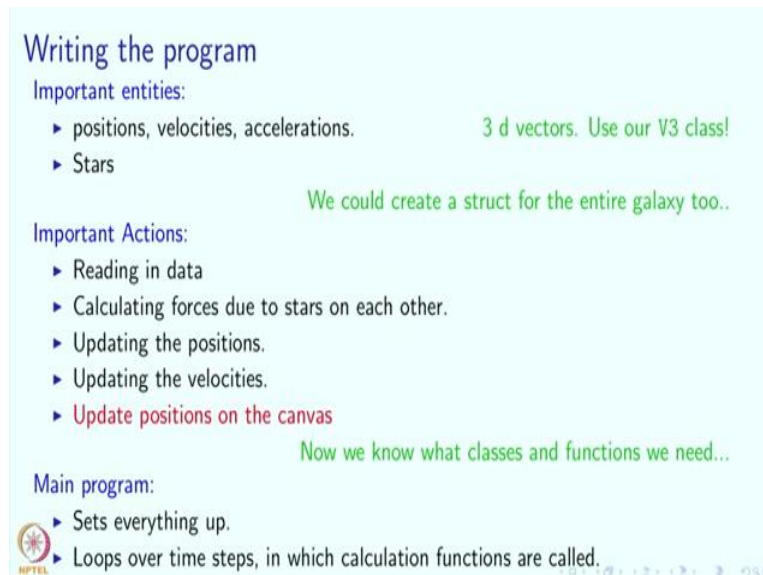


**An Introduction to Programming through C++**  
**Professor Abhiram G. Ranade**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Bombay**  
**Lecture No. 27**  
**Cosmological Simulation**  
**Part 3**  
**The Program**

Welcome back, in the last segment we discussed the second-order method for the cosmological problem and also an algorithm based on it. So before the program let us make a few remarks. So as always, in order to write the program we should first try and enumerate the important entities.

(Refer Slide Time: 0:36)



**Writing the program**

**Important entities:**


- ▶ positions, velocities, accelerations. 3 d vectors. Use our V3 class!
- ▶ Stars We could create a struct for the entire galaxy too..

**Important Actions:**

- ▶ Reading in data
- ▶ Calculating forces due to stars on each other.
- ▶ Updating the positions.
- ▶ Updating the velocities.
- ▶ Update positions on the canvas Now we know what classes and functions we need...

**Main program:**

- ▶ Sets everything up.
- ▶ Loops over time steps, in which calculation functions are called.



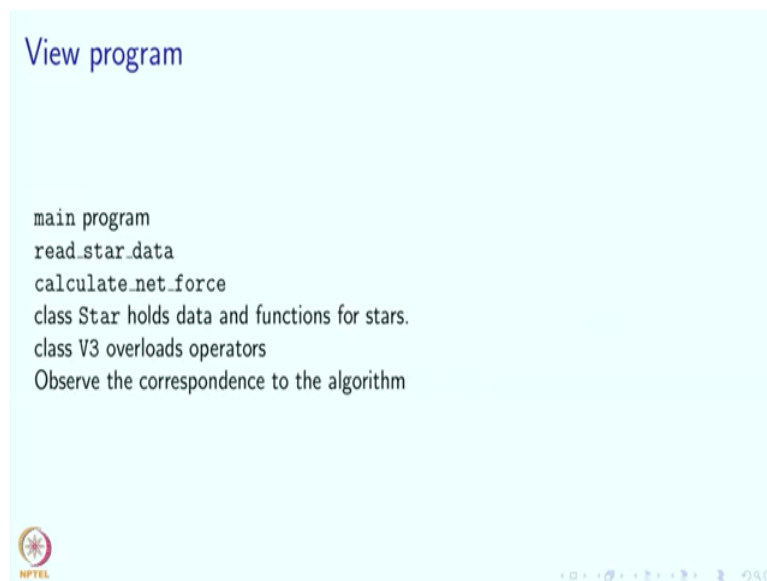
Now in this program, this program is all about positions, velocities, accelerations and these are not entirely simple quantities. So they are actually 3 dimensional vectors and since they are important entities we should use a class for them and luckily we did develop such a class which was our V3 class which we developed some time earlier. So that is exactly what we will use. So instead of, instead of thinking of it as an array of 3 elements or something like that we will use our V3 class. And if you remember our V3 class was developed so that we can add 2 vectors together by writing plus or if we could multiply a vector by a scalar just by writing star. So that will come in very handy and that will make our code very easy to understand and everywhere we will not have to write down loop looping over the 3 components.

We can just say  $U + V$  if we want to add vectors  $U$  and vectors vector  $V$ . Then there were the other the major entities from a different point of view I guess are the stars. Well the Stars possess the position, velocities and acceleration but the stars are important entities. So clearly we should have a struct. Now we could create a struct for the entire galaxy as well, so that struct would just contain the array of stars. But here we have chosen not to do this we will discuss this a little bit later when you have seen the entire program. But what we have said so far, is that the entity is sort of the physical entities are the stars and inside the stars will have  $V3$  entities which will represent the positions, velocities and accelerations of the stars. What are the important actions that the program needs to take? Well first of all of course the program has to read in data then it has to calculate the force due to stars on each other. So this will involve all stars, then it has to update the position.

So what do we need in order to update the position? Well we should change the velocity of sorry change the position of each star individually and for that we need the velocity of that star. So this is a bit of a local operation if we know the velocities of the stars. And similarly we have to update the velocities, so this is also a local operation because if we know the accelerations then the new velocity of a star depends on its old velocity and its acceleration. But these will be important actions that we are going to perform and yeah and we want to show all this on our canvas. So we should actually move things around as well, so that is another important action. So this suggests that suggests what what functions we should have. Because we said at some point that if something is an important action, then we should make it into a function or a member function.

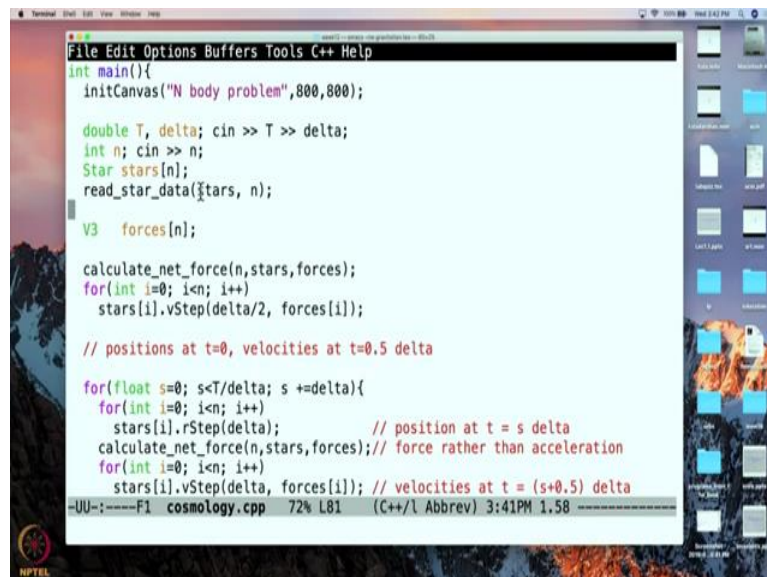
So we will almost do that some of these if something happens to be really tiny then maybe we will just write it directly rather than making it into a function. But yes, in general if something is important we should sort of make it have a name in our program and the way to do that is by making it a function or a member function and then of course there is the main program, if you have to see what happens in the main program? Well the main program sets everything up and it also loops over the time steps. So in that sense the main program is kind of a controller a kind of a choreographer if you will. So it sort of it sort of sends messages to everyone saying, oh now we do the next step of the iteration now, now that it is over, do the next step and next step and so on. So that is what our program is going to be, basically it is going to follow this outline.

(Refer Slide Time: 5:16)



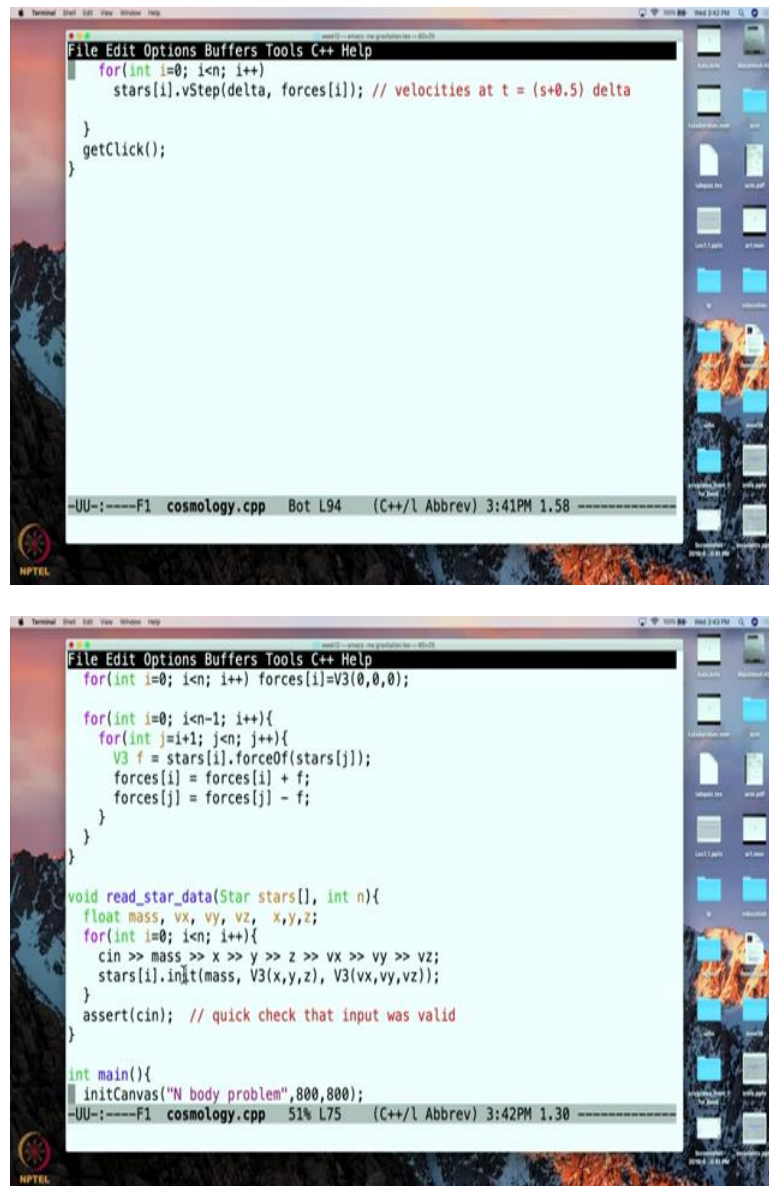
So we, so now I am going to show you the program I am not going to so, so I am going to show you the program and I will begin with the main program. I will talk about the function which reads the data then the function that calculates the forces. I will show you the star class, the class V3 and while doing this, I would like you to observe that really the whole thing corresponds to the algorithm that we have discussed. So let me get out of this.

(Refer Slide Time: 5:48)



So here is our main program, so we begin by creating the canvas on which everything is going to be shown. And then we read in the duration of the simulation, the step size, and the number of stars and then we have a function which will read in the star related information.

(Refer Slide Time: 6:14)



The image consists of two screenshots of a C++ code editor, likely Xcode, showing the implementation of a physics simulation. The top screenshot shows a loop that iterates over a star array and calls a function to update velocities. The bottom screenshot shows the same code with additional force calculations and a main function that initializes the simulation.

```
File Edit Options Buffers Tools C++ Help
for(int i=0; i<n; i++){
    stars[i].vStep(delta, forces[i]); // velocities at t = (s+0.5) delta
}
getClick();
}

--UU--:-----F1 cosmology.cpp Bot L94 (C++/l Abbrev) 3:41PM 1.58
```

```
File Edit Options Buffers Tools C++ Help
for(int i=0; i<n; i++) forces[i]=V3(0,0,0);

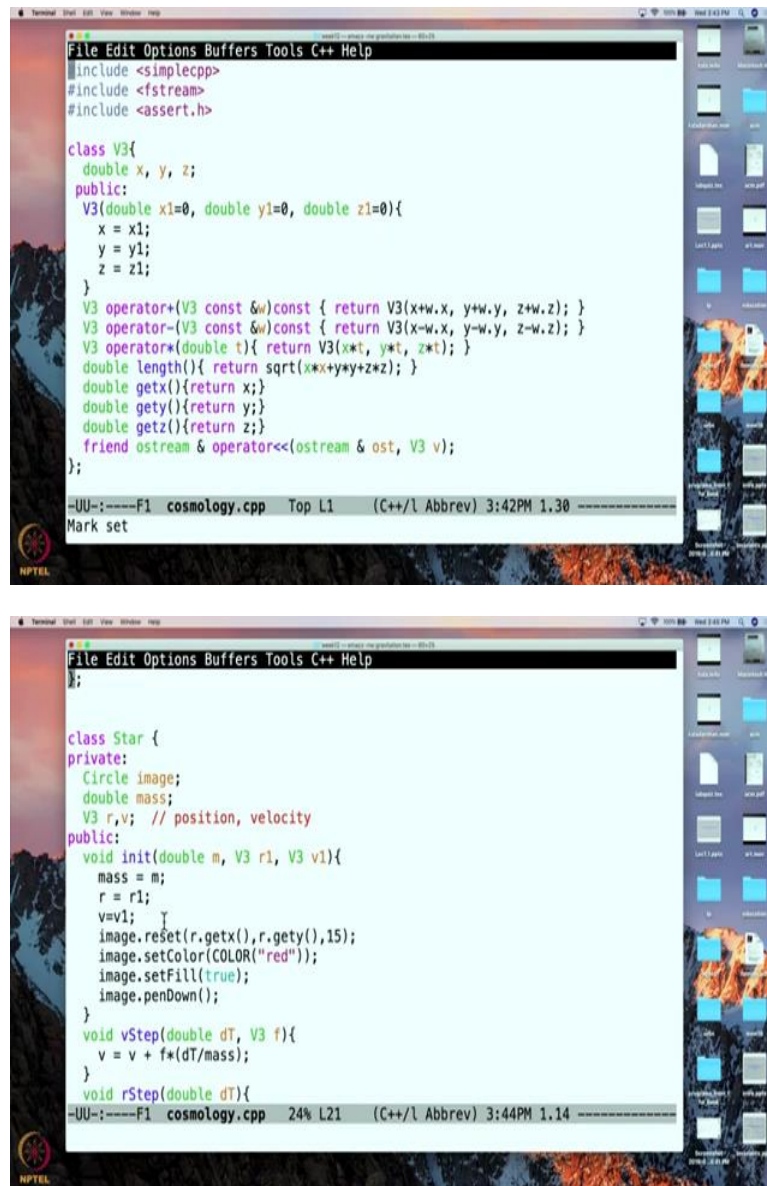
for(int i=0; i<n-1; i++){
    for(int j=i+1; j<n; j++){
        V3 f = stars[i].forceOf(stars[j]);
        forces[i] = forces[i] + f;
        forces[j] = forces[j] - f;
    }
}

void read_star_data(Star stars[], int n){
    float mass, vx, vy, vz, x,y,z;
    for(int i=0; i<n; i++){
        cin >> mass >> x >> y >> z >> vx >> vy >> vz;
        stars[i].init(mass, V3(x,y,z), V3(vx,vy,vz));
    }
    assert(cin); // quick check that input was valid
}

int main(){
    initCanvas("N body problem",800,800);
    --UU--:-----F1 cosmology.cpp 51% L75 (C++/l Abbrev) 3:42PM 1.30
```

So this is the function, so it does not do a whole lot of thing. It will read in the values, the mass X, Y and Z, the positions and the velocities. So these are the initial things that it is going to read and we are going to have a star array, so this is it is going to be read into this star array and we are going to initialize each element of that star array using these using these values. So these 3 values we are going to convert into a position vector by calling this constructor if you remember this is the constructor for V3. So this is giving the position vector, this is giving the velocity vector and this is the mass. So it is going to in it, it is going to initialize each star.

(Refer Slide Time: 7:10)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>
#include <fstream>
#include <assert.h>

class V3{
    double x, y, z;
public:
    V3(double x1=0, double y1=0, double z1=0){
        x = x1;
        y = y1;
        z = z1;
    }
    V3 operator+(V3 const &w) const { return V3(x+w.x, y+w.y, z+w.z); }
    V3 operator-(V3 const &w) const { return V3(x-w.x, y-w.y, z-w.z); }
    V3 operator*(double t){ return V3(x*t, y*t, z*t); }
    double length(){ return sqrt(x*x+y*y+z*z); }
    double getx(){return x;}
    double gety(){return y;}
    double getz(){return z;}
    friend ostream & operator<<(ostream & ost, V3 v);
};

class Star {
private:
    Circle image;
    double mass;
    V3 r,v; // position, velocity
public:
    void init(double m, V3 r1, V3 v1){
        mass = m;
        r = r1;
        v=v1;
        image.reset(r.getx(),r.gety(),15);
        image.setColor(COLOR("red"));
        image.setFill(true);
        image.pendown();
    }
    void vStep(double dT, V3 f){
        v = v + f*(dT/mass);
    }
    void rStep(double dT){
};
```

The first screenshot shows the V3 class implementation with vector operations and a friend function for output. The second screenshot shows the Star class with private data members (Circle image, double mass, V3 r,v) and public methods (init, vStep, rStep) for initializing and updating the star's position and velocity.

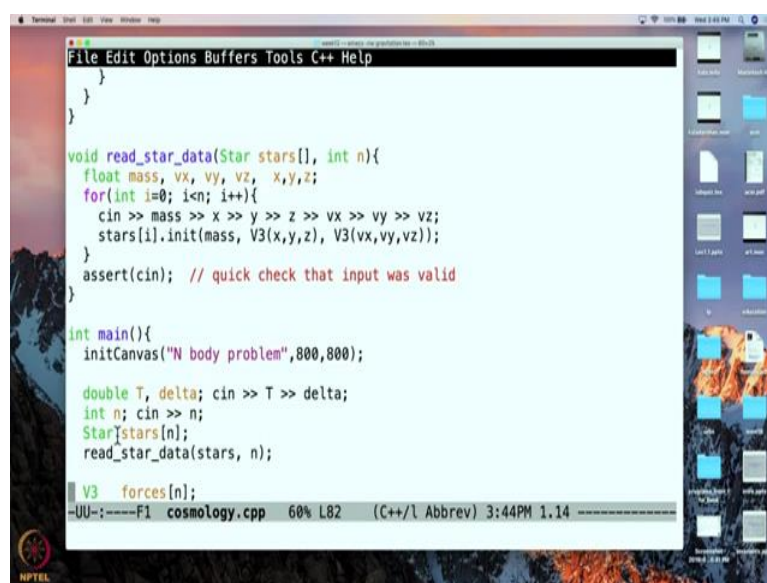
So let us quickly take a look at that because there is something in some thing that happens in addition over there. So here is that initialization. So M, V3 r1, V3 v1 so these are the positions and velocities. So they are going to be put inside the corresponding members. So V and r are the velocity and position of this star, so these are these are the data members and mass is also there so M is going to go into mass. But we want everything to be seen on the screen as well so in this construct this is not a constructor this is kind of a reset like things so it is the initialization thing. So in this initialization we will actually make this circle appear on our screen by resetting it and where should it go, well it should go at position x and y where x and y are the positions of the star. So the way we have done it is that we have used we are we are pretending that our stars are moving in sort of this pixel space. So after all this is only for



a demonstration and so we are not doing any kind of a change of coordinates over here, so this is all pixel space.

So pixel coordinate x coordinate we get, pixel y-coordinate we get and over there we are going to show the star. So that star is going to appear as a circle and we are going to make that circle be red and filled. And we will have a pen for it which will go down and it will go down because we want the orbit to be seen as the circle as the star moves. So yeah so this is initializing the data part as well as initializing what happens on the screen.

(Refer Slide Time: 9:08)



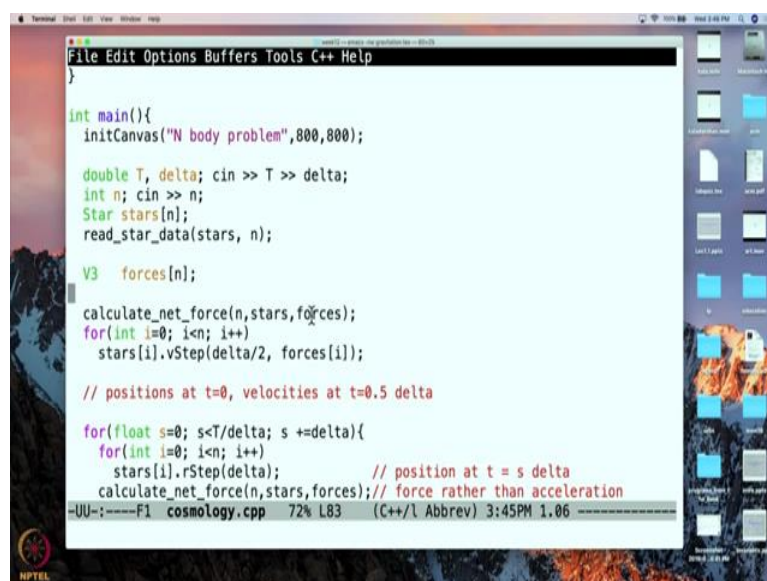
```
File Edit Options Buffers Tools C++ Help
}
}
}

void read_star_data(Star stars[], int n){
    float mass, vx, vy, vz, x,y,z;
    for(int i=0; i<n; i++){
        cin >> mass >> x >> y >> z >> vx >> vy >> vz;
        stars[i].init(mass, V3(x,y,z), V3(vx,vy,vz));
    }
    assert(cin); // quick check that input was valid
}

int main(){
    initCanvas("N body problem",800,800);

    double T, delta; cin >> T >> delta;
    int n; cin >> n;
    Star stars[n];
    read_star_data(stars, n);

    V3 forces[n];
    UU-:-----F1 cosmology.cpp 60% L82 (C++/l Abbrev) 3:44PM 1.14
```



```
File Edit Options Buffers Tools C++ Help
}

int main(){
    initCanvas("N body problem",800,800);

    double T, delta; cin >> T >> delta;
    int n; cin >> n;
    Star stars[n];
    read_star_data(stars, n);

    V3 forces[n];

    calculate_net_force(n,stars,forces);
    for(int i=0; i<n; i++)
        stars[i].vStep(delta/2, forces[i]);

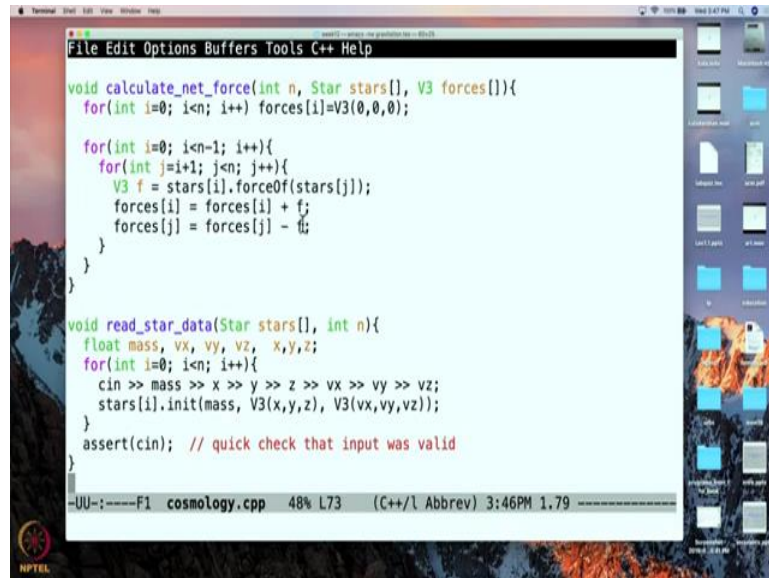
    // positions at t=0, velocities at t=0.5 delta

    for(float s=0; s<T/delta; s +=delta){
        for(int i=0; i<n; i++){
            stars[i].rStep(delta); // position at t = s delta
            calculate_net_force(n,stars,forces); // force rather than acceleration
        }
    }
    UU-:-----F1 cosmology.cpp 72% L83 (C++/l Abbrev) 3:45PM 1.06
```

So we read the data that is that is over here and next we are going to calculate the forces. So how do we calculate the forces? Well for that we need the information about all stars and that

is we pass it the array stars and we also need to calculate the result in some array called forces so that is what is going to happen over here. So in in the array forces we will calculate the forces. So how does that happen?

(Refer Slide Time: 9:41)

A screenshot of a C++ code editor window titled 'cosmology.cpp'. The code defines two functions: 'calculate\_net\_force' and 'read\_star\_data'. The 'calculate\_net\_force' function takes an integer 'n', an array of 'Star' objects 'stars', and a vector 'V3' 'forces'. It initializes the 'forces' array to zero and then iterates over each star 'i', calculating the net force on it by summing the forces from all other stars 'j'. The 'read\_star\_data' function reads input for each star's mass and position (x, y, z) and initializes the 'stars' array. The status bar at the bottom shows 'F1 cosmology.cpp 48% L73 (C++/l Abbrev) 3:46PM 1.79'.

```
void calculate_net_force(int n, Star stars[], V3 forces[]){
    for(int i=0; i<n; i++) forces[i]=V3(0,0,0);

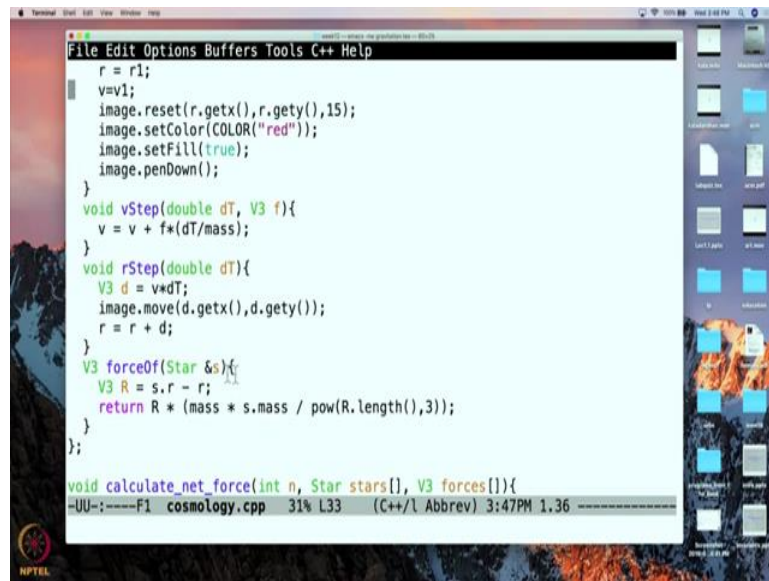
    for(int i=0; i<n-1; i++){
        for(int j=i+1; j<n; j++){
            V3 f = stars[i].forceOf(stars[j]);
            forces[i] = forces[i] + f;
            forces[j] = forces[j] - f;
        }
    }
}

void read_star_data(Star stars[], int n){
    float mass, vx, vy, vz, x,y,z;
    for(int i=0; i<n; i++){
        cin >> mass >> x >> y >> z >> vx >> vy >> vz;
        stars[i].init(mass, V3(x,y,z), V3(vx,vy,vz));
    }
    assert(cin); // quick check that input was valid
}
```

So it is as you might expect, so we are going to go through every pair of stars and we are going to calculate the mutual force. So how is the mutual force calculated? Well we have this function this member function on a star object. So what this is saying is that tell me the force due to star, due to star i on star j. So and tell it to me as a vector or rather tell me the force on star i due to star j. And that force we are going to add to the total force on i and we know that by symmetry, or by a Newton's third law exactly the equal force will be exerted on star j and so we are going to subtract from it and notice that these are vectors remember that these are vectors and this is also a vector and the result of this calculation is also a vector.

So this calculation will produce vectors we are we have overloaded the plus and minus operators we will see that and you already saw that in an earlier lecture. So that we can just directly add the incremental contribution due to star j on star i and on star star i do to star j. So this is the incremental contribution and exactly where does Newton's law of gravitation come in? Well let me show you so this is where it comes in.

(Refer Slide Time: 11:12)

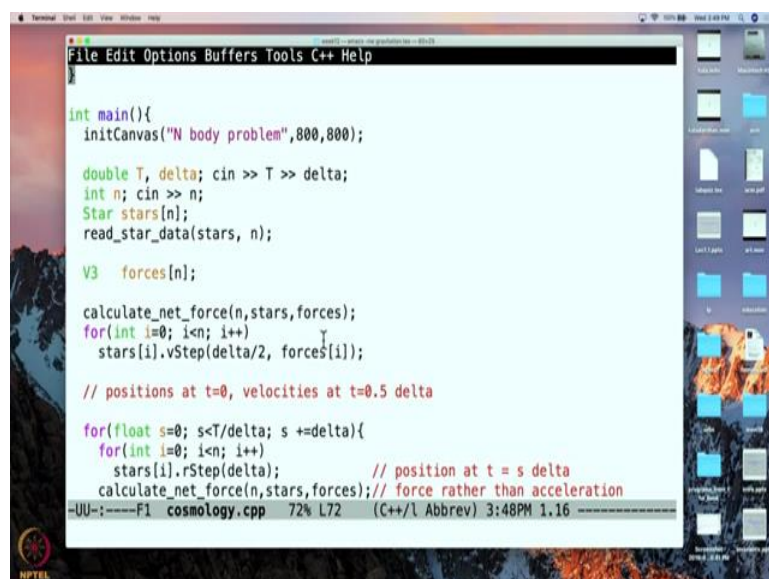


```
File Edit Options Buffers Tools C++ Help
r = r1;
v=v1;
image.reset(r.getx(),r.gety(),15);
image.setColor(COLOR("red"));
image.setFill(true);
image.pendown();
}
void vStep(double dT, V3 f){
    v = v + f*(dT/mass);
}
void rStep(double dT){
    V3 d = v*dT;
    image.move(d.getx(),d.gety());
    r = r + d;
}
V3 forceOf(Star &s){
    V3 R = s.r - r;
    return R * (mass * s.mass / pow(R.length(),3));
}
};

void calculate_net_force(int n, Star stars[], V3 forces[]){
-uu-:-----F1 cosmology.cpp 31% L33 (C++/l Abbrev) 3:47PM 1.36
```

So this  $r$  is the distance but distance expressed as a the distance expressed as a vector. So this is  $r_j$ . So the other minus  $r_i$  if you look at our formula and that is the vector distance and that is, so our formula says it is the vector distance times the gravitational constant which we are taking as one here and times the mass times, the two masses divided by the power of the distance to the cube because we already have a distance term over here in the vector in the vector part. So this is again exactly like what we had in the slides. So we have calculated the net force.

(Refer Slide Time: 12:06)



```
File Edit Options Buffers Tools C++ Help

int main(){
    initCanvas("N body problem",800,800);

    double T, delta; cin >> T >> delta;
    int n; cin >> n;
    Star stars[n];
    read_star_data(stars, n);

    V3 forces[n];

    calculate_net_force(n,stars,forces);
    for(int i=0; i<n; i++){
        stars[i].vStep(delta/2, forces[i]);

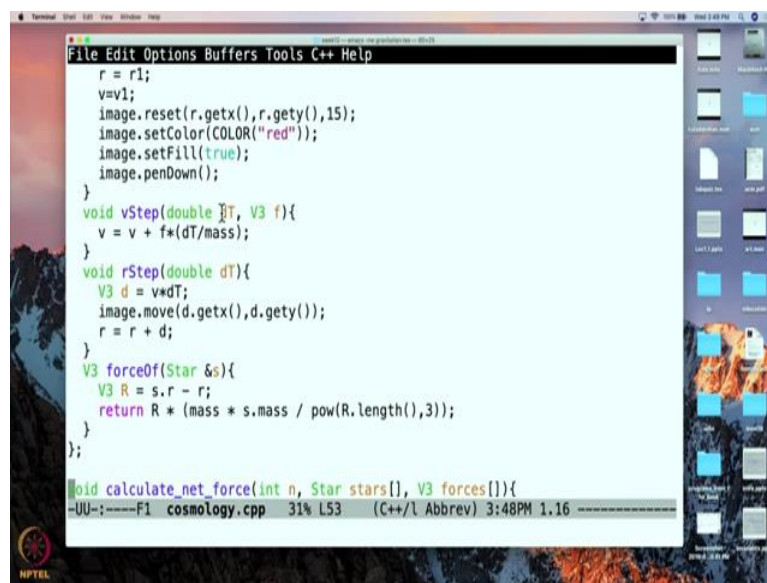
        // positions at t=0, velocities at t=0.5 delta

    for(float s=0; s<T/delta; s +=delta){
        for(int i=0; i<n; i++){
            stars[i].rStep(delta); // position at t = s delta
            calculate_net_force(n,stars,forces); // force rather than acceleration
-uu-:-----F1 cosmology.cpp 72% L72 (C++/l Abbrev) 3:48PM 1.16
```



So going back to the main program, so once we have the net force why did we calculate the net force? Well if you remember, we need to have the velocities at time step  $\Delta t$  by 2. So if you remember when we started things of we wanted the positions at time step 0 and velocities at time step  $\Delta t/2$ . So we got that by a first-order update, so what was that first-order update? Well we are going to take the original velocity of each star and to that we are going to add acceleration times  $\Delta t/2$ . But here we have forces, so in this member function will convert the forces into accelerations and add them.

(Refer Slide Time: 13:02)



```

File Edit Options Buffers Tools C++ Help
r = r1;
v=v1;
image.reset(r.getx(),r.gety(),15);
image.setColor(COLOR("red"));
image.setFill(true);
image.penDown();
}
void vStep(double dt, V3 f){
    v = v + f*(dt/mass);
}
void rStep(double dt){
    V3 d = v*dt;
    image.move(d.getx(),d.gety());
    r = r + d;
}
V3 forceOf(Star &s){
    V3 R = s.r - r;
    return R * (mass * s.mass / pow(R.length(),3));
}
};

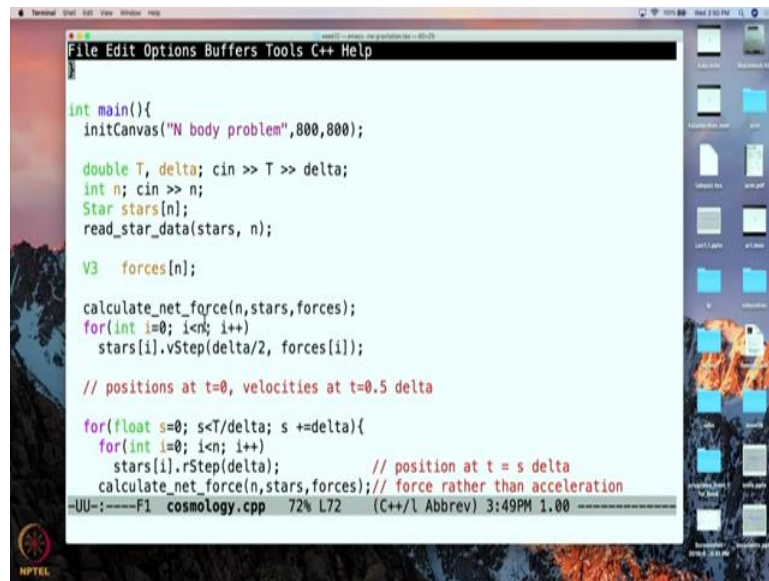
void calculate_net_force(int n, Star stars[], V3 forces){
}

```

cosmology.cpp 31% L53 (C++/l Abbrev) 3:48PM 1.16

So let us see that, so there is this velocity step that I am going to show you. So V step says that you had an original velocity to it you are going to add the force divided by mass which is the acceleration and whatever time displacement you want. How much how by how much ever you want to advance.

(Refer Slide Time: 13:27)



```
File Edit Options Buffers Tools C++ Help

int main(){
    initCanvas("N body problem",800,800);

    double T, delta; cin >> T >> delta;
    int n; cin >> n;
    Star stars[n];
    read_star_data(stars, n);

    V3 forces[n];

    calculate_net_force(n,stars,forces);
    for(int i=0; i<n; i++){
        stars[i].vStep(delta/2, forces[i]);

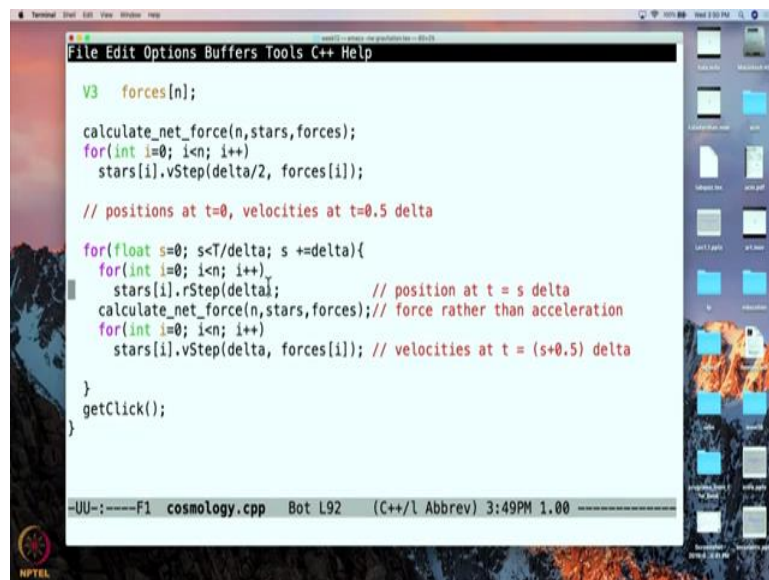
        // positions at t=0, velocities at t=0.5 delta

    for(float s=0; s<T/delta; s +=delta){
        for(int i=0; i<n; i++){
            stars[i].rStep(delta);           // position at t = s delta
            calculate_net_force(n,stars,forces); // force rather than acceleration
        }
    }

    //UU-:-----F1 cosmology.cpp 72% L72 (C++/l Abbrev) 3:49PM 1.00
```

So in this particular call we are advancing everything by delta by 2 because we started with all the data at 0 and we wanted the velocities to have been calculated for delta by 2. So this gives us at the end of this we will have velocities at  $t$  equal to  $0.5 \delta$  and positions will be at  $T$  equal to 0 because that is what we started of and we have not modified the positions.

(Refer Slide Time: 13:53)



```
File Edit Options Buffers Tools C++ Help

V3 forces[n];

calculate_net_force(n,stars,forces);
for(int i=0; i<n; i++){
    stars[i].vStep(delta/2, forces[i]);

    // positions at t=0, velocities at t=0.5 delta

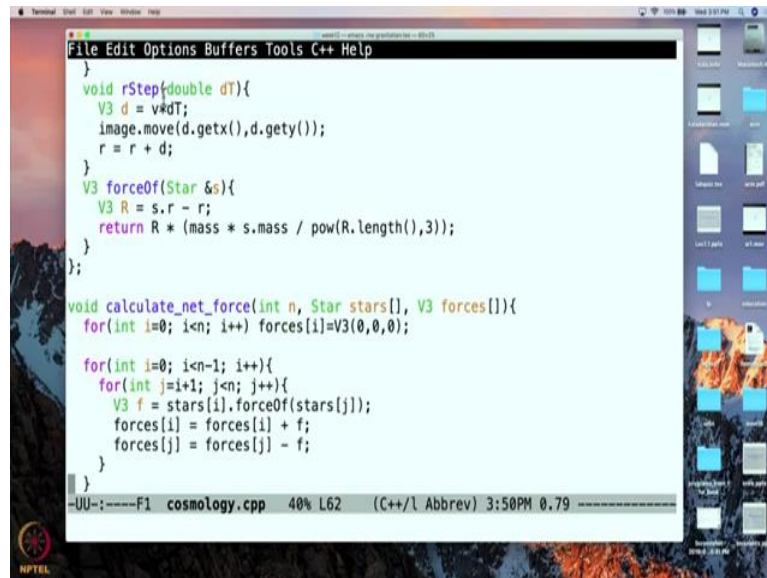
for(float s=0; s<T/delta; s +=delta){
    for(int i=0; i<n; i++){
        stars[i].rStep(delta);           // position at t = s delta
        calculate_net_force(n,stars,forces); // force rather than acceleration
        for(int i=0; i<n; i++){
            stars[i].vStep(delta, forces[i]); // velocities at t = (s+0.5) delta
        }
    }
    getClick();
}

//UU-:-----F1 cosmology.cpp Bot L92 (C++/l Abbrev) 3:49PM 1.00
```

Then there is the main loop, so how long does the loop run? Well exactly as before it goes from 0 to  $t/\delta$  and in the increment of  $\delta$ . So here we are going to calculate, we are going to update the position. So this is this update is what we are calling as `rStep` and this

time we are going to advance the position by the full delta. So therefore the argument that we give to this is delta, so let us see this rStep.

(Refer Slide Time: 14:26)



```
File Edit Options Buffers Tools C++ Help
}
void rStep(double dT){
    V3 d = v*dT;
    image.move(d.getx(),d.gety());
    r = r + d;
}
V3 forceOf(Star &s){
    V3 R = s.r - r;
    return R * (mass * s.mass / pow(R.length(),3));
}
};

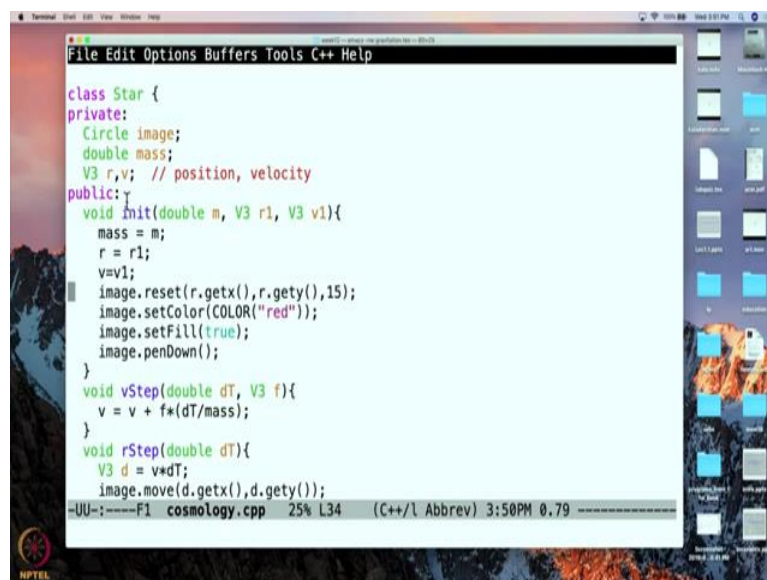
void calculate_net_force(int n, Star stars[], V3 forces[]){
    for(int i=0; i<n; i++){ forces[i]=V3(0,0,0);

    for(int i=0; i<n-1; i++){
        for(int j=i+1; j<n; j++){
            V3 f = stars[i].forceOf(stars[j]);
            forces[i] = forces[i] + f;
            forces[j] = forces[j] - f;
        }
    }
}

--UU--F1 cosmology.cpp 40% L62 (C++/l Abbrev) 3:50PM 0.79
```

So we are passing the duration and that duration is going to be multiplied by V well what is V? So V is just the velocity, velocity of this star. So remember the velocities are kept along with the Stars so see this as apart of the star the velocities are kept there itself.

(Refer Slide Time: 14:41)



```
File Edit Options Buffers Tools C++ Help

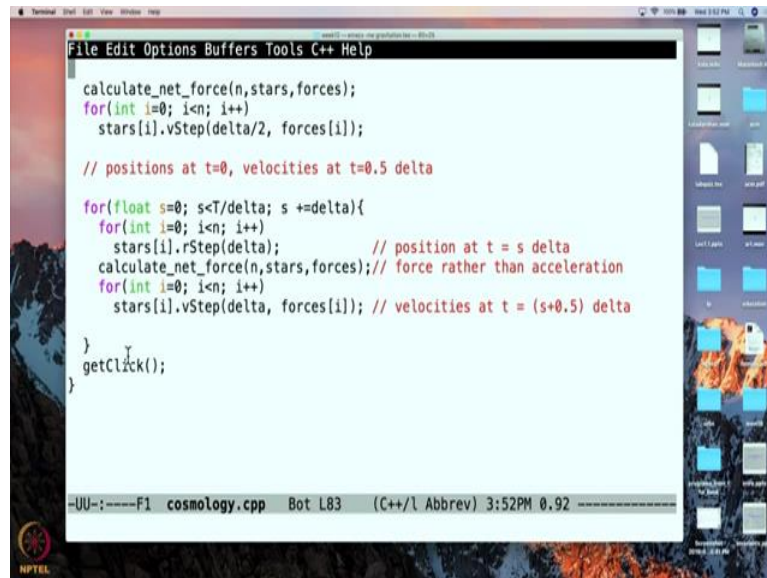
class Star {
private:
    Circle image;
    double mass;
    V3 r,v; // position, velocity
public:
    void init(double m, V3 r1, V3 v1){
        mass = m;
        r = r1;
        v=v1;
        image.reset(r.getx(),r.gety(),15);
        image.setColor(COLOR("red"));
        image.setFill(true);
        image.penDown();
    }
    void vStep(double dT, V3 f){
        v = v + f*(dT/mass);
    }
    void rStep(double dT){
        V3 d = v*dT;
        image.move(d.getx(),d.gety());
    }
};

--UU--F1 cosmology.cpp 25% L34 (C++/l Abbrev) 3:50PM 0.79
```

So in the vStep this velocity that is there with the star we are taking and we are getting the acceleration and that is got by F divided by mass and then we are multiplying it by the

duration. In this case, delta was passed, so this is just saying a times delta.  $V$  equal to  $V$  plus a times delta that is what this is.

(Refer Slide Time: 15:20)



```
File Edit Options Buffers Tools C++ Help

calculate_net_force(n,stars,forces);
for(int i=0; i<n; i++)
    stars[i].vStep(delta/2, forces[i]);

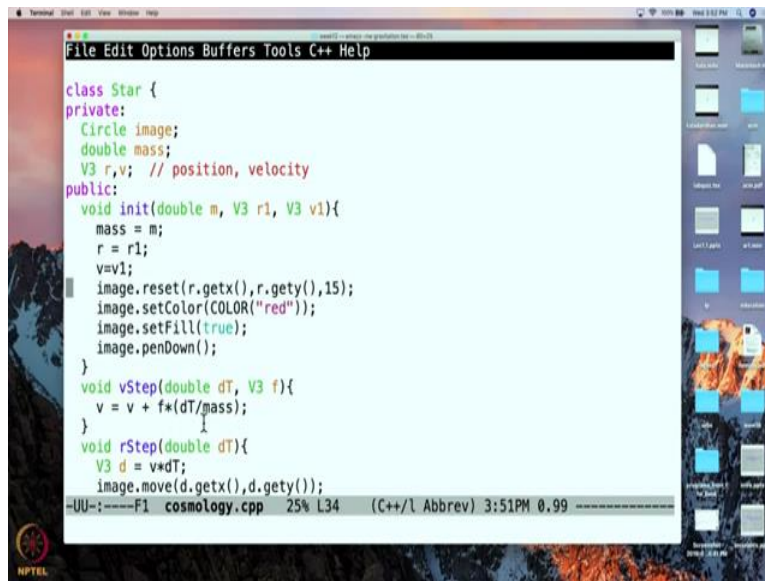
// positions at t=0, velocities at t=0.5 delta

for(float s=0; s<T/delta; s +=delta){
    for(int i=0; i<n; i++)
        stars[i].rStep(delta); // position at t = s delta
    calculate_net_force(n,stars,forces); // force rather than acceleration
    for(int i=0; i<n; i++)
        stars[i].vStep(delta, forces[i]); // velocities at t = (s+0.5) delta
    }
    getClick();
}

--UU--:-----F1 cosmology.cpp Bot L83 (C++/l Abbrev) 3:52PM 0.92
```

So we have come to this point, we have updated the positions the positions and we have updated the velocities and yeah so before we have update the velocities we need to of course calculated the forces. So it is it is exactly the same force calculation function and we updated the velocities and then this iteration has to happen as many times as needed. So if you want to advance time by  $T$  it has to happen  $T/\text{delta}$  times. So that is it, so at the end of it we are just putting a get click over here just, so that though at the end the display does not vanish right away.

(Refer Slide Time: 16:07)



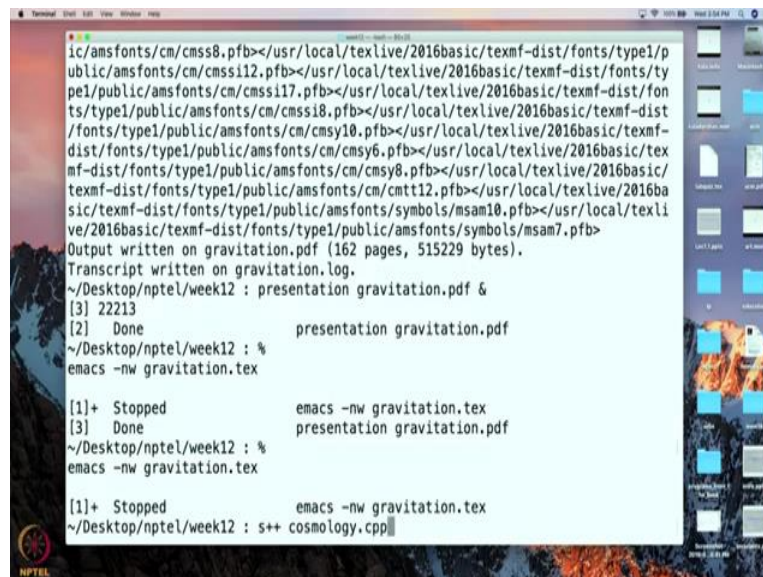
```
File Edit Options Buffers Tools C++ Help

class Star {
private:
    Circle image;
    double mass;
    V3 r,v; // position, velocity
public:
    void init(double m, V3 r1, V3 v1){
        mass = m;
        r = r1;
        v=v1;
        image.reset(r.getx(),r.gety(),15);
        image.setColor(COLOR("red"));
        image.setFill(true);
        image.penDown();
    }
    void vStep(double dT, V3 f){
        v = v + f*(dT/mass);
    }
    void rStep(double dT){
        V3 d = v*dT;
        image.move(d.getx(),d.gety());
    }
}

-UU-:-----F1 cosmology.cpp 25% L34 (C++/l Abbrev) 3:51PM 0.99
```

So the vector class is pretty much like what we had before except that we have added a minus because we had you have to subtract two vectors, so therefore we have added this. And we have so ignore this line this line is not really needed. This is somewhat complicated this is these this is described in the book and this is a way to allow the class to be printed but do not worry about it I should I should really remove this line, so that is the vector class.

(Refer Slide Time: 16:41)



```
ic/amsfonts/cm/cmss8.pfb></usr/local/texlive/2016basic/texmf-dist/fonts/type1/p
ublic/amsfonts/cm/cmss12.pfb></usr/local/texlive/2016basic/texmf-dist/fonts/ty
pe1/public/amsfonts/cm/cmss17.pfb></usr/local/texlive/2016basic/texmf-dist/fon
ts/type1/public/amsfonts/cm/cmss18.pfb></usr/local/texlive/2016basic/texmf-dist
/fonts/type1/public/amsfonts/cm/cmss10.pfb></usr/local/texlive/2016basic/texmf-
dist/fonts/type1/public/amsfonts/cm/cmssy6.pfb></usr/local/texlive/2016basic/tex
mf-dist/fonts/type1/public/amsfonts/cm/cmssy8.pfb></usr/local/texlive/2016basic/
texmf-dist/fonts/type1/public/amsfonts/cm/cmssy12.pfb></usr/local/texlive/2016ba
sic/texmf-dist/fonts/type1/public/amsfonts/symbols/msam10.pfb></usr/local/texli
ve/2016basic/texmf-dist/fonts/type1/public/amsfonts/symbols/msam7.pfb>
Output written on gravitation.pdf (162 pages, 515229 bytes).
Transcript written on gravitation.log.
~/Desktop/npTEL/week12 : presentation gravitation.pdf &
[3] 22213
[2] Done presentation gravitation.pdf
~/Desktop/npTEL/week12 : %
emacs -nw gravitation.tex

[1]+ Stopped emacs -nw gravitation.tex
[3] Done presentation gravitation.pdf
~/Desktop/npTEL/week12 : %
emacs -nw gravitation.tex

[1]+ Stopped emacs -nw gravitation.tex
~/Desktop/npTEL/week12 : s++ cosmology.cpp
```



```
ts/type1/public/amsfonts/cm/cmssi8.pfb></usr/local/texlive/2016basic/texmf-dist
/fonts/type1/public/amsfonts/cm/cmssy10.pfb></usr/local/texlive/2016basic/texmf-
dist/fonts/type1/public/amsfonts/cm/cmssy6.pfb></usr/local/texlive/2016basic/tex
mf-dist/fonts/type1/public/amsfonts/cm/cmssy8.pfb></usr/local/texlive/2016basic/
texmf-dist/fonts/type1/public/amsfonts/cm/cmmt12.pfb></usr/local/texlive/2016ba
sic/texmf-dist/fonts/type1/public/amsfonts/symbols/msam10.pfb></usr/local/texli
ve/2016basic/texmf-dist/fonts/type1/public/amsfonts/symbols/msam7.pfb>
Output written on gravitation.pdf (162 pages, 515229 bytes).
Transcript written on gravitation.log.
~/Desktop/npTEL/week12 : presentation gravitation.pdf &
[3] 22213
[2] Done presentation gravitation.pdf
~/Desktop/npTEL/week12 : %
emacs -nw gravitation.tex

[1]+ Stopped emacs -nw gravitation.tex
[3] Done presentation gravitation.pdf
~/Desktop/npTEL/week12 : %
emacs -nw gravitation.tex

[1]+ Stopped emacs -nw gravitation.tex
~/Desktop/npTEL/week12 : s++ cosmology.cpp
+ g++ cosmology.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhi
ram/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week12 : more sunEarth.txt
```

```
ve/2016basic/texmf-dist/fonts/type1/public/amsfonts/symbols/msam7.pfb>
Output written on gravitation.pdf (162 pages, 515229 bytes).
Transcript written on gravitation.log.
~/Desktop/npTEL/week12 : presentation gravitation.pdf &
[3] 22213
[2] Done presentation gravitation.pdf
~/Desktop/npTEL/week12 : %
emacs -nw gravitation.tex

[1]+ Stopped emacs -nw gravitation.tex
[3] Done presentation gravitation.pdf
~/Desktop/npTEL/week12 : %
emacs -nw gravitation.tex

[1]+ Stopped emacs -nw gravitation.tex
~/Desktop/npTEL/week12 : s++ cosmology.cpp
+ g++ cosmology.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhi
ram/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week12 : more sunEarth.txt
30000
10
2
400 400 400 0 0 -.004 0
1 600 400 0 0 1.6 0
~/Desktop/npTEL/week12 : ./a.out <sunEarth.txt
```

```
File Edit Options Buffers Tools C++ Help

class Star {
private:
    Circle image;
    double mass;
    V3 r,v; // position, velocity
public:
    void init(double m, V3 r1, V3 v1){
        mass = m;
        r = r1;
        v=v1;
        image.reset(r.getx(),r.gety(),15);
        image.setColor(COLOR("red"));
        image.setFill(true);
        image.penDown();
    }
    void vStep(double dT, V3 f){
        v = v + f*(dT/mass);
    }
    void rStep(double dT){
        V3 d = v*dT;
        image.move(d.getx(),d.gety());
    }
}
~UU-:----F1 cosmology.cpp 25% L24 (C++/l Abbrev) 3:52PM 0.92
```

```
File Edit Options Buffers Tools C++ Help
V3 d = v*dt;
image.move(d.getx(),d.gety());
r = r + d;
}
V3 forceOf(Star &s){
V3 R = s.r - r;
return R * (mass * s.mass / pow(R.length(),3));
}
};

void calculate_net_force(int n, Star stars[], V3 forces[]){
for(int i=0; i<n; i++){ forces[i]=V3(0,0,0);

for(int i=0; i<n-1; i++){
for(int j=i+1; j<n; j++){
V3 f = stars[i].forceOf(stars[j]);
forces[i] = forces[i] + f;
forces[j] = forces[j] - f;
}
}
}

-UU-:----F1 cosmology.cpp 41% L43 (C++/l Abbrev) 3:53PM 0.79
```

```
File Edit Options Buffers Tools C++ Help

void read_star_data(Star stars[], int n){
float mass, vx, vy, vz, x,y,z;
for(int i=0; i<n; i++){
cin >> mass >> x >> y >> z >> vx >> vy >> vz;
stars[i].init(mass, V3(x,y,z), V3(vx,vy,vz));
}
assert(cin); // quick check that input was valid
}

int main(){
initCanvas("N body problem",800,800);

double T, delta; cin >> T >> delta;
int n; cin >> n;
Star stars[n];
read_star_data(stars, n);

V3 forces[n];

calculate_net_force(n,stars,forces);

-UU-:----F1 cosmology.cpp 61% L63 (C++/l Abbrev) 3:53PM 0.79
```

```
File Edit Options Buffers Tools C++ Help

calculate_net_force(n,stars,forces);
for(int i=0; i<n; i++)
stars[i].vStep(delta/2, forces[i]);

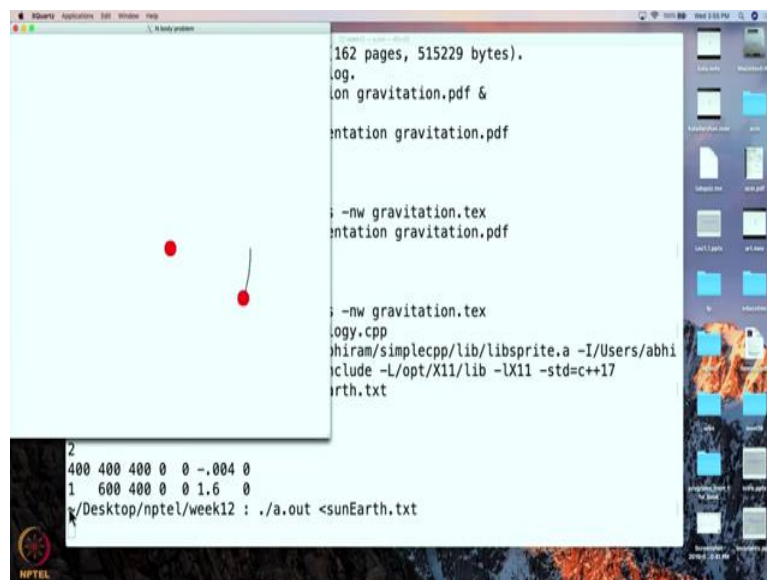
// positions at t=0, velocities at t=0.5 delta

for(float s=0; s<T/delta; s +=delta){
for(int i=0; i<n; i++){
stars[i].rStep(delta); // position at t = s delta
calculate_net_force(n,stars,forces); // force rather than acceleration
for(int i=0; i<n; i++)
stars[i].vStep(delta, forces[i]); // velocities at t = (s+0.5) delta
}
}
getClick();
}

-UU-:----F1 cosmology.cpp Bot L83 (C++/l Abbrev) 3:53PM 0.79
```

And so now you can see that we have the vector class we have the star class and we have these two functions and we have the main program, so let us compile it. And we are supposed to type in all the data about the Stars. So let me show you two pieces of data that I have so one piece is the model of a Sun and the earth. So let me type that out first, so what has happened over here is that I am asking time to go up to 30,000 and my step size is 10 so this is the mass these three things are the initial position and these three things are the velocities. So one is 400 one has mass 400 times the other, so really you can think of this as the Sun and this as the earth if you like. So let me run this and I want to take this as my input, so I will just redirect, redirect this file as standard input and let us see what happens.

(Refer Slide Time: 17:58)



```

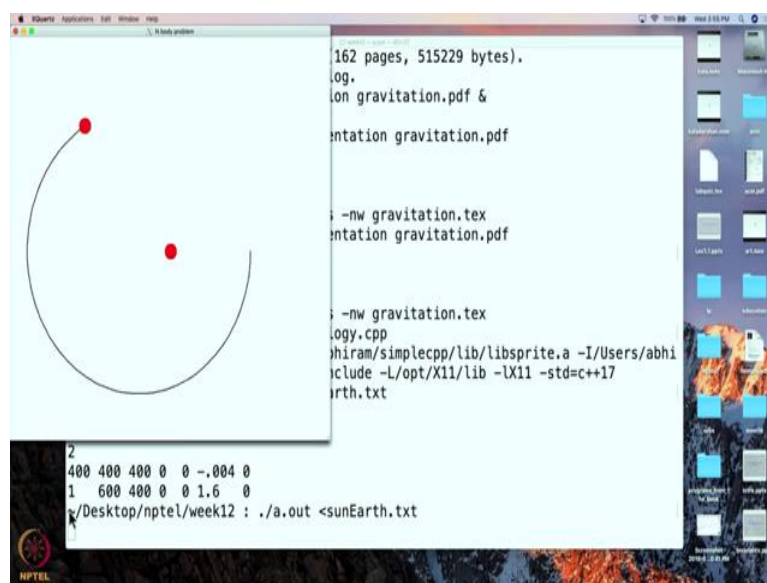
162 pages, 515229 bytes).
log.
on gravitation.pdf &
entation gravitation.pdf

s -nw gravitation.tex
entation gravitation.pdf

s -nw gravitation.tex
logy.cpp
hram/simplecpp/lib/libsprite.a -I/Users/abhi
clude -L/opt/X11/lib -lX11 -std=c++17
rth.txt

2
400 400 400 0 0 -.004 0
1 600 400 0 0 1.6 0
/Desktop/npTEL/week12 : ./a.out <sunEarth.txt

```



```

162 pages, 515229 bytes).
log.
on gravitation.pdf &
entation gravitation.pdf

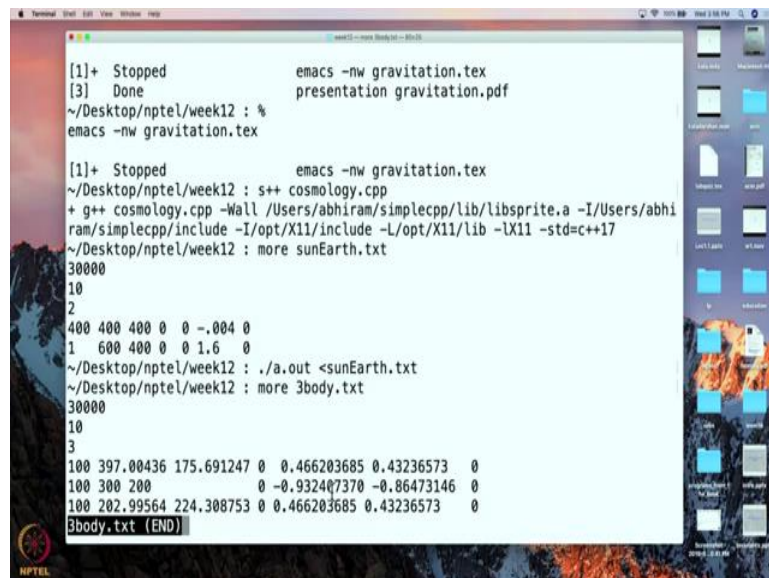
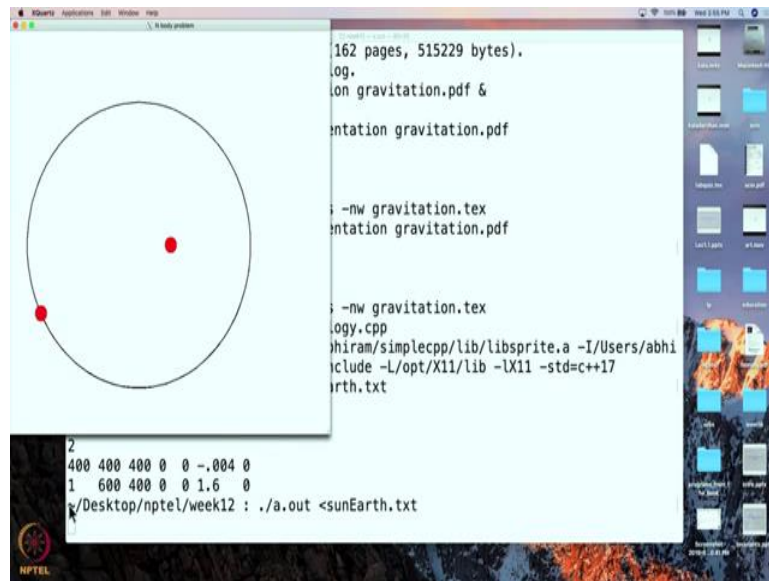
s -nw gravitation.tex
entation gravitation.pdf

s -nw gravitation.tex
logy.cpp
hram/simplecpp/lib/libsprite.a -I/Users/abhi
clude -L/opt/X11/lib -lX11 -std=c++17
rth.txt

2
400 400 400 0 0 -.004 0
1 600 400 0 0 1.6 0
/Desktop/npTEL/week12 : ./a.out <sunEarth.txt

```



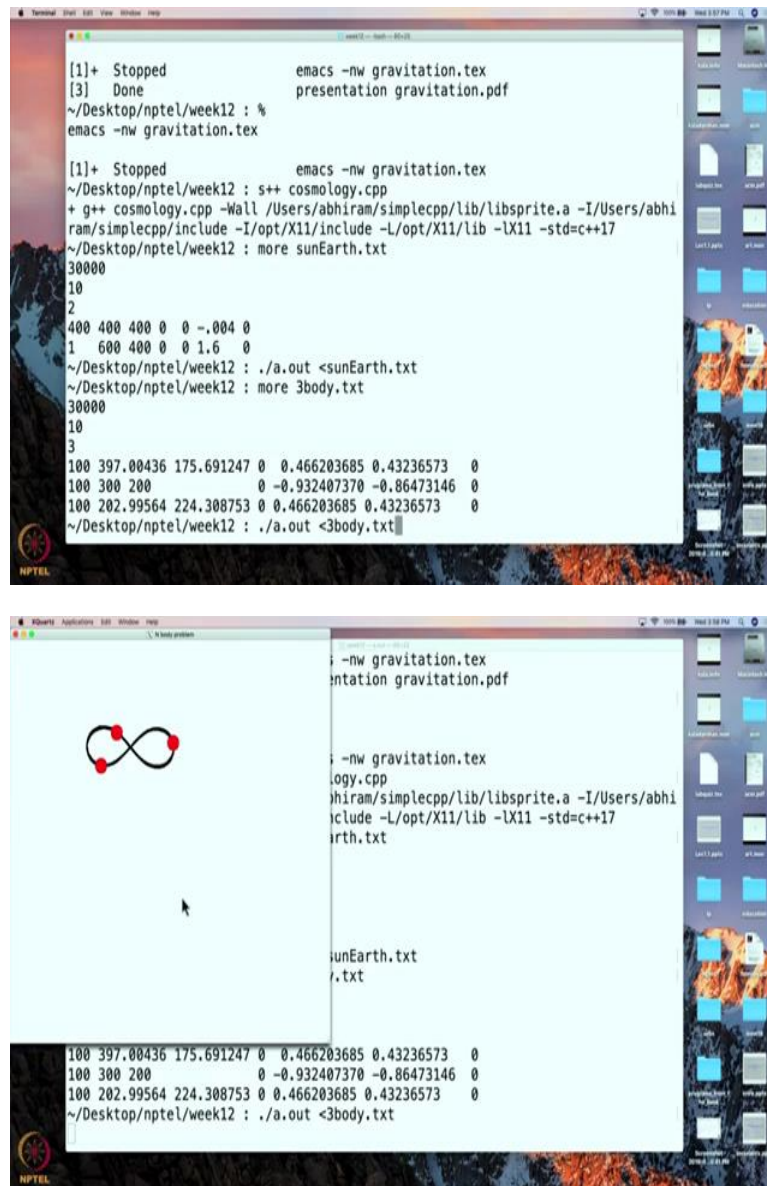


So that is the Sun and this is the earth which is rotating around it and you will see that it traces pretty much the same but the same orbit. It does not really diverge, so despite numerical (calculus) errors in numerical calculations and things like that this is pretty good. So yeah I could have put in more planets in fact in the demo that I had showed that on the very first day of class you have this exact this exact program was used. And over there I had many planets orbiting the Sun. So you can go back and look at that demo as well. But here today I want to show you something else which is quite interesting, so let me show you the data first.

So here here I have 30,000 steps again the 10 being the step size and there being three bodies. So they are all of mass 100, so there all equal and these are the coordinates and velocities. Now that the positions and the velocities have been calculated very cleverly. So this is what

some astronomers did, this is the this has been taken from the internet okay, so these positions and velocities have been taken from the internet and these were calculated by some astronomers and the interesting thing about these positions and velocities are that you will see that the attractive force is such that the bodies will describe a very strange looking orbit. So let us see that,

(Refer Slide Time: 19:52)



The first screenshot shows a terminal window with the following commands and output:

```
[1]+ Stopped          emacs -nw gravitation.tex
[3] Done              presentation gravitation.pdf
~/Desktop/nptel/week12 : %
emacs -nw gravitation.tex

[1]+ Stopped          emacs -nw gravitation.tex
~/Desktop/nptel/week12 : s++ cosmology.cpp
+ g++ cosmology.cpp -Wall -I/Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhi
ram/simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/nptel/week12 : more sunEarth.txt
30000
10
2
400 400 400 0 0 -.004 0
1 600 400 0 0 1.6 0
~/Desktop/nptel/week12 : ./a.out <sunEarth.txt
~/Desktop/nptel/week12 : more 3body.txt
30000
10
3
100 397.00436 175.691247 0 0.466203685 0.43236573 0
100 300 200 0 -0.932407370 -0.86473146 0
100 202.99564 224.308753 0 0.466203685 0.43236573 0
~/Desktop/nptel/week12 : ./a.out <3body.txt
```

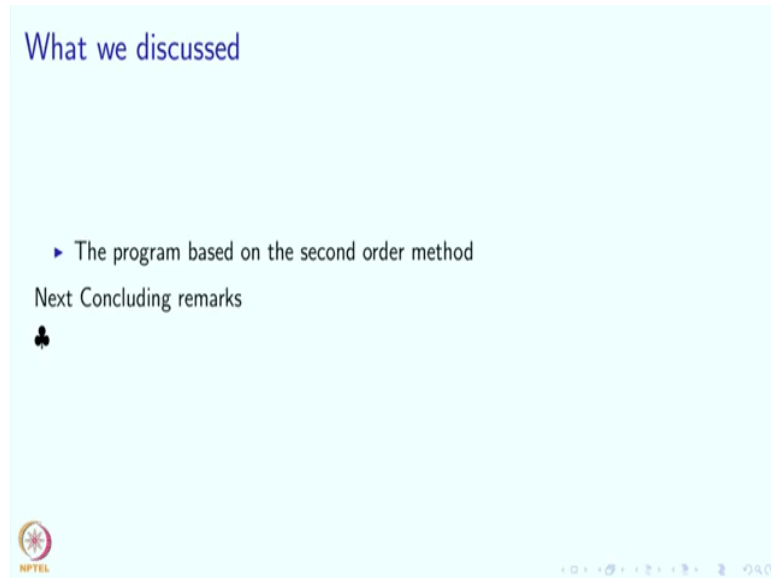
The second screenshot shows a window titled "3 body problem" displaying a 3-body simulation. The window contains a plot of three bodies (represented by red dots) moving in a complex, non-circular orbit. The terminal window is still visible in the background, showing the same commands and output as the first screenshot.

So I am going to give it this 3 body data. So this is what this is what the Stars do as they are moving under gravitational force is it not strange that they should that first of all 3 bodies should trace such an orbit they are sort of following each other. But you can see that they are actually that they may be doing this because of gravitation. Because just think about say this at this point this thing which is in the middle is being dragged by the two others and therefore



they are staying staying in this. So this is a curiosity but it goes to show that our code is actually correct. This is behavior that should be expected for these initial velocities and positions and masses.

(Refer Slide Time: 20:56)



What did we discuss in the last segment? Well we discussed a program based on the second order method and in the next we are going to conclude but we will take a break before that.