#include <CL/sycl.hpp>

#include <unordered\_map>

#include <chrono>

#include <iostream>

namespace sycl = cl::sycl;

class PointCloudHash {

public:

PointCloudHash(size\_t size, sycl::queue& q) : hash\_table(size, q) {}

void insert(const std::vector<float>& point, sycl::queue& q) {

uint32\_t hash = compute\_hash(point);

hash\_table[hash].push\_back(point, q);

}

std::vector<std::vector<float>> query(const std::vector<float>& point, sycl::queue& q) {

uint32\_t hash = compute\_hash(point);

return hash\_table[hash].get\_vector(q);

}

private:

uint32\_t compute\_hash(const std::vector<float>& point) {

// TODO: compute hash

}

sycl::usm::vector<std::vector<float>> hash\_table;

};

void build\_hash\_table(std::vector<std::vector<float>>& points, sycl::queue& q) {

size\_t size = points.size() \* 2;

PointCloudHash hash\_table(size, q);

// Insert points into the hash table

for (size\_t i = 0; i < points.size(); i++) {

hash\_table.insert(points[i], q);

}

q.wait();

}

void query\_hash\_table(std::vector<std::vector<float>>& points, PointCloudHash& hash\_table, sycl::queue& q) {

// Query hash table for each point

for (size\_t i = 0; i < points.size(); i++) {

auto result = hash\_table.query(points[i], q);

// TODO: process result

}

q.wait();

}

int main() {

std::vector<std::vector<float>> points(1000);

// Initialize points

for (size\_t i = 0; i < 1000; i++) {

points[i] = {static\_cast<float>(i), static\_cast<float>(i + 1), static\_cast<float>(i + 2)};

}

sycl::queue q(sycl::gpu\_selector{});

auto start = std::chrono::high\_resolution\_clock::now();

build\_hash\_table(points, q);

auto end = std::chrono::high\_resolution\_clock::now();

auto build\_time = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Build hash table time: " << build\_time.count() << " us" << std::endl;

PointCloudHash hash\_table(points.size() \* 2, q);

start = std::chrono::high\_resolution\_clock::now();

build\_hash\_table(points, q);

end = std::chrono::high\_resolution\_clock::now();

build\_time = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Build hash table time: " << build\_time.count() << " us" << std::endl;

start = std::chrono::high\_resolution\_clock::now();

query\_hash\_table(points, hash\_table, q);

end = std::chrono::high\_resolution\_clock::now();

auto query\_time = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Query hash table time: " << query\_time.count() << " us" << std::endl;

return 0;

}