Karin Metzgar

ASTE 546

HW 9

7/27/2024

Following the instruction in the HW assignment I have made the following updates to the cex-original code:

Updated the simulation domain:

/\*initialize domain\*/

World world(31,81,71);

world.setExtents({0,-0.4,0},{0.3,0.4,0.7});

world.setTime(2e-7,10000);

Changed addSphere and inSphere to addBox and inBox

void World::addBox(double3 min, double3 max, double phi\_object){

for (int i=0;i<ni;i++)

for (int j=0;j<nj;j++)

for (int k=0;k<nk;k++)

{

/\*compute node position\*/

double3 x = pos(i,j,k);

if (inBox(x, min, max))

{

object\_id[i][j][k] = 1;

phi[i][j][k] = phi\_object;

}

}

}

bool World::inBox(double3 point, double3 min, double3 max) {

//returns TRUE when in the box

if ( point[0] >= min[0] && point[0] <= max[0] &&

point[1] >= min[1] && point[1] <= max[1] &&

point[2] >= min[2] && point[2] <= max[2]) return true;

else return false;

}

Changed the species to Xenon neutrals and ions:

/\*set up particle species\*/

vector<Species> species;

species.push\_back(Species("Xenon", 131.3\*AMU, 0, 2e9, world));

species.push\_back(Species("Xenon+", 131.3\*AMU, QE, 5e8, world));

Species &neutrals = species[0];

Species &ions = species[1];

Enabled MCC\_CEX collisions:

/\*setup material interactions\*/

vector<unique\_ptr<Interaction>> interactions;

interactions.emplace\_back(new MCC\_CEX(ions,neutrals,world));

Changed the solver to QN and updated the reference values:

/\*initialize potential solver and solve initial potential\*/

PotentialSolver solver(world,SolverType::QN,1000,1e-4);

solver.setReferenceValues(30,2.5,ndi); //30 V, 2.5eV

solver.solve();

Updated Species.advance to add particle reflection on the x-min boundary and updated check for “inBox” instead of “inSphere”:

/\*keep iterate while time remains and the particle is alive\*/

while (part.dt>0 && part.mpw>0) {

//double3 pos\_old = part.pos;

part.pos += part.vel\*part.dt;

//check for symmetry on the X face

if (part.pos[0]<world.getX0()[0]) {

part.pos[0] = -part.pos[0];

part.vel[0] = -part.vel[0];

}

/\*did this particle leave the domain?\*/

if (!world.inBounds(part.pos) || world.inBox(part.pos))

{

part.mpw = 0; //kill the particle

}

Updated constant sigma in Collisions.cpp

/\*contant cross-section for the demo\*/

double evalSigma(double rel\_g)

{

return 5e-18;

}

… update it in the MCC\_CEX code

/\*evaluate cross-section \*/

//double sigma = 1e-16;

double sigma = 5e-18;

Modify the Sources:

/\*setup injection sources\*/

const double nda = 2e20; //neutral density

const double ndi = 1e17; //mean ion density --- changed in step 5

vector<unique\_ptr<Source>> sources;

sources.emplace\_back(new WarmBeamSource(neutrals,world,{0,0,0.1}, 0.02, 0.1e-6 ,500, 300)); //neutral source

sources.emplace\_back(new WarmBeamSource(ions,world,{0,0,0.1}, 0.02, 0.8e-6 , 15000, 1e4)); //ions source

the cex-original code ran in 25 minutes:

A screen shot of a computer

Description automatically generated

But at the rate my modified version is going it would be like 25 hours for it to finish, each time step is almost a minute. I don’t understand which of my changes is causing this…

I am going to change one/two things at a time and run again to see what is slowing it down.

* First I changed just the simulation bounds etc. That didn’t slow things down too much, maybe a little but also gave this message: A black screen with white text

  Description automatically generated
* I updated addBox and inBox and changed the code a bit from above:

/\*set objects\*/

double phi\_sphere = -100; //set default

if (argc>1)

phi\_sphere = atof(args[1]); //convert argument to float

cout<<"Sphere potential: "<<phi\_sphere<<" V"<<endl;

world.addBox({-0.1,-0.1,0},{0.1,0.1,0.1},phi\_sphere);

world.addInlet();

void World::addBox(const double3 &x0, const double3 &xm, double phi\_sphere){

for (int i=0;i<ni;i++)

for (int j=0;j<nj;j++)

for (int k=0;k<nk;k++)

{

/\*compute node position\*/

double3 x = pos(i,j,k);

if (inBox(x))

{

object\_id[i][j][k] = 1;

phi[i][j][k] = phi\_sphere;

}

}

}

bool World::inBox(const double3 &x) {

//returns TRUE when in the box

if ( x[0] >= x0[0] && x[0] <= xm[0] &&

x[1] >= x0[1] && x[1] <= xm[1] &&

x[2] >= x0[2] && x[2] <= xm[2]) return true;

else return false;

}

Okay ran that, the front part where PCG took a while to run but the individual time steps seem to be progressing okay.

After like 20 minutes ish the progression seems to have slowed down significantly I’m only at ts = 535 now

A computer screen with white text

Description automatically generated

It turns out I didn’t update the function call in Main.cpp to be “addBox” so it’s still a sphere, shown below.

A red and blue square with a hole in the center

Description automatically generated

I changed the number of time steps to 500 so I can get a time for how long it takes, and I updated the Main.cpp call to addBox.

A black screen with white text

Description automatically generated

Was slightly faster, only 2 minutes, the second time running

A screen shot of a computer

Description automatically generated

Well. Now it is a block but it is clearly not the desired size in the simulation, looks like I must be doing something wrong where I assign object\_id.

A grey and red cube with numbers

Description automatically generated

I defined the bounds as given?

A computer screen shot of a program code

Description automatically generated

I think the problem is that I was using x0 and xm which are saved as the corners of the simulation domain, I added variables box\_x0 and box\_xm and used those instead.

Also, I updated Species.cpp advance function to call inBox and not inSphere.

A computer screen with white text

Description automatically generated

That didn’t work so great so I am going to change the solver also.

Well that didn’t work either, I see no object at all when I apply the threshold filter.

void World::addBox(const double3 &box\_x0, const double3 &box\_xm, double phi\_sphere){

for (int i=0;i<ni;i++)

for (int j=0;j<nj;j++)

for (int k=0;k<nk;k++)

{

/\*compute node position\*/

double3 x = pos(i,j,k);

if (inBox(x))

{

object\_id[i][j][k] = 1;

phi[i][j][k] = phi\_sphere;

}

}

}

bool World::inBox(const double3 &x) {

//returns TRUE when in the box

if ( x[0] >= box\_x0[0] && x[0] <= box\_xm[0] &&

x[1] >= box\_x0[1] && x[1] <= box\_xm[1] &&

x[2] >= box\_x0[2] && x[2] <= box\_xm[2]) return true;

else return false;

}

Got the box definition working:

Code:

World.h

//sugarcube a prism with corners x0 and xm

void addBox(const double3 &box\_x0, const double3 &box\_xm, double phi\_object);

bool inBox(const double3 &x);

define initial box coordinates

double3 \_box\_x0 {0,0,0}; //box corner

double3 \_box\_xm {0,0,0}; //box corner

World.cpp

void World::addBox(const double3 &box\_x0, const double3 &box\_xm, double phi\_sphere){

\_box\_x0 = box\_x0;

\_box\_xm = box\_xm;

for (int i=0;i<ni;i++)

for (int j=0;j<nj;j++)

for (int k=0;k<nk;k++)

{

/\*compute node position\*/

double3 x = pos(i,j,k);

if (inBox(x))

{

object\_id[i][j][k] = 1;

phi[i][j][k] = phi\_sphere;

}

}

}

bool World::inBox(const double3 &x) {

//returns TRUE when in the box

if ( x[0] >= \_box\_x0[0] && x[0] <= \_box\_xm[0] &&

x[1] >= \_box\_x0[1] && x[1] <= \_box\_xm[1] &&

x[2] >= \_box\_x0[2] && x[2] <= \_box\_xm[2]) return true;

else return false;

}

A white box in a room

Description automatically generated

Now I will modify the sources and add reflection.

----

Well, I found the culprit for the code being so slow its essentially stuck… apparently it’s the species definition..

Modify the source call from Main.cpp:

/\*setup injection sources\*/

const double nda = 2e20; //neutral density

const double ndi = 1e17; //mean ion density --- changed in step 5

vector<unique\_ptr<Source>> sources;

sources.emplace\_back(new WarmBeamSource(neutrals,world,{0,0,0.1}, 0.02, 0.1e-6 ,500, 300)); //neutral source

sources.emplace\_back(new WarmBeamSource(ions,world,{0,0,0.1}, 0.02, 0.8e-6 , 15000, 1e4)); //ions source

Modify Source.h file:

//simple monoenergetic source

class WarmBeamSource: public Source {

public:

WarmBeamSource(Species &species, World &world, const double3 &x0, double rad, double mdot, double v\_drift, double T) :

sp{species}, world{world}, x0{x0}, rad{rad}, mdot{mdot}, v\_drift{v\_drift}, T{T} {}

//generates particles

void sample();

protected:

Species &sp; //reference to the injected species

World &world; //reference to world

double v\_drift; //mean drift velocity

double den; //injection density

double T; //temperature

double rad; //radius of circle

double mdot; //mass flow rate of thruster

double3 x0; //center of source (circle)

};

Modify Source.cpp file:

//samples particles with finite thermal and drift velocity

void WarmBeamSource::sample()

{

double3 dh = world.getDh();

double3 x0 = world.getX0();

//area of the XY plane, A=Lx\*Ly

//double Lx = dh[0]\*(world.ni-1);

//double Ly = dh[1]\*(world.nj-1);

//double A = Lx\*Ly;

//compute number of real particles to generate: (#/s) = n\*v\*A; # = (#/s)\*dt

//double num\_real = den\*v\_drift\*A\*world.getDt();

//compute number of real particles to generate (#/s)

//mass flow rate = mdot, divided by atomic mass (kg/particle) == particles/sec

double num\_real = mdot / (sp.mass);

//number of simulation particles

int num\_sim = (int)(num\_real/sp.mpw0+rnd());

//inject particles

for (int i=0;i<num\_sim;i++)

{

//double3 pos {x0[0]+rnd()\*Lx, x0[1]+rnd()\*Ly, x0[2]};

double3 pos;

double angle;

double radius;

double x, y;

angle = rnd() \* 360;

radius = sqrt(rnd() \* (0.02 \* 0.02));

x = radius \* cos(angle\*(M\_PI/180));

y = radius \* sin(angle\*(M\_PI/180));

pos = {x,y, x0[2]};

double3 vel = sp.sampleIsotropicVel(T);

vel[2] += v\_drift; //add drift component

sp.addParticle(pos,vel);

}

}

So with everything except the species changed this is what it looks like .. I’m thinking there may be something wrong with my source since it doesn’t look like it’s coming from the single source like it should be?

A graph of a red square with a blue bar and a white square

Description automatically generated with medium confidence