Simulation of a Galaxy

3rd Year B.Sc. Project
1999/2000

Andrew Wedgbury
Exam number: 19477

Abstract

The aim of this project was to simulate the temporal evolution of a disk of stars interacting under gravitation using the particle in cell method. The simulation was developed to achieve the greatest accuracy and speed possible in the time allowed and its limitations were discussed. It was then used to obtain results for a variety of galactic models, some were comparable to observations of real galaxies, giving a unique insight into the theory of galactic formation and dynamics.

Table of Contents

Introduction

This section will introduce the subject of galactic simulation by exploring briefly the previous work that has been done in the field. Objectives for this project will be stated along with anticipated results and further plans.

History of galaxy simulation

Using computers to simulate galaxies has been performed since the late 1960s and has got more and more sophisticated as more powerful computers have been available. The earliest simulations, such as those performed by Hohl (see [6]) used integer arithmetic and took a very long time to run due to the computer capabilities at the time. The results obtained, however, proved to be very interesting and valuable to the development of our theories regarding the structure of galaxies. During the 1970s, a lot of progress was made and various models were developed to simulate more and more stars with greater accuracy and speed.

Many people have independently produced simulations that produce models of galaxies that appear very similar to that observed in the sky, however, one problem seems to be that spiral patterns that appear in the simulations rarely last more than a few galactic rotations. We believe that galaxies have been around for enough time for them to complete in excess of 50[1] rotations, so it seems unlikely that these models are accurate considering the fact that a great proportion (over 2/3) of observable galaxies still have a well defined spiral structure.

More progress has been made by Zhang (see [11]), who simulates the gas and dust that makes up the interstellar medium along with the stars to achieve more realistic results. Alongside the development of galaxy simulation techniques there have been lots of theoretical developments. These include the density wave theory, which suggests that the spiral structure rotates as a density wave, i.e. the stars do not rotate at the same speed as the spiral pattern.

There have also been great improvements in the field of astronomy, giving us more information about observed galaxies, and allowing us to see galaxies that are further away. Using spectroscopy, we can observe the red shifts of different parts of galaxies to determine if they are rotating, and at what speed. With the Hubble Space Telescope, we are now able to observe galaxies that lie at incredible distances, so far away that the light from them has travelled for over 85% of the age of the universe to reach us. This gives us the amazing ability to see how galaxies looked back then, this can be compared with the results from our simulations.

[1] **This depends on the age of the universe, which is presently believed to be about 12 Billion years (1.2×10^{10})**

Objectives

Constructing the model

This project will involve constructing a computer simulation of a galaxy that can be used to accurately simulate its temporal evolution for several billion years, and hopefully up to the estimated age of the universe. The main objectives for constructing the model are:

- **Investigate methods used to simulate disks of stars and general n particle systems.**
- **Compare methods of calculating the forces between particles in systems with large numbers of particles.**
- **Construct a computer program using the most appropriate method to perform the simulation.**
- **Test the program by running with a simple system and comparing different methods of calculating the forces.**
- **Obtain an estimate of the errors in the model and how they relate to the model parameters used.**
- **Rigorously test and ascertain the validity of any assumptions that were made to construct the model.**

Using the model

Once the simulation has been tested and is producing results of an acceptable accuracy, it will be used to simulate large numbers of particles interacting in a galactic model. The objectives will be:

- **Investigate initial conditions used in other models of galaxies.**
- **Construct a routine to set up the initial positions and velocities of the particles according to the model.**
- **Run the program and observe the output for various models, investigating transient features and also seeing what happens after long periods of time.**
- **See what models, if any, produce output that is comparable to actual galaxies.**

Further plans

A galactic simulation can be used to investigate a huge variety of different galactic models, if time allows, these extra objectives will be investigated:

- **Further optimising the program to increase speed and accuracy of the simulations.**
- **Construct a simulation of interstellar gas as well as stars, including star formation and evolution within the galaxy.**
- **Extend the model to include satellite galaxies. Many of the galaxies in our local group have small satellite galaxies, including our own. Observe what effect this has on the structure.**

- **Simulate galactic collisions and compare with real observable colliding galaxies.**
- **Simulate clusters of galaxies.**

What I hope to find

It is hoped that the simulation will, for some parameters, produce results that are comparable with observable galaxies. In comparing the simulation results with data from actual galaxies, the following criteria will be used:

- **Long lasting, well-defined spiral structure.**
- **Particles remain bound in the system (i.e. negative total energy), but it is expected that a few may reach escape velocity in the event of a collision.**
- **Exponential decrease in luminosity with distance from galactic centre, this is what is observed from most spiral galaxies. How the luminosity is dependent on the particle density will need to be considered.**
- **Rotation curve that compares with actual galaxy rotation curves, most appear to have higher than expected angular velocities at large radii.**

Project outline

The next section will discuss the construction of the simulation program along with simple galactic models. The rest of the project will be concerned with expanding and improving on these models and testing them using the simulation. Results will be compared throughout with actual data from astronomical observations, including pictures of galaxies from the Hubble Space Telescope. Further analysis of these will be performed to try to ascertain whether the simulations are realistic or not, indicating if our galactic model is correct.

The Model

The most critical part of this project is the construction of the computer simulation model as it is from this that all the results will be obtained. This section will look into the different types of particle simulation models that could be used, discussing the relative merits of each when used for simulating systems containing large numbers of particles. The most appropriate simulation model will be chosen and used to construct a computer program to perform the simulation. This will be kept very simple at first so it can be tested and analysed easily to make sure the simulation is working correctly. Once this has been achieved, the program can be optimised to improve speed and accuracy if possible, whilst referring back to the previous test results as a check. At the end of this section the simulation should be ready to use and initial conditions will be discussed.

Particle methods

The most appropriate way of simulating a system such as this is using particle methods, this involves tracking the trajectories of a number of particles as they move through the system interacting with each other appropriately. The particles used in the simulation do not necessarily have to correspond to actual particles such as stars or atoms, instead they could be groups of stars or just represent a small amount of a fluid being simulated. In this way, particle methods are very versatile and are often used for simulating flowing liquids, gases and plasmas. The advantage of using particle methods is that they can be used in very complicated systems where there is no analytical solution. Take for example, a single planet orbiting a star - the orbit of the planet can easily be obtained using differential equations. If, however, we introduce a second planet, the interactions become a lot more complicated and the orbits cannot be expressed so simply. Using particle methods, large numbers of interacting particles can be simulated, the forces between each pair of particles being considered at every step.

A medium to large galaxy contains ~10^{11} stars, this is obviously far too many for individual stars to be used as particles: this would require more than a Terabyte of memory! instead the particles will have to represent a large number of stars. The exact mass of these 'superstars' will depend on the number of particles used and the mass of the galaxy being simulated.

The Particle-Particle (PP) method

The biggest problem with using particle methods arises when the interactions between the particles need to be evaluated. The movement of every particle depends on the gravitational force on it, this is made up of contributions from *every other particle* in the system. If N particles are used in the simulations, with the force between each pair of particles given by:

$$\mathbf{F}_{1,2} = -\frac{Gm_1m_2}{r^2}\hat{\mathbf{r}} \qquad (1)$$

Then this needs to be evaluated N^2 times to perform each step of the simulation. On a fairly powerful modern computer this would require more than an hour if more than a few tens of thousands of particles are used. This would result in a program that is far too slow and inaccurate.

The Particle-Mesh (PM) method

An alternative to calculating the force on every particle is to calculate the gravitational field over the space occupied by the particles, then applying the force to the particles due to the field at their position. Instead of calculating the field at every point in space, it can be calculated at a series of points lying on a *mesh*. Also, instead of calculating the field due to every particle, we can make the approximation that the mass of each particle lies at its nearest grid point. We can then calculate the field at one grid point by considering the masses at the other points. This is known as the *particle-mesh* or PM method and is described in Hockney and Eastwood [1]. If a mesh of C by C points are used, then of the order of C^2 x C^2 = C^4 calculations of equation 1 are required. Somewhere in the region of 100 by 100 cells is ample for this type of simulation, resulting in a program that runs 100 times faster than the equivalent PP method with 100,000 particles.

Cell weighting methods

In the method described above, the mass of each particle is assumed to lie at its nearest grid point. For obvious reasons this is known as the *nearest grid point* (or NGP) method. A better approximation may be to share out the mass of each particle between the adjacent grid points in proportion to the particle's distance from each, however, this would require a lot more calculation for each particle, this is known as the *cloud in cell* (CIC) method. The NGP method will initially be used for the simulation, although CIC may be used later on if appropriate.

Collisionless model

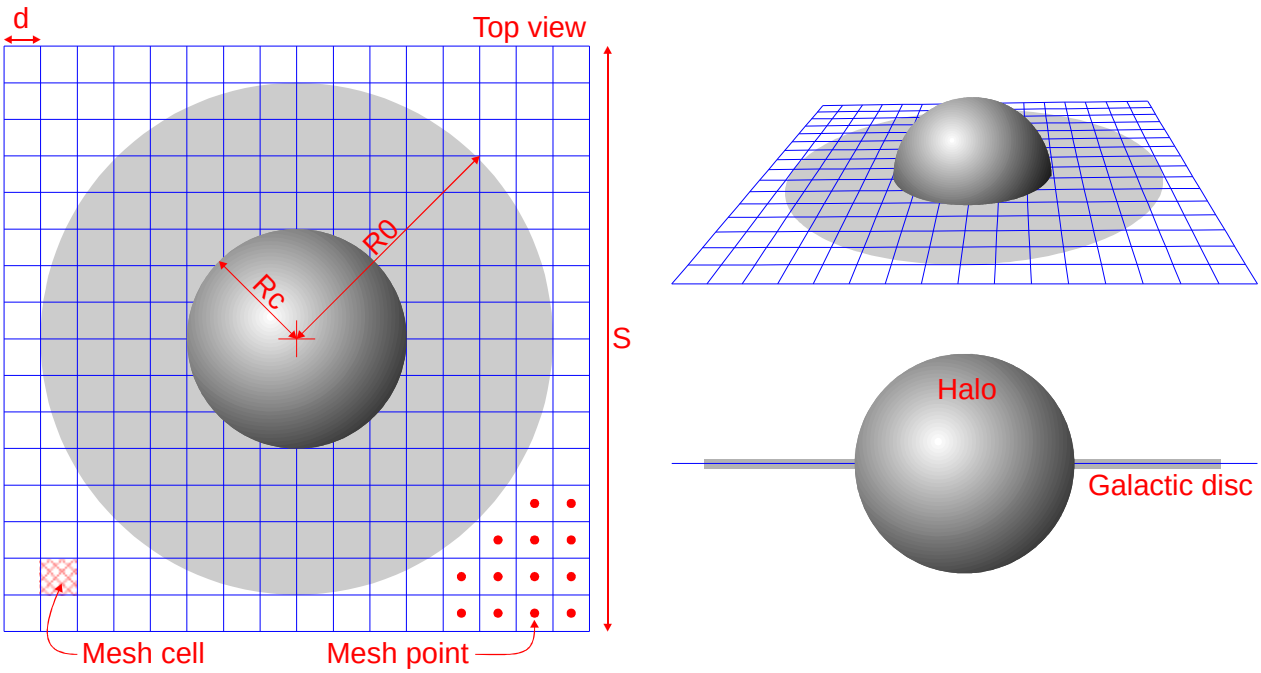
In a typical galaxy the minimum distance between stars (excluding multiple star systems) is around 10^{16}m, while the maximum radius of the very largest stars is no more than 10^{10}m. In other words, the mean separation is more than 1 million stellar radii. In comparison, molecules in a typical gas have a mean separation of around 50 molecular radii. This means that the chances of two stars colliding in the simulation are *extremely* unlikely (note the term "collision" here refers to particles being close enough for their trajectories to be significantly deflected). The mean free time between collisions for a system of stars can be obtained in a similar manner for that of a gas, and is:

$$t_c = \frac{v^3}{4pG^2m^2n} \qquad (2)$$

Where v, m and n are the mean values for the velocity, mass and number density of the stars in the system. This will be used to check the simulation to see if the collisionless approximation holds. It is important that the system is collisionless if the PM model is to be used because particles cannot interact with particles within the same mesh cell, making collisions impossible. This is a real advantage from a programming point of view because collisions can lead lead to particles gaining a huge velocity and exceeding the range of the variables used to store it. In effect, the PM model introduces *gravitational softening* at short ranges, a similar effect could be achieved in PP models by replacing r with (r-d), where d is a constant, in equation 1. This would make the force go to a finite value instead of infinity for zero separation.

Setting up the mesh

As a circular disk of stars is to be simulated, The possibility of using a polar co-ordinate system for the mesh was investigated. However, this creates more computational complexity due to the cells having different areas and the distances between mesh points becomes more difficult to calculate. It was therefore decided that a regular Cartesian mesh would be more appropriate. It was observed that the stars that make up the spiral structure in spiral type galaxies lie mainly in a highly flattened disc, with a spherical bulge or halo in the centre. it was therefore decided to simulate just the disc stars and assume the halo is stationary so it can be added on as a radial force to every particle. Because of the high degree of flattening seen in the discs of stars of most galaxies, it was decided that the particles should be constrained to move in 2 dimensions, in this way a 2 dimensional mesh could be used, reducing the calculation time extensively.



a Figure 1 – Mesh layout.

Figure 1 shows the layout of the mesh and how it fits into the galactic model that will be used in the simulation (a finer mesh than that shown here will be used in the actual simulations). Note the mesh points lie at the centre of each mesh cell, the mesh spacing is denoted by *d*, the length of the mesh edges is S (units will be discussed later). R0 is the initial galactic disc radius, Rc is the central halo radius. The halo (shown as a shaded grey sphere in the diagram) will be ignored for this initial model, but will be included later to see what effect it has.

Units

An important point to consider is the system of units to be used in the simulation, using SI units would result in very large numbers being used as a typical galaxy has a radius of about 6×10^{20} metres. It is better programming practice to scale the units so they are closer to unity, this is because it reduces the truncation error involved in storing numbers and also makes the numbers easier to deal with from the point of view of the user.

A more appropriate system would be to use kiloparsecs as the length unit, with solar masses and years as the mass and time units respectively:

Dimension	Unit	SI Equivalent
Length	Kiloparsec (Kpc)	3.08×10^{19} m
Mass	Solar mass (M_{SUN})	1.99×10^{30} Kg
Time	Year	3.16×10^{7} s

The gravitational constant G has dimensions [L^3/MT^2], in this unit system its value becomes G*, given by:

$$G^* = G_{SI} \frac{\left(3.155\times10^{7}\right)^2 \left(1.99\times10^{30}\right)}{\left(3.08\times10^{19}\right)^3}$$

$$= 4.5870\times10^{-24}\ \mathrm{Kpc}^3\ \mathrm{M}_{\mathrm{SUN}}^{-1}\ \mathrm{Year}^{-2}$$

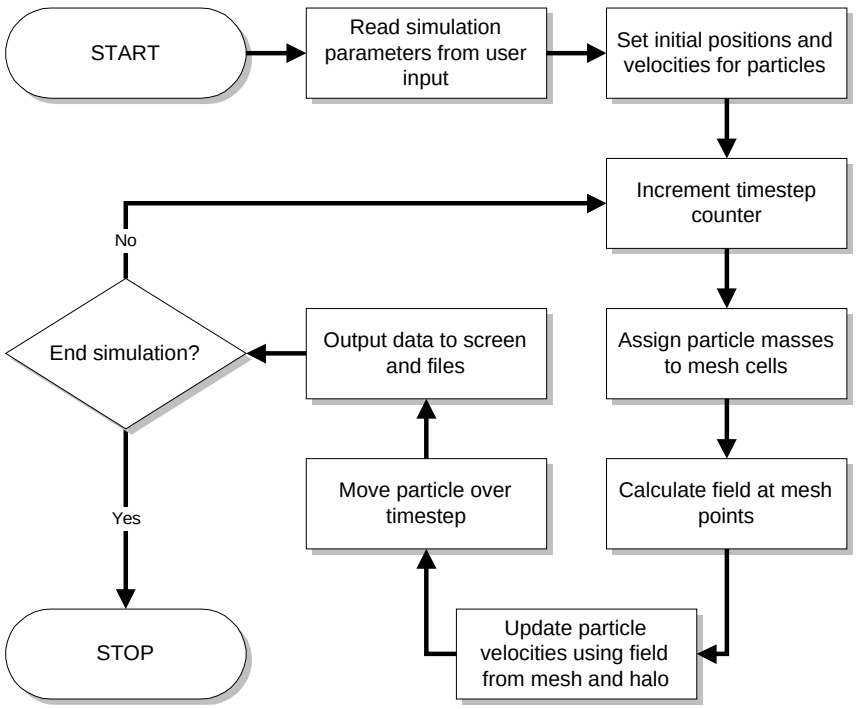
Programming considerations

In order to perform the simulation, an appropriate computer program needs to be constructed. The 32 bit Windows platform was chosen because it allows large amounts of memory to be accessed in continuous blocks[2], as well as supporting multitasking. In this way one part of the program can be dedicated to running the simulation, while another is outputting the results, and other programs can be used simultaneously to examine these results. In order to provide graphics output at a reasonable speed, the Microsoft DirectDraw library was used. This allows direct access to the screen buffer so pixel values can be placed straight into display memory from the simulation, bypassing Windows' slow pixel writing routines. The C++ language was chosen because it is the most appropriate for mathematical programming under Windows where speed is of the utmost importance.

Overall program structure

It was decided to keep the structure of the program fairly simple at first so it could be thoroughly tested and debugged easily. The program runs under Windows 95 using full screen mode to display the current state of the simulation. The simulation routines run in a background thread using the Win32 multi thread model, this means the system is not tied up when lengthly calculations are being run, so the program can be more interactive. It also means that all available processor time is allocated to the calculations when the system is not doing anything else.

A flowchart showing the structure of the simulation part of the program is shown in Figure 2, each aspect is also described in detail in this section.



a Figure 2 – Flowchart showing overview of program operation.

[2] **In 16 bit systems, memory is accessed in 64KB segments, this slows down execution time if large amounts of memory are required because the program has to keep flipping between segments in order to access a large array.**

Data storage

The program parameters are stored as global variables, The most important ones are desrcribed here:

Name	C++ Type	Dim	Description
D	Double	[T]	Timestep
T	Integer	-	Current timestep
N	Integer	-	Number of particles (superstars)
M0	Double	[M]	Mass of each particle
R0	Double	[L]	Initial disc radius
VDISP	Double	$[LT^{-1}]$	Initial velocity dispersion
Mh	Double	[M]	Central halo mass
Rc	Double	[L]	Central halo radius
C	Integer	-	Number of mesh cells along edge
S	Double	[L]	Length of mesh along edge
D	Double	[L]	Mesh spacing

To make it easier to work in 2 dimensions, we define a structure called VECTOR:

VECTOR structure		
'x'	Double	X component
'y'	Double	Y component

The particle data is stored in an array of N STAR structures, each STAR structure contains:

STAR structure		
'r'	VECTOR	Position of star
'V'	VECTOR	Velocity of star
'm'	Double	Mass of star
'c'	Integer	Type of star

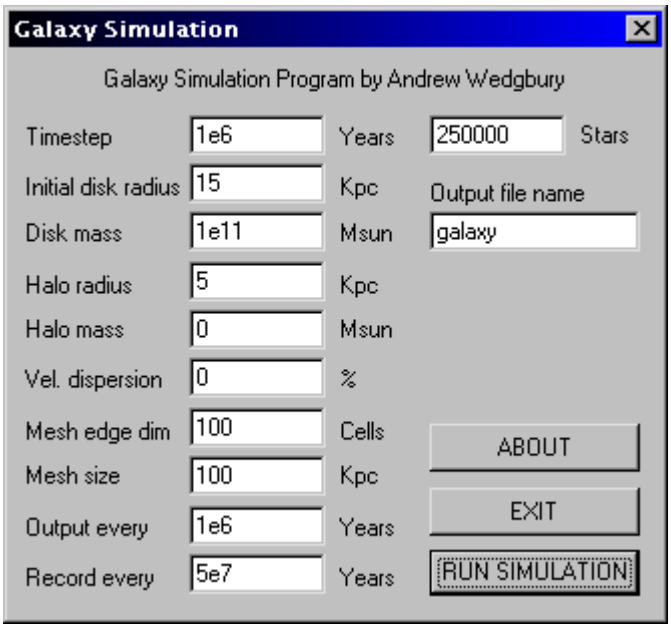
Storing the mass of each star is unnecessary if all stars have the same mass, as they will for most of the simulations, but it allows the effect of a mass distribution to be investigated. The star type can be used to pick out particular stars, stars of different type will be displayed differently by the output routines.

The mesh is stored as a C by C array of MESHCELL structures, each MESHCELL structure contains:

MESHCELL structure		
'g'	VECTOR	Gravitational field at mesh point
'm'	Double	Total mass in cell

There are also lost of other variables that store other information such as data to be displayed, file handles and buffers.

When the program starts, it pops up a dialog box to allow program parameters to be input, this is shown in Figure 3. These values are stored in the variables described above when the user presses the "run simulation" button and the program proceeds with the simulation. The user can stop the simulation and return to this dialog box at any time in order to change values or start over again.



[a] Figure 3 – Parameter input dialog box displayed by the program.

Field calculations over the mesh

This is one of the most computationally expensive parts of the program - the field at each mesh point is evaluated by considering contributions from every other mesh point. First the program loops over the stars array to assign the particle masses to their nearest grid points, then the program performs two nested loops over the mesh to calculate the field at each mesh point (u,v) using:

$$\mathbf{g}_{u,v} = -\sum_{x,y \neq u,v} \frac{Gm_{x,y}}{d^2\left[(x-u)^2+(y-v)^2\right]^{3/2}}\left[(x-u)\hat{\mathbf{u}}+(y-v)\hat{\mathbf{v}}\right] \quad \textbf{(1)}$$

Where $m_{x,y}$ is the total mass of particles in cell (x,y). The summation is taken over all mesh points using indices (x,y) but not including the cell x=u and v=y (the current cell). It was noticed that the $-(G/r^3)r$ part of this equation is the same for pairs of mesh points that have the same x and y displacements, the calculation would be greatly speeded up by caching these values before the simulation starts and referring to them in an array. This required two C by C arrays of double precision variables (one for the u and one for the v component), which is a lot of extra memory but well worth it for the increase in speed.

It was also noticed that, especially at the start of the simulation, the particles may only occupy a small proportion of the mesh cells. It is therefore very wasteful in computer time to keep checking every mesh point for field contributions when more often than not the contribution will be zero. To improve this, the program loops over the mesh and stores the u,v co-ordinates of each cell that contains a particle in another array. The program then loops over this new array instead of the whole mesh. Again, this uses more memory but was found to increase the overall speed by over 20% if only half the mesh is being used.

Updating particle positions and velocities

Next the program loops over the stars array and updates the velocities of each particle. The new acceleration of each particle can be obtained using the mesh, it is then trivial to update its velocity:

$$\mathbf{V}_s^{t+1} = \mathbf{V}_s^t + D\mathbf{g}_{u,v} \quad (2)$$

Where $g_{u,v}$ is the field in the cell containing the particle s and V_s^t is the particle's previous velocity. D is the time-step parameter.

Its new position, r_s is then given by:

$$\mathbf{r}_s^{t+1} = \mathbf{r}_s^t + D\mathbf{V}_s^t \quad (3)$$

The "CheckBounds" subroutine is run to check if the particles are within the mesh. Particles leaving the mesh cause problems because the forces on them or their force on other particles can no longer be calculated. Some other simulations track particles using the PP method if they leave the mesh, but for this model it was decided to make them "bounce" off the edge of the mesh, i.e. reversing their velocity perpendicular to the edge that was hit. The optimisations made in the mesh calculation stage make it possible for much larger meshes to be used so larger gaps can be left around the simulated disc to allow for any particles flying off. The program also records the number of edge hits and the mesh usage so conclusions can be drawn about the accuracy when analysing the output.

Data output

One of the most difficult parts to program is the data analysis and output routines. This is mainly due to the enormous amount of data the program produces, exactly what data was relevant had to be decided upon because outputting all available data would quickly fill up any computer's storage resources. It was decided to split the output into different files, one for small amounts of data to be output to continually in order to draw temporal graphs, and others for large amounts of data such as the entire stars or mesh array, or pictures of the galaxy at different stages. To the temporal graph file, the program calculates and outputs the time, mesh status, kinetic, potential and total energy of the system. Also the radii within which 50%, 90% and 100% of the mass is contained.

Less frequently, the program outputs a portion of the star array, the radial density distribution, and the total field and mass at each mesh point so graphs of these can be drawn. The program also outputs small bitmap files showing the star positions, some of which will be used in the results section of this report. The exact output interval for all this data can be selected by the user in the parameters dialog box.

Choice of parameters and accuracy

For an initial test of the accuracy of the simulation, the program was set up to simulate a simple 2-body system, data the Sun and the Earth was used in the initial conditions. A full description of this investigation can be found in the December 1999 interim report for this project. By comparing the results with that obtained using PP methods, and examining the total energy evolution, it was observed that the accuracy is improved by using larger mesh sizes and smaller timesteps upto a point.

When using the PM method, the particles behave as if they are smeared out over a cell. If we are looking to see features such as spiral arms and small structures developing in the disc, we need to make sure the mesh spacing is less than the size of these structures. A mesh size of 60x60 Kpc with 100x100 cells was chosen

by careful examination of some typical spiral galaxies. To determine the timestep, we need to make sure particles do not move too far in one step. About 200 steps per galactic rotation is adequate to ensure this, therefore a timestep of 1 Million Years was chosen, which is about right for a typical galaxy.

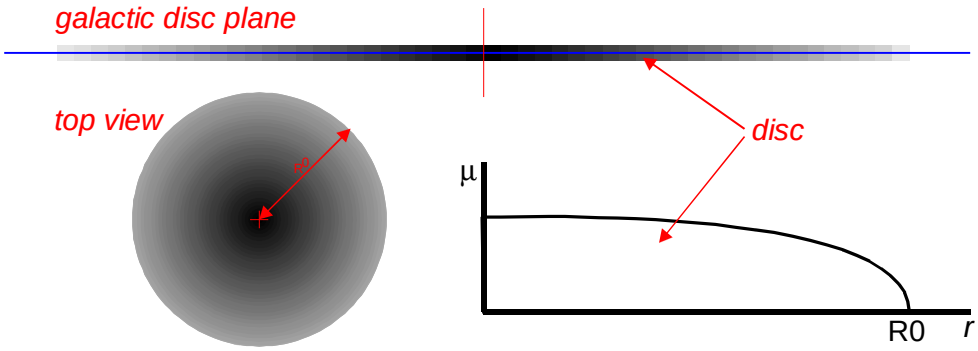
In the next section, a simple galactic model will be used to obtain some initial results using the simulation program.

A Simple Stellar Disc

The program will be used to simulate simple discs of stars with several different initial density distributions. The particle velocities will be set to just balance the rotation of the disc. The simulation will be run until the fate of the initial disc is clear. This will allow the program to be fine-tuned and hopefully give an indication as to how our model can be improved.

Setting up the model

The model used for the simple disc is shown in Figure 4. A value of 15Kpc was used as the initial disc radius as this is approximately the radius of our galaxy, thought to be a typical spiral system. A galactic mass of $1x10^{11}$ M_{SUN} was used, which is approximately the mass of the disc of our galaxy.



[a] Figure 4 – Galactic model and initial surface density distribution used in this simulation.

For this first simulation, the initial surface density distribution, μ(r), used was:

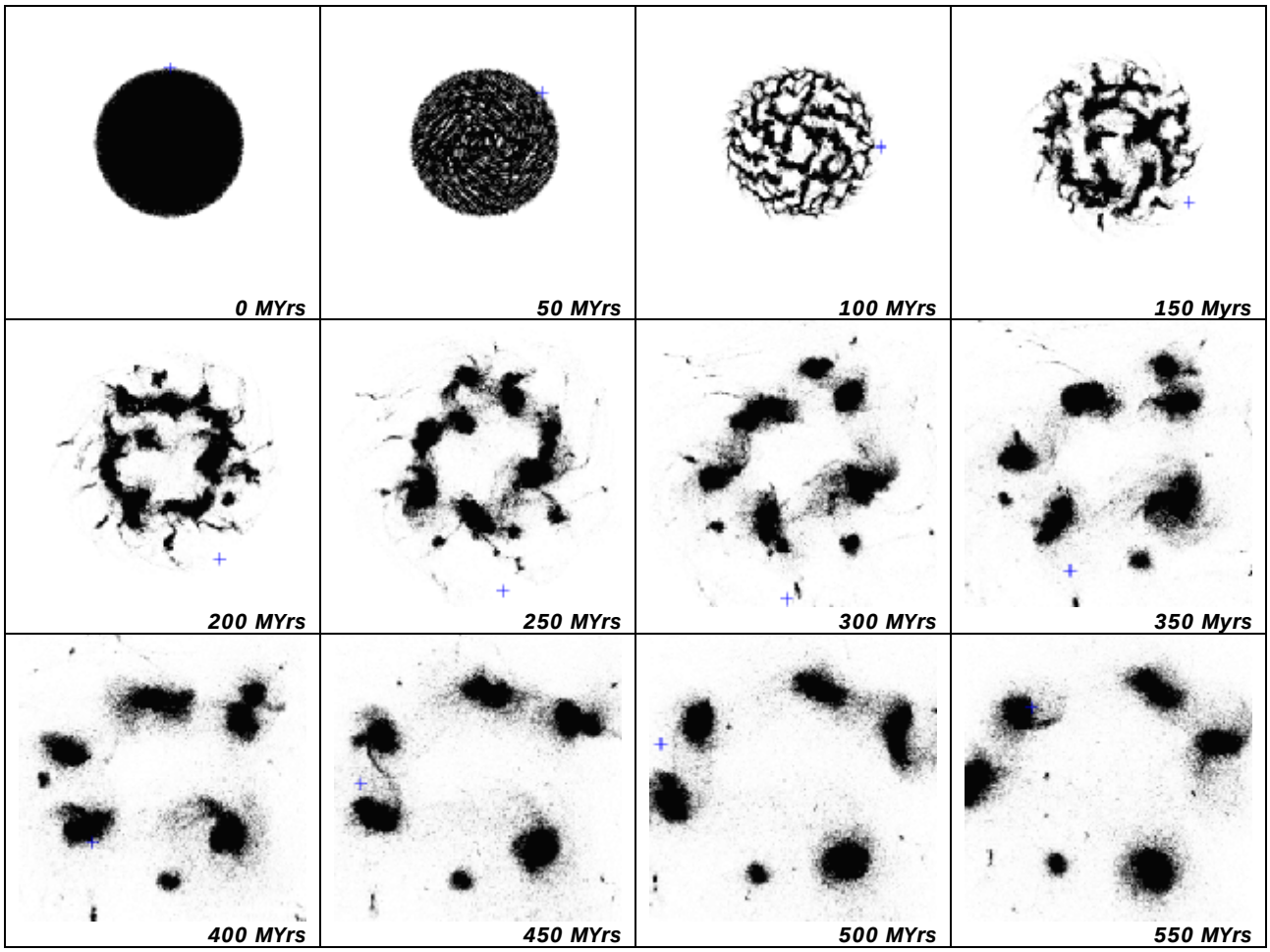
$$m(r) = m(0)\sqrt{1-\left(\frac{r}{R_0}\right)^2} \qquad (4)$$

Where μ(0) is the central surface mass density and r is the radial co-ordinate. This was set up in the "SetInitial" subroutine and used a rejection method random number generator. The resulting distribution looked like that shown on the graph in Figure 4, which is thought to be a reasonable estimate of the surface density distribution of a stellar disc. The particles were assigned a purely rotational velocity just enough to balance the disc against gravitational collapse.

Results

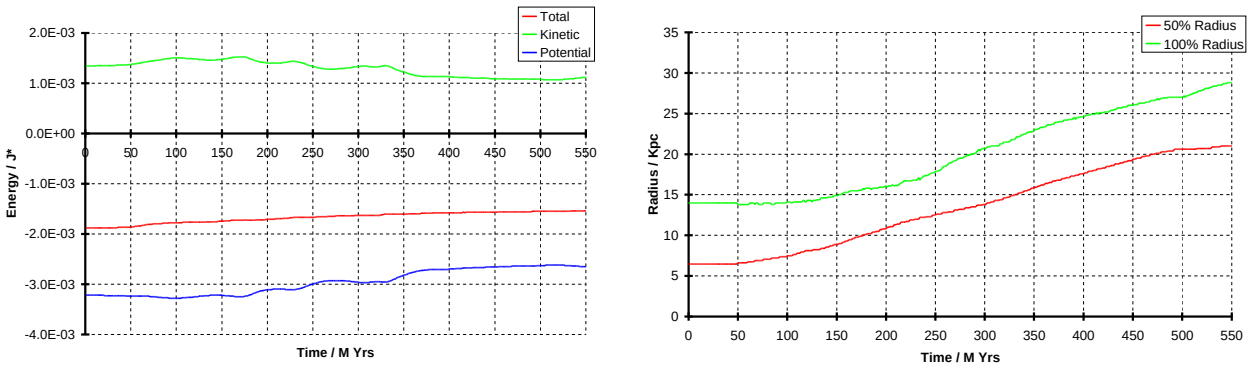
The program was run for 550 time steps, by which time it was clear that a large amount of particles had hit the sides of the mesh and the disc had broken up. The results for this simulation are shown in Figure 5. The last image shows the disc has split into six large globular structures, which formed at around 350 Myrs and appear quite stable. There was very little indication of spiral structure, which would suggest that spiral galaxies do not form with these parameters, although objects that appear as small elliptical galaxies are readily produced. It must be emphasised, however, that this is a 2 dimensional model and is not particularly suitable for simulating elliptical galaxies or globular clusters, as they do not exhibit the high degree of flattening seen in spiral discs.

Using the numerical output from the program, the kinetic, potential and total energy for the whole system was plotted against time, shown in Figure 6 (left). Conservation of energy requires that the total energy of the system remains constant, so this is a good indication of the accuracy of the simulation. The value was actually observed to increase by about 18% over the duration of the simulation, although the particles hitting the sides could account for some of this.



a Figure 5 – Evolution of galactic disc shown at intervals of 50 MYrs, the disc breaks up very rapidly within half a rotation. Note the particle shown by the blue cross, which can be used to trace the rotation of the disc. The program uses grey-scale to represent the particle density, with completely black areas being the densest.

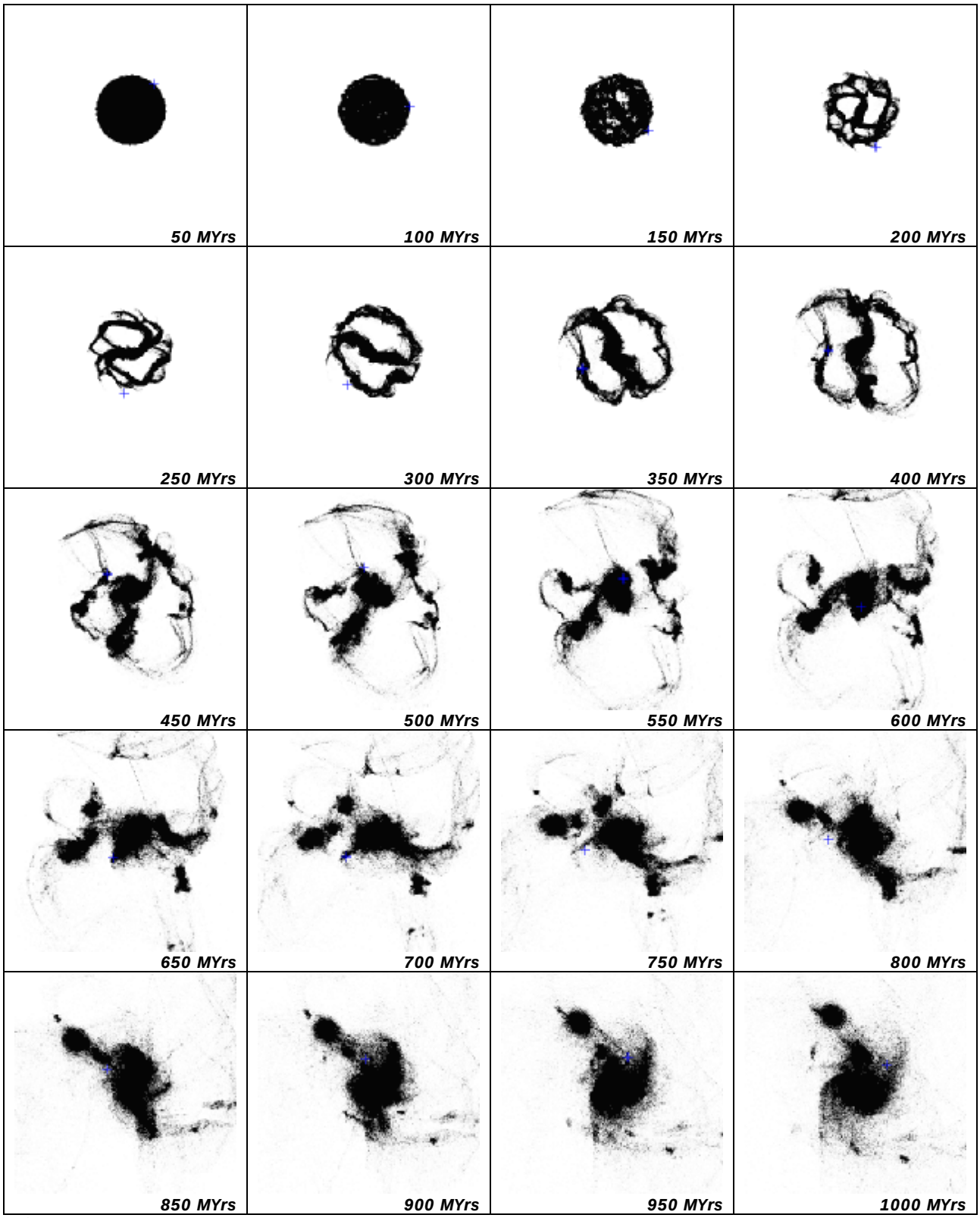
Figure 6 (right) shows the radii within which 50% and 100% of the mass of the system is contained plotted against time. This was seen to increase steadily after 50 Myrs.



a Figure 6 – Evolution of kinetic, potential and total energy *(left)* and radii within which 50% and 100% of the mass is contained *(right)* for initial run with no halo. The energy unit J* corresponds to about 1.90 x10^{54} Joules.

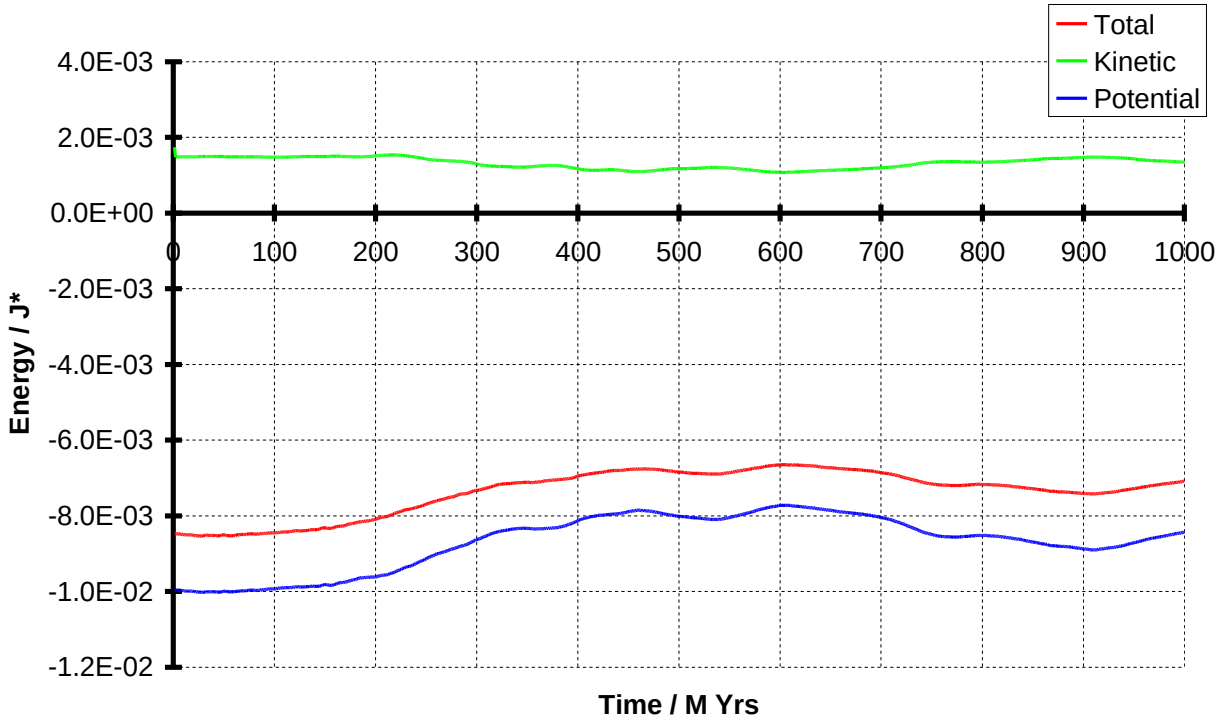
Improvements

To attempt to improve the model, the initial disc was changed so it was much denser in the centre. The actual density distribution used was $r^{-1/2}$, where r < R0. The simulation was run for 1000 MYrs and the results are shown in Figure 7.



[a] Figure 7 – Evolution of galactic disc for second simulation

The disc assumed a′θ′ shape between 200 and 400 MYrs, which then split apart with arms emerging and hitting the side of the mesh at 550 MYrs. The central region stayed intact, with arm-like structures trailing as it rotated. The final state at 1000 MYrs has a vague spiral shape with a large circular cluster orbiting at a radius of about 25 Kpc. In Figure 8 the energy is plotted against time, revealing a 25% variation in total energy.



[a] Figure 8 – Energy / time graph for second simulation.

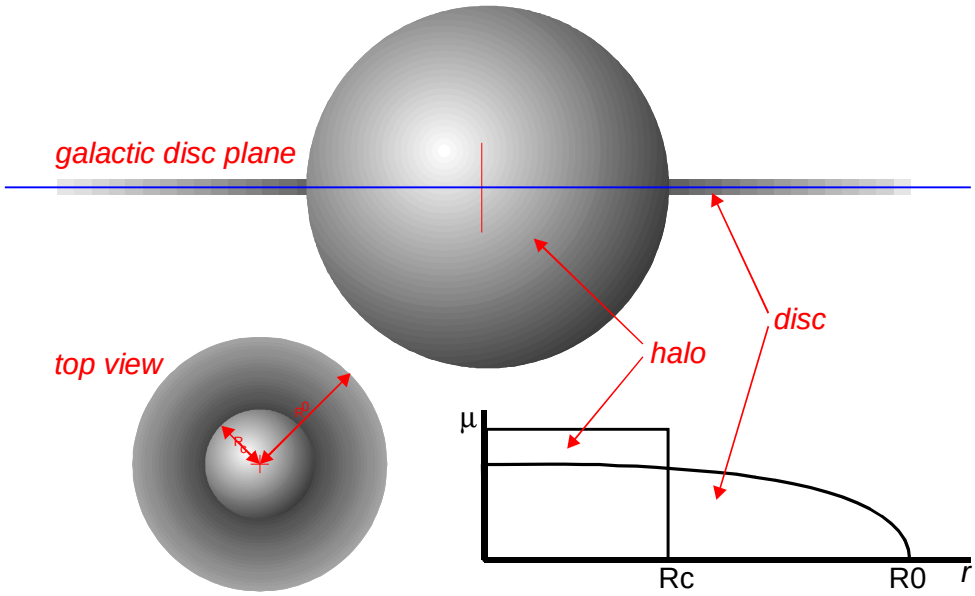
It was found that using a larger number of mesh cells could reduce this error. 60x60 cells were used in the last simulation in an attempt to speed up the calculations, but 100x100 will be used in future simulations.

Disc With Fixed Halo

The last section revealed that a large concentration of mass is required at the centre of the disc to prevent it from breaking up into small clusters. Actual galaxies are observed to have a central bulge and a roughly spherical "halo" consisting of globular clusters. Several different models for the halo component will be constructed and tested in this section.

Uniform density halo

The initial model for this section is shown in Figure 9, with a fixed solid sphere of uniform density and mass Mh at the centre of the star disc.



[a] Figure 9 – Galactic model used in this simulation, including a fixed, uniform density, spherical halo.

The gravitational field at *r* due to a solid sphere is given by:

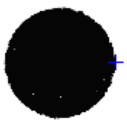
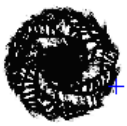
$$\mathbf{g}_{sphere} = \begin{cases} \dfrac{-GM_h}{r^2}\hat{\mathbf{r}} & \text{for } r > R_c \\ \dfrac{-GM_h r}{R_c^3}\hat{\mathbf{r}} & \text{for } r < R_c \end{cases} \qquad (1)$$

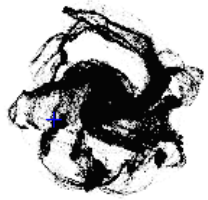
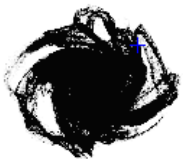
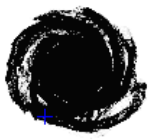
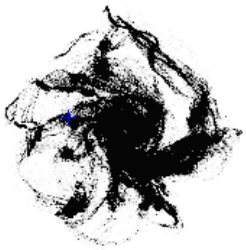
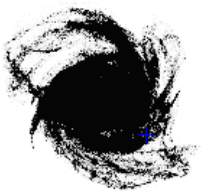
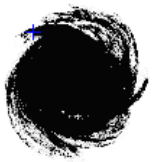
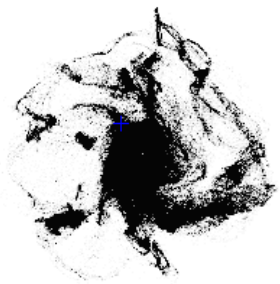
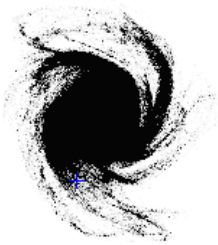
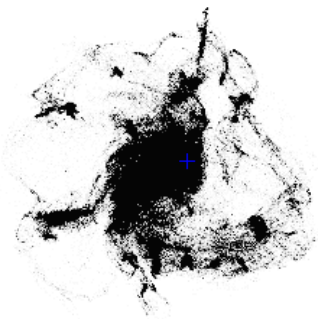
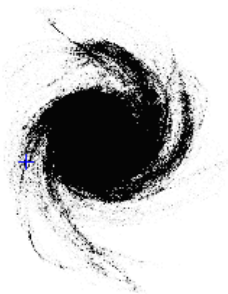
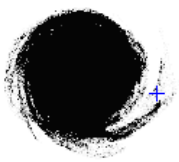
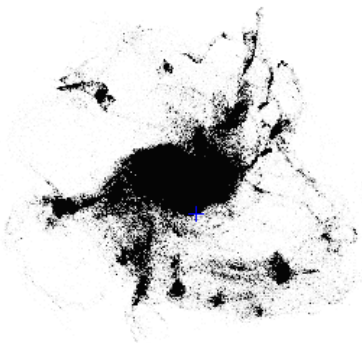
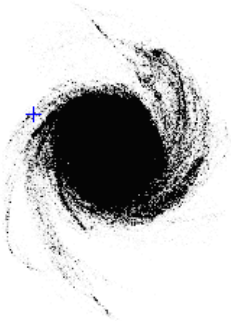
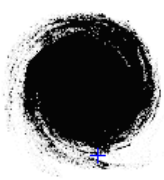
Where Rc is the radius of the sphere and Mh is its mass. The function "CalcHaloField" was added to the program to calculate this, and the equations of motion were adjusted to include the extra field due to the halo.

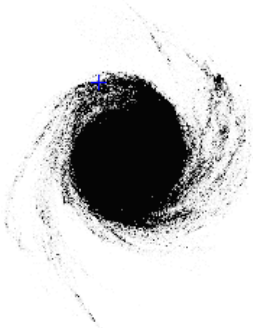
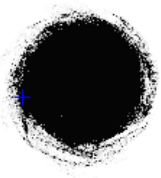
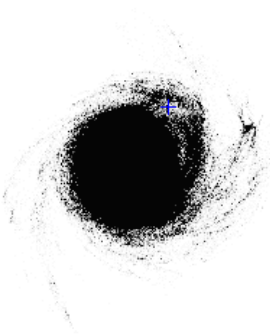
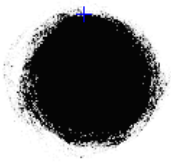
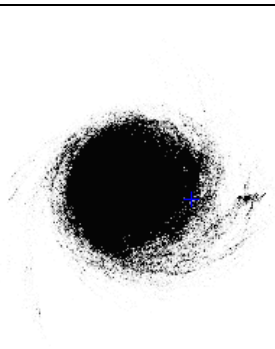
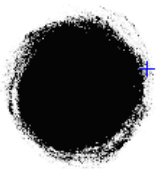
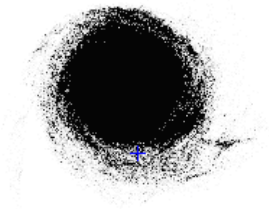
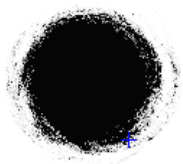
The initial disc radius was kept at 15 Kpc and the halo radius was set to one third of this, which is approximately the halo size in our galaxy. Little is known about the mass of galactic halos, so values of 1x, 5x and 10x the disc mass were tried.

Initial Results

The results are shown side by side for comparison in Figure 10. The 1x disc mass halo evolves in much the same way as the last simulation in the last section, but a more pronounced spiral system is visible. This expanded and hit the sides of the mesh at 500 MYrs, where the simulation was ended. In the other two simulations, however, the disc stars stayed well within the mesh and were run to 650 MYrs. A well-defined two-arm spiral system was seen to develop in the 5x disc mass halo system at about 400 MYrs, but seemed to disappear at 650 MYrs. The particles in the 10x disc mass halo system were more confined to the centre, but did produce small spiral arms, which were visible between 250 and 450 MYrs.

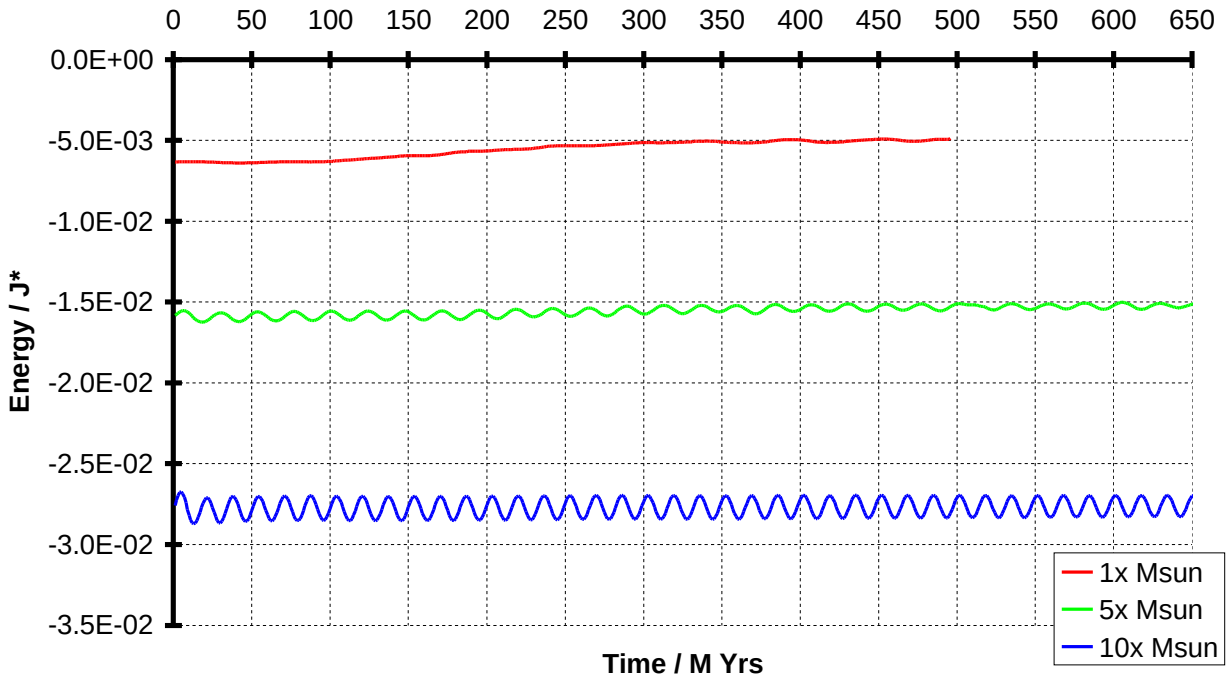
Time	Halo mass		
	1x disc mass ($1x10^{11}$ M_{SUN})	5x disc mass ($5x10^{11}$ M_{SUN})	10x disc mass ($1x10^{12}$ M_{SUN})
50 MYrs			
100 MYrs			
150 MYrs			
200 MYrs			

250 MYrs			
300 MYrs			
350 MYrs			
400 MYrs			
450 MYrs			

500 Myrs	***(Disc broken up)***		
550 Myrs			
600 Myrs			
650 Myrs			

a Figure 10 – Evolution of galactic discs for different values of the halo mass. The disc mass was kept constant at $1x10^{11}$ M_{SUN}.

Total energy variation



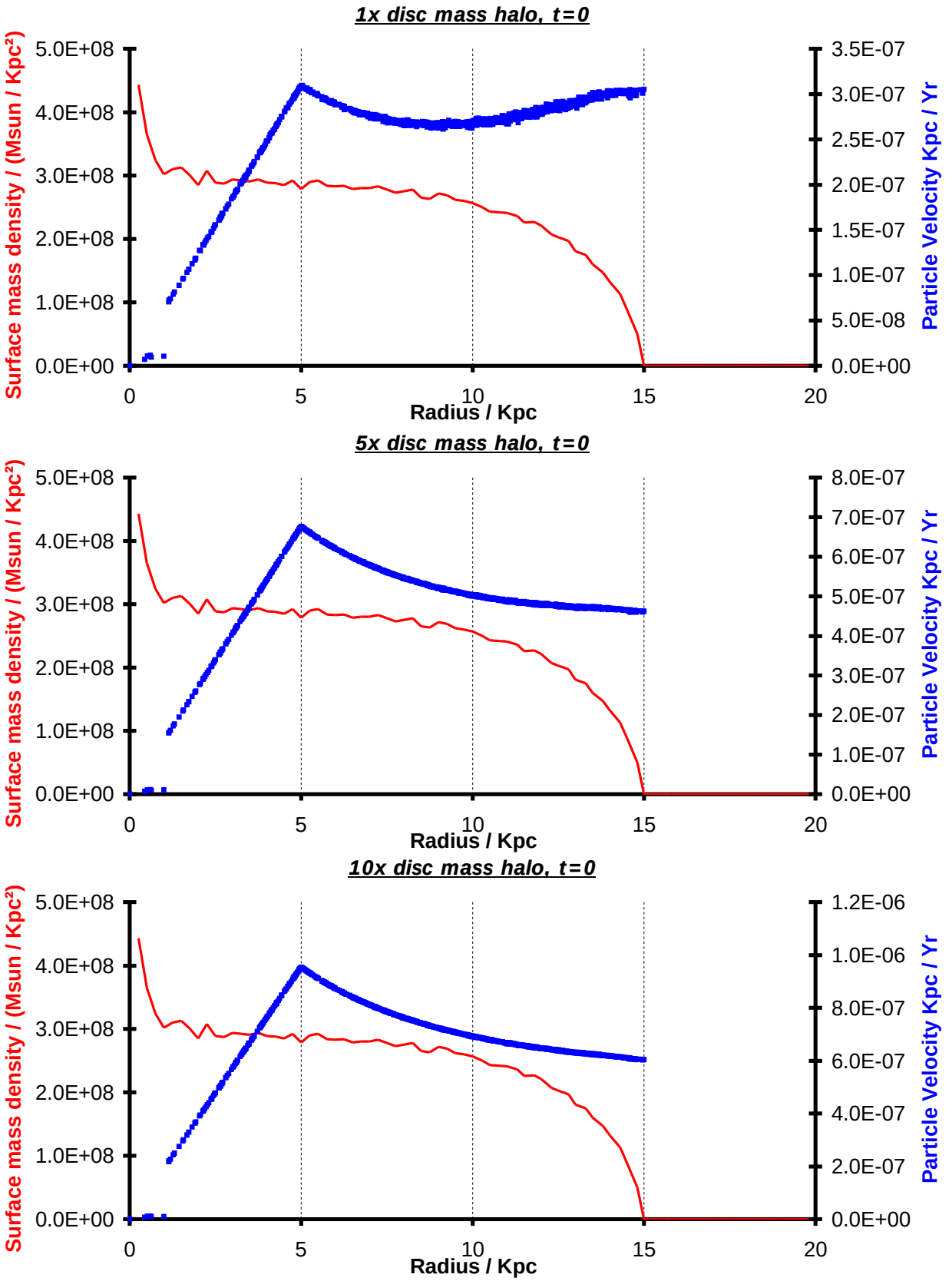
[a] Figure 11 – Total energy / time plots for different halo masses.

To assess the accuracy of these simulations, the total energy against time plots are shown in Figure 11. The total energy of the systems vary by about the same absolute amount, but curious oscillations are observed in the 5x and 10x cases. One explanation could be the error due to the mesh approximation, with particles moving faster in the systems with heavier halos.

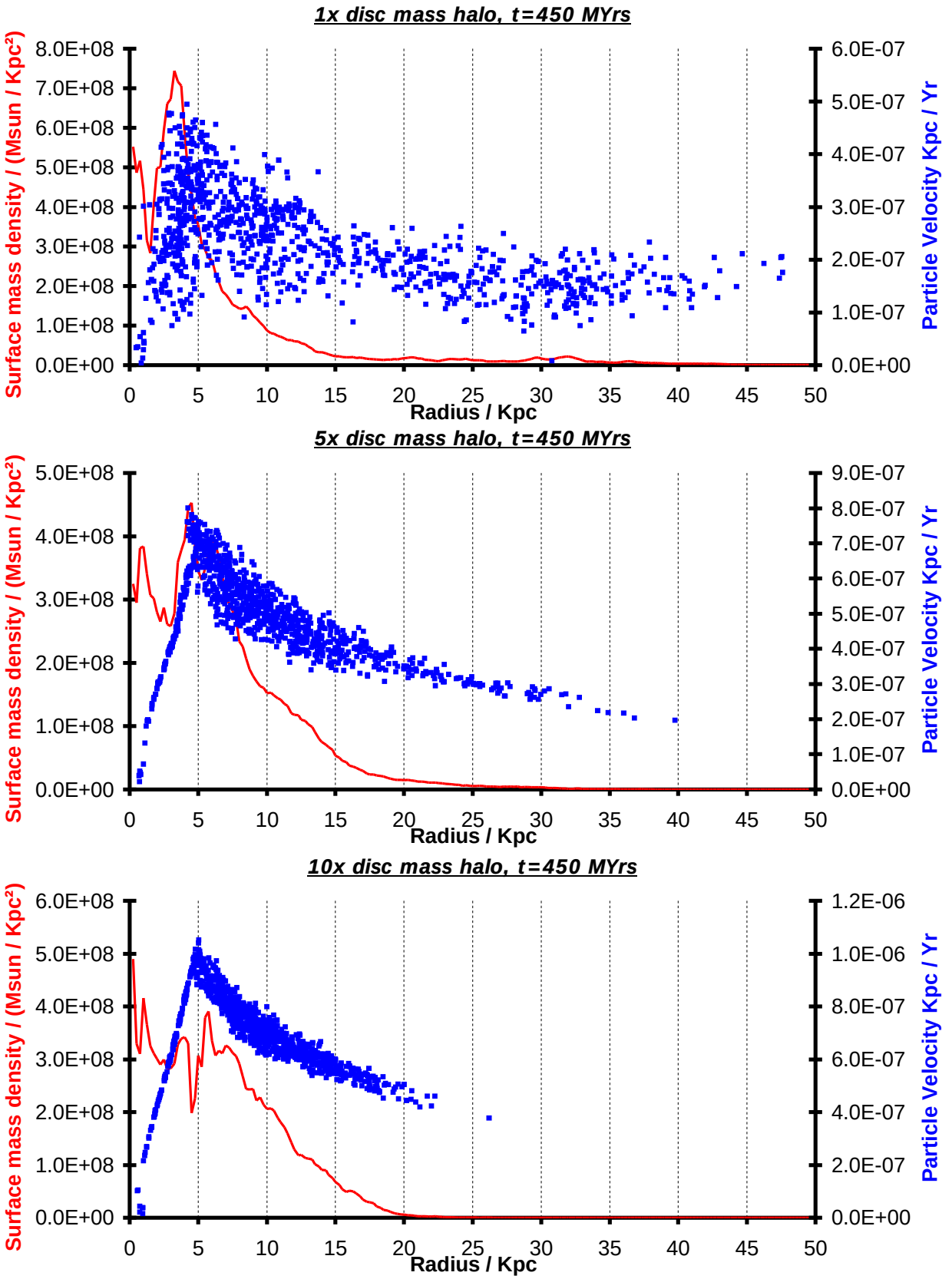
Density and velocity distributions

The density and velocity vs. radius graphs for the three different halo masses are shown in Figure 12. Here we can see that the maximum velocity in the 10x system is about three times greater than the 1x system. The maximum velocity occurs at the halo radius, where the field due to the halo is strongest. Inside the halo, the velocity varies proportionally to radius. Outside the velocity decreases gradually, except in the 1x case where it dips and rises again at the disc radius.

Figure 13 shows the density and velocity against radius plots at 450 MYrs, in all cases the particles have spread out to some degree, enough to reach the edge of the mesh in the 1x system. The velocity distribution appears erratic in the 1x system, but in the 5x and 10x systems it still seems to follow a trend similar to that at t=0, with some dispersion.

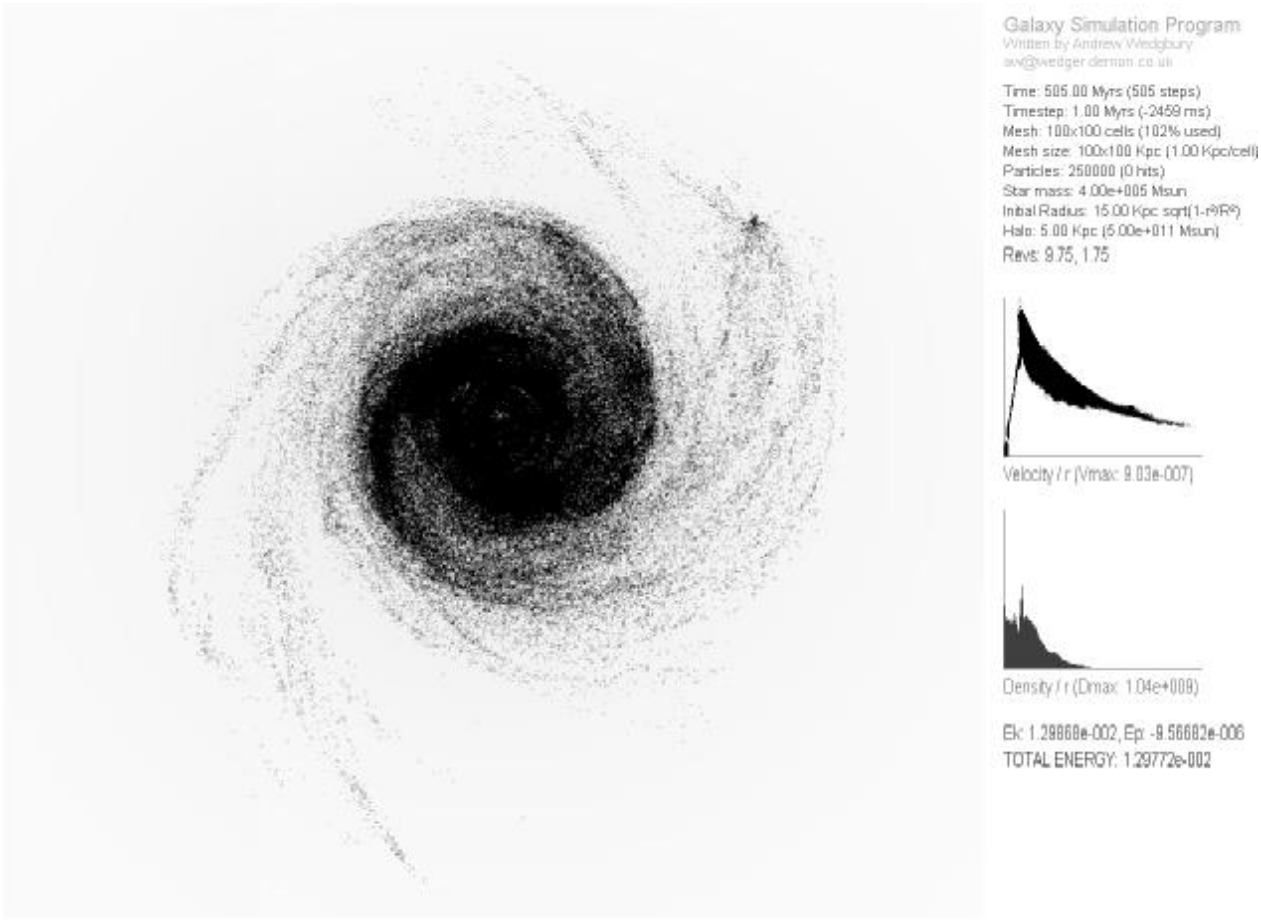


[a] Figure 12 - Plots of disc surface mass density and total velocity against r for the different halo masses at t=0. The density distribution is the same for each, but the velocity increases with halo mass, this is the initial azimuthal velocity assigned to the particles to balance the gravitational force. Note the discontinuity in velocity at r=5 Kpc due to the halo, which is modelled as a solid sphere of constant density.



[a] Figure 13 – As Figure 12 but at t=450 MYrs.

The most prominent spiral structure was observed with the 5x disc mass halo, an example of which is shown in Figure 14 below.

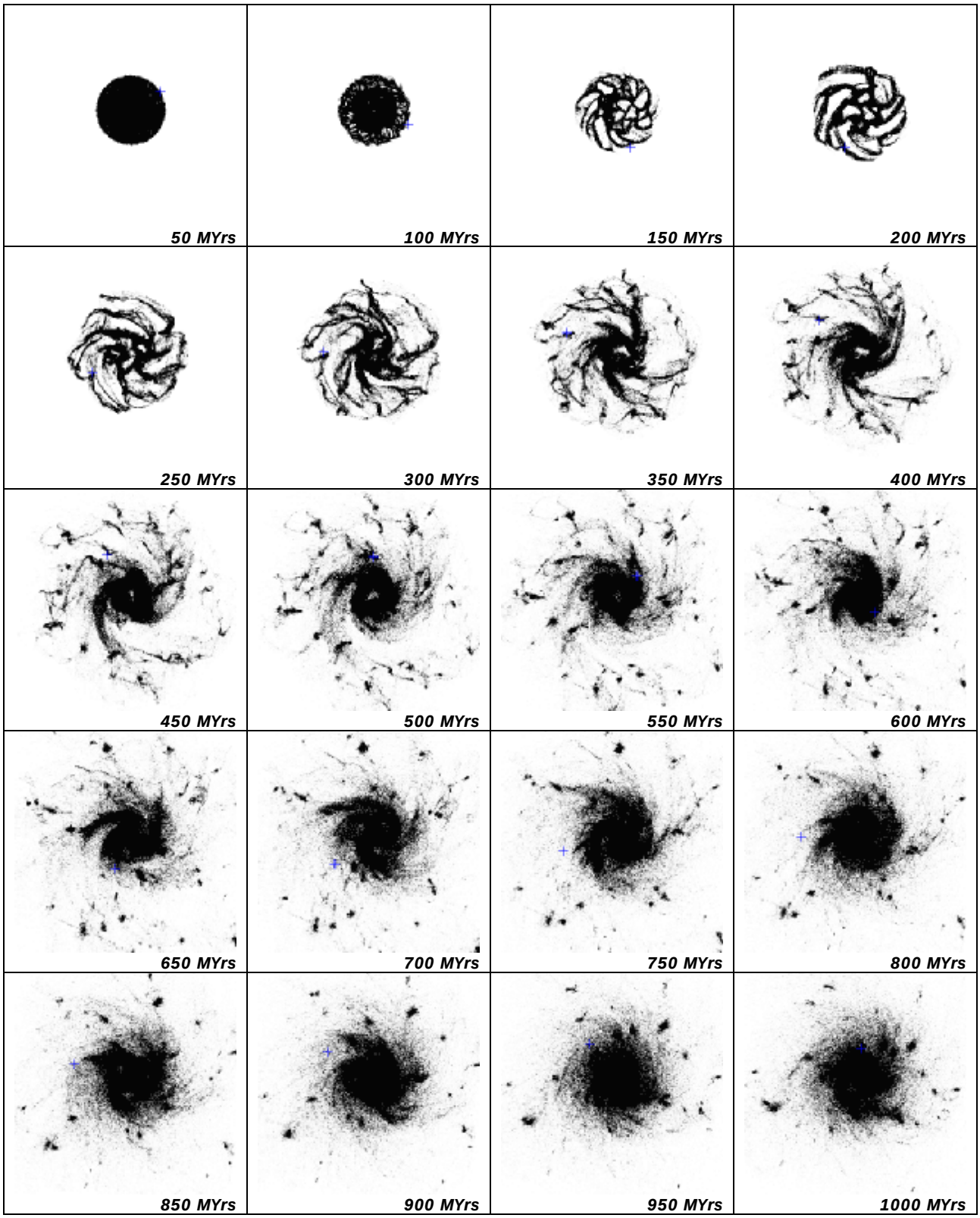


[a] Figure 14 – Program screen shot for the 5x halo mass system at 505 MYrs, showing a well defined two-arm spiral structure. (The image has been inverted and converted to grey scale for clarity)

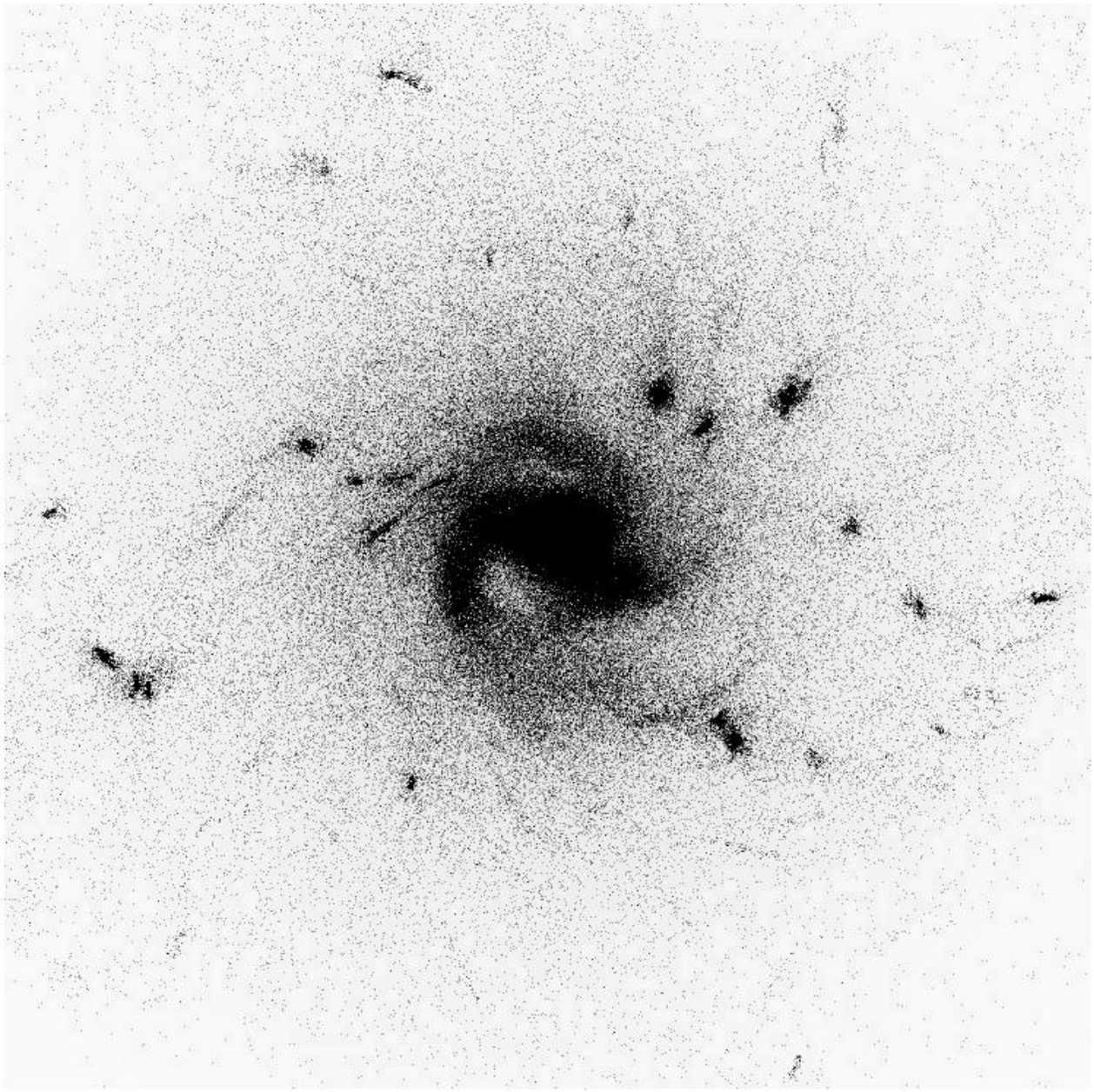
Varying the halo radius

Larger halos were tried, the results for a 10 Kpc radius halo (2/3 of the disc radius) are shown in Figure 15. Comparing this with the 5 Kpc halo of the same mass (shown in Figure 10), we can see that less expansion and break-up is seen with the larger radius halo.

Some galaxies are observed to have halos smaller than 1/3 of the disc radius. The effect of reducing the halo size was investigated, although using very small halos was found to cause problems because particles in the centre of the disc were given extremely high velocities.



[a] Figure 15 – Evolution of galactic disc with a 10 Kpc halo with 1x disc mass



[a] Figure 16 – High-resolution view of the galaxy from Figure 15 at 1000 MYrs showing more detail in the central regions. The magnitude of the gravitational field is shown as levels of grey.

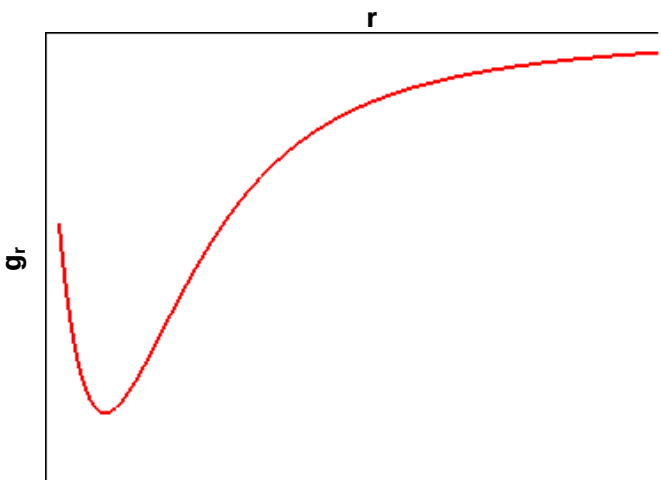
Non-uniform density halo

A better model for a galactic halo would involve the density decreasing gradually at higher radii, much like a globular cluster or elliptical galaxy. The observed luminosity of most elliptical galaxies falls off approximately exponentially with radius, so a new halo model will be constructed with a density distribution:

$$r_h \propto \exp(-r/R_c) \qquad \textbf{(0)}$$

If we require that the total mass of the halo be Mh, then this becomes:

$$r_h = \frac{M_h}{8pR_c^3}\exp(-r/R_c) \qquad \textbf{(0)}$$



a Figure 17 – Field due to exponential density halo.

The gravitational field due to this density distribution looks like Figure 17.

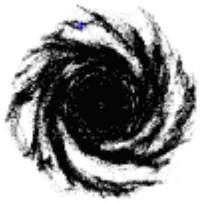
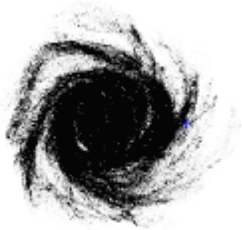
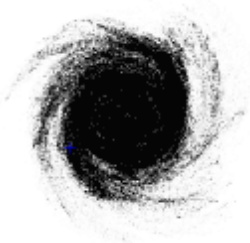
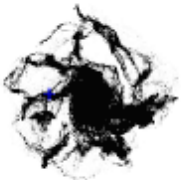
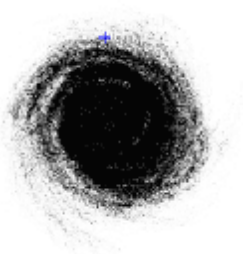
The "CalcHaloField" function in the program was updated to use this type of halo. See appendix for the complete program listing.

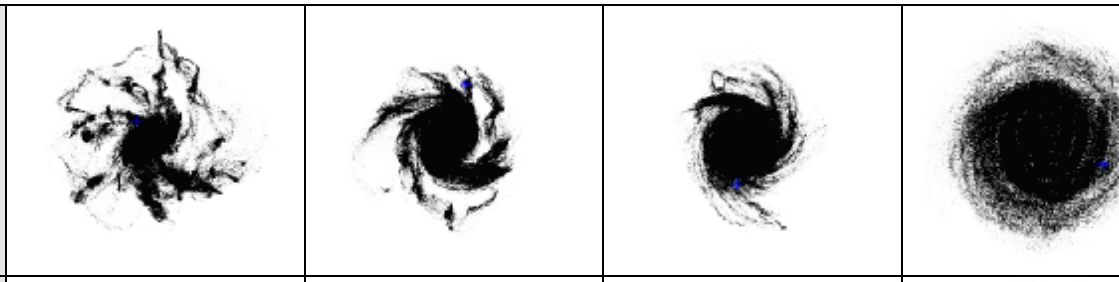
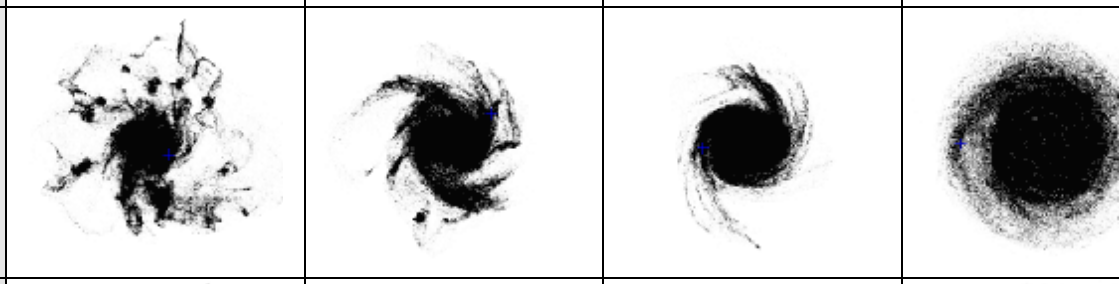
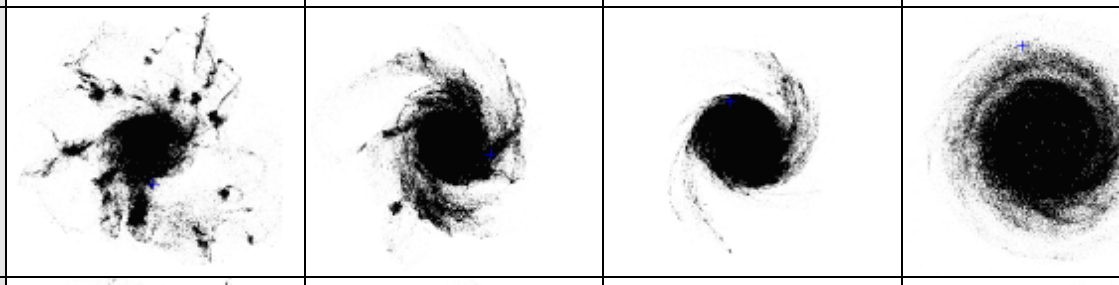
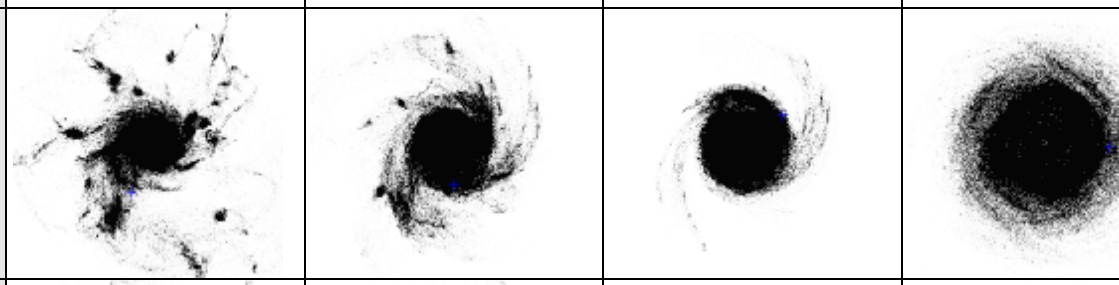
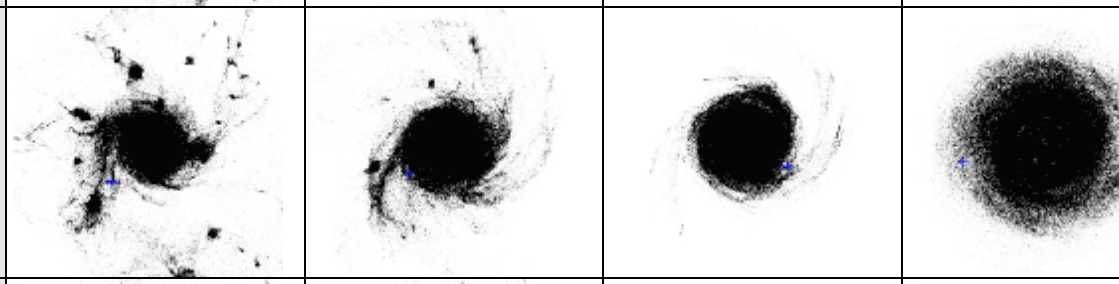
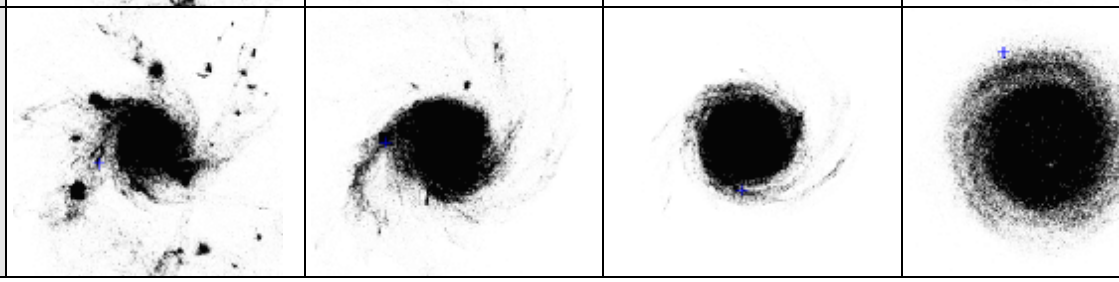
A number of simulations with different values of Mh and Rc were tried using this halo model. It was found that small values of Rc could be used without introducing instabilities into the simulation.

Results

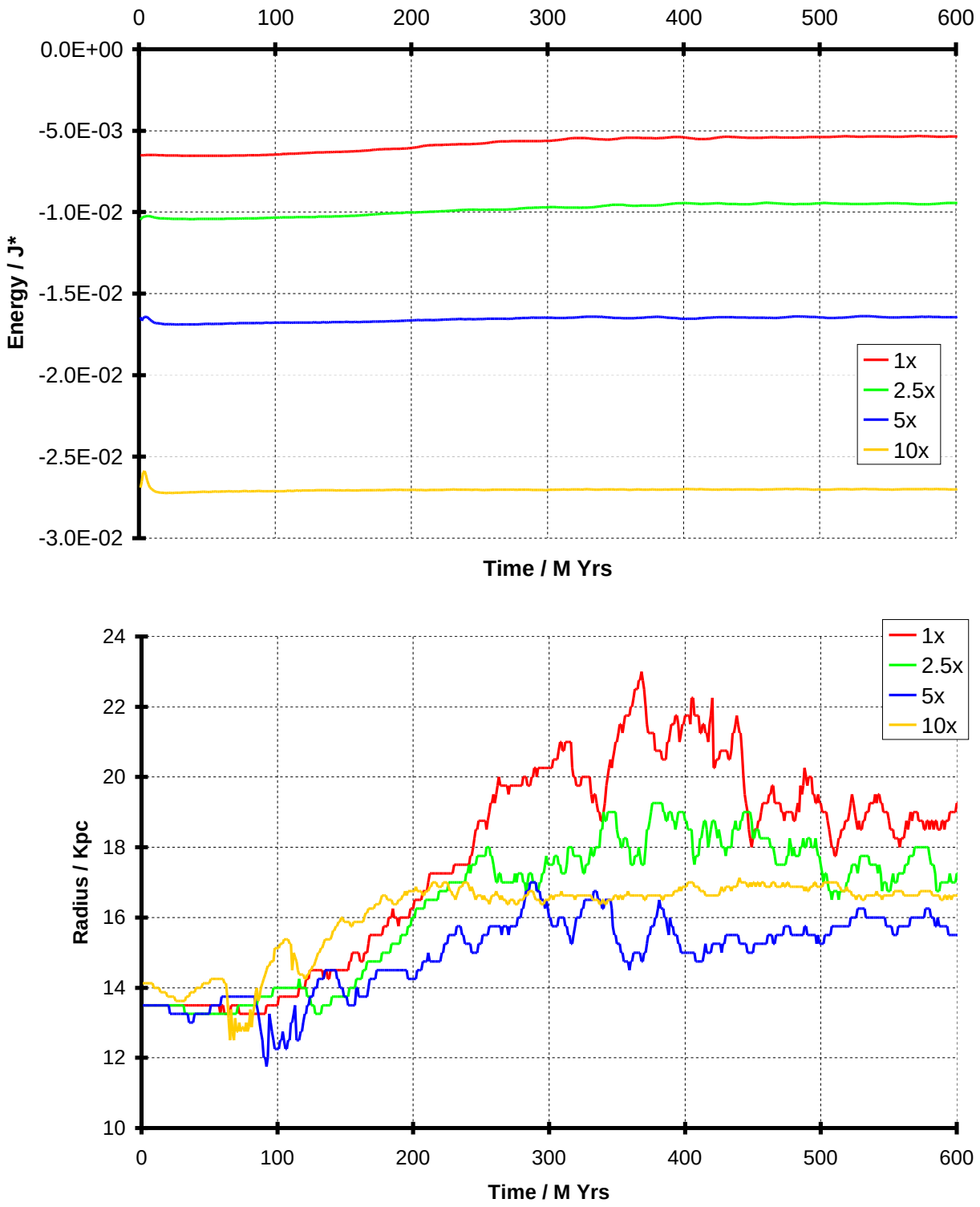
The results for a Rc=5 Kpc halo with masses 1x, 5x and 10x the disc mass were tried as in the previous section, and the results were found to be quite similar. Next, Rc was reduced to 1 Kpc, giving a smaller halo, and masses of 1x, 2.5x, 5x and 10x the disc mass were tried. The results for these simulations are shown side by side in Figure 18. In the 10x case, the disc did not seem to expand as it did with the others so it was run again on a smaller mesh size (with the same number of cells) to obtain increased accuracy.

Figure 19 shows the variation of total energy and galactic radius over time for these four simulations. It is interesting to see that the 10x halo system is the most accurate in terms of conservation of energy and also settles down to a fairly constant radius over time.

Time	Halo mass			
	1x disc mass ($1x10^{11}$ M_{SUN})	2.5x disc mass ($2.5x10^{11}$ M_{SUN})	5x disc mass ($5x10^{11}$ M_{SUN})	10x disc mass ($1x10^{12}$ M_{SUN})
50 MYrs				
100 MYrs				
150 MYrs				
200 MYrs				
250 MYrs				
300 MYrs				

350 MYrs				
400 MYrs				
450 MYrs				
500 Myrs				
550 Myrs				
600 Myrs				

a Figure 18 – Evolution of galactic discs for different values of the halo mass for small exponential halos. The disc mass was kept constant at $1x10^{11}$ M_{SUN}. The 10x disc mass simulation was done with a smaller mesh because it did not expand from its original radius to a great extent, this is why is appears bigger in the illustration.



[a] Figure 19 – Total energy (top) and galaxy radius (bottom) vs. time plots for small exponential halos of 1, 2.5, 5 and 10 times the disc mass.

Conclusions

The results obtained by the simulation will be analysed and compared to existing galactic images in an attempt to ascertain the validity of the models used.

The initial galactic model was a disc containing N particles, each representing a large number of stars. Initially the particles were given just enough rotational velocity to balance the disc, this was confirmed by the disc radius staying constant for approximately 150 MYrs. The disc was then seen to break up into several elliptical clusters, which appeared to be stable against further break up. A 3-dimensional model of this system would probably confirm that globular clusters and elliptical galaxies can be formed in this way, however, the flattened disc systems seen in spiral galaxies was the primary focus of this project.

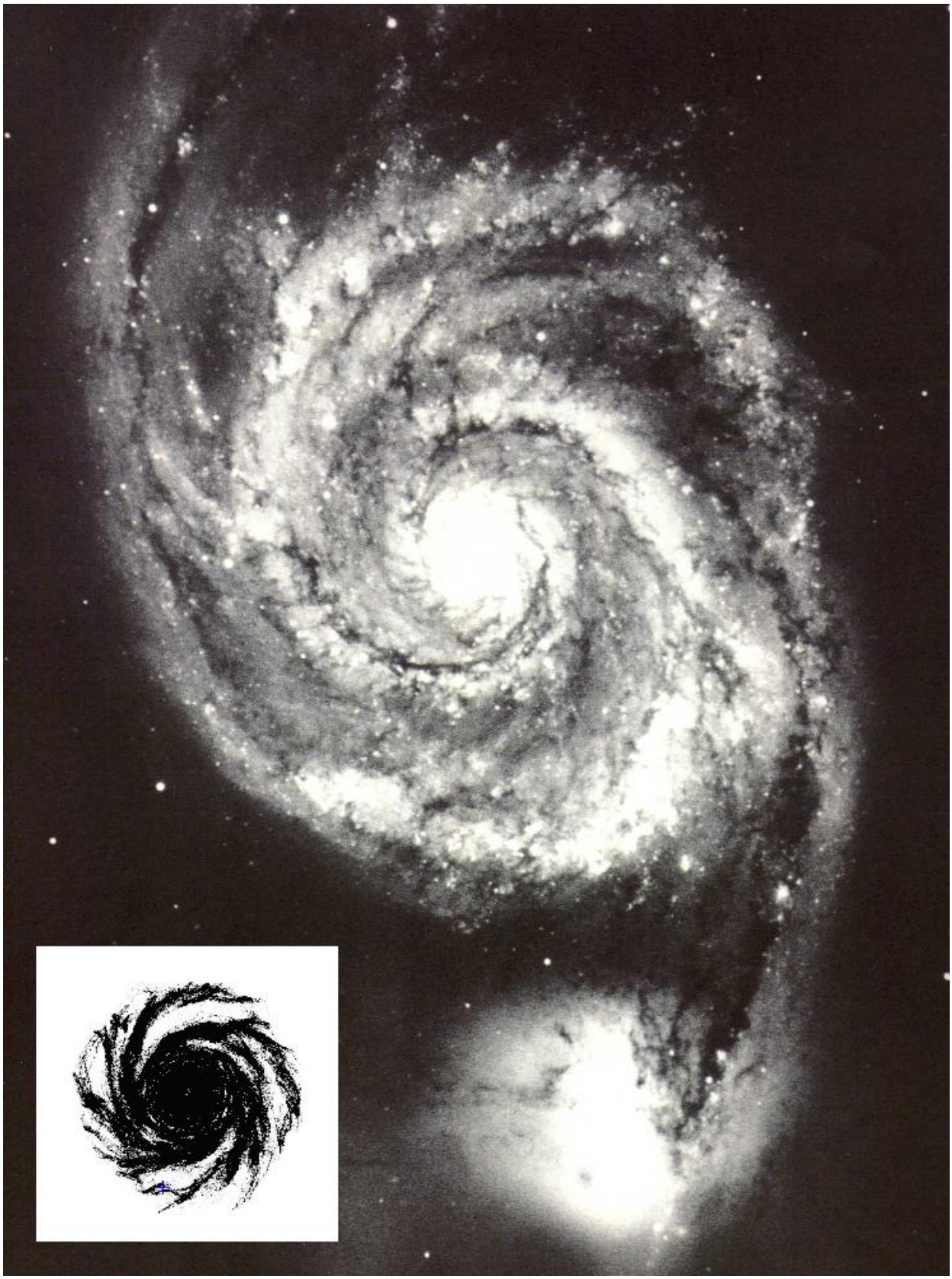
Putting a greater proportion of the disc mass in the centre was seen to increase the stability of the disc. In an attempt to model the halos seen in most spiral galaxies, the field due to a fixed solid sphere of uniform density was added to the model. By increasing the mass of this halo, the disc could be prevented from breaking up. Different halo masses were tried, masses of around 5x the disc mass were found to produce the most prominent spiral structure. A halo with density decreasing exponentially with radius was then used and produced better results for more centrally condensed halos.

When spiral structures were observed, they were seen to form quickly, within one disc rotation. They were also relatively short-lived, none were seen to last for more than about half a billion years, whereas spiral structures in real galaxies must have lasted for twenty times as long if our theories on the age of the universe are correct.

[a] Figure 20 – Galaxy M81, Sb type spiral with a large diffuse halo

None of the spirals produced were as regular as that exhibited by most of the spiral galaxies that we can observe, such as M81 (see Figure 20). Real spiral galaxies seem to have no problem retaining a regular, well defined structure,

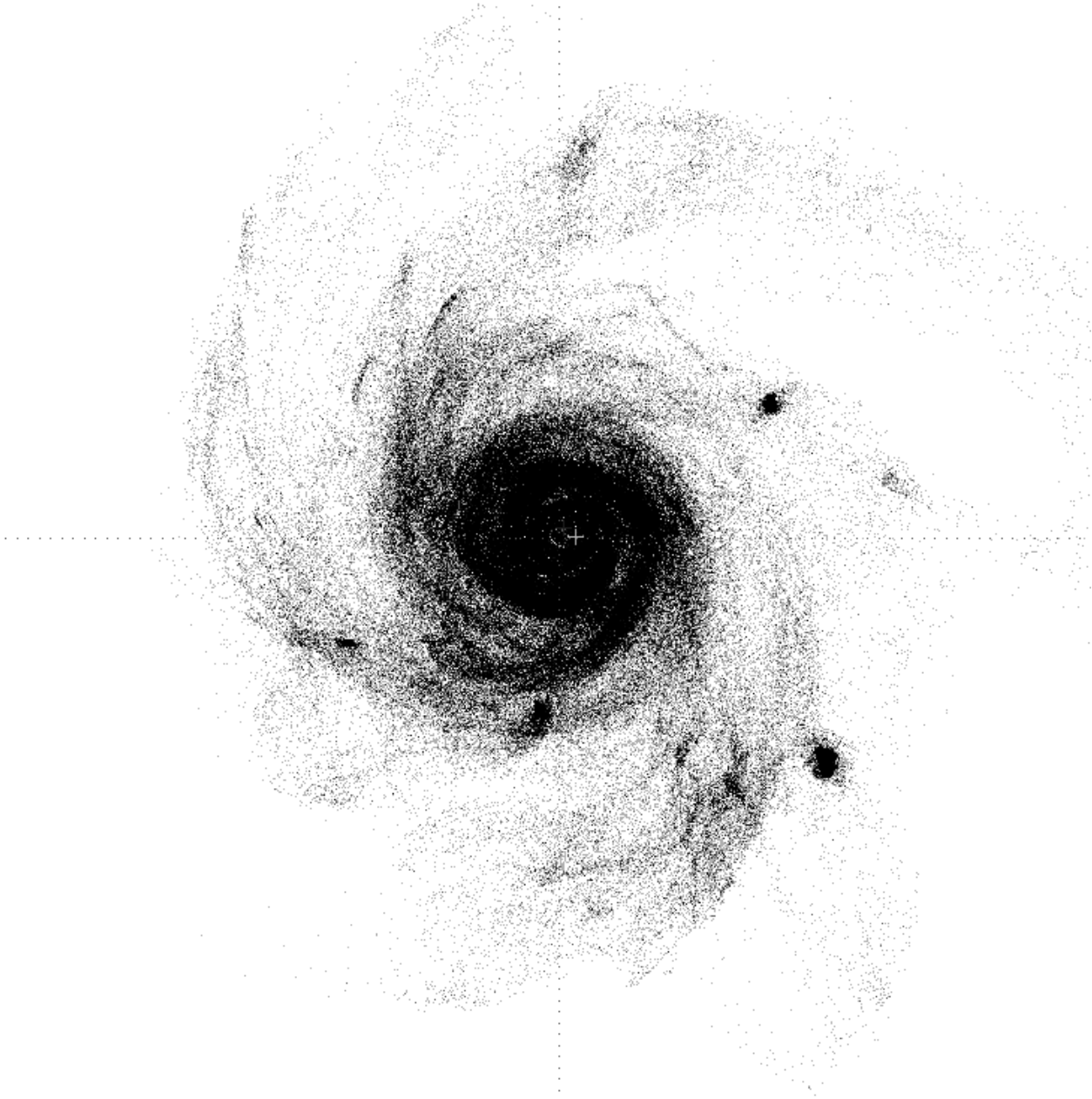
even in the event of having large companion galaxies, such as M51 in which would exert a considerable gravitational force on its neighbour.



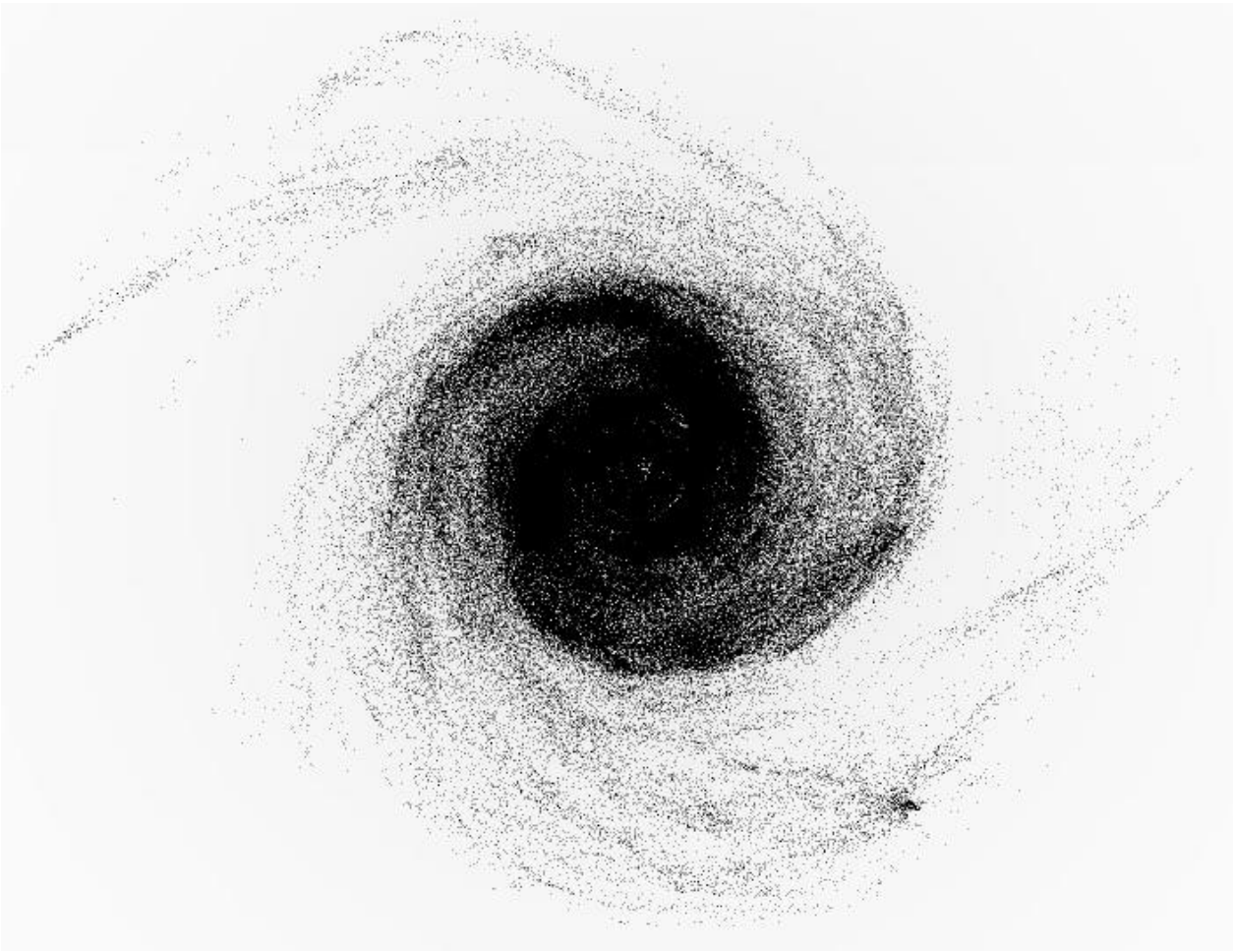
[a] Figure 21 – (Main image) Galaxy M51, Sc type spiral with small halo and companion galaxy, (inset) computer simulation

[a] Figure 22 – (top) Galaxy M101, Sc type spiral with small halo (bottom) computer simulation

There is, however an extremely wide range of galaxy types observable, and some comparisons can be made with the results obtained, see Figure 21 and Figure 22.

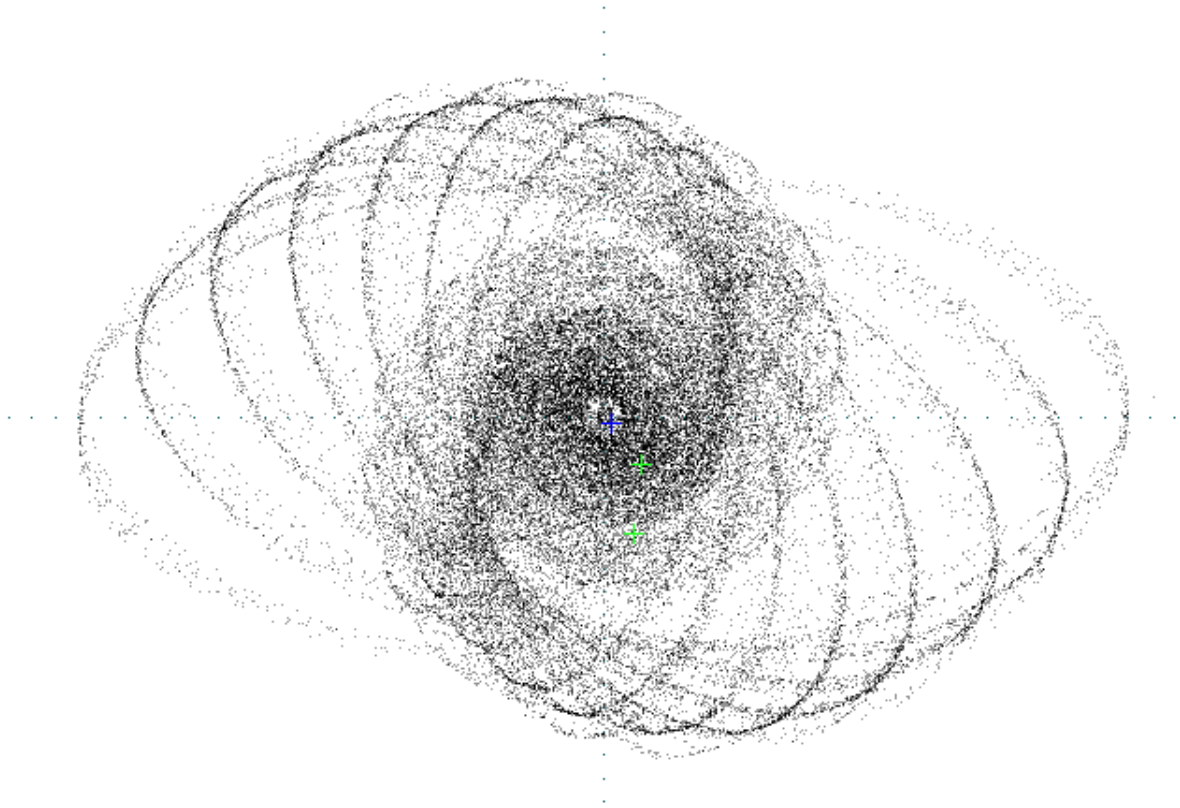


[a] Figure 23- Simulation result also comparable to observable galaxies



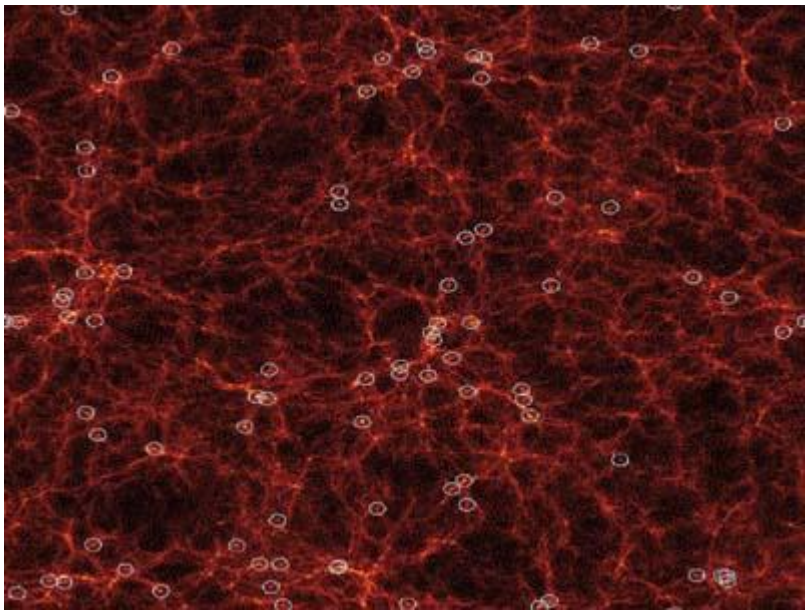
a Figure 24- Another result comparable to observable galaxies.

A large number of simulations were performed with a variety of initial conditions, too numerous to describe in detail here. An interesting result was obtained using a highly elliptical initial disc, and produced the output shown in Figure 25, but unfortunately time did not allow the investigation to go any further than this.



[a] Figure 25- Initial result obtained using a highly elliptical initial disc

This is interesting because the spiral structure seems to be formed by bands of particles flowing around the system in different orbits, the spiral was seen to last approximately twice as long as in the previous models. Universe simulations have been performed, such as that shown in Figure 26, and seem to show galaxies forming when large amount of matter condense to form a halo, with elongated structures between them. It may be these that play some part in the forming of stable spiral arms.



[a] Figure 26- Results of a universe simulation, from [1]

It is clear that there is a lot we do not yet understand about galactic formation. Observations are of little use on our short time scales, except perhaps with more powerful telescopes that can see galaxies earlier on in their evolution. The easiest and quickest way to rigorously test such a theory is by a computer simulation such as this one. Even though nothing new has been discovered here (similar results were observed in 1968 by Hohl [6]), the development of better simulations, galactic theories and models will bring us one step closer to understanding how galaxies are formed.

Appendix

Program listing

galaxy.cpp

```
// ------------------------------------------------
//          SIMULATION OF A GALAXY
// Using Nearest Grid Point / Particle-Mesh method
// ------------------------------------------------
//
// galaxy.cpp
//
// Written by Andrew Wedgbury
// Copyright ©1999
// All Rights Reserved
//

#include <windows.h>
#include <commctrl.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <io.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <ddraw.h>
#pragma hdrstop

#include "resource.h"

//
// WINDOWS INTERFACE DATA
//
HINSTANCE    hInst;                                 // INSTANCE HANDLE
HWND         hDispWin,hControlWin;                  // WINDOW HANDLES
HANDLE       hCalcThread;                           // CALCULATION THREAD HANDLE

//
// SIMULATION DATA TYPES
//
typedef struct tagVECTOR                            // CARTESIAN VECTOR
{    double  x;                                     // X-COMPONENT
     double  y;                                     // Y-COMPONENT
} VECTOR;

typedef struct tagMESHCELL                          // MESH CELL STRUCTURE
{    VECTOR  g;                                     // GRAVITATIONAL FIELD IN X AND Y [L/TT]
     double  m;                                     // TOTAL MASS IN CELL [M]
} MESHCELL;

typedef struct tagSTAR                              // STAR STRUCTURE
{    VECTOR  r;                                     // POSITION OF STAR [L]
     VECTOR  V;                                     // VELOCITY OF STAR [L/T]
     double  m;                                     // MASS OF STAR [M]
     int             c;                             // TYPE OF STAR
} STAR;

typedef struct tagPIXEL                             // PIXEL OR GENERAL int X,Y STRUCTURE
{
     WORD    x,y;
} PIXEL;

//
// PHYSICAL CONSTANTS AND UNITS
//
#define      M              (double)1.99e30                // Mass unit    (solar mass)
#define      L              (double)3.08e19                // Length unit  (Kiloparsec)
#define      T              (double)3.15569259747e7        // Time unit    (Year)
const double G              = 6.67259e-11 * T*T*M/(L*L*L); // Gravitational constant [LLL/TTM]
#define      PI             3.14159265359                  // PI
const double TWOPI          =2.0*PI;                       // PRECALCULATED 2*PI

//
// PARTICLE DATA
//
#define      NMAX           250000          // MAXIMUM NUMBER OF STARS
int          N              = 50000;        // NUMBER OF STARS IN SIMULATION
double       D              = 1.0e6;        // TIMESTEP [T]
DWORD        t              = 0;            // TIME [steps]
double       R0             = 15.0;         // INITIAL DISK RADIUS [L]
```

```
double      Rc              = R0/4.0;       // INITIAL CENTRAL HALO RADIUS [L]
double      MGX             = 1e11;         // MASS oF GALAXY [M]
double      M0              = MGX/N;        // STAR MASS [M]
double      Mh              = 0;            // CENTRAL HALO MASS [M]
double      VDISP           = 0.0e-8;       // VELOCITY DISPERSION [L/T]
STAR        stars[NMAX];                    // STARS ARRAY
int         nDiscType       = 5;            // INITIAL DISC TYPE (see SetInitial)
int         nHaloType       = 0;            // HALO TYPE (see CalcHaloField)

//
// MESH DATA
//
#define     CMAX            256             // MAXIMUM EDGE DIMENSION (CELLS)
int         C               = 60;           // MESH EDGE DIMENSION (CELLS)
double      S               = 60.0;         // MESH SIZE [L]
double      d               = S/C;          // MESH SPACING [L]
double      CM              = (d*C/2);      // CENTER OF MESH [L]
MESHCELL    mesh[CMAX][CMAX];               // MESH ARRAY
double      RxCache[CMAX][CMAX];            // ARRAY FOR CACHED Rx VALUES
double      RyCache[CMAX][CMAX];            // ARRAY FOR CACHED Ry VALUES
PIXEL       meshused[CMAX*CMAX];            // STORES COORDINATES OF MESH CELLS BEING USED

//
// FILE OUTPUT DATA
//
int         nEdgeCollisions=0;              // COUNTER FOR EDGE COLLISIONS
int         nCellsUsed=0;                   // COUNTER FOR NUMBER OF CELLS OCCUPIED
char        szBuffer[512];                  // TEXT BUFFER FOR O/P

char        szDataFile[MAX_PATH];           // DATA O/P FILE NAME
double      OPINTDATA       = 0.0e7;        // O/P INTERVAL FOR CONFIG DATA [T] (0 to disable)
int         hDataFile       = NULL;         // DATA O/P FILE HANDLE
int         hGraphFile      = NULL;

double      OPINTBIN        = 0.0e6;        // O/P INTERVAL FOR BINARY DATA [T] (0 to disable)
int         hBinFile        = NULL;         // BINARY O/P FILE HANDLE
PIXEL       binout[NMAX];                   // BINARY O/P BUFFER

//
// GRAPHICS OUTPUT DATA
//
int         RESX            = 1024;         // SCREEN X RESOLUTION [pixels]
int         RESY            = 768;          // SCREEN Y RESOLUTION [pixels]
int         WINDOWX         = RESY-20;      // O/P WINDOW X DIMENSION [pixels]
int         WINDOWY         = RESY-20;      // O/P WINDOW Y DIMENSION [pixels]
int         SIDEX           = RESY;         // X POSITION OF SIDE INFO BAR [pixels]
double      xs              = WINDOWX/S;    // X SCALE [pixels/L]
double      ys              = WINDOWY/S;    // Y SCALE [pixels/L]
int         xt              = 0;            // X TRANSFORM
int         yt              = 0;            // Y TRANSFORM
double      vmax            = 0;            // SCALE FOR VELOCITY GRAPH
double      dmax            = 0;            // SCALE FOR DENSITY GRAPH
double      dbins[400];                     // RADIAL DENSITY ARRAY
double      fmax            = 0;            // SCALE FOR FIELD COLOURING
#define     NIS             2               // NUMBER OF INDICATOR STARS
VECTOR      Ir[NIS];                        // INDICATOR STAR POSITIONS
double      nRevs[NIS];                     // NUMBER OF REVOLUTIONS OF INDICATORS
double      Ep              = 0.0;          // TOTAL POTENTIAL ENERGY ACCUMULATOR
double      Ek              = 0.0;          // TOTAL KINETIC ENERGY ACCUMULATOR

LPDIRECTDRAW                lpDD;           // DirectDraw OBJECT
LPDIRECTDRAWSURFACE         lpDDSPrimary;   // DirectDraw PRIMARY SURFACE
LPDIRECTDRAWSURFACE         lpDDSBack;      // DirectDraw BACK SURFACE
DWORD                       dwS=0,dwF=0;    // DirectDraw INIT DATA
HFONT                       hBigFont;       // FONT USED FOR PROGRAM TITLE
HFONT                       hSmallFont;     // FONT USED FOR GENERAL TEXT

//
// SIMULATION CONTROL FLAGS
//
BOOL        bDrawField      = 0;            // DRAW FIELD FLAG
BOOL        bDrawStars      = 1;            // DRAW STARS FLAG
BOOL        bDrawText       = 1;            // DRAW TEXT PANEL FLAG
BOOL        bDrawIndic      = 1;            // DRAW INDICATOR STARS FLAG
BOOL        bDrawAxes       = 1;            // DRAW AXES FLAG
BOOL        bDrawHalo       = 0;            // DRAW HALO FLAG
BOOL        bPaused         = 0;            // PAUSE / GO FLAG


//
// FUNCTION PROTOTYPES
//
DWORD _cdecl CalcThread(LPDWORD);
void Timestep(void);
void CalcMesh(void);
void CheckBounds(int);
void OutputData(void);
```

```
void WriteDataFile(void);
void SetInitial(void);
void CalcHaloField(double,double,VECTOR*);
double CalcEnergy(int);
void InitRCache(void);
double LookupRx(int,int);
double LookupRy(int,int);
double radius(double,double);
inline void WDXSetPixel(LPDDSURFACEDESC lpddsd,int sx,int sy,int red,int green,int blue);
inline void WDXIncPixel(LPDDSURFACEDESC lpddsd,int sx,int sy,int red,int green,int blue);
void PaintText(HDC hDC,int x,int y,LPCSTR text,COLORREF col);


//
// THREAD: CalcThread
//  Performs calculation in background
//
DWORD _cdecl CalcThread(LPDWORD lpdwParam)
{
     while (1)
     {
            Sleep(0);                       // Relinquish time slice if required by system
            if (!bPaused)
                   Timestep();              // Run one timestep if not paused
     }
     return 0;
}


//
// FUNCTION: Timestep
//  Performs one timestep of the simulation
//
void Timestep(void)
{
     //
     // Save calc start time
     //
     dwS= GetTickCount();

     //
     // Set initial conditions if t=0
     //
     if (t==0)
     {
            SetInitial();
            OutputData();
     }

     //
     // Calculate Gravitational field over Mesh
     //
     else
            CalcMesh();

     //
     // Increment timestep counter
     //
     t++;

     //
     // Loop over all stars
     //
     int is; Ek=0.0;
     for (is=0; is<N; is++)
     {
            //
            // Check star is within mesh
            //
            CheckBounds(is);

            //
            // Get nearest grid point
            //
            int cx = (int)(stars[is].r.x / d);
            int cy = (int)(stars[is].r.y / d);

            if ( ((cx >= 0 ) && (cx < C)) && ((cy >= 0) && (cy < C)) )
            {
                   VECTOR g;
                   CalcHaloField(stars[is].r.x,stars[is].r.y,&g);

                   //
                   // Update particle velocities from halo and mesh
                   //
                   stars[is].V.x += (mesh[cx][cy].g.x + g.x) * D;
                   stars[is].V.y += (mesh[cx][cy].g.y + g.y) * D;
```

```
                        Ek += CalcEnergy(is);
                }

                //
                // Update particle positions
                //
                stars[is].r.x += stars[is].V.x *D;
                stars[is].r.y += stars[is].V.y *D;
        }

        //
        // Save calc end time
        //
        dwF = GetTickCount();

        //
        // Output data to screen and files
        //
        OutputData();
}


//
// FUNCTION: random
//  Returns a random integer between 0 and max
//  (note: has a resolution of 32768)
//
inline int random(int max)
{
        return MulDiv(rand(),max,32768);
}


//
// FUNCTION drandom
//  Returns a random double between 0 and max
//
inline double drandom(double max)
{
        return max * rand()/32768.0;
}


//
// FUNCTION gausdev
//  Returns a gaussian deviate using Box Muller algorithm
//
double gausdev(void)
{
        double rsq,x,y;

        do
        {
                x = drandom(2.0)-1.0;
                y = drandom(2.0)-1.0;
                rsq = x*x + y*y;

        }
        while ((rsq >= 1.0) || (rsq == 0.0));

        return x*sqrt(-2.0*log(rsq)/rsq);
}


//
// FUNCTION: radius
//  Converts x,y to r using pythagoras' theorem
//
inline double radius(double x, double y)
{
        return sqrt((x*x)+(y*y));
}


//
// INITIAL CONDITIONS TEMPOARY DATA
//  (Too big for local storage)
//
double r[NMAX];
double theta[NMAX];


//
// FUNCTION: SetInitial
//  Setup initial conditions for stars
//
```

```
void SetInitial(void)
{
     int is;
     double u;

     //
     // Assign variables
     //
     M0 = (double)MGX / N;

     //
     // Open files
     //
     if (hDataFile)
            _close(hDataFile);
     if (hBinFile)
            _close(hBinFile);
     if (hGraphFile)
            _close(hGraphFile);

     char szFileName[MAX_PATH];
     if (OPINTDATA)
            lstrcpy(szFileName,szDataFile);
            lstrcat(szFileName,"_t.csv");
            hDataFile = _open(szFileName,_O_WRONLY | _O_CREAT | _O_TRUNC,_S_IREAD | _S_IWRITE);
     if (OPINTBIN)
     {
            lstrcpy(szFileName,szDataFile);
            lstrcat(szFileName,"_g.csv");
            hGraphFile = _open(szFileName,_O_WRONLY | _O_CREAT | _O_TRUNC,_S_IREAD | _S_IWRITE);
     }

     //
     // Set initial star postions
     //
     for (is=0; is<N; is++)
     {
            //
            // Star mass & type
            //
            stars[is].m  = M0;
            stars[is].c = 0;

            //
            // Generate r
            //
            switch (nDiscType)
            {
                   case 1:                                  // UNIFORM SPATIAL DISTRIBUTION
                          r[is] = R0*sqrt(drandom(1));
                          break;

                   case 2:                                  // UNIFORM r
                          r[is] = drandom(R0);
                          break;

                   case 3:                                  // RING
                          r[is] = R0;
                          break;

                   case 4:                                  // GAUSSIAN DENSITY DISTRIBUTION
                          do
                          {
                                 double rand = gausdev();
                                 r[is] = (R0/4.0)*sqrt(sqrt(rand*rand));
                          } while (r[is] > R0);
                          break;

                   case 5:                                  // μ = μ0 sqrt(1 - (r/R)²)
                          do
                          {
                                 u = drandom(1.0);
                                 r[is] = drandom(1.0);
                          } while ( u > sqrt(1.0-(r[is]*r[is])) );
                          r[is] = R0*sqrt(r[is]);
                          break;
            }

            //
            // Generate random theta between 0 and 2*PI
            //
            theta[is] = drandom(TWOPI);

            //
            // Set star cartesian position from r and theta
            //
            stars[is].r.x = CM + (r[is] * cos(theta[is]));
```

```
                stars[is].r.y = CM + (r[is] * sin(theta[is]));
        }

        //
        // Centre star
        //
        theta[0]=0;
        r[0]=0;
        stars[0].r.x = CM;
        stars[0].r.y = CM;
        stars[0].c = 3;

        //
        // Indicator stars
        //
        theta[1]=-PI/2;
        r[1]=Rc;
        stars[1].r.x = Ir[0].x = CM;
        stars[1].r.y = Ir[0].y = CM-r[1];
        stars[1].c = 3;

        theta[2]=-PI/2;
        r[2]=R0;
        stars[2].r.x = Ir[1].x = CM;
        stars[2].r.y = Ir[1].y = CM-r[2];
        stars[2].c = 3;

        //
        // Calculate initial forces to get balancing velocity
        //
        CalcMesh();

        for (is=0; is<N; is++)
        {
                int cx = (int)(stars[is].r.x / d);
                int cy = (int)(stars[is].r.y / d);

                //
                // Check particle is within mesh
                //
                if ( ((cx >= 0 ) && (cx < C)) && ((cy >= 0) && (cy < C)) )
                {
                        //
                        // Calculate required centripetal velocity and set
                        //
                        VECTOR g;
                                                // BALANCE DISC
                        CalcHaloField(stars[is].r.x,stars[is].r.y,&g);
                        double a = radius(g.x+mesh[cx][cy].g.x,g.y+mesh[cx][cy].g.y);
                        double F = 1.0 * sqrt(a*r[is]);

                        stars[is].V.x = F * -sin(PI-theta[is]);
                        stars[is].V.y = F * -cos(PI-theta[is]);
                }

                //
                // Add gaussian velocity dispersion if required
                //
                if (VDISP)
                {
                        stars[is].V.x += (r[is]/R0)* VDISP * gausdev();
                        stars[is].V.y += (r[is]/R0)* VDISP * gausdev();
                }
        }
}


//
// FUNCTION: CalcHaloField
//  Calculate field due to halo
//
void CalcHaloField(double x,double y,VECTOR* g)
{
        g->x=0;
        g->y=0;

        if (Mh==0.0 || r<d)
                return;

        double R = radius(x-CM,y-CM);
        double f = 0.0;
        double Ru = (x-CM) / R;
        double Rv = (y-CM) / R;
```

```
	switch (nHaloType)
	{
		case 1:
			if (R > Rc)				// SOLID SPHERE
				f = -G*Mh / (R*R);
			else
				f = -G*Mh*R / (Rc*Rc*Rc);
			break;

		case 2:					// EXPONENTIAL DENSITY SPHERE
			double t1 = Rc*Rc;
			double t4 = R*R;
			double t7 = exp(-R/Rc);
			f = 1/t4*Mh*(t7*t4+2.0*Rc*t7*R+2.0*t1*t7-2.0*t1)/t1*G/2.0;
			break;
	}
	g->x = Ru * f;
	g->y = Rv * f;
}


//
// FUNCTION CalcMesh
//  Calculate field at mesh points
//
void CalcMesh()
{
	int iu,iv,ix,iy,is;

	// Zero radial density array
	ZeroMemory(&dbins,sizeof(dbins));
	Ep = 0.0;

	// Reset cells used counter
	nCellsUsed=0;

	//
	// Set all mesh cells to zero mass and field
	//
	for (iu=0; iu<C; iu++)
	{
		for (iv=0; iv<C; iv++)
		{
			mesh[iu][iv].m = 0;
			mesh[iu][iv].g.x = 0;
			mesh[iu][iv].g.y = 0;
		}
	}

	//
	// Get mass of stars in each mesh cell
	//
	for (is=0; is<N; is++)
	{
		//
		// Get mesh cell containing star
		//
		int cx = (int)(stars[is].r.x / d);
		int cy = (int)(stars[is].r.y / d);

		//
		// Make sure star is inside mesh
		//
		if ( ((cx >= 0 ) && (cx < C)) && ((cy >= 0) && (cy < C)) )
		{
			// Add star on to mesh cell
			mesh[cx][cy].m += stars[is].m;

			// Add star on to radial density array
			dbins[(int)( 400*radius(stars[is].r.x-CM,stars[is].r.y-CM)/S)] +=
				stars[is].m;
		}
	}

	//
	// Run over mesh and save list of cells containing stars
	// (this saves time and allows bigger meshes)
	//
	for (iu=0; iu<C; iu++) // Loop over u
	{
		for (iv=0; iv<C; iv++) // Loop over v
		{
			if (mesh[iu][iv].m || bDrawField)
			{
```

```
                                meshused[nCellsUsed].x = iu;
                                meshused[nCellsUsed].y = iv;
                                nCellsUsed++;
                        }
                }
        }

        //
        // Calculate field at each (used) mesh point due to all other (used) mesh points
        //
        for (ix=0; ix<nCellsUsed; ix++)
        {
                for (iy=0; iy<nCellsUsed; iy++)
                {
                        // Don't include (u,v) cell
                        if (!((meshused[iy].x==meshused[ix].x) && (meshused[iy].y==meshused[ix].y)))
                        {
                                //
                                // Calc components of field at (u,v) due to mass at (x,y) and add on
                                //
                                mesh[meshused[ix].x][meshused[ix].y].g.x +=
                                        mesh[meshused[iy].x][meshused[iy].y].m *
                                        LookupRx(meshused[iy].x-meshused[ix].x,
                                                meshused[iy].y-meshused[ix].y);

                                mesh[meshused[ix].x][meshused[ix].y].g.y +=
                                        mesh[meshused[iy].x][meshused[iy].y].m *
                                        LookupRy(meshused[iy].x-meshused[ix].x,
                                                meshused[iy].y-meshused[ix].y);

                                //
                                // Calc potential energy
                                //
                                Ep += -G * mesh[meshused[ix].x][meshused[ix].y].m *
                                        mesh[meshused[iy].x][meshused[iy].y].m /
                                        radius(meshused[iy].x-meshused[ix].x,
                                                meshused[iy].y-meshused[ix].y);
                        }
                }
        }

        //
        // Calc potential energy due to halo
        //
        for (is=0; is<N; is++)
        {
                VECTOR g;
                CalcHaloField((double)stars[is].r.x,(double)stars[is].r.y,&g);
                Ep += -stars[is].m * radius(g.x,g.y) *
                        radius((double)stars[is].r.x-CM,(double)stars[is].r.y-CM);
        }
}


//
// FUNCTION: CheckBounds
//  checks to see if particle is in mesh,
//  if not, its velocity is reversed.
//
inline void CheckBounds(int is)
{
        if ( (stars[is].r.x <= 0) && (stars[is].V.x < 0) )
        {
                nEdgeCollisions++;
                stars[is].V.x *= -1;
        }

        if ( (stars[is].r.y <= 0) && (stars[is].V.y < 0) )
        {
                nEdgeCollisions++;
                stars[is].V.y *= -1;
        }

        if ( (stars[is].r.x >= S) && (stars[is].V.x > 0) )
        {
                nEdgeCollisions++;
                stars[is].V.x *= -1;
        }

        if ( (stars[is].r.y >= S) && (stars[is].V.y > 0) )
        {
                nEdgeCollisions++;
                stars[is].V.y *= -1;
        }
}
```

```
//
// FUNCTION: WDXSetPixel
//  Sets pixel on DD3 Surface
//
void WDXSetPixel(LPDDSURFACEDESC lpddsd,int sx,int sy,int red,int green,int blue)
{
//   sx+=10; sy+=10;
//   if ( ((sx>0) && (sy>0)) && ((sx<lpddsd->dwWidth) && (sy<lpddsd->dwHeight)) )
     if ( ((sx>0) && (sy>0)) && ((sx<RESX) && (sy<RESY)) )
     {
            UINT offs= (sy*lpddsd->lPitch)+(4*sx);
            ((BYTE *)lpddsd->lpSurface)[offs] = blue;
            ((BYTE *)lpddsd->lpSurface+1)[offs] = green;
            ((BYTE *)lpddsd->lpSurface+2)[offs] = red;
     }
}


//
// FUNCTION: WDXIncPixel
//  Increments pixel color value on DD3 Surface
//
void WDXIncPixel(LPDDSURFACEDESC lpddsd,int sx,int sy,int red,int green,int blue)
{
//   sx+=10; sy+=10;
//   if ( ((sx>0) && (sy>0)) && ((sx<lpddsd->dwWidth) && (sy<lpddsd->dwHeight)) )
     if ( ((sx>0) && (sy>0)) && ((sx<RESX) && (sy<RESY)) )
     {
            UINT offs= (sy*lpddsd->lPitch)+(4*sx);
            if (((BYTE *)lpddsd->lpSurface)[offs] + blue < 255)
                   ((BYTE *)lpddsd->lpSurface)[offs] += blue;
            else
                   ((BYTE *)lpddsd->lpSurface)[offs] = 255;

            if (((BYTE *)lpddsd->lpSurface+1)[offs] + green < 255)
                   ((BYTE *)lpddsd->lpSurface+1)[offs] += green;
            else
                   ((BYTE *)lpddsd->lpSurface+1)[offs] = 255;

            if (((BYTE *)lpddsd->lpSurface+2)[offs] + red < 255)
                   ((BYTE *)lpddsd->lpSurface+2)[offs] += red;
            else
                   ((BYTE *)lpddsd->lpSurface+2)[offs] = 255;
     }
}


//
// FUNCTION: PaintText
//  Paints text
//
void PaintText(HDC hDC,int x,int y,LPCSTR text,COLORREF col)
{
     SetTextColor(hDC,col);
     ExtTextOut(hDC,x,y,NULL,NULL,text,lstrlen(text),(LPINT)NULL);
}


//
// FUNCTION: OutputData
//  Send data to screen and files
//
void OutputData(void)
{
     int is;

     //
     // Calculate number of revolutions of indicator stars
     //
     for (is=0; is<NIS; is++)
     {
            if (Ir[is].x<CM && stars[is+1].r.x>CM)
                   nRevs[is] += 0.25;
            if (Ir[is].x>CM && stars[is+1].r.x<CM)
                   nRevs[is] += 0.25;
            if (Ir[is].y<CM && stars[is+1].r.y>CM)
                   nRevs[is] += 0.25;
            if (Ir[is].y>CM && stars[is+1].r.y<CM)
                   nRevs[is] += 0.25;
            Ir[is].x = stars[is+1].r.x;
            Ir[is].y = stars[is+1].r.y;
     }
```

```
	//
	// Graphics output
	//
	if (lpDDSBack)
	{
		//
		// Erase background before drawing starts
		//
		DDBLTFX ddbfx;
		ZeroMemory(&ddbfx,sizeof(ddbfx));
		ddbfx.dwSize = sizeof(ddbfx);
		ddbfx.dwFillColor = RGB(0,0,0);
		lpDDSBack->Blt(NULL,NULL,NULL,DDBLT_WAIT | DDBLT_COLORFILL,&ddbfx);

		//
		// Draw halo
		//
		if (bDrawHalo)
		{
			HDC hDC;
			if (SUCCEEDED( lpDDSBack->GetDC(&hDC) ))
			{
				SetBkColor(hDC,RGB(0,0,0));
				SetBkMode(hDC,TRANSPARENT);

				SelectObject(hDC,GetStockObject(WHITE_BRUSH));
				SelectObject(hDC,GetStockObject(WHITE_PEN));
				Ellipse(hDC,xs*(CM-Rc),ys*(CM-Rc),xs*(CM+Rc),ys*(CM+Rc));

				lpDDSBack->ReleaseDC(hDC);
			}
		}

		//
		// Lock surface for direct graphics operations
		//
		DDSURFACEDESC ddsd;
		ddsd.dwSize = sizeof(ddsd);
		if (SUCCEEDED( lpDDSBack->Lock(NULL,&ddsd,DDLOCK_SURFACEMEMORYPTR |
				DDLOCK_WAIT,NULL) ))
		{
			int x,y;

			//
			// Draw field intensities
			//
			if (bDrawField)
			{
				int iu,iv,ix,iy;
				for (iu=0; iu<C; iu++) // Loop over u
				{
					for (iv=0; iv<C; iv++) // Loop over v
					{
						VECTOR g;
						CalcHaloField((double)(iu+0.5)*d,
							(double)(iv+0.5)*d,&g);
						double f = sqrt(radius(g.x+mesh[iu][iv].g.x,
								g.y+mesh[iu][iv].g.y) );
						if ((f > fmax) || (fmax==0.0))
							fmax = f;
						f = f*200/fmax;
						if (t)
						{
							for (ix=0; ix<(xs*d); ix++)
							{
								for (iy=0; iy<(ys*d); iy++)
								{

								WDXSetPixel(&ddsd,xt+(iu*xs*d)+ix,
									yt+(iv*ys*d)+iy,f,0,0);
								}
							}
						}
					}
				}

				//
				// Draw stars
				//
				for (is=0; is<N; is++)
				{
					int colR=255,colG=255,colB=255;

					switch (stars[is].c)
					{
						default:
```

```
                                colR=80; colG=80; colB=80;
                                break;
                        case 1:
                                colR=0; colG=255; colB=0;
                                break;
                        case 2:
                                colR=0; colG=0; colB=255;
                                break;
                        case 3:
                                colR=255; colG=0; colB=0;
                                break;
                        case 4:
                                colR=100; colG=255; colB=255;
                                break;
                }

                //
                // Draw star on galaxy display
                //
                if (bDrawStars)
                {
                        x = (int)(xs*stars[is].r.x);
                        y = (int)(ys*stars[is].r.y);
                        WDXIncPixel(&ddsd,xt+x,yt+y,colR,colG,colB);
                        if (OPINTBIN)
                        {
                                binout[is].x = LOWORD((WORD)x);
                                binout[is].y = LOWORD((WORD)y);
                        }

                }

                //
                // Draw this star on total velocity - radius graph
                //
                if (bDrawText)
                {
                        double v = radius(stars[is].V.x,stars[is].V.y);
                        if ((v>vmax) || (vmax==0)) vmax = v;
                        x = SIDEX + (int)(radius(stars[is].r.x-CM,
                                        stars[is].r.y-CM)*400.0/S);
                        y = 400 - (int)abs(120*v/vmax);
                        WDXIncPixel(&ddsd,x,y,colR,colG,colB);
                }
        }

        if (bDrawIndic)
        {
                //
                // Draw central star
                //
                x = (int)(xs*stars[0].r.x);
                y = (int)(ys*stars[0].r.y);
                for (is=-5; is<=5; is++)
                        WDXSetPixel(&ddsd,xt+x+is,yt+y,255,255,0);
                for (is=-5; is<=5; is++)
                        WDXSetPixel(&ddsd,xt+x,yt+y+is,255,255,0);

                //
                // Draw revolution indicator star
                //
                for (int it=0; it<NIS; it++)
                {
                        x = (int)(xs*stars[it+1].r.x);
                        y = (int)(ys*stars[it+1].r.y);
                        for (is=-5; is<=5; is++)
                                WDXSetPixel(&ddsd,xt+x+is,yt+y,192,0,192);
                        for (is=-5; is<=5; is++)
                                WDXSetPixel(&ddsd,xt+x,yt+y+is,192,0,192);
                }
        }

        //
        // Draw density - radius graph from dbins
        //
        if (bDrawText)
        {
                for (is=1; is<200; is++)
                {
                        double density = dbins[is]*400.0/(is*d*S);
                        if ((density>dmax) || (dmax==0))
                                        dmax = density;
                        for (int iy=(int)(120*density/dmax); iy>0; iy--)
                                if (iy < 120)
                                        WDXSetPixel(&ddsd,SIDEX+is,
                                                560-iy,192,192,192);
                }
```

```
                        for (is=0; is<200; is++)
                                WDXSetPixel(&ddsd,SIDEX+is,560,128,128,255);
                        for (is=0; is<120; is++)
                                WDXSetPixel(&ddsd,SIDEX,560-is,128,128,255);
                }

                //
                // Draw axes for galaxy view
                //
                if (bDrawAxes)
                {
                        for (is=1; is<C; is++)
                                WDXSetPixel(&ddsd,xt+(int)(xs*d*(0.5+is)),
                                        yt+(int)(ys*d*C/2),255,128,128);
                        for (is=1; is<C; is++)
                                WDXSetPixel(&ddsd,xt+(int)(xs*d*C/2),
                                        yt+(int)(ys*d*(0.5+is)),255,128,128);
                }

                //
                // Draw axes for total velocity graph
                //
                if (bDrawText)
                {
                        for (is=0; is<200; is++)
                                WDXSetPixel(&ddsd,SIDEX+is,400,128,128,255);
                        for (is=0; is<120; is++)
                                WDXSetPixel(&ddsd,SIDEX,400-is,128,128,255);
                }

                lpDDSBack->Unlock(NULL);
        }

        //
        // Draw side info bar
        //
        if (bDrawText)
        {
                HDC hDC;
                if (SUCCEEDED( lpDDSBack->GetDC(&hDC) ))
                {
                        SetBkColor(hDC,RGB(0,0,0));
                        SetBkMode(hDC,TRANSPARENT);

                        SelectObject(hDC,hBigFont);
                        PaintText(hDC,SIDEX,10,"Galaxy Simulation Program",RGB(255,0,0));

                        SelectObject(hDC,hSmallFont);
                        PaintText(hDC,SIDEX,32,"Written by Andrew Wedgbury",RGB(255,0,0));
                        PaintText(hDC,SIDEX,50,"aw@wedger.demon.co.uk",RGB(255,0,0));

                        #define TS 20
                        int     cl=4;

                        sprintf(szBuffer,"Time: %.2f Myrs (%d steps)",(t*D)/1e6,t);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Timestep: %.2f Myrs (%d ms)",D/1e6,dwF-dwS);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Mesh: %dx%d cells (%d%% used)  ",
                                C,C,(int)(100.0*nCellsUsed/((C-1)*(C-1))));
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Mesh size: %.fx%.f Kpc (%.2f Kpc/cell)",S,S,d);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Particles: %d (%d hits)",N,nEdgeCollisions);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Star mass: %.2e Msun",M0);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Initial Radius: %.2f Kpc sqrt(1-r²/R²)",R0);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Halo: %.2f Kpc (%.2e Msun)",Rc,Mh);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"Revs: %.2f, %.2f",nRevs[0],nRevs[1]);
                        PaintText(hDC,SIDEX,TS*cl++,szBuffer,RGB(192,192,255));

                        cl=0;
                        if (bPaused)
                        {
                                sprintf(szBuffer,"<PAUSED>");
                                PaintText(hDC,SIDEX,600+(TS*cl++),szBuffer,RGB(255,0,0));
```

```
                        }

                        sprintf(szBuffer,"Ek: %.5e, Ep: %.5e",Ek,Ep);
                        PaintText(hDC,SIDEX,600+(TS*cl++),szBuffer,RGB(192,192,255));

                        sprintf(szBuffer,"TOTAL ENERGY: %.5e",Ek+Ep);
                        PaintText(hDC,SIDEX,600+(TS*cl++),szBuffer,RGB(255,255,0));

                        // Graph titles & scales
                        sprintf(szBuffer,"Velocity / r (Vmax: %.2e)",vmax);
                        PaintText(hDC,SIDEX,405,szBuffer,RGB(0,255,0));

                        sprintf(szBuffer,"Density / r (Dmax: %.2e)",dmax);
                        PaintText(hDC,SIDEX,565,szBuffer,RGB(0,255,0));

                        lpDDSBack->ReleaseDC(hDC);
                }
        }

        //
        // Flip surface onto screen
        //
        while(1)
        {
                HRESULT ddrval;
                ddrval = lpDDSPrimary->Flip(NULL,0);
                if (ddrval == DD_OK)
                        break;
                if (ddrval == DDERR_SURFACELOST)
                {
                        ddrval = lpDDSPrimary->Restore();
                        if (ddrval != DD_OK)
                                break;
                }
                if (ddrval != DDERR_WASSTILLDRAWING)
                        break;
        }
}

//
// Write data to files
//
if (OPINTDATA)
        if ((int)(t*D) % (int)OPINTDATA == 0)
                WriteDataFile();

if (OPINTBIN)
        if ((int)(t*D) % (int)OPINTBIN == 0)
        {
                sprintf(szBuffer,"\nCONFIGURATION at Dt,%e\n",(double)D*t);
                _write(hGraphFile,szBuffer,lstrlen(szBuffer));

                //
                // Write density graph
                //
                sprintf(szBuffer,"\nMASS DENSITY DISTRIBUTION\nr,density\n");
                _write(hGraphFile,szBuffer,lstrlen(szBuffer));
                for (is=1; is<200; is++)
                {
                        sprintf(szBuffer,"%.2e,%.2e\n",
                                (double)S*is/400.0,dbins[is]*400.0/(is*d*S));
                        _write(hGraphFile,szBuffer,lstrlen(szBuffer));
                }

                //
                // Write stars data
                //
                sprintf(szBuffer,"\nSTARS\ntype,mass,r,V,Ek,rx,ry\n");
                _write(hGraphFile,szBuffer,lstrlen(szBuffer));

                for (is=0; is<1000; is++)
                {
                        sprintf(szBuffer,"%d,%.2e,%.4e,%.4e,%.4e,%.2e,%.2e\n",
                                stars[is].c,stars[is].m,
                                radius(stars[is].r.x-CM,stars[is].r.y-CM),
                                radius(stars[is].V.x,stars[is].V.y),
                                CalcEnergy(is),
                                stars[is].r.x-CM,stars[is].r.y-CM);
                        _write(hGraphFile,szBuffer,lstrlen(szBuffer));
                }

                //
                // Write field at mesh points
                //
                sprintf(szBuffer,"\nMESH FIELD\n");
                _write(hGraphFile,szBuffer,lstrlen(szBuffer));
                int ix,iy;
```

```
for (ix=0; ix<C; ix++)
{
        for (iy=0; iy<C; iy++)
        {
                double F = sqrt((mesh[ix][iy].g.x*mesh[ix][iy].g.x)+
                        (mesh[ix][iy].g.y*mesh[ix][iy].g.y));
                sprintf(szBuffer,"%.3e,",F);
                _write(hGraphFile,szBuffer,lstrlen(szBuffer));
        }
        sprintf(szBuffer,"\n");
        _write(hGraphFile,szBuffer,lstrlen(szBuffer));
}

//
// Write mass in mesh cells
//
sprintf(szBuffer,"\nMESH MASS\n");
_write(hGraphFile,szBuffer,lstrlen(szBuffer));
for (ix=0; ix<C; ix++)
{
        for (iy=0; iy<C; iy++)
        {
                sprintf(szBuffer,"%.2e,",mesh[ix][iy].m);
                _write(hGraphFile,szBuffer,lstrlen(szBuffer));
        }
        sprintf(szBuffer,"\n");
        _write(hGraphFile,szBuffer,lstrlen(szBuffer));
}

sprintf(szBuffer,"END\n");
_write(hGraphFile,szBuffer,lstrlen(szBuffer));

//
// Write to bitmap
//
#define BMPDIM 256
RGBTRIPLE bits[BMPDIM][BMPDIM];
BITMAPINFOHEADER bmih;
BITMAPFILEHEADER bmfh;

bmfh.bfType = 0x4D42;
bmfh.bfReserved1=0;
bmfh.bfReserved2=0;
bmfh.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
bmfh.bfSize = bmfh.bfOffBits + (3*BMPDIM*BMPDIM);

bmih.biSize = sizeof(BITMAPINFOHEADER);
bmih.biWidth = BMPDIM;
bmih.biHeight = BMPDIM;
bmih.biPlanes = 1;
bmih.biBitCount = 24;
bmih.biCompression = BI_RGB;
bmih.biSizeImage = 0;
bmih.biXPelsPerMeter = 300;
bmih.biYPelsPerMeter = 300;
bmih.biClrUsed =0;
bmih.biClrImportant =0;

for (ix=0; ix<BMPDIM; ix++)
{
        for (iy=0; iy<BMPDIM; iy++)
        {
                bits[ix][iy].rgbtBlue =255;
                bits[ix][iy].rgbtGreen =255;
                bits[ix][iy].rgbtRed =255;
        }
}

for (is=N; is>=0; is--)
{
        int y = stars[is].r.x*BMPDIM/S;
        int x = BMPDIM-(stars[is].r.y*BMPDIM/S);
        if ((x<BMPDIM && y<BMPDIM) && (x>=0 && y>=0))
        {
                if (is == 2)
                {
                        int ii;
                        for (ii=-5; ii<=5; ii++)
                        {
                                bits[x+ii][y].rgbtBlue = 255;
                                bits[x+ii][y].rgbtGreen = 0;
                                bits[x+ii][y].rgbtRed = 0;
                        }
                        for (ii=-5; ii<=5; ii++)
                        {
                                bits[x][y+ii].rgbtBlue = 255;
                                bits[x][y+ii].rgbtGreen = 0;
```

```
                                                bits[x][y+ii].rgbtRed = 0;
                                        }
                                }
                                else
                                {
                                        int cnew = bits[x][y].rgbtBlue;
                                        cnew -= 5;
                                        if (cnew < 0) cnew=0;
                                        bits[x][y].rgbtBlue += cnew;
                                        bits[x][y].rgbtGreen += cnew;
                                        bits[x][y].rgbtRed += cnew;
                                }
                        }
                }


                        int             hBMPFile;
                        char    szFileName[MAX_PATH];
                        sprintf(szFileName,"%s_%u.bmp",szDataFile,(int)(t*D/1e6));
                        hBMPFile= _open(szFileName,_O_RDWR | _O_BINARY | _O_CREAT |
                                _O_TRUNC,_S_IREAD | _S_IWRITE);
                        _write(hBMPFile,&bmfh,sizeof(BITMAPFILEHEADER));
                        _write(hBMPFile,&bmih,sizeof(BITMAPINFOHEADER));
                        _write(hBMPFile,&bits,3*BMPDIM*BMPDIM);
                        _close(hBMPFile);

                }
}


//
// FUNCTION: WriteDataFile
//  Writes entry in temporal data file
//
void WriteDataFile(void)
{
     int is,ix,iy;

     //
     // Write file header if t=0
     //
     if (t==0)
     {
             sprintf(szBuffer,"GALAXY CONFIGURATION DATA FILE\nProduced by Galaxy
                     Simulation\nWritten by Andrew Wedgbury\naw@wedger.demon.co.uk\n\n");
             _write(hDataFile,szBuffer,lstrlen(szBuffer));

             sprintf(szBuffer,"N,%d\nD,%e\nR0,%e\nRc,%e\nMGX,%e\nM0,
                     %e\nMh,%e\nVDISP,%e\nC,%d\nS,%e\nd,%e\n",
                     (int)N,(double)D,(double)R0,(double)Rc,(double)MGX,
                     (double)M0,(double)Mh,(double)VDISP,(int)C,(double)S,(double)d);
             _write(hDataFile,szBuffer,lstrlen(szBuffer));
                             sprintf(szBuffer,"Dt,t,Hits,Mesh,Ek,Ep,Etot,R50,R90,R100\n",
                             (int)N,(double)D,(double)R0,(double)Rc,(double)MGX,
                             (double)M0,(double)Mh,(double)VDISP,(int)C,(double)S,(double)d);
             _write(hDataFile,szBuffer,lstrlen(szBuffer));
     }

     //
     // Get 50% 90% and 100% mass radii
     //
     double Utot=0.0,U=0.0,R50,R90,R100;
     for (is=1; is<200; is++)
             Utot += dbins[is]*400.0/(is*d*S);
     for (is=1; is<200; is++)
     {
             U += dbins[is]*400.0/(is*d*S);
             if (U > Utot/2.0)
             {
                     R50 = (double)S*is/400.0;
                     break;
             }
     }
     for (; is<200; is++)
     {
             U += dbins[is]*400.0/(is*d*S);
             if (U > 1.0-(Utot/9.0))
             {
                     R90 = (double)S*is/400.0;
                     break;
             }
     }
     for (; is<200; is++)
     {
             U += dbins[is]*400.0/(is*d*S);
             if (U >= Utot)
```

```
                {
                        R100 = (double)S*is/400.0;
                        break;
                }
        }

        //
        // Write row for this t
        //
        sprintf(szBuffer,"%.2e,%d,%d,%d,%.6e,%.6e,%e,%.6e,%.6e,%.6e\n",
                (double)D*t,(int)t,(int)nEdgeCollisions,
                (int)(100.0*nCellsUsed/((C-1)*(C-1))),Ek,Ep,Ep+Ek,R50,R90,R100);
        _write(hDataFile,szBuffer,lstrlen(szBuffer));
}


//
// FUNCTION: CalcEnergy
//  Calculates kinetic energy of star
//
inline double CalcEnergy(int is)
{
        return 0.5 * stars[is].m * ((stars[is].V.x*stars[is].V.x) + (stars[is].V.y*stars[is].V.y));
}


//
// FUNCTION: InitRCache
//  Initializes Rx and Ry arrays by precalculating values
//
void InitRCache(void)
{
        int ix,iy;
        RxCache[0][0]= 1;
        RyCache[0][0]= 1;

        for (ix=0; ix<C; ix++)
        {
                for (iy=0; iy<C; iy++)
                {
                        if ((ix+iy)!=0)
                        {
                                double x = d*ix;
                                double y = d*iy;
                                double r = radius(x,y);

                                RxCache[ix][iy]= G*x / (r*r*r);
                                RyCache[ix][iy]= G*y / (r*r*r);
                        }
                }
        }

}


//
// FUNCTION: LookupRx
//  Looks up cached Rx value in array
//
inline double LookupRx(int ix, int iy)
{
        if ((ix>=0) && (iy>=0))
                return RxCache[ix][iy];

        if ((ix<=0) && (iy>=0))
                return -RxCache[-ix][iy];

        if ((ix>=0) && (iy<=0))
                return RxCache[ix][-iy];

        if ((ix<=0) && (iy<=0))
                return -RxCache[-ix][-iy];

        OutputDebugString("Rx lookup failed\n");
        return 1.0;
}


//
// FUNCTION: LookupRy
//  Looks up cached Ry value in array
//
inline double LookupRy(int ix, int iy)
{
        if ((ix>=0) && (iy>=0))
                return RyCache[ix][iy];
```

```
	if ((ix<=0) && (iy>=0))
		return RyCache[-ix][iy];

	if ((ix>=0) && (iy<=0))
		return -RyCache[ix][-iy];

	if ((ix<=0) && (iy<=0))
		return -RyCache[-ix][-iy];

	OutputDebugString("Ry lookup failed\n");
	return 1.0;
}


//
// WNDPROC: Main Window Procedure
//
LRESULT CALLBACK DispWndProc(HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam)
{
	switch(message)
	{
		case WM_CREATE:
			break;

		case WM_PAINT:
		case WM_ERASEBKGND:
		case WM_NCPAINT:
			return 0;

		case WM_CHAR:                                          // PROCESS KEYBOARD INPUT
			switch(wParam)
			{
				case 'f':
					bDrawField = !bDrawField;
					OutputData();
					break;

				case 's':
					bDrawStars = !bDrawStars;
					OutputData();
					break;

				case 't':
					bDrawText = !bDrawText;
					OutputData();
					break;

				case 'i':
					bDrawIndic = !bDrawIndic;
					OutputData();
					break;

				case 'a':
					bDrawAxes = !bDrawAxes;
					OutputData();
					break;

				case 'h':
					bDrawHalo = !bDrawHalo;
					OutputData();
					break;

				case 'd':
					bDraw3D = !bDraw3D;
					OutputData();
					break;

				case ' ':
					bPaused = !bPaused;
					if (bPaused)
					{
						Sleep(0);
						OutputData();
					}
					break;

				case '+':
					xs *= 2;
					ys *= 2;
					xt = -xs*(double)(CM);
					yt = -ys*(double)(CM);
					OutputData();
					break;

				case '-':
					xs /= 2;
					ys /= 2;
```

```
                                        xt = -xs*(double)(CM);
                                        yt = -ys*(double)(CM);
                                        OutputData();
                                        break;

                                case ']':
                                        xt -= xs;
                                        OutputData();
                                        break;
                                case '[':
                                        xt += xs;
                                        OutputData();
                                        break;
                                case ';':
                                        yt += xs;
                                        OutputData();
                                        break;
                                case '.':
                                        yt -= xs;
                                        OutputData();
                                        break;
                                case '/':
                                        xs = WINDOWX/S;
                                        ys = WINDOWY/S;
                                        xt = 0; yt = 0;
                                        break;

                                case 'r':
                                        t = 0;
                                        nRevs[0]=0;
                                        nRevs[1]=0;
                                        vmax=dmax=fmax=0;
                                case 'c':
                                case 'q':
                                        vmax=dmax=fmax=0;
                                        DestroyWindow(hDispWin);
                                        break;

                                case 'w':
                                        if (!bPaused)
                                        {
                                                bPaused = 1;
                                                Sleep(0);
                                                WriteDataFile();
                                                bPaused = 0;
                                        }
                                        else
                                                WriteDataFile();
                                        break;
                        }
                        break;

                case WM_CLOSE:
                        DestroyWindow(hDispWin);
                        break;

                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
        }
        return(DefWindowProc(hWnd,message,wParam,lParam));
}



BOOL FAR PASCAL SimDlgProc(HWND hDlg,UINT message,WPARAM wParam,LPARAM lParam)
{
        switch(message)
        {
                case WM_INITDIALOG:
                        sprintf(szBuffer,"%.3e",D);
                        SetDlgItemText(hDlg,IDCE_D,szBuffer);
                        sprintf(szBuffer,"%.3e",R0);
                        SetDlgItemText(hDlg,IDCE_R0,szBuffer);
                        sprintf(szBuffer,"%.3e",MGX);
                        SetDlgItemText(hDlg,IDCE_MGX,szBuffer);
                        sprintf(szBuffer,"%.3e",Rc);
                        SetDlgItemText(hDlg,IDCE_Rc,szBuffer);
                        sprintf(szBuffer,"%.3e",Mh);
                        SetDlgItemText(hDlg,IDCE_Mh,szBuffer);
                        sprintf(szBuffer,"%.3e",VDISP);
                        SetDlgItemText(hDlg,IDCE_VDISP,szBuffer);
                        sprintf(szBuffer,"%d",C);
                        SetDlgItemText(hDlg,IDCE_C,szBuffer);
                        sprintf(szBuffer,"%.3e",S);
                        SetDlgItemText(hDlg,IDCE_S,szBuffer);
```

```
                        sprintf(szBuffer,"%.3e",OPINTDATA);
                        SetDlgItemText(hDlg,IDCE_OPINTDATA,szBuffer);
                        sprintf(szBuffer,"%.3e",OPINTBIN);
                        SetDlgItemText(hDlg,IDCE_OPINTBIN,szBuffer);
                        sprintf(szBuffer,"%d",N);
                        SetDlgItemText(hDlg,IDCE_N,szBuffer);
                        SetDlgItemText(hDlg,IDCE_DATAFILE,szDataFile);
                        return TRUE;

                case WM_COMMAND:
                        switch (LOWORD(wParam))
                        {
                                case IDCANCEL:
                                        EndDialog(hDlg,0);
                                        break;

                                case IDOK:
                                        GetDlgItemText(hDlg,IDCE_D,szBuffer,sizeof(szBuffer));
                                        D = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_R0,szBuffer,sizeof(szBuffer));
                                        R0 = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_MGX,szBuffer,sizeof(szBuffer));
                                        MGX = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_Rc,szBuffer,sizeof(szBuffer));
                                        Rc = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_Mh,szBuffer,sizeof(szBuffer));
                                        Mh = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_VDISP,szBuffer,sizeof(szBuffer));
                                        VDISP = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_C,szBuffer,sizeof(szBuffer));
                                        C = atoi(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_S,szBuffer,sizeof(szBuffer));
                                        S = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_OPINTDATA,
                                                szBuffer,sizeof(szBuffer));
                                        OPINTDATA = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_OPINTBIN,
                                                szBuffer,sizeof(szBuffer));
                                        OPINTBIN = atof(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_N,szBuffer,sizeof(szBuffer));
                                        N = atoi(szBuffer);
                                        GetDlgItemText(hDlg,IDCE_DATAFILE,
                                                szDataFile,sizeof(szDataFile));
                                        EndDialog(hDlg,1);
                                        break;

                                case IDCB_PLAYBACK:
                                        break;

                                case IDCB_ABOUT:
                                        break;
                        }
                        return TRUE;
        }
        return FALSE;
}


//
//  PROGRAM ENTRY POINT
//
int PASCAL WinMain(HINSTANCE hInstance,HINSTANCE,LPSTR,int)
{
        OutputDebugString("WinMain: Program started\n");

        const char      szControlClassName[] = "WGSim";
        const char      szDispClassName[] = "WGSimOutput";
        hInst = hInstance;

        //
        // Register main window class
        //
        WNDCLASS wc;
        wc.style = 0;
        wc.lpfnWndProc = DispWndProc;
        wc.cbClsExtra = 0;
        wc.cbWndExtra = 0;
        wc.hInstance = hInst;
        wc.hIcon = LoadIcon(hInst,(LPSTR)IDI_APP);
        wc.hCursor = LoadCursor(NULL,IDC_ARROW);
        wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
        wc.lpszMenuName = (LPSTR)IDM_APP;
        wc.lpszClassName = szDispClassName;
        if (!RegisterClass(&wc)) return -1;

        hBigFont = CreateFont((int)(20.0*WINDOWX/640.0),0,0,0,0,0,0,0,0,0,0,0,0,"Arial");
        hSmallFont = CreateFont((int)(15.0*WINDOWX/640.0),0,0,0,0,0,0,0,0,0,0,0,0,"Arial");
```

```
	//
	// Init variables to default
	//
	lstrcpy(szDataFile,"galaxy");

	//
	// Display simulation config dialog box
	//
	while (DialogBox(hInst,MAKEINTRESOURCE(IDD_SIM),NULL,(DLGPROC)SimDlgProc))
	{
		//
		// Init precalculated values
		//
		d = (double)S/C;
		CM = (double)(d*C/2.0);
		InitRCache();
		xs = WINDOWX/S;
		ys = WINDOWY/S;

		//
		// Create main window
		//
		hDispWin = CreateWindow(szDispClassName,"Galaxy Simulation output",
			WS_OVERLAPPEDWINDOW,
			CW_USEDEFAULT,CW_USEDEFAULT,WINDOWX+10,WINDOWY+34,
			NULL,NULL,hInst,NULL);

		if (!IsWindow(hDispWin))
			return -1;

		//
		// Create DirectDraw Objects
		//
		DDSURFACEDESC	ddsd;
		DDSCAPS		ddscaps;

		if (FAILED( DirectDrawCreate(NULL,&lpDD,NULL) ))
			return -1;

		// Get exclusive mode
		if (FAILED(lpDD->SetCooperativeLevel(hDispWin,DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN)))
			return -1;

		//
		// Create the primary surface with 1 back buffer
		//
		ddsd.dwSize = sizeof(ddsd);
		ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
		ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP | DDSCAPS_COMPLEX;
		ddsd.dwBackBufferCount = 1;
		if (FAILED( lpDD->CreateSurface(&ddsd,&lpDDSPrimary,NULL) ))
			return -1;

		//
		// Get a pointer to the back buffer
		//
		ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
		if (FAILED( lpDDSPrimary->GetAttachedSurface(&ddscaps,&lpDDSBack) ))
			return -1;

		//
		// Setup surfaces ready for drawing
		//
		HDC hDC;
		RECT rWin;
		lpDDSBack->GetDC(&hDC);
		SetBkColor(hDC,RGB(0,0,0));
		SetBkMode(hDC,OPAQUE);
		GetWindowRect(hDispWin,&rWin);
		FillRect(hDC,&rWin,(HBRUSH)GetStockObject(BLACK_BRUSH));
		lpDDSBack->ReleaseDC(hDC);

		//
		// Show & update windows
		//
		ShowWindow(hDispWin,SW_SHOWNORMAL);
		OutputData();

		//
		// Create background calculation thread
		//
		DWORD dwThreadId,dwThrdParam=1;
		hCalcThread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)CalcThread,
			&dwThrdParam,0,&dwThreadId);
		SetThreadPriority(hCalcThread,THREAD_PRIORITY_NORMAL);
```

```
		//
		// Start Windows message server and wait for simulation to end
		//
		OutputDebugString("WinMain: Starting message server\n");
		MSG msg;
		while (GetMessage(&msg,NULL,0,0))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}

		//
		// Shutdown simulation
		//
		OutputDebugString("WinMain: Shutting down\n");

		TerminateThread(hCalcThread,2);
		CloseHandle(hCalcThread);

		_close(hDataFile);
		_close(hBinFile);

		if( lpDD != NULL )
		{
			if( lpDDSPrimary != NULL )
			{
				lpDDSPrimary->Release();
				lpDDSPrimary = NULL;
			}
			lpDD->Release();
			lpDD = NULL;
		}
	}
	DeleteObject(hBigFont);
	DeleteObject(hSmallFont);

	return 0;

}
```

resource.h

```
#define IDI_APP                         101
#define IDM_APP                         102
#define IDD_SIM                         103
#define IDCE_N                          1000
#define IDCE_DATAFILE                   1002
#define IDCB_PLAYBACK                   1003
#define IDCE_D                          1004
#define IDCE_R0                         1005
#define IDCE_MGX                        1006
#define IDCE_Rc                         1007
#define IDCE_Mh                         1008
#define IDCE_VDISP                      1009
#define IDCE_C                          1010
#define IDCE_S                          1011
#define IDCE_OPINTDATA                  1012
#define IDCE_OPINTBIN                   1013
#define IDCB_ABOUT                      1018
#define ID_FILE_PLAYBACKSIM             40001
#define ID_FILE_RECORDSIM               40002
#define ID_FILE_EXIT                    40003
#define ID_SIMULATION_NEW               40004
#define ID_SIMULATION_STOP              40005
#define ID_SIMULATION_CONTINUE          40006
#define ID_VIEW_FULLSCREEN              40007
```

resource.rc

```
#include "resource.h"

IDI_APP                 ICON    DISCARDABLE     "icon1.ico"

IDD_SIM DIALOG DISCARDABLE  0, 0, 217, 175
STYLE DS_MODALFRAME | DS_CENTER | WS_CAPTION | WS_SYSMENU
CAPTION "Galaxy Simulation"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDCE_D,60,20,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDCE_R0,60,35,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDCE_MGX,60,50,45,12,ES_AUTOHSCROLL
    EDITTEXT        IDCE_Rc,60,65,45,12,ES_AUTOHSCROLL
```

```
      EDITTEXT        IDCE_Mh,60,80,45,12,ES_AUTOHSCROLL
      EDITTEXT        IDCE_VDISP,60,95,45,12,ES_AUTOHSCROLL
      EDITTEXT        IDCE_C,60,110,45,12,ES_AUTOHSCROLL
      EDITTEXT        IDCE_S,60,125,45,12,ES_AUTOHSCROLL
      EDITTEXT        IDCE_OPINTDATA,60,140,45,12,ES_AUTOHSCROLL
      EDITTEXT        IDCE_OPINTBIN,60,155,45,12,ES_AUTOHSCROLL
      EDITTEXT        IDCE_N,140,20,45,12,ES_AUTOHSCROLL
      EDITTEXT        IDCE_DATAFILE,140,50,70,12,ES_AUTOHSCROLL
      PUSHBUTTON      "ABOUT",IDCB_ABOUT,140,115,70,15
      PUSHBUTTON      "EXIT",IDCANCEL,140,135,70,15
      DEFPUSHBUTTON   "RUN SIMULATION",IDOK,140,154,70,14
      LTEXT           "Stars",IDC_STATIC,190,22,17,8
      LTEXT           "Timestep",IDC_STATIC,5,23,30,8
      LTEXT           "Initial disk radius",IDC_STATIC,5,38,52,8
      LTEXT           "Disk mass",IDC_STATIC,5,52,33,8
      LTEXT           "Halo radius",IDC_STATIC,5,68,36,8
      LTEXT           "Halo mass",IDC_STATIC,5,82,34,8
      LTEXT           "Vel. dispersion",IDC_STATIC,5,97,46,8
      LTEXT           "Mesh edge dim",IDC_STATIC,5,113,49,8
      LTEXT           "Mesh size",IDC_STATIC,5,127,32,8
      LTEXT           "Output every",IDC_STATIC,5,143,42,8
      LTEXT           "Record every",IDC_STATIC,5,158,44,8
      LTEXT           "Years",IDC_STATIC,110,23,19,8
      LTEXT           "Kpc",IDC_STATIC,110,38,14,8
      LTEXT           "Msun",IDC_STATIC,110,52,18,8
      LTEXT           "Kpc",IDC_STATIC,110,68,14,8
      LTEXT           "Msun",IDC_STATIC,110,82,18,8
      LTEXT           "%",IDC_STATIC,110,97,8,8
      LTEXT           "Cells",IDC_STATIC,110,113,16,8
      LTEXT           "Kpc",IDC_STATIC,110,127,14,8
      LTEXT           "Years",IDC_STATIC,110,143,19,8
      LTEXT           "Years",IDC_STATIC,110,158,19,8
      LTEXT           "Output file name",IDC_STATIC,140,40,52,8
      CTEXT           "Galaxy Simulation Program by Andrew Wedgbury",
                      IDC_STATIC,5,5,205,10
END
```

Icon file: "icon1.ico"
Link with: kernel32.lib, user32.lib, gdi32.lib, ddraw.lib

References

[1] **Baugh C, Frenk C – *How are galaxies made?***
1999, Physics World vol 12, issue 5

[2] **Combes F, et al. - *Galaxies and Cosmology***
1995, Springer

[3] **Hockney R W, Eastwood J W – *Computer Simulation using Particles***
1988, Adam Hilger

[4] **Hohl F – *N-Body Simulations of Disks***
1975, Dynamics of Stellar Systems 349-366

[5] **Hohl F – *Numerical Experiments with a Disk of Stars***
1971, The Astrophysical Journal 168:343-359

[6] **Hohl F, Hockney R W – *A Computer Model of Disks of Stars***
1968, Journal of Computational Physics 4, 306-324

[7] **Hoyle F – *Astronomy and Cosmology: A modern course***
1975, W H Freeman and company

[8] **International Astronomical Union – *The Spiral Structure of our Galaxy***
1970, D Reidel Publishing co.

[9] **Sandage A – *The Hubble Atlas of Galaxies***
1961, Carnegie Institution of Washington

[10] **Tayler R J - *Galaxies: Structure and evolution***
1978, CUP

[11] **Zhang X – *Secular Evolution of Spiral Galaxies***
1998, The American Astrophysical Journal, 449:93-111