GALAXY\_SYSTEM

Compute the motion of objects based on the gravitational forces. **Add a tail to each object**.

The counter mNumFrames represents the number of frames in the simulation.

Key usage

‘1’: one large object

‘2’: two large objects

‘3’: three large objects

‘r’, ‘R’: reset

There should be 100 objects (including large objects) at the beginning.

**Major tasks**

Generate the objects

Compute the motion of objects

Compute force

Compute acceleration

Compute velocity

Compute position

Object *i* attributes: position ***p****i*, acceleration ***a***i, velocity ***v***i, radius *r*i, mass *m*i.

Consider two objects *i* and *j* as follows.

Force exerts on object i

**f**ij = -(G mi mj / *dij*2) **n**ij,

where **n**ij is a unit vector whose direction is from *j* to *i,* and *dij* is the distance between the two objects.

**a**i = **f**i /mi, based on Newton’s second law

where **f**i is the total force exerting on object i.

Update velocity: ***v***i 🡨 ***v***i + **a**i Δt, where Δt is the time step.

Update position: ***p****i* 🡨***p****i +****v***iΔt

Update tails of all the objects

===

The followings are for your reference. You do not need to use the same set of variables. You can delete all the variables and use your own variables.

GALAXY\_SYSTEM::GALAXY\_SYSTEM( ) {

mTimeStep = 0.0025;

mMaxV = 100.0;

mNumOfObjs = 100;

mNumLargeObjs = 2;

mMinR = 0.5;

mMaxR = 3.0;

mSpaceSize = 300.0;

mG = 5.0;

generateObjects( );

}

/\*

Generate objects.

mX, mY: x-coordinates and y-coordinates of objects

range [ -mSpaceSize/2, mSpaceSize/2]

mR : radii of objects; range [mMinR, mMaxR]

mMass: masses of objects; mMass[i] = mR[i]\*mR[i]

mVx, mVy: x-components and y-components of velocities of objects

range [-100.0, 100.0]

mAlive: alive flags of objects

For large objects:

mR[i] = mMaxR\*2.0;

mMass[i] = mMaxR\*mMaxR\*1000;

The number of large objects = mNumLargeObjs

If mNumLargeObjs =1, set the position of the large object to origin.

\*/

void GALAXY\_SYSTEM::generateObjects( )

{

generate the positions (mX and mY) of objects

generate the radii (mR) of objects

generate the velocities (mVx and mVy) of objects

set alive flags of objects

}

Make sure that the position component is in [-halfSpaceSize, halfSpaceSize];

where halfSpaceSize = mSpaceSize/2;

/\*

Handle the key events.

Return true if the key event is handled.

case ‘1’: mNumLargeObjs = 1;

case '2': mNumLargeObjs = 2;

case '3': mNumLargeObjs = 3;

etc…

\*/

bool GALAXY\_SYSTEM::handleKeyPressedEvent( int key )

/\*

Generate the objects.

Call generateObjects( ).

\*/

void GALAXY\_SYSTEM::reset( ) {

generateObjects( );

}

/\*

Show the system title.

Show the key usage.

\*/

void GALAXY\_SYSTEM::askForInput( )

{

cout << "GALAXY\_SYSTEM::askForInput" << endl;

cout << "Key usage:" << endl;

cout << "1: one large object" << endl; // generate the objects. Only one large object

cout << "2: two large object" << endl; // generate the objects. Only two large objects

cout << "3: three large object" << endl; // generate the objects. Only three large objects

cout << "r: reset" << endl; // reset the positions of the objects

}

/\*

Return the number of objects

\*/

int GALAXY\_SYSTEM::getNumOfObjs( ) const

{

return mNumOfObjs;

}

/\*

\* Return the number of active objects

\*

\*/

int GALAXY\_SYSTEM::getNumberOfActiveObjects() const

{

return mActiveObjectNumber;

}

/\*

\* Compute the number of active objects

\*/

void GALAXY\_SYSTEM::computeNumberOfActiveObjects()

{

mActiveObjectNumber = 0;

for (int i = 0; i < mNumOfObjs; ++i) {

if (!mAlive[i]) continue;

++mActiveObjectNumber;

}

}

/\*

Get the object information based on the object index.

(x,y) = position

r = radius

Return the alive flag.

\*/

bool GALAXY\_SYSTEM::getObjInfo( int objIndex, double &x, double &y, double &r ) const

/\*

**Merge the objects if they overlap with each other.** Modify the velocities of the objects after merging based on the conservation of momentum. Set the alive flags of the objects accordingly.

Pseudo-code

For each pair of the objects

if they do not overlap, continue

If they overlap

do

turn off the alive flag of the object with smaller radius

compute the new velocity of the larger object

\*/

void GALAXY\_SYSTEM::mergeObjects( )

{

for ( int i = 0; i < mNumOfObjs; ++i ) {

if ( !mAlive[i] ) continue;

for ( int j = i+1; j < mNumOfObjs; ++j ) {

if ( !mAlive[j] ) continue;

check if two spheres overlap

if they overlap,

set the smaller one to be ‘not alive’

new velocity = ( mi \* vi + mj\*vj) / (total mass) ; momentum conservation

Hint:

Assume object i is larger. Then, we have

mivi\_mjvj\_X = mVx[i]\*mMass[i]+mVx[j]\*mMass[j];

mivi\_mjvj\_Y = mVy[i]\*mMass[i]+mVy[j]\*mMass[j];

mMass[i] += mMass[j]; // total mass

mVx[i] = mivi\_mjvj\_X / mMass[i];

mVy[i] = mivi\_mjvj\_Y / mMass[i];

} // end loop j

} // end loop i

}

/\*

**Update the position of the objects**

Steps:

1. compute the total force exerting on each object. Handle each pair of objects

2. compute the velocity of each object

3. compute the position of each object

Constraints:

The component of a velocity must be inside [-mMaxV, mMaxV].

The component of a position must be inside [-halfSpaceSize, halfSpaceSize].

Warp the position if it's outside of the range.

Consider a position (x,y).

For example, if x > halfSpaceSize, set x = -halfSpaceSize;

\*/

void GALAXY\_SYSTEM::update( )

{

mergeObjects();

computeForcesOfObjects();

updateVelocitiesOfObjects();

updatePositionsOfObjectsAndTails();

computeNumberOfActiveObjects();

++mNumFrames;

}

mergeObjects( ); // merge the overlapping objects first

**Compute forces as follows**:

For each pair of objects (i,j) and they are alive

double d = their distance

//compute a unit vector which indicates the force direction

double nx = dx/d; //normalize along the x-axis

double ny = dy/d; //normalize along the y-axis

where dx = mX[i] – mX[j], and dy = mY[i] – mY[j]

double mij = mi\*mj;

mFx[i] -= mij\*mG\*nx/d2; //notice the negative sign

mFy[i] -= mij\*mG\*ny/d2; //notice the negative sign

mFx[j] += mij\*mG\*nx/d2; //notice the positive sign

mFy[j] += mij\*mG\*ny/d2; //notice the positive sign

**Compute velocity as follows:**

For each object i do

double m = mMass[i];

mVx[i] = mVx[i] + mFx[i]/m\*mTimeStep;

mVy[i] = mVy[i] + mFy[i]/m\*mTimeStep;

Make sure the velocity component in [-mMaxV, mMaxV];

**Update the positions of all the object as follows**:

For each object i {

mX[i] = mX[i] + mVx[i]\*mTimeStep;

mY[i] = mY[i] + mVy[i]\*mTimeStep;

}

**Add a tail to each object. Use the data structure to store the tail of each object:**

vector<TAIL\_PLANET> mTails

Each element of mTails is an object of TAIL\_PLANET. Each element of an object of TAIL\_PLANET is a pair of coordinates (x,y) which represents a point in the two-dimensional space.

Thus, the meaning of

mTails[planet\_index].points[sample\_point\_index]

is that it is the pair of coordinates of a sample point with index sample\_point\_index in a tail of an object with index planet\_index.

A new point of an object is added to the tail of the object per n frames, where n = skipFrames. Set skipFrames = 10.

Insert

if (skipFrames == 0 || mNumFrames%skipFrames == 0) {

mTails[i].add\_to\_front(mX[i], mY[i]);

}

at an appropriate position. This block adds a point of an object to the front of the tail of the object, i.e., the first point of the tail.

Implement the following to functions:

1. int getTail\_NumSamplePoints( int planet\_index ) const;
2. COORD\_2D getTail\_SamplePointCoords(

int planet\_index,

int sample\_point\_index ) const

Implement void add\_to\_front(double x, double y) in TAIL\_PLANET.