МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский ядерный университет «МИФИ»

ОТЧЕТ ПО ДИСЦИПЛИНЕ «ПРОЕКТНАЯ ПРАКТИКА»
РАЗРАБОТКА ПРОГРАММЫ-СИМУЛЯТОРА РАБОТЫ ДЕТЕКТОРА ИОНИЗИРУЮЩЕГО ИЗЛУЧЕНИЯ

Исполнитель:
Студент группы Б24-602 Варакин Д.А.

подпись, дата

Руководитель работы: Левцов И.С.

подпись, дата

Москва 2025

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ..... 1
РЕФЕРАТ..... 2
ВВЕДЕНИЕ..... 3
Взаимодействие гамма излучения с веществом..... 3
Вычисление длины свободного пробега..... 4
Определение сечений взаимодействия гамма-излучения с веществом..... 5
Описание конфигурации излучателя и поглотителя..... 6
Описание основного алгоритма..... 7
Получение спектра посредством уширения гистограммы с помощью распределения Гаусса..... 8
Пересечение луча с кубом..... 8
ПРАКТИЧЕСКАЯ ЧАСТЬ РАБОТЫ..... 9
ЗАКЛЮЧЕНИЕ..... 12
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ..... 13
ПРИЛОЖЕНИЕ..... 14

РЕФЕРАТ

Отчёт в рамках проектной практики 18 стр, 7 рис., 4 источника.

Цель работы: разработка симулятора работы детектора ионизирующего излучения.

Методы исследования: обзор источников литературы, разработка программы-симулятора на языке C++.

Область применения: экспериментальная ядерная физика.

ВВЕДЕНИЕ

Детектор ионизирующего излучения - устройство, преобразующее энергию радиоактивного излучения в другой вид энергии, удобный для регистрации, например в электрическую. Прохождение частиц через чувствительный объем такого детектора сопровождается импульсами электрического тока на выходе детектора. Целью работы является получение спектра сцинтилляционного детектора, сделанного из иодида натрия(NaI) с помощью компьютерного моделирования. В данной работе реализован метод Монте-Карло для моделирования взаимодействия гамма-излучения с детектором. Данный метод является классическим для моделирования процессов внутри детекторов.

Взаимодействие гамма излучения с веществом

Гамма-излучение - вид электромагнитного излучения, которое имеет высокую энергию и короткую длину волны. Проникая вглубь материала, гамма-излучение взаимодействует с атомами и молекулами вещества, через которое оно проходит.

Вероятность взаимодействия излучения с атомами характеризуется величиной, которую называют сечением взаимодействия. Чем больше сечение взаимодействия, тем выше вероятность того, что гамма-квант взаимодействует с атомами вещества на своем пути. Когда говорится о взаимодействии гамма-квантов с веществом, подразумеваются отдельные столкновения частицы с атомами вещества.

Существует несколько основных способов взаимодействия гамма-излучения с веществом:

- Фотоэффект. Гамма-квант передает всю свою энергию атому, выбивая из него электрон.

- Комптоновское рассеяние. Гамма квант сталкивается с электроном, теряет часть своей энергии и изменяет направление движения.
- Образование электрон-позитронных пар. Гамма-квант поглощается, происходит образование электрона и позитрона.

Вычисление длины свободного пробега.

Длина свободного пробега (λ) - расстояние, которое гамма-квант проходит, прежде чем столкнуться с каким либо атомом и взаимодействовать с ним. Для гамма-излучения длина свободного пробега зависит от плотности материала и его состава.

В данной работе будет использоваться алгоритм по созданию случайной величины свободного пробега. Для этого рассмотрим функцию закона экспоненциального распределения – интеграл по всей её длине будет равняться 1. Это означает, что гамма-квант, что пролетит бесконечное расстояние провзаимодействует с вероятностью в 100%.

$$\int_0^\infty p(x)dx = 1$$

Тогда, пусть, шанс взаимодействия на некотором отрезке (a,b) длиной (a, ξ) будет равен γ.

$$\int_0^\xi p(x)dx = \gamma$$

Следовательно, γ на случайном отрезке (a, ξ) будет случайной величиной от 0 до 1, а точка ξ будет являться длиной свободного пробега λ. Закон экспоненциального распределения:

$$p(x) = \Sigma \cdot \exp(-\Sigma \cdot x)$$

Где Σ – полное сечение взаимодействия, x – пройденный путь частицы. Следовательно:

$$\int_{a}^{\lambda} \Sigma \cdot \exp(-\Sigma \cdot x)\, dx = \gamma$$

$$1 - \exp(-\Sigma \cdot \lambda) = \gamma$$

$$\lambda = -(1 / \Sigma) \ln(\gamma)$$

Где γ – случайная величина от 0 до 1.

Определение сечений взаимодействия гамма-излучения с веществом

Полное сечение взаимодействия фотонов с электронами атомов складывается из:

1) сечения фотоэффекта:

$$\sigma_{ф} = 6{,}651 \cdot \ 10^{-25} \cdot 4 \cdot \sqrt{2} \cdot \frac{Z^5}{137^4}\left(\frac{mc^2}{E}\right)^{7/2}$$

где mc^2 – энергия покоя электрона равная 0,511 МэВ, Е- энергия фотона, а Z – заряд атома

2) сечения комптоновского рассеяния:

$$\sigma_K = 6,651 \cdot 10^{-25} \cdot \frac{3 \cdot Z}{8 \cdot \gamma} \cdot \left\{\left[1 - \frac{2(\gamma+1)}{\gamma^2}\right] \cdot ln(2 \cdot \gamma + 1) + \frac{1}{2} + \frac{4}{\gamma} - \frac{1}{2 \cdot (2\gamma+1)^2}\right\}$$

где $\gamma = E / m_0 c^2$

Энергия гамма-кванта после взаимодействия:

$$E_{\gamma}^{`} = \frac{E_{\gamma}}{1+\frac{E_{\gamma}}{m_e c^2}(1-cos\theta)}$$

3) сечения образования пар:

Процесс образования пар происходит только при энергиях фотонов, превышающих суммарную энергию покоя электрона и позитрона, т.е. при

$$E_{\gamma} > 2 * m_0 * c^2$$

Описание конфигурации излучателя и поглотителя

В качестве источника гамма-излучения принимается точечный источник Cs-137, испускающий гамма-кванты равномерно по сфере. В качестве детектора используется куб иодида натрия (NaI). Пространственная конфигурация приведена на рисунке 1:

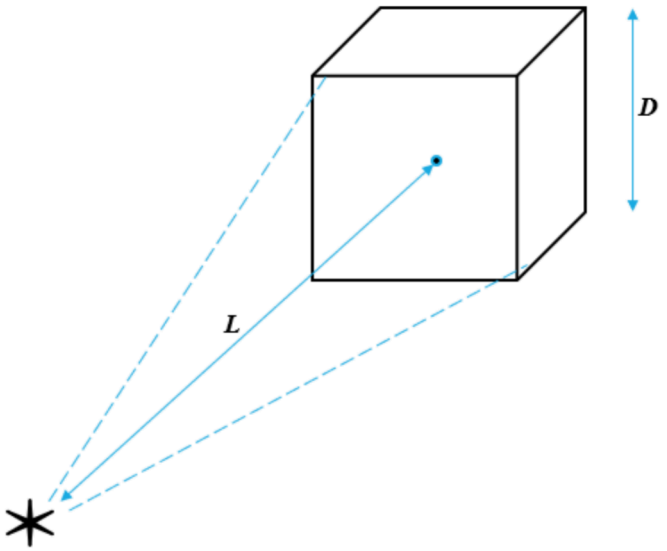

Рисунок 1 - *модель куба и источника излучения*

В программе куб помещается в начало координат, а фотон появляется с случайным направлением в точке с координатой (L + D/2, 0, 0)

Данная модель исследуется при следующих значениях D, L, N, где N - число испускаемых фотонов:

1. D = 100мм, L = 100мм, N = 10^5 фотонов
2. D = 100мм, L = 200мм, N = 5 * 10^5 фотонов
3. D = 100мм, L = 500мм, N = 10^6 фотонов

Описание основного алгоритма

Фотон появляется в начальной точке с энергией 0.662МэВ. Далее, если фотон пересекается с кубом, то перемещаем фотон к границе куба и используем метод Монте-Карло: пока фотон не поглощен вычисляем длину свободного пробега, перемещаем фотон на длину свободного пробега(Если в процессе перемещения фотон прошел через границу куба, то цикл останавливается, фотон считается ушедшим на бесконечность).

После перемещения происходит одно из трех событий: фотоэффект, комптоновское рассеяние или образование электрон-позитронной пары(образование пар в данной работе отсутствует, так как энергия 0.662МэВ слишком мала, преобладает фотоэффект и комптон рассеяние). По завершению этого цикла есть 2 возможных случая:

1) фотон польностью поглощен(delta_E = E_full)

2) фотон потерял часть энергии (delta_E < E_full) и покинул объем детектора, после чего он считается улетевшим на бесконечность.

В результате, величина delta_E заносится в динамический массив. Этот массив отданных энергий - итог работы программы, на основании него строится гистограмма и график. Программа реализована на языке программирования C++, основная логика размещена в файле main.cpp, содержимое которого приведено в Приложении 1.

Получение спектра посредством уширения гистограммы с помощью распределения Гаусса.

$$FW(E) = F * \sqrt{E}$$

$$\sigma = \frac{FW(E)}{2 * \sqrt{2 * ln2}}$$

$$N(E, i, \sigma) = \frac{N_{old}(E)}{\sigma * \sqrt{2\pi}} * exp(\frac{-(i-P)^2}{2 * \sigma^2})$$

Где FW(E) – ПШПВ исправленного пика, F – произвольный коэффициент, N_{old} (E) – ранее высчитанное значение отчетов для данной энергии E, P – номер пика центра фотопика, i – номер канала.

По итогу набор нормальных распределений в количестве равному числу каналов, складывается в единый аппаратурный спектр.

Пересечение луча с кубом.

Для поиска пересечения луча и куба используется внешняя библиотека трассировки лучей nanort.

ПРАКТИЧЕСКАЯ ЧАСТЬ РАБОТЫ

1. D = 100мм, L = 100мм, N = 10^5 фотонов:

Вывод основной программы:

```
P_absorption: 0.885443
cnt_fully_absorbed = 5448
cnt_partitialy_absorbed = 968
Intersection counter: 6416
Histogram saved to: histogram.txt
Execution time: 142 ms
```

Гистограмма:

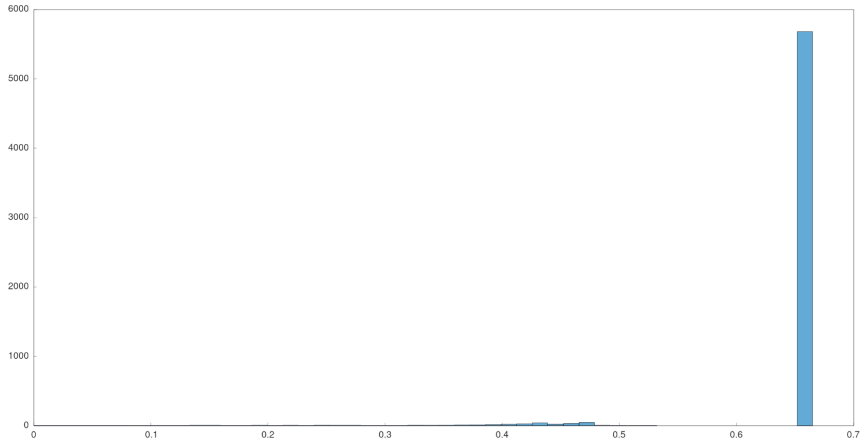

Рисунок 2 - Идеальный спектр конфигурации №1

Аппаратурный спектр:

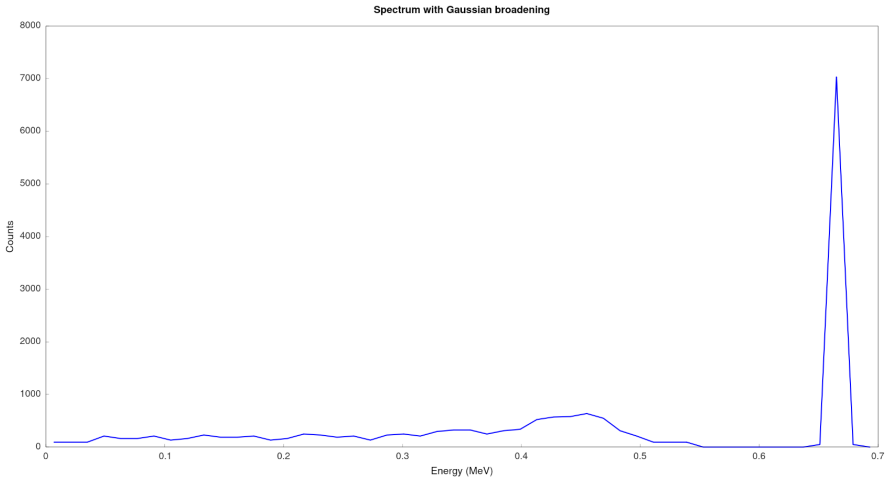

Рисунок 3 - Аппаратурный спектр конфигурации №1

2. D = 100мм, L = 200мм, N = 5 * 10^5 фотонов

Вывод основной программы:

```
P_absorption: 0.911482
cnt_fully_absorbed = 8326
cnt_partitialy_absorbed = 1141
Intersection counter: 9467
Histogram saved to: histogram.txt
Execution time: 512 ms
```

Гистограмма:

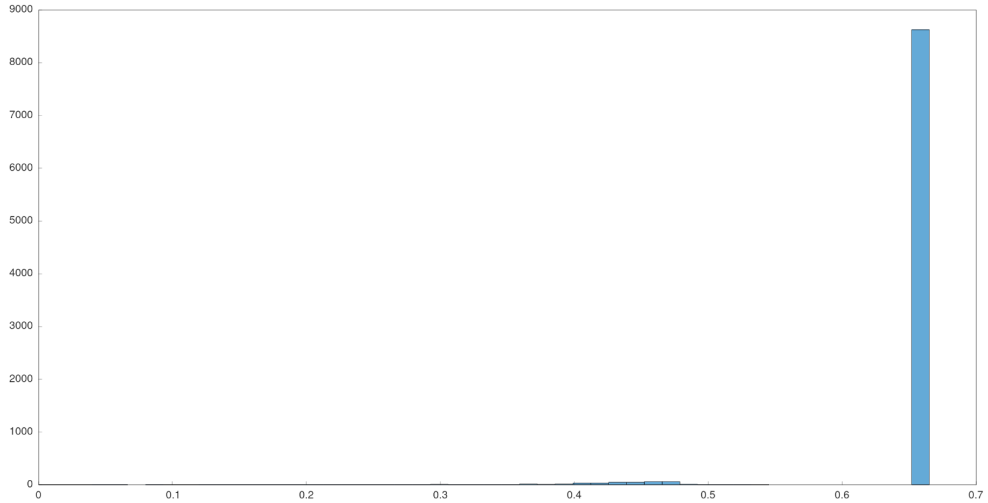

Рисунок 4 - Идеальный спектр конфигурации №2

Аппаратурный спектр (рис. 2б):

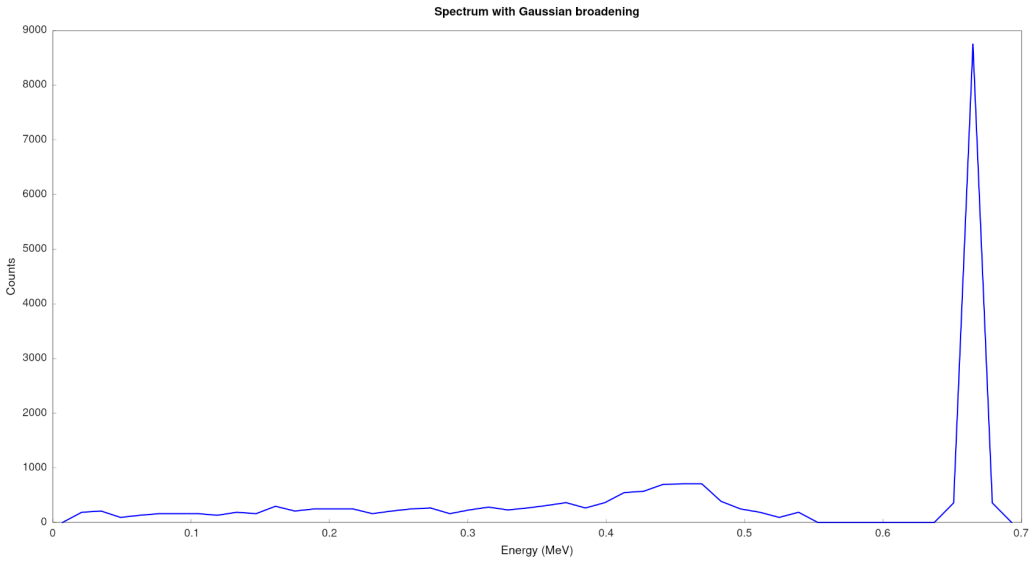

Рисунок 5 - Аппаратурный спектр конфигурации №2

3. D = 100мм, L = 500мм, N = 10^6 фотонов

Вывод основной программы:

```
P_absorption: 0.933039

cnt_fully_absorbed = 2850

cnt_partitialy_absorbed = 316

Intersection counter: 3166

Histogram saved to: histogram.txt

Execution time: 871 ms
```

Гистограмма (рис. 3а):

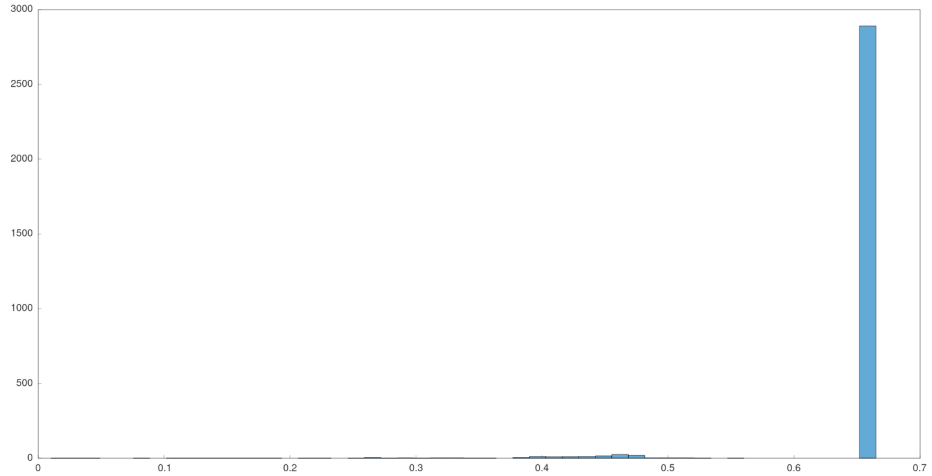

Рисунок 6 - Идеальный спектр конфигурации №3

Аппаратурный спектр (рис. 3б):

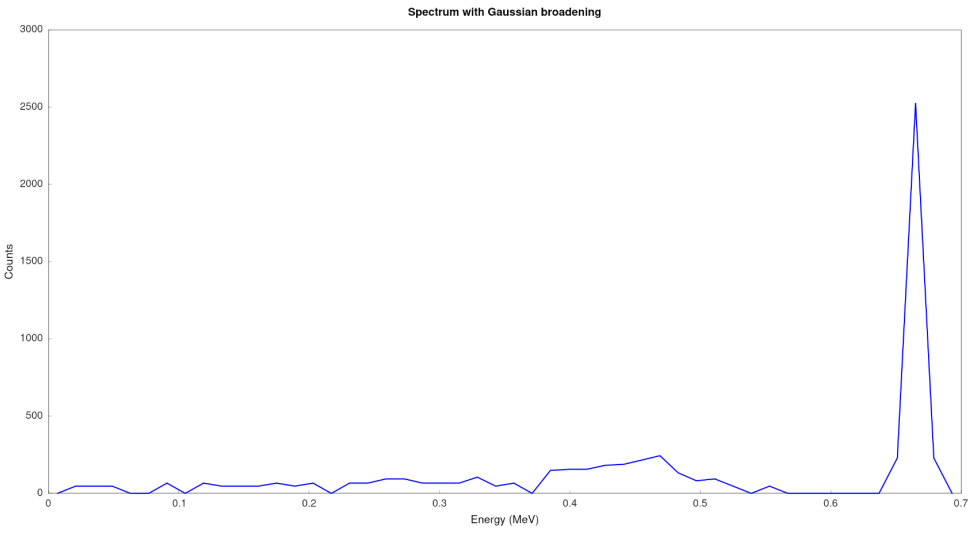

Рисунок 7 - Аппаратурный спектр конфигурации №3

ЗАКЛЮЧЕНИЕ

Разработан симулятор детектора ионизирующего излучения. Программа реализована на языке программирования C++, протестирована в трех сценариях размещения детектора относительно источника.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Практическая спектрометрия ядерных излучений: учебное пособие / А. В. Бушуев, А. Ф. Кожин, Т. Б. Алеева [и др.]. – Москва: Национальный исследовательский ядерный университет "МИФИ", 2016. – 260 с.
2. Бойко, Н. В. Физика взаимодействия заряженных частиц и гамма-излучения с веществом / Н. В. Бойко, С. В. Колесников, С. Г. Рудаков. – Обнинск: ФГБУ «ВНИИГМИ-МЦД», 2023. – 112 с.
3. Программное обеспечение «Lsrm». Алгоритмические основы – функции обработки спектрометрической информации. – Московская обл., г. Солнечногорск: ООО «ЛСРМ». 2023 – 56 с
4. NanoRT, single header only modern ray tracing kernel. URL - https://github.com/lighttransport/nanort (дата обращения: 01.11.2025)

ПРИЛОЖЕНИЕ

Приложение 1

```
#include <iostream>
#include <random>
#include <numbers>

#include <cxxopts.hpp>

#include "Gamma3.hpp"

using namespace gamma3;

#define N_A 6.02214076e23
#define RAY_COUNT 1000000
#define DISTANCE 500
#define ENERGY_MEV 0.662

physics::Photon spawn_photon_at_point(const geometry::Point& p, double E_MeV);
geometry::Surface create_cube_detector_mm(double cube_size_mm = 100.0, const
geometry::Point& center = geometry::Point(0, 0, 0));
double compute_sigma_Atom(double E_MeV, const materials::Atom& atom,
etc::Function2<double>& f);
double compute_sigma_Molecule(double E_MeV, const materials::Molecule& molecule,
etc::Function2<double>& f);
double compute_sigma_Material(double E_MeV, const materials::Material& material,
etc::Function2<double>& f);


int main(int argc, char** argv) {

    //CLI INTERFACE

    cxxopts::Options options("Gamma3", "Gamma simulation");

    options.add_options()
        ("r,repo", "Path to Database.json",
cxxopts::value<std::string>()->default_value("../../Database.json"))
        ("o,output", "Output histogram file",
cxxopts::value<std::string>()->default_value("histogram.txt"))
        ("h,help", "Print help");

    auto result = options.parse(argc, argv);

    if (result.count("help")) {
        std::cout << options.help() << std::endl;
        return 0;
    }

    std::string repo_path = result["repo"].as<std::string>();
    std::string output_path = result["output"].as<std::string>();
```

```
    //SIMULATION
    try {
        auto start = std::chrono::high_resolution_clock::now();

        auto repo = io::JsonRepository(repo_path);

        auto NaI = repo.loadMaterial("NaI");
        auto sigma_photo_formula = repo.loadFormula2("SigmaPhoto");
        auto sigma_compton_formula = repo.loadFormula2("SigmaCompton");
        auto sigma_pair_formula = repo.loadFormula2("SigmaPair");

        auto cube = create_cube_detector_mm();

        //cube.rotate(std::numbers::pi/4, geometry::Vector(0,0,1));
        auto gen = std::mt19937(std::random_device{}());
        auto dist = std::uniform_real_distribution<double>(0, 1);
        int cnt = 0;
        int cnt_fully_absorbed = 0;
        int cnt_partitialy_absorbed = 0;
        int absorbed_photons = 0;
        std::vector<double> histogram;

        for (int i = 0; i < RAY_COUNT; ++i) {
            auto ph = spawn_photon_at_point(geometry::Point(50 + DISTANCE, 0, 0),
ENERGY_MEV);
            auto intersection = cube.intersect(ph.ray());
            if (intersection != std::nullopt) {
                cnt++;
                ph.move(intersection->t * 1.0000001);

                bool photon_absorbed = false;

                double delta_E = 0.0;

                while (!photon_absorbed) {
                    auto gamma = dist(gen);

                    double sigma_photo = compute_sigma_Material(ph.energy(), NaI,
sigma_photo_formula);
                    double sigma_compton = compute_sigma_Material(ph.energy(), NaI,
sigma_compton_formula);
                    double sigma_pair = compute_sigma_Material(ph.energy(), NaI,
sigma_pair_formula);
                    double sigma = sigma_photo + sigma_compton + sigma_pair;

                    double lambda = -(1 / sigma) * log(gamma);
                    ph.move(lambda);

                    if (cube.intersect(ph.ray())->t < lambda) break;

                    double P_photo = sigma_photo / sigma;
                    double P_compton = sigma_compton / sigma;
                    double P_pair = sigma_pair / sigma;
```

```
                    double random_value = dist(gen);

                    if (random_value < P_photo) {
                        // Photoabsorption
                        delta_E += ph.energy();
                        absorbed_photons++;
                        photon_absorbed = true;
                    }
                    else if (random_value < P_photo + P_compton) {
                        // Compton scattering
                        auto old_dir = glm::normalize(ph.ray().direction());
                        double alpha = ph.energy() / 0.511;
                        double theta = 2.0 * std::numbers::pi * dist(gen);
                        double phi = std::acos(1.0 - 2.0 * dist(gen));

                        geometry::Vector direction(
                            std::sin(phi) * std::cos(theta),
                            std::sin(phi) * std::sin(theta),
                            std::cos(phi)
                            );

                        auto new_dir = glm::normalize(direction);
                        double cos_theta = old_dir.x * new_dir.x + old_dir.y *
new_dir.y + old_dir.z * new_dir.z;
                        double new_energy = ph.energy() / (1.0 + alpha * (1.0 -
cos_theta));

                        double energy_loss = ph.energy() - new_energy;
                        ph.setDirection(new_dir);
                        ph.setEnergy(new_energy);
                        delta_E += energy_loss;
                    }
                    else if (random_value < P_photo + P_compton + P_pair) {
                        // Pair production
                        delta_E += ph.energy();
                        absorbed_photons++;
                        photon_absorbed = true;
                    }
                }
                if (delta_E == ENERGY_MEV) cnt_fully_absorbed++;
                else {
                    cnt_partitialy_absorbed++;
                }

                if(delta_E != 0) histogram.push_back(delta_E);
            }
        }


        std::cout << std::endl;

        std::cout << "P_absorption: " << static_cast<double>(absorbed_photons) / cnt
<< std::endl;
```

```
        std::cout << "cnt_fully_absorbed = " << cnt_fully_absorbed << std::endl;

        std::cout << "cnt_partitialy_absorbed = " << cnt_partitialy_absorbed <<
std::endl;

        std::cout << "Intersection counter: " << cnt << std::endl;

        {
            std::ofstream fout(output_path);
            if (!fout) {
                std::cerr << "Cannot open output file: " << output_path << std::endl;
            } else {
                for (double v : histogram) fout << v << "\n";
                std::cout << "Histogram saved to: " << output_path << std::endl;
            }
        }

        auto end = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end -
start);
        std::cout << "Execution time: " << duration.count() << " ms" << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}

// HELPER FUNCTIONS
physics::Photon spawn_photon_at_point(const geometry::Point& p, double E_MeV) {
    std::random_device rd;
    static std::mt19937 rng(rd());
    std::uniform_real_distribution<double> dist(0.0, 1.0);

    double theta = 2.0 * std::numbers::pi * dist(rng);
    double phi = std::acos(1.0 - 2.0 * dist(rng));

    geometry::Vector direction(
        std::sin(phi) * std::cos(theta),
        std::sin(phi) * std::sin(theta),
        std::cos(phi)
        );

    return physics::Photon(geometry::Ray(p, direction), E_MeV);
}

geometry::Surface create_cube_detector_mm(double cube_size_mm, const geometry::Point&
center) {
    double half_size = cube_size_mm / 2.0;

    std::vector<geometry::Point> vertices = {
        {center.x - half_size, center.y - half_size, center.z - half_size},
```

```
        {center.x + half_size, center.y - half_size, center.z - half_size},
        {center.x + half_size, center.y + half_size, center.z - half_size},
        {center.x - half_size, center.y + half_size, center.z - half_size},

        {center.x - half_size, center.y - half_size, center.z + half_size},
        {center.x + half_size, center.y - half_size, center.z + half_size},
        {center.x + half_size, center.y + half_size, center.z + half_size},
        {center.x - half_size, center.y + half_size, center.z + half_size}
    };

    std::vector<glm::u32vec3> faces = {
        {0, 2, 1}, {0, 3, 2},
        {4, 5, 6}, {4, 6, 7},
        {0, 1, 5}, {0, 5, 4},
        {1, 2, 6}, {1, 6, 5},
        {2, 3, 7}, {2, 7, 6},
        {3, 0, 4}, {3, 4, 7}
    };

    return geometry::Surface(vertices, faces);
}

double compute_sigma_Atom(double E_MeV,
                          const materials::Atom& atom,
                          etc::Function2<double>& f) {
    return f(atom.Z(), E_MeV);
}

double compute_sigma_Molecule(double E_MeV,
                              const materials::Molecule& molecule,
                              etc::Function2<double>& f) {
    auto& composition = molecule.components();
    double sum = 0.0;
    for(const auto& [atom, count] : composition) sum += count *
compute_sigma_Atom(E_MeV, atom, f);
    return sum;
}

double compute_sigma_Material(double E_MeV,
                              const materials::Material& material,
                              etc::Function2<double>& f) {
    auto& composition = material.components();
    double sum = 0.0;
    for(const auto& [molecule, mass_fraction] : composition) {
        double M = molecule.mass();
        sum += mass_fraction * (N_A / M) * compute_sigma_Molecule(E_MeV, molecule,
f);
    }
    return sum * material.density();
}
```

Листинг 1 - main.cpp