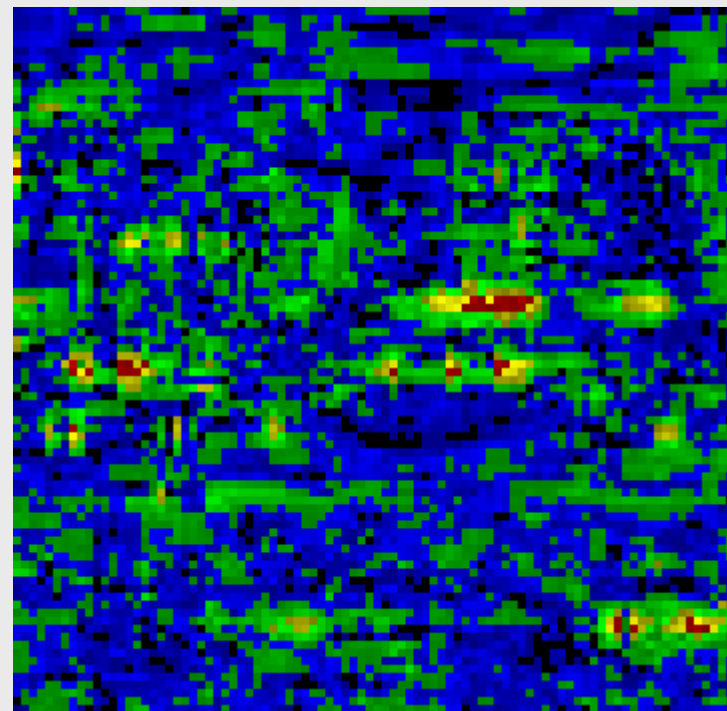
6-Pixel: T = 60

Сравнение алгоритмов (2)

4-х и 6-ти пиксельные методы



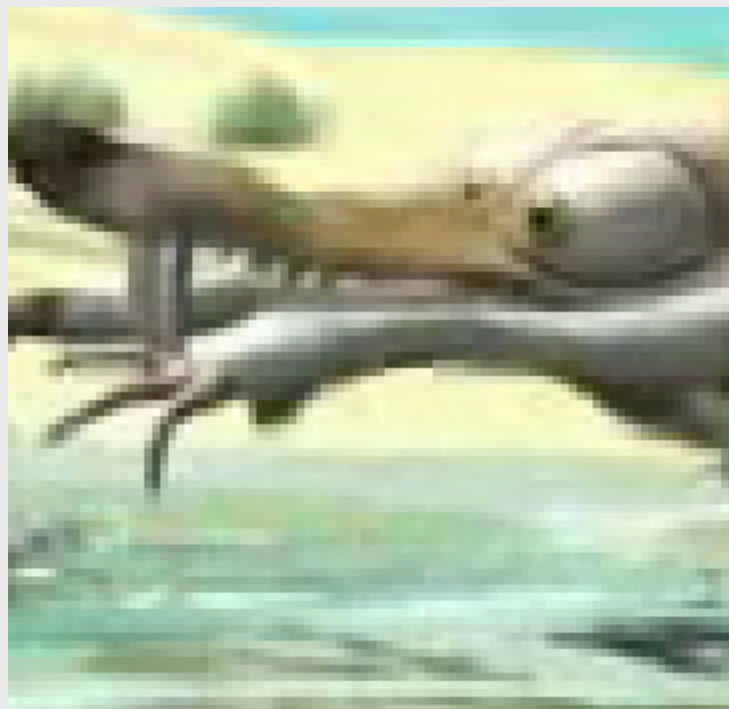
4-Pixel: Y-PSNR = 34.77



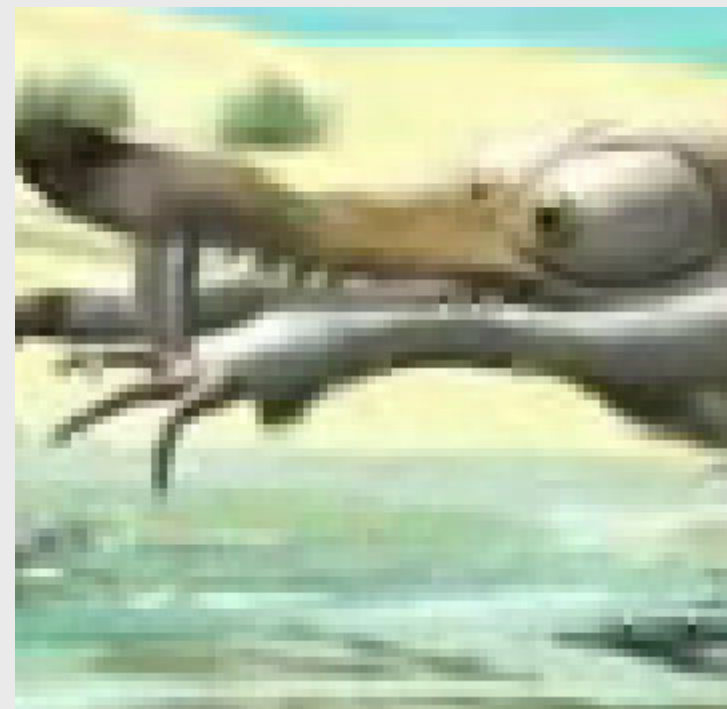
6-Pixel: Y-PSNR = 34.70

Сравнение алгоритмов

4-х пиксельный и метод ffdshow



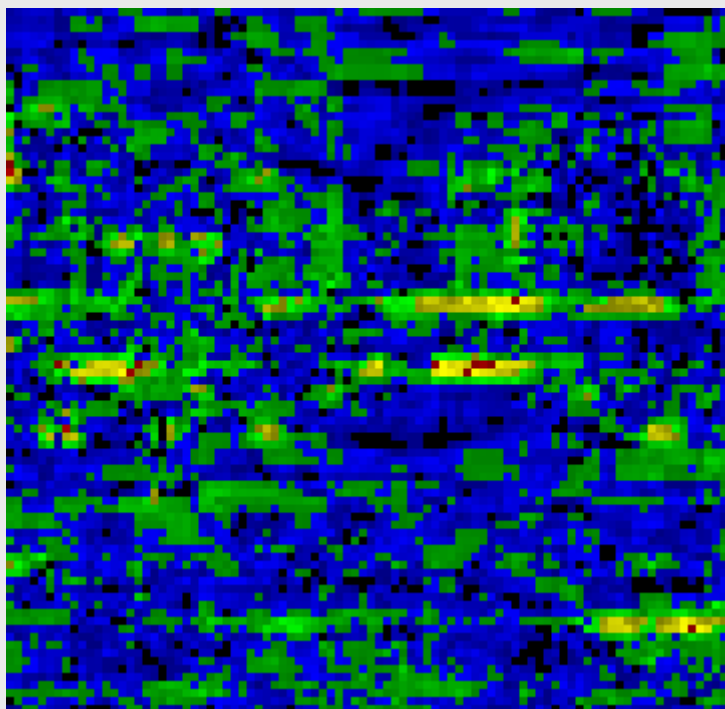
4-Pixel: T = 80



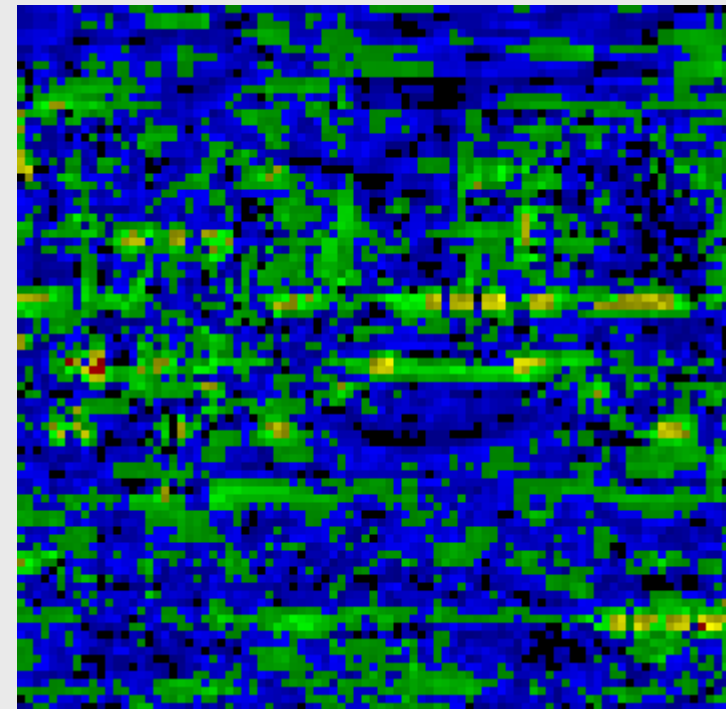
Ffdshow: T =80

Сравнение алгоритмов (2)

4-х пиксельный и метод ffdshow



4-Pixel: Y-PSNR = 34.77



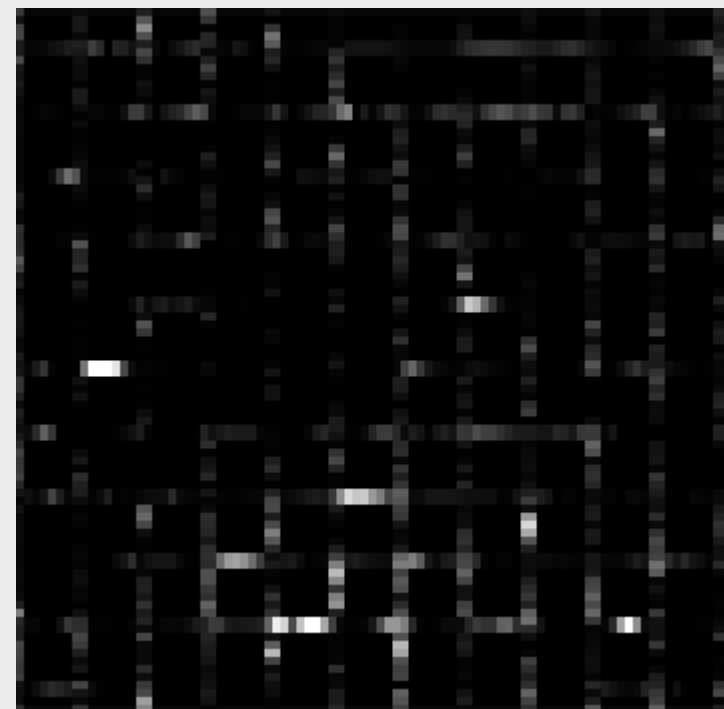
Ffdshow: Y-PNSR = 34.83

Сравнение алгоритмов

Алгоритм MSU



IceAge.avi



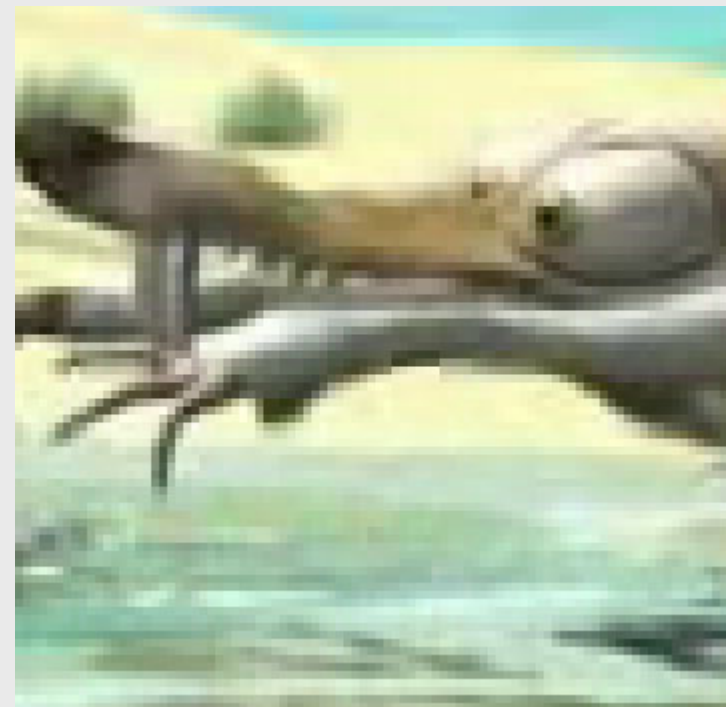
MSU-metric

Сравнение алгоритмов

Алгоритм MSU и 4-х пиксельный



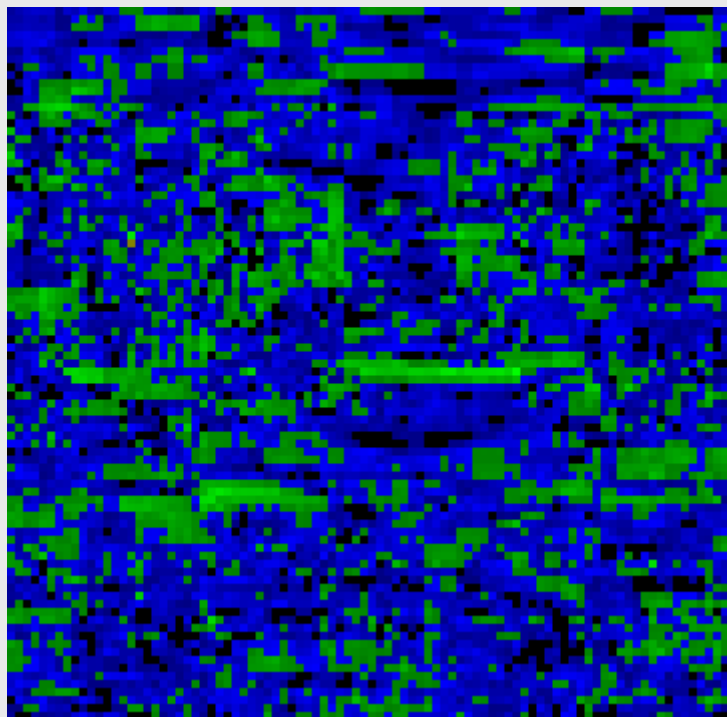
MSU



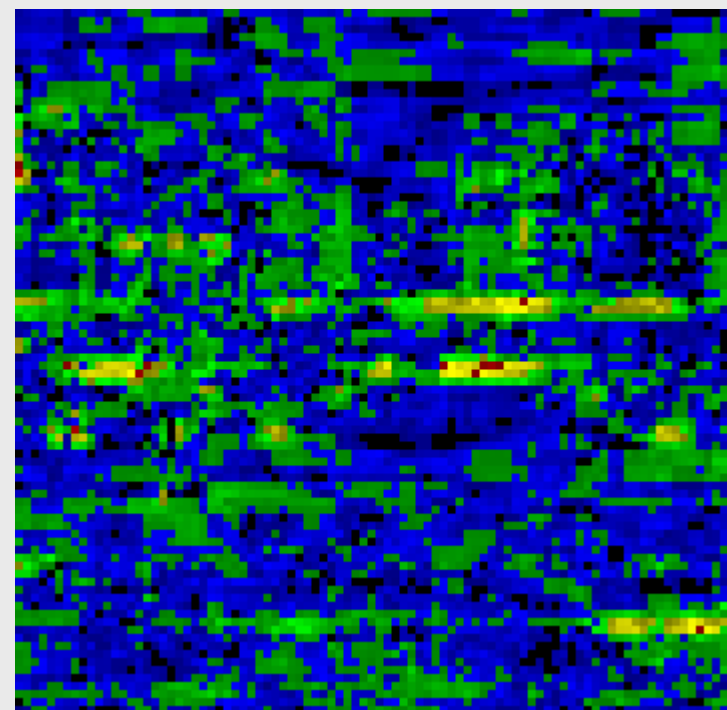
4-Pixel: T = 80

Сравнение алгоритмов

Алгоритм MSU



MSU: Y-PSNR = 36.69 db



4-Pixel: Y-PSNR = 34.77

Сравнение алгоритмов

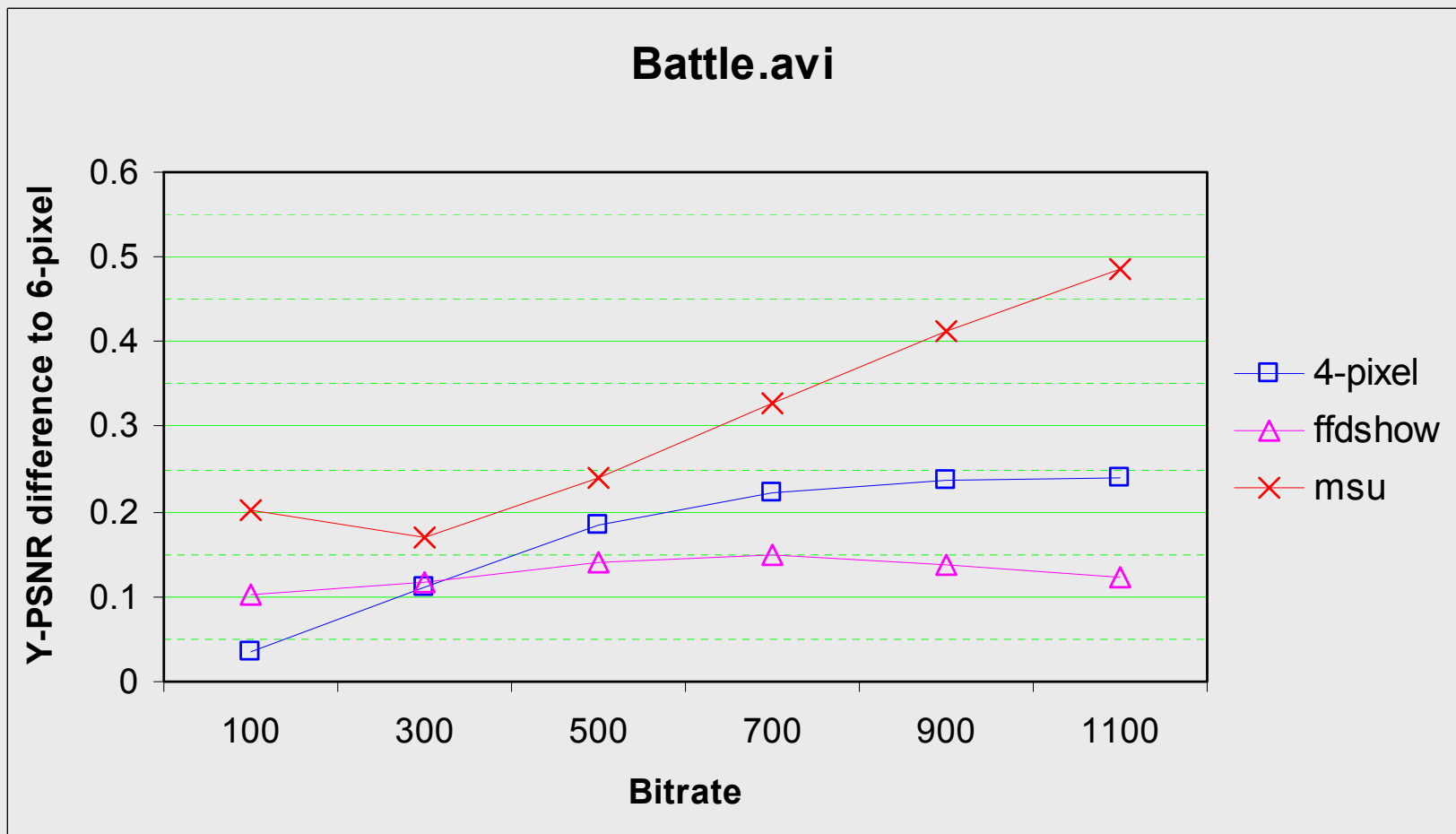
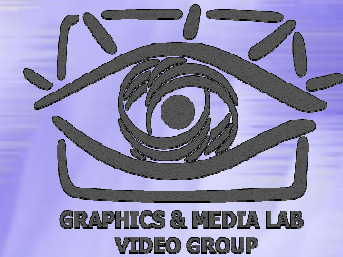
Методика сравнения



1. Видео последовательности сжимаются на разных брейтах алгоритмом без деблокинга
 - ◆ Битрейты – 100, 300, 500, 700, 900, 1100
 - ◆ Кодек – Xvid 1.9
2. Каждый ролик обрабатывается различными фильтрами деблокинга
3. Для каждого метода в зачет идет тот параметр, при котором было достигнуто максимальное значение метрики Y-PSNR
4. Измеряется PSNR по сравнению с оригинальным, несжатым фильмом
5. Для каждого метода строится отдельная ветвь графика

Сравнение алгоритмов

Разница с 6-ти пиксельным методом



Сравнение алгоритмов

Разница с 6-ти пиксельным методом

