

Иерархия памяти CUDA. Текстуры в CUDA. Цифровая обработка сигналов

Лектор:

[Боресков А.В. \(ВМК МГУ\)](mailto:steps3d@narod.ru), steps3d@narod.ru

План

- Работа с Текстурами
- Свертка
- Шумоподавление
- Масштабирование изображений



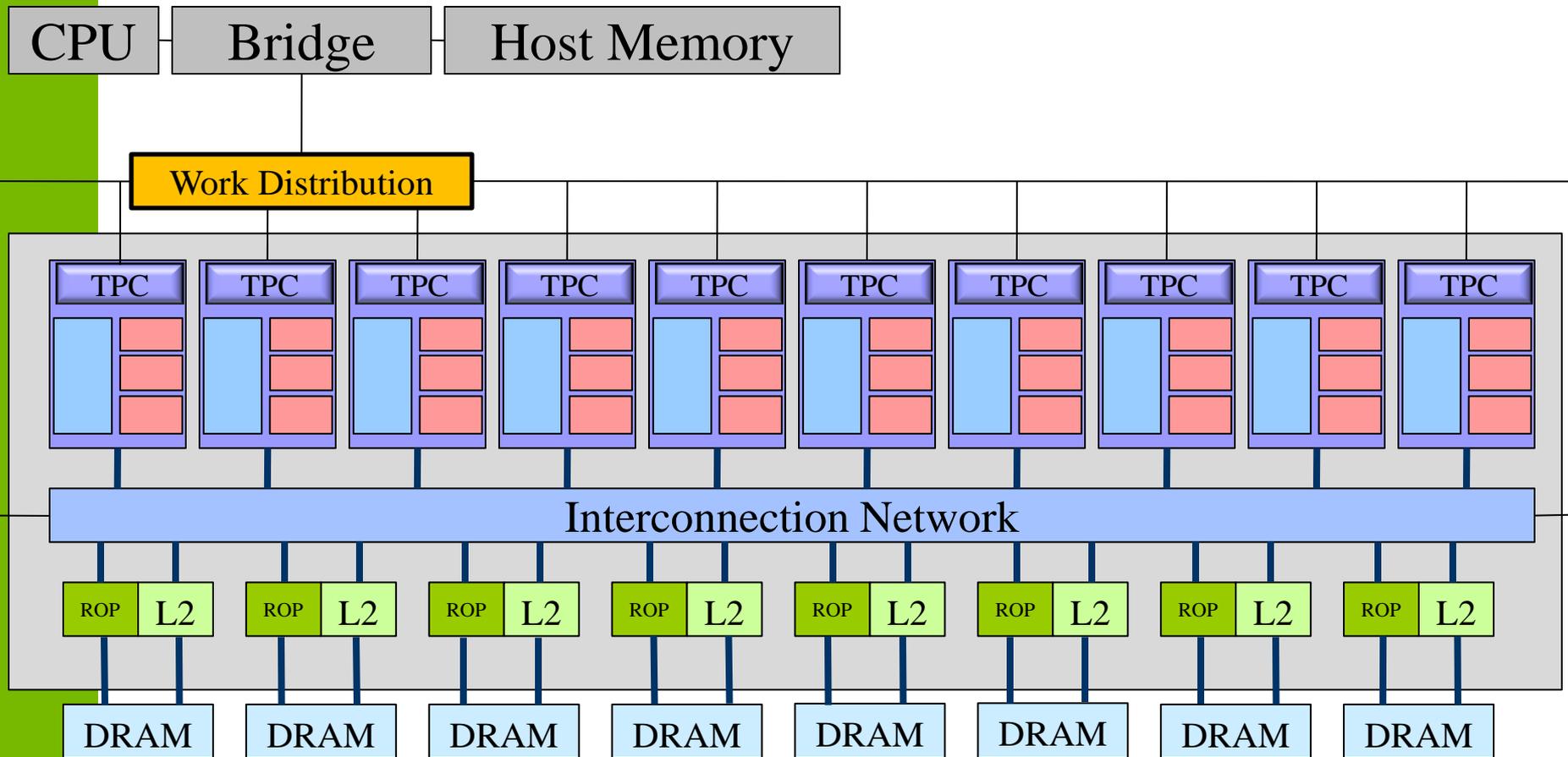
РАБОТА С ТЕКСТУРАМИ

Типы памяти в CUDA

Тип памяти	Доступ	Уровень выделения	Скорость работы	Расположение
Регистры	R/W	Per-thread	Высокая	SM
Локальная	R/W	Per-thread	Низкая	DRAM
Shared	R/W	Per-block	Высокая	SM
Глобальная	R/W	Per-grid	Низкая	DRAM
Constant	R/O	Per-grid	Высокая	DRAM
Texture	R/O	Per-grid	Высокая	DRAM

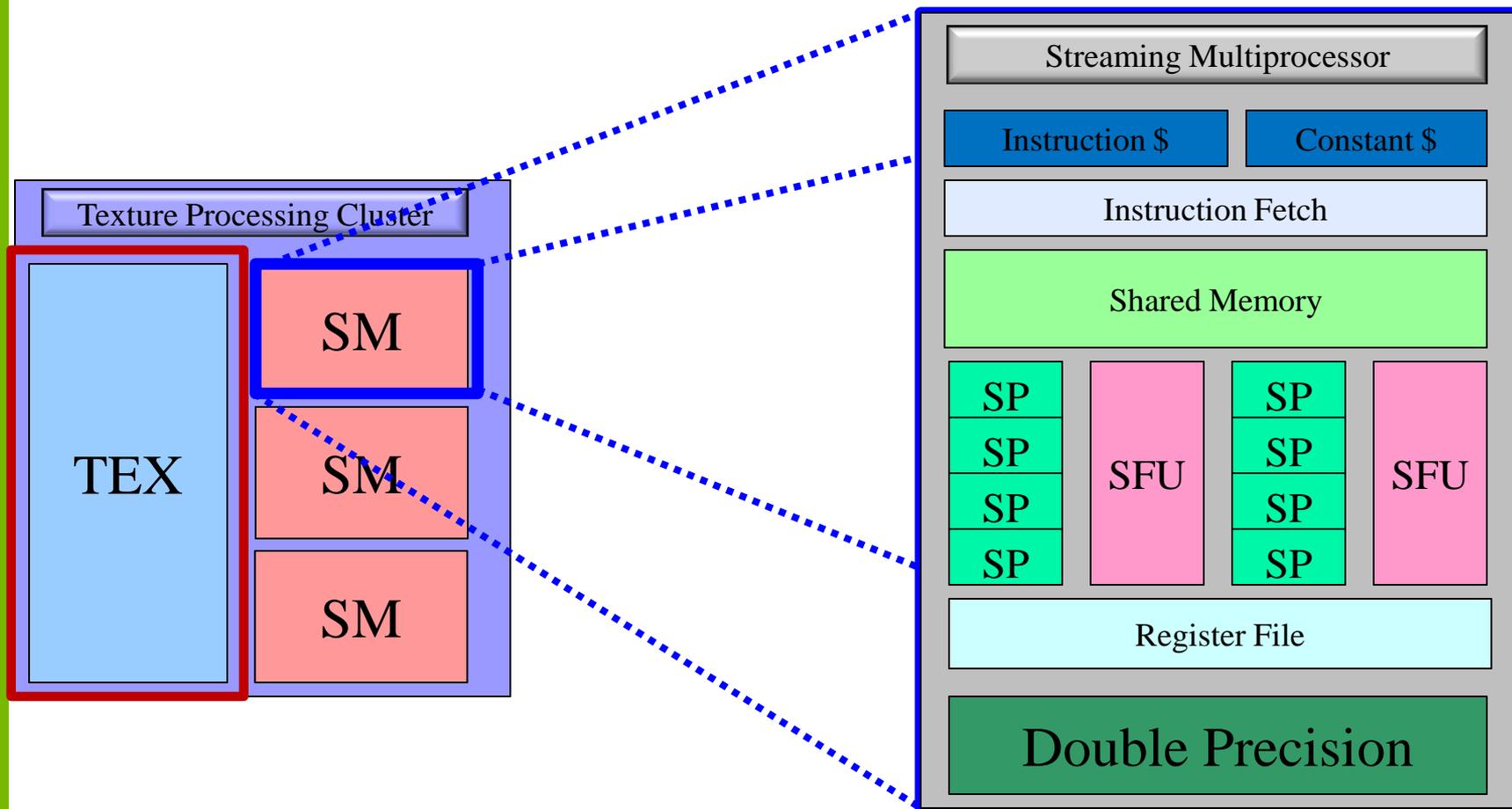
Легенда:
-интерфейсы
доступа 

Архитектура Tesla 10



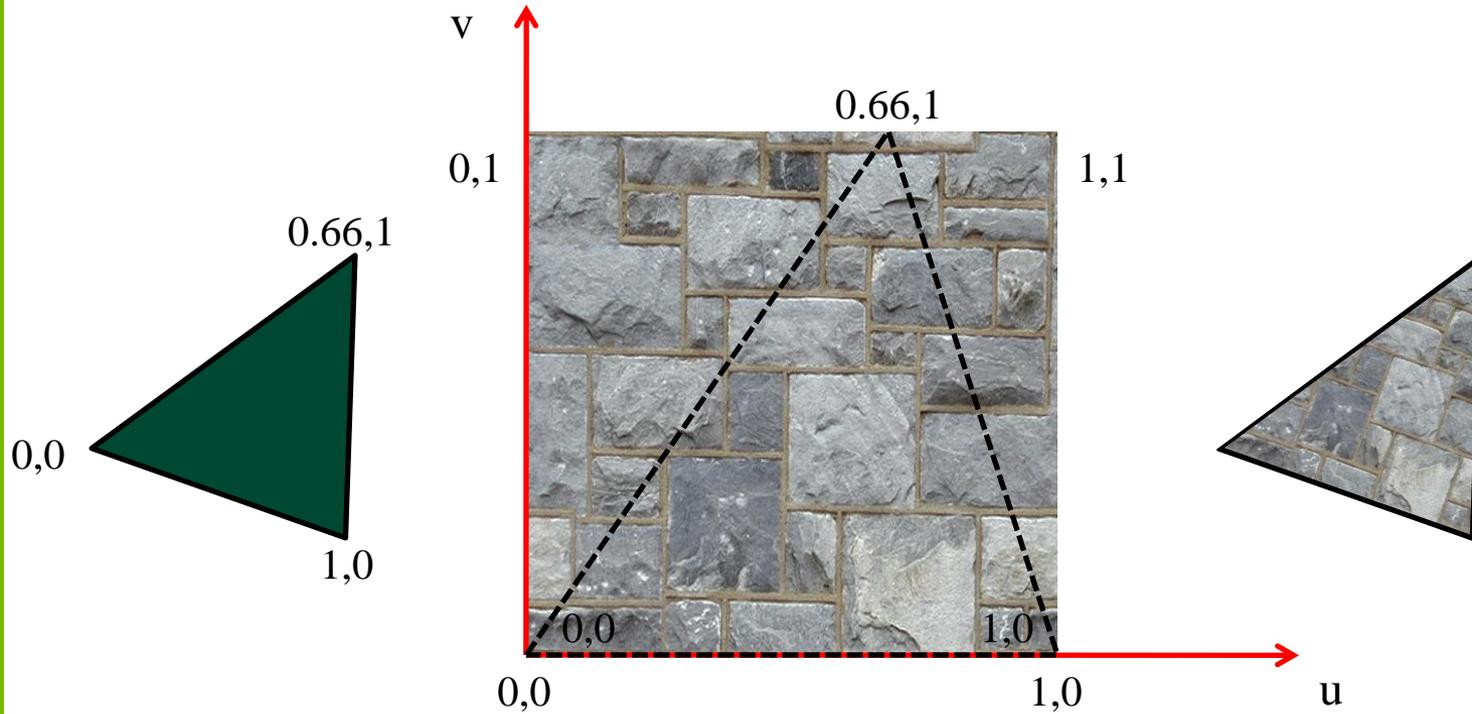
Архитектура Tesla

Мультипроцессор Tesla 10



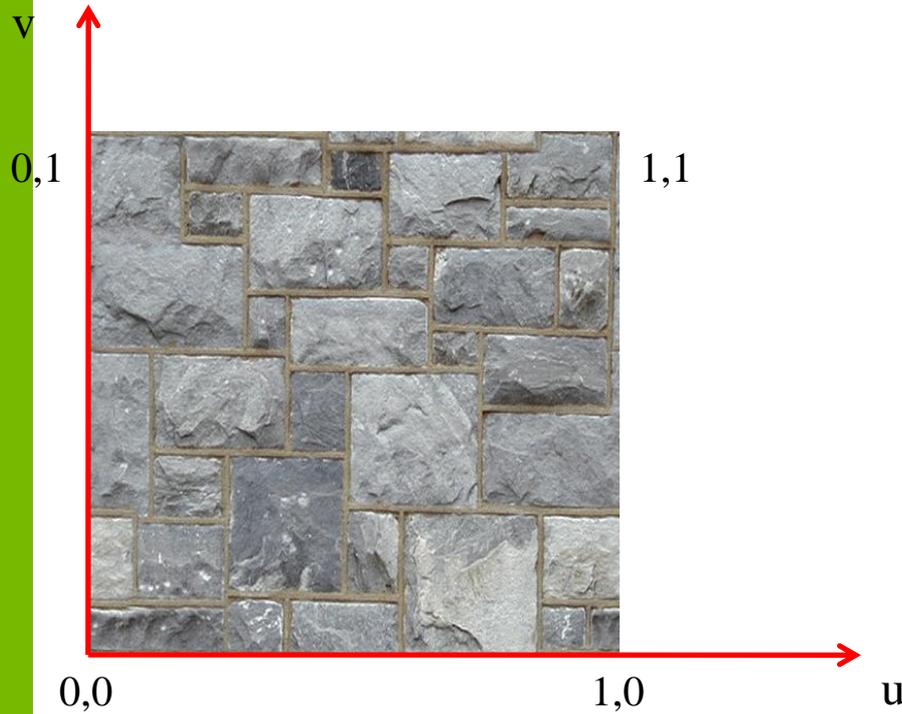
Texture в 3D

- В CUDA есть доступ к fixed-function HW: Texture Unit



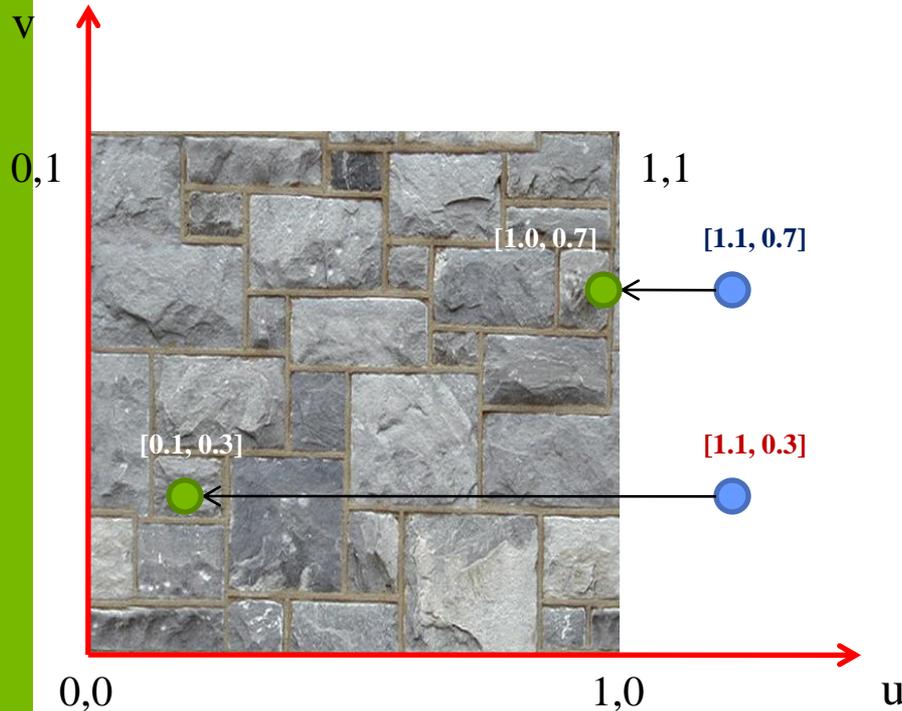
Texture HW

- Нормализация координат:
 - Обращение по координатам, которые лежат в диапазоне $[0, 1]$



Texture HW

- Преобразование координат:
 - Координаты, которые не лежат в диапазоне $[0, 1]$ (или $[w, h]$)



Clamp:

- Координата «обрубается» по допустимым границам

Wrap

- Координата «заворачивается» в допустимый диапазон

Texture HW

- Фильтрация:

- Если вы используете float координаты, что должно произойти если координата не попадает точно в *texel*?



Point:

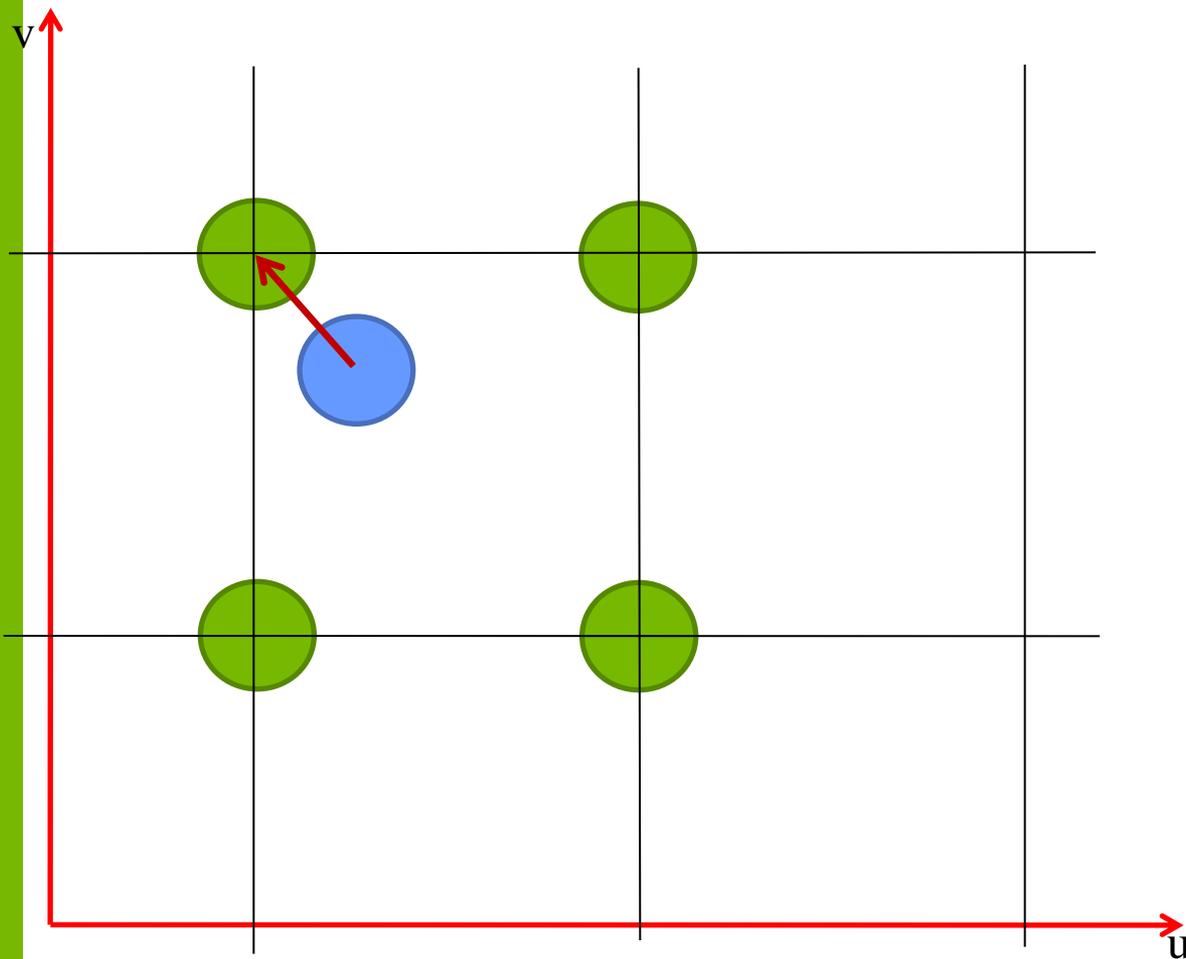
- Берется ближайший texel

Linear:

- Билинейная фильтрация

Texture HW

- Фильтрация

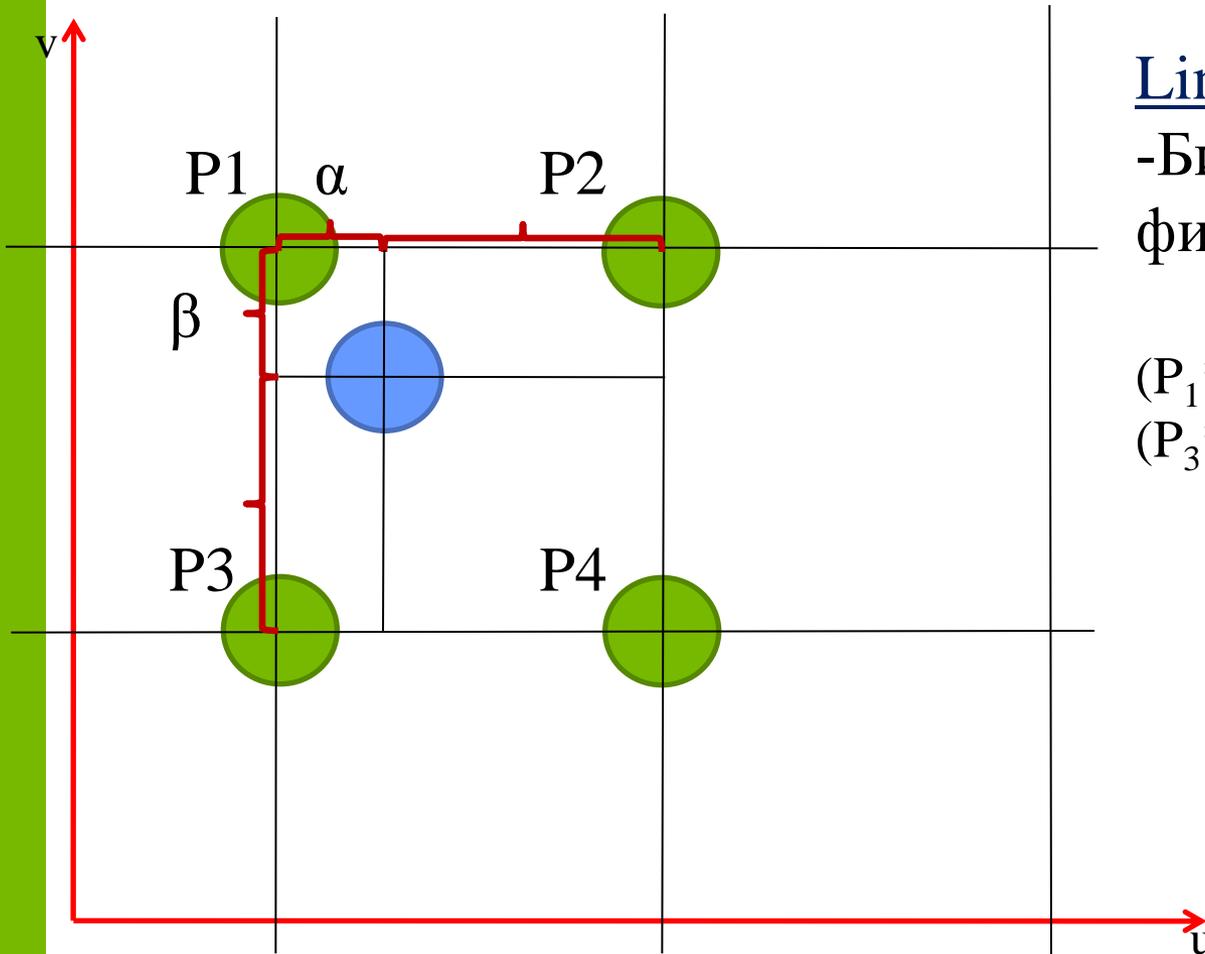


Point:

-Берется
ближайший
texel

Texture HW

- Фильтрация



Linear:

-Билинейная
фильтрация

$$(P_1 * (1 - \alpha) + P_2 * (\alpha)) * (1 - \beta) + (P_3 * (1 - \alpha) + P_4 * (\alpha)) * (\beta)$$

Texture в CUDA

- Преобразование данных:
 - **cudaReadModeNormalizedFloat** :
 - Исходный массив содержит данные в *integer*, возвращаемое значение во *floating point* представлении (доступный диапазон значений отображается в интервал $[0, 1]$ или $[-1, 1]$)
 - **cudaReadModeElementType**
 - Возвращаемое значение то же, что и во внутреннем представлении

cudaArray

- Особый контейнер памяти
- Черный ящик для приложения
- Позволяет организовывать данные в 1D/ 2D/3D массивы данных вида:
 - 1/2/4 компонентные векторы
 - 8/16/32 bit signed/unsigned integers
 - 32 bit float
 - 16 bit float (driver API)
- На CC 2.x возможна запись из ядра

Texture<> в CUDA (cudaArray)

- Только чтение
- Обращение к 1D/2D/3D массивам данных по:
 - Целочисленным индексам
 - Нормализованным координатам
- Преобразование адресов на границах
 - Clamp, Wrap
- Фильтрация данных
 - Point, Linear
- Преобразование данных
 - Данные могут храниться в формате `uchar4`
 - Возвращаемое значение - `float4`

Surface<> в CUDA (cudaArray)

- Чтение и запись
 - `cudaArraySurfaceLoadStore`
- Обращение к 1D/2D массивам данных по:
 - Целочисленным индексам
 - Byte-адресация по x
- Преобразование адресов на границах
 - Clamp, Zero, Trap

Texture<> в CUDA (linear)

- Можно использовать обычную *линейную* память
- Ограничения:
 - 1D / 2D
 - Нет фильтрации
 - Доступ по целочисленным координатам
 - Обращение по адресу вне допустимого диапазона возвращает ноль

Texture в CUDA (linear)

```
texture<float, 1, cudaReadModeElementType> texRef;

__global__ void kernell ( float * data )
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    data [idx] = tex1Dfetch(texRef, idx);
}

int main(int argc, char ** argv)
{
    float *hA = NULL, *hB = NULL, *dA = NULL, *dB = NULL;

    for (int idx = 0; idx < numThreads * numBlocks; idx++)
        hA[idx] = sinf(idx * 2.0f * PI / (numThreads * numBlocks) );

    cudaMemcpy ( dA, hA, memSizeInBytes, cudaMemcpyHostToDevice );

    cudaBindTexture(0, texRef, dA, memSizeInBytes);

    dim3 threads = dim3( numThreads );
    dim3 blocks  = dim3( numBlocks );

    kernell <<<blocks, threads>>> ( dB );
    cudaDeviceSynchronize();
    cudaMemcpy ( hB, dB, nMemSizeInBytes, cudaMemcpyDeviceToHost );

    return 0;
}
```

Texture в CUDA (cudaArray)

```
texture<float, 2, cudaReadModeElementType> texRef;
__global__ void kernel ( float * data )
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    data [idx + blockIdx.y * gridDim.x * blockDim.x] = tex2D(g_TexRef, idx, blockIdx.y);
}
int main ( int argc, char * argv [] )
{
    float * hA = NULL, *hB = NULL, *dA = NULL, *dB = NULL; // linear memory pointers
    cudaArray * paA = NULL; // device cudaArray pointer

    cudaChannelFormatDesc cfD=cudaCreateChannelDesc(32,0,0,0, cudaChannelFormatKindFloat);
    cudaMallocArray(&aA, &cfD, numBlocksX * numThreads, numBlocksY);

    for (int idx = 0; idx < numThreads * numBlocksX; idx++) {
        hA[idx] = sinf(idx*2.0f*PI/(numThreads*numBlocksX) );
        hA[idx + numThreads * numBlocksX] = cosf(idx*2.0f*PI/(numThreads*numBlocksX) );
    }

    cudaMemcpyToArray(aA, 0, 0, hA, memSizeInBytes, cudaMemcpyHostToDevice);
    cudaBindTextureToArray(texRef, paA);

    kernel2<<<dim3( numBlocksX, numBlocksY ), numThreads>>> ( dB );

    cudaDeviceSynchronize();
    cudaMemcpy ( hB, dB, memSizeInBytes, cudaMemcpyDeviceToHost );

    return 0;
}
```

Texture HW

- Латентность больше, чем у прямого обращения в память
 - Дополнительные стадии в конвейере:
 - Преобразование адресов
 - Фильтрация
 - Преобразование данных
- Есть кэш
 - Разумно использовать, если:
 - Объем данных не влезает в shared память
 - Шаблон доступа хаотичный
 - Данные переиспользуются разными потоками



BINDLESS TEXTURES

- Kepler и выше (3.0)
- До миллиона текстур
- Не глобальный объект, можно передавать как параметр
- Можно в runtime определить к какой текстуре будем обращаться
- Новый объект - `cudaTextureObject_t`
- Соответствует OpenGL bindless textures
- Обращение дешевле чем к texture reference

BINDLESS TEXTURES

- Создается через `cudaCreateTextureObject` по описателям
 - Ресурса - `cudaResourceDesc`
 - Задаёт подключаемую память - линейную или `cudaArray`
 - Текстуры - `cudaTextureDesc`
 - Задаёт параметры текстуры
 - Режим адресации (отсечения) текстурных координат (`Wrap` или `Clamp`)
 - Режим фильтрации
 - Режим чтения
 - Являются ли текстурные координаты нормализованными
- `cudaArray` создается при помощи описателя канала `cudaChannelFormatDesc`
 - Задаёт число бит для каждой компоненты
 - Тип компонент

Texture object API (cudaArray)

```
struct cudaResourceDesc resDesc;
struct cudaTextureDesc texDesc;

memset(&resDesc, 0, sizeof(resDesc));
resDesc.resType          = cudaResourceTypeArray; // на основе cudaArray
resDesc.res.array.array = cuArray;

memset(&texDesc, 0, sizeof(texDesc));           // параметры текстуры
texDesc.addressMode[0] = cudaAddressModeWrap;   // отсечения для s,t
texDesc.addressMode[1] = cudaAddressModeWrap;
texDesc.filterMode     = cudaFilterModeLinear; // режим фильтрации
texDesc.readMode       = cudaReadModeElementType; // режим чтения
texDesc.normalizedCoords = 1;                  // нормализованность

cudaTextureObject_t texObj = 0;                // создаем объект
cudaCreateTextureObject(&texObj, &resDesc, &texDesc, NULL);

kernel<<<blocks, threads>>> ( texObj );

cudaDestroyTextureObject ( texObj );
```

Texture object API (linear memory)

```
cudaResourceDesc resDesc;

memset ( &resDesc, 0, sizeof ( resDesc ) );
resDesc.resType          = cudaResourceTypeLinear;
resDesc.res.linear.devPtr = buffer;
resDesc.res.linear.desc.f = cudaChannelFormatKindFloat;
resDesc.desc.x           = 32; // bits per channel
resDesc.res.linear.sizeInBytes = N * sizeof ( float );

cudaTextureDesc texDesc;

memset ( &texDesc, 0, sizeof ( texDesc ) );
texDesc.readMode = cudaReadModeElementType;

cudaTextureObject_t tex;

cudaCreateTextureObject ( &tex, &resDesc, &texDesc, NULL );

kernel<<<blocks, threads>>> ( tex );

cudaDestroyTextureObject ( texObj );
```

Свертка

- Определение свертки:

$$r(i) = (s * k)(i) = \int s(i - n)k(n)dn$$

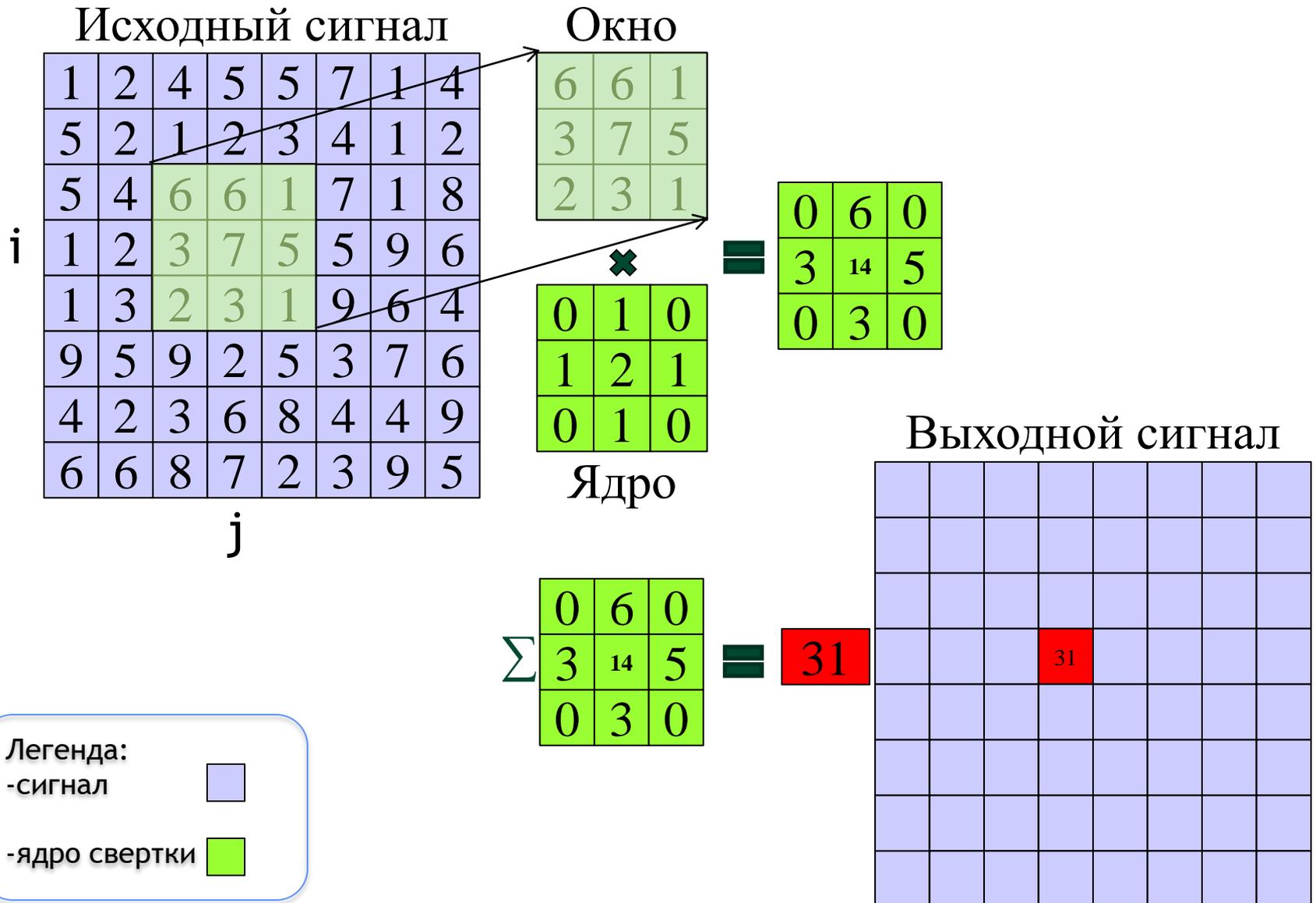
- В Дискретном случае:

$$r(i) = (s * k)(i) = \sum s(i - n)k(n)$$

- В 2D для изображений:

$$r(i, j) = (s * k)(i, j) = \sum_n \sum_m s(i - n, j - m)k(n, m)$$

Свертка



Свертка

- Вычислительная сложность:

$$- \underbrace{W \times H}_{\text{Размер входного сигнала}} \times \underbrace{N \times K}_{\text{Размер ядра}} - \text{умножений}$$

- Сепарабельные фильтры

-1	0	1
-2	0	2
-1	0	1

Ядро

 $=$

1
2
1

Ядро Y

 \times

-1	0	1
----	---	---

Ядро X



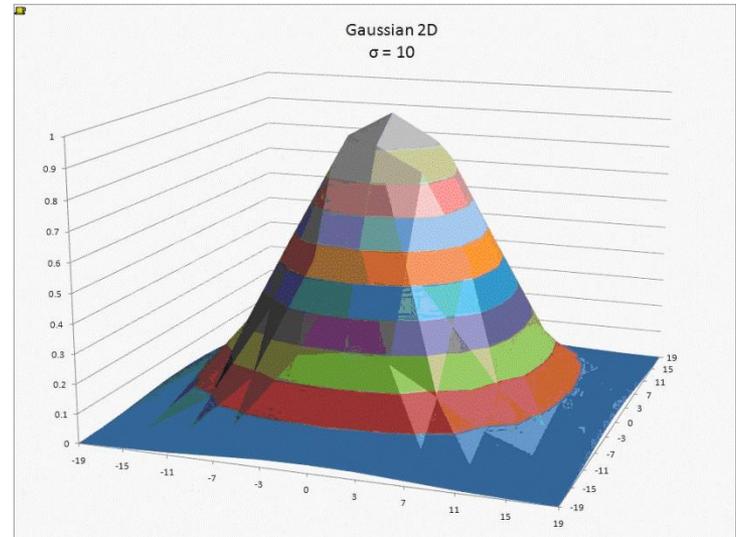
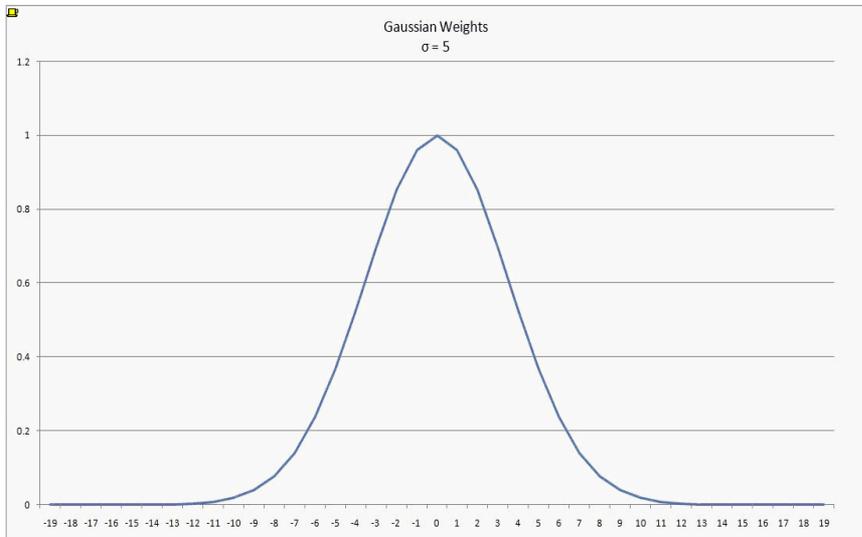
Примеры

- Gaussian Blur
- Edge Detection

Gaussian Blur

- Свертка с ядром:

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



Gaussian Blur

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

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

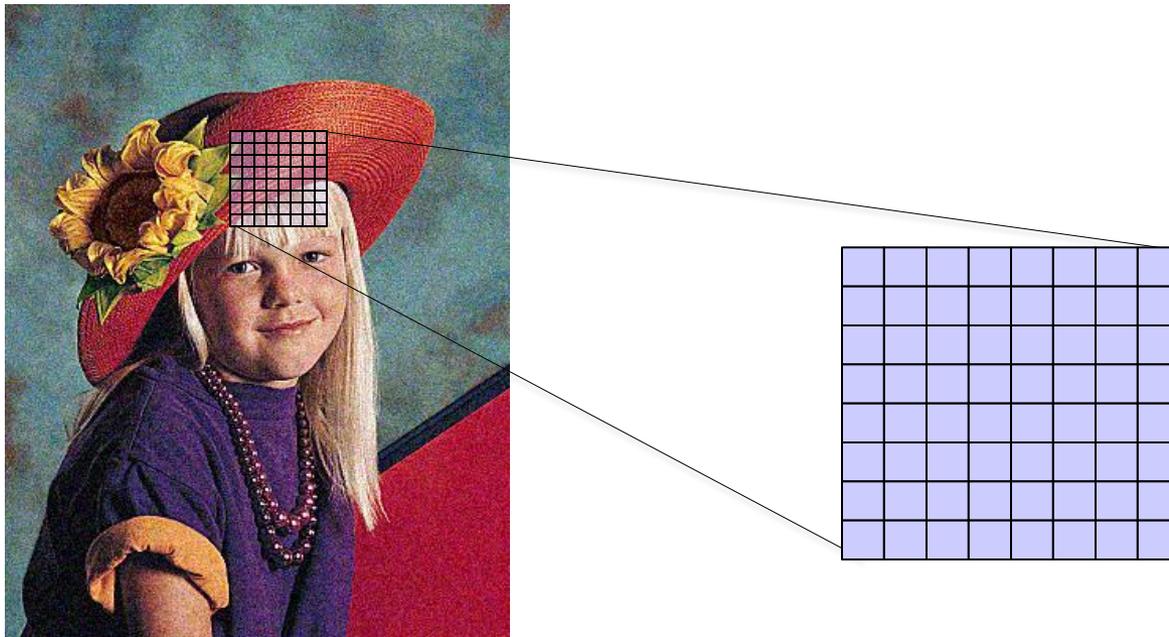
    float sum = 0.0f;
    float result = 0.0f;
    for (int ix = -r; ix <= r; ix++){
        for (int iy = -r; iy <= r; iy++)
        {
            float w = exp( -(SQR(ix) + SQR(iy)) / SQR(r) );
            result += w * tex2D(texRef, idx + ix, idy + iy);
            sum += w;
        }
    }
    result /= sum;

    filteredImage[idx + idy * W] = result;
}
```

Свертка Оптимизации

- Использовать сепарабельные фильтры
 - Существенно меньше алгоритмическая сложность
- Использовать *shared* память

Свертка Оптимизации

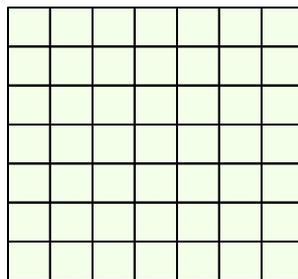
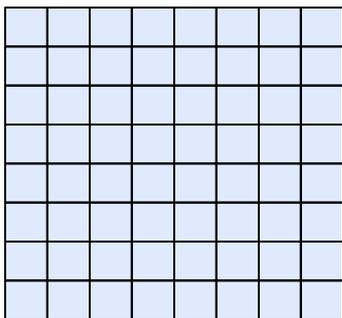


Исходное изображение

Эффективно ли такое разбиение изображения



Свертка Стет Оптимизации



Легенда:

-сигнал

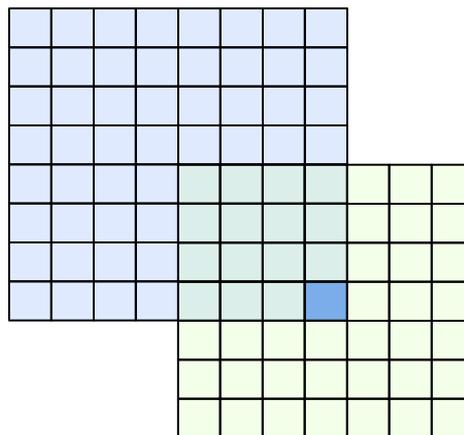
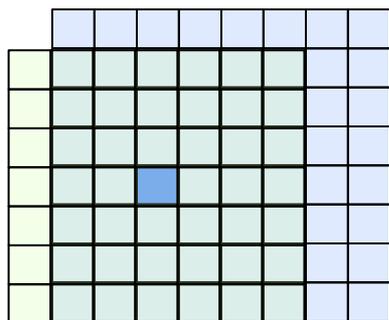
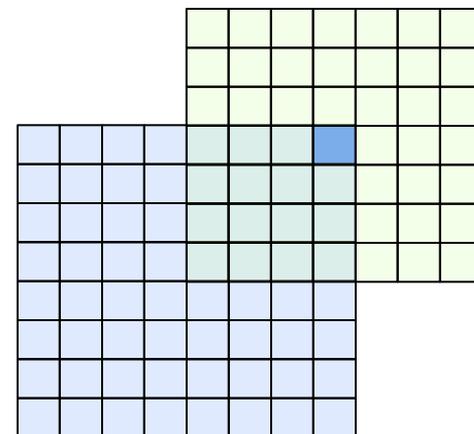
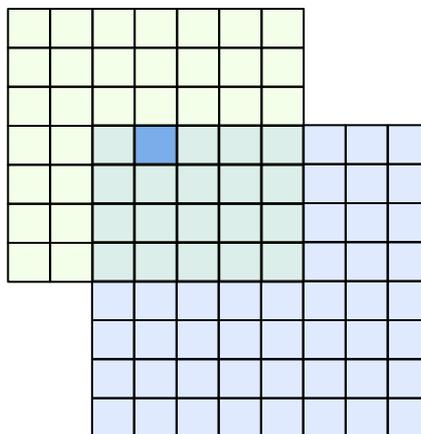
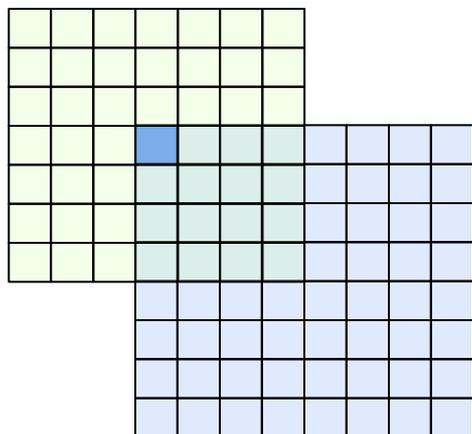


-ядро свертки



Свертка

Смет Оптимизации

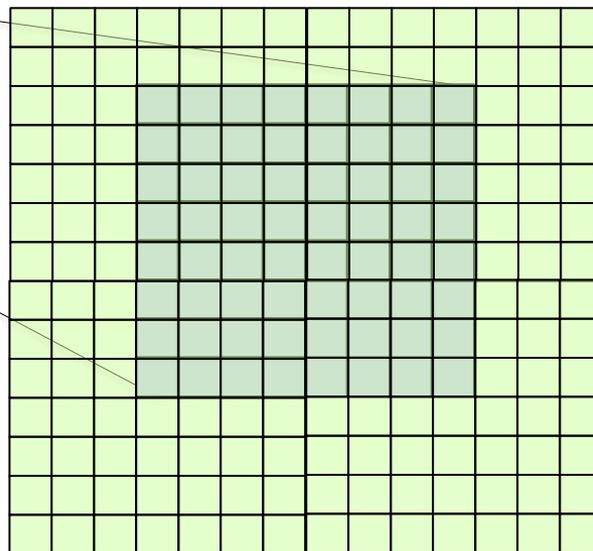
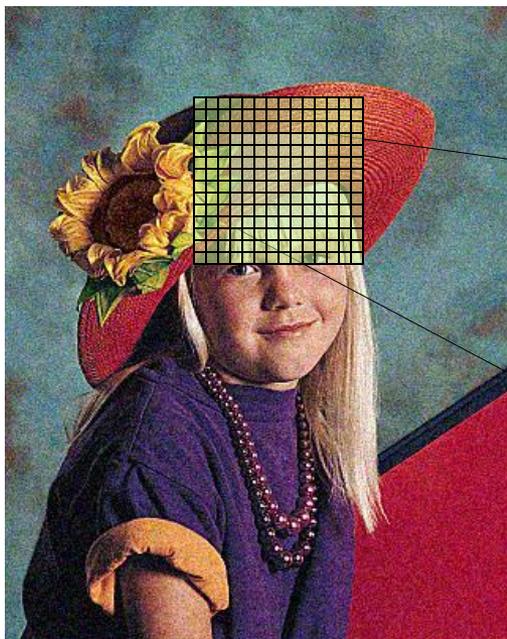


Легенда:

- сигнал 
- ядро свертки 



Свертка Стет Оптимизации



Легенда:

-сигнал



-ядро свертки



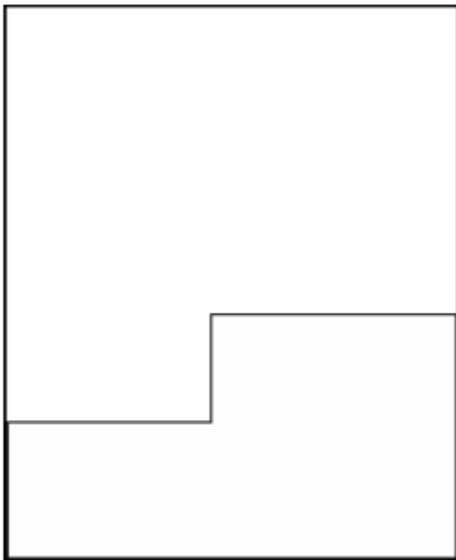
Свертка Смет Оптимизации



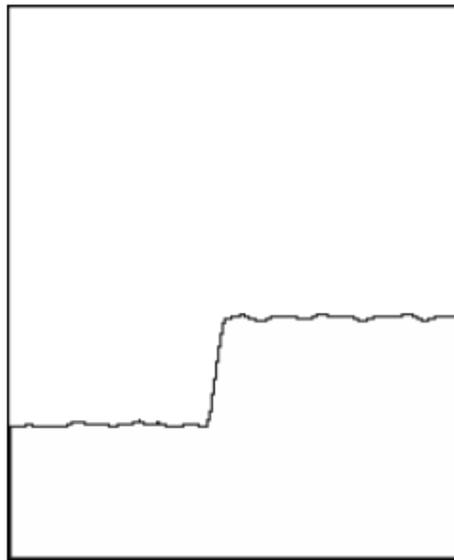
EDGE DETECTION

Edge Detection

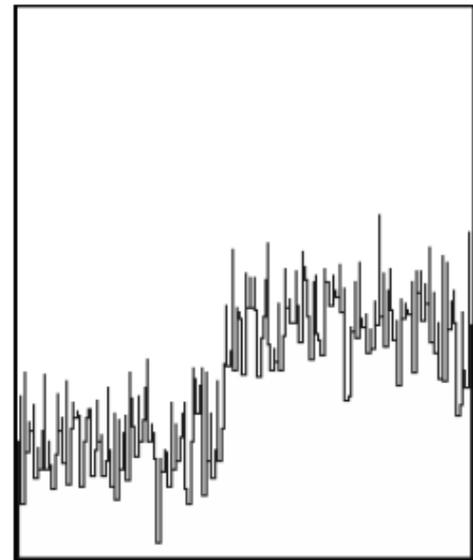
- Обнаружение границ - поиск разрывов в яркости изображения



Идеальная
граница



«реальная»
граница



«шумная»
граница

Edge Detection

- Градиент функции $f(x, y)$
 - Это вектор который показывает направление роста
 - Определяется как $\mathbf{G} = \left\{ \begin{matrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{matrix} \right\}$

$|\mathbf{G}(x, y)| = [G_x^2 + G_y^2]^{\frac{1}{2}}$

$\theta(x, y) = \text{atan}(G_y/G_x)$

Edge Detection

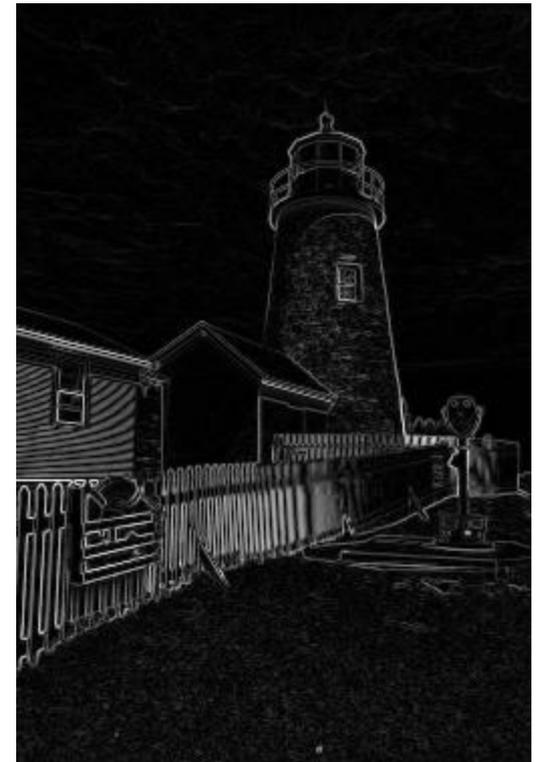
- Разностная производная:

$$\frac{\partial f(x, y)}{\partial x} \approx \frac{f(x + \Delta x, y) - f(x, y)}{\Delta x}$$

$$\frac{\partial f(x, y)}{\partial y} \approx \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y}$$

- Свертка с ядром:

$$D_{1y} = [-1 \ 1] \quad D_{1x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$



Edge Detection

- Разностная производная:

$$\frac{\partial f(x, y)}{\partial x} \approx \frac{f(x + \Delta x, y) - f(x - \Delta x, y)}{2\Delta x}$$

$$\frac{\partial f(x, y)}{\partial y} \approx \frac{f(x, y + \Delta y) - f(x, y - \Delta y)}{2\Delta y}$$

- Свертка с ядром:

$$D_{2y} = [-1 \ 0 \ 1] \quad D_{2x} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

Edge Detection

- Prewitt mask:

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$



Edge Detection

- Sobel mask:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



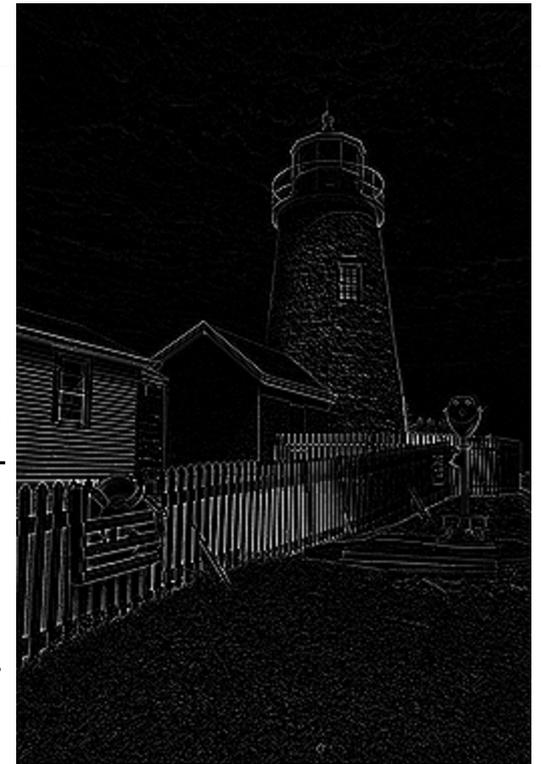
Edge Detection

- Оператор Лапласа:

$$L[f(x, y)] = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

$$\frac{\partial^2 f(x, y)}{\partial x^2} \approx \frac{f(x + \Delta x, y) - 2f(x, y) + f(x - \Delta x, y)}{\Delta x^2}$$

$$\frac{\partial^2 f(x, y)}{\partial y^2} \approx \frac{f(x, y + \Delta y) - 2f(x, y) + f(x, y - \Delta y)}{\Delta y^2}$$



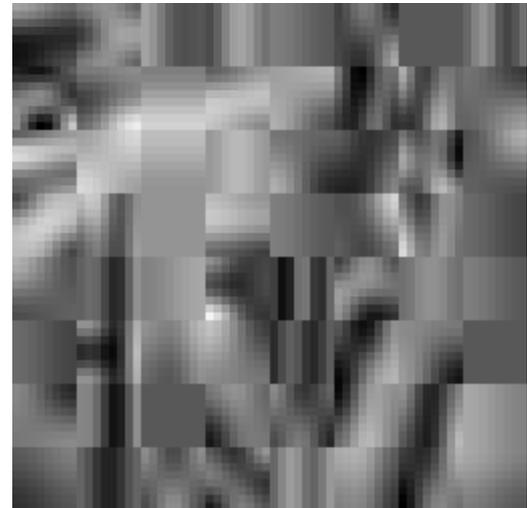
ШУМОПОДАВЛЕНИЕ

Discrete Cosine Transform

- Является основой современных алгоритмов сжатия данных с потерями (JPEG, MPEG)



JPEG, 2/10



Discrete Cosine Transform

- Представитель семейства пространственно-частотных 1D преобразований, задается формулами:

- Прямое:
$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{\pi(2x+1)u}{2N}\right], \quad u = 0, 1, \dots, N-1$$

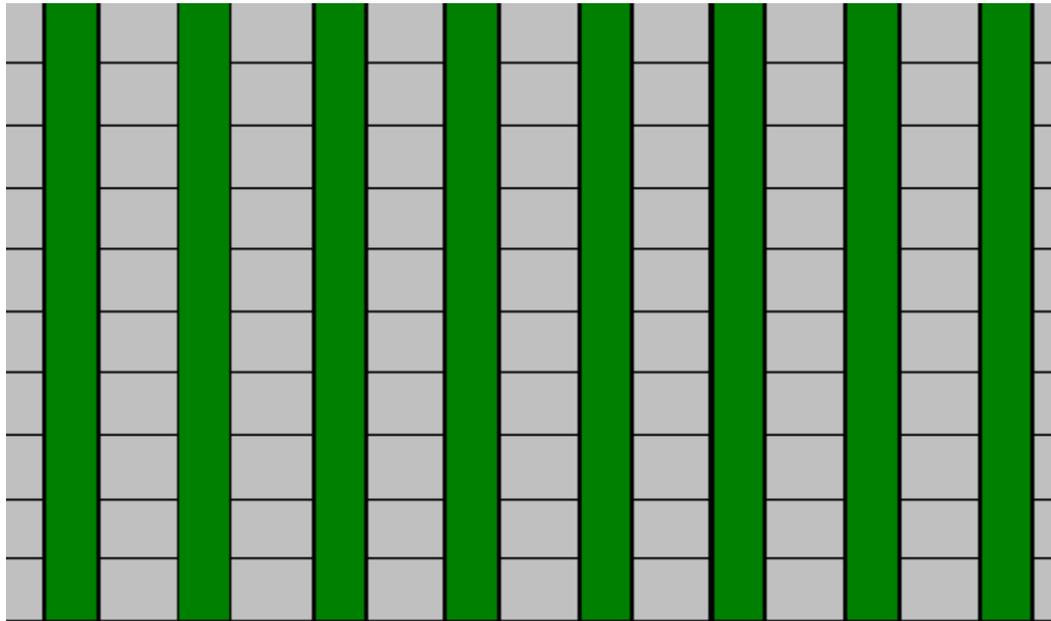
- Обратное:
$$f(x) = \sum_{u=0}^{N-1} \alpha(u) C(u) \cos\left[\frac{\pi(2x+1)u}{2N}\right], \quad x = 0, 1, \dots, N-1$$

- Нормировочные коэффициенты:

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0 \\ \sqrt{\frac{2}{N}}, & u \neq 0 \end{cases}$$

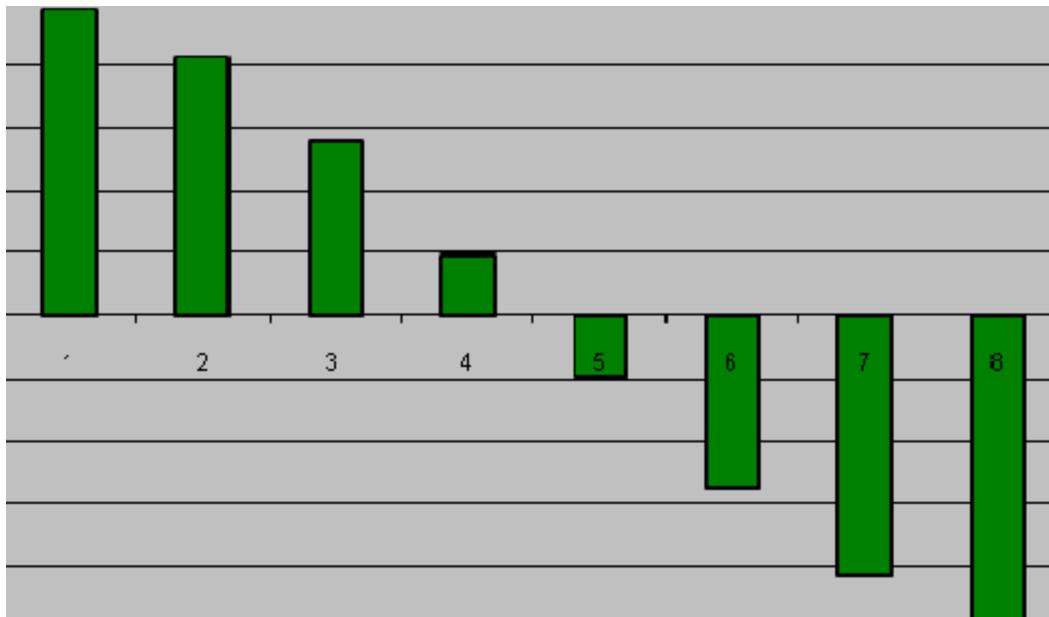
Discrete Cosine Transform

- 8-точечный случай: $u=0$



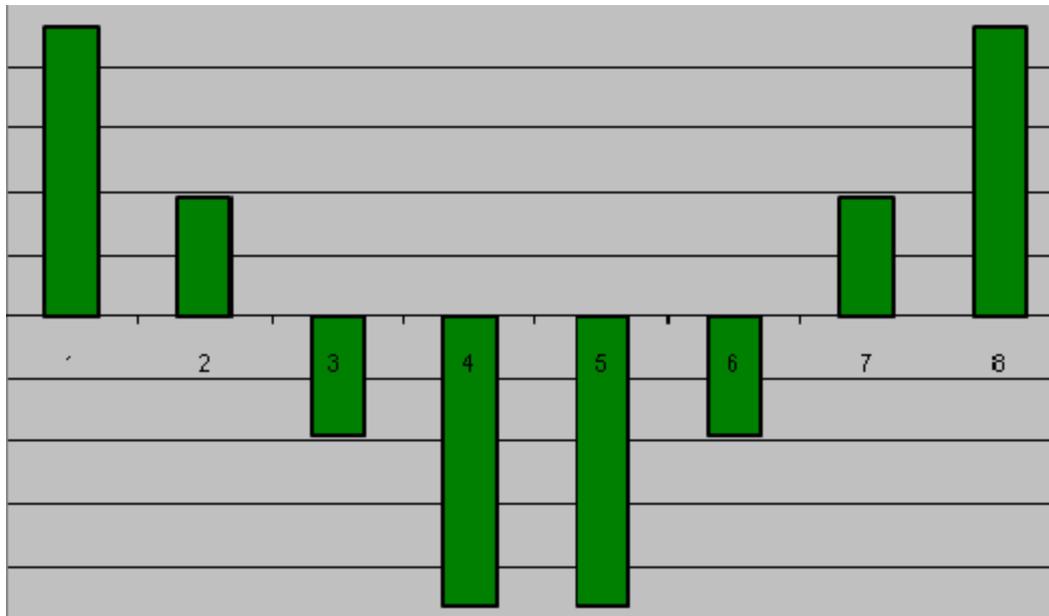
Discrete Cosine Transform

- 8-точечный случай: $u=1$



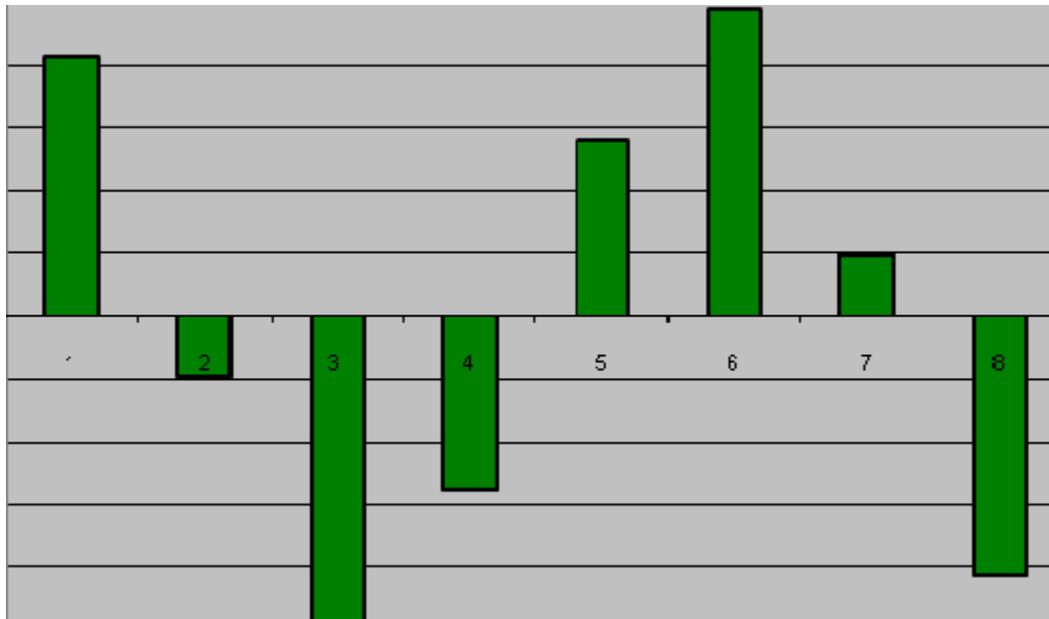
Discrete Cosine Transform

- 8-точечный случай: $u=2$



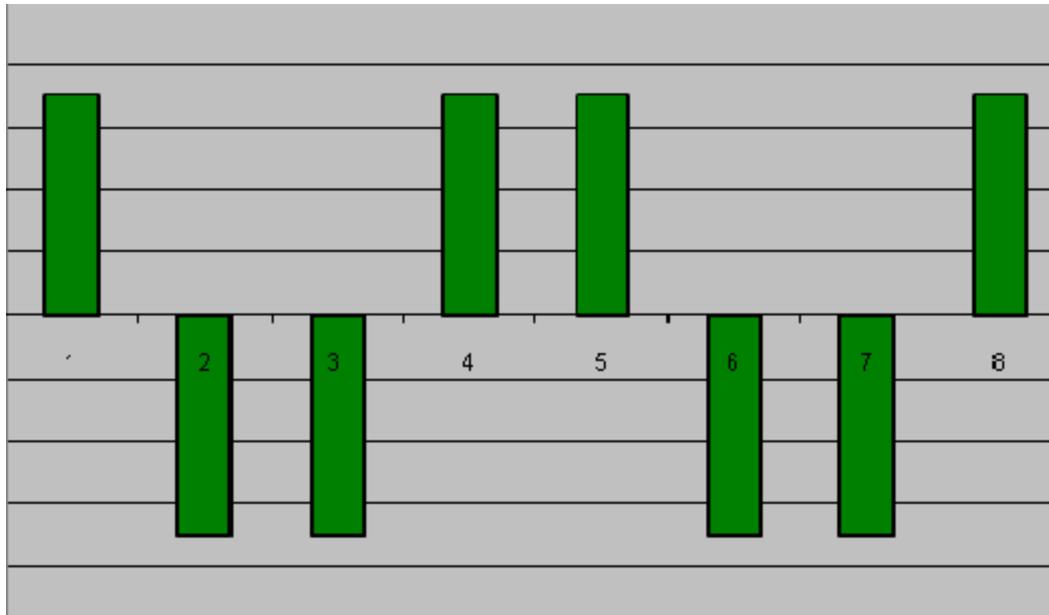
Discrete Cosine Transform

- 8-точечный случай: $u=3$



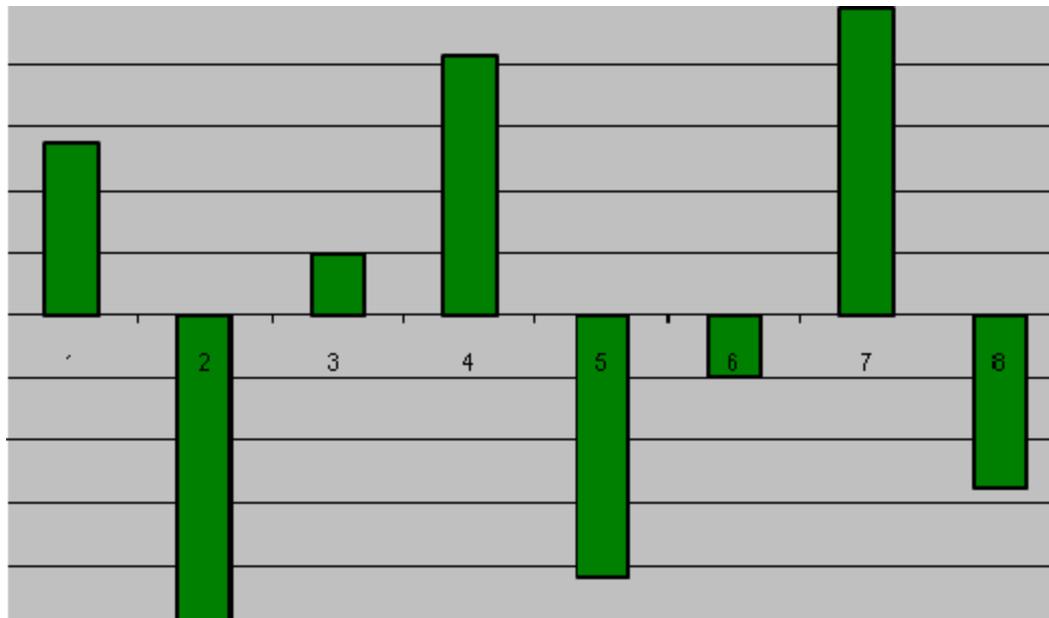
Discrete Cosine Transform

- 8-точечный случай: $u=4$



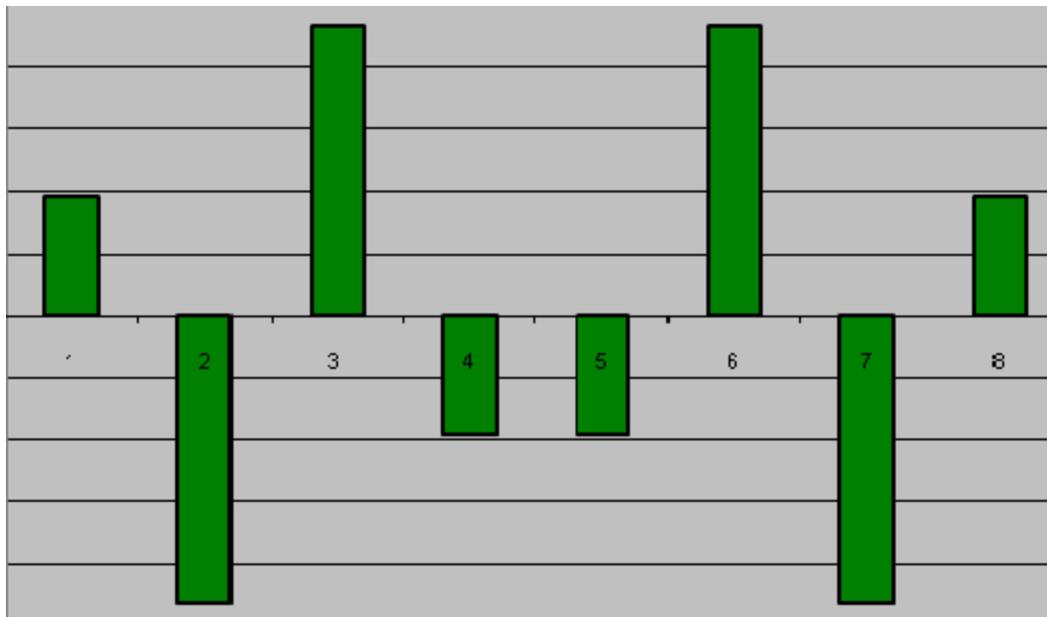
Discrete Cosine Transform

- 8-точечный случай: $u=5$



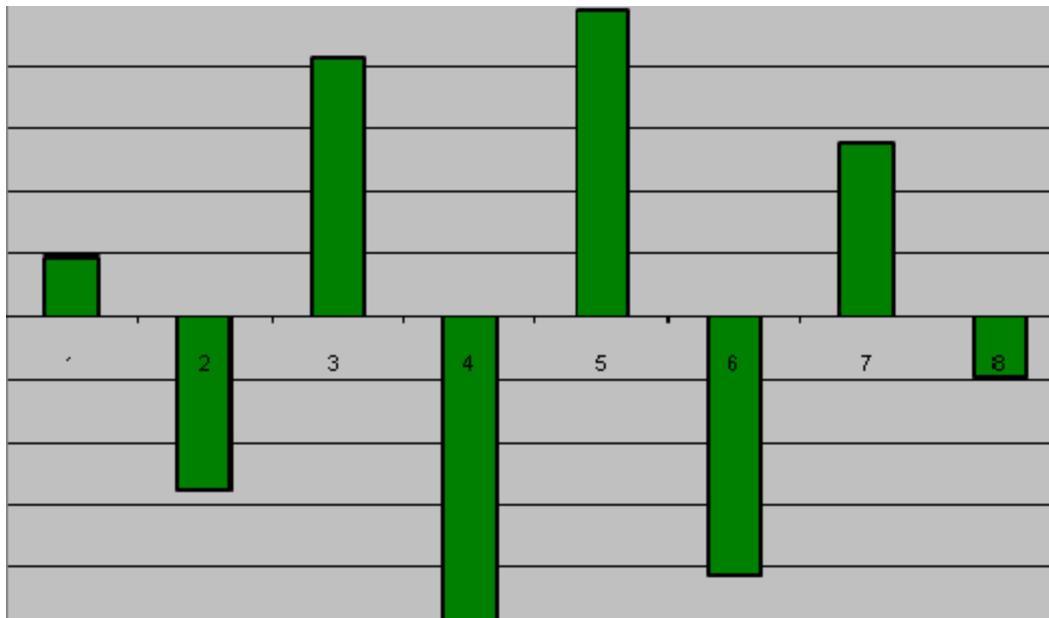
Discrete Cosine Transform

- 8-точечный случай: $u=6$



Discrete Cosine Transform

- 8-точечный случай: $u=7$



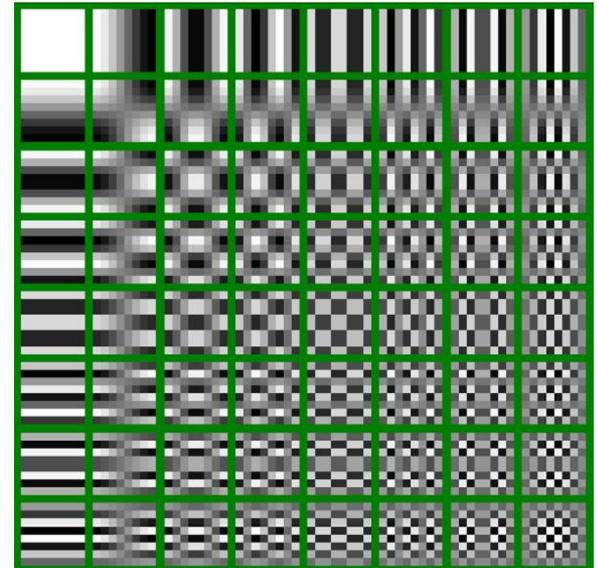
Discrete Cosine Transform

- N-мерное преобразование обладает свойством сепарабельности

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

- Коэффициенты $A[8 \times 8]$ преобразования вычисляются один раз

$$C(u, v) = A^T X A$$



Discrete Cosine Transform

- Наивный: 64 нити на блок (8x8)
 - Загрузка одного пикселя из текстуры
 - Барьер
 - Поток вычисляет один коэффициент
 - Барьер
 - Запись коэффициента в глобальную память

Насколько это эффективно?



Discrete Cosine Transform

- Блок потоков обрабатывает несколько блоков 8×8
- Один поток обрабатывает вектор 8×1 (1×8)

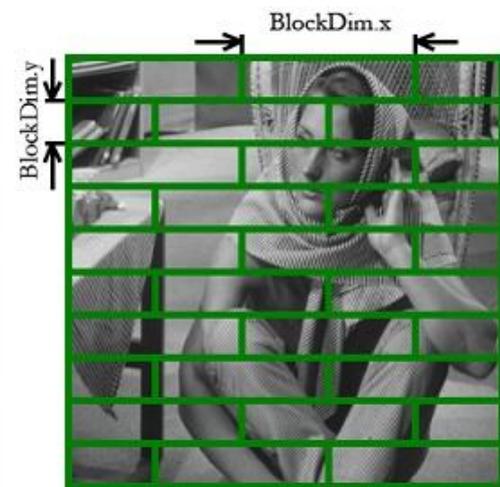
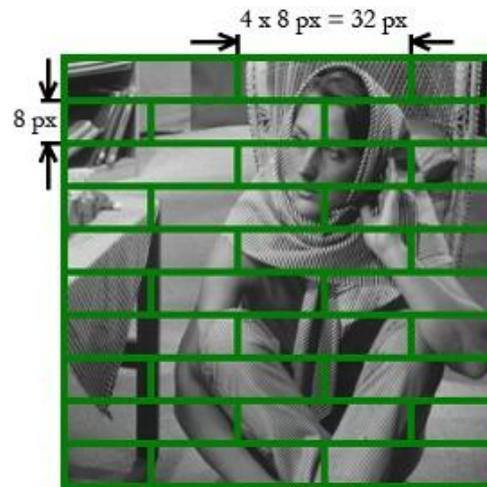
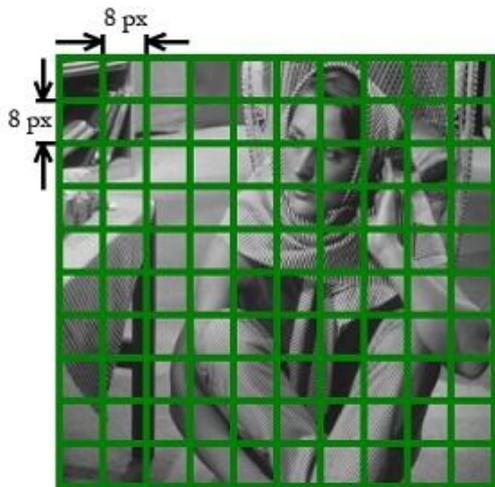


Image Denoising

- 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;
```

```
}
```

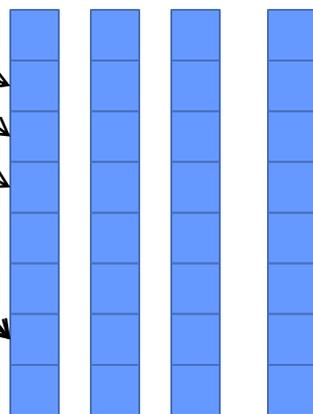
Медиана

Построение гистограммы

СИГНАЛ



Сигнал + фартук в $S_{мет}$



...

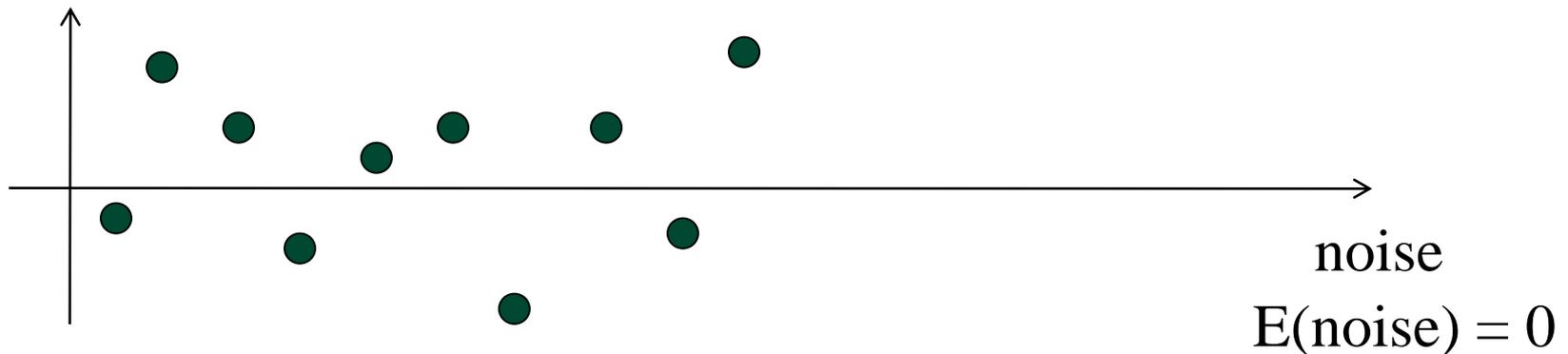


Что с банк-конфликтами?



Фильтрация (Аддитивный шум)

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

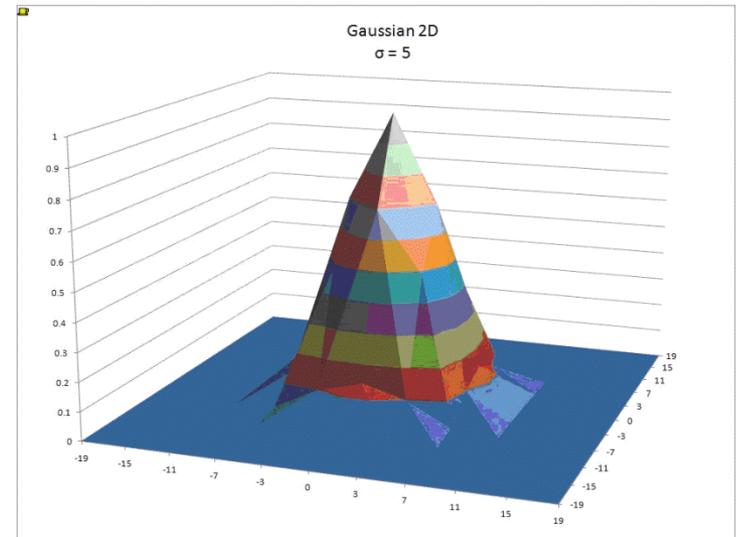
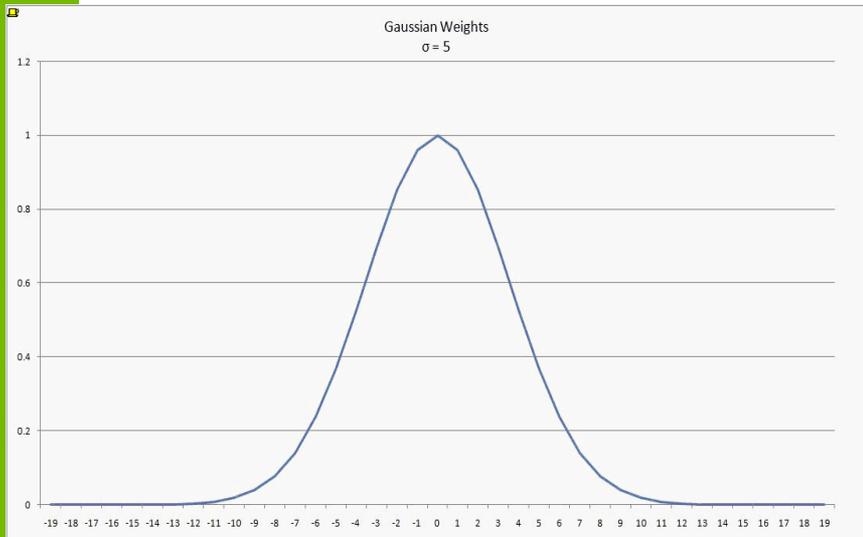


Gaussian Blur

- Blur (размытие) изображение
- Свертка с ядром:

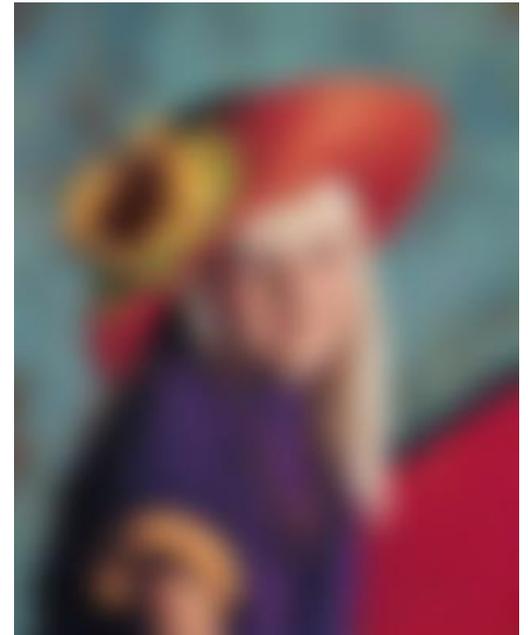
$$k_{\sigma}(i) = \exp(-i^2 / \sigma^2)$$

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



Gaussian Blur

- Blur (размытие) изображение
- Свертка с ядром:

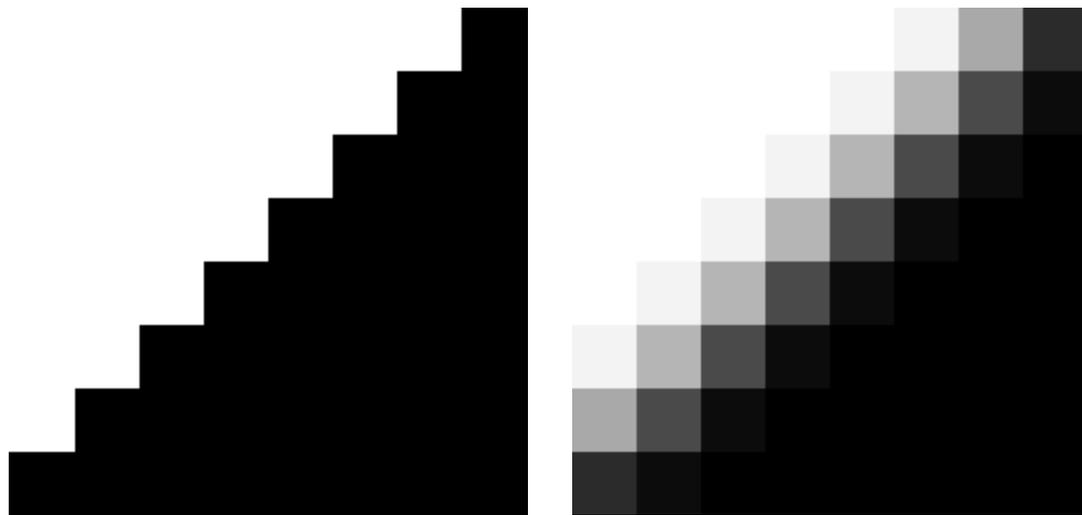


Адаптивное размытие

- Свертка с ядром:

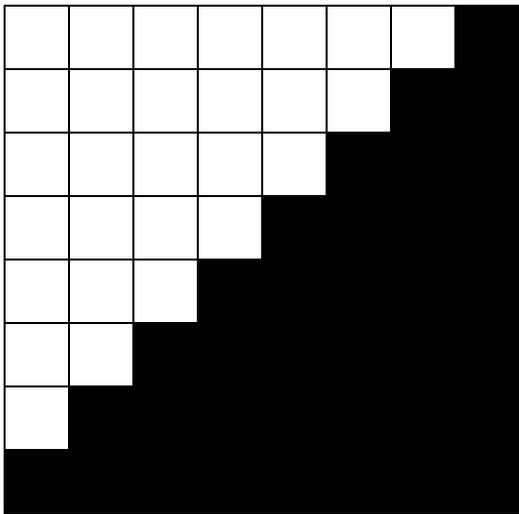
$$k_{\sigma}(i, j) = \exp(-(i^2 + j^2) / \sigma^2) \exp(-ClrSpaceDist(i, j) / h^2)$$

- *ClrSpaceDist* - это фотометрическая близость

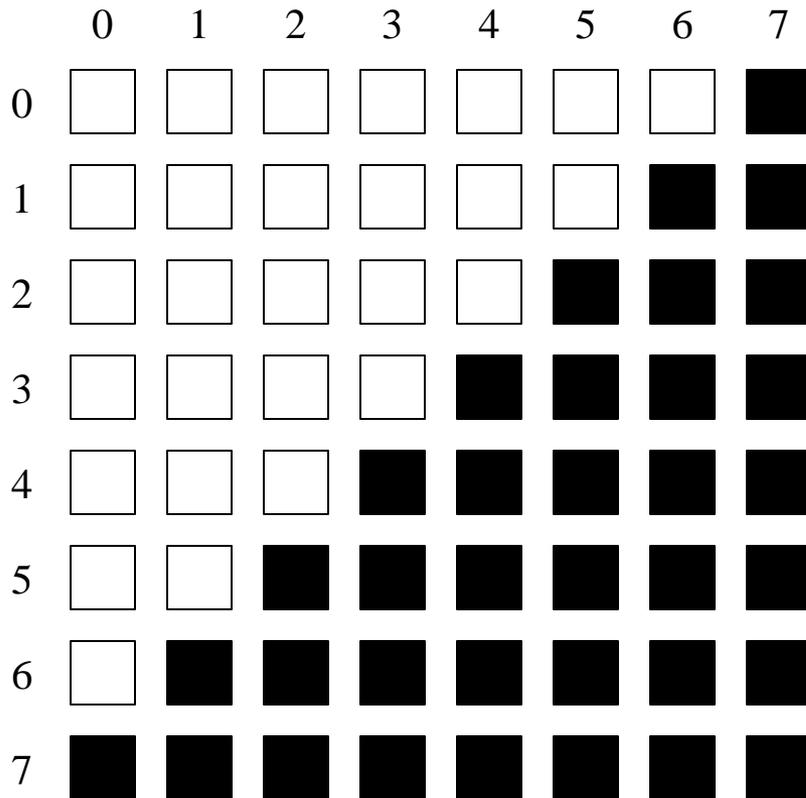


Bilateral

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



Bilateral



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

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

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

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

Bilateral Kernel

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

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

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

    filteredImage[idx + idy * W] = result;
}
```

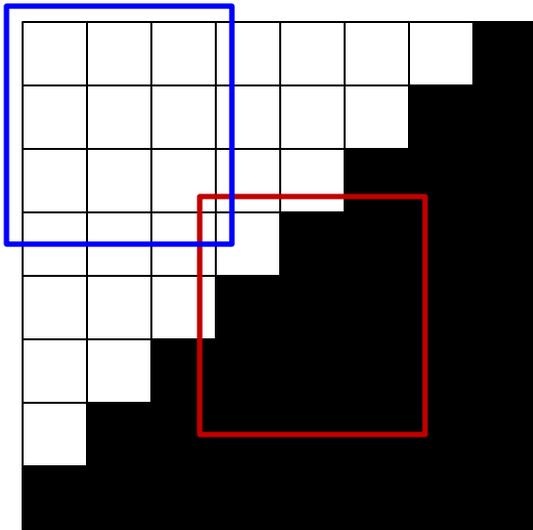
Bilateral Оптимизации

- 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.)

Non Local Means

- *ClrSpaceDist* - оценивать по блокам пикселей

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



Non Local Means

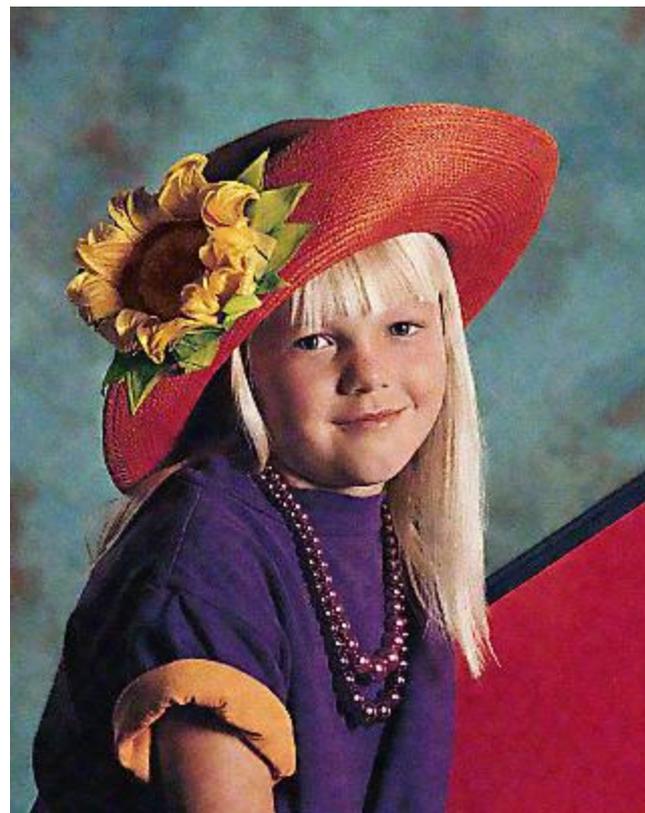
- На вычисление одного веса:
 - $N_b \times N_b$ вычислений, N размер блока
- На фильтрацию одного пиксела:
 - $N_b \times N_b \times R \times R$, R размер окна



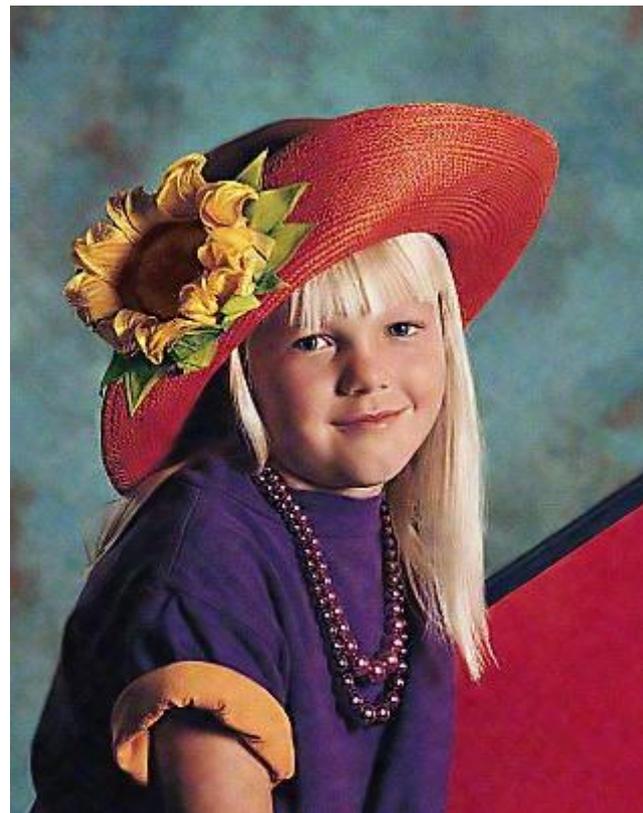
Сравнение



Сравнение Bilateral



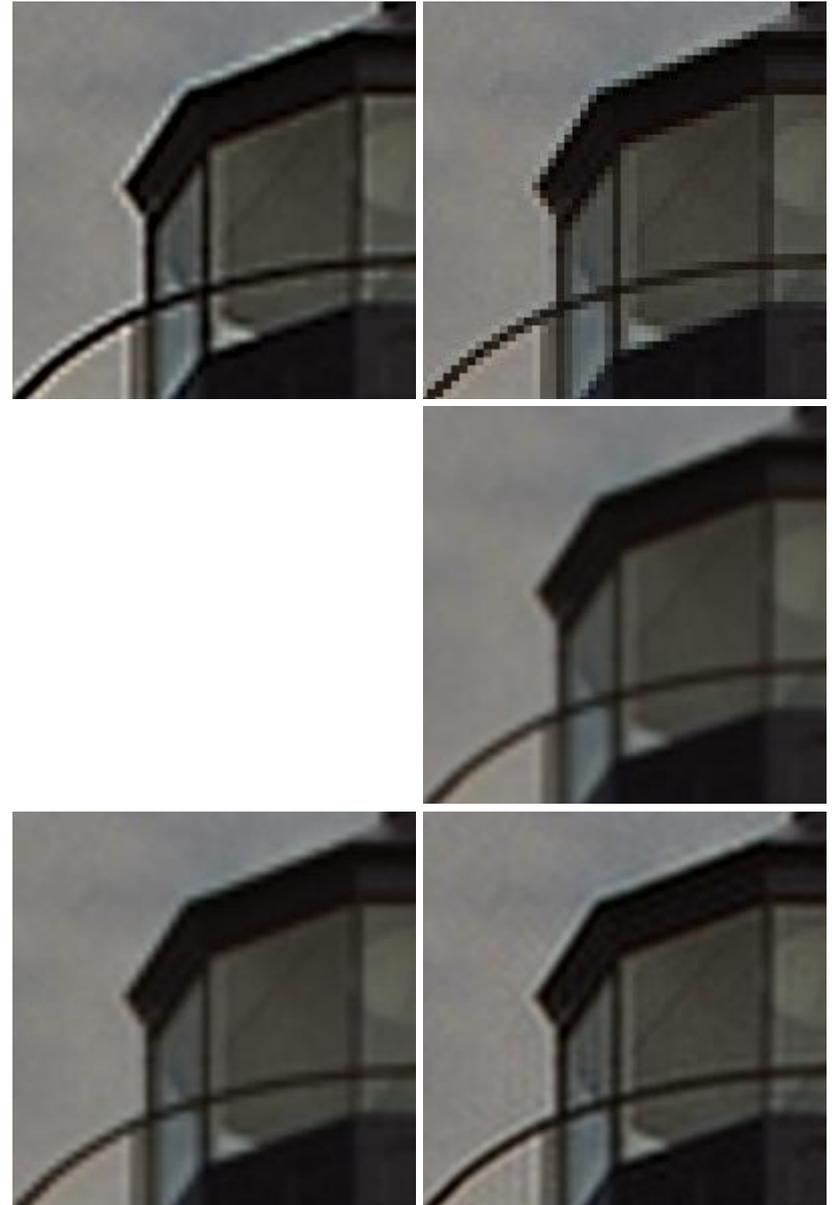
Сравнение NLM



МАСШТАБИРОВАНИЕ ИЗОБРАЖЕНИЙ

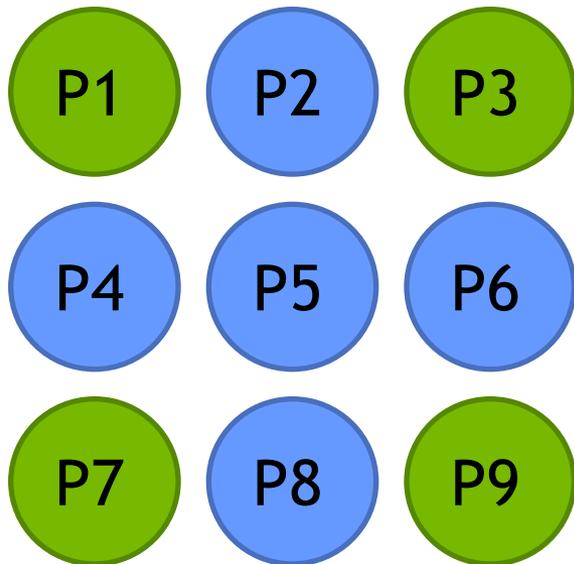
Артифакты

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



Простые методы

- Билинейная интерполяция



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

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

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

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

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

Легенда:

-исходные
пиксели



-искомые
пиксели

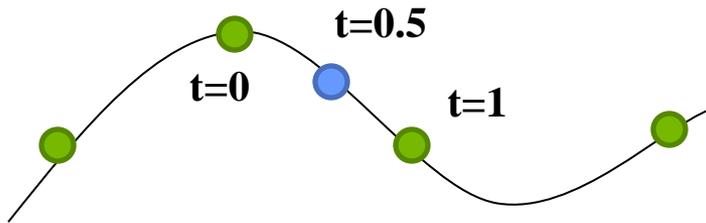


Простые методы

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

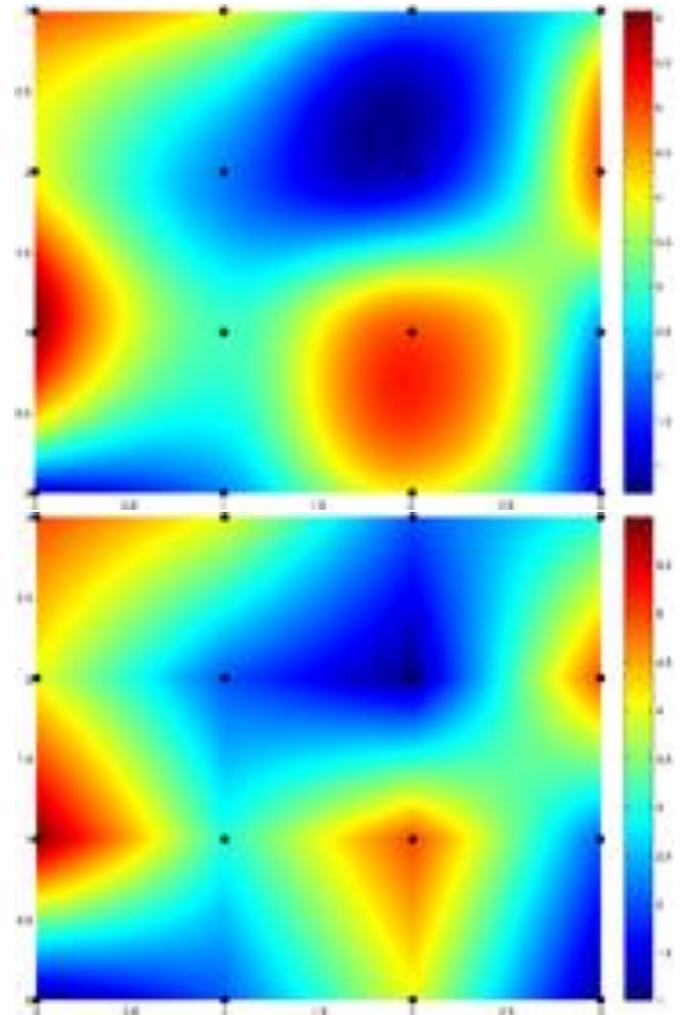
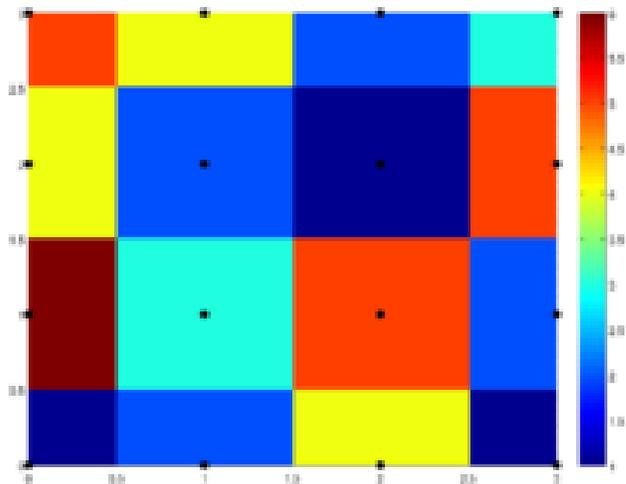
Простые методы

- Бикубическая интерполяция
 - Сепарабельная
 - Лучшее качество



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

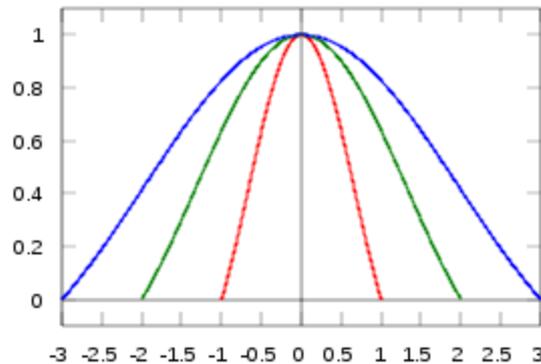
Сравнение



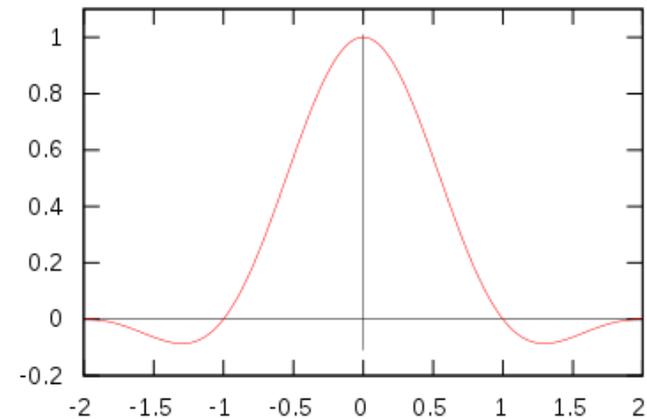
Lanczos

$$x(t) = \sum x(k\Delta t) \frac{\sin(\pi F_D (t - k\Delta t))}{\pi F_D (t - k\Delta t)}$$

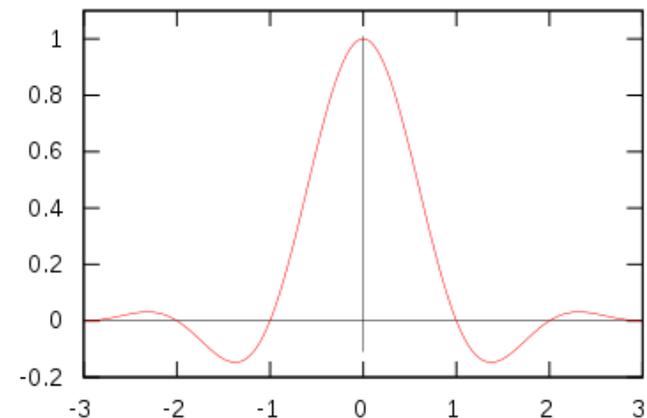
$$L(x) = \begin{cases} \sin c(x) \sin c(x/a), & -a < x < a, x \neq 0 \\ 1 & x = 0 \\ 0 & \text{иначе} \end{cases}$$



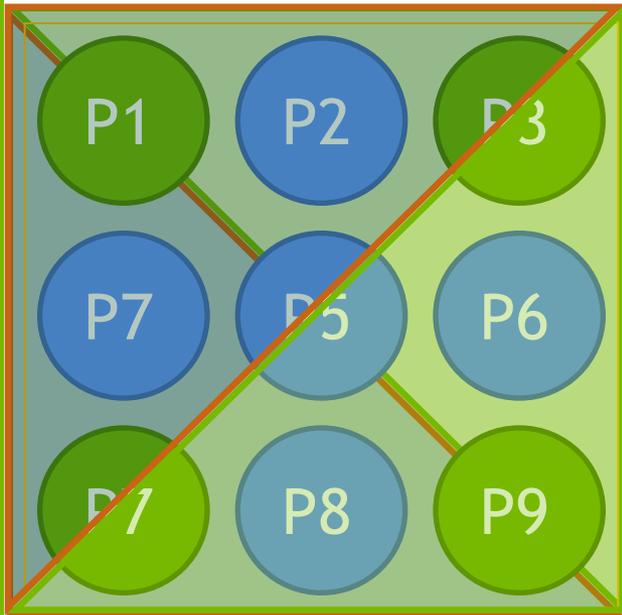
Lanczos kernel for a=2



Lanczos kernel for a=3



Gradient interpolation



$$D_{xd} = \text{abs}(P3 - P5)$$

$$D_{yd} = \text{abs}(P1 - P9)$$

If ($D_{xd} > D_{yd}$)

//граница P1P5P9

$$P5 = (P1 + P9) * 0.5f;$$

If ($D_{yd} > D_{xd}$)

//граница P3P5P7

$$P5 = (P3 + P7) * 0.5f;$$

If ($D_{yd} \sim D_{xd}$)

//граница не определена

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

Легенда:

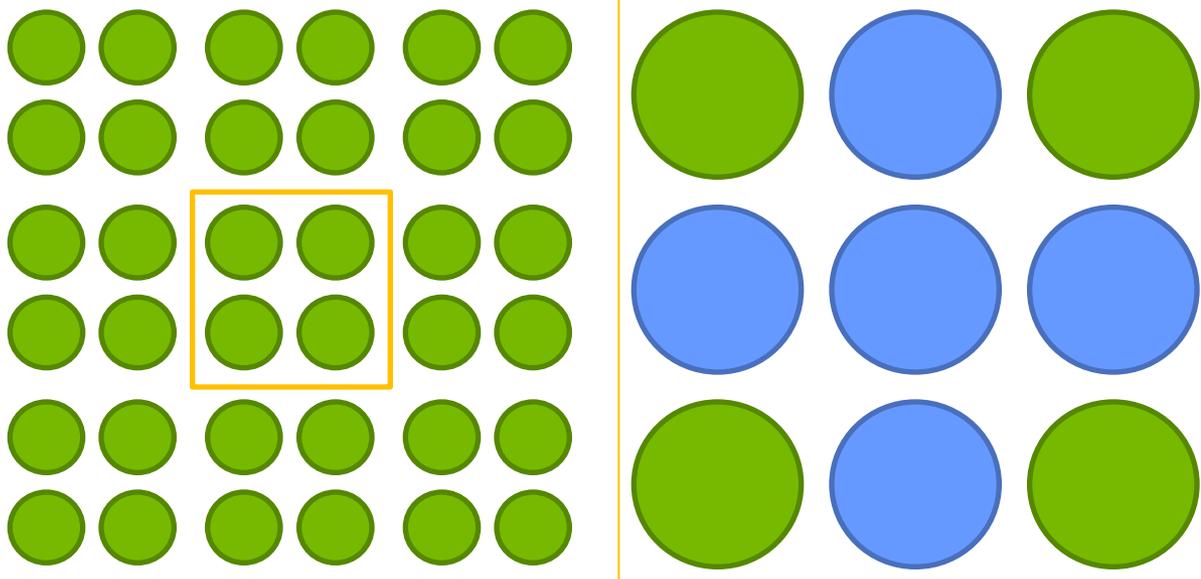
-исходные
пиксели



-искомые
пиксели



NEDI (New Edge-Directed Interpolation)



Легенда:

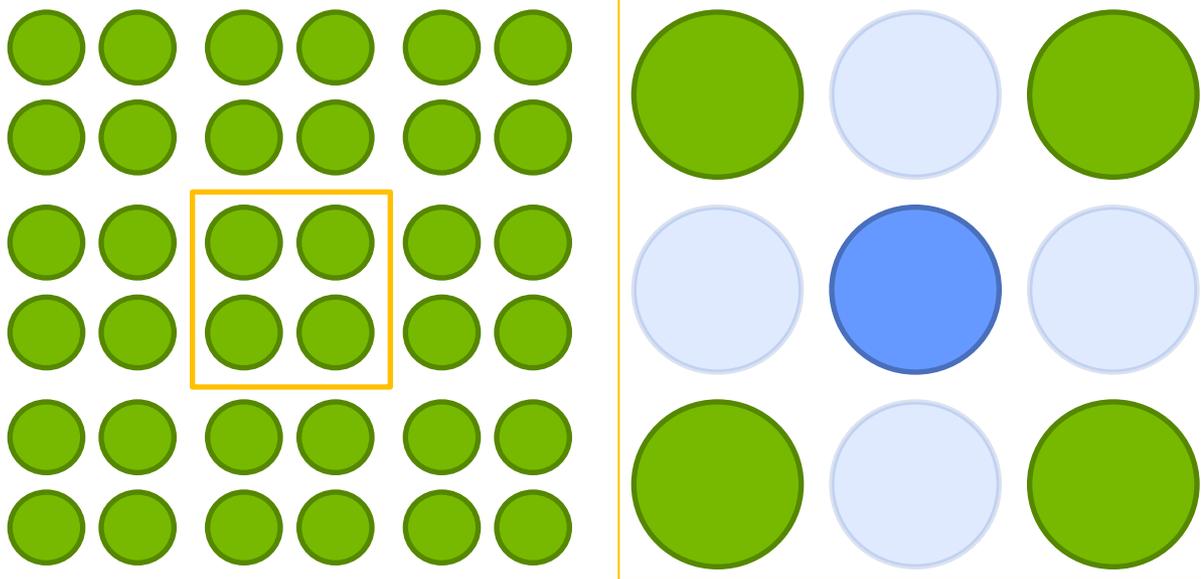
-исходные
пиксели



-искомые
пиксели



NEDI (New Edge-Directed Interpolation)



Легенда:

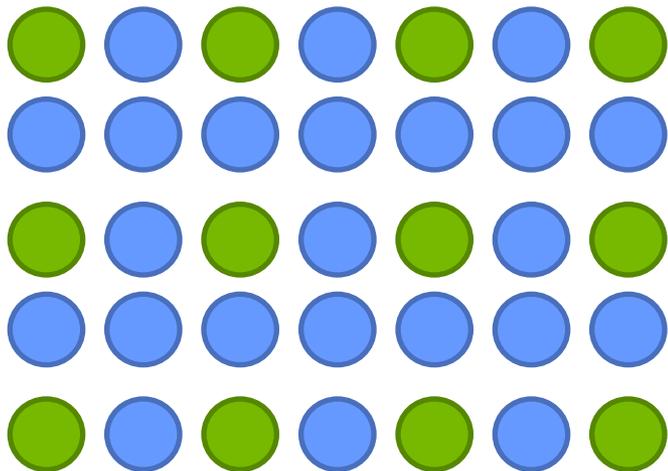
-исходные
пиксели



-искомые
пиксели



NEDI (New Edge-Directed Interpolation)



Легенда:

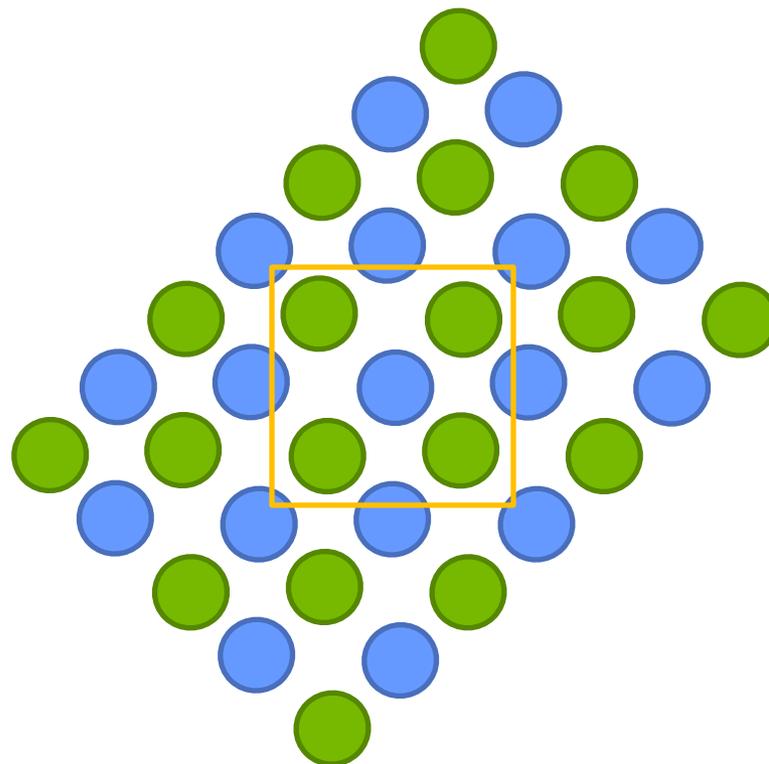
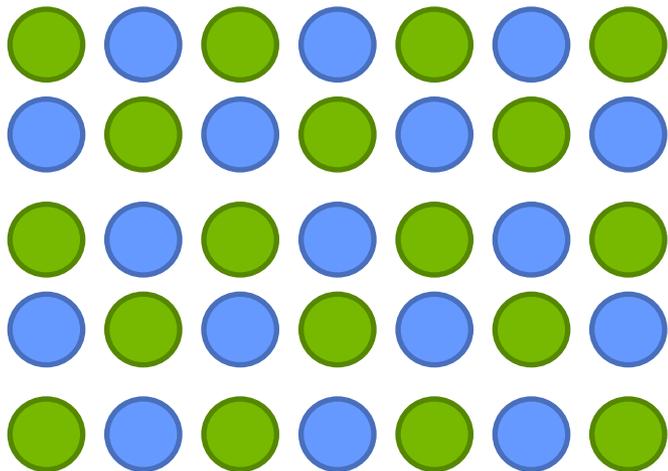
-исходные
пиксели



-искомые
пиксели



NEDI (New Edge-Directed Interpolation)



Легенда:

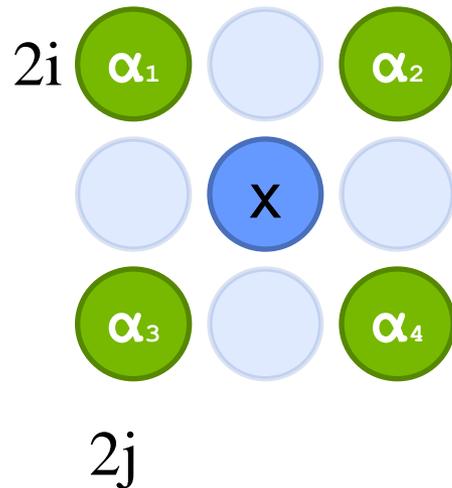
-исходные
пиксели



-искомые
пиксели



NEDI



$$X = F(2i+1, 2j+1) = \alpha_1 * F(2i, 2j) + \alpha_2 * F(2i+2, 2j) + \alpha_3 * F(2i, 2j+2) + \alpha_4 * F(2i+2, 2j+2) ;$$

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

Легенда:

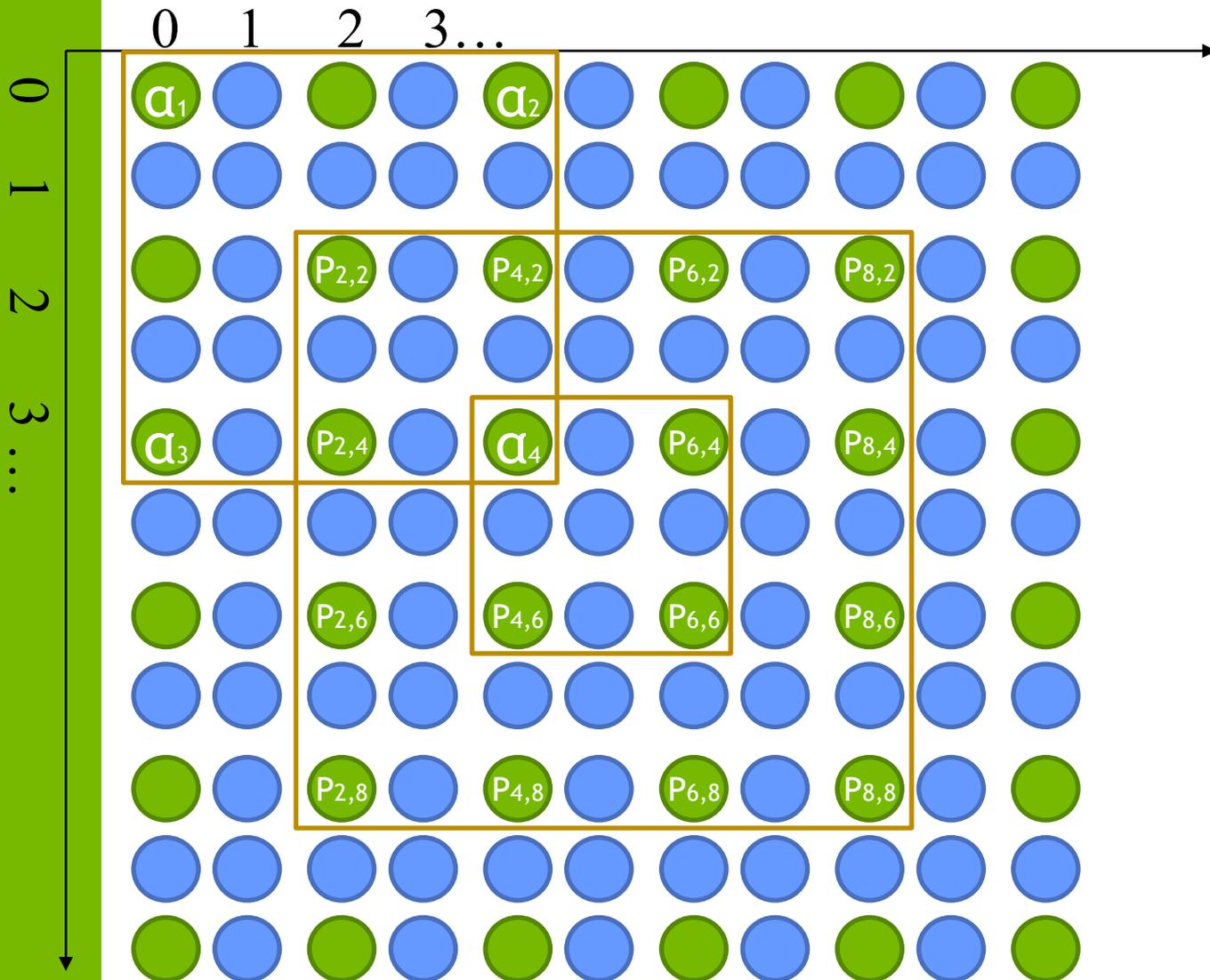
-исходные
пиксели



-искомые
пиксели



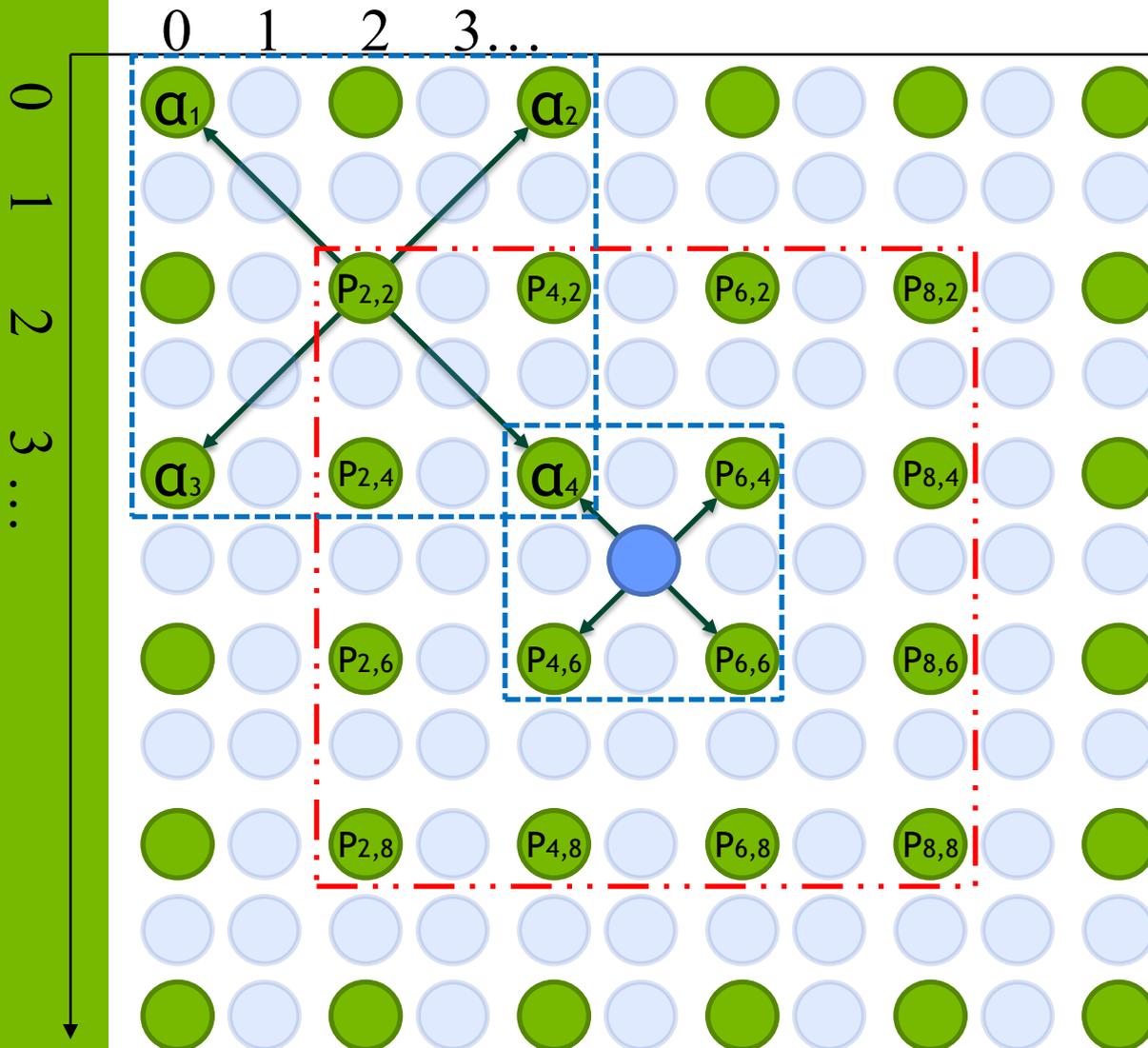
NEDI



Легенда:

- исходные пиксели 
- искомые пиксели 

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 $j = 2,4,6,8$

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

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

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

$$\alpha$$

$$P = \{P_{i,j}\}$$

$$C\alpha = P$$

$$\partial(\text{SSE}) / \partial \alpha = 0.0$$

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

Легенда:

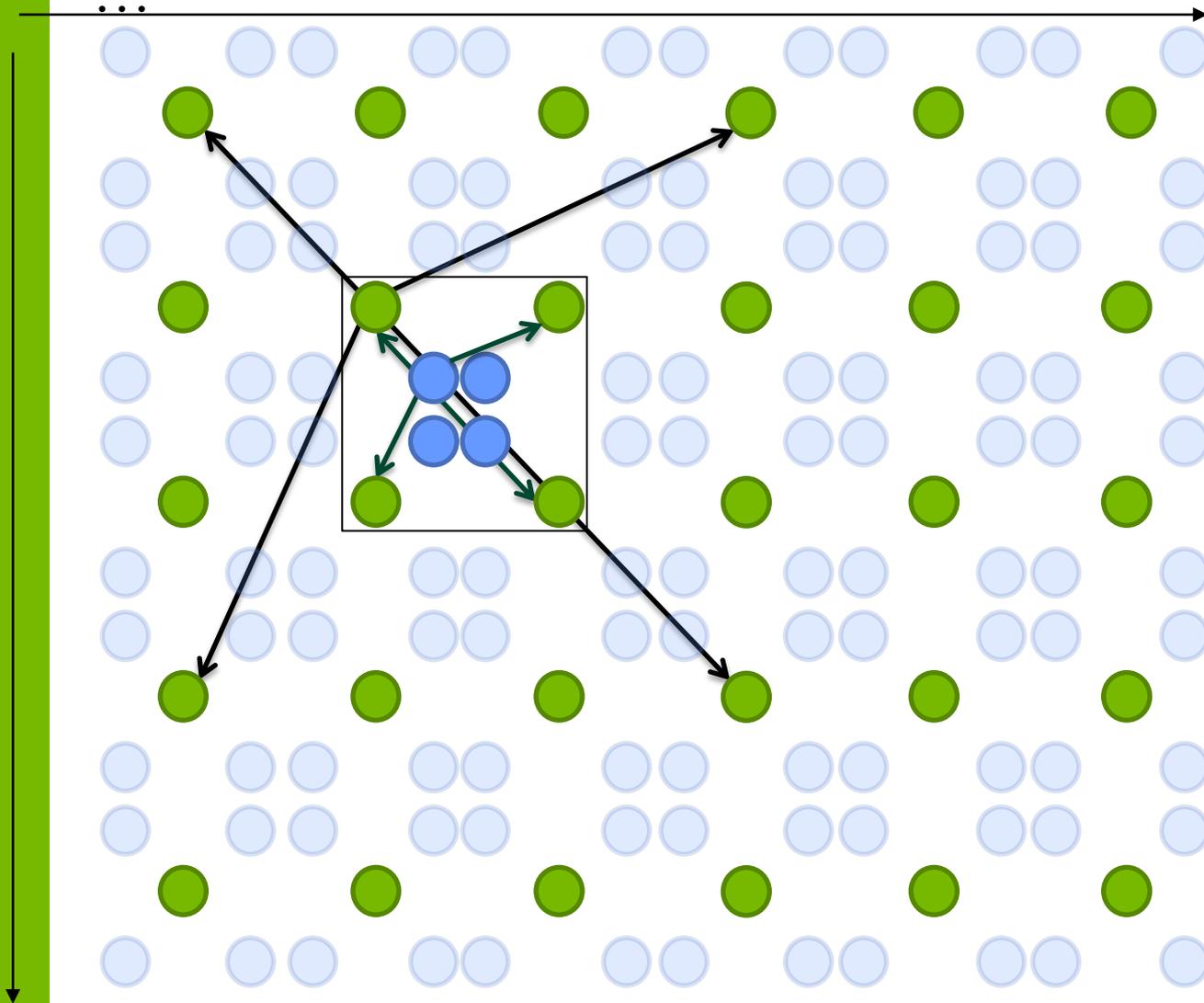
-исходные
пиксели



-искомые
пиксели



NEDI (improvement)



Легенда:

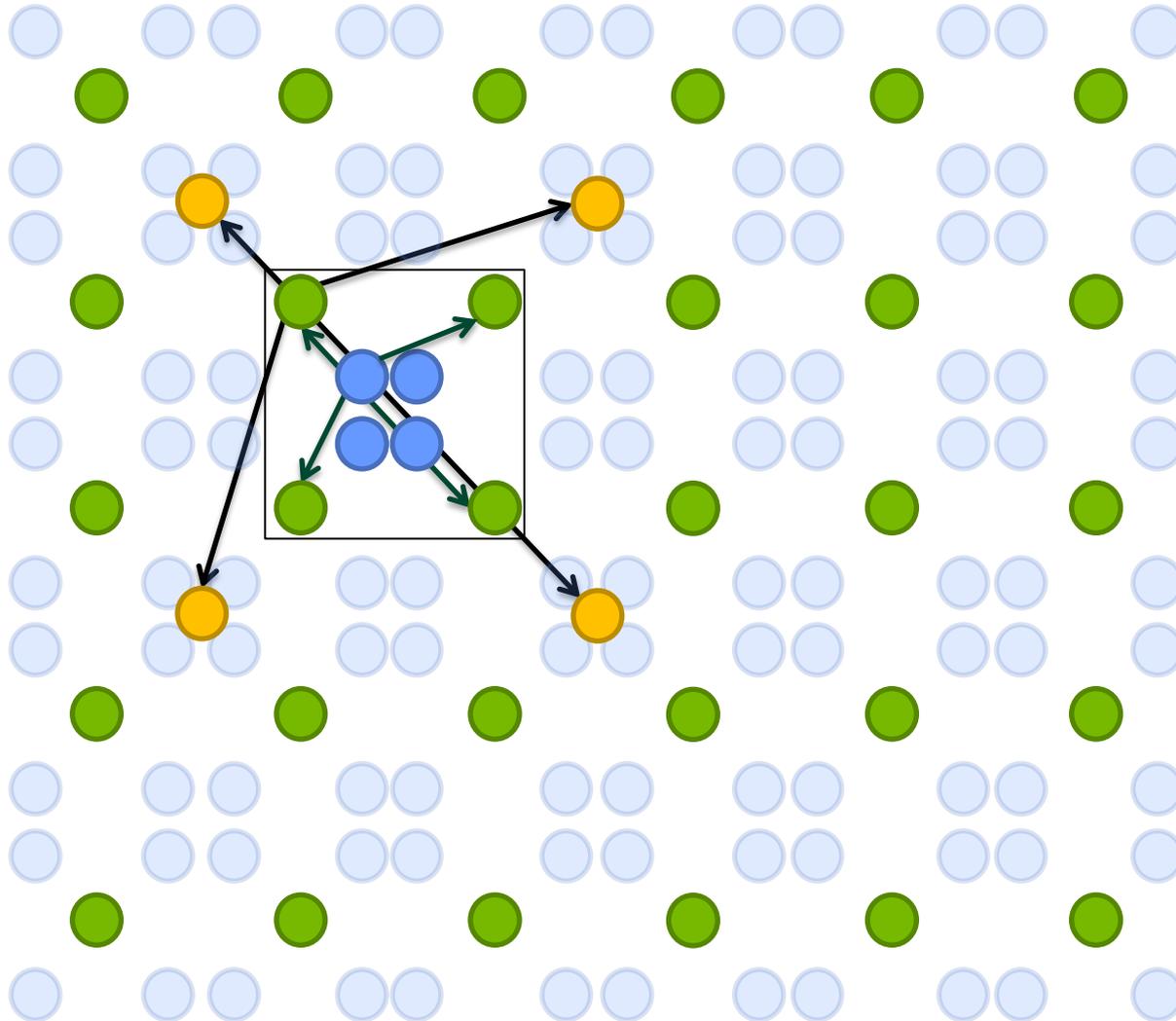
-исходные
пиксели



-искомые
пиксели



NEDI (improvement)



Легенда:

-исходные
пиксели



-искомые
пиксели



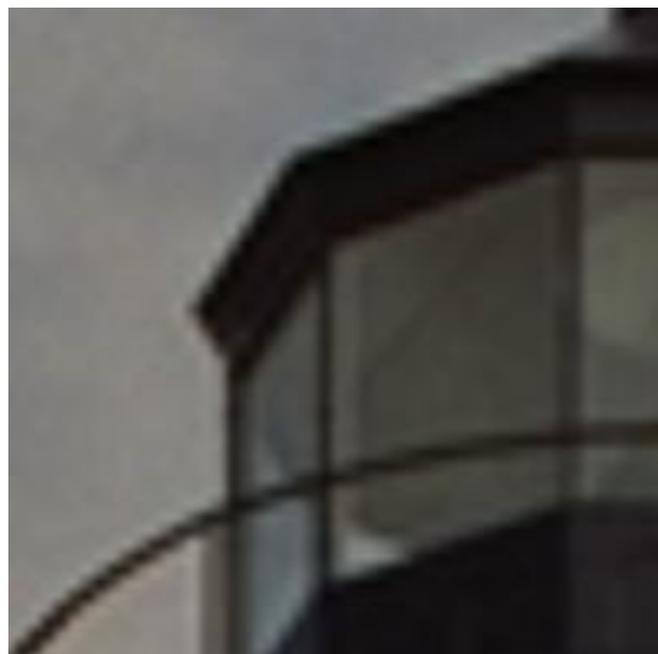
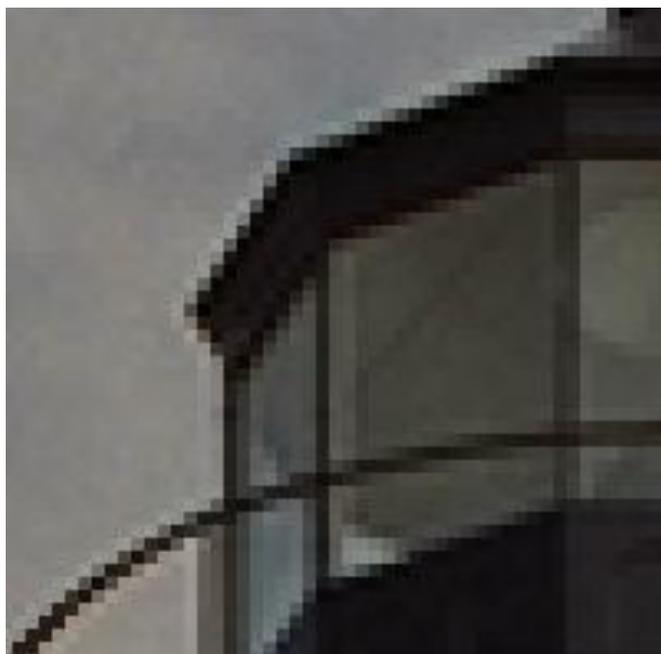
- вспомогательные
пиксели



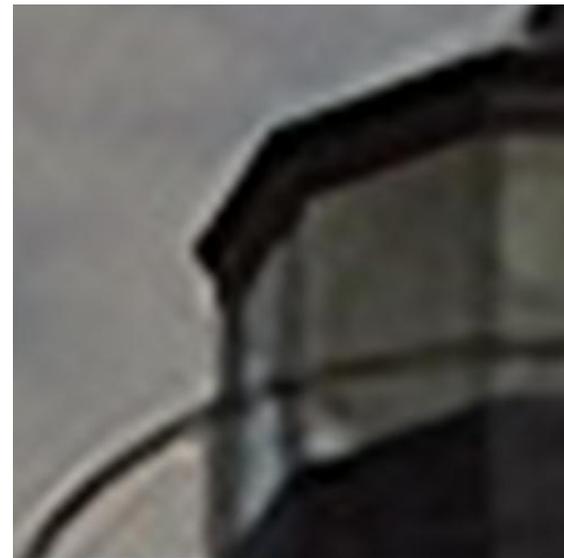
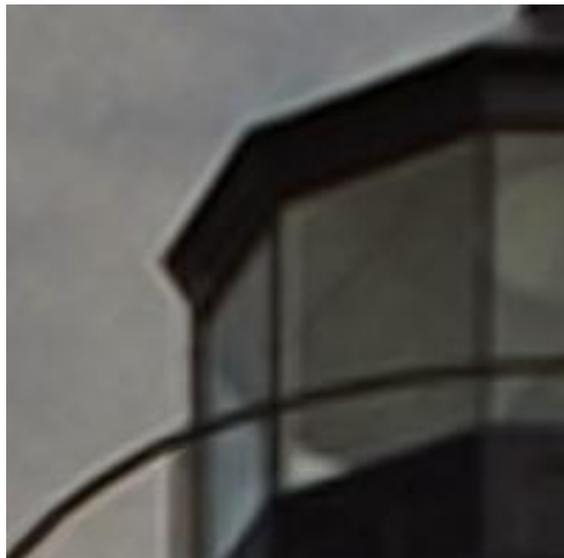
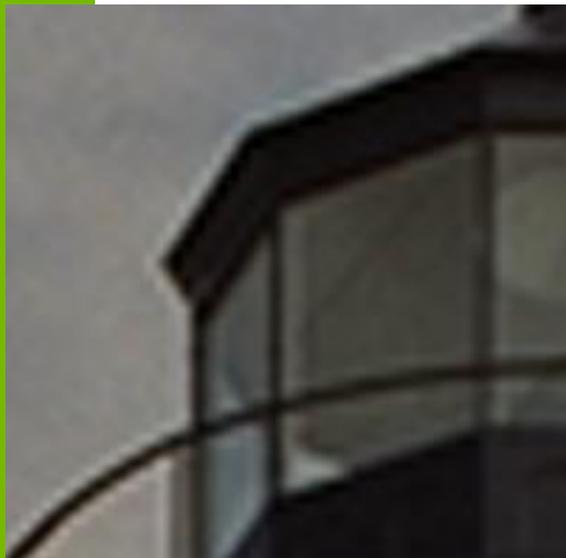
NEDI: Pros and Cons

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

Сравнение



Сравнение



Вопросы



Ресурсы нашего курса

- [Steps3d.Narod.Ru](#)
- [Google Site CUDA.CS.MSU.SU](#)
- [Google Group CUDA.CS.MSU.SU](#)
- [Google Mail CS.MSU.SU](#)
- [Google SVN](#)
- [Tesla.Parallel.Ru](#)
- [Twirpx.Com](#)
- [Nvidia.Ru](#)

Преобразование Фурье

- Линейный оператор вида:

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

- Обратный оператор:

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

Преобразование Фурье

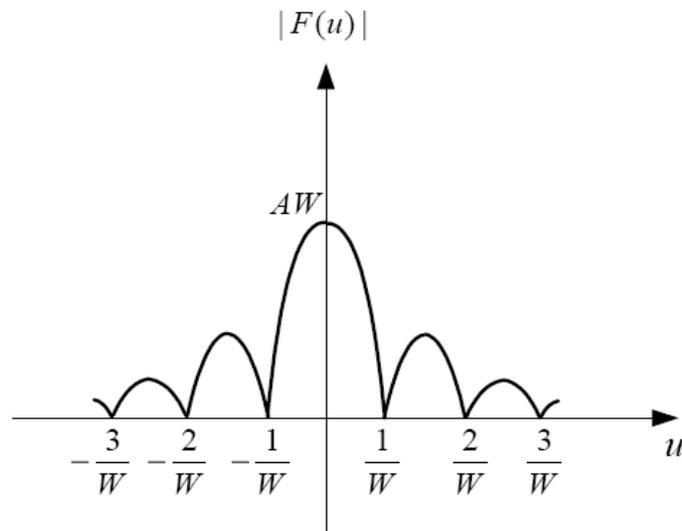
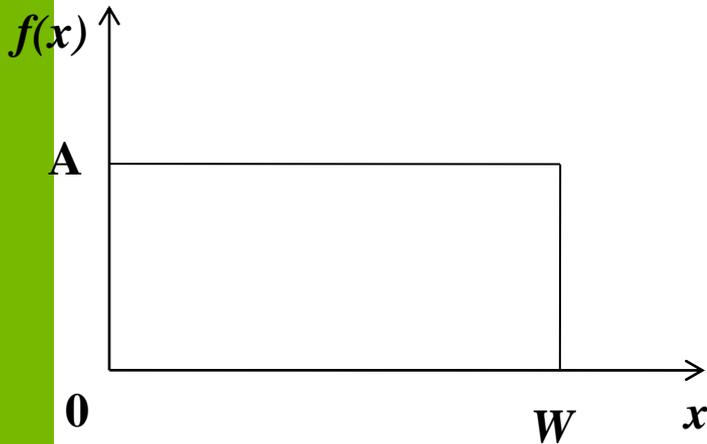
- Условие существования

- $\int_{-\infty}^{+\infty} |f(x)| dx < \infty$ $\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} |f(x, y)| dx dy < \infty$

- Конечное число устранимых разрывов

Преобразование Фурье

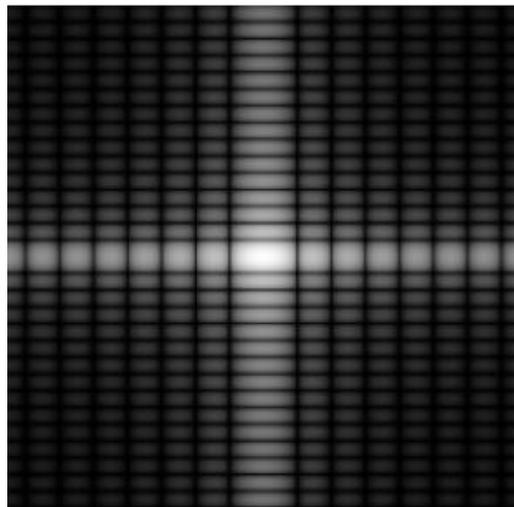
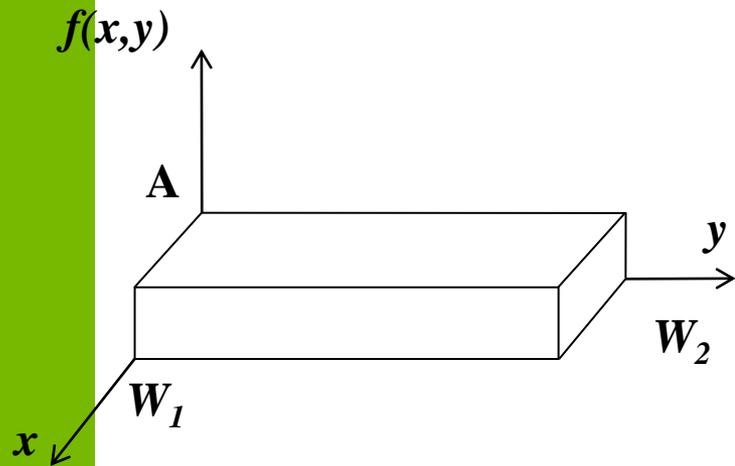
Пример 1D



$$\begin{aligned}
 F(u) &= \int_{-\infty}^{+\infty} f(x) e^{-2\pi i \cdot x u} dx = A \int_0^W e^{-2\pi i \cdot x u} dx \\
 &= \frac{-A}{2\pi i u} \left[e^{-2\pi i \cdot x u} \right]_0^W = \frac{-A}{2\pi i u} \left[e^{-2\pi i \cdot W u} - 1 \right] \\
 &= \frac{-A}{2\pi i u} e^{-\pi i \cdot W u} \left[e^{-\pi i \cdot W u} - e^{\pi i \cdot W u} \right] \\
 &= \frac{-A}{2\pi i u} e^{-\pi i \cdot W u} \left[-2i \sin(\pi u W) \right] \\
 &= AW \frac{\sin(\pi u W)}{\pi u W} e^{-\pi i \cdot W u} \\
 &= AW \operatorname{sinc}(uW) e^{-\pi i \cdot W u} \\
 |F(u)| &= AW \operatorname{sinc}(uW)
 \end{aligned}$$

Преобразование Фурье

Пример 2D



$$\begin{aligned} F(u, v) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{-2\pi i(ux+vy)} dx dy \\ &= A \int_0^{W_1} e^{-2\pi i u x} dx \int_0^{W_2} e^{-2\pi i v y} dy \\ &= A \left[\frac{e^{-2\pi i u x}}{-2\pi i u} \right]_0^{W_1} \left[\frac{e^{-2\pi i v y}}{-2\pi i v} \right]_0^{W_2} \\ &= A W_1 W_2 \operatorname{sinc}(u W_1) \operatorname{sinc}(v W_2) e^{-\pi i(u W_1 + v W_2)} \end{aligned}$$

$$|F(u, v)| = A W_1 W_2 |\operatorname{sinc}(u W_1)| |\operatorname{sinc}(v W_2)|$$

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. $F\{f(ax, by)\} = \frac{F(u/a, v/b)}{|ab|}$
7. $F\{f(r, \theta + \theta_0)\} = F(w, \phi + \theta_0)$

Преобразование Фурье

Свойства

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

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

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

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

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

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

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

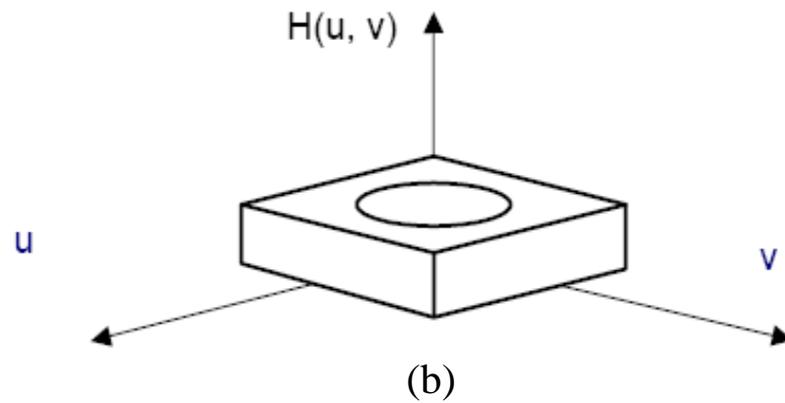
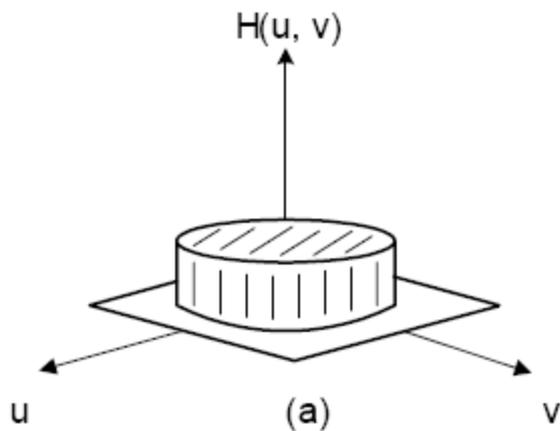
4. $F(u, v) = \int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} f(x, y) e^{-2\pi i \cdot ux} dx \right] e^{-2\pi i \cdot vy} dy = \int_{-\infty}^{+\infty} F(u, y) e^{-2\pi i \cdot vy} dy$

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

Фильтры

a) Низкочастотные (low-pass)

b) Высокочастотные (high-pass)



Фильтры

