#include <iostream>

#include <vector>

#include <omp.h>

// Define field size constants

const int SIZE\_X = 100;

const int SIZE\_Y = 100;

// Define field data structures

struct Field {

std::vector<float> data;

};

// Function to initialize fields

void initializeFields(Field& Ez, Field& Hx, Field& Hy) {

Ez.data.resize(SIZE\_X \* SIZE\_Y);

Hx.data.resize(SIZE\_X \* SIZE\_Y);

Hy.data.resize(SIZE\_X \* SIZE\_Y);

// Initialize fields to some initial values if needed

}

// Function to update Ez field using FDTD algorithm

void updateEz(Field& Ez, const Field& Hx, const Field& Hy, float dt) {

#pragma omp parallel for collapse(2)

for (int y = 1; y < SIZE\_Y - 1; ++y) {

for (int x = 1; x < SIZE\_X - 1; ++x) {

int idx = y \* SIZE\_X + x;

Ez.data[idx] = Ez.data[idx] + dt \* (Hy.data[idx] - Hy.data[idx - 1] - Hx.data[idx] + Hx.data[idx - SIZE\_X]);

}

}

}

// Function to update Hx field using FDTD algorithm

void updateHx(Field& Hx, const Field& Ez, float dt) {

#pragma omp parallel for collapse(2)

for (int y = 0; y < SIZE\_Y - 1; ++y) {

for (int x = 0; x < SIZE\_X - 1; ++x) {

int idx = y \* SIZE\_X + x;

Hx.data[idx] = Hx.data[idx] + dt \* (Ez.data[idx + SIZE\_X] - Ez.data[idx]);

}

}

}

// Function to update Hy field using FDTD algorithm

void updateHy(Field& Hy, const Field& Ez, float dt) {

#pragma omp parallel for collapse(2)

for (int y = 0; y < SIZE\_Y - 1; ++y) {

for (int x = 0; x < SIZE\_X - 1; ++x) {

int idx = y \* SIZE\_X + x;

Hy.data[idx] = Hy.data[idx] + dt \* (Ez.data[idx + 1] - Ez.data[idx]);

}

}

}

int main() {

// Initialize fields

Field Ez, Hx, Hy;

initializeFields(Ez, Hx, Hy);

// Time parameters

float dt = 0.1; // Time step

// Number of time steps

int numSteps = 100;

// Main simulation loop

for (int t = 0; t < numSteps; ++t) {

// Update fields

updateEz(Ez, Hx, Hy, dt);

updateHx(Hx, Ez, dt);

updateHy(Hy, Ez, dt);

// Additional computations or output if needed

}

return 0;

}