Нерегулярный Параллелизм в Обработке Цифровых Сигналов.

- Лекторы:
  - Боресков А.В. (ВМиК МГУ)
  - Харламов А.А. (NVidia)

План

- Преобразование Фурье
- Шумоподавление
  - Медианная фильтрация
  - Bilateral
  - Non Local Means
- Масштабирование
  - Линейные методы
  - Адаптивные

Преобразование Фурье

- Линейный оператор вида:

$$F(u)=\int_{-\infty}^{+\infty} f(x)e^{-2\pi i\cdot xu}dx \qquad F(u,v)=\iint_{-\infty}^{+\infty} f(x,y)e^{-2\pi i(ux+vy)}dxdy$$

- Обратный оператор:

$$f(x)=\int_{-\infty}^{+\infty} F(u)e^{2\pi i\cdot xu}du \qquad f(x,y)=\iint_{-\infty}^{+\infty} F(u,v)e^{2\pi i(ux+vy)}dudv$$

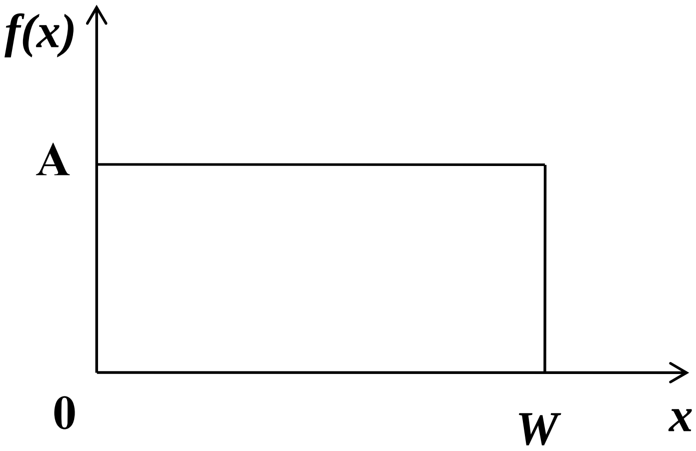
Преобразование Фурье

⌘Условие существования

1. $$\int_{-\infty}^{+\infty} | f(x) | \, dx < \infty \qquad \int_{-\infty}^{+\infty}\int | f(x, y) | dxdy < \infty$$

2. Конечное число устранимых разрывов

Преобразование Фурье
Пример 1D



$$F(u) = \int_{-\infty}^{+\infty} f(x) e^{-2\pi i \cdot xu} dx = A \int_{0}^{W} e^{-2\pi i \cdot xu} dx$$

$$= \frac{-A}{2\pi i u} \left[e^{-2\pi i \cdot xu} \right]_0^W = \frac{-A}{2\pi i u} \left[e^{-2\pi i \cdot Wu} - 1 \right]$$

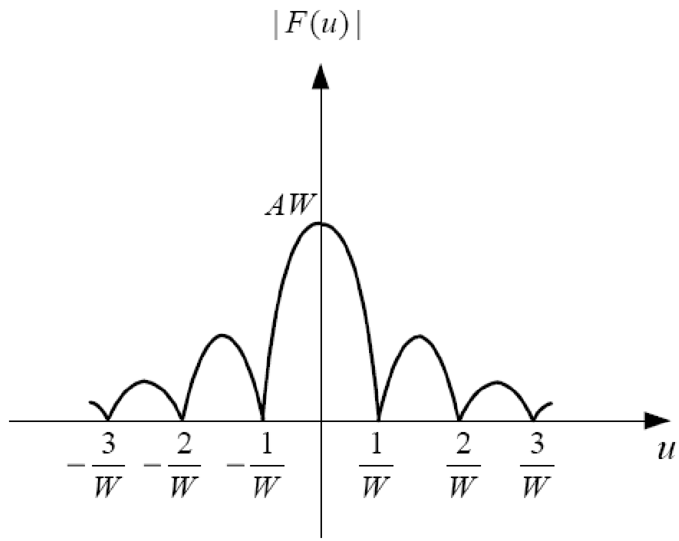
$$= \frac{-A}{2\pi i u} e^{-\pi i \cdot Wu} \left[e^{-\pi i \cdot Wu} - e^{\pi i \cdot Wu} \right]$$

$$= \frac{-A}{2\pi i u} e^{-\pi i \cdot Wu} \left[-2i \sin(\pi u W) \right]$$

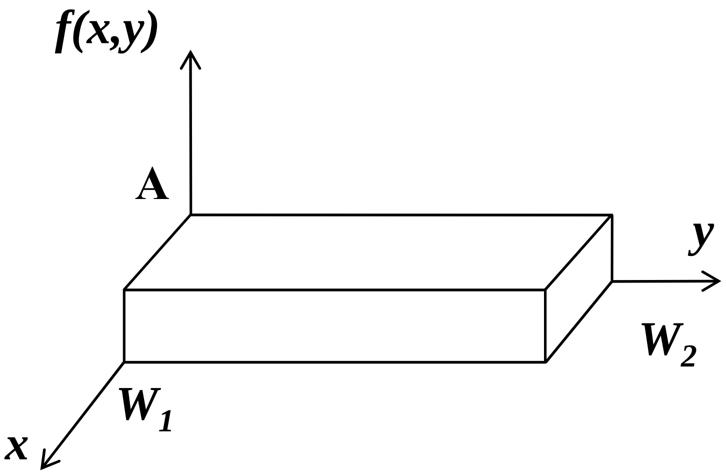
$$= AW \frac{\sin(\pi u W)}{\pi u W} e^{-\pi i \cdot Wu}$$

$$= AW \sin c(uW) e^{-\pi i \cdot Wu}$$

$$| F(u) | = AW \sin c(uW)$$



Преобразование Фурье Пример 2D



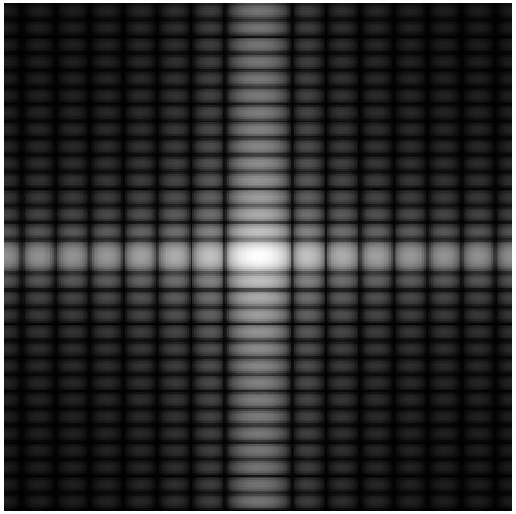
$$F(u,v) = \iint_{-\infty}^{+\infty} f(x,y)e^{-2\pi i(ux+vy)}dxdy$$

$$= A\int_0^{W_1} e^{-2\pi iux}dx \int_0^{W_2} e^{-2\pi ivy}dy$$

$$= A\left[\frac{e^{-2\pi iux}}{-2\pi iu}\right]_0^{W_1}\left[\frac{e^{-2\pi ivy}}{-2\pi iv}\right]_0^{W_2}$$

$$= AW_1W_2 \sin c(uW_1)\sin c(vW_2)e^{-\pi i(uW1+vW2)}$$

$$|F(u,v)| = AW_1W_2\,|\sin c(uW_1)|\,|\sin c(vW_2)|$$



Преобразование Фурье Свойства

1. $f(x,y) = f_1(x) f_2(y) \Rightarrow F(u,v) = F_1(u) F_2(v)$
2. $F\{f^*(x,y)\} = F^*(-u,-v)$
3. $f(x) \in R \Rightarrow |F(u)| = |F^*(-u)|$
4. $f(x,y) \in R \Rightarrow |F(u,v)| = |F^*(-u,-v)|$
5. $F\{f(-x,-y)\} = F(-u,-v)$
6. $F\{f(ax,by)\} = \dfrac{F(u/a, v/b)}{|ab|}$
7. $F\{f(r, \theta + \theta_0)\} = F(w, \phi + \theta_0)$

Преобразование Фурье Свойства

1. $F\{f(x,y)\otimes h(x,y)]\} = F(u,v)H(u,v)$

   $F\{f(x,y)h(x,y)]\} = F(u,v)\otimes H(u,v)$

2. $\dfrac{\partial f(x,y)}{\partial x} = F^{-1}\{2\pi iuF(u,v)\}$

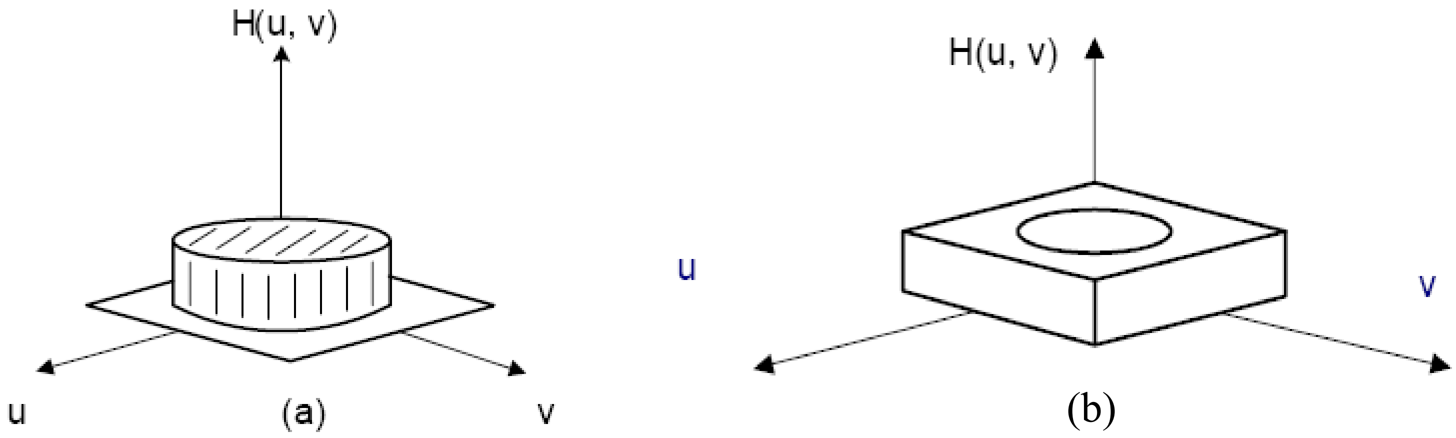
3. $F\{\Delta f(x,y)\} = -4\pi^2(u^2+v^2)F(u,v)$

4. $F(u,v) = \int\limits_{-\infty}^{+\infty}\left[\int\limits_{-\infty}^{+\infty} f(x,y)e^{-2\pi i\cdot ux}dx\right]e^{-2\pi i\cdot vy}dy = \int\limits_{-\infty}^{+\infty} F(u,y)e^{-2\pi i\cdot vy}dy$

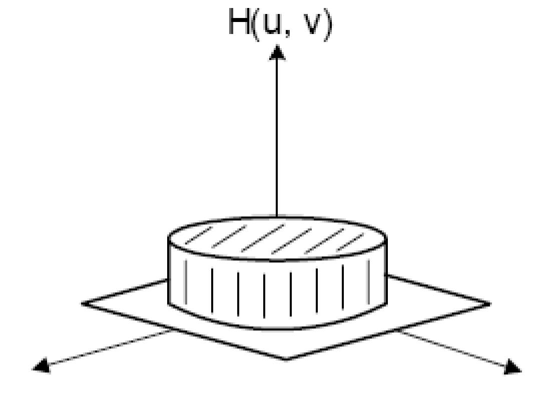
5. $F\{f(x)\}\in C$

Фильтры

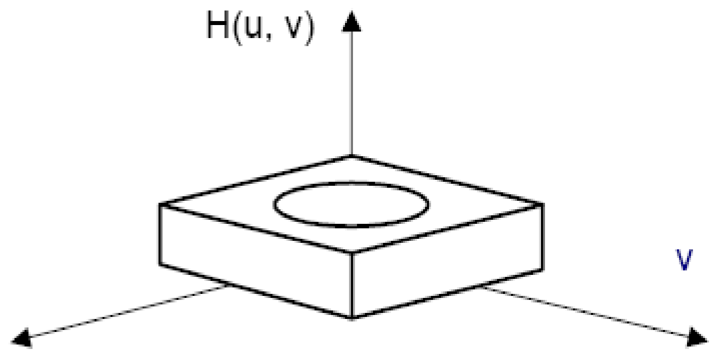
a) Низкочастотные (low-pass)
b) Высокочастотные (high-pass)



Фильтры



(a)



(b)

Image Denoising

- Шумы в изображении
  - Импульсный
    - Salt & pepper
  - Аддитивный
    - Uniform
    - Gaussian

Ранговые фильтры

- Алгоритм Р.Ф. ранга N:
  - Для каждого отсчета сигнала *i*
  - Выбор окрестности вокруг отсчета *i*
    - Сортируем по значению
    - Выбираем N-ое значение как результат

Медиана

- Ранговый фильтр N=0.5
- Сортировка не обязательна для 8bit значений
  - Строим гистограмму

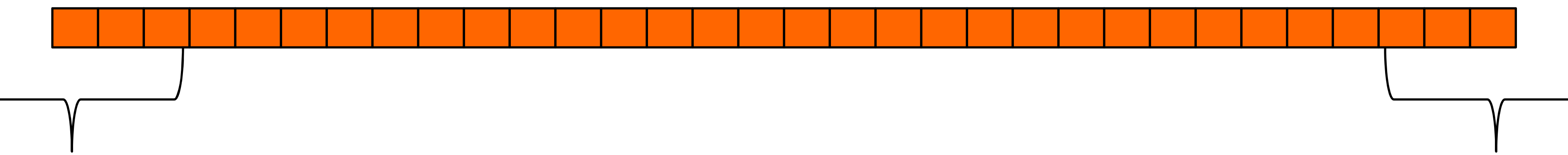
```
for(int i<-R; i<R; i++) h[ signal[i] ]++;
```

  - Сканируем Гистограмму:

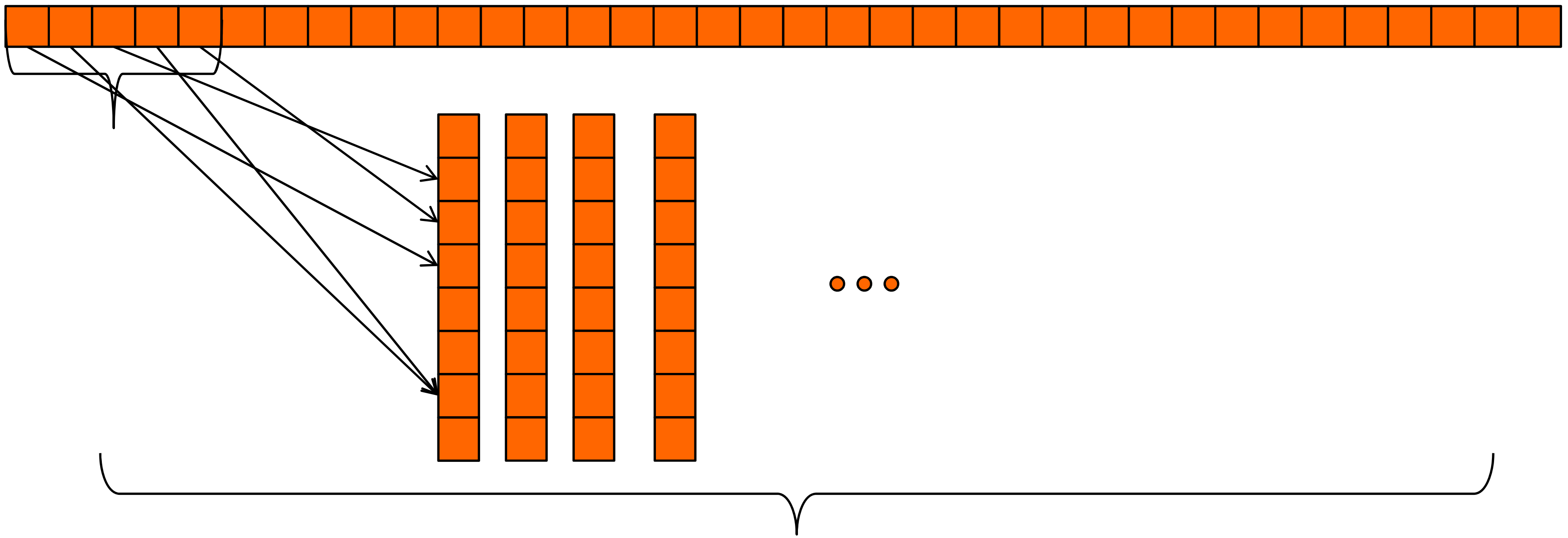
```
int sum = 0;
int targetSum = N*rank;
for(int i<0; i<256; i++)
{
    sum += h[i];
    if (sum > targetSum) return i;
}
```

Медиана
Построение гистограммы

сигнал



Сигнал + фартук в Smem



Что с банк-конфликтами?

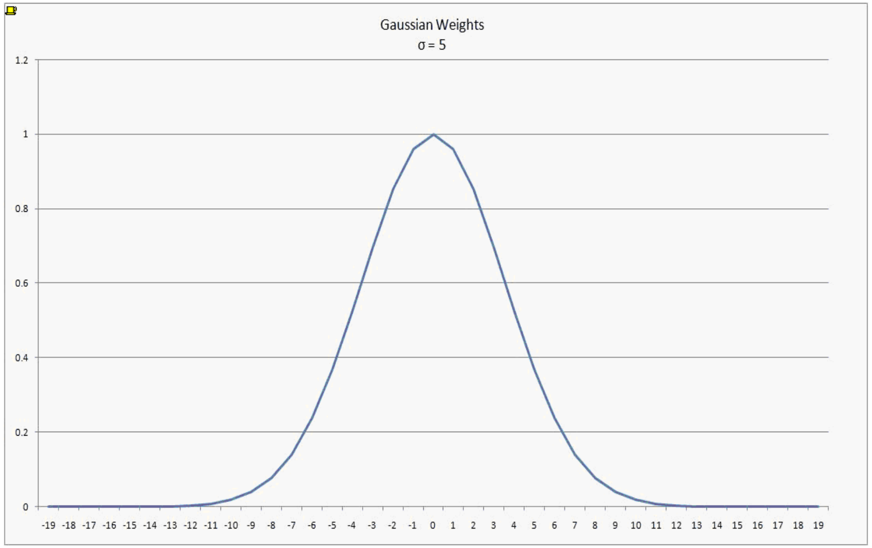
Фильтрация (Аддитивный шум)

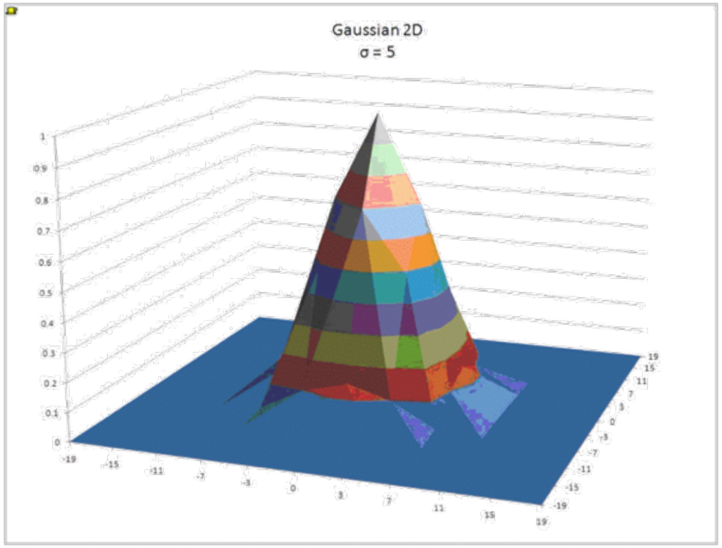
- Размытие – это low-pass фильтр
- Каким должен быть фильтр?
  - Подавлять шум?
  - Сохранять детальность?

Gaussian Blur

- Blur (размытие) изображение
- Свертка с ядром:

$$k_\sigma(i) = \exp(-i^2 / \sigma^2) \qquad k_\sigma(i,j) = \exp(-(i^2 + j^2)/\sigma^2)$$



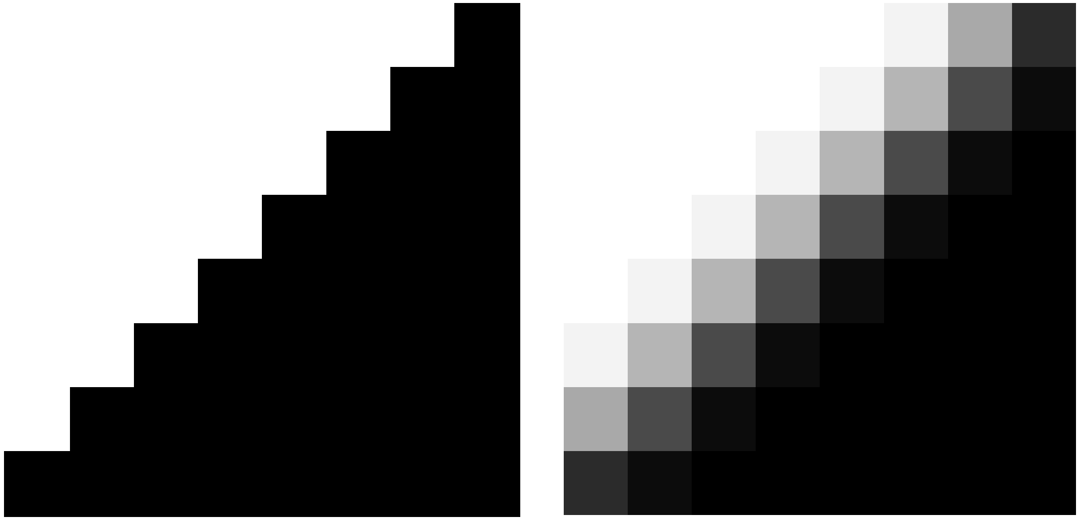


Адаптивное размытие

- Свертка с ядром:

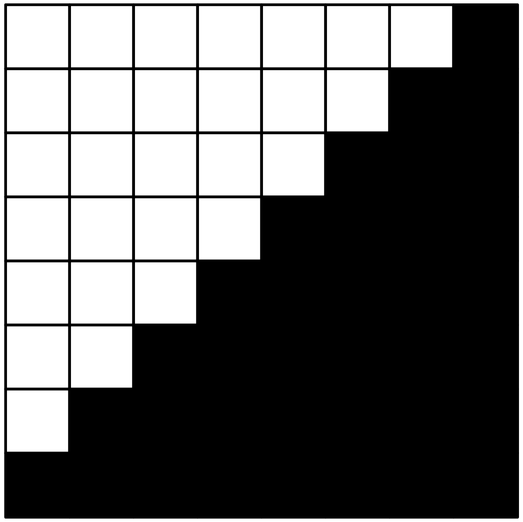
$$k_\sigma(i, j) = \exp(-(i^2 + j^2)/\sigma^2)\exp(-ClrSpaceDist(i, j)/h^2)$$

- *ClrSpaceDist* – это фотометрическая близость

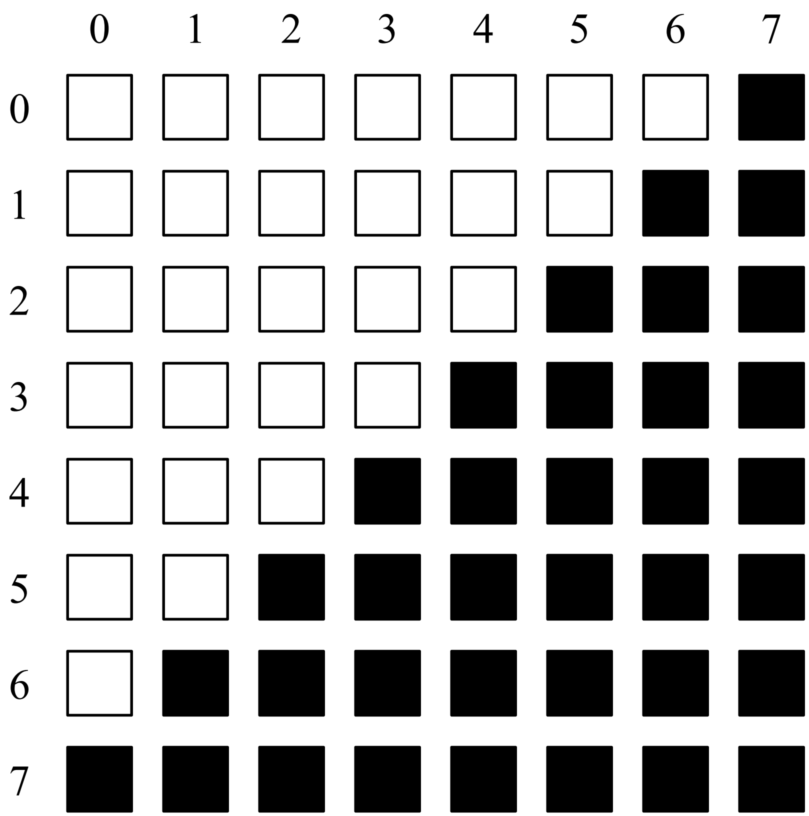


Bilateral

$$W_{\mathbf{c}}(\mathbf{y}) = e^{-\frac{|\mathbf{y}-\mathbf{c}|^2}{r^2}} e^{-\frac{|u(\mathbf{y})-u(\mathbf{c})|^2}{h^2}}$$



Bilateral



$$W_{\mathbf{c}}(\mathbf{y}) = e^{-\frac{|\mathbf{y}-\mathbf{c}|^2}{r^2}} e^{-\frac{|u(\mathbf{y})-u(\mathbf{c})|^2}{h^2}}$$

$$= e^{\frac{-((0-4)^2+(0-4)^2)}{3^2}} e^{-\frac{(1-0)^2}{h^2}}$$

$$W_{\mathbf{c}}(\mathbf{y}) = e^{-\frac{|\mathbf{y}-\mathbf{c}|^2}{r^2}} e^{-\frac{|u(\mathbf{y})-u(\mathbf{c})|^2}{h^2}}$$

$$= e^{\frac{-((7-4)^2+(7-4)^2)}{3^2}} e^{-\frac{(1-1)^2}{h^2}}$$

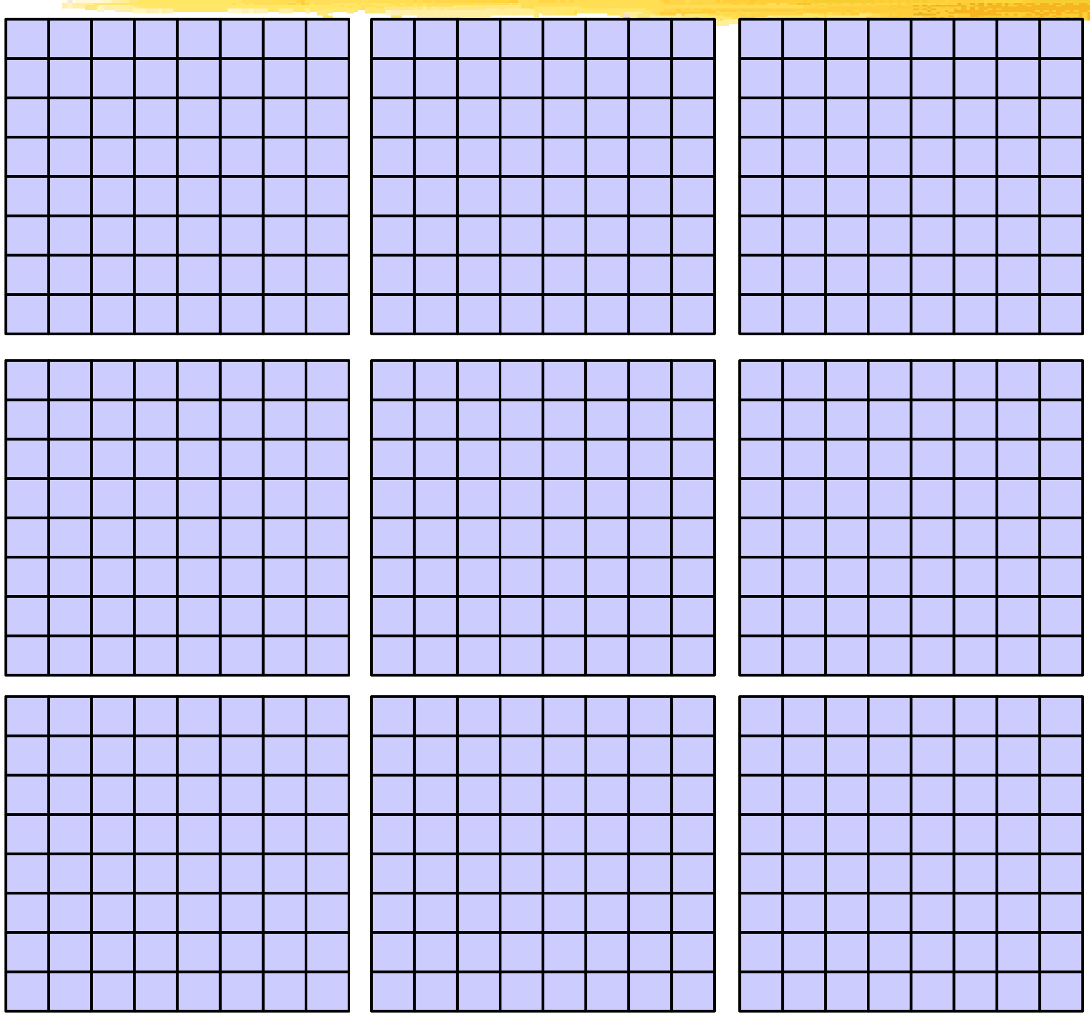
Bilateral Kernel

```
#define SQR(x) ((x) * (x))
texture<float, 2, cudaReadModeElementType> g_TexRef;

__global__ void BilateralBlur( float * pFilteredImage, int W, int H, float r)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int idy = blockIdx.y * blockDim.y + threadIdx.y;

    float wSum = 0.0f;
    float rResult = 0.0f;
    float c = tex2D(g_TexRef, idx, idy);
    for (int ix = -r; ix <= r; ix++)
        for (int iy = -r; iy <= r; iy++)
        {
             float clr = tex2D(g_TexRef, idx + ix, idy + iy);
             float w = exp( -(SQR(ix) + SQR(iy)) / SQR(r) – SQR(clr-c)/SQR(h));
             rResult += w * clr;
             wSum += w;
        }
     rResult = rResult / wSum;

     pFilteredImage[idx + idy * W] = rResult;
}
```

Bilateral
Оптимизации

- Bilateral не сепарабельный фильтр
  - Но можно его разделить
- Смешивать исходное изображение с фильтрованным
  - Если в блоке много ненулевых коэф., то с большой вероятностью в этом блоке шум был подавлен успешно
  - Если в блоке много нулевых коэф., то с большой вероятностью в блоке много деталей (границы, текстура и т.д.)

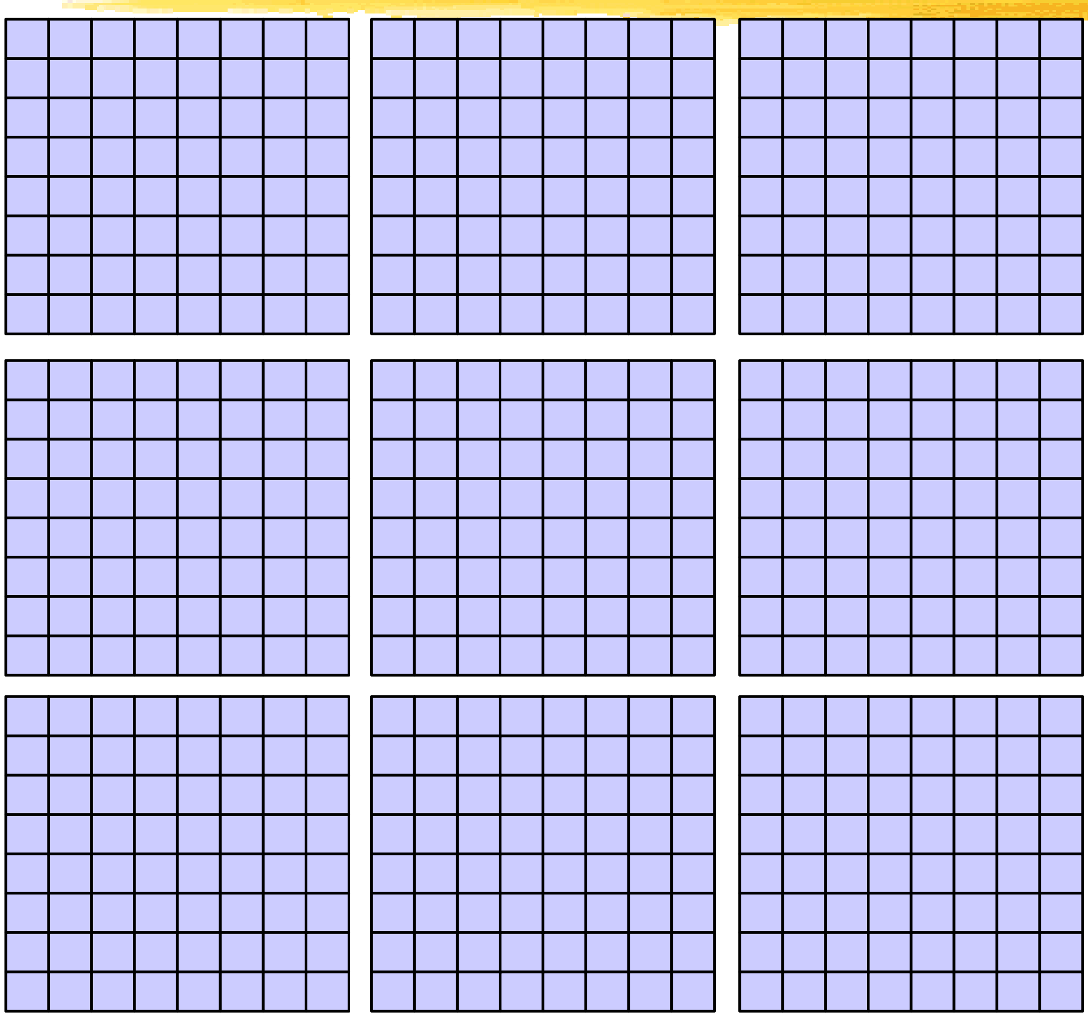
Bilateral
Оптимизации



Исходное изображение

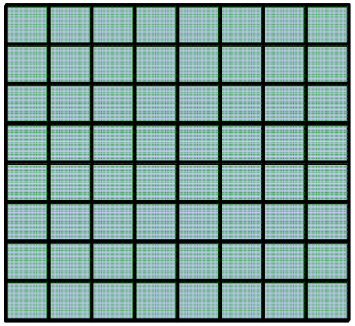
Эффективно ли такое разбиение изображения

Bilateral Оптимизации

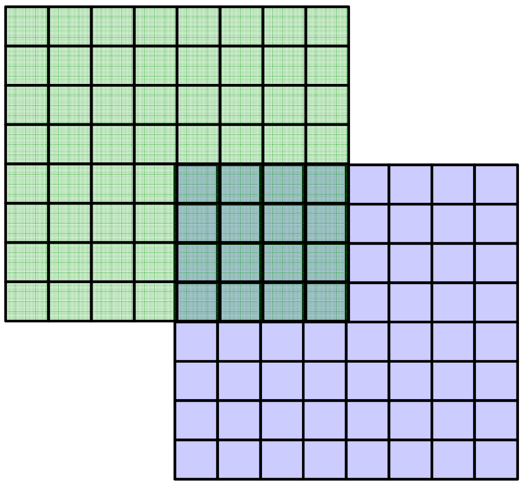


Исходное изображение

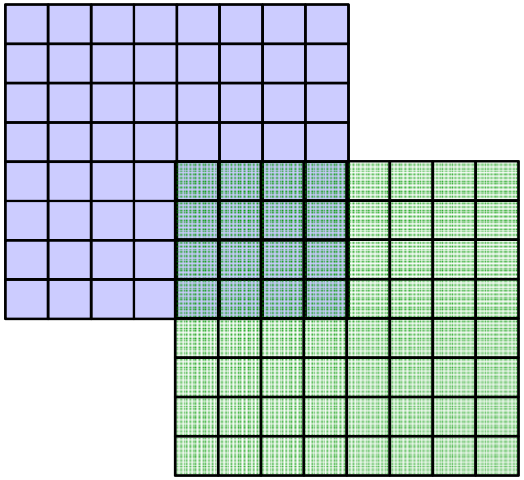
Bilateral
Smem Оптимизации



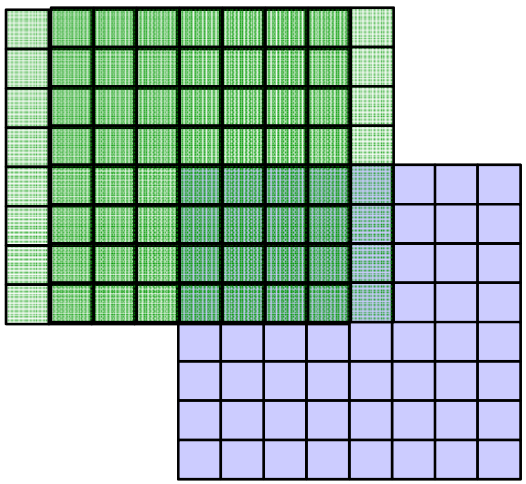
Bilateral
Smem Оптимизации



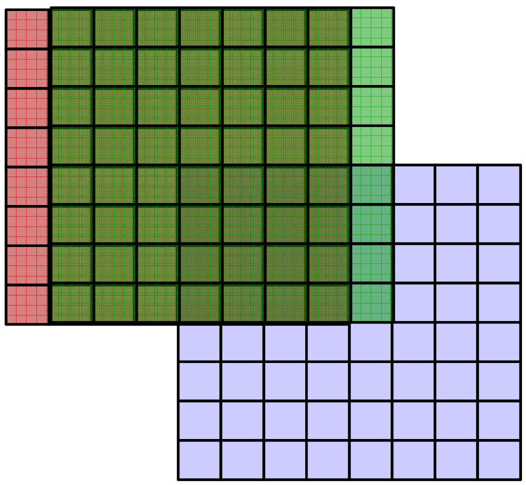
Bilateral Smem Оптимизации



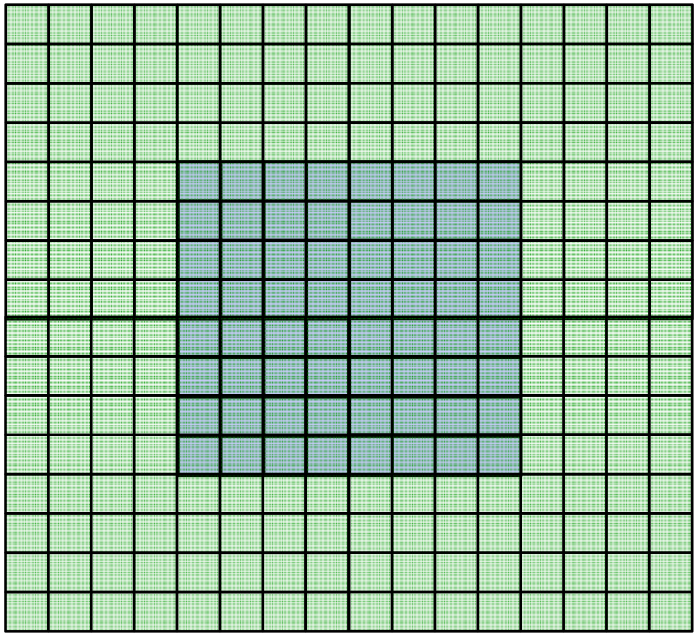
Bilateral
Smem Оптимизации



Bilateral Smem Оптимизации



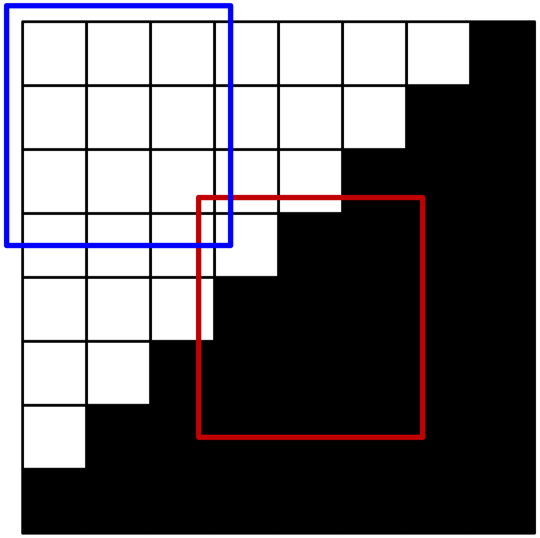
Bilateral
Smem Оптимизации



Non Local Means

- *ClrSpaceDist* – оценивать по блокам пикселей

$$W_c(y) = \rho(\mathbf{B}(\mathbf{c}), \mathbf{B}(\mathbf{y})) = \frac{1}{S(\mathbf{B})} \int_{\mathbf{B}(\mathbf{c})} |\, u(\mathbf{y} + (\mathbf{c} - \boldsymbol{\alpha})) - u(\boldsymbol{\alpha}) \,|^2 \, d\boldsymbol{\alpha}$$



Non Local Means

- На вычисление одного веса:
  - $N_b x N_b$ вычислений, N размер блока
- На фильтрацую одного пиксела:
  - $N_b x N_b$ x RxR, R размер окна

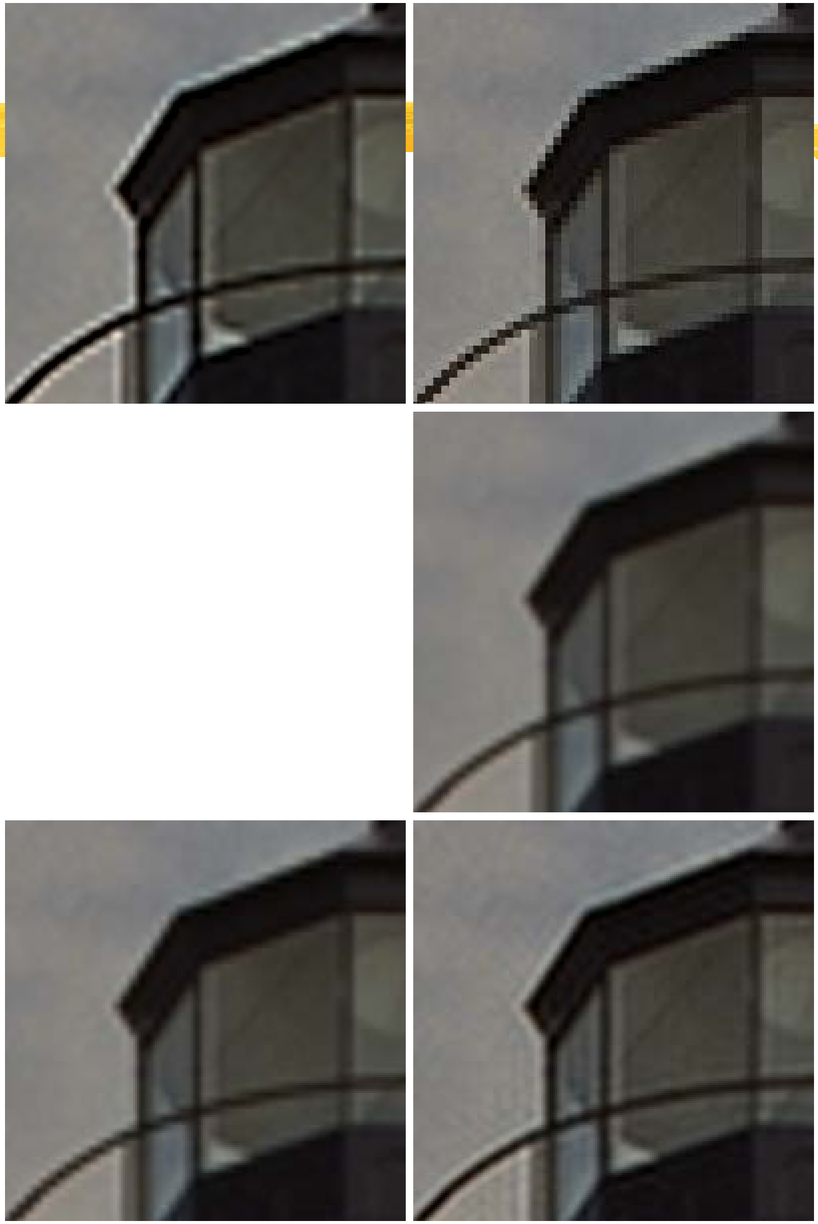
Сравнение

Сравнение Bilateral

Сравнение NLM

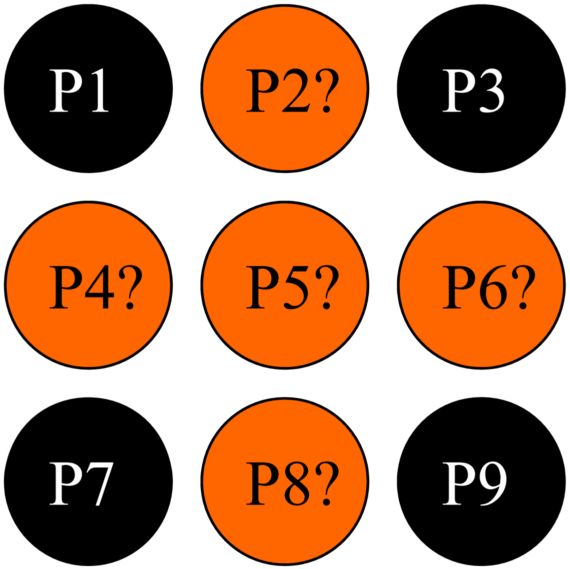
Артифакты

- Алиасинг
  - При увелечении – ступенчатость
  - При уменьшении - муар
- Ringing
- Потеря четкости
- Субпиксельный сдвиг
  - Влияет на формальные метрики



Простые методы

- Билинейная интерполяция



$$P2 = \frac{(P1 + P3)}{2}$$

$$P4 = \frac{(P1 + P7)}{2}$$

$$P6 = \frac{(P3 + P9)}{2}$$

$$P8 = \frac{(P7 + P9)}{2}$$

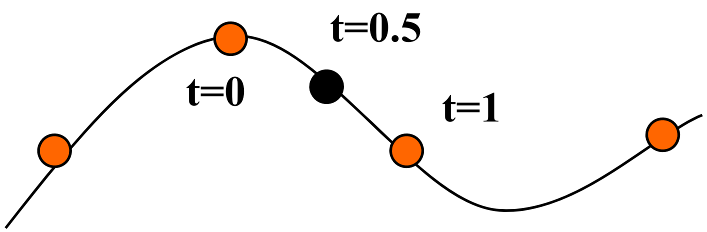
$$P5 = \frac{(P1 + P3 + P7 + P9)}{4}$$

Простые методы

- Билинейная интерполяция
  - Сепарабельная
  - Очень быстрая
  - Поддерживается в HW
    - Точность фильтрации

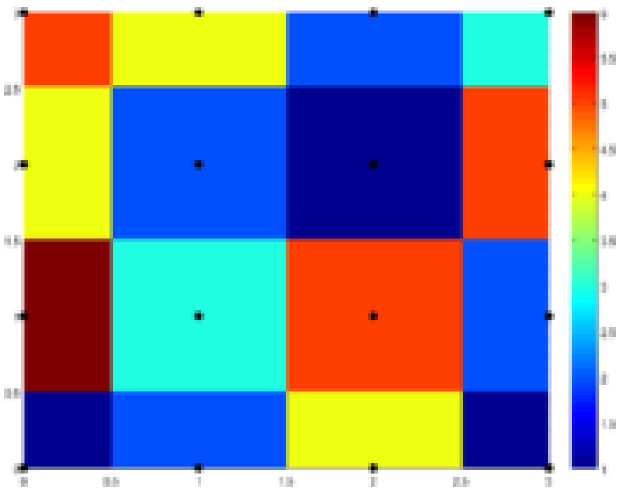
Простые методы

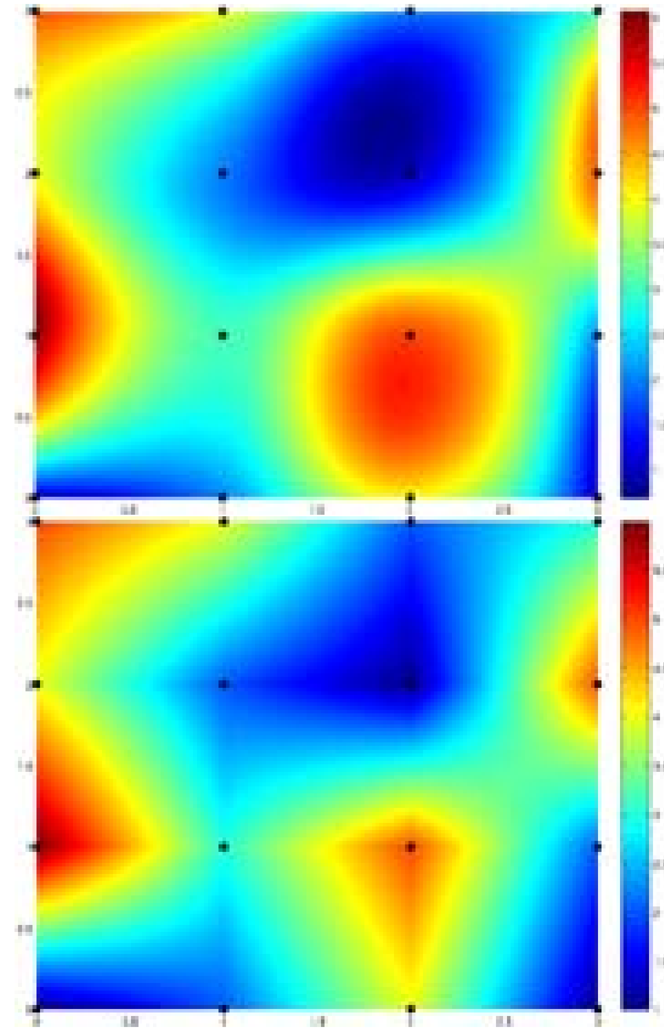
- Бикубическая интерполяция
  - Сепарабельная
  - Лучше качество



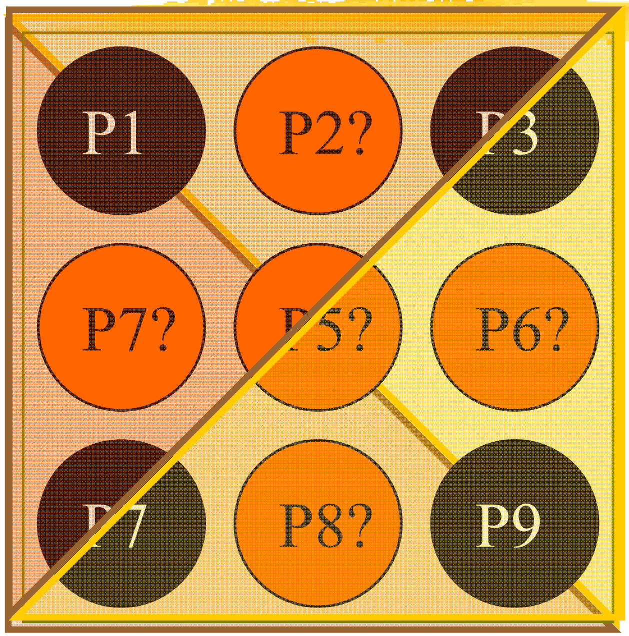
$$p(t) = \frac{1}{2}\begin{bmatrix}1 & t & t^2 & t^3\end{bmatrix}\begin{bmatrix}0 & 2 & 0 & 0\\ -1 & 0 & 1 & 0\\ 2 & -5 & 4 & -1\\ -1 & 3 & -3 & 1\end{bmatrix}\begin{bmatrix}a_{-1}\\ a_0\\ a_1\\ a_2\end{bmatrix}$$

Сравнение





Gradient interpolation



Dxd = abs(P3 – P5)
Dyd = abs(P1 – P9)

If (Dxd > Dyd)

//граница P1P5P9

P5 = (P1 + P9) * 0.5f;

If (Dyd > Dxd)

//граница P3P5P7

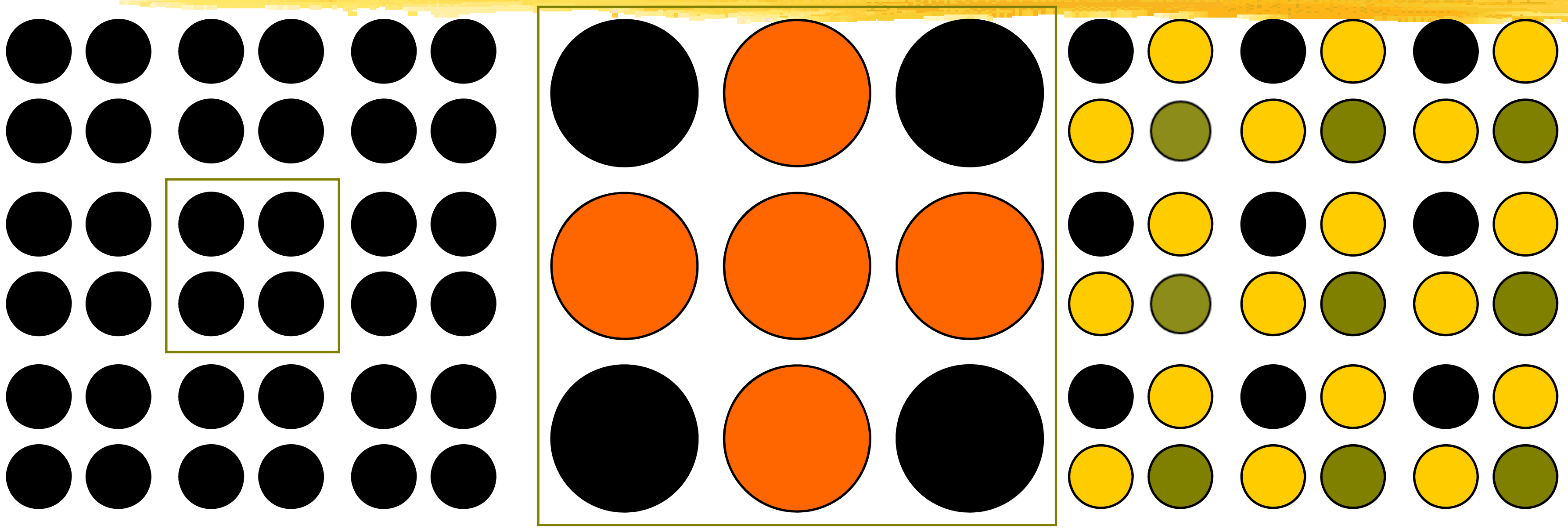
P5 = (P1 + P9) * 0.5f;

If (Dyd ~= Dxd)

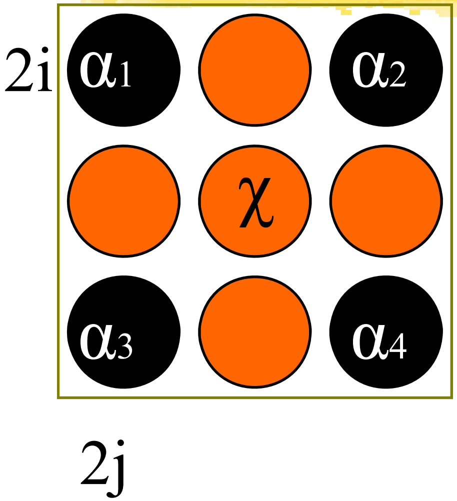
//граница не определена

P5 = (P1 + P3 + P7 + P9) * 0.25f;

NEDI
(New Edge-Directed Interpolation)



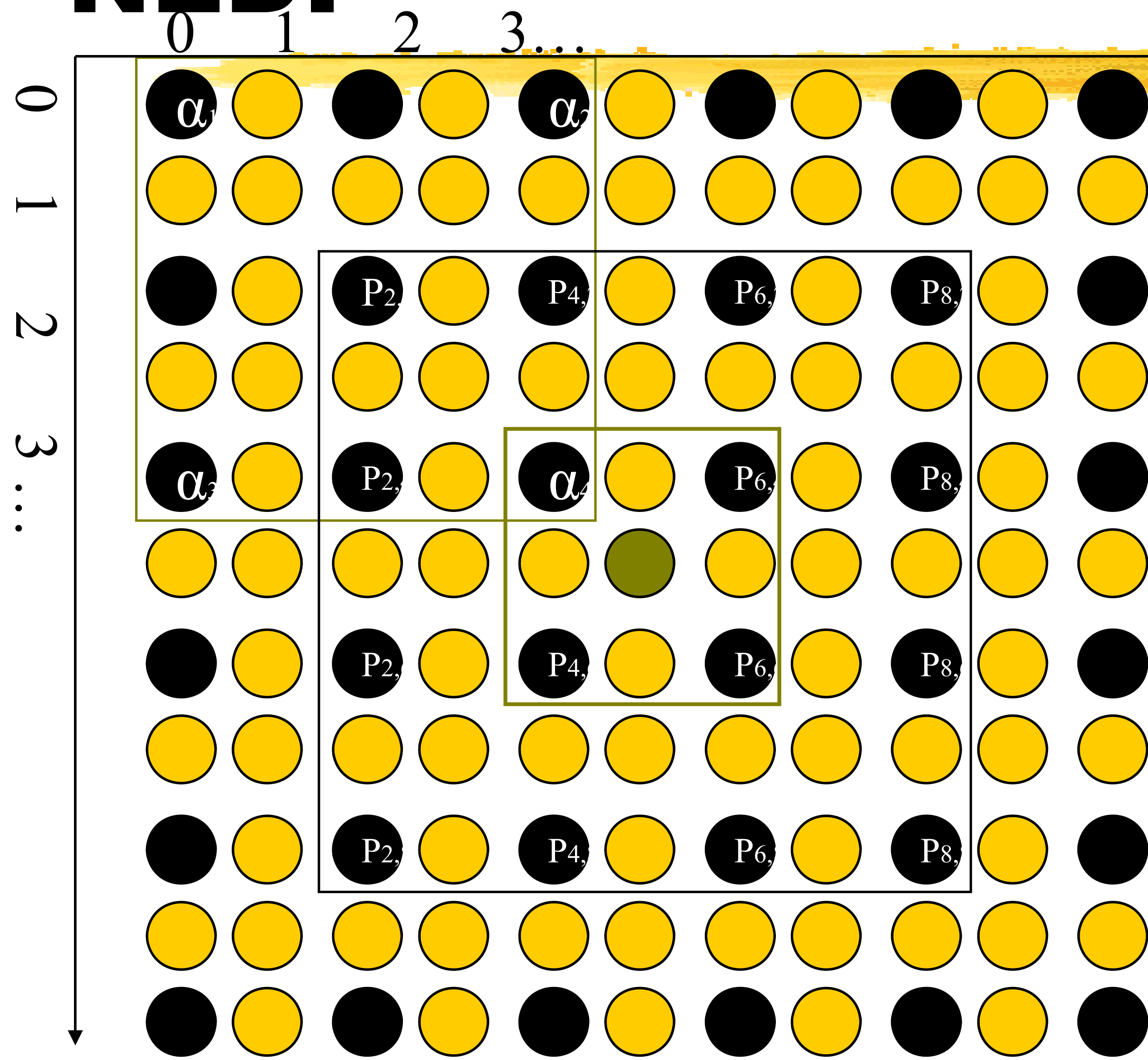
NEDI



$$
\begin{aligned}
X = F(2i+1,2j+1) = \; & \alpha_1 * F(2i,2j) + \\
& \alpha_2 * F(2i+2,2j) + \\
& \alpha_3 * F(2i,2j+2) + \\
& \alpha_4 * F(2i+2,2j+2);
\end{aligned}
$$

$$\alpha = \{\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4\}\ ?$$

NEDI



$$X_{i,j} = \alpha_1 * F(i-2,j-2) + \alpha_2 * F(i+2,j-2) + \alpha_3 * F(i-2,j+2) + \alpha_4 * F(i+2,j+2);$$

For i = 2,4,6,8 | For j = 2,4,6,8

$E_{i,j} = P_{j,j} - X_{i,j}$ – Approximation Error

$$SSE = \Sigma\Sigma \, sqr(E_{i,j});$$

$$\alpha = \underset{\alpha}{\text{Arg min}}(SSE);$$

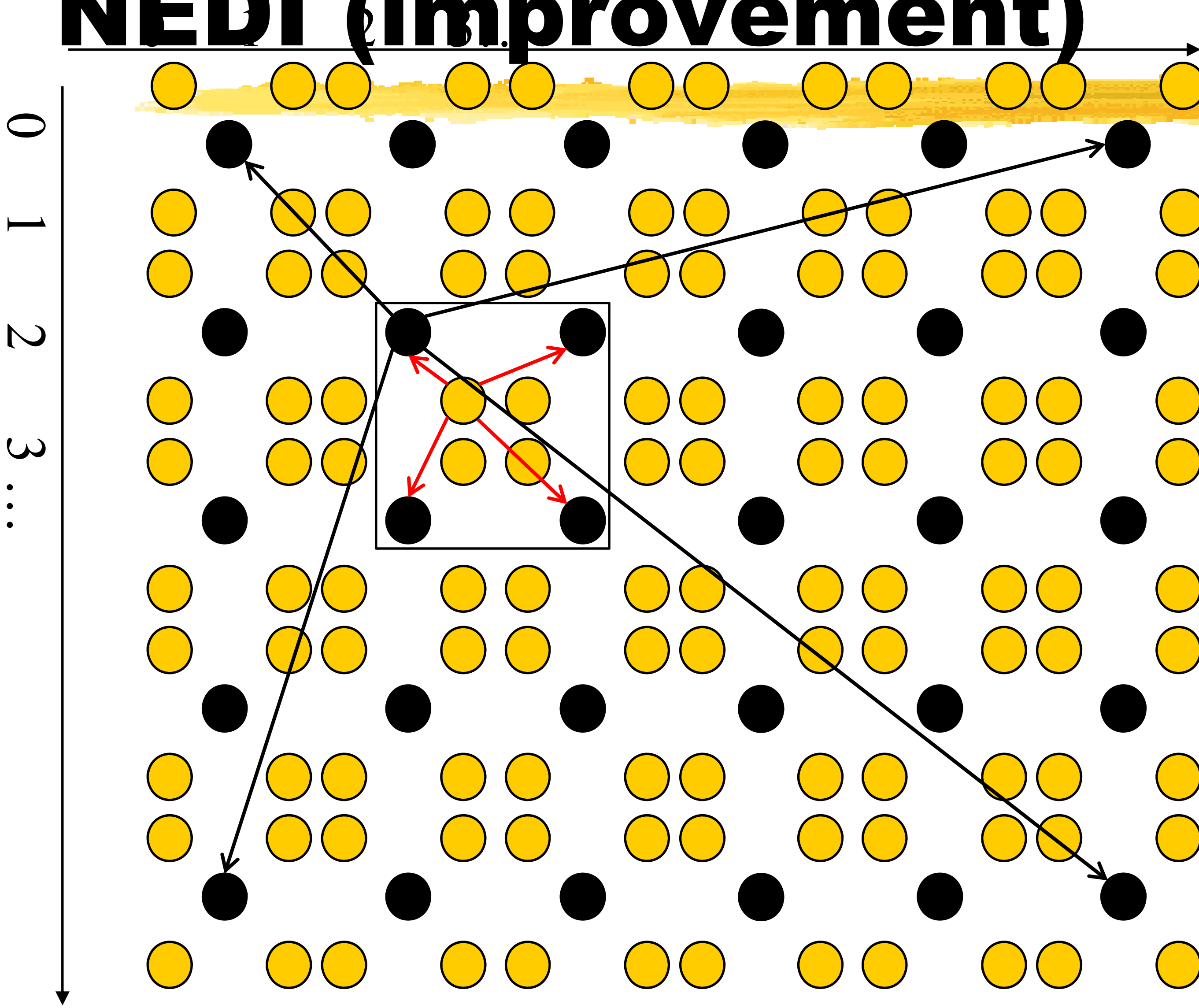
$$P = \{P_{ij}\}$$

$$C\alpha = P$$

$$\partial(SSE) / \partial\alpha = 0.0$$

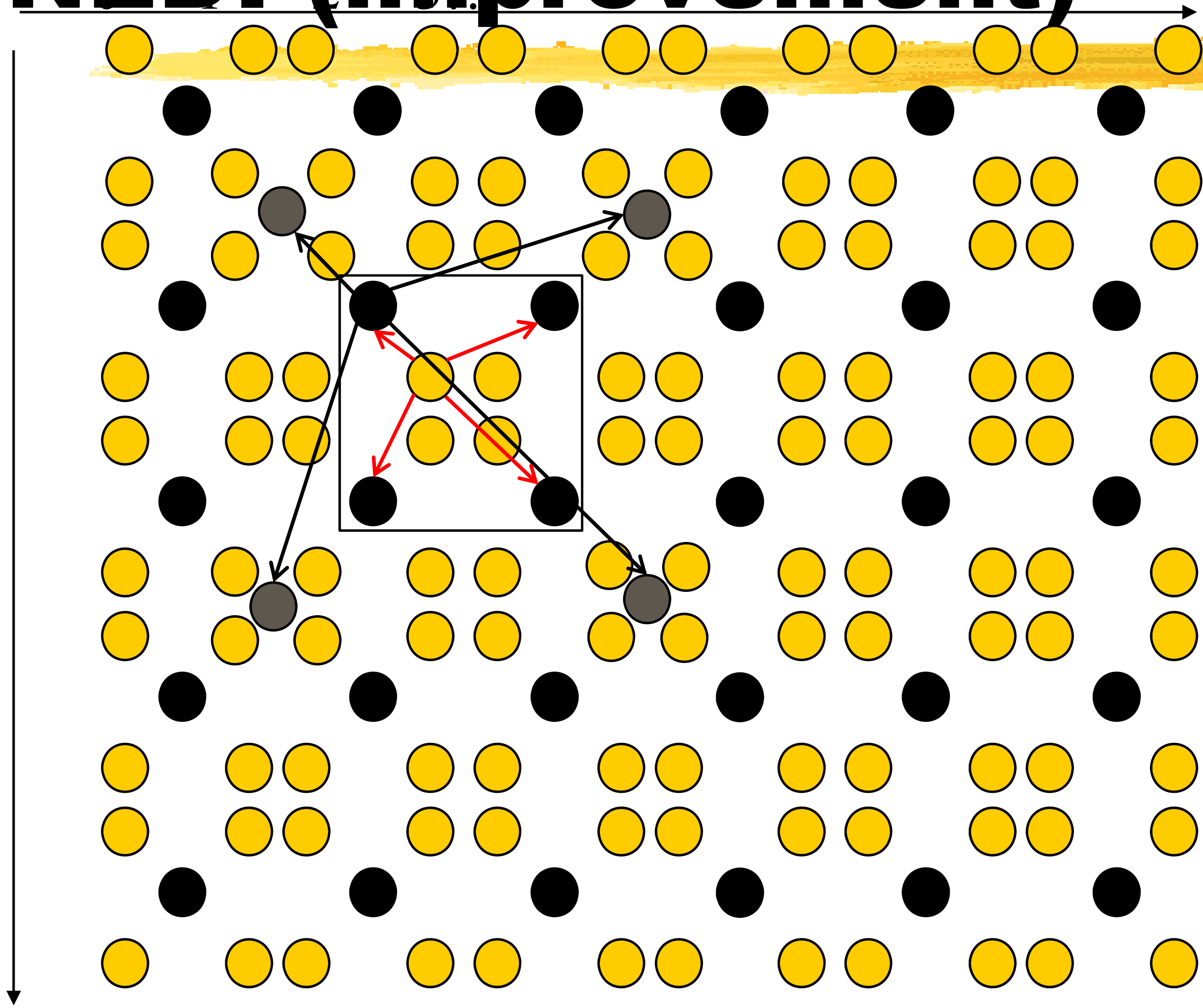
$$\alpha = (C^T C)^{-1} C^T P$$

NEDI (improvement)

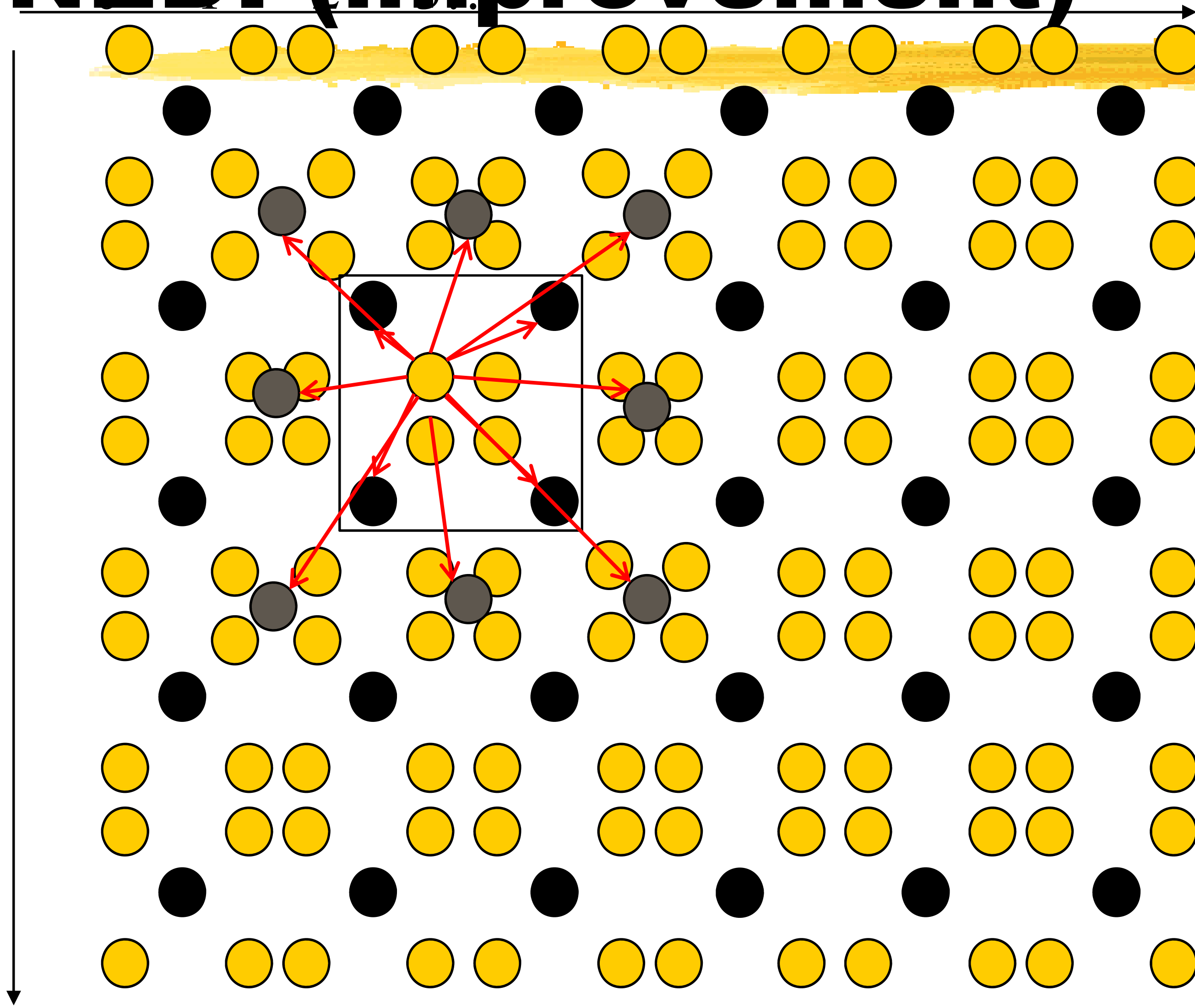


Расстояние между пикселами может быть слишком большим

NEDI (improvement)



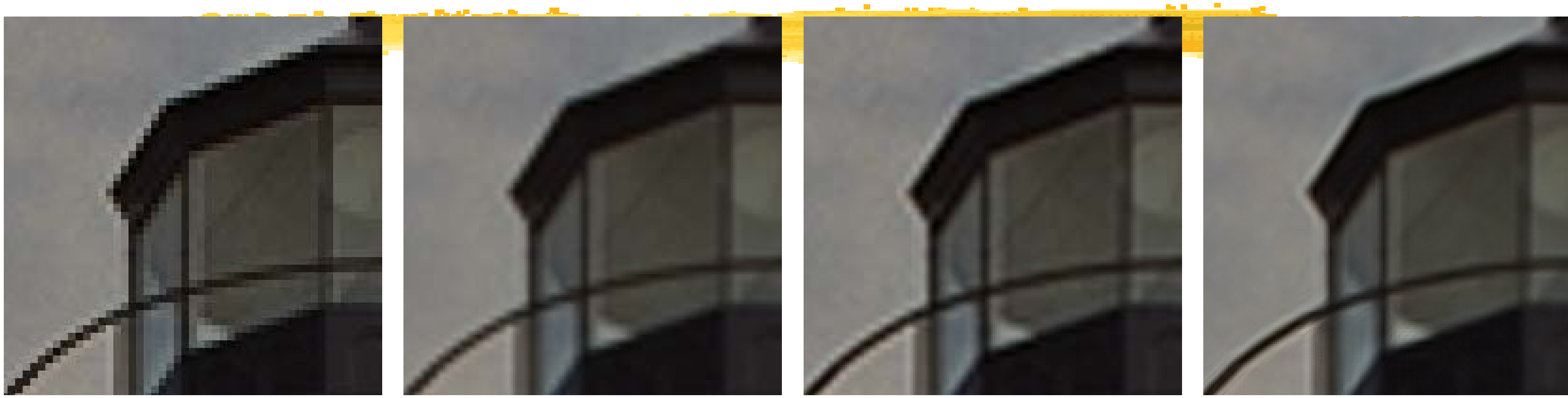
NEDI (improvement)



NEDI: Pros and Cons

- Pros: NEDI is doing a great job!
  - Четкие тонкие края
  - Рингинг
- Cons: Очень медленно на CPU
  - Умножение матриц 4x16
  - Обращение матрицы
- CUDA:
  - Большой объем данных на тред
  - Много регистров
  - Сложно использовать Smem
  - Много ветвлений

Сравнение



Вопросы