Лекция 7

- Константная память.
- Текстурная память.

lab6.cu

```
#define CM_SIZE 4

__constant__ int c_a[CM_SIZE];
.....
void hInitCM(int M){
  int* tmp_a=(int*)calloc(M,sizeof(int));

  for(int i=0; i<M;i++)
    tmp_a[i]=-2*i;

  CUDA_CHECK_RETURN(cudaMemcpyToSymbol(c_a, tmp_a,
                                       M*sizeof(int), 0, cudaMemcpyHostToDevice));

  free(tmp_a);
}
```

```
__global__ void gTestCM(int* a, int* b){
 int i=threadIdx.x+blockIdx.x*blockDim.x;

  b[i]=a[i]*c_a[i%CM_SIZE];
}

__global__ void gTestGM(int* a, int* b, int* c){
 int i=threadIdx.x+blockIdx.x*blockDim.x;

 b[i]=a[i]*c[i%CM_SIZE];
}
```

lab6.cu

nvprof ./lab6 1024 128

……

5.61% 2.3680us 1 **2.3680us** 2.3680us 2.3680us **gTestCM**(int*, int*)

5.38% 2.2720us 1 **2.2720us** 2.2720us 2.2720us **gTestGM**(int*, int*, int*)

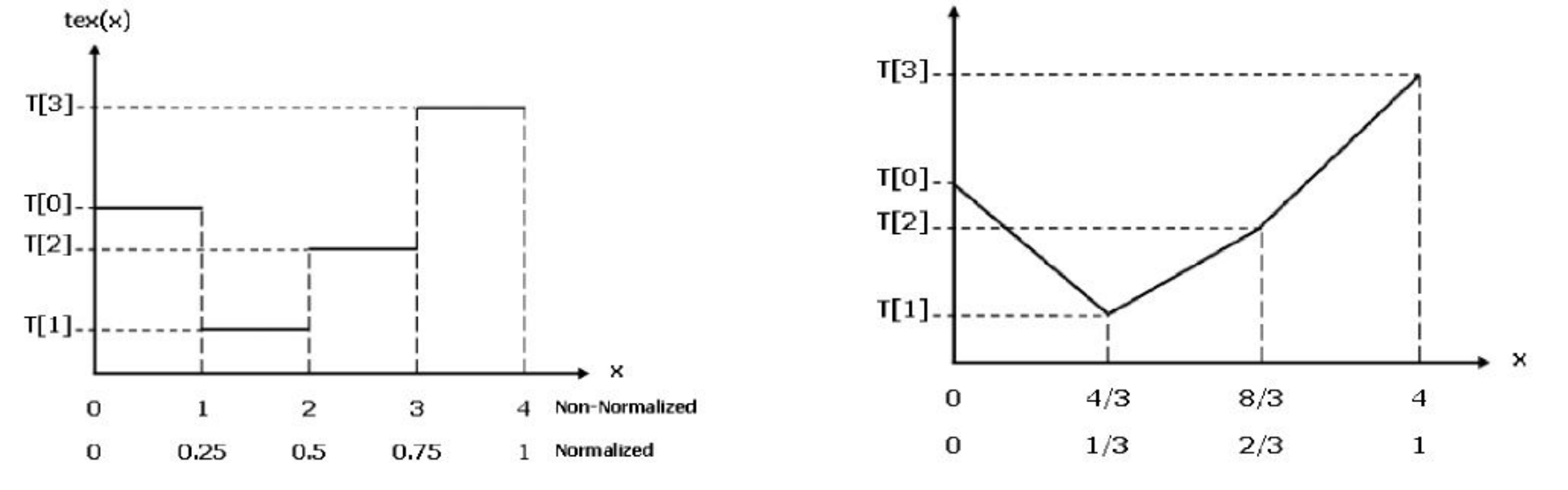
lab61.cu

```
#include <stdio.h>
#define CM_SIZE 4

extern __constant__ int c_a[CM_SIZE];

__global__ void gTestCM1(){
 int i=threadIdx.x+blockIdx.x*blockDim.x;

 printf("%d\n", c_a[i]);
}
```

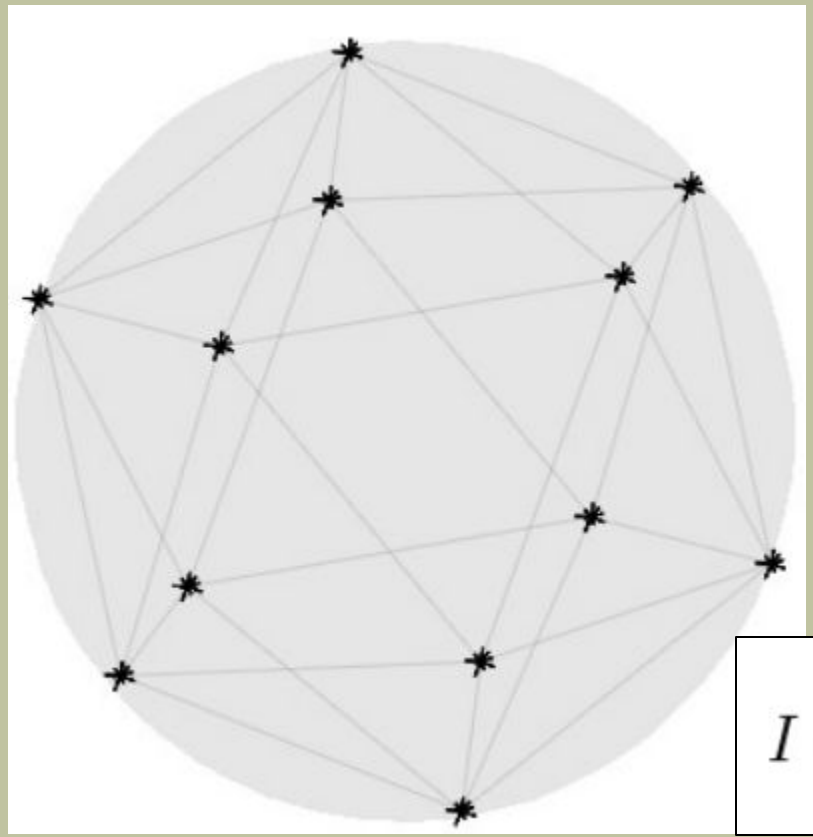
```
…/Lab6> nvcc -rdc=true  lab6.cu lab61.cu  -o lab6
…/Lab6> nvprof ./lab6 1024 128
0
-2
-4
-6
……
74.37%  31.392us  1  31.392us  31.392us  31.392us  gTestCM1(void)
5.61%  2.3680us    1  2.3680us  2.3680us  2.3680us  gTestCM(int*, int*)
5.38%  2.2720us    1  2.2720us  2.2720us  2.2720us  gTestGM(int*, int*, int*)
```

Текстурная память

● Текстура – специальный интерфейс доступа к глобальной памяти, обеспечивающий 1D, 2D и 3D целочисленную и **вещественную** индексацию.



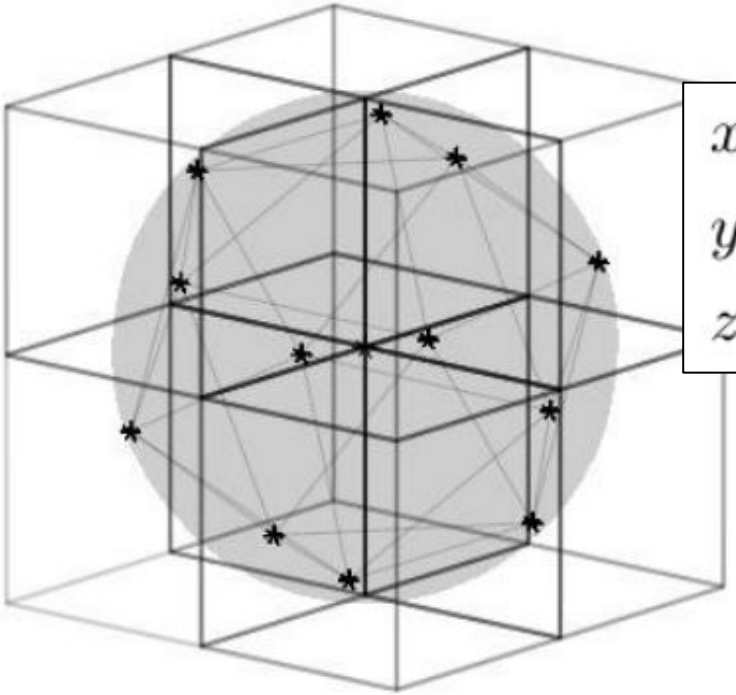
Интегрирование по сфере



Интеграл по сфере функции g(φ, ψ):

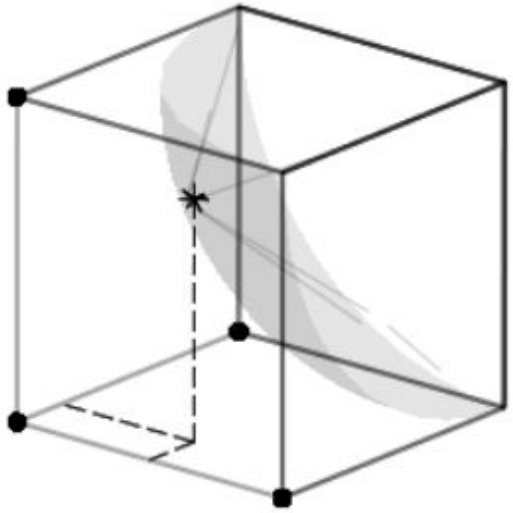
$$I = \int_{S^2} g(\phi, \psi) |\sin(\psi)| d\phi d\psi \approx \sum_i g_i (\Delta s)_i$$

Текстуры: аппаратная интерполяция (пример использования)



$$x = \sin(\psi)\cos(\phi),$$
$$y = \sin(\psi)\sin(\phi),$$
$$z = \cos(\psi).$$

Необходимость интерполяции:



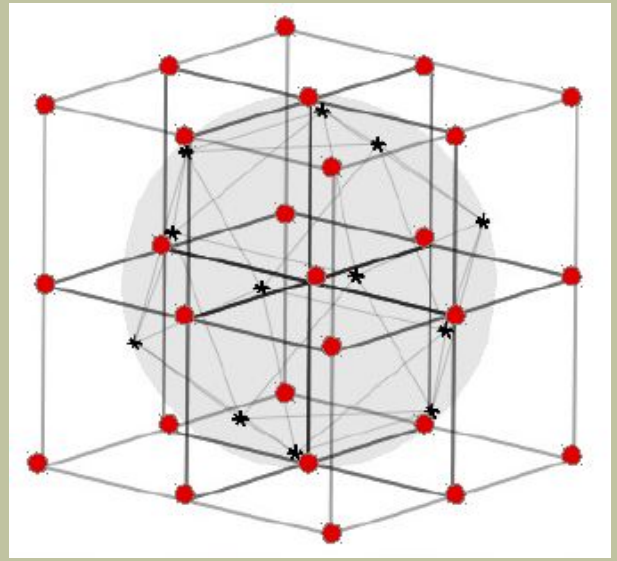
Тестовые функции, заданные в 3D пространстве

$$d_{yz} = \frac{1}{2}\sqrt{\frac{15}{\pi}}\frac{yz}{r^2};\ d_{z^2} = \frac{1}{4}\sqrt{\frac{5}{\pi}}\frac{-x^2-y^2+2z^2}{r^2};\ d_{x^2-y^2} = \frac{1}{4}\sqrt{\frac{15}{\pi}}\frac{x^2-y^2}{r^2}.$$

Вещественные сферические функции образуют **ортонормированный** базис в гильбертовом пространстве. Первые функции в разложении по этому базису (“поперечно-скошенная”, квадрупольная и бароподобная моды).

Задание: определить эффективность аппаратной интерполяции при использовании текстурной памяти.

Шаг 1: сохранить значения тестовой функции/функций в узлах трехмерной сетки в глобальной памяти.



Шаг 2: определить значения декартовых координат узлов сферической координатной сетки на сфере радиуса RADIUS.

```
struct Vertex
{
float x, y, z;
};

__constant__ Vertex vert[VERTCOUNT];

void init_vertices(){
  Vertex *temp_vert = (Vertex *)malloc(sizeof(Vertex) * VERTCOUNT);
  int i = 0;
```

```
  for (int iphi = 0; iphi < 2 * COEF; ++iphi)
    for (int ipsi = 0; ipsi < COEF; ++ipsi, ++i) {
      float phi = iphi * M_PI / COEF;
      float psi = ipsi * M_PI / COEF;
      temp_vert[i].x = RADIUS * sinf(psi) * cosf(phi);
      temp_vert[i].y = RADIUS * sinf(psi) * sinf(phi);
      temp_vert[i].z = RADIUS * cosf(psi);
    }
  cudaMemcpyToSymbol(vert, temp_vert, sizeof(Vertex) * VERTCOUNT, 0,
                                                    cudaMemcpyHostToDevice);
free(temp_vert);
}
```

Шаг 3: сохранить значения тестовой функции/функций, определенные на первом шаге в текстуре.

```
cudaArray* df_Array = 0;

void  load_texture(float *df_h){
  const cudaExtent volumeSize = make_cudaExtent(FGSIZE, FGSIZE,
                                                FGSIZE);
  cudaChannelFormatDesc channelDesc = cudaCreateChannelDesc<float>();
  cudaMalloc3DArray(&df_Array, &channelDesc, volumeSize);
  cudaMemcpy3DParms cpyParams={0};
```

```
  cpyParams.srcPtr = make_cudaPitchedPtr( (void*)df_h,
          volumeSize.width*sizeof(float),
           volumeSize.width, volumeSize.height);
  cpyParams.dstArray = df_Array;

  cpyParams.extent = volumeSize;
  cpyParams.kind = cudaMemcpyHostToDevice;

  cudaMemcpy3D(&cpyParams);
}
```

Шаг 4: настроить интерфейс доступа к текстуре.

```
texture<float, 3, cudaReadModeElementType>  df_tex;

void   tune_texture(cudaChannelFormatDesc channelDesc){
  df_tex.normalized = false;
  df_tex.filterMode = cudaFilterModeLinear;
  df_tex.addressMode[0] = cudaAddressModeClamp;
  df_tex.addressMode[1] = cudaAddressModeClamp;
  df_tex.addressMode[2] = cudaAddressModeClamp;

  cudaBindTextureToArray(df_tex, df_Array, channelDesc);
}
```

Шаг 5: реализовать вычисление интеграла, используя текстурную ссылку.

```
__global__ void kernel(float *a){
  __shared__ float cache[THREADSPERBLOCK];
  int tid = threadIdx.x + blockIdx.x * blockDim.x;
  int cacheIndex = threadIdx.x;
  float x = vert[tid].x;
  float y = vert[tid].y;
  float z = vert[tid].z;
  cache[cacheIndex] = tex3D(df_tex, z, y, x);

__syncthreads();
```

```
  for (int s = blockDim.x / 2; s > 0; s >>= 1){
    if (cacheIndex < s)
      cache[cacheIndex] += cache[cacheIndex + s];
      __syncthreads();
  }

  if (cacheIndex == 0)
  a[blockIdx.x] = cache[0];
}
```

суммирование посредством редукции

Шаг 6: реализовать функцию, выполняющую три-линейную интерполяцию и реализовать ядро, заменив текстурную ссылку вызовом этой функции.

Шаг 7: освободить ресурсы.

```
void release_texture(){
cudaUnbindTexture(df_tex);
cudaFreeArray(df_Array);
}
```

Шаг 8: сравнить время вычисления интеграла с использованием аппаратной и программной реализации интерполяции.