

Лекция 5

Текстурная и константная память.

- **Текстура** – специальный интерфейс доступа к глобальной памяти, обеспечивающий 1D, 2D и 3D целочисленную и вещественную (sic!) индексацию.
- **Константная память** – кэшируемая память для чтения небольшого размера (64K).

Текстуры: аппаратная интерполяция

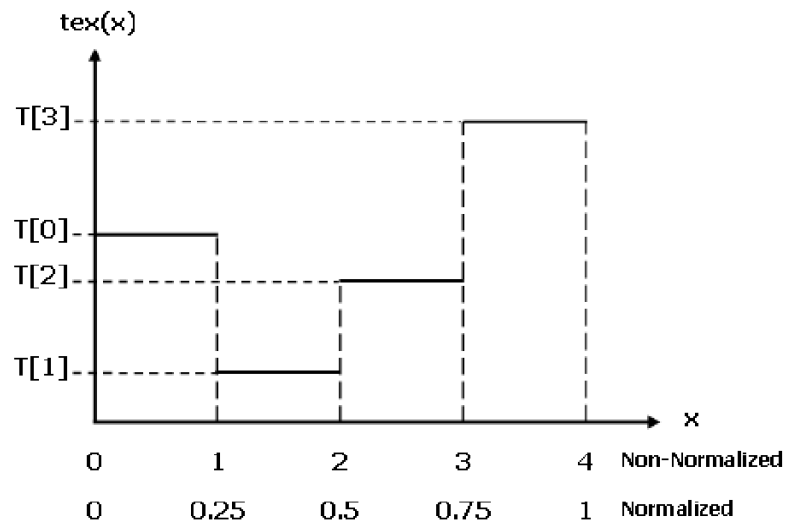


Figure 12 Nearest-Point Sampling of a One-Dimensional Texture of Four Texels

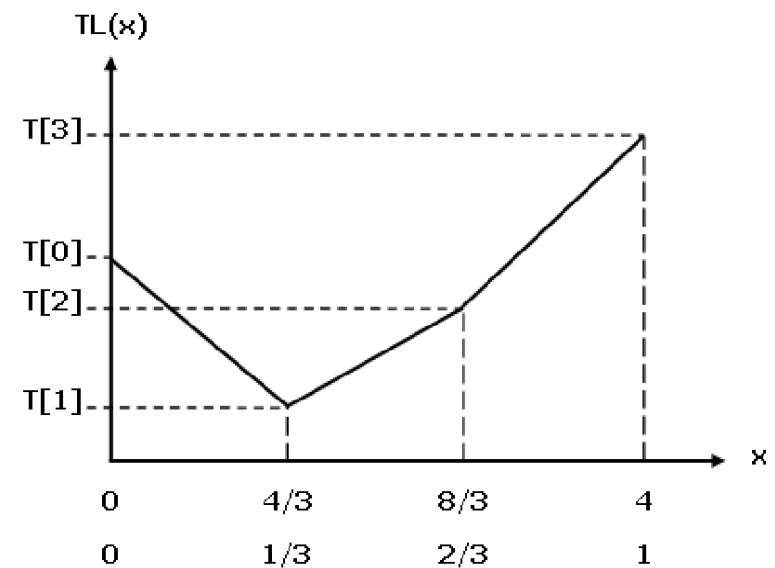
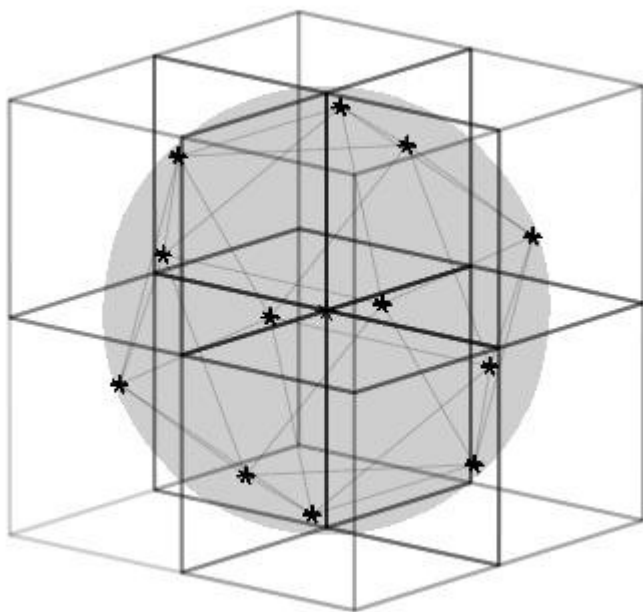
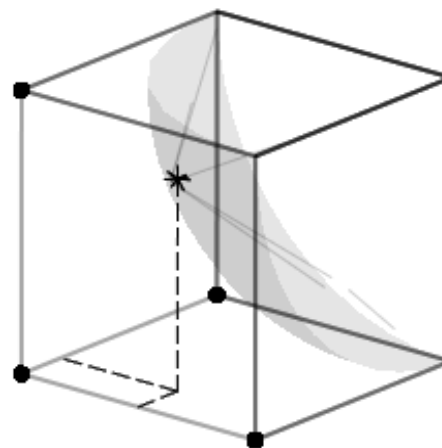


Figure 14 One-Dimensional Table Lookup Using Linear Filtering

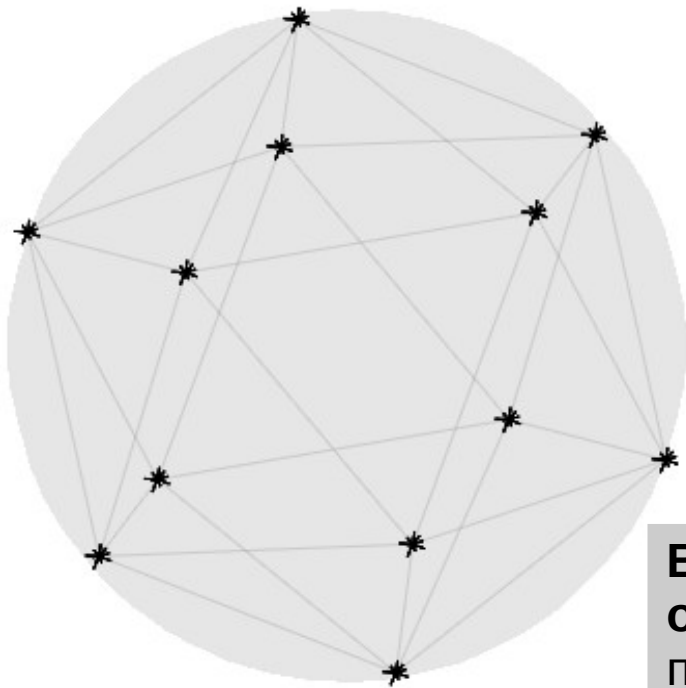
Текстуры: аппаратная интерполяция (пример использования)



Необходимость интерполяции:



Интегрирование по сфере



$$\begin{aligned}x &= \sin(\psi) \cos(\phi), \\y &= \sin(\psi) \sin(\phi), \\z &= \cos(\psi).\end{aligned}$$

Связь декартовых и сферических координат (на единичной сфере).

Интеграл по сфере функции $g(\phi, \psi)$:

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

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

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

Заголовок программы

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda.h>

#define M_PI 3.14159265358979323846
#define COEF 48
#define VERTCOUNT COEF*COEF*2
#define RADIUS 10.0f
#define FGSIZE 20
#define FGSHIFT FGSIZE/2
#define IMIN(A,B) (A<B?A:B)
#define THREADSPERBLOCK 256
#define BLOCKSPERGRID IMIN(32,
(VERTCOUNT+THREADSPERBLOCK-1)/THREADSPERBLOCK)
```

Объявление текстуры и глобальной переменной для использования константной памяти

```
typedef float(*ptr_f)(float, float, float);
```

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

```
__constant__ Vertex vert[VERTCOUNT];
```

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

```
cudaArray* df_Array = 0;
```

(без нормализации)

(текстурная ссылка)

(размерность текстуры)

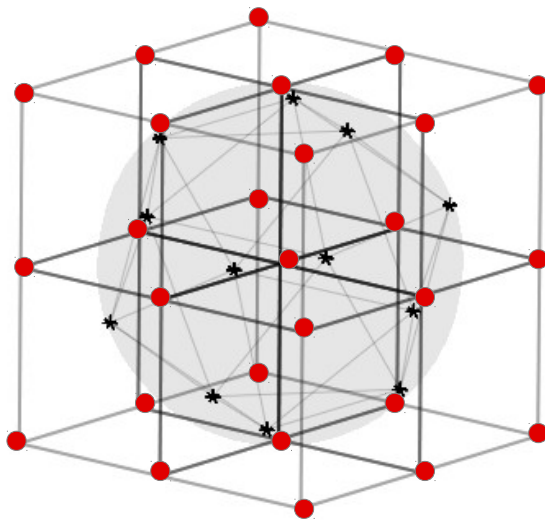
(указатель на область памяти, предназначенную для работы с текстурой)

Определение тестовой и проверочной функций

```
float func(float x, float y, float z){  
    return (0.5*sqrtf(15.0/M_PI))*(0.5*sqrtf(15.0/M_PI))*  
           z*z*y*y*sqrtf(1.0f-z*z/RADIUS/RADIUS)/RADIUS/RADIUS  
           /RADIUS/RADIUS;}  
  
float check(Vertex *v, ptr_f f){  
    float sum = 0.0f;  
  
    for (int i = 0; i < VERTCOUNT; ++i)  
        sum += f(v[i].x, v[i].y, v[i].z);  
  
    return sum;  
}
```

Дискретизация функций на прямоугольной сетке

```
void calc_f(float *arr_f, int x_size, int y_size, int z_size, ptr_f f){  
    for (int x = 0; x < x_size; ++x)  
        for (int y = 0; y < y_size; ++y)  
            for (int z = 0; z < z_size; ++z)  
                arr_f[z_size * (x * y_size + y) + z] = f(x - FGSHIFT, y -  
                                                            FGSHIFT, z - FGSHIFT);  
}
```



Определение узлов квадратуры на сфере в константной памяти.

Контрольное вычисление квадратуры

```
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);
        }
    }
    printf("sumcheck = %f\n", check(temp_vert, &func)*M_PI*M_PI/
                                                COEF/COEF);
    cudaMemcpyToSymbol(ver, temp_vert, sizeof(Vertex) *
                        VERTCOUNT, 0, cudaMemcpyHostToDevice);
    free(temp_vert);
}
```

Копирование данных с хоста в текстуру

```
void init_texture(float *df_h){  
    const cudaExtent volumeSize = make_cudaExtent(FG_SIZE,  
                                                    FG_SIZE, FG_SIZE);  
    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);  
}
```

Конфигурация текстуры: параметры фильтрации. Привязка текстуры к CUDA массиву и освобождение ресурсов

```
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);
}

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

Функция ядра для вычисления квадратуры (кэширование фильтрованных значений функции в узлах)

```
__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 + FGSHIFT + 0.5f;
    float y = vert[tid].y + FGSHIFT + 0.5f;
    float z = vert[tid].z + FGSHIFT + 0.5f;
    cache[cacheIndex] = tex3D(df_tex, z, y, x);

    __syncthreads();
}
```

Функция ядра для вычисления квадратуры (суммирование посредством редукции)

```
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];
}
```

Программа-драйвер для тестирования текстурной памяти

```
int main(void){  
    float *arr = (float *)malloc(sizeof(float) * FGSIZE * FGSIZE * FGSIZE);  
    float *sum = (float*)malloc(sizeof(float) * BLOCKSPERGRID);  
    float *sum_dev;  
  
    cudaMalloc((void**)&sum_dev, sizeof(float) * BLOCKSPERGRID);  
  
    init_vertexes();  
    calc_f(arr, FGSIZE, FGSIZE, FGSIZE, &func);  
    init_texture(arr);  
}
```

Программа-драйвер для тестирования текстурной памяти

```
kernel<<<BLOCKSPERGRID,THREADSPERBLOCK>>>(sum_dev);
CudaThreadSynchronize();

cudaMemcpy(sum, sum_dev, sizeof(float) * BLOCKSPERGRID,
           cudaMemcpyDeviceToHost);

float s = 0.0f;
for (int i = 0; i < BLOCKSPERGRID; ++i)
    s += sum[i];
printf("sum = %f\n", s*M_PI*M_PI / COEF/COEF);

cudaFree(sum_dev);
free(sum);
release_texture();
free(arr);
return 0;
}
```

Компиляция и выполнение программы

```
ewgenij@linux-715l:~/EDUCATION/workshop/cuda_course/Practicals/Practical4> nvcc sphere.cu -o sphere
ewgenij@linux-715l:~/EDUCATION/workshop/cuda_course/Practicals/Practical4> ./sphere
sumcheck = 1.0000000
sum = 1.003616
Time: 0.1 ms
```


ПРИЛОЖЕНИЯ

Объявление структур и функция для конфигурирования текстуры

```
struct cudaExtent {  
    size_t width;  
    size_t height;  
    size_t depth;  
};  
  
struct cudaExtent make_cudaExtent(size_t w, size_t h, size_t d);  
  
struct cudaChannelFormatDesc {  
    int x, y, z, w;  
    enum cudaChannelFormatKind f;  
};  
  
struct textureReference{  
    enum cudaTextureAddressMode  addressMode [3];  
    struct cudaChannelFormatDesc channelDesc;  
    enum cudaTextureFilterMode   filterMode;  
    int normalized;  
    int sRGB;  
};
```

Объявление структур и функция для конфигурирования текстуры (продолжение)

```
cudaError_t cudaMalloc3DArray(  
    struct cudaArray **    array,  
    const struct cudaChannelFormatDesc * desc,  
    struct cudaExtent      extent,  
    unsigned int           flags = 0  
)  
  
struct cudaMemcpy3DParms {  
    struct cudaArray *srcArray;  
    struct cudaPos   srcPos;  
    struct cudaPitchedPtr srcPtr;  
    struct cudaArray *dstArray;  
    struct cudaPos   dstPos;  
    struct cudaPitchedPtr dstPtr;  
    struct cudaExtent extent;  
    enum cudaMemcpyKind kind;  
};
```

(*NVIDIA CUDA Library Documentation*)