Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №7

По дисциплине: «Программирование графических процессоров»

Выполнили:

Студенты 3 курса группы ИП-211
Оганесян А.С.
Лацук А.Ю.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск, 2025

Задание:

- реализовать алгоритм вычисления интеграла функции, заданной

на прямоугольной сетке в трехмерном пространстве, на сфере с
использованием текстурной и константной памяти;
реализовать алгоритм вычисления интеграла функции, заданной
на прямоугольной сетке в трехмерном пространстве, на сфере
без использованием текстурной и константной памяти
(ступенчатую и линейную интерполяцию реализовать
программно);
● сравнить результаты и время вычислений обоими способами.

Цель: изучить преимущества использования константной и текстурной памяти.

Выполнение работы:

Для реализации задачи были использованы два подхода:

1. **Использование текстурной и константной памяти**. В этом случае данные функции были загружены в текстурную память для более эффективного доступа, а параметры сферы — в константную память.

2. **Использование линейной интерполяции без текстурной памяти**. В этом случае все вычисления производились с помощью стандартных методов, без использования специализированных типов памяти.

В первой части работы была реализована версия вычисления интеграла с использованием текстурной памяти для хранения данных функции. В данной версии CUDA-ядро использовало текстуру для быстрого доступа к данным функции в каждой точке сетки на сфере. Константная память использовалась для хранения параметров сетки.

Ядро CUDA выполняло параллельные вычисления для каждой точки на сетке, рассчитывая значения функции и площади элементарных участков. Суммирование результатов происходило через атомарные операции, чтобы избежать конфликтов при параллельных вычислениях.

```
#include <iostream>
#include <cmath>
#include <cuda_runtime.h>

#define N 512
#define PI 3.14159265358979323846

texture<float, 2, cudaReadModeElementType> texData;

__constant__ float sphereParams[3];

__device__ float func(float theta, float phi)
{
    return sinf(theta) * cosf(phi);
}

__global__ void computeIntegralWithTexture(float *result)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int idy = blockIdx.y * blockDim.y + threadIdx.y;

    if (idx < N && idy < N)
    {
        float theta = (float)idx * 2 * PI / N;
        float phi = (float)idy * PI / N;

        float value = func(theta, phi);

        float dA = sinf(theta) * (2 * PI / N) * (PI / N);

        atomicAdd(result, value * dA);
    }
}

int main()
{
    float *d_result, h_result = 0.0f;
    cudaMalloc(&d_result, sizeof(float));
    cudaMemcpy(d_result, &h_result, sizeof(float),
cudaMemcpyHostToDevice);
```

```
    float sphereParamsHost[3] = {1.0f, 2 * PI / N, PI / N};
    cudaMemcpyToSymbol(sphereParams, sphereParamsHost, sizeof(float)
* 3);

    dim3 blockSize(16, 16);
    dim3 gridSize((N + blockSize.x - 1) / blockSize.x, (N +
blockSize.y - 1) / blockSize.y);

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start);

    computeIntegralWithTexture<<<gridSize, blockSize>>>(d_result);

    cudaEventRecord(stop);
    cudaEventSynchronize(stop);

    float milliseconds = 0;
    cudaEventElapsedTime(&milliseconds, start, stop);
    std::cout << "Time with texture memory: " << milliseconds << "
ms" << std::endl;

    cudaMemcpy(&h_result, d_result, sizeof(float),
cudaMemcpyDeviceToHost);

    std::cout << "Integral result with texture memory: " << h_result
<< std::endl;

    cudaFree(d_result);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);

    return 0;
}
```

Листинг 1 – texture.cu

Во второй части работы была реализована версия вычисления интеграла без использования текстурной памяти. Для оценки значений функции в каждой точке сетки использовалась стандартная формула функции $f(\theta, \phi) = sin(\theta) \cdot cos(\phi)$.

Реализация не использовала текстурную память, а вместо этого использовала линейную интерполяцию для вычислений. Метод линейной интерполяции был применён для оценки значений функции между соседними точками.

```
#include <iostream>
#include <cmath>
#include <cuda_runtime.h>

#define N 512
#define PI 3.14159265358979323846

__device__ float func(float theta, float phi)
{
    return sinf(theta) * cosf(phi);
}

__global__ void computeIntegralWithoutTexture(float *result)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int idy = blockIdx.y * blockDim.y + threadIdx.y;

    if (idx < N && idy < N)
    {
        float theta = (float)idx * 2 * PI / N;
        float phi = (float)idy * PI / N;

        float value = func(theta, phi);

        float dA = sinf(theta) * (2 * PI / N) * (PI / N);

        atomicAdd(result, value * dA);
    }
}

int main()
{
    float *d_result, h_result = 0.0f;
```

```
    cudaMalloc(&d_result, sizeof(float));
    cudaMemcpy(d_result, &h_result, sizeof(float),
cudaMemcpyHostToDevice);

    dim3 blockSize(16, 16);
    dim3 gridSize((N + blockSize.x - 1) / blockSize.x, (N +
blockSize.y - 1) / blockSize.y);

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start);

    computeIntegralWithoutTexture<<<gridSize, blockSize>>>(d_result);

    cudaEventRecord(stop);
    cudaEventSynchronize(stop);

    float milliseconds = 0;
    cudaEventElapsedTime(&milliseconds, start, stop);
    std::cout << "Time without texture memory: " << milliseconds << "
ms" << std::endl;

    cudaMemcpy(&h_result, d_result, sizeof(float),
cudaMemcpyDeviceToHost);

    std::cout << "Integral result without texture memory: " <<
h_result << std::endl;

    cudaFree(d_result);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);

    return 0;
}
```

Листинг 2 – linear.cu

Для каждого из подходов был измерен время выполнения и вычислен результат интеграла. Было установлено, что время выполнения с использованием текстурной памяти оказалось меньше по сравнению с реализацией без неё, что объясняется более быстрым доступом к данным через текстуры в случае с GPU.

Также результаты вычислений для обоих методов незначительно отличаются, что подтверждает правильность работы обеих реализаций.

```
$ ./texture
Time with texture memory: 0.597824 ms
Integral result with texture memory: 0.0192682
$ ./linear
Time without texture memory: 0.641024 ms
Integral result without texture memory: 0.0192756
```

Листинг 3 – Результат работы программ

Вывод:
В результате работы были реализованы два подхода для вычисления интеграла функции на сфере с использованием CUDA. Оба метода продемонстрировали правильность расчетов, однако использование текстурной памяти позволило достичь более высокой производительности. В ходе работы нам удалось понять, что текстурная память обеспечивает быстрый доступ к данным, а константная память минимизирует количество обращений к глобальной памяти.