Иерархия памяти CUDA. Текстуры в CUDA. Цифровая обработка сигналов

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

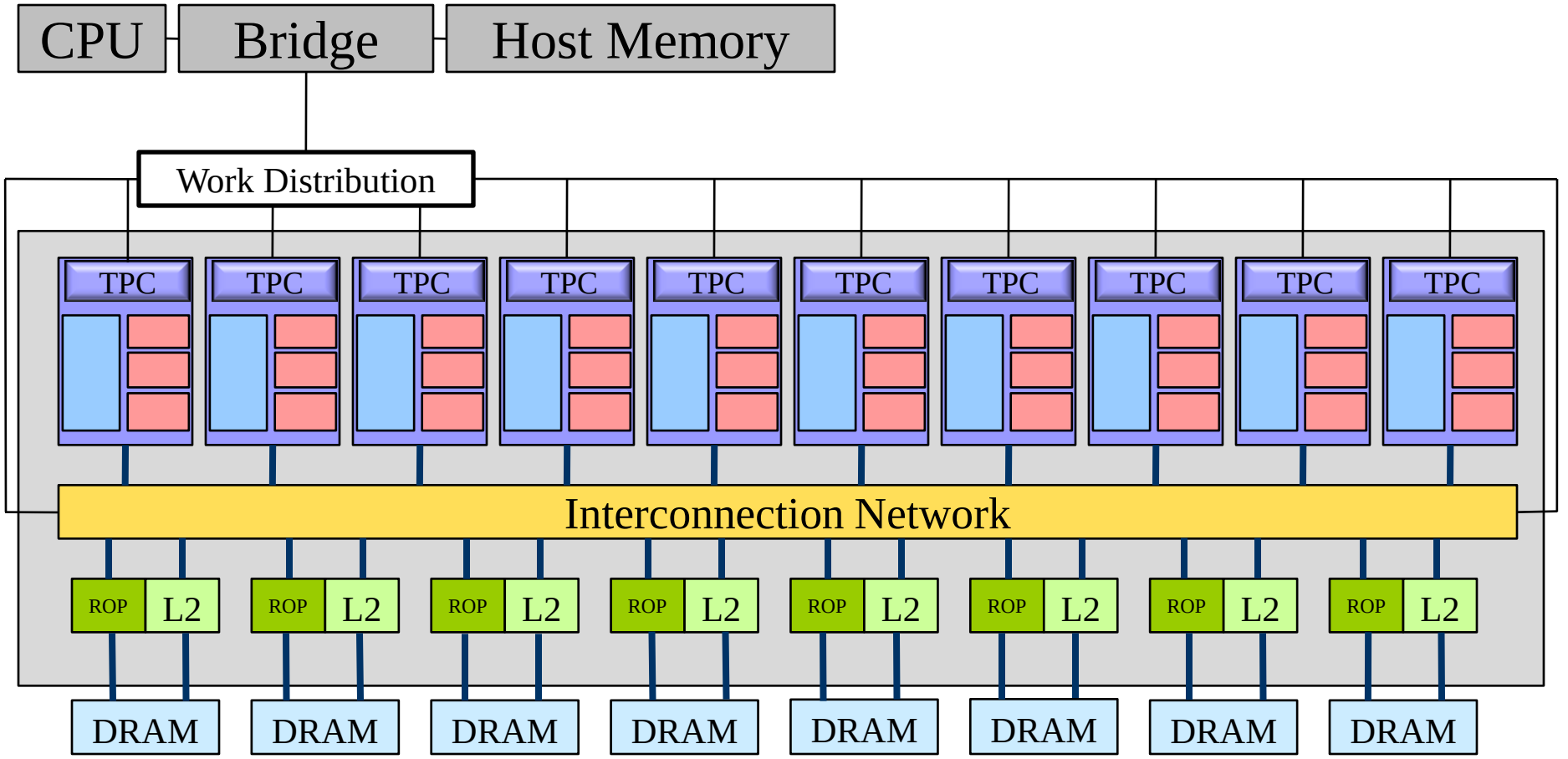
Типы памяти в CUDA

Тип памяти	Доступ	Уровень выделения	Скорость работы
Регистры	R/W	Per-thread	Высокая(on-chip)
Локальная	R/W	Per-thread	Низкая (DRAM)
Shared	R/W	Per-block	Высокая(on-chip)
Глобальная	R/W	Per-grid	Низкая (DRAM)
Constant	R/O	Per-grid	Высокая(L1 cache)
Texture	**R/O**	**Per-grid**	[-] **Низкая(DRAM)** [+] **L1 cache**

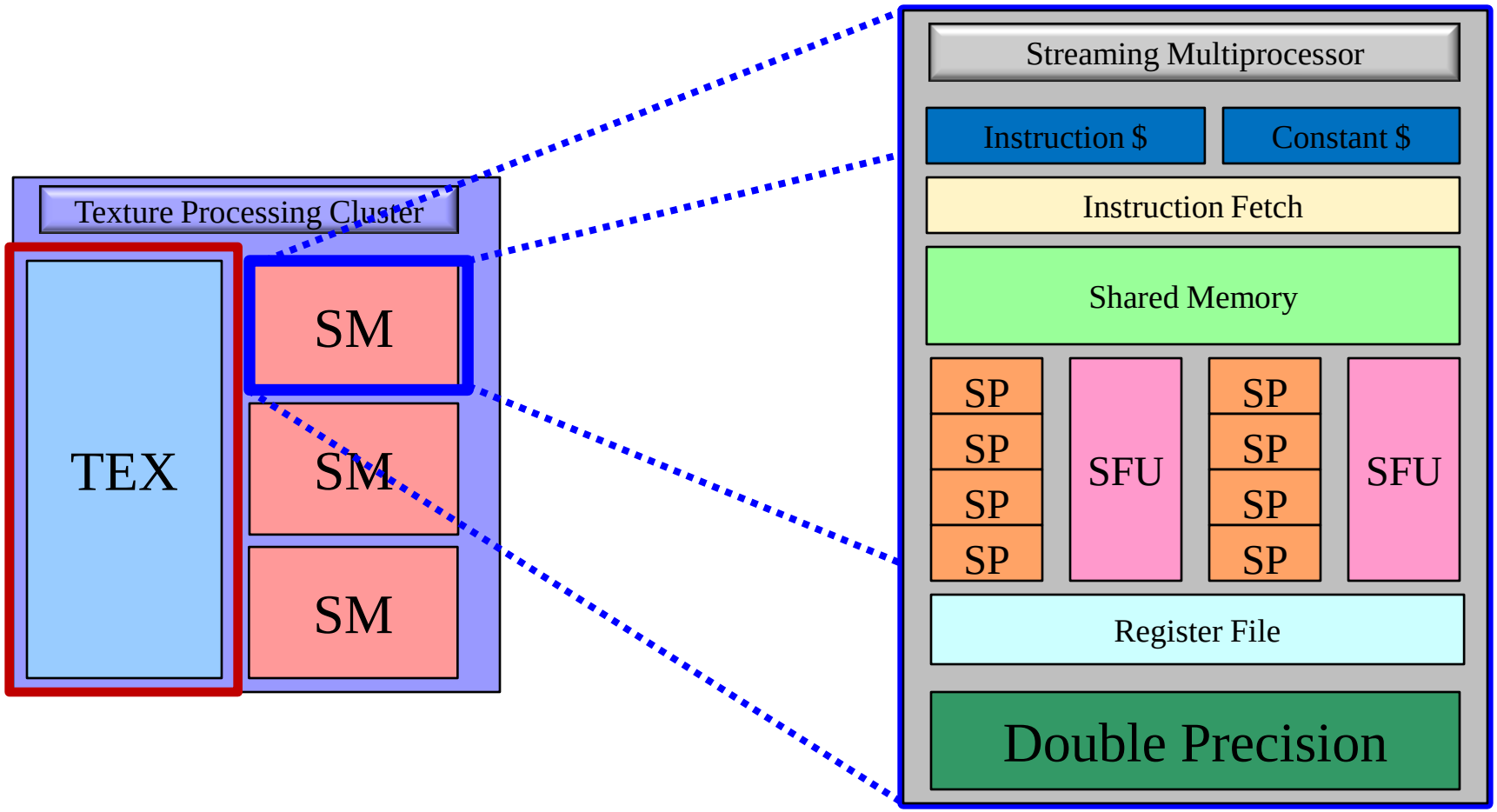
Архитектура

- Сегодня мы рассмотрим Texture Unit

Архитектура Tesla 10

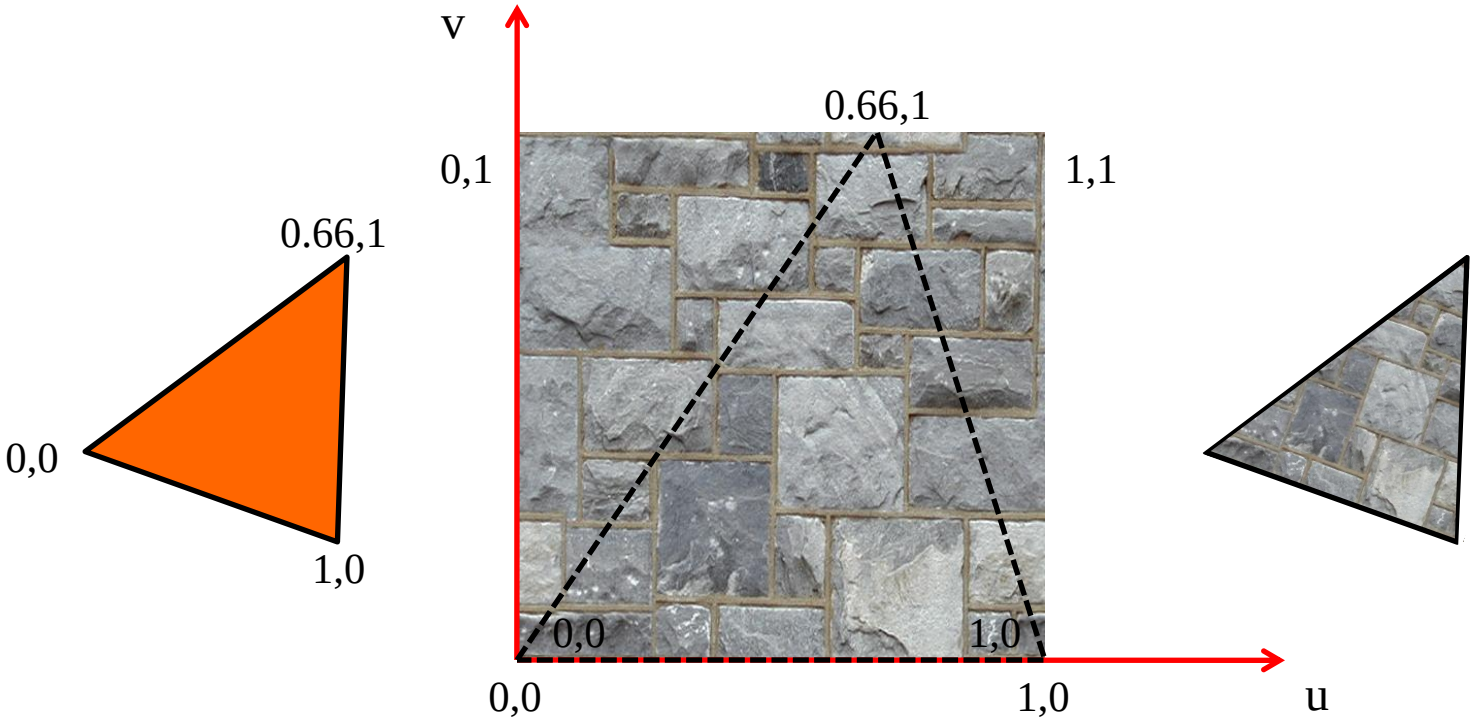


Архитектура Tesla
Мультипроцессор Tesla 10



Texture в 3D

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



Texture HW

- Латентность больше, чем у прямого обращения в память
  - Дополнительные стадии в конвеере:
    - Преобразование адресов
    - Фильтрация
    - Преобразование данных
- Но зато есть кэш
  - Разумно использовать, если:
    - Объем данных не влезает в shared память
    - Паттерн доступа хаотичный
    - Данные переиспользуются разными потоками

Texture в CUDA (cudaArray)

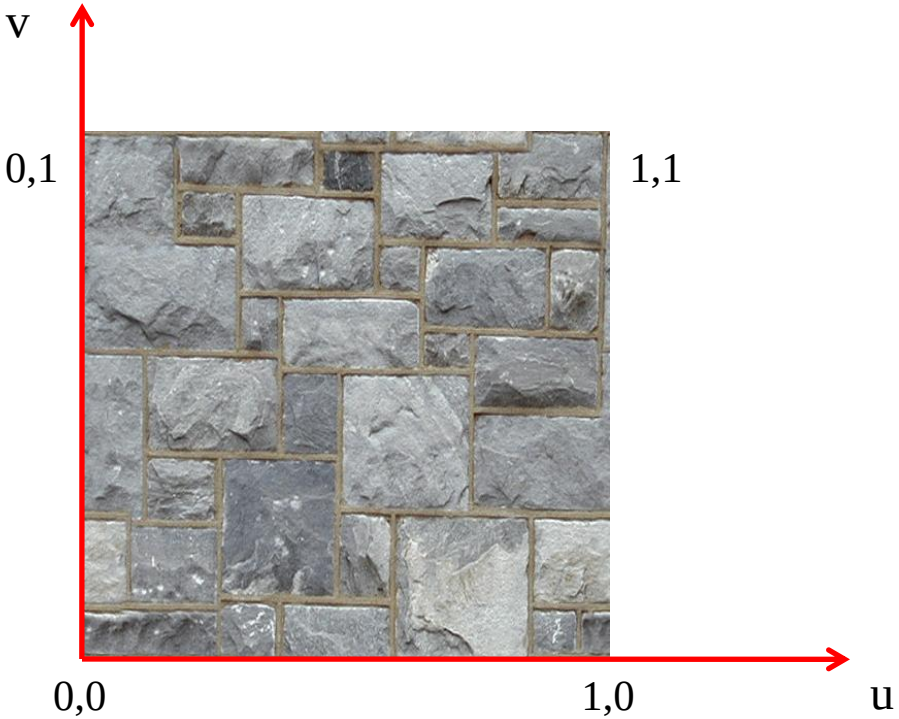
- Особый контейнер памяти: cudaArray
- Черный ящик для приложения
- Позволяет организовывать данные в 1D/ 2D/3D массивы данных вида:
  - 1/2/4 компонентные векторы
  - 8/16/32 bit signed/unsigned integers
  - 32 bit float
  - 16 bit float (driver API)
- Доступ по семейству функций tex1D()/tex2D()/tex3D()

Texture в CUDA (cudaArray)

- Особенности текстур:
  - 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`

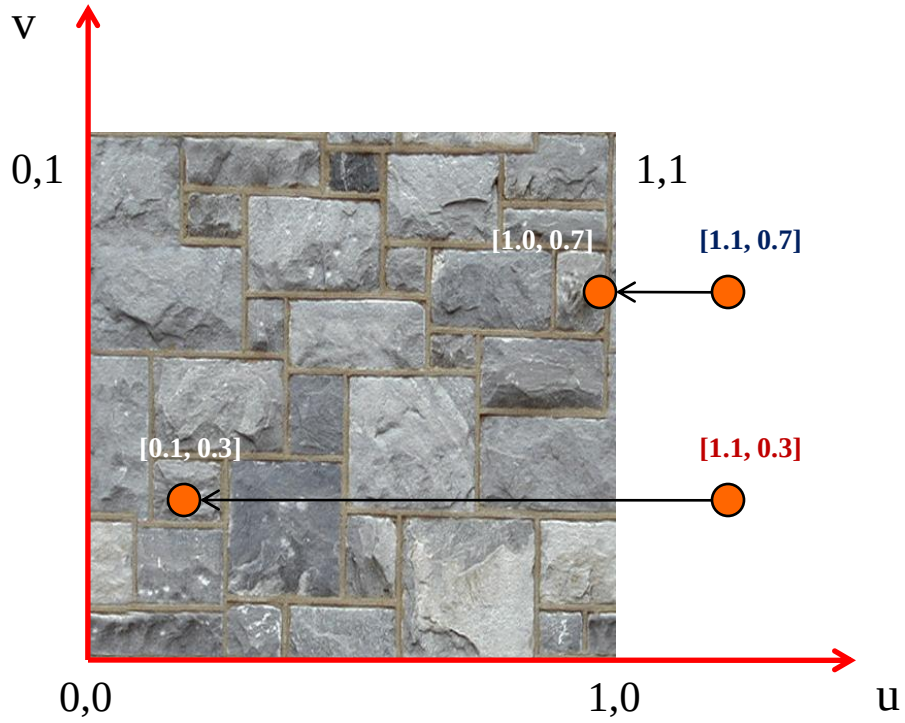
Texture в CUDA

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



Texture в CUDA

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



Clamp:

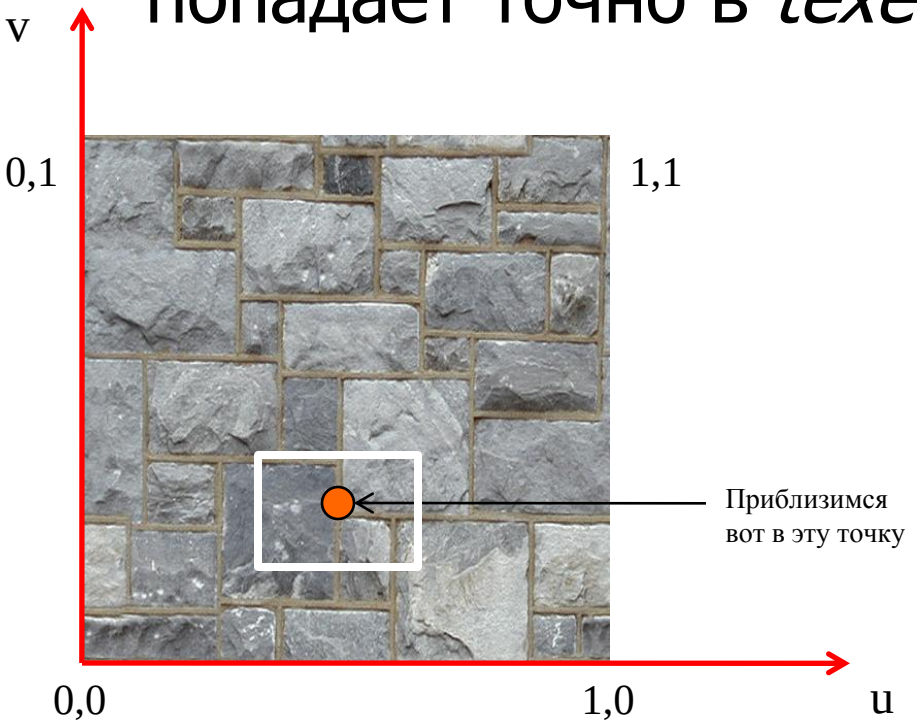
-Координата «обрубается» по допустимым границам

Wrap

- Координата «заворачивается» в допустимый диапозон

Texture в CUDA

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



Point:

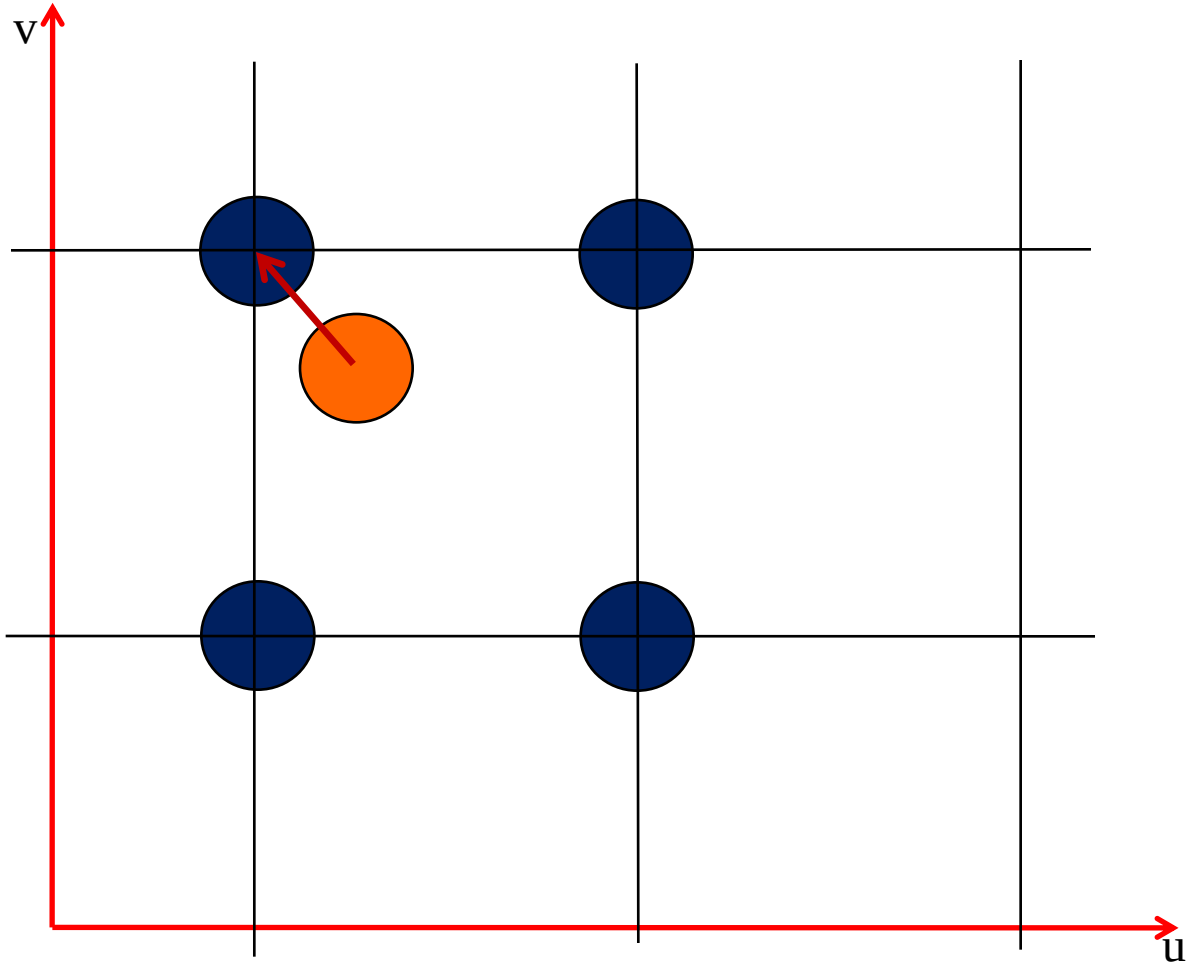
-Берется ближайший texel

Linear:

- Билинейная фильтрация

Texture в CUDA

- Фильтрация

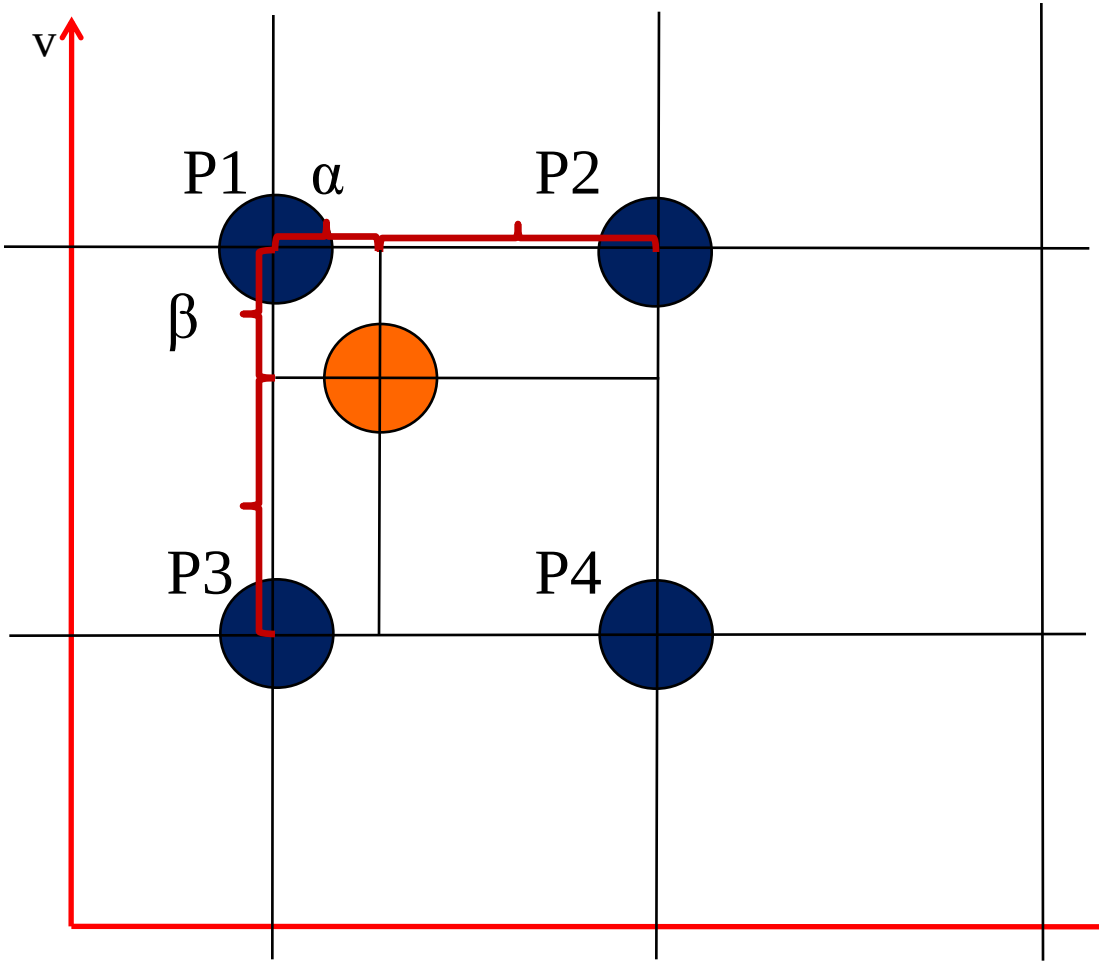


Point:

-Берется ближайший texel

Texture в CUDA

- Фильтрация



Linear:

-Билинейная фильтрация

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

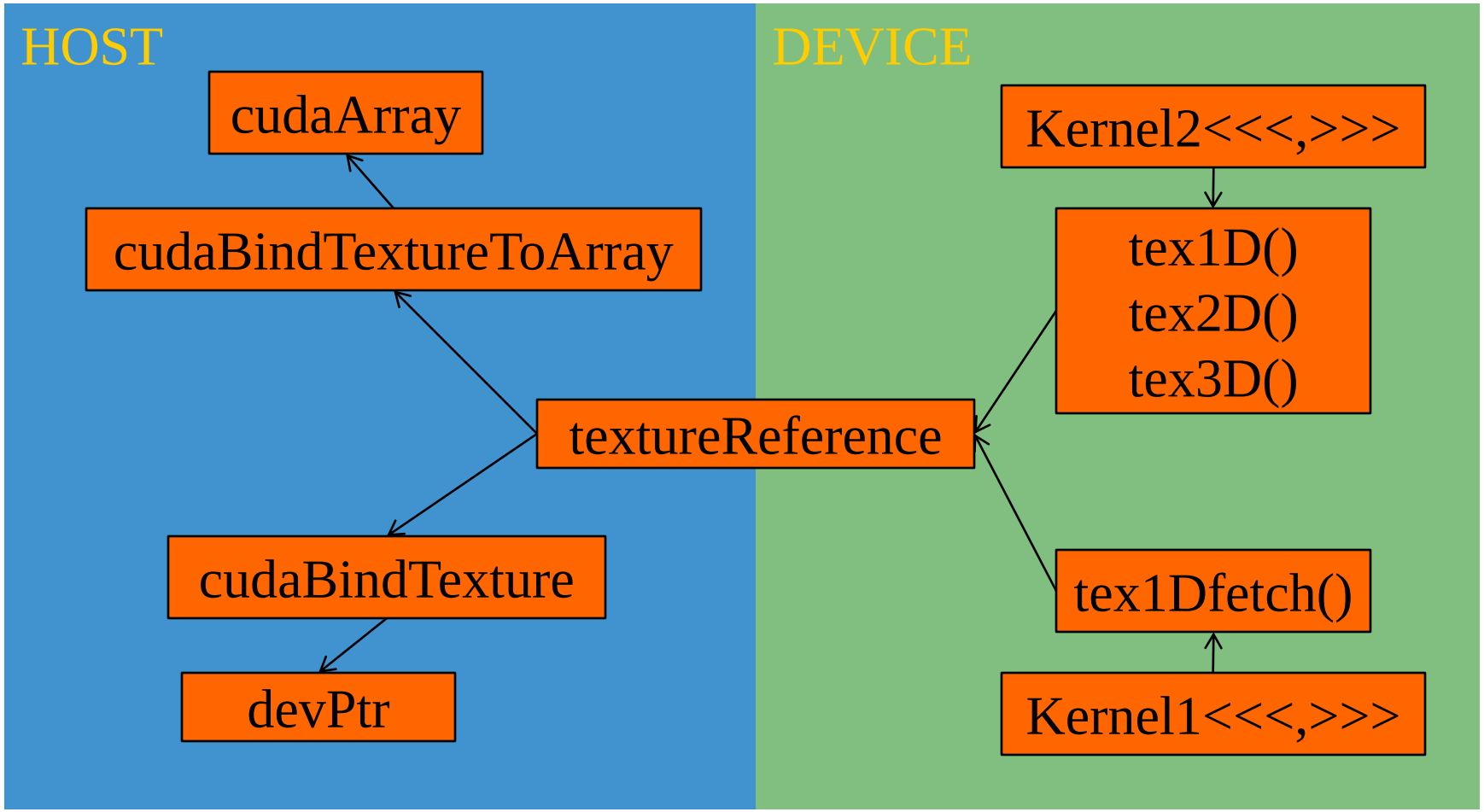
Texture в CUDA

- 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

Texture в CUDA (linear)

- Можно использовать обычную *линейную* память
- Ограничения:
  - Только для одномерных массивов
  - Нет фильтрации
  - Доступ по целочисленным координтам
  - Обращение по адресу вне допустимого диапазона возвращает ноль

Texture в CUDA



Texture в CUDA (linear)

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

__global__ void kernel1 ( float * data )
{
   int idx = blockIdx.x * blockDim.x + threadIdx.x;

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

int main(int argc, char ** argv)
{
    float *phA = NULL, *phB = NULL, *pdA = NULL, *pdB = NULL;
    // -- memory allocation

    for (int idx = 0; idx < nThreads * nBlocks; idx++)
         phA[idx] = sinf(idx * 2.0f * PI / (nThreads * nBlocks) );

    CUDA_SAFE_CALL( cudaMemcpy ( pdA, phA, nMemSizeInBytes, cudaMemcpyHostToDevice ) );

    CUDA_SAFE_CALL( cudaBindTexture(0, g_TexRef, pdA, nMemSizeInBytes) );

    dim3 threads = dim3( nThreads );
    dim3 blocks  = dim3( nBlocks );

    kernel1 <<<blocks, threads>>> ( pdB );
    CUDA_SAFE_CALL( cudaThreadSynchronize() );

    CUDA_SAFE_CALL( cudaMemcpy ( phB, pdB, nMemSizeInBytes, cudaMemcpyDeviceToHost ) );

    // -- results check & memory release
    return 0;
}
```

Texture в CUDA (cudaArray)

```
texture<float, 2, cudaReadModeElementType> g_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 *phA = NULL, *phB = NULL, *pdA = NULL, *pdB = NULL;      // linear memory pointers
    cudaArray * paA = NULL;                                        // device  cudaArray pointer
    // -- memory allocation

    cudaChannelFormatDesc cfDesc = cudaCreateChannelDesc(32, 0, 0, 0, cudaChannelFormatKindFloat);
    CUDA_SAFE_CALL( cudaMallocArray(&paA, &cfDesc, nBlocksX * nThreads, nBlocksY) );

    for (int idx = 0; idx < nThreads * nBlocksX; idx++) {
        phA[idx]                          = sinf(idx * 2.0f * PI / (nThreads * nBlocksX) );
        phA[idx + nThreads * nBlocksX] = cosf(idx * 2.0f * PI / (nThreads * nBlocksX) ); }

    CUDA_SAFE_CALL( cudaMemcpyToArray ( paA, 0, 0, phA, nMemSizeInBytes, cudaMemcpyHostToDevice ) );
    CUDA_SAFE_CALL( cudaBindTextureToArray(g_TexRef, paA) );

    dim3 threads = dim3( nThreads );
    dim3 blocks  = dim3( nBlocksX, nBlocksY );

    kernel2<<<blocks, threads>>> ( pdB );
    CUDA_SAFE_CALL( cudaThreadSynchronize() );

    CUDA_SAFE_CALL( cudaMemcpy ( phB, pdB, nMemSizeInBytes, cudaMemcpyDeviceToHost ) );

    // -- results check & memory release
    return 0;
}
```

Свертка

- В DSP свертка - это один из основных инструментов
- Определение свертки:

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

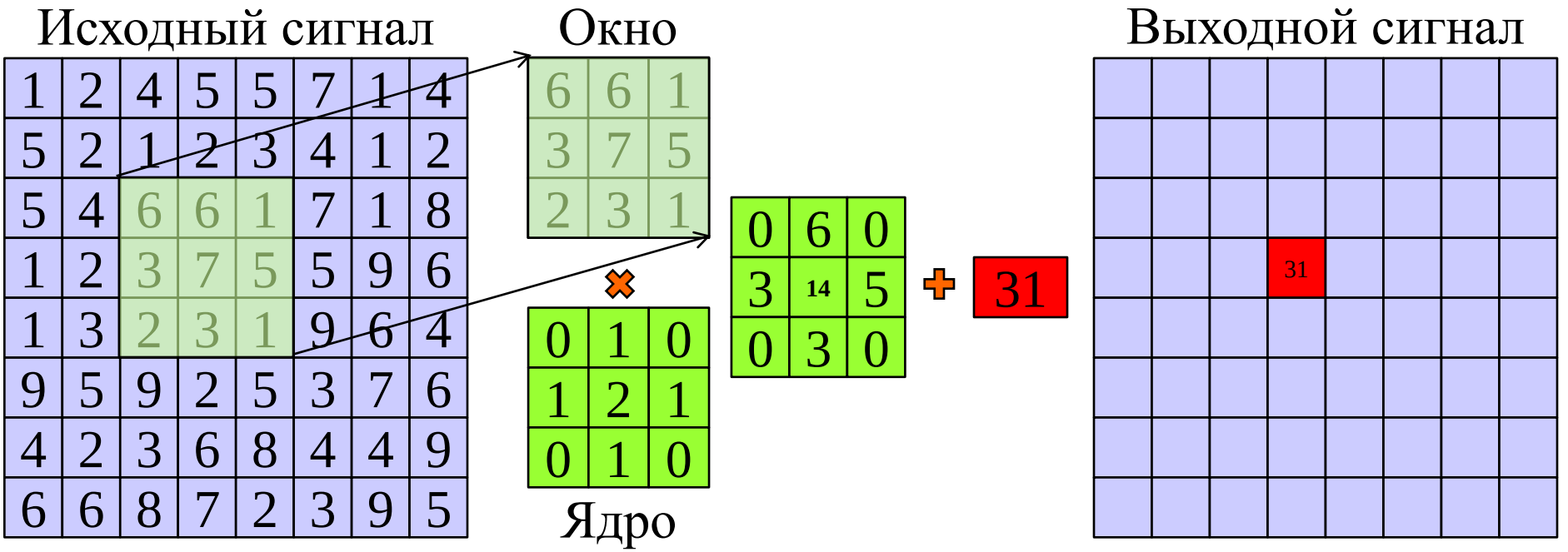
- В Дискретном случае:

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

- В 2D для изображений:

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

Свертка


Исходный сигнал
1 2 4 5 5 7 1 4
5 2 1 2 3 4 1 2
5 4 6 6 1 7 1 8
1 2 3 7 5 5 9 6
1 3 2 3 1 9 6 4
9 5 9 2 5 3 7 6
4 2 3 6 8 4 4 9
6 6 8 7 2 3 9 5
Окно
6 6 1
3 7 5
2 3 1
Ядро
0 1 0
1 2 1
0 1 0
0 6 0
3 14 5
0 3 0
31
Выходной сигнал
31

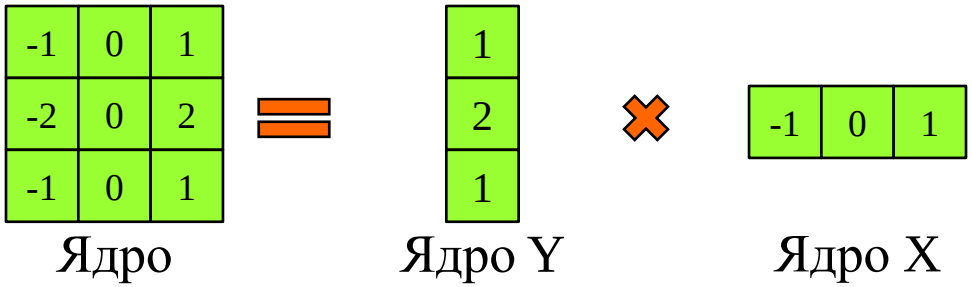
Свертка

- Вычислительная сложность:
  - W x H x N x K – умножений

Размер входного сигнала

Размер ядра

- Сепарабельные фильтры



Ядро = Ядро Y × Ядро X

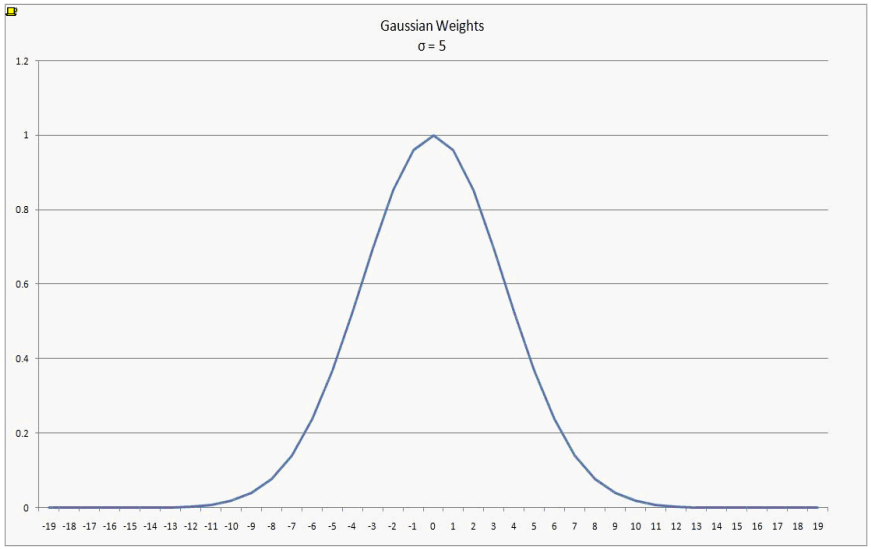
Примеры

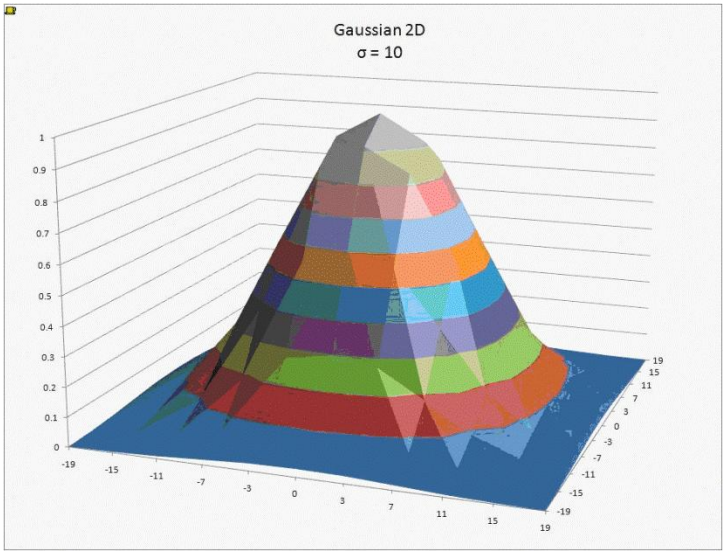
- Gaussian Blur
- Edge Detection

Gaussian Blur

- Blur (размытие) изображение
- Свертка с ядром:

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





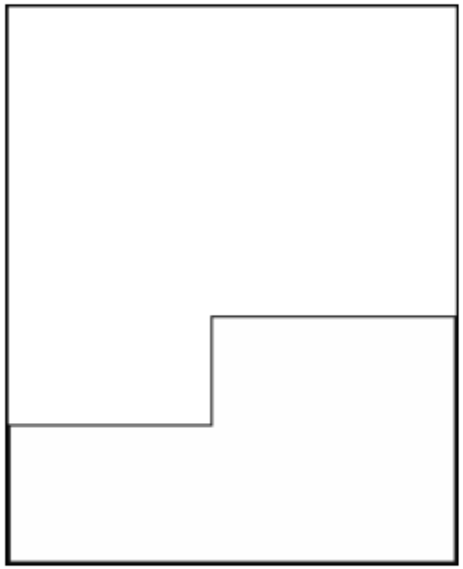
Gaussian Blur

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

__global__ void GaussBlur( 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;
    for (int ix = -r; ix <= r; ix++)
        for (int iy = -r; iy <= r; iy++)
        {
            float w = exp( -(SQR(ix) + SQR(iy)) / SQR(r) );
            rResult += w * tex2D(g_TexRef, idx + ix, idy + iy);
            wSum += w;
        }
     rResult = rResult / wSum;

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

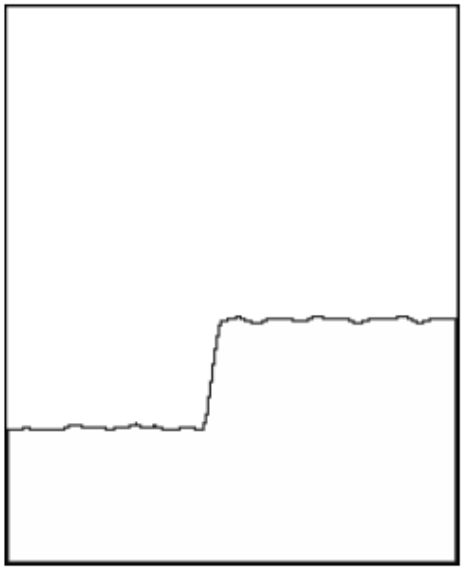
Свертка: Вопрос Вам

Edge Detection

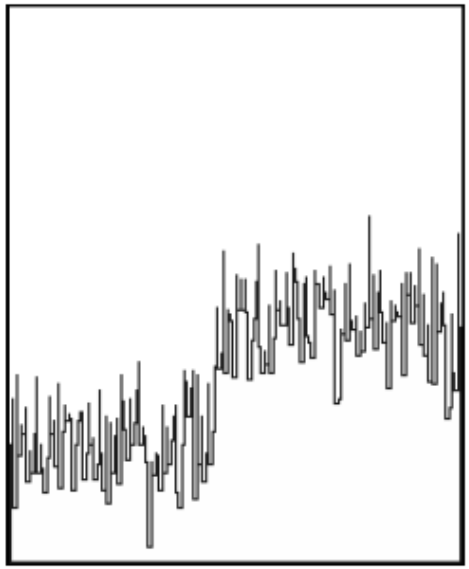
- Обнаружение границ – поиск разрывов в яркости изображения



Идеальная граница



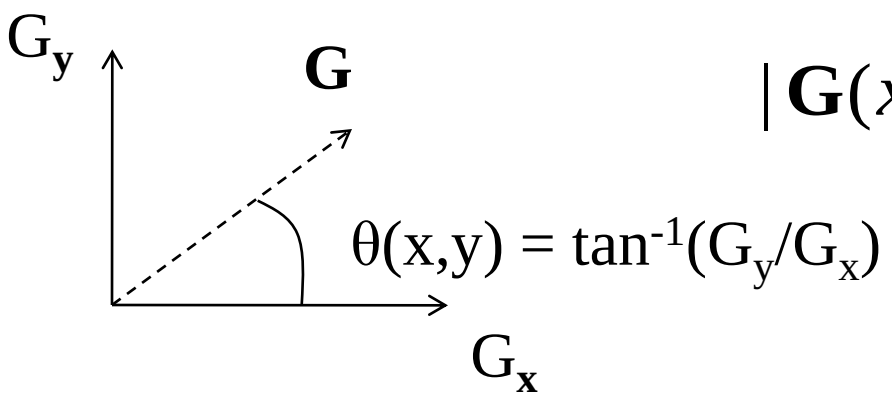
«реальная» граница



«шумная» граница

Edge Detection

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



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

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:

$$D_{1y} = [-1 \; 1] \qquad D_{1y} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

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:

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

Edge Detection

- Prewitt mask:

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

- 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:

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

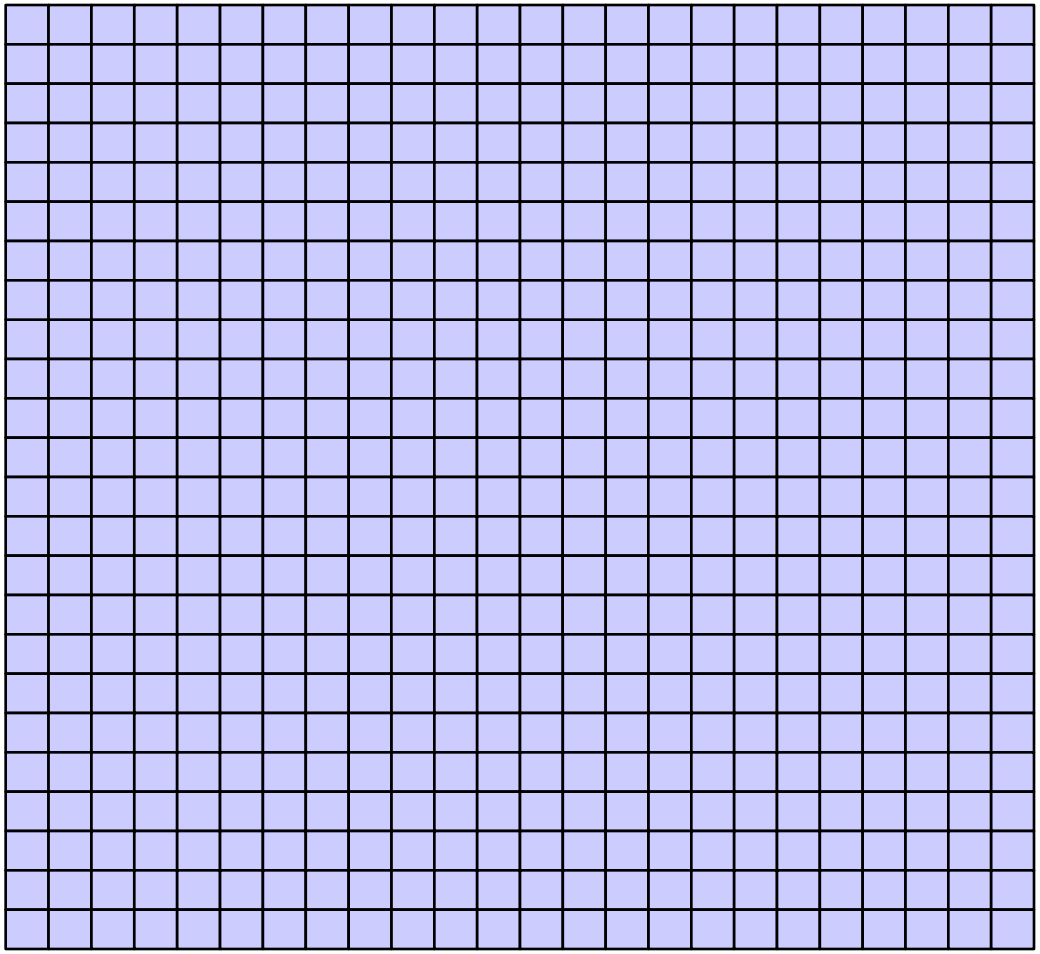
$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{f(x+\Delta x, y) - 2f(x,y) + f(x-\Delta x, y)}{\Delta x^2}$$

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

Свертка Оптимизации

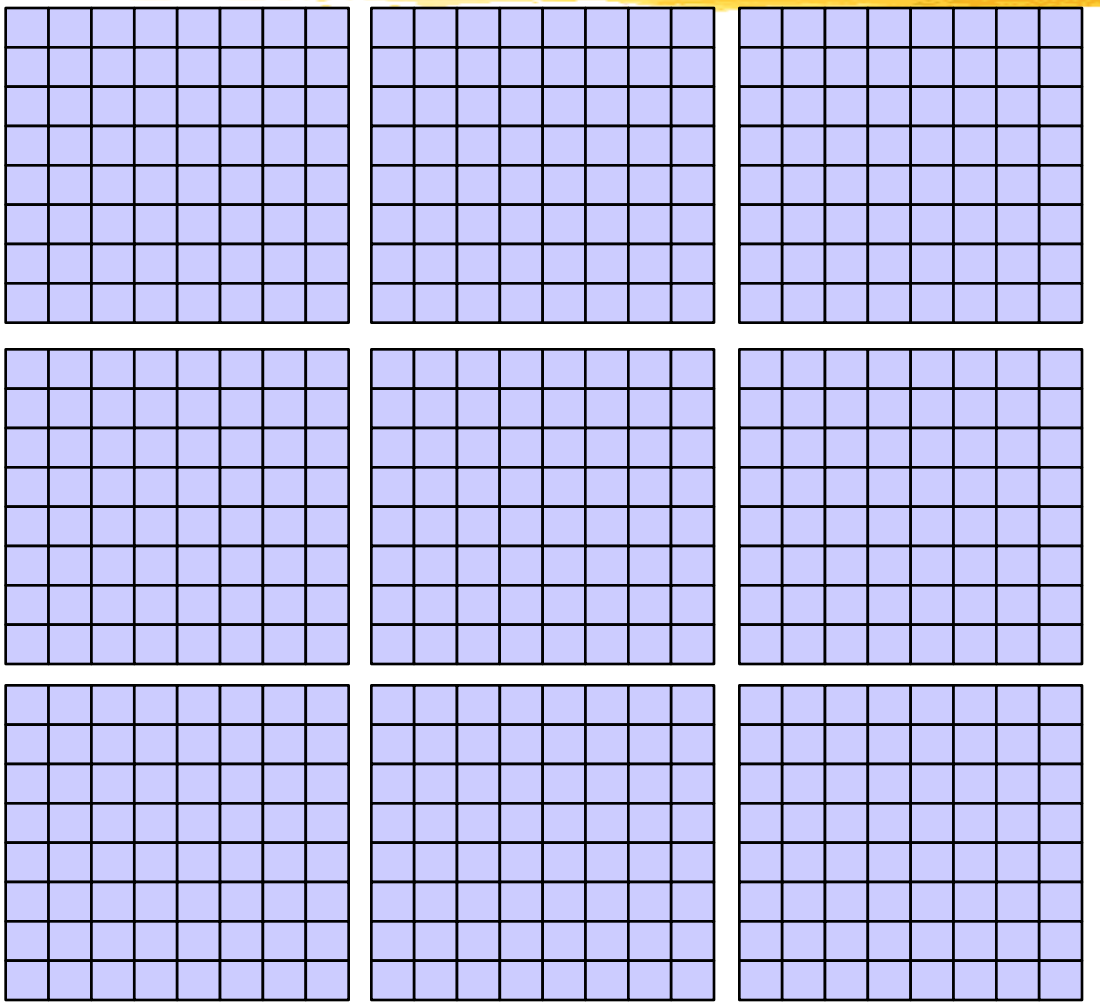
- Использовать сепарабельные фильтры
  - Существенно меньше алгоритмическая сложность
- Использовать *shared* память

Свертка Оптимизации



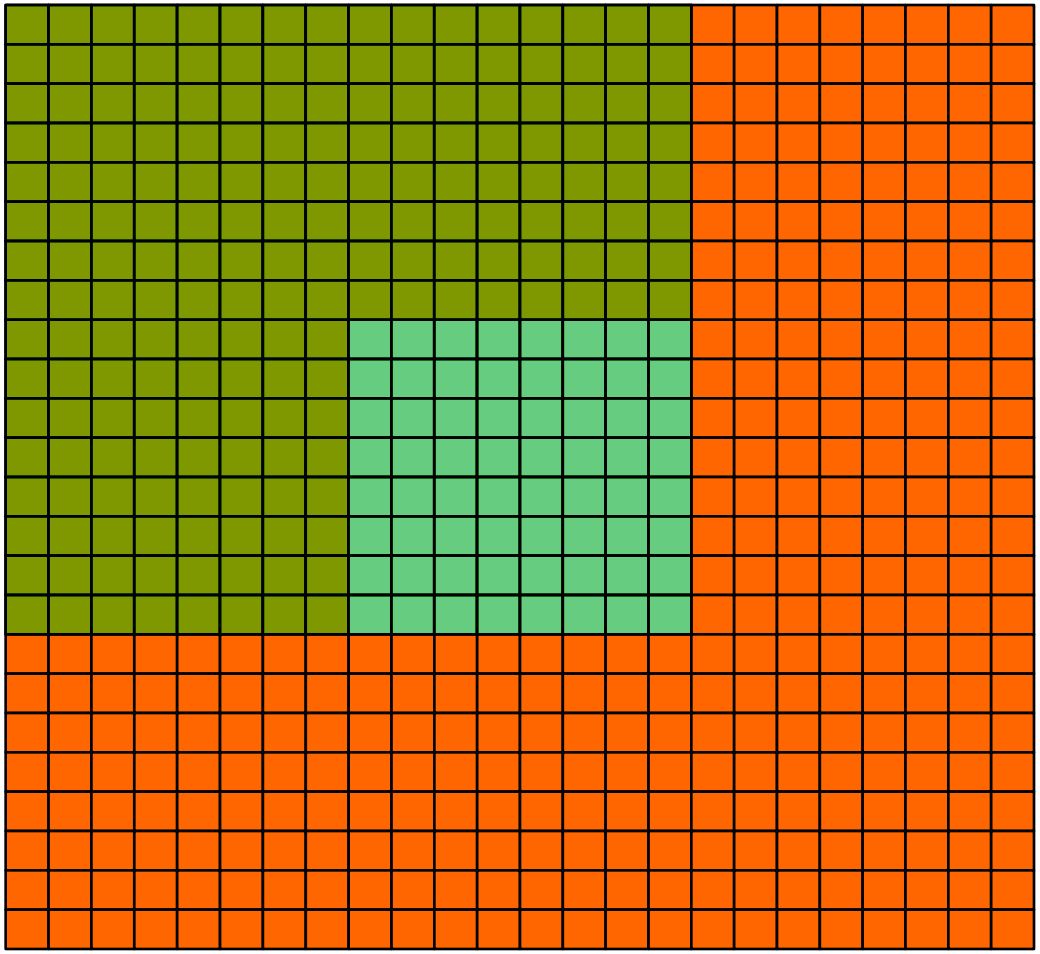
Исходное изображение

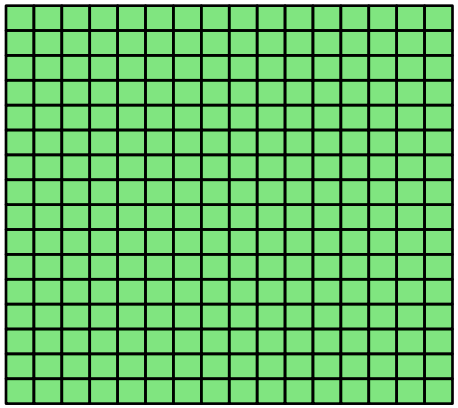
Свертка Оптимизации



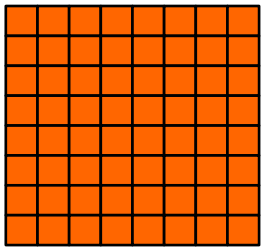
Исходное изображение

Свертка Оптимизации



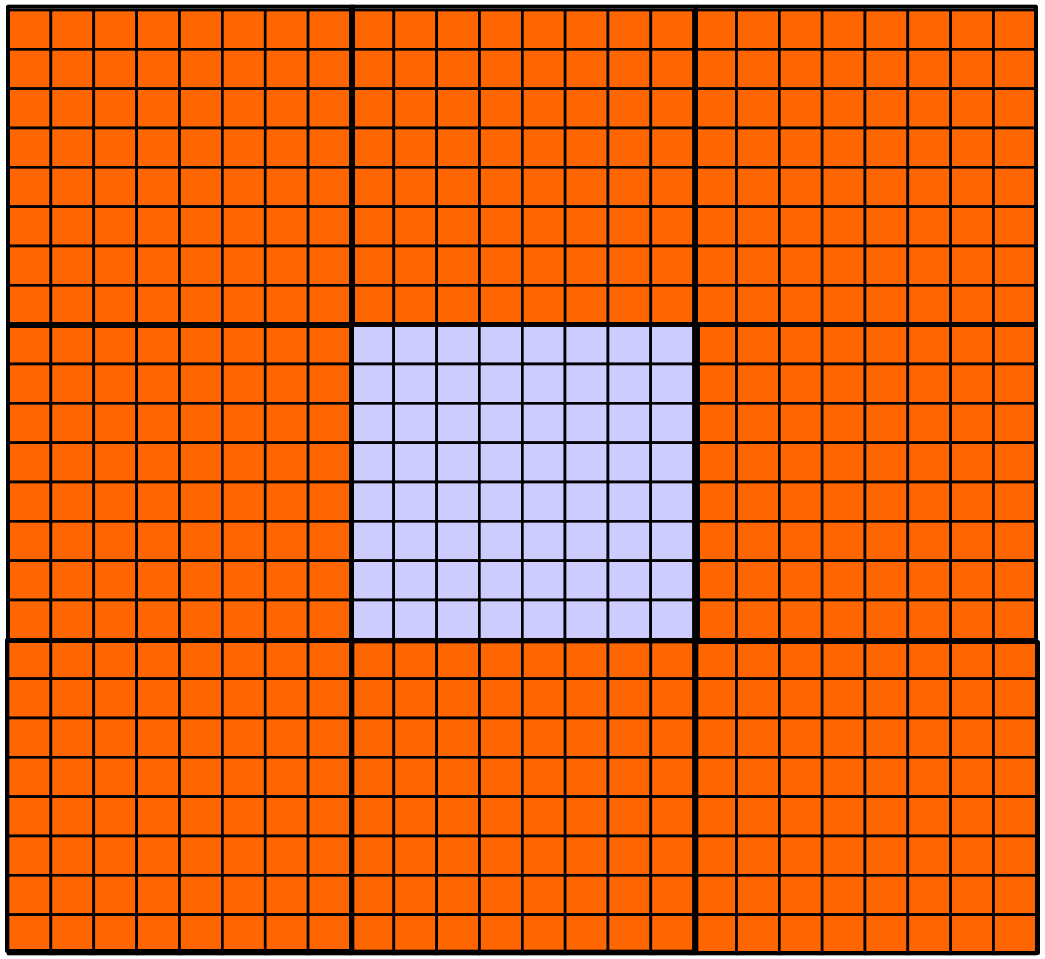


Ядро фильтра



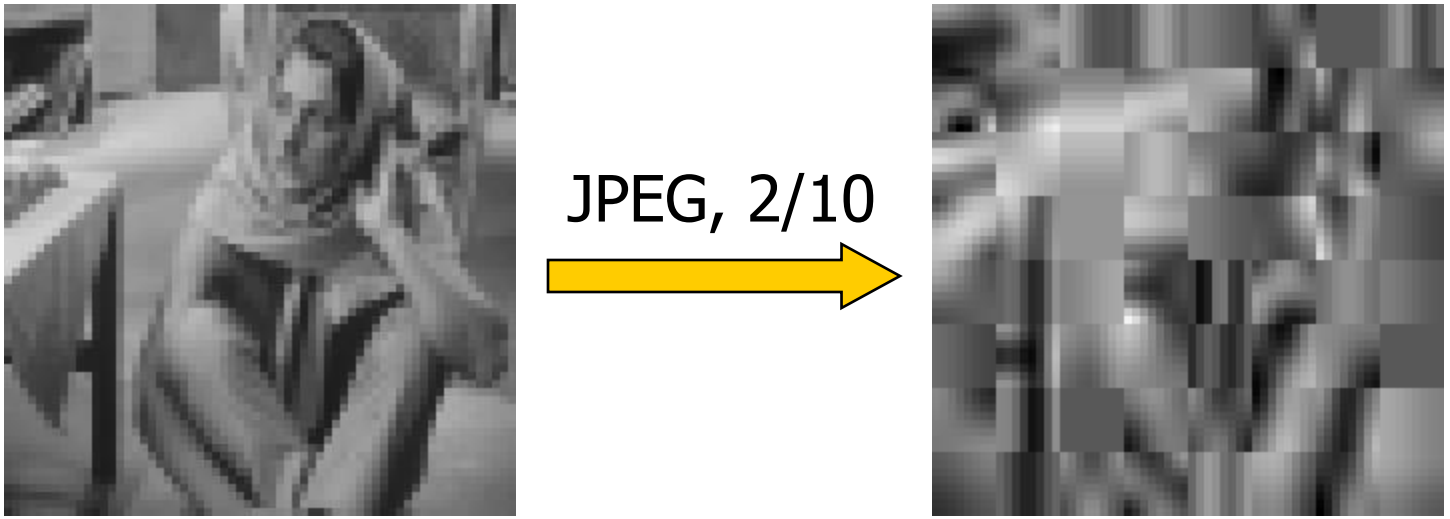
«Фартук» фильтра

Свертка Оптимизации



Discrete Cosine Transform

- Широко используется в ЦОС
- Является основой современных алгоритмов сжатия данных с потерями (JPEG, MPEG)



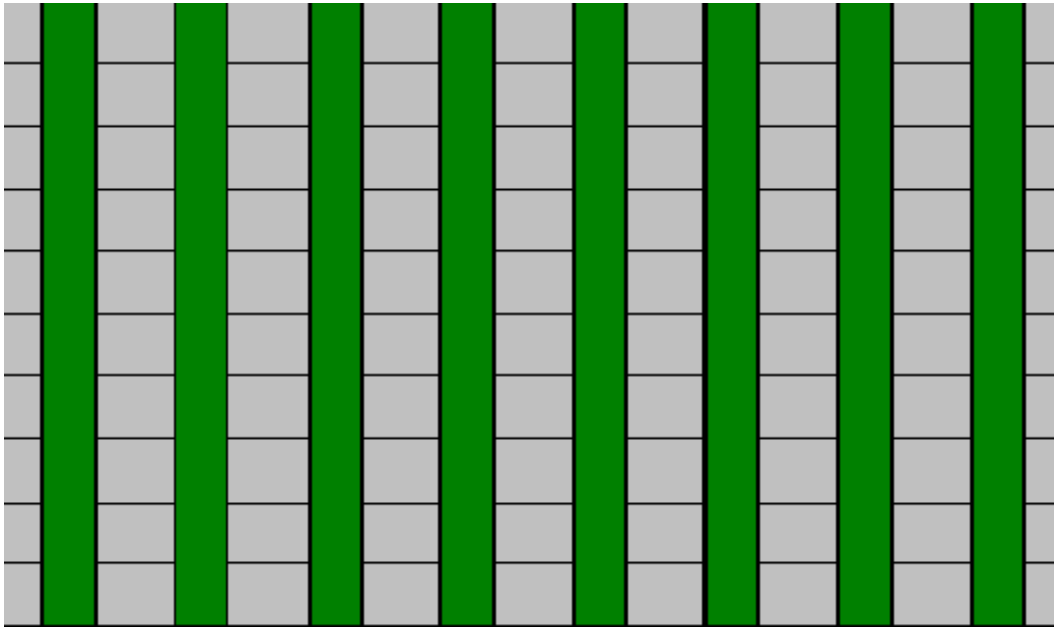
Discrete Cosine Transform

- Представитель семейства пространственно-частотных 1D преобразований, задается формулами:
- Прямое: $$C(u)=\alpha(u)\sum_{x=0}^{N-1} f(x)\cos\left[\frac{\pi(2x+1)u}{2N}\right], \qquad u=0,1,\ldots,N-1$$
- Обратное: $$f(x)=\sum_{u=0}^{N-1}\alpha(u)C(u)\cos\left[\frac{\pi(2x+1)u}{2N}\right], \qquad x=0,1,\ldots,N-1$$
- Нормировочные коэффициенты:

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

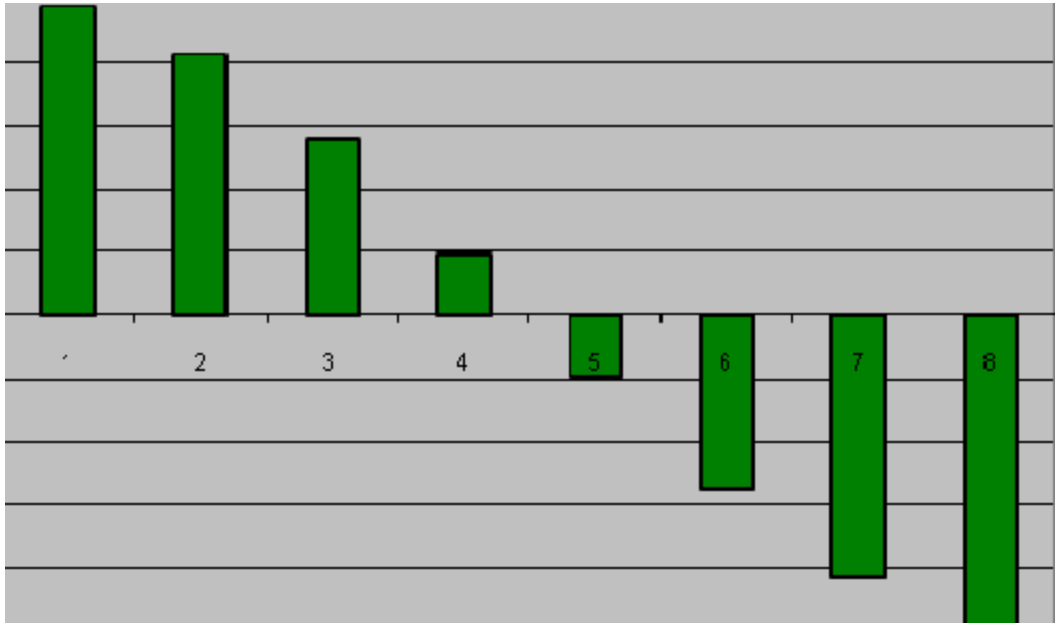
Discrete Cosine Transform

- 8-точечный случай: *u=0*



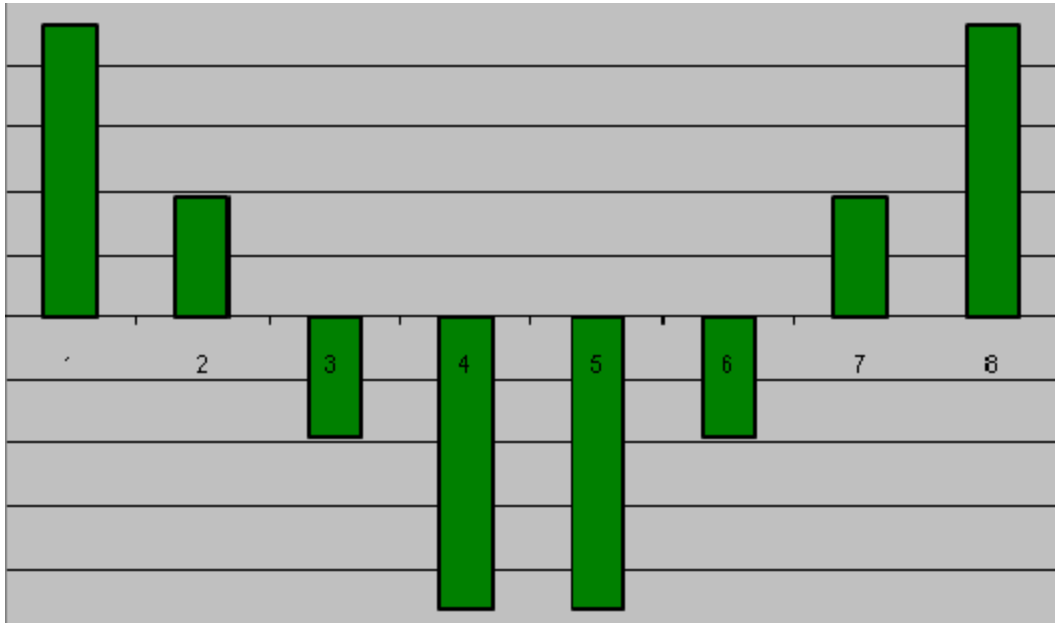
Discrete Cosine Transform

- 8-точечный случай: *u=1*



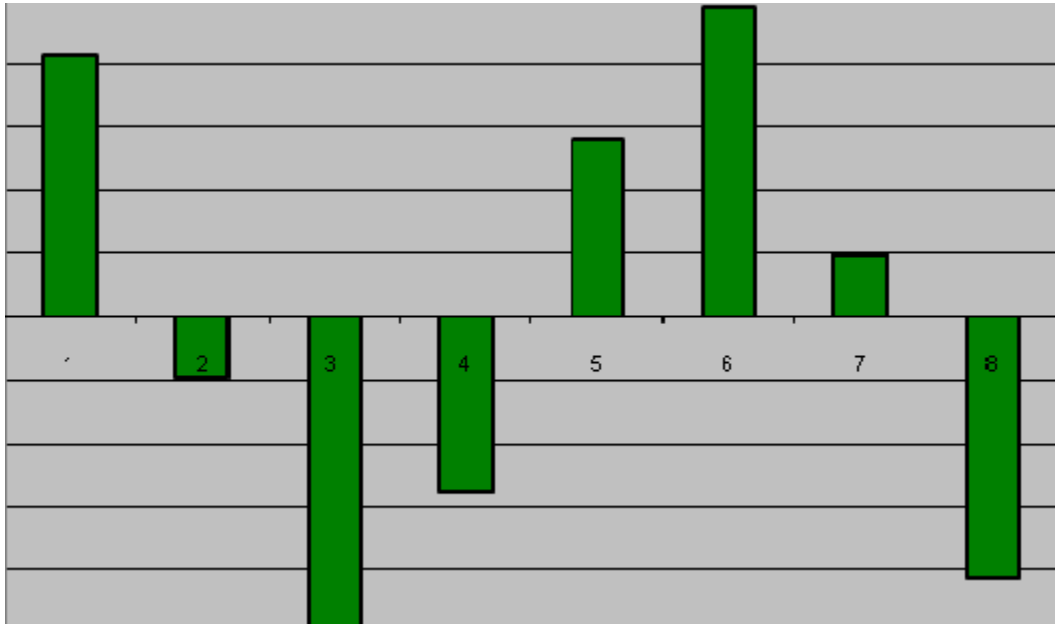
Discrete Cosine Transform

- 8-точечный случай: *u=2*



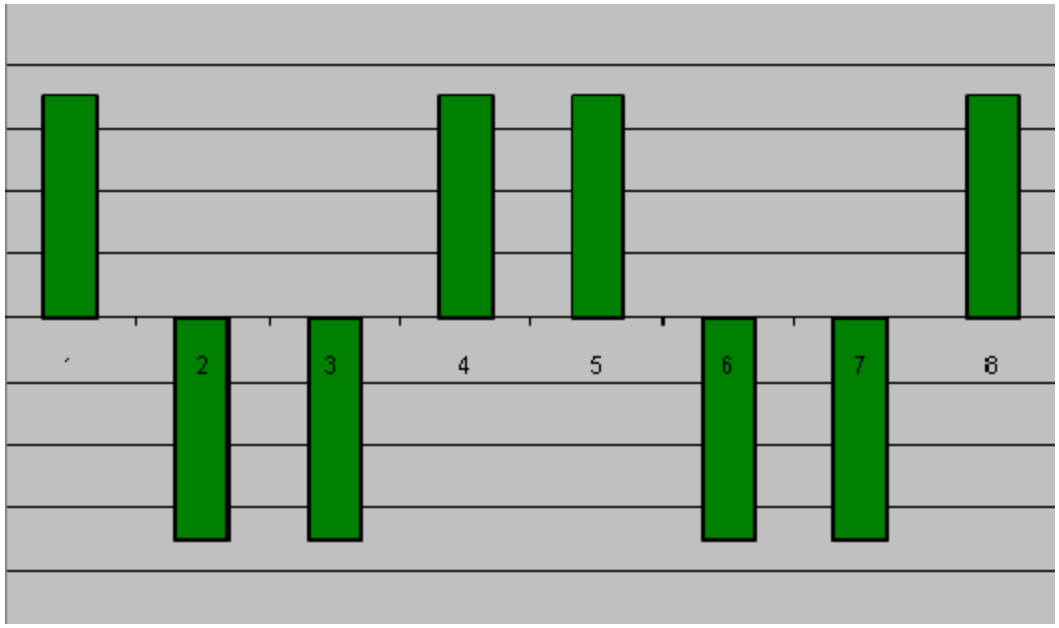
Discrete Cosine Transform

- 8-точечный случай: *u=3*



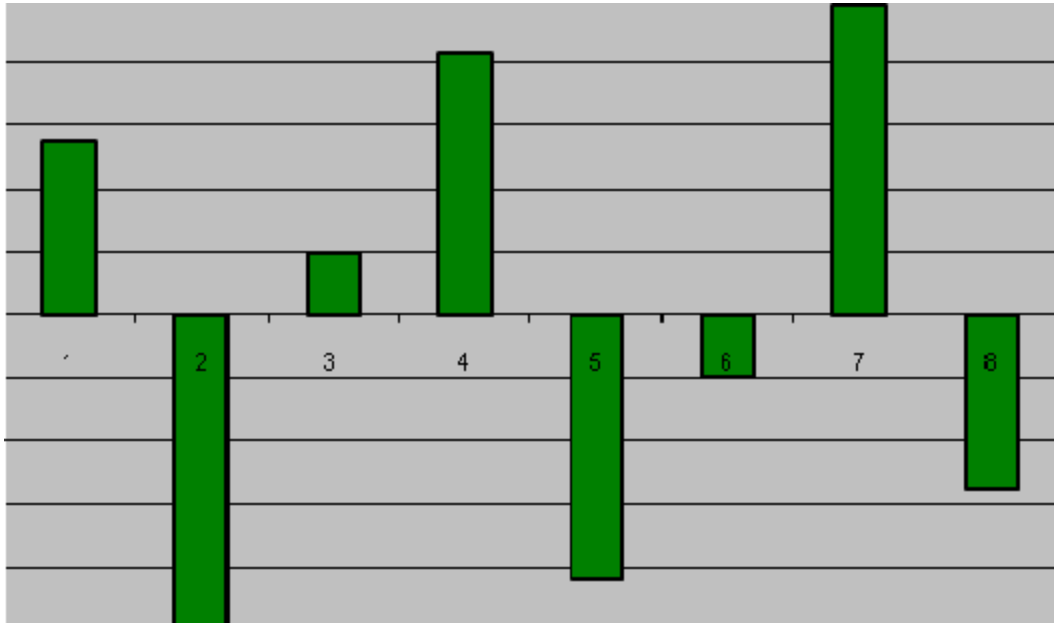
Discrete Cosine Transform

- 8-точечный случай: *u=4*



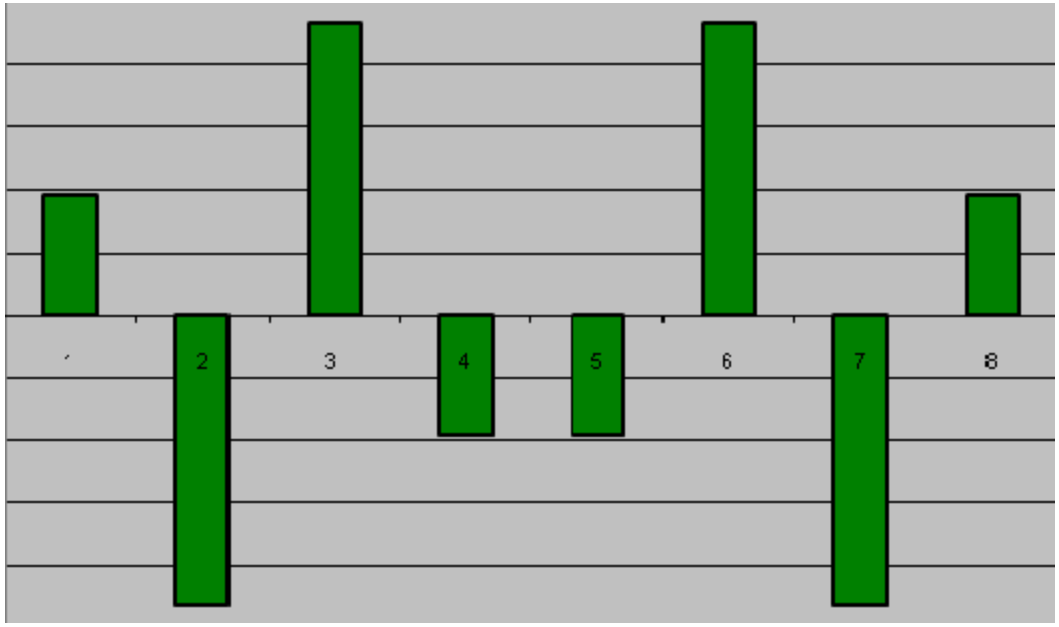
Discrete Cosine Transform

- 8-точечный случай: *u=5*



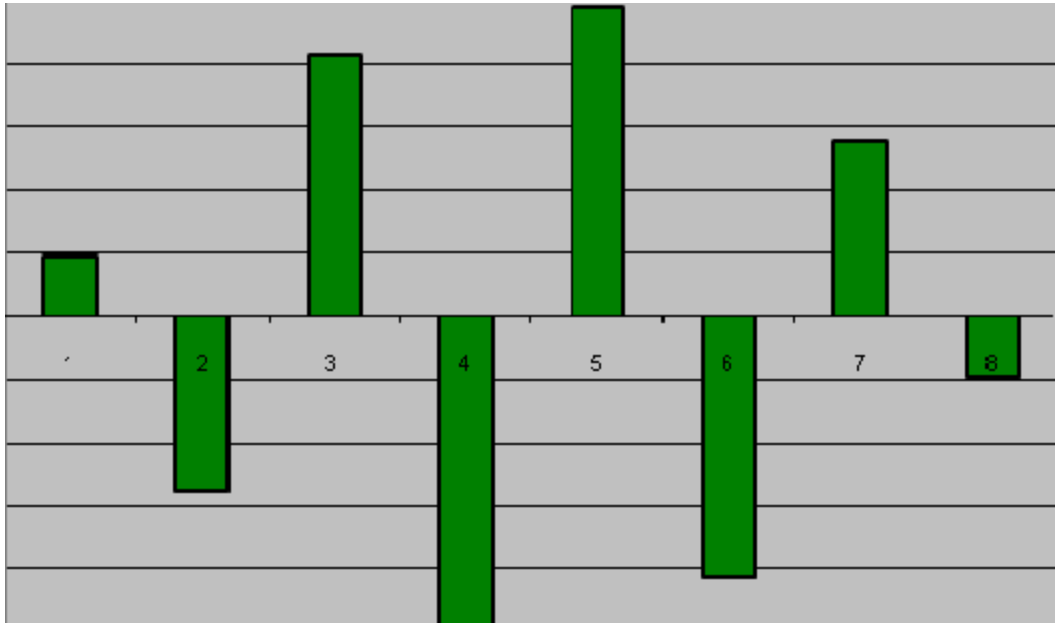
Discrete Cosine Transform

- 8-точечный случай: *u=6*



Discrete Cosine Transform

- 8-точечный случай: *u=7*

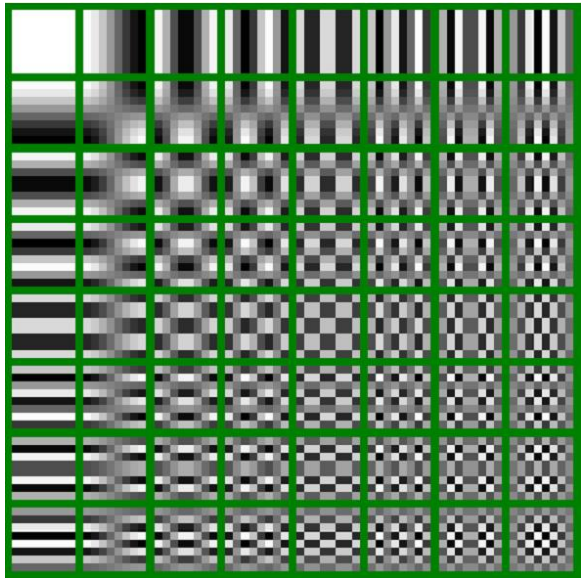


Discrete Cosine Transform

- N-мерное преобразование обладает свойством сепарабельности

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

- 2D-визуализация коэффициентов для случая 8x8 (изображение справа)
- Коэффициенты *A[8x8]* преобразования вычисляются один раз
- $C(u,v) = A^T XA$



Discrete Cosine Transform

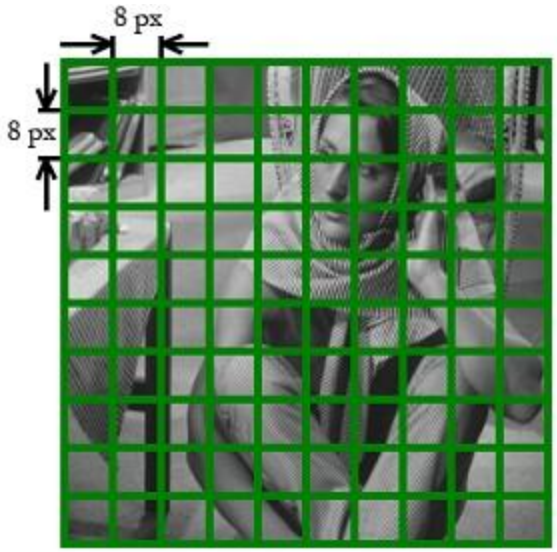
- 2 способа вычисления:
  - Наивный, по формуле с предыдущего слайда (вычисление значения коэффициента путем подстановки индексов в формулу)
  - С использованием сепарабельности
  - Оба способа представлены в примере

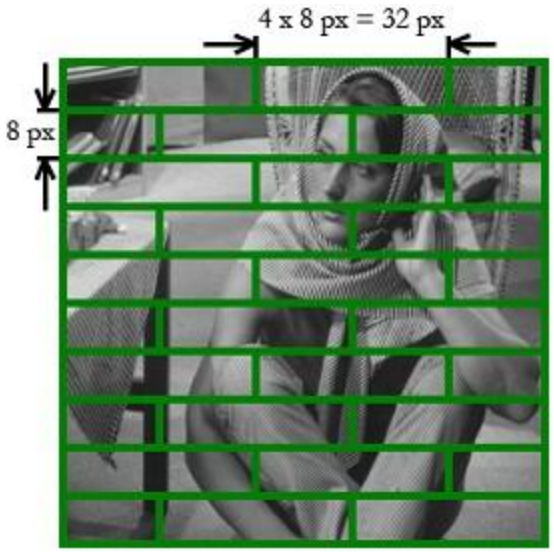
Discrete Cosine Transform

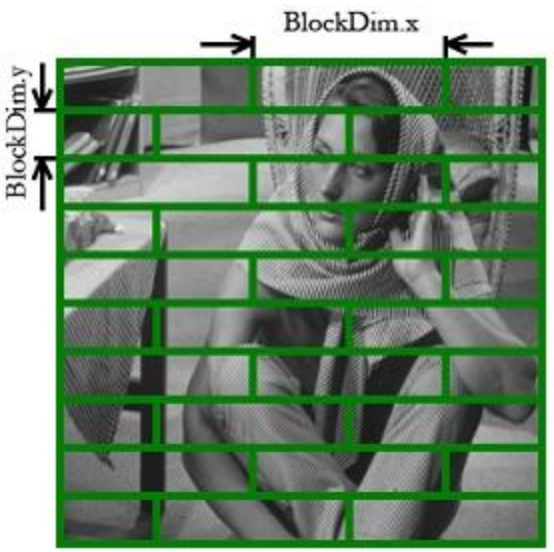
- Наивный: 64 нити на блок (8x8)
  - Загрузка одного пикселя из текстуры
  - Барьер
  - Вычисление линейной комбинации столбца threadIdx.y матрицы A (он же – строка A^T) со столбцом threadIdx.x матрицы A, запись
  - Барьер
  - Запись коэффициента в глобальную память

Discrete Cosine Transform

- Сепарабельный (блок обрабатывает несколько блоков 8x8, один тред обрабатывает вектор 8x1 или 1x8 целиком)







Discrete Cosine Transform: Вопрос Вам

Assignement #1 доступно

- Срок сдачи – следующий вторник

Ресурсы нашего курса

- CUDA.CS.MSU.SU
  - Место для вопросов и дискуссий
  - Место для материалов нашего курса
  - Место для ваших статей!
    - Если вы нашли какой-то интересный подход!
    - Или исследовали производительность разных подходов и знаете, какой из них самый быстрый!
    - Или знаете способы сделать работу с CUDA проще!
- www.steps3d.narod.ru
- www.nvidia.ru

Вопросы