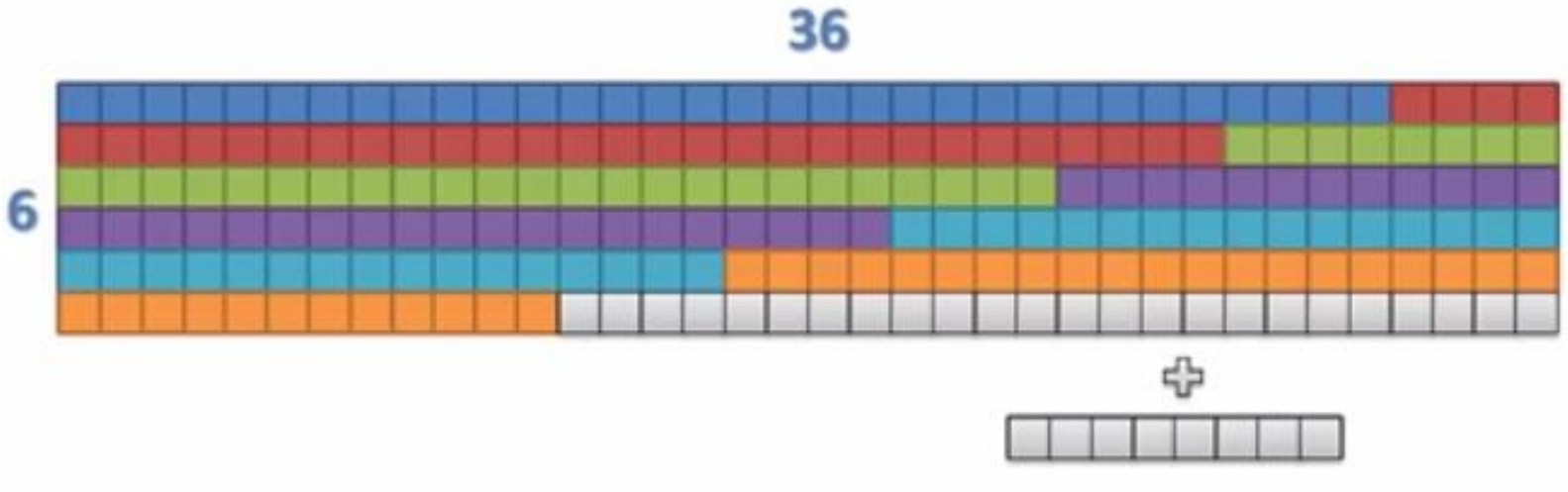
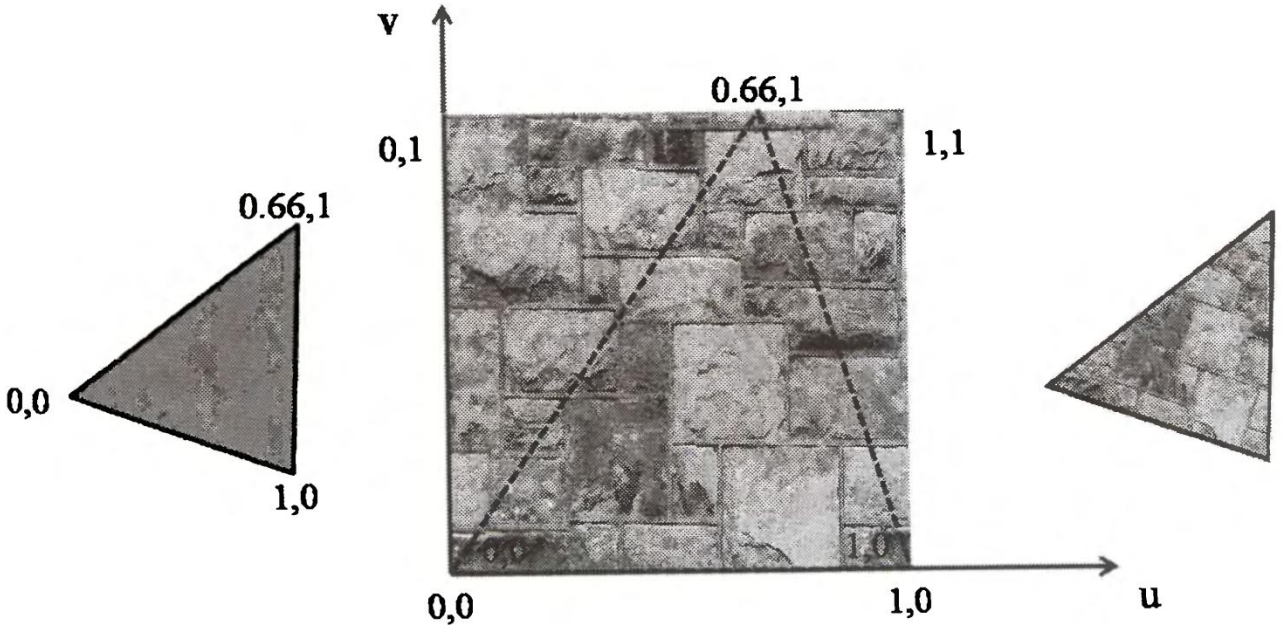
Работа с текстурной памятью

Лекция 4

Количество нитей в блоке НЕ должно быть кратно размеру warp’а



Что такое текстура



Работа с текстурной памятью

- Обладает спецификой работы графических приложений
- Возможности модуля:
  - фильтрация текстурных координат;
  - билинейная и точечная интерполяция;
  - встроенная обработка текстурных координат, в случае, когда значения выходят за допустимые границы;
  - обращение по нормализованным или целочисленным координатам;
  - возвращение нормализованных значений;
  - кеширование данных.

Специфика текстурной памяти

- Быстрая, кешируемая в 2-х измерениях, только для чтения
- Текстура может быть расположена в
  - Линейной памяти на GPU
  - В текстурном массиве (cudaArray)
- Основные функции для работы с текстурным массивом:
  - Выделение массива памяти

```
cudaError_t cudaMallocArray(struct cudaArray **arrayPtr,
   const struct cudaChannelFormatDesc *desc,size_t width, size_t height)
cudaError_t cudaFreeArray(struct cudaArray *array)
```

  - Привязка массива памяти к текстурной ссылки

```
cudaError_t cudaBindTextureToArray(const struct textureReference
           *texref, const struct cudaArray * array,
            const struct cudaChannelFormatDesc *desc)
cudaError_t cudaBindTextureToArray(const struct
       texture<T, dim, readMode> & tex, const struct cudaArray * array)
```

Текстурная ссылка

- `texture<Type, Dim, ReadMode>`
- Параметры:
  - *Type* – тип элемента (например, float3)
  - *Dim* – размерность текстуры (1, 2 или 3)
  - *ReadMode* – нужна ли нормализация прочитанных значений:
    - **cudaReadModeElementType**
    - **cudaReadModeNormalizedFloat**

Чтение из текстур их ядра

```
#include “cuda_runtime.h”

template<class T, enum cudaTextureReadMode readMode>
T tex1D ( texture<T, 1, readMode> texRef, float x );

template<class T, enum cudaTextureReadMode readMode>
T tex2D ( texture<T, 2, readMode> texRef, float x, float y );

template<class T, enum cudaTextureReadMode readMode>
T tex3D ( texture<T, 3, readMode> texRef, float x, float y, float z );
```

Пример:

```
texture<uchar4, 2, cudaReadModeElementType> texName;
uchar4 a = tex2D(texName, texcoord.x + 0.5f, texcoord.y + 0.5f);
```

Работа с текстурной ссылкой

- Доступ к текстурным ссылкам по имени шаблона:

```
const textureReference * pTexRef = NULL;
cudaGetTExtureReference(&pTexRef, ‘texName’);
```

- Связывание текстурной ссылки и линейной памяти

```
cudaError_t cudaBindTexture( size_t offset,
                const struct texture<T, dim, readMode> & tex,
                const void * dev_ptr, size_t size)

cudaError_t cudaBindTexture2D( size_t offset,
                    const struct texture<T, dim, readMode> & tex,
                    const void * dev_ptr,
                    const struct cudaChannelFormatDesc *desc,
                    size_t width, size_t height,
                    size_t pitch_in_bytes)
```

Создание текстурной ссылки и линейной памяти

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

__global__ void kernel ( float * data )
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    data [idx] = tex1D( texRef, idx );
}

float * a, * aDev;

cudaMalloc       ( (void **) &aDev, numBytes );
cudaMemcpy       ( aDev, a, numBytes, cudaMemcpyHostToDevice );
cudaBindTexture ( NULL, &texRef, aDev, &texRef.channelDesc, numBytes );
. . . .
kernel<<<blocks, threads>>> ( bDev );
. . . .
cudaThreadSynchronize ();
cudaUnbindTexture     ( &texRef );
cudaFree              ( aDev );
```

cudaArray

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

cudaArray * array;
cudaChannelFormatDesc chDesc = cudaCreateChannelDesc ( 32, 0, 0, 0,
                                                       cudaChannelFormatKindFloat );

cudaMallocArray   ( &array, &chDesc, width, height );
cudaMemcpyToArray ( array, wOffs, hOffs, src, numBytes,
                    cudaMemcpyHostToDevice );
cudaBindTextureToArray ( &texRef, array, &chDesc );

.  .  .  .  .  .  .  .  .

cudaUnbindTexture ( &texRef );
cudaFreeArray     ( array   );
```

Цифровая обработка сигналов. Фильтры преобразования цвета

Негатив – фильтр, который вычисляет для заданного цвета дополнение его до белого:

$$R=1-I.$$

Фильтр яркости отбрасывает цветовую информацию, оставляя информацию о яркости пикселя:

$$lum=\{0.3,0.59,0.11\}$$
$$R=dot(lum,I).$$

Гамма-коррекция – это коррекция функции яркости в зависимости от характеристик устройства вывода:

$$R=I^{\gamma}.$$

Цифровая обработка сигналов. Фильтры преобразования цвета

```
float4 f4(float x) {float4 r = { x, x, x, x}; return r;}
float3 f3(float x, float y, float z) {float3 r = { x, y, z}; return r;}
float   dot3(float4 v, float3 u)
{float r =  v.x*u.x + v.y*u.y + v.z*u.z; return r;}
texture<uchar4, 2, cudaReadModeNormalizedFloat> g_Texture;
__global__ void Simple_kernel(uchar4 * pDst, float g, uint32 w, uint32 h, uint32 p
{
  int tidx = threadIdx.x + blockIdx.x * blockDim.x;
  int tidy = threadIdx.y + blockIdx.y * blockDim.y;
   // проверка, что текущие индексы не выходят за границы изображения
  if ( tidx < w && tidy < h){
  float4 c = text2D(g_Texture, tidx + 0.5f, tidy + 0.5f );
// преобразование для негатива
   float4 r = 1.0f - c;

 //преобразование цвета для гамма коррекции
  //float4 r = {pow(c.x,g), pow(c.y,g), pow(c.z,g), pow(c.w,g) };
  //преобразование для яркости
  // float l = dot3(c, f3(0.30f, 0.59f, 0.11f));
  //float4 r = f4(l);
  pDst[tidx + tidy * w] = uc4(r*255.0f, 0.0f, 255.0f);
  }
}
```

Цифровая обработка сигналов. Фильтры преобразования цвета

```
void Simple(cuImage &dst, cuImage & src, float g)
{
   cudaError_t err;
   uint3 dim = src.m_data.dim();
   uint3 whd = src.m_data.whd();
   //присоединить src cudaArray к текстуре
    err = cudaBindTextureToArray(g_Texture, src.m_data.aPtr());
    if (err != cudaSuccess)
      std::cerr<<“Error binding texture to array in Simple”<<std::endl;
   //размеры блока выбираются такими, чтобы покрывать изобр. целиком
   dim3 block(32,8);
   dim3 grid(dim.x/block.x + ((dim.x % block.x) ? 1 :0),
                dim.y/block.y + ((dim.y % block.y) ? 1 :0));
   Simple_kernel<<<grid, block>>>(dst.m_data.dPtr(), g, dim.x, dim.y,
whd.x);
   err = cudaThreadSynchronize();
   if (err != cudaSuccess)
     std::cerr << “Error during Simple_kernel execution” << std::endl;
}
```

Цифровая обработка сигналов. Свертка

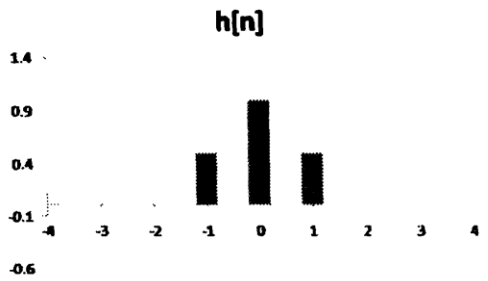
Даны две вещественные функции f(x) и g(x), интегрируемые на R, то свертка представляет собой функцию вида:

$$(f * g)(t) = \int_R f(\tau)g(t-\tau)d\tau.$$

В дискретном виде: $$y[n] = \sum_{k=-\infty}^{\infty} x[k]g[n-k].$$

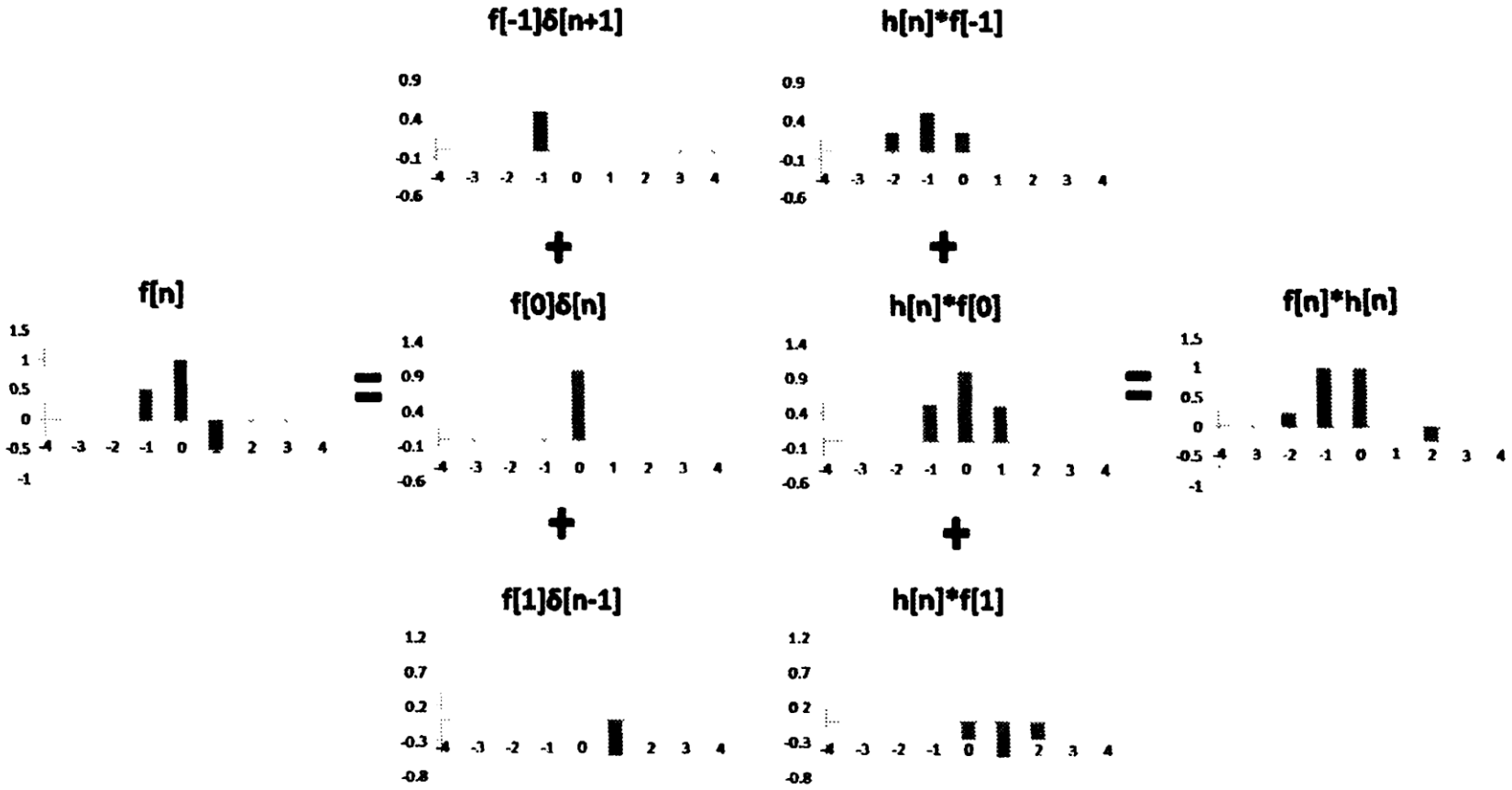
Цифровая дельта-функция: $$\delta[n] = \begin{cases} 0, n \neq 0 \\ 1, n = 0 \end{cases}.$$

Ядро свертки (импульсная характеристика фильтра) h(n):



Цифровая обработка сигналов. Свертка

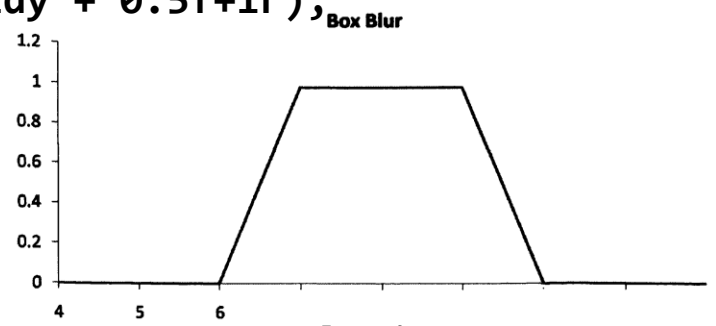
Свертка данной функции с ядром h – это линейная комбинация откликов системы на входные значения f[i]:



Цифровая обработка сигналов. Фильтр BoxBlur

Фильтрация, при которой сигнал усредняется в некоторой окрестности радиуса R с равными весами, называется box blur (размытие коробкой).

```
texture<uchar4, 2, cudaReadModeNormalizedFloat> g_BoxBlur;
__global__ void BoxBlur_kernel(uchar4* pDst, float radius, uint32 w, uint32 h)
{
  int tidx = threadIdx.x + blockIdx.x * blockDim.x;
  int tidy = threadIdx.y + blockIdx.y * blockDim.y;
  // проверка, что текущие индексы не выходят за границы изображения
  if ( tidx < w && tidy < h)
  {
    float4 r = {0.0f, 0.0f, 0.0f, 0.0f};
    for(int ir = -radius; ir <= radius; ir++ )
       for(int ic = -radius; ic <= radius; ic++ )
       {
         r += tex2D(g_BoxBlur, tidx + 0.5f+ic, tidy + 0.5f+ir);
       }
       //нормализация полученных результатов
       r /= ((2*radius+1)*(2*radius+1));
       pDst[tidx+tidy*w] = uc4(r*255.0f);
    }
}
```



Цифровая обработка сигналов. Фильтр Gaussian Blur

Фильтрация, при которой сигнал усредняется в некоторой окрестности радиуса R с весами равными:

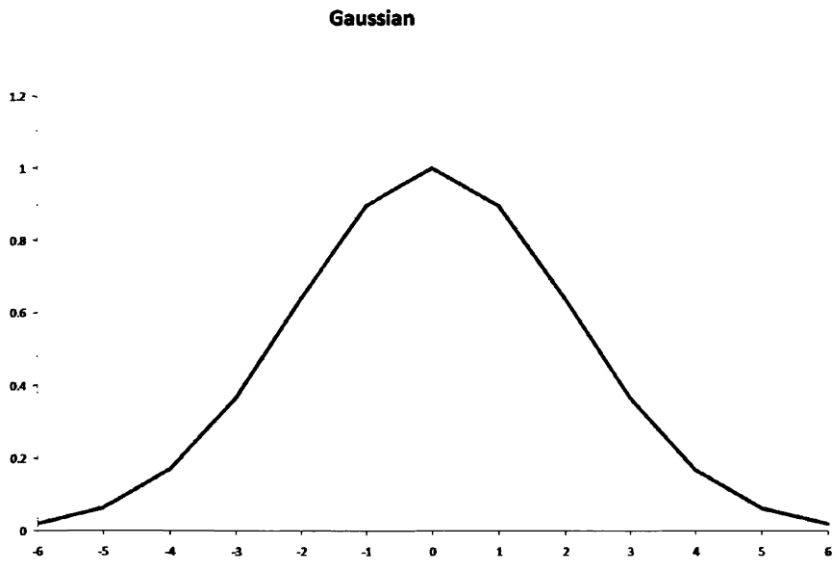
- для одномерного случая:

$$W(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

- для двумерного случая:

$$W(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

называется Гауссово размытие.



Цифровая обработка сигналов. Фильтр Gaussian Blur

```
texture<uchar4, 2, cudaReadModeNormalizedFloat> g_Gaussian;
__global__ void GaussianK (uchar4* pDst, float radius, float sigma_sq, uint32 w,
uint32 h)
{
     int tidx = threadIdx.x + blockIdx.x * blockDim.x;
     int tidy = threadIdx.y + blockIdx.y * blockDim.y;
     // проверка, что текущие индексы не выходят за границы изображения
     if ( tidx < w && tidy < h)
     {
          float4 r = {0.0f, 0.0f, 0.0f, 0.0f};
          float  weight_sum = 0.0f;
          float  weight = 0.0f;
          for(int ic = -radius; ic <= radius; ic++ )
          {
               weight = exp(-(ic*ic)/ sigma_sq);
               r += tex2D(g_ Gaussian, tidx + 0.5f+ic, tidy + 0.5f)* weight;
               weight_sum += weight;
           }
           //нормализация полученных результатов
           r /= weight_sum;
           pDst[tidx+tidy*w] = uc4(r*255.0f);
}
```

Цифровая обработка сигналов. Масштабирование изображений

Теорема Котельникова (теорема Найквиста-Шеннона): если аналоговый сигнал x(t) имеет ограниченный спектр, то он может быть восстановлен однозначно и без потерь по своим дискретным отсчетам, взятым с частотой, более удвоенной максимальной частоты спектра F_{max}:

$$f_D > 2F_{max},$$

тогда можно будет точно восстановить сигнал, используя sinc-интерполяцию:

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

Цифровая обработка сигналов. Масштабирование изображений

Артефакты, которые возникают при увеличении изображения:

а) **размытие** – новое изображение теряет резкость;

б) **эффект Гиббса** – артефакт, который проявляет себя в виде ореолов вокруг тонких границ;

в) **алиасинг** – проявляет себя в виде ступенчатых краев при увеличении и цветовом муаре при уменьшении изображения.

Рассмотрим два линейных интерполирующих фильтра:

1) билинейный;
2) ланкзос.

Цифровая обработка сигналов.

Билинейный фильтр

Даны два значения f(k) и f(k+1), то промежуточное значение можно приблизить с помощью линейной функции вида:

$$f(k+\alpha) = (1-\alpha)f(k) + \alpha f(k+1).$$

В двумерном случае: $f(m+\alpha, n+\beta) = (1-\beta)((1-\alpha)f(m,n) + \beta f(m+1,n)) + \beta((1-\alpha)f(m,n+1) + \alpha f(m+1,n+1))$.

```
texture<uchar4, 2, cudaReadModeNormalizedFloat> g_Bilinear;
__global__ void Bilinear (uchar4* pDst, float factor, uint32 w, uint32 h)
{
    int tidx = threadIdx.x + blockIdx.x * blockDim.x;
    int tidy = threadIdx.y + blockIdx.y * blockDim.y;
    // проверка, что текущие индексы не выходят за границы изображения
    if ( tidx < w && tidy < h){
        float center = tidx/factor;
        int32 start = (int32) center;
        int32 stop = start + 1.0f;
        float t = center – start;
        float4 a = tex2D(g_Bilinear, tidy + 0.5f, start + 0.5f);
        float4 b = tex2D(g_Bilinear, tidy + 0.5f, stop + 0.5f);
         float4 linear = lerp(a, b, t);
         pDst[tidx+tidy*w] = uc4(linear *255.0f);
    }
}
```

Цифровая обработка сигналов.

Ланкзос интерполяция

Фильтр Ланкзоса – это «оконная» версия sinc-интерполяции.

Импульсная характеристика – это нормализованная sinc(x)-функция, взвешенная с окном Ланкзоса.

Окно Ланкзоса – это центральная область sinc(x/a) для интервала $x \in [-a,a]$.

Ядро фильтра:

$$L(x) = \begin{cases} \operatorname{sinc}(x)\operatorname{sinc}(x/a), & -a < x < a, x \neq 0 \\ 1 & x = 0 \\ 0 & \textit{иначе} \end{cases}.$$

```
texture<uchar4, 2, cudaReadModeNormalizedFloat> g_Lanczos;
__divice__ float lanczos(float x, float r)
{
      const float m_pi = 3.14159265f;
      float result = 0.0f;
      if ( x >= -r && x <= r ){
           float a = x*m_pi;
           float b = (r*(sin(a/r)) * (sin(a))/(a*a));
           result = (x == 0.0f) ? 1.0f :b;
     }
     return result;
}
```

Цифровая обработка сигналов.

Ланкзос интерполяция

```
__global__ void Lanczos_kernel(uchar4* pDst, float factor, float blur, float radius, float
support, float scale, uint32 w, uint32 h)
{
     int tidx = threadIdx.x + blockIdx.x * blockDim.x;
     int tidy = threadIdx.y + blockIdx.y * blockDim.y;
     // проверка, что текущие индексы не выходят за границы изображения
     if ( tidx < w && tidy < h)
     {
          float4 r = {0.0f, 0.0f, 0.0f, 0.0f};
          float  weight_sum = 0.0f, weight = 0.0f;
          float center = tidx/factor;
          int32 start = (int32) max (center-support + 0.5f, (float)0);
          int32 stop = (int32) min (center+support + 0.5f, (float)w);
          float nmax = stop - start;
          float s = start - center;
          for(int n = 0; n < nmax; ++n, ++s ){
               weight = lanczos(s*scale, radius);
               weight_sum += weight;
               r += (tex2D(g_Lanczos, tidy + 0.5f, start +n+0.5f)* weight);
           }
           if (weight_sum != 0.0f)
           {//нормализация полученных результатов
            r /= weight_sum;
           }
           pDst[tidx+tidy*w] = uc4(r *255.0f, 0, 255.0f);
     }
}
```