3D Hydrodynamic Simulation of Classical Nova Explosions

New Mexico
Supercomputing Challenge
Final Report
April 2, 2014

Team 66
Los Alamos High School

Team Members
Coleman Kendrick

Teacher
Brian Kendrick

Project Mentor
Brian Kendrick

Contents

1 Summary **3**

2 Introduction **4**
2.1 Technical Challenges 6

3 Simulation Model **7**

4 Particle Recycling and Blocking Methods **15**

5 Code Development History **19**

6 OpenCL and MPI **21**

7 Moving Least Squares **26**

8 U Scorpii **27**

9 Results **28**

10 Conclusions **35**

11 Future Work **36**

A Other Parameter Studies **37**

B Summary of Uploaded Code **44**

References **46**

1 Summary

The purpose of this project is to develop a computer model to investigate the formation and life cycle of classical novae. A nova is an orbiting system consisting of a white dwarf and star. Over time, the white dwarf pulls hydrogen gas from the star which gathers onto the surface of the white dwarf (the accretion phase). Once the critical conditions are reached for thermonuclear runaway (TNR) the hydrogen gas on the surface of the white dwarf will ignite and explode as a rapidly expanding gas shell (the nova phase). Each particle in my computer simulation represents a volume of hydrogen gas and is initialized randomly in the outer shell of the companion star. The core of the star is modeled by a repulsive wall using a Wendland weight function. The motion of each particle is computed by solving Newton's equations of motion. The forces on each particle include: gravity, centrifugal, Coriolis, friction, and Langevin. The friction and Langevin thermostat forces are used to model the viscosity and internal pressure of the gas. A rotating frame is used and the centrifugal and Coriolis forces are included to model the rotation. A velocity Varlet method with a one second time step is used to compute velocities and positions of the particles. A new particle recycling method was developed which was critical for computing an accurate and stable accretion rate (the rate at which particles stick to the white dwarf) and keeping the particle count reasonable. TNR occurs with only around 500 particles with the recycling method, compared to 500,000 without particle recycling. I used C++ with OpenCL and MPI to parallelize my simulations which were run on two Nvidia GTX 580s. The simulations used between 32,768 and 1 million particles and each run required between 1 and 10 hours to complete. I used my model to investigate how parameters such as the white dwarf mass, star mass and separation will affect the evolution of the nova. These parameters affect the accretion rate, the frequency of the nova explosions and light curves (the Nova's visual magnitude versus time). My computed light curve results were fit to the observed light curve from the 2010 U Scorpii nova outburst. The program parameters were then varied to determine the effects they had on the light curve and frequency of nova explosions. When a small white dwarf mass was used, the light curve peaked higher and cooled off slower. The opposite was observed for a heavier white dwarf mass; the light curve peaked lower and cooled off faster. As the white dwarf mass is increased from 0.4 to 1.4 solar masses, the time period between explosions decreases dramatically from 100,000 to 1 yr. My simulations also show that the companion star blocks the expanding gas shell leading to an asymmetrical expanding shell. The results of my simulations compared well with the professional 1D and 3D hydrodynamic simulations and observed experimental data. I also investigated the effects of other parameters on my simulation results, such as: the time step, number of particles, Langevin K_{eff}, friction coefficient, and particle mass.

2 Introduction

What is a Nova?

A nova (not to be confused with a supernova, see below) consists of a rotating binary system, usually consisting of a main sequence star (although sometimes a red giant) and a white dwarf. The white dwarf accretes hydrogen gas from its companion star which builds up on the white dwarf's surface. Once the gas reaches a critical density and temperature, it undergoes a thermonuclear runaway (TNR) explosion and leaves the white dwarf's surface in a rapidly expanding shell of burning hydrogen gas.[1, 2] The accretion and nova outburst phases typically repeat over time with explosions occurring as often as 1 year to over 100,000 years (or more) depending on the system. The white dwarf itself may also eventually explode as a type 1A supernova if it gains enough mass over time and passes the Chandrasekhar limit (about 1.44 solar masses). However, whether the white dwarf explodes as a supernova also depends on the composition of white dwarf. If it is a CO (carbon-oxygen) white dwarf, it can collapse and explode as a type 1A supernova, but if it is a ONe (oxygen-neon) white dwarf it will collapse into a very dense neutron star and not explode.[3]

Why are Novae Important?

Understanding the life cycle of a nova is important since over time the white dwarf can gain mass and eventually explode as a type 1A supernova. Type 1A supernova are used as "standard candles" for measuring cosmic distances and lead to the 1998 discovery of the accelerating expansion of the universe. Studying novae is also important because they can be used to measure distances as well which allows us to get accurate distance information on nearby galaxies. Nova are also important to understand for stellar evolution, since the gas that gets pulled off of the star causes the star to look a different age.

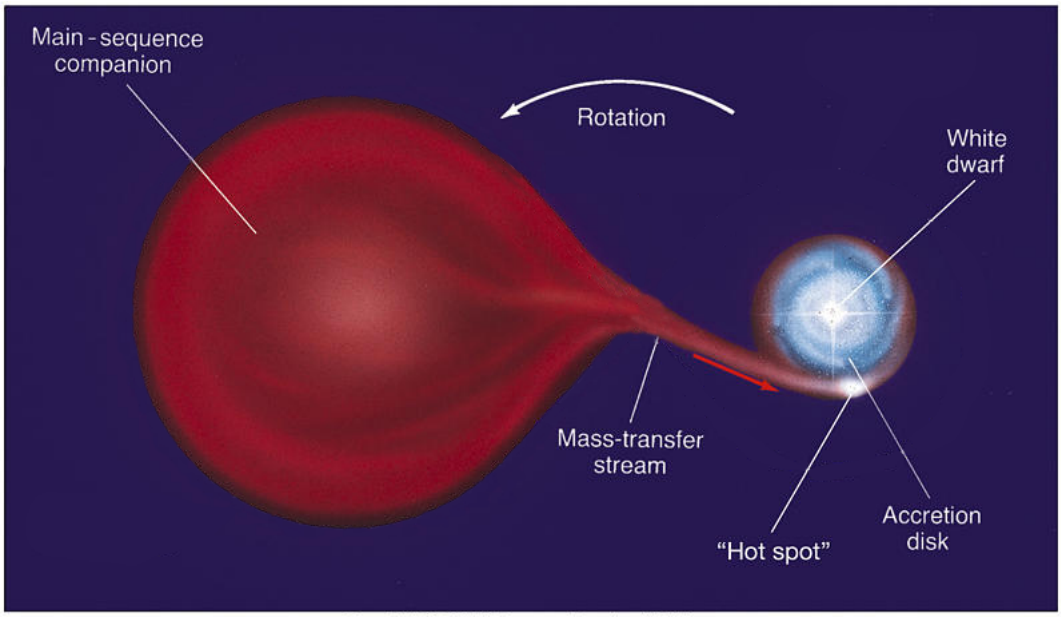


Figure 1: Shows the accretion of gas from the companion star to the white dwarf, forming the accretion disk.

2.1 Technical Challenges

There are many technical challenges when modeling a nova system. First off, there are large differences in the timescales between the accretion phase and the explosion phase. Accretion can take years (about 10 for U Scorpii), whereas the explosion only takes about 20 minutes to expand past its companion star. To account for this, different time steps (dt) were used in order to accurately model these different phases. Secondly, simulating accretion takes many particles, and it would take a lot of processing power to do that over the course of multiple years. This is why MPI (Message Passing Interface) and OpenCL (Open Computing Language) are used to significantly decrease simulation time. MPI allows multiple cores of a CPU to work simultaneously, splitting the work among them. Similarly, OpenCL allows the program to use the GPU (Graphical Processing Unit) to increase performance on large problems. OpenCL provides greater performance due to the fact that a GPU can have many more cores (Nvidia GTX580 has 512 Cuda Cores, and the AMD HD7970 has 2048 Stream Processors) compared to the CPU. OpenCL is executed in kernels which holds the majority of the calculation. The speedup of using OpenCL compared to a single core on a processor was about 90-100 times. Studies were done to investigate the differences between OpenCL and MPI in terms of performance and architecture. Because the accretion phase takes many particles and the particle mass is so small (about $1.0e-11$ for U Scorpii), it would take around 500,000 particles just to reach the critical TNR condition of 1 kg/cm^3. To avoid simulating this many particles, and to get a stable recurrent nova system, a particle recycling method was developed in order to decrease the particle count required to reach explosion. This method reuses particles that have been stuck on the surface of the white dwarf, and also those that leave the system. The particle recycling method will be discussed in greater detail later on.

3 Simulation Model

To model the nova system, particles of hydrogen gas are randomly distributed in a shell around the star. Since this simulation is not trying to model star stability or formation, the inner core of the star is not modeled with particles and only the outer layer is being modelled. The outer shell of particles experience Langevin thermostat forces (friction + random kicks) which are used to model the gas viscosity and pressure.[4, 5] It is unrealistic to model individual particles of hydrogen gas, so each particle represents a volume of hydrogen gas. The core of the star is modelled using a single mass point at the center (which is used to compute gravitational forces) and a repulsive wall starting at the surface of the core. The white dwarf is placed a given distance away from the core of the star (6.5 Solar Units for U Scorpii), which is also being modeled as a single mass point with a repulsive wall. If a particle comes near the surface of the white dwarf, it is effectively labelled as being "stuck" to the white dwarf. Since a nova is a rotating binary system, a rotating frame is used. Coriolis and centrifugal forces are included in order to model the rotating frame. Figure 2 shows the various forces acting upon the particles in the simulation, depicted by different colored arrows.

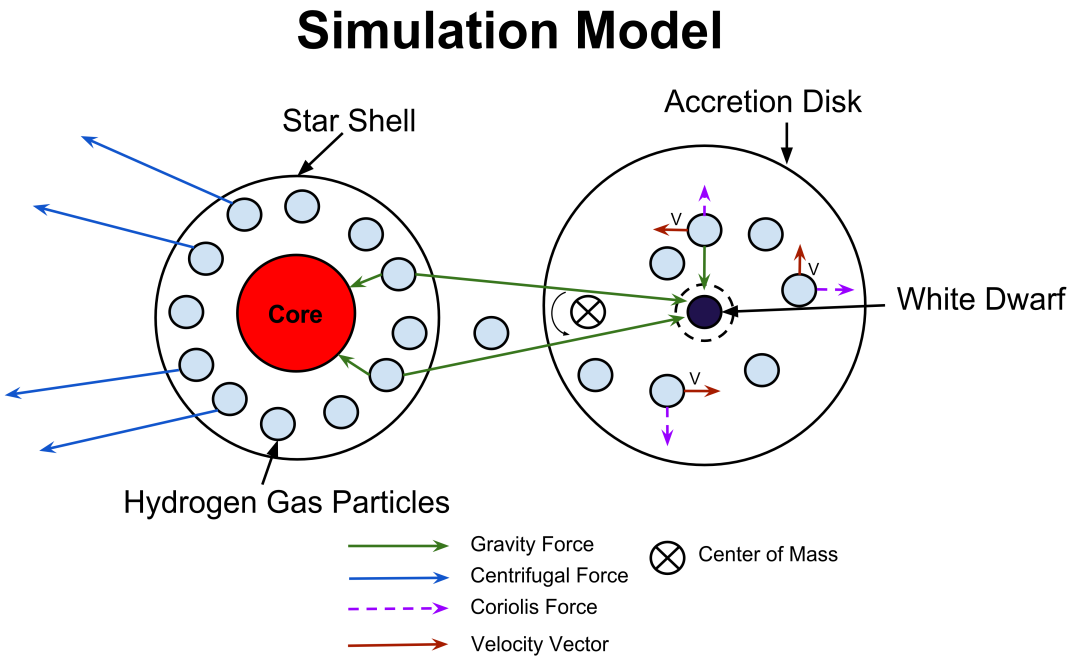


Figure 2: Depicts all forces acting upon the particles. Shows the rotating frame about the center of mass

The centrifugal force has a greater effect the farther the particle is from the center of mass, which pulls particles out from the shell of the star. This is seen in figure 2 by the large blue arrows on the particles on the far left side of the gas shell. The Coriolis force is perpendicular to the velocity of the particle and is depicted by the dashed purple arrows. Particles being pulled towards the white dwarf have larger velocities than the particles on the star gas shell, which enables the Coriolis force to have a much greater effect; causing the particles to bend around the white dwarf forming an accretion disk. All forces being acted upon a particle are shown in equation 1, and each force equation labelled beneath it.

$$F_{\text{total}} = F_{\text{gravity}} + F_{\text{centrifugal}} + F_{\text{coriolis}} + F_{\text{friction}} + F_{\text{Langevin}} \tag{1}$$

$$F_{\text{gravity}} = \frac{Gm_1m_2}{r^2}$$

$$F_{\text{centrifugal}} = mr\Omega^2$$

$$F_{\text{coriolis}} = mv\Omega$$

$$F_{\text{friction}} = -mv\epsilon$$

$$F_{\text{Langevin}} = (2\epsilon K_{eff} T m)^{\frac{1}{2}} N_{rand}$$

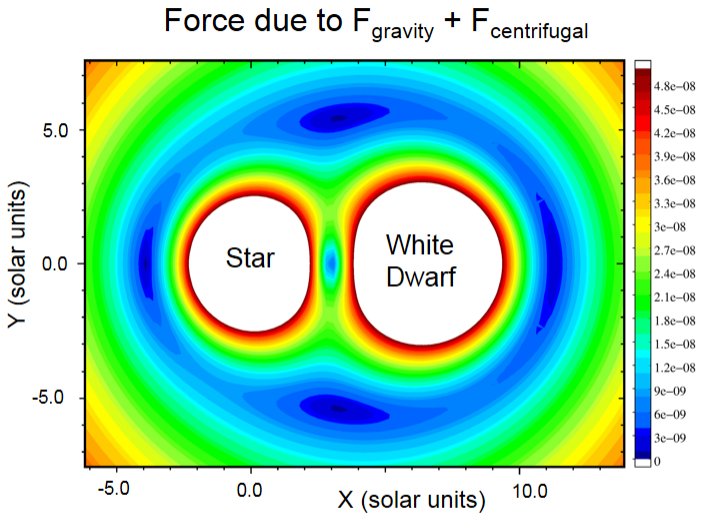


Figure 3: Contour plot depicting gravitational and centrifugal forces of the nova system.

Above in figure 3, high gravitational forces are seen directly around the star and white dwarf. The force was not plotted when it reached higher than 4.8^{-}08 (depicted by the white regions centered on the star and white dwarf). In comparison, the white dwarf has a much stronger gravitational pull than the star, which causes the particles to accrete. The dark blue "trough" around the star and white dwarf corresponds to the region where the centrifugal and gravitational forces exactly cancel (giving zero force). The blue region between the star and white dwarf is where the gravitational force from the star and white dwarf cancel.

A "wall" force is added to represent the dense core of the star to keep shell particles from collapsing into the core due to the large force of gravity the particles feel. This wall is modelled using the Wendland weight function [6] and is then scaled by the gravity force at the core in order to make the repulsion force. The Wendland weight function is shown in equation 2 where r represents radius, and h represents a "width" parameter which adjusts how quickly the function will fall off to zero. The companion white dwarf has a much stronger wall force, as the gravity of the white dwarf is much larger.

$$w = 1 - 10u^2 + 20u^3 - 15u^4 + 4u^5 \tag{2}$$

$$u = r/h$$

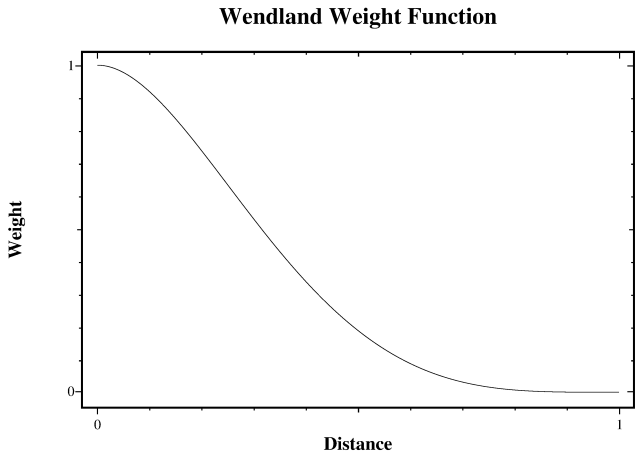


Figure 4: Wendland weight function versus distance $u = r/h$ (for $h = 1$).

The velocity Verlet method is used to solve Newton's laws of motion and gravity, and to update particle positions. A comparison of methods was done, shown in figure 5. Euler and Symplectic Euler methods were not used since they are 1st order, and not as accurate as a higher order method. Between the Velocity Verlet and the Predictor Corrector methods [7], the Predictor Corrector is much slower since it requires two acceleration updates per time step. The Velocity Verlet method was finally chosen since it has a combination of both speed and accuracy.

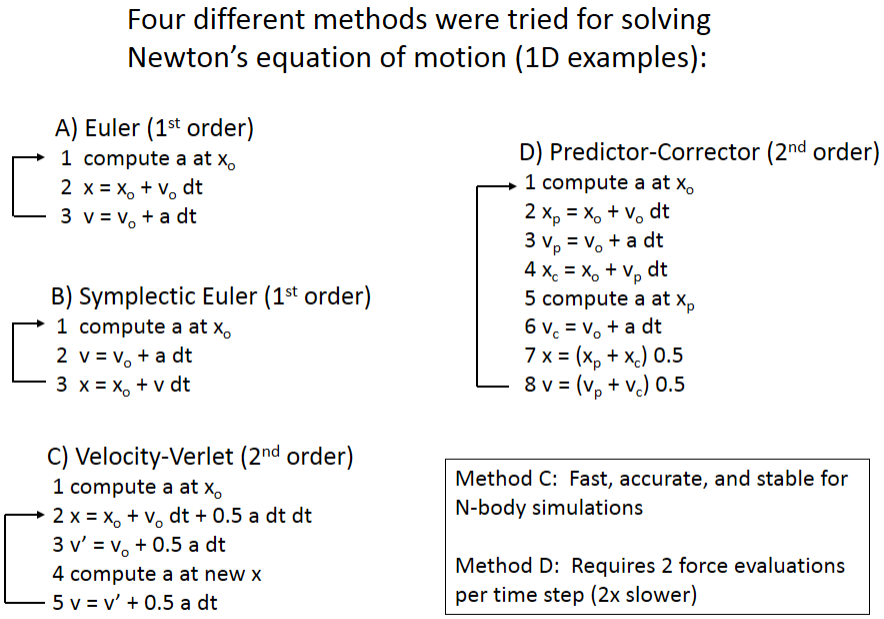


Figure 5: Comparison of four different numerical methods. Velocity Verlet (C) was used for my simulations since it is computationally faster than Predictor Corrector [7] (D).

During the simulation, the gravitational force from the white dwarf is slowly switched on over time. This is achieved by multiplying the gravitational force by a factor ($dwarfm$) that is incremented from 0 initially to 1 when the white dwarf is considered fully active. This allows for the particles in the star shell to equilibrate before feeling a large gravity force acting upon them. Over time, particles will slowly accrete over to the white dwarf forming an accretion disk. As particles come into contact with the white dwarf shell, they will "stick" to the surface of the white dwarf (their velocities and accelerations are set to zero). As simulation time grows, the amount of hydrogen gas stuck to the white dwarf increases until it reaches the critical density of 1 kg/cm^3 and critical temperature of 20 million kelvin. At these critical conditions, the nova will undergo TNR (Thermonuclear Runaway) and explode the shell of gas on its surface.[2, 8]

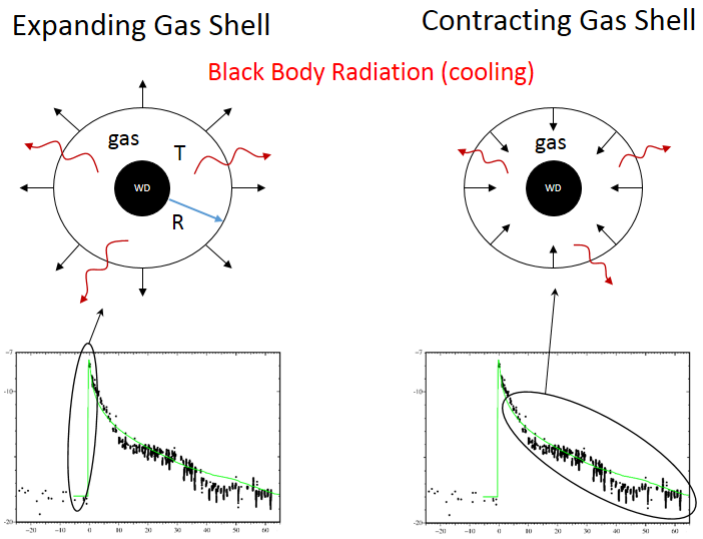


Figure 6: Depicts the expanding gas shell model at TNR.

At explosion, the gas shell expands outward, emitting black body radiation which serves as a cooling factor for the burning hydrogen gas. The expanding shell represents the large rise in the light curve, as circled in black on the left side of figure 6. Over time, the gravity from the white dwarf slows the expanding gas shell, causing the shell to contract back towards the white dwarf. The gas shell continues to radiate and cool during this time. The contraction of the gas shell back towards the white dwarf corresponds to the "fall off" part of the light curve, as circled in black on the right side of figure 4. The temperature of the expanding gas shell is computed from the following equation

$$T = T_o - aL/C_v\Delta t \tag{3}$$

where T_o is the gas temperature (in Kelvin) at the previous time step. The initial value at ignition is adjusted (as a fitting parameter) to match the maximum in the light curve. C_v is the specific heat capacity of the hydrogen gas ($C_v = 3/2nR$). The parameter a (the cooling rate) is adjusted to match the decreasing part of the light curve. The luminosity of the light curve is related to the temperature (T) and radius (R) of the gas shell and is computed using 3 below

$$L = L_{solar}R^2T^4 \tag{4}$$

where L is the luminosity of the gas shell in watts, and L_{solar} is the luminosity of the sun.

The luminosity is used to compute the visual magnitude (light curve) using the equation

$$mag = mag_{solar} - 2.5 \log_{10}(L_{total}/R^2_{nova}) \tag{5}$$

where $mag_{solar} = -26.73$ is our sun's magnitude, R_{nova} is the distance to the nova from earth (8 kpc = 26,000 light years for U Scorpii), and $L_{total} = L_{star} + L_{nova}$ (L_{nova} in solar units is computed from 4 and $L_{star} = 3.45$ is the companion star's luminosity).

Figure 7 shows the computed gas temperature (on the surface of the white dwarf) and the computed gas density (on the surface of the white dwarf) for four different simulations with different white dwarf mass: $M_{wd} = 0.5$ (red), $M_{wd} = 0.75$ (green), $M_{wd} = 1.0$ (blue), and $M_{wd} = 1.3$ (black). The red horizontal and vertical dashed lines indicate the critical gas temperature and density for TNR. The cross hatched region depicts the region where both the critical temperature and density occur for TNR (20 million Kelvin and 1 kg/cm^3).

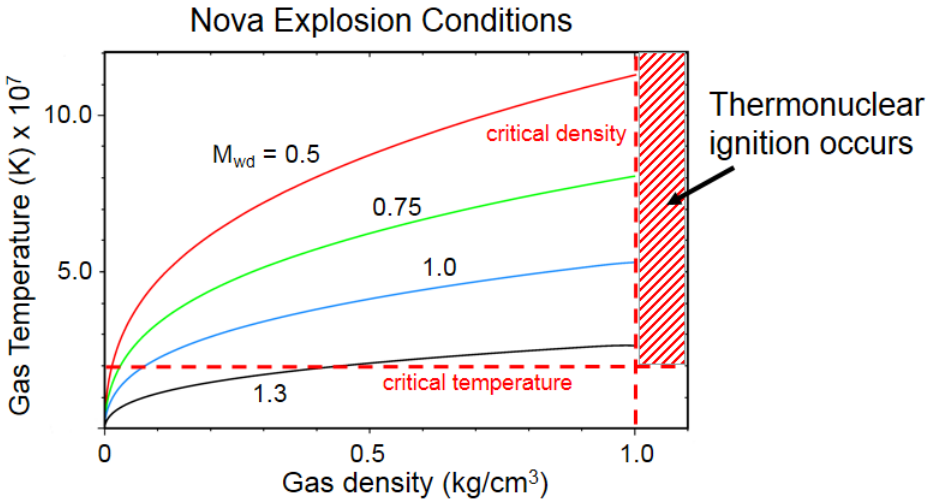


Figure 7: Gas temperature and density (on the white dwarf surface). The red crossed box represents thermonuclear ignition conditions.

The density of the gas on the white dwarf surface is computed below

$$\rho = n_p m_p / V_{shell} \tag{6}$$

where n_p is the number of particles that have been stuck onto the surface of the white dwarf, m_p is the mass of each particle, and V_{shell} is the volume of the gas shell, later depicted in figure 26.

The temperature of the gas on the white dwarf surface is computed below [3]

$$T = 1.4 \times 10^7 K(M'\rho^2/M_{wd})^{\frac{2}{11}} \tag{7}$$

M' is the mass accretion rate, computed in Solar Masses per year, ρ is the gas density computed from equation 6, and M_{wd} is the mass of the white dwarf. At each time step, the program uses these two equations to determine whether or not the nova should explode.

4 Particle Recycling and Blocking Methods

The particle recycling method was developed in order to reduce the amount of particles needed to cause an explosion. At each time step, the program determines if the particle count on the white dwarf has exceeded a value (*wd_recycle* - which was 1000 for most simulations). If the count has exceeded this value, the program will re-initialize all stuck particles (excluding one for scaling and all other particles that have already been scaled) around the star and scale the mass of the remaining particle by the *wd_recycle* parameter. This ensures that the particle mass on the white dwarf is conserved. Stuck particles are colored green in figure 8. Since particles are being moved back to the star, the particle count required to undergo explosion is reduced by a factor of *wd_recycle*, therefore only about 500 particles are actually needed (for a *wd_recycle* = 100) in order to explode the nova at the same critical conditions.

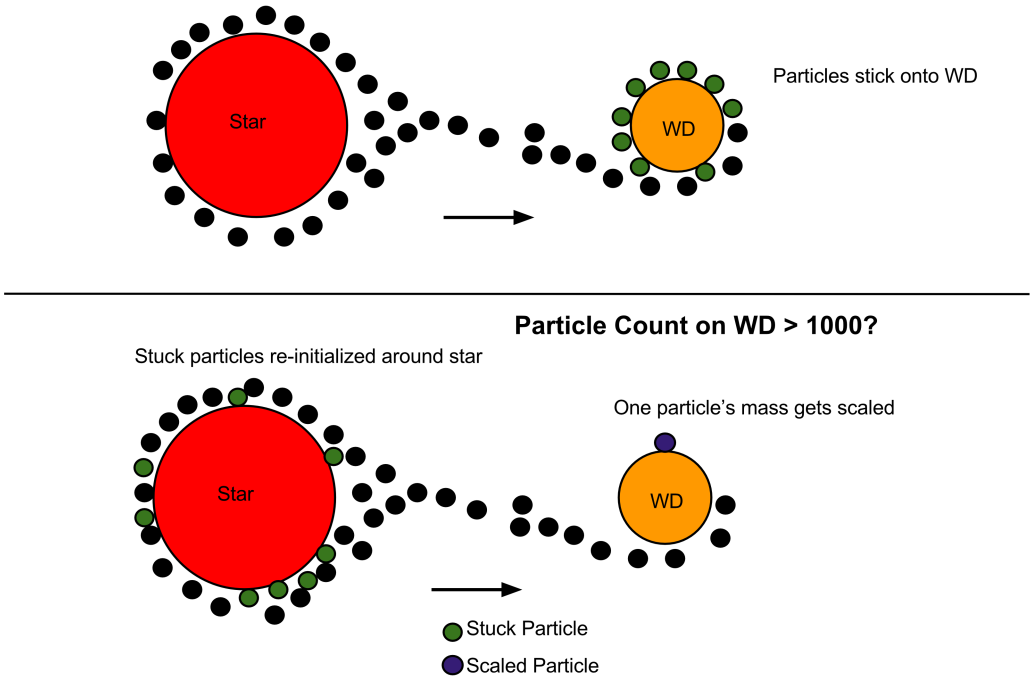


Figure 8: Depicts the Particle Recycling Method. Shows accretion of gas particles onto the surface of the white dwarf and recycling that occurs when the surface particles reach the particle count limit on the white dwarf.

Each particle has a state code which represents the role the particle has in the simulation at a given time. A state flag of -1 is given to inactive particles (later shown in figure 9), 0

is given to particles in the gas shell surrounding the star, 1 is given to the particles outside the shell, within the limits of $minDisk$ and $maxDisk$, 2 is given to particles stuck on the white dwarf, and 3 is given to particles which have been exploded. At each time step the program loops through all particles and counts the number in each state flag. The recycling method is not only applied to just the white dwarf, it is also applied to particles which exceed $maxDisk$ which are then recycled back to the star in order to prevent the loss of particles during the simulation.

Even with the recycler method, the particle count in the star's shell will decrease over time, due to the accretion disk forming around the white dwarf. To counteract this, a blocking method was introduced. The program is ran with an initial particle count (n), a total maximum particle count (max_n), a block size (blk_size), and a desired particle count on the shell. At each file write interval, the program will check the particle count on the star, and compare it to the desired particle count, if the current count is lower, it will add a particle block to the simulation. At time zero, the program initializes all the particles, including the initially inactive block. The initially inactive (reserve) block ($max_n - n$) is divided into sub-blocks with the given sub-block size (typically $blk\_size = 64$). The program can then activate these sub-blocks as needed to maintain the desired particle count in the star's shell.

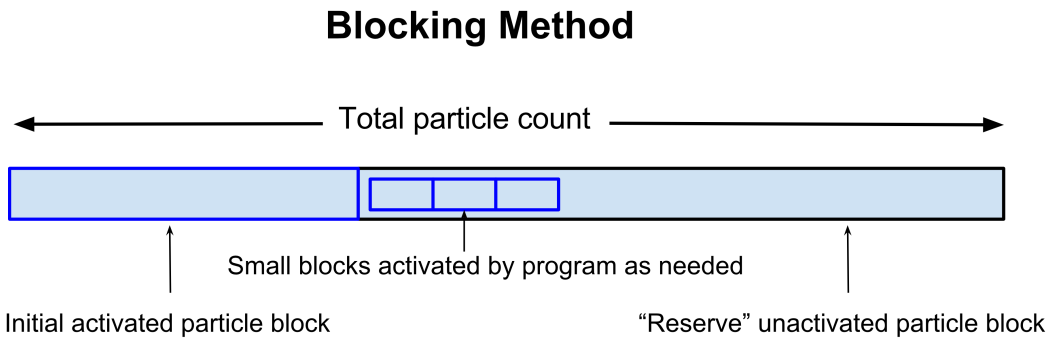


Figure 9: The large block in blue represents the initially activated block, whereas the black block represents the reserve particles. Small sub-blocks shown in blue are activated as needed by the program.

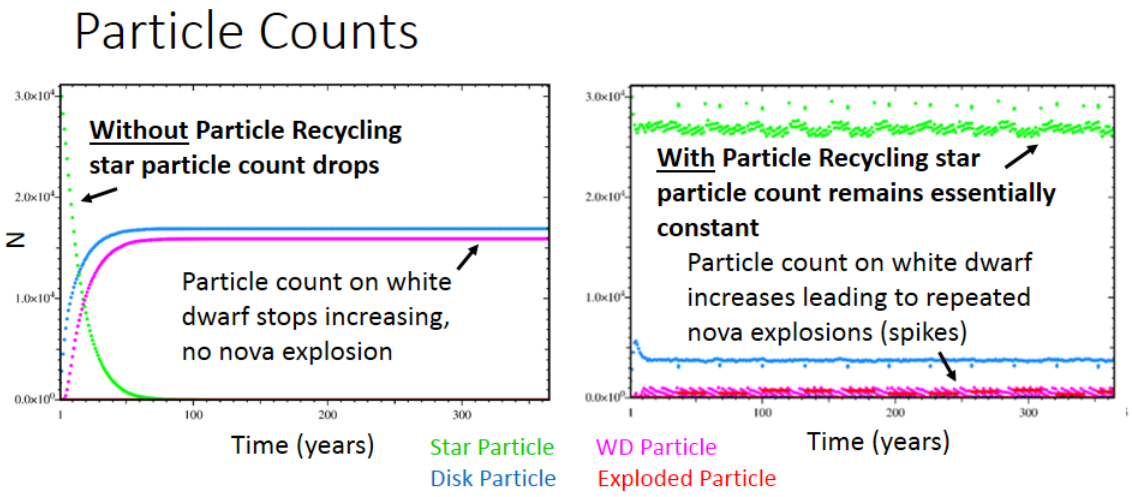


Figure 10: Particle counts over time, comparing with particle recycling method to without particle recycling.

As seen in figure 10, particle recycling is necessary to achieve a repeating nova simulation, and to have a steady particle count on the star. Without the recycling method, the particle count on the star drops off; whereas, with the particle recycling method, the star particle count stays steady which also allows for steady accretion rates. A nova explosion was not achieved without the particle recycling method because there were no more particles left to be pulled off from the star. With particle recycling, there is a small number of particles on the white dwarf because they are getting scaled and recycled back to the star.

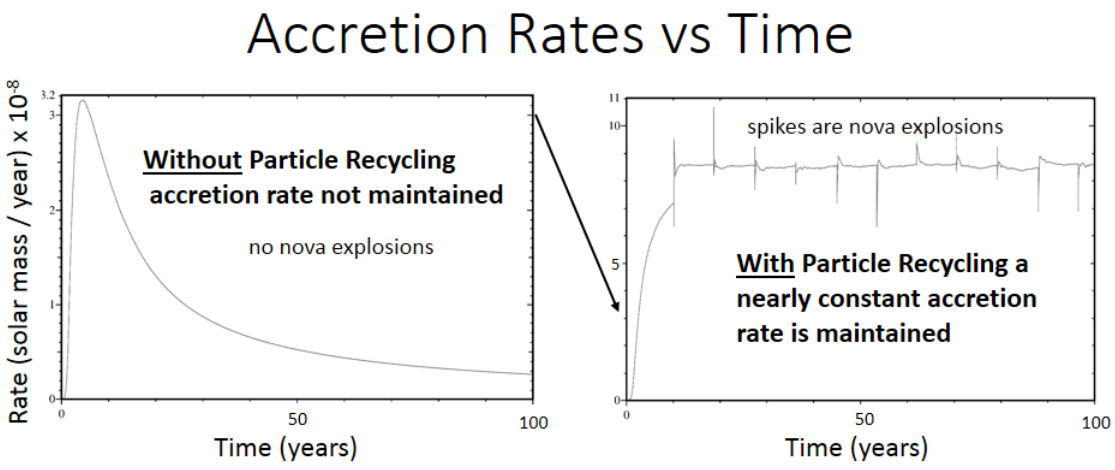
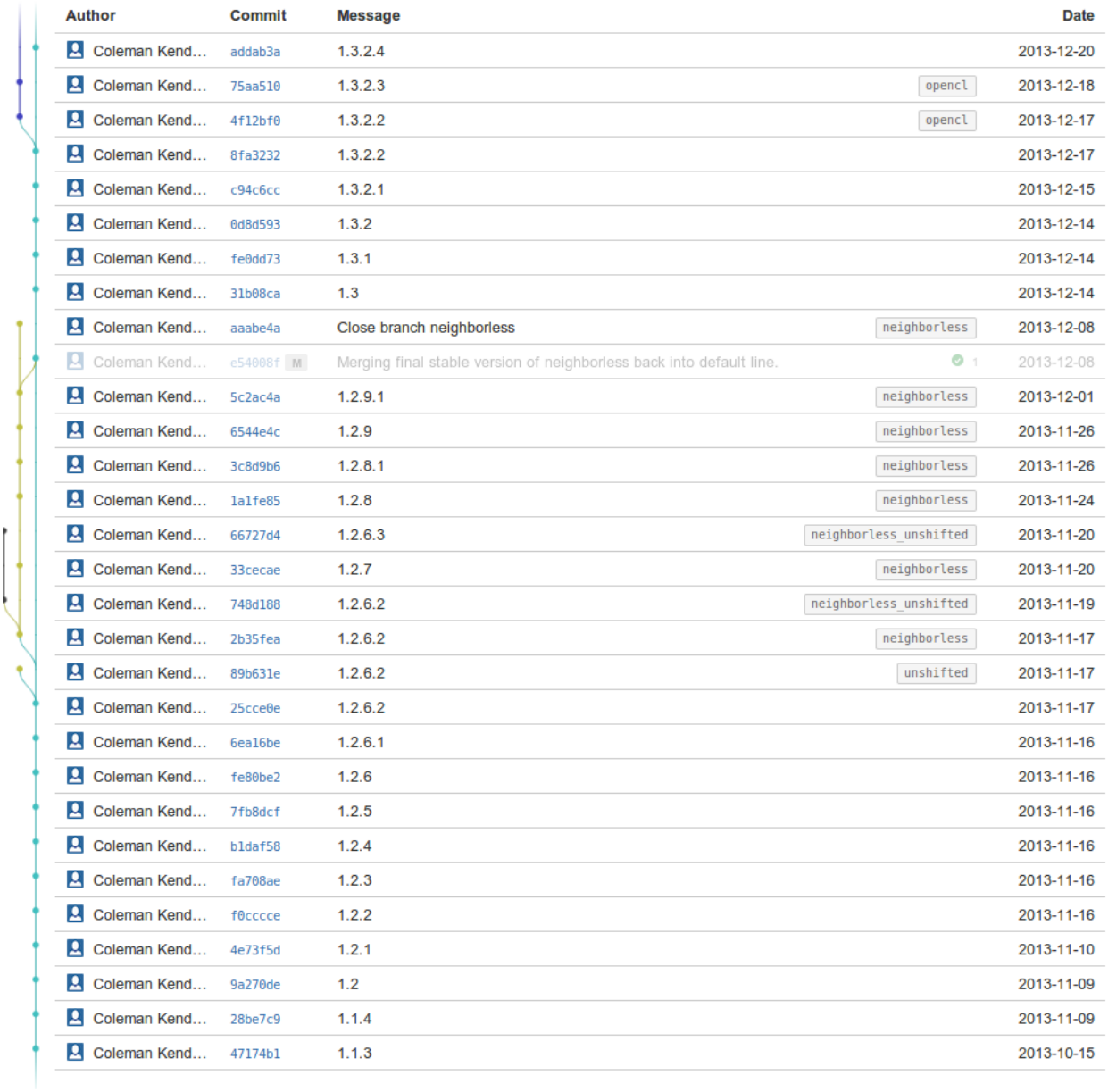


Figure 11: Accretion rate over time, nova explosions are able to occur with particle recycling as compared to no nova explosions without particle recycling.

Without particle recycling, the accretion rate drops after all the particles on the star have been stuck onto the white dwarf. With particle recycling, the accretion rate is able to increase and then maintain a nearly constant value over time, allowing the system to explode.

5 Code Development History

Initially, the program consisted of the star and white dwarf orbiting in a binary system. Next I focused on creating a stable star. Particles were randomly distributed around a single mass point to represent the core. An exponential wall force was added to model the back pressure. Later on, this was changed to the Wendland weight in order to get a better wall function, and one that was quicker to evaluate. Once the stable star was achieved, the white dwarf was added to the model. The rotating frame was added in order to simulate the orbit between the two. After running the simulation, there was no nova explosion happening. This led to the recycling and blocking methods, as there were not enough particles collecting on the surface of the white dwarf to initiate a nova explosion. Throughout the code development process, the file output format was constantly changing in order to allow for different variables to be saved. A post-processing program was written so that the data files output by the program could be read in at a later time. This program was constantly being updated in order to read the latest format. The data files are read in, and then depending on the input parameters, were output accordingly. One of the main features of this program was the ability to output ParaView file formats from my data. I included support for multiple file formats (.csv, .xyz, and .vtk), however the VTK file extension was used the majority of the time since it allowed for color plotting of the data and generating mpeg movies of the simulation. As mentioned earlier, the program gives codes to each particle. This was adapted into my processing program so that you could specify which particular code (particle type) you wanted to process. I used Mercurial to do revision control over the code, so that I could implement new features in branches and choose whether or not to merge them into the default branch. Bitbucket was used as a hosting for the Mercurial repository.



Author	Commit	Message		Date
Coleman Kend…	addab3a	1.3.2.4		2013-12-20
Coleman Kend…	75aa510	1.3.2.3	opencl	2013-12-18
Coleman Kend…	4f12bf0	1.3.2.2	opencl	2013-12-17
Coleman Kend…	8fa3232	1.3.2.2		2013-12-17
Coleman Kend…	c94c6cc	1.3.2.1		2013-12-15
Coleman Kend…	0d8d593	1.3.2		2013-12-14
Coleman Kend…	fe0dd73	1.3.1		2013-12-14
Coleman Kend…	31b08ca	1.3		2013-12-14
Coleman Kend…	aaabe4a	Close branch neighborless	neighborless	2013-12-08
Coleman Kend…	e54008f M	Merging final stable version of neighborless back into default line.	1	2013-12-08
Coleman Kend…	5c2ac4a	1.2.9.1	neighborless	2013-12-01
Coleman Kend…	6544e4c	1.2.9	neighborless	2013-11-26
Coleman Kend…	3c8d9b6	1.2.8.1	neighborless	2013-11-26
Coleman Kend…	1a1fe85	1.2.8	neighborless	2013-11-24
Coleman Kend…	66727d4	1.2.6.3	neighborless_unshifted	2013-11-20
Coleman Kend…	33cecae	1.2.7	neighborless	2013-11-20
Coleman Kend…	748d188	1.2.6.2	neighborless_unshifted	2013-11-19
Coleman Kend…	2b35fea	1.2.6.2	neighborless	2013-11-17
Coleman Kend…	89b631e	1.2.6.2	unshifted	2013-11-17
Coleman Kend…	25cce0e	1.2.6.2		2013-11-17
Coleman Kend…	6ea16be	1.2.6.1		2013-11-16
Coleman Kend…	fe80be2	1.2.6		2013-11-16
Coleman Kend…	7fb8dcf	1.2.5		2013-11-16
Coleman Kend…	b1daf58	1.2.4		2013-11-16
Coleman Kend…	fa708ae	1.2.3		2013-11-16
Coleman Kend…	f0cccce	1.2.2		2013-11-16
Coleman Kend…	4e73f5d	1.2.1		2013-11-10
Coleman Kend…	9a270de	1.2		2013-11-09
Coleman Kend…	28be7c9	1.1.4		2013-11-09
Coleman Kend…	47174b1	1.1.3		2013-10-15

Figure 12: Screenshot from BitBucket showing the commit changes over time.

6 OpenCL and MPI

A combination of both MPI (Message Passing Interface) and OpenCL [9] (Open Computing Language) are used to parallelize the simulation. MPI takes advantage over multiple cores on a CPU in a computer; OpenCL takes advantage over the GPU (Graphical Processing Unit) in a computer. Using OpenCL for the main part of the calculation decreased computational time by a factor of about 100. While an OpenCL program may be large, only the kernel is actually run on the GPU. A kernel can be thought of as a subroutine that is executed in parallel across the device. Two OpenCL devices were used: the Nvidia GTX 580 and the AMD HD7970. However, the GTX 580 was primarily used for the majority of the calculations. Computational performance benchmarks were done with my program, and a matrix-matrix-multiply in order to determine how efficient my program was with GPU resources.

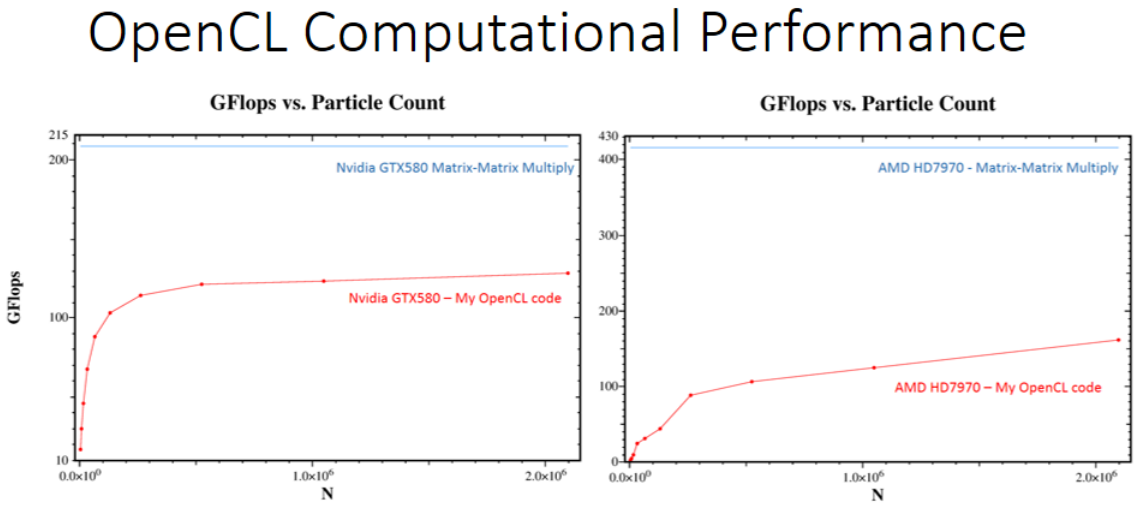
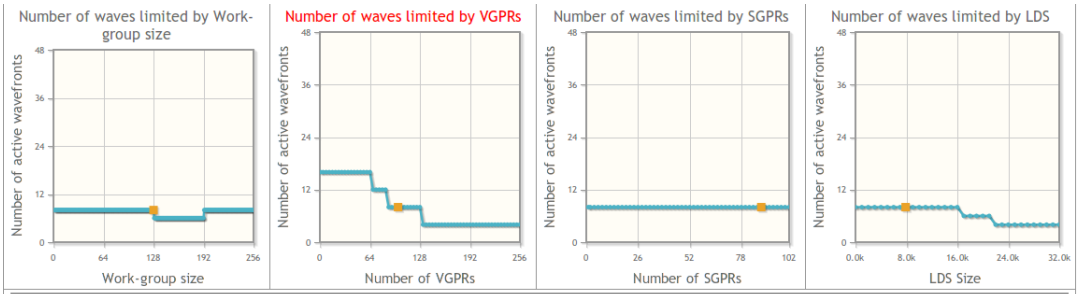


Figure 13: GFLOPS vs. number of particles. Left shows the Nvidia GTX 580, right shows the AMD HD7970.

Figure 13 shows the computational performance measured in GFLOPS (Floating Point Operations Per Second x 10^9) versus the particle count. The matrix-matrix-multiply is considered the optimal or more realistic way of measuring GPU performance rather than looking at the theoretical FLOP yield for a device. The Nvidia GTX 580 performs at about 208 GFlops whereas the AMD HD7970 performs at about 415 GFlops for the matrix-matrix-multiply. My program reaches a peak of about 130 GFlops on the GTX580, and flattens out quickly as the particle count is increased. This means that my program is able to saturate the GPU with low particle counts, rather than only using a small portion of the device. My

program reaches a peak of about 180 GFlops on the HD7970, however the low particle count performance is much worse than the GTX580. For this reason, the GTX580 was used for the majority of calculations, and the HD7970 was only used on simulations where the particle count exceeded 2 million. The main difference in the performance between the two cards is their architectures. The GTX580 uses the Nvidia Fermi GF110 chip, whereas the HD7970 uses the AMD GCN (Graphics Core Next) architecture. Due to the GCN register limits, my program was not able to achieve full kernel occupancy (using AMD's CodeXL profiler [10]), which means that it is not utilizing the full GPU because of too much vector and scalar registers (VGPR and SGPR).



Variable	Value	Device Limit
Device Info		
Device name	Tahiti	
Number of compute units	32	
Max number of waves per compute unit	40	
Max number of work-groups per compute unit	16	
Wavefront size	64	
Kernel Info		
Kernel name	core_grav	
Vector GPR usage per work-item	100	256
Scalar GPR usage per work-item	88	102
LDS usage per work-group	8064	65536
Flattened work-group size	128	256
Flattened global work size	32768	16777216
Number of waves per work-group	2	4
Kernel Occupancy		
Number of waves limited by Vector GPR and Work-group size	8	40
Number of waves limited by Scalar GPR and Work-group size	20	40
Number of waves limited by LDS and Work-group size	16	40
Number of waves limited by Work-group size	32	40
Limiting factor(s)	VGPR	
Estimated occupancy	20%	

Figure 14: Kernel performance analysis from the AMD CodeXL application [10]

Figure 14 shows the profiled values from the kernel, against a HD7970. The second box shows that the kernel is limited by the amount of VGPRs. This profiler was used quite frequently to update the kernel with new features. Using this profiler helped create optimal kernels for the device. However, as depicted above, decreasing the amount of VGPRs the kernel uses would require creating separate kernels to break up the computation into steps. This was done, however the memory overhead made it slower than one kernel to do everything.

When running my program on a single core of a CPU, about 2 GFlops was achieved on a Intel Core i7 3770K overclocked to 4.3Ghz. The dramatic difference in performance shows

that OpenCL really is needed in order to do calculations like these.

Parallelization using MPI is relatively simple, it can be done by splitting up the particle count among all the cores of a CPU. The number of particles per core is determined by

$$n_{PerCore} = n/n_{cores} \tag{8}$$

where $n_{PerCore}$ is the amount of particles each processor will compute, n is the total particle count, and n_{cores} is the number of cores running the program.

A traditional loop on a single core would behave as such:

for $i = 0$ to n step 1 do

When MPI is ran on a program, each core essentially runs a copy of that program, so each core will have a copy of all the program variables. However, it is necessary to synchronize the processors by sharing the data amongst them whenever they will be needing information that another core is computing. In my program, only the base core ($id = 0$) creates the particle distributions and initializes all information. Afterwards, that core will broadcast its information to all the other processors, so that every core has all the information before computing. The loop below shows how the full particle count is split up among all the cores.

for $i = my_{CoreId} * n_{PerCore}$ to $(my_{CoreId} + 1) * n_{PerCore}$ step 1 do

The above loop is significantly faster than the previous example, since each processor is only doing computations on blocks of $n_{PerCore}$ particle, rather than n particles. A consequence of splitting up the calculation among cores is that each core will only need to update its block of particles. At each file write interval, the base core writes all the simulation data to file. In order to do this with MPI, it needs to get the other cores variables for its particle block. This is done by using a MPI Gather [11] to receive all the information onto the base core. For large particle counts, sending and receiving data could take quite awhile, but often the performance boost in the calculation phase overcomes the cost of sending and receiving data.

Rather than the whole code being run on multiple cores, like in MPI, OpenCL only executes a designated kernel across all GPU Cores. A drawback of OpenCL is that it requires much more overhead in order to launch a kernel. In the main program, buffers have to be created and copied to the device, and a OpenCL command queue, context, and platform have to be created [12]. The kernel is also compiled by OpenCL at runtime. In OpenCL, synchronizing memory is done by copying the buffers from the host to the device, and vice versa. This is done through the PCI express lane, however it is still not quite as

fast as memory transfers in MPI. In order for OpenCL to achieve maximum performance increases, the host code should call kernels repeatedly and then read data back occasionally, rather than reading and writing memory at each kernel call. When calling a kernel, a global and local work size is specified. The global work size tells the GPU how many workitems will be executed (essentially thought of as individual kernel calls), and the local size defines the group size of the workitems (called a workgroup). Each workgroup has a local memory space, in which data can be shared between each workitem in the group. The local size is usually 512 for Nvidia cards, and 64 for AMD cards, however changing the local size can cause performance benefits as well. Each GPU has its own maximum global work size and local sizes, which cannot be exceeded. If the total particle count is greater than the global work size, then the problem needs to be split up. Each workitem has a global memory space (a large shared space across the GPU - slow to access!), a local memory (about 32kB of shared memory across the workgroup - fast to access), and private memory (very small memory, only accessible by the workitem). Since global memory is very slow, the current particle variables are read in and then stored in either local memory or private memory for the remainder of the calculation. At the end of the kernel, the temporary copy is written back out to global memory where it can then be accessed by the CPU. A very simple 1D example OpenCL program is outlining this behavior below.

A simple kernel structure and memory access outline:

```
begin
  i = get_global_id                                   Get workitem IDs
  j = get_local_id
  for block = 0 to GlobalSize step 1 do               Loop over blocks
    global_pointer = GlobalSize * block + i           Pointer to global memory
    x' = x[global_pointer]                            Store a local copy
    comment: Use the local copy in any calculations
    x[global_pointer] = x'                            Write local copy back to global memory
  od
end
```

Figure 15 shows an flowchart of the kernel that is called at each time step, and a brief outline of what the host code and post processing programs do. A fast random number generator [5] and fast math subroutines (square root and inverse square root) [13] were used in the program to significantly improve performance.

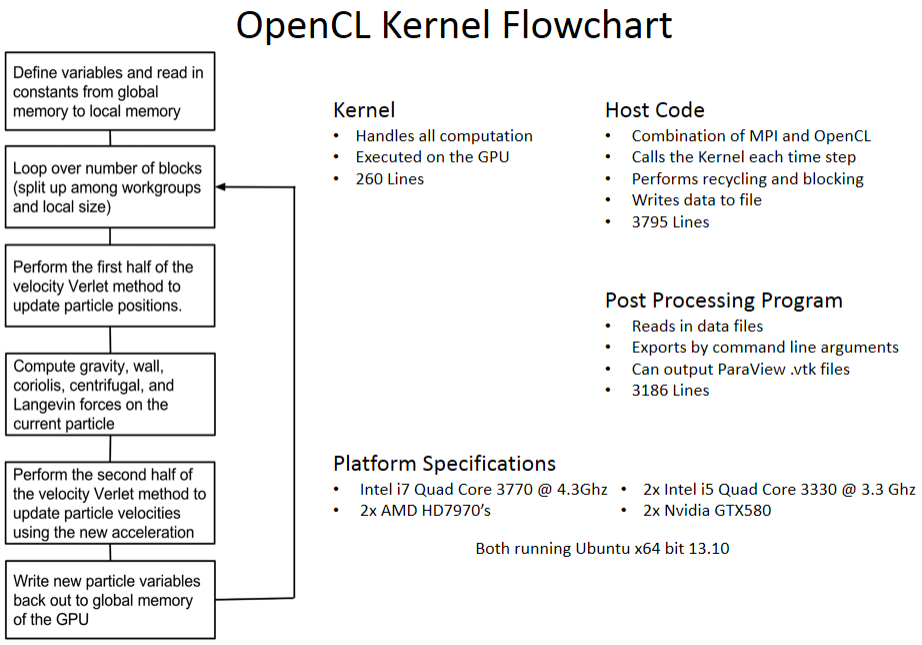


Figure 15: OpenCL Kernel flowchart. This is the main calculation phase of the program.

7 Moving Least Squares

Hydrodynamic methods in astrophysics simulations often involve the use of SPH [14] (Smooth Particle Hydrodynamics). However, Moving Least Squares [15] (MLS) is another particle based method that is more general and accurate than SPH. MLS is based on a local fit within a neighborhood (of radius r) using a set of polynomial functions. MLS has advantages over using a mesh based method, however its computational requirements can be large. A particle based method can handle dramatic deformations easier than a mesh, which is needed for the nova simulation. A 3D polynomial function set is shown below

$$1, x, y, z, x^2, y^2, z^2, xy, xz, yz$$

The polynomial basis is used to construct and diagonalize a matrix (seen below in figure 16 from [16]). The sum over i in figure 16 extends over only the particles within the local neighborhood. The matrix inverse is used to construct shape and derivative shape functions.

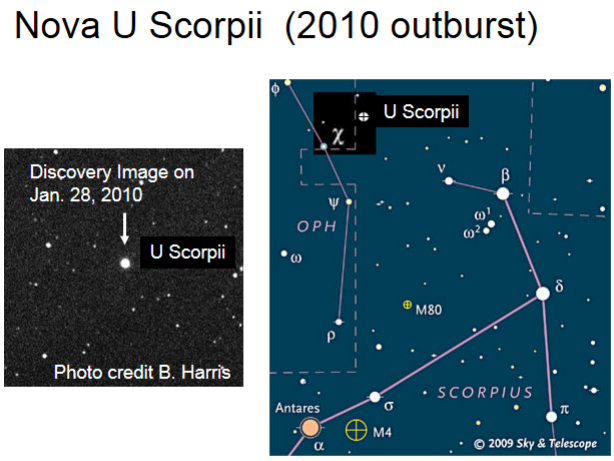
$$\sum_i \begin{bmatrix} 1 & x_i & y_i & x_i^2 & x_i y_i & y_i^2 \\ x_i & x_i^2 & x_i y_i & x_i^3 & x_i^2 y_i & x_i y_i^2 \\ y_i & x_i y_i & y_i^2 & x_i^2 y_i & x_i y_i^2 & y_i^3 \\ x_i^2 & x_i^3 & x_i^2 y_i & x_i^4 & x_i^3 y_i & x_i^2 y_i^2 \\ x_i y_i & x_i^2 y_i & x_i y_i^2 & x_i^3 y_i & x_i^2 y_i^2 & x_i y_i^3 \\ y_i^2 & x_i y_i^2 & y_i^3 & x_i^2 y_i^2 & x_i y_i^3 & y_i^4 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} = \sum_i \begin{bmatrix} 1 \\ x_i \\ y_i \\ x_i^2 \\ x_i y_i \\ y_i^2 \end{bmatrix} f_i.$$

Figure 16: Symmetric 2D MLS matrix constructed from a polynomial function set. [16]

The MLS derivative shape functions are used to solve the hydrodynamic equations for the conservation of mass, momentum and energy.[15] This calculation is repeated for many overlapping neighborhoods in order to cover the entire computational space. The MLS method is partly implemented on the CPU, however due to incredibly slow computation times, it is in the process of being parallelized using OpenCL, and will be finished during future work.

8 U Scorpii

This projects focus is simulating the nova U Scorpii, which is a recurrent nova that has an explosion period of about 10 years on average. U Scorpii (U Sco) is about 20,000 light years from Earth, and is one of the best studied recurrent nova due to its short explosion period.[17, 18, 8] The most recent outburst was in 2010, although there have been three more outbursts since 1975, as seen in figure 17.[19]



Discovery image (left), sky chart (right) showing the location of Nova U Scorpii.

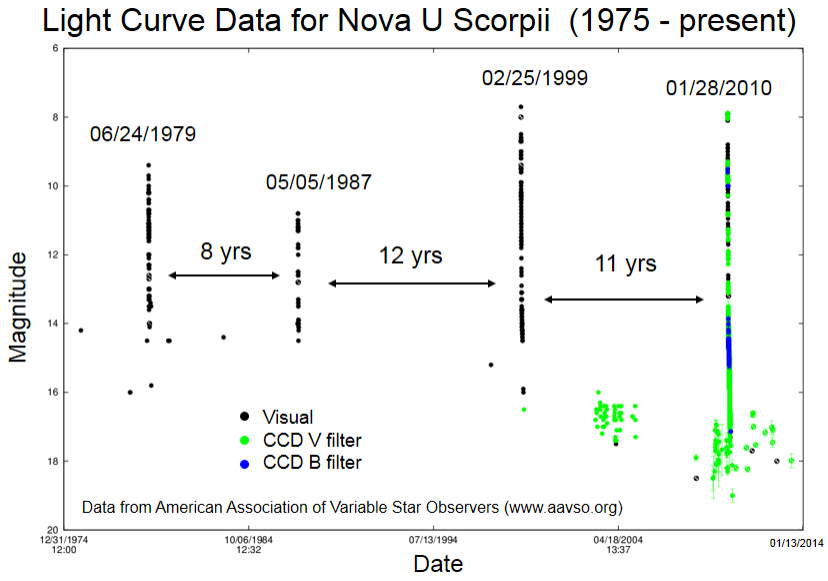


Figure 17: Light curve data from 1975-present. Each spike shows an outburst.

9 Results

The primary goal of this project is to create a 3D computer model to accurately simulate the formation and life cycle of a nova. The model was applied to the U Scorpii nova and the effects of various model parameters on the nova evolution from accretion to explosion were studied. Many simulations were run with 32,768 particles to adjust these parameters. Simulations with one million particles were also run (which are shown below in figures 18-21). The light curve computed from my simulation was fit to the nova U Scoropii (see equations 3 - 5) by adjusting the initial temperature (T_o) and cooling rate (a) of the exploding hydrogen gas shell. The simulation parameters were then adjusted to study their effects. The main parameters adjusted were the white dwarfs mass, the star mass, and the separation distance between the white dwarf and star. Another goal of this project is to study the entire life cycle of a nova. By developing the particle recycler and blocking methods, my simulation model can compute the actual mass accretion rate onto the white dwarf. Due to the large differences in mass and time scales, most professional nova simulations do not compute the actual accretion rate but take it to be an input parameter.

By changing the system's parameters, the accretion rate will change which can also be used to predict when the nova system will explode. The base case for U Scorpii was run to see how the system explodes normally, this is shown below in figures 18-21. Blue is used to show the particles that are in the star shell, red is used to show particles that are in the accretion disk and that are stuck to the white dwarf. A tan color is used to show particles in the explosion shell. In figure 18, the system is shown before explosion. The area between the star and the white dwarf has been filled in by particles due to the white dwarf pulling them off the shell. On the far right side, a small red ball can be seen which shows all the particles accumulating onto the surface of the white dwarf. Figure 19 shows the nova system just 100 seconds after the explosion. The gas shell will continue to expand outward, hitting the star on its way. 14 minutes after explosion the shell has moved through the accretion disk and is shown hitting the shell of the star in figure 20. After 22 minutes have passed since the explosion, in figure 21, the shell has moved through the accretion disk and shell, and a resulting flat edge develops.

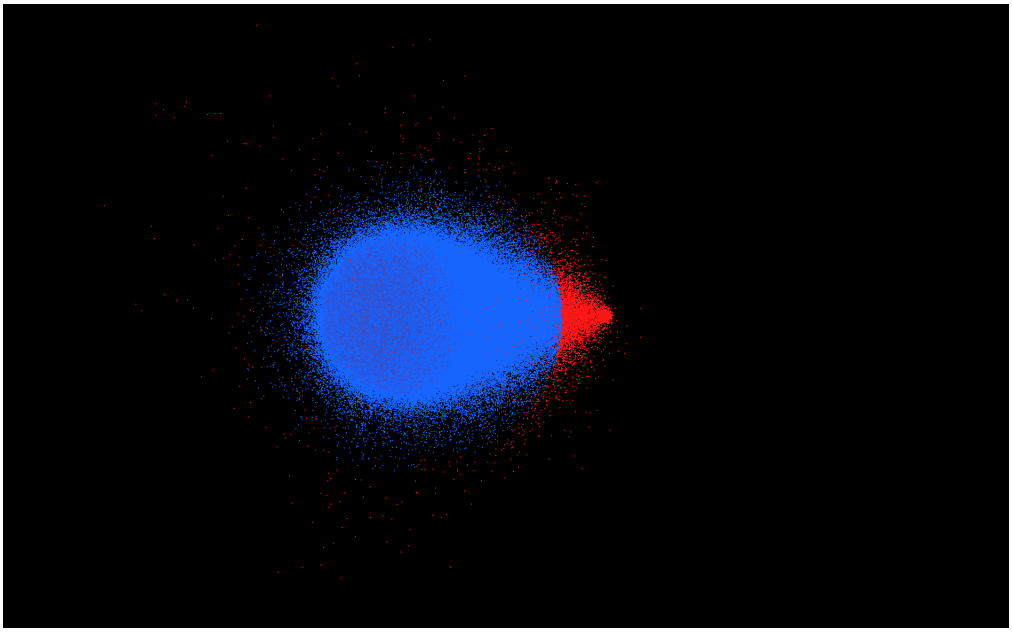


Figure 18: Nova U Scorpii before explosion.

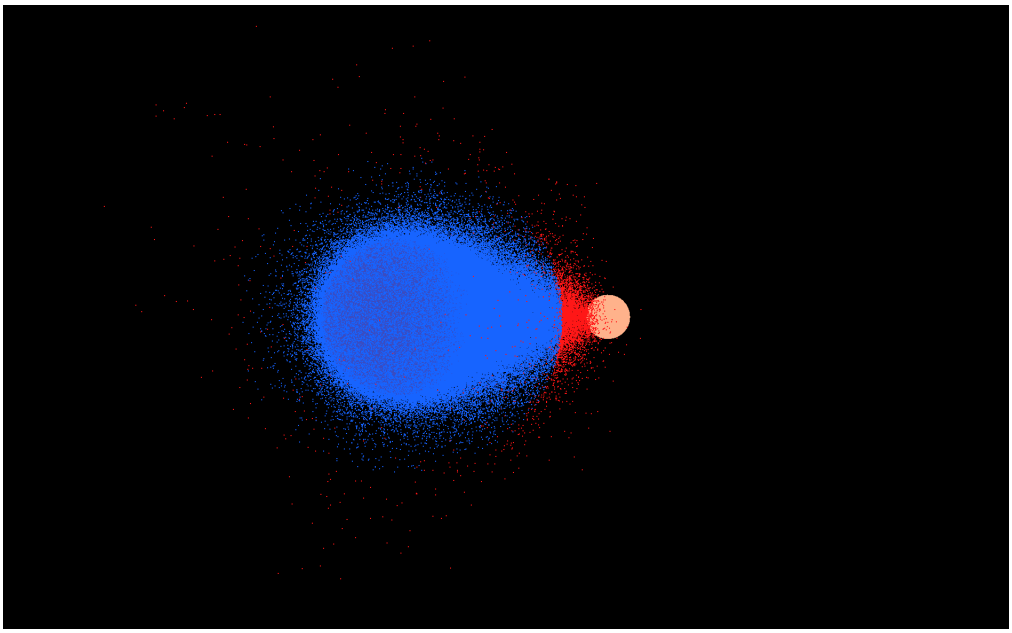


Figure 19: Nova U Scorpii 100 seconds after explosion.

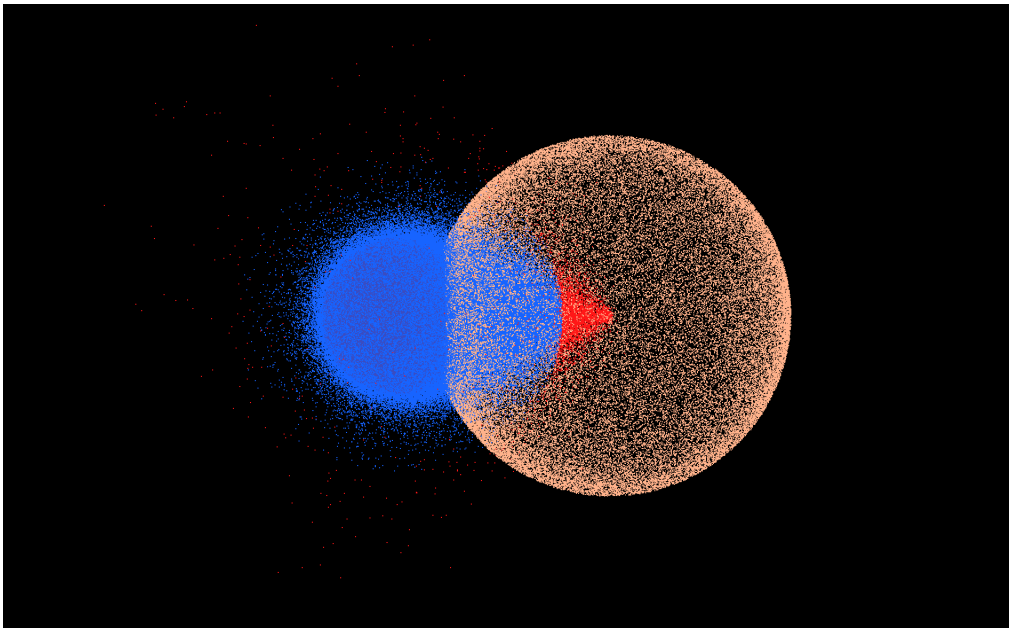


Figure 20: Nova U Scorpii 14 minutes after explosion.

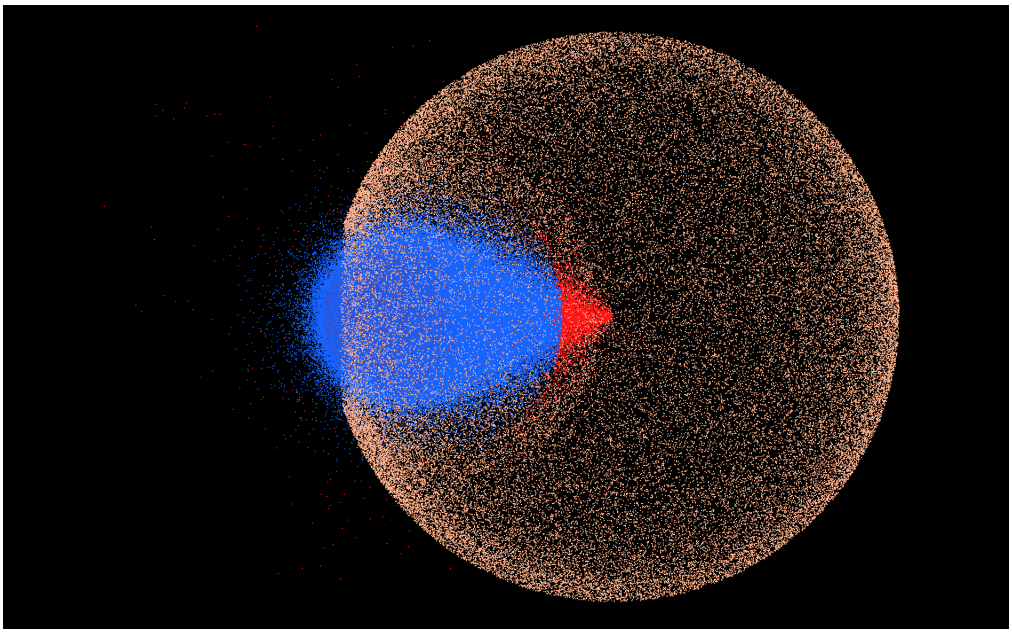


Figure 21: Nova U Scorpii 22 minutes after explosion.

The resulting flat edge in the simulation is shown below in figure 22, compared to a professional simulation [18]

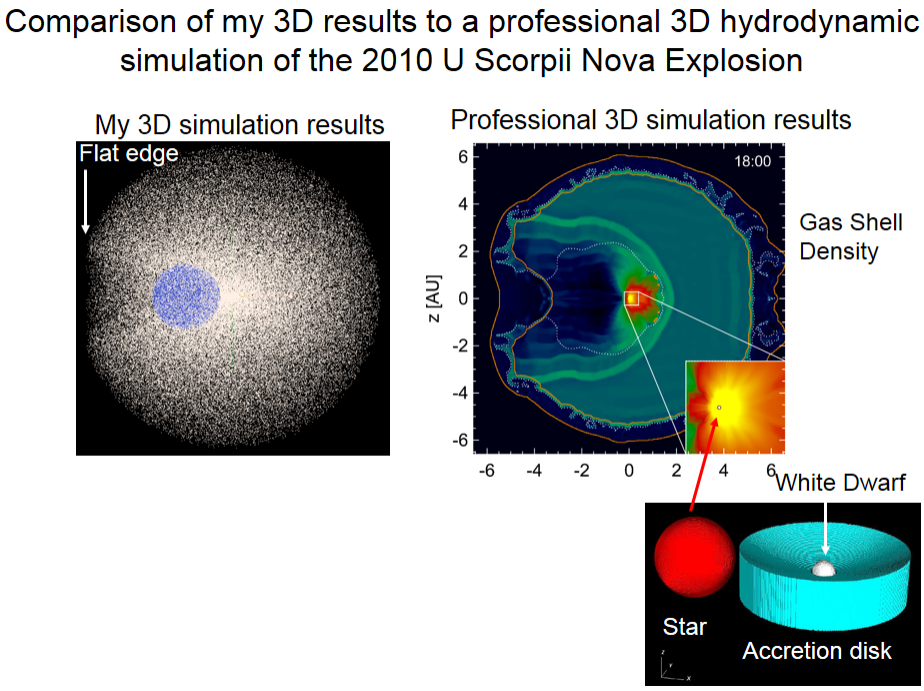


Figure 22: Flat edge from my simulation (left) compared to a 3D professional simulation [18]

Figure 23 shows my model against the U Sco 2010 outburst. The light curve for U Scorpii is shown in black, my simulation fit is shown in green. This light curve is my best fit to the data and was the basis of my comparisons with different model parameters.

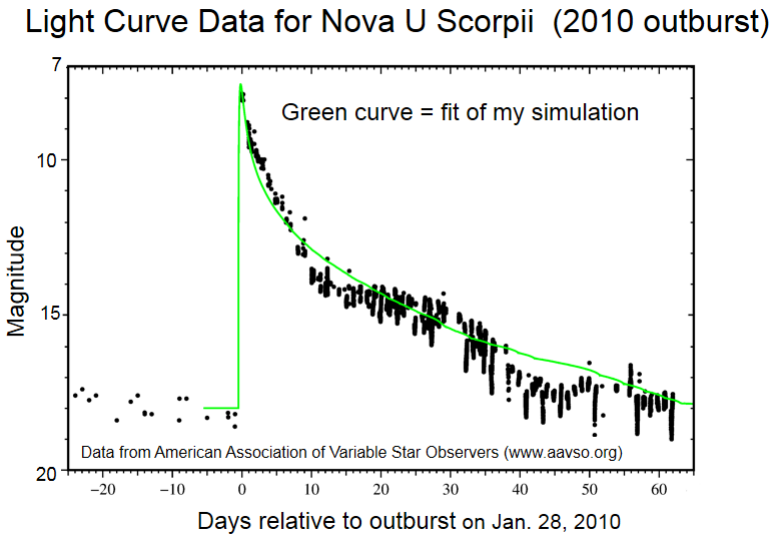


Figure 23: My simulation (green) and light curve data (black) from the 2010 outburst.

The first parameter adjusted was the white dwarf mass. As predicted, a heavier white dwarf mass results in a smaller time between nova explosions, and a smaller white dwarf mass results in a longer time between explosions. Figure 24 and 25 compare my simulation results to professional hydrodynamic models.

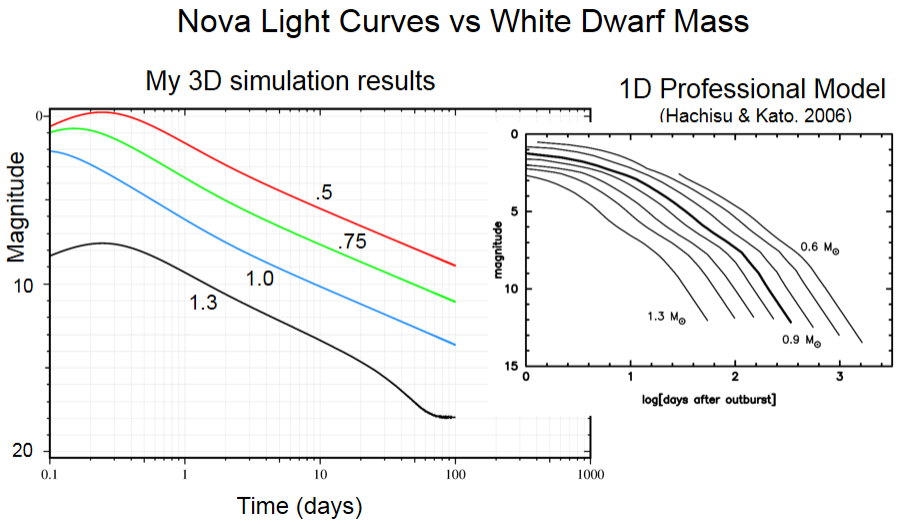


Figure 24: Light curves with different white dwarf masses compared to a 1D professional model [20]

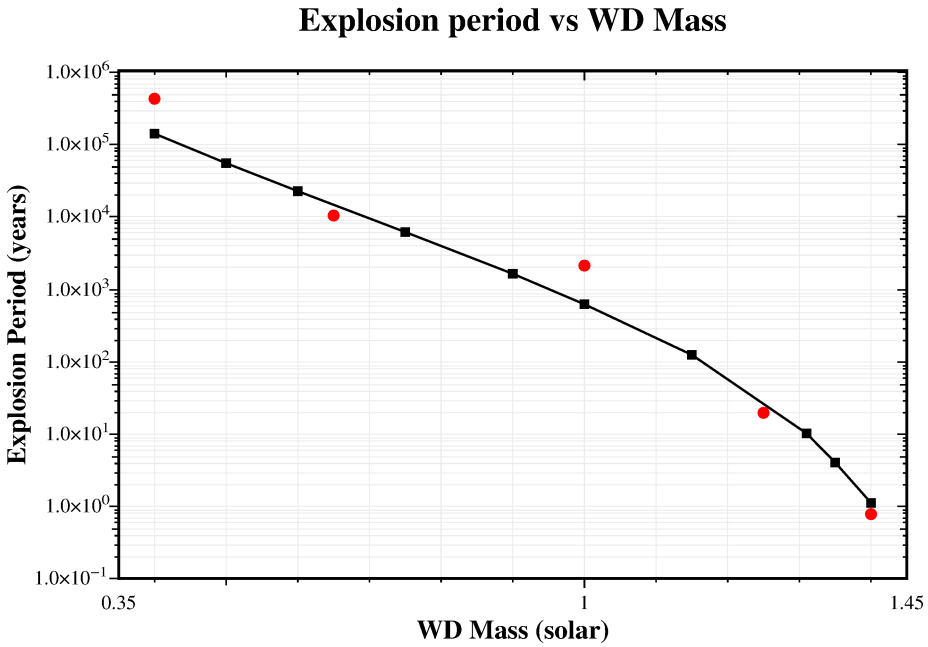


Figure 25: Nova explosion period vs. white dwarf mass. My simulation results plotted in black, a professional 1D hydrodynamic model [21] is plotted in red.

As seen in figure 24, heaver white dwarf masses make the curve fall off faster after explosion due to a greater gravitational pull from the white dwarf, and less hydrogen gas accreted onto the surface. Since the gravitational pull is much stronger from a larger mass white dwarf, it keeps the shell from expanding outward as much as a lighter mass white dwarf would. With a lighter mass white dwarf, the opposite occurs. There is more hydrogen gas accreted onto the surface at explosion time (see 26, which relates to more hydrogen gas burn during the explosion. With less gravitational pull, and more temperature from the hydrogen burn, the light curve peaks higher and falls off much slower.

A heavier mass white dwarf takes less time for the hydrogen gas to reach the critical temperature and density due to the small volume of the gas shell. Since the gas shell in a lighter mass white dwarf has much more volume (as seen in figure 26), it takes much longer for the nova to reach the critical conditions. The time between nova explosions is plotted in figure 25, which shows a heavier mass white dwarf has a much smaller period between explosions when compared to a lighter mass white dwarf.

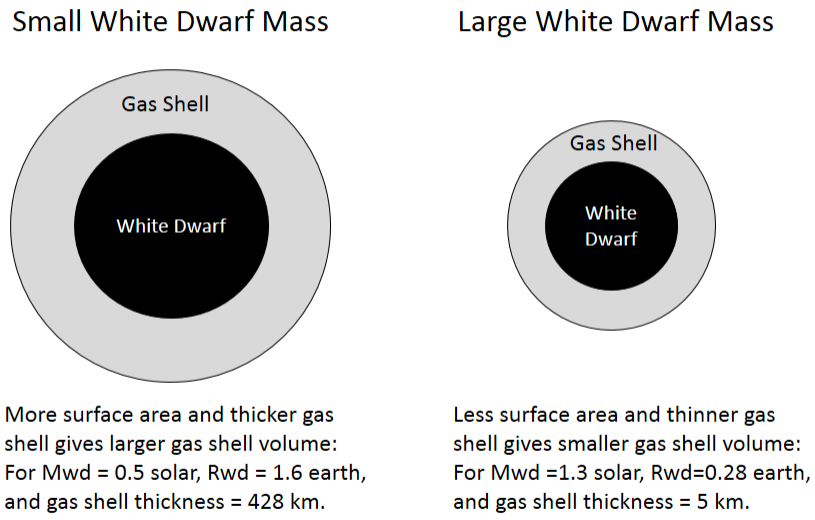


Figure 26: Scale differences between a small white dwarf mass, and a large white dwarf mass

Other parameters that were varied include the separation distance between the star and white dwarf, and the star mass. In figure 27, the nova explosion period greatly increases as the separation distance is increased. This makes sense, because the gravitational force from the white dwarf is reduced which decreases the accretion rate. Similarly, in figure 28,

the explosion period increases with a heavier mass star because the gravitational pull on the particles by the sun is greater, keeping the white dwarf from accreting them as easily.

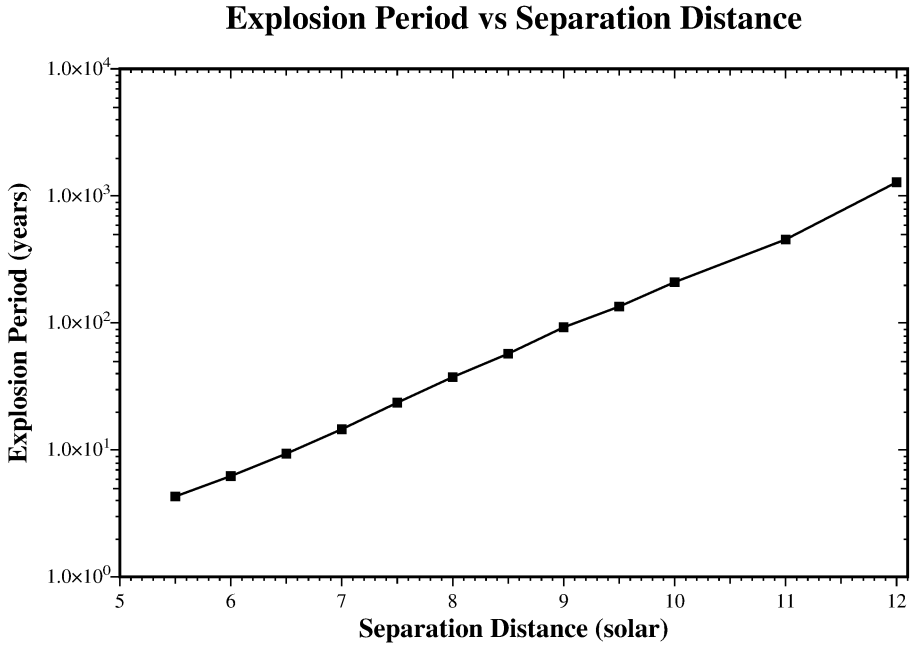


Figure 27: Explosion period of the nova versus the separation distance.

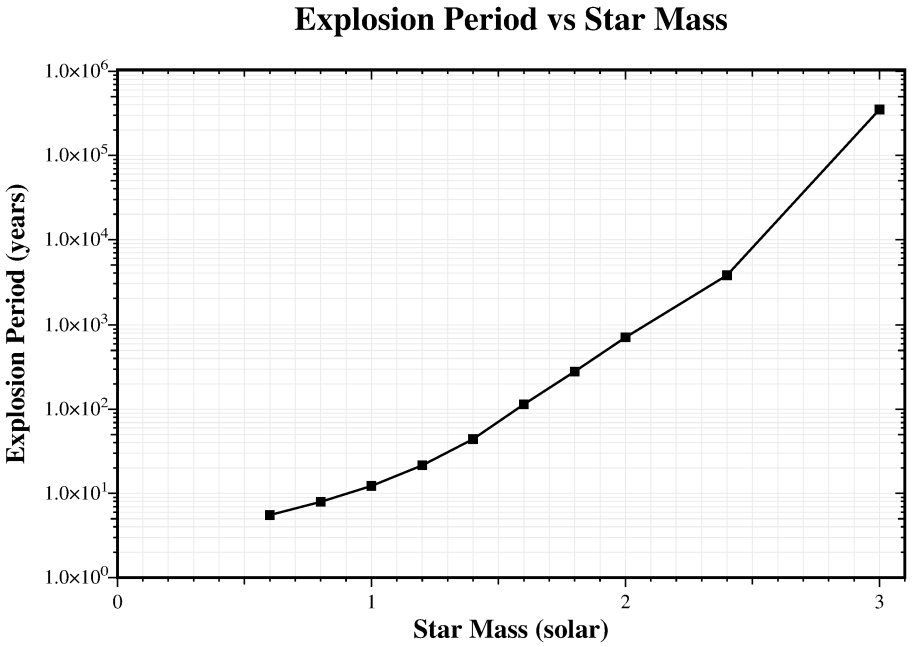


Figure 28: Explosion period of the nova versus the star mass.

10 Conclusions

I was able to successfully create a 3D computer simulation of the full life cycle of a nova including its accretion and explosion phases. The development of a new particle recycler method was required in order to compute stable and accurate accretion rates. My nova model was successfully applied to the recurrent nova U Scorpii which explodes about every 10 years. My simulations were able to reproduce the experimentally observed light curve. The effects of the white dwarfs mass on the explosion frequency and the light curve were also investigated and my results are in good agreement with professional simulations. My results show that by using a simple model, I can achieve the same conclusions as professional hydrodynamic simulations. Understanding the life cycles of nova are important since they often evolve into a Type Ia supernova which are used as "standard candles" for measuring cosmic distances and the accelerated expansion rate of our universe. Nova are also important to understand for stellar evolution. Simulations of nova can help understand what causes unusual behavior (such as plateaus, oscillations, and sudden dips [8]) that have been observed in many light curves.

11 Future Work

Future work includes studying the effects of the accretion disk on the exploding gas shell using: (1) my particle based method with friction, (2) a hydrodynamic method using MLS (Moving Least Squares) [15], and (3) include stellar wind effects in order to study nova with very large separation distances. Other novae such as Rs Ophiuchi and T Pyxidis [22, 23] will be investigated as well, to show that my model can be adapted to other novae [24]. An octree method is currently being implemented, which will be used to implement particle-particle gravity interactions, and the hydrodynamic MLS method.[25]

A Other Parameter Studies

Studies were also done on other model parameters to investigate changes on the nova simulation results due to different particle counts, particle mass, time step, different Langevin K_{eff}, and friction coefficients. Since the nova U Scorpii has an explosion period of about 10 years, simulating the full 10yr period requires a large amount of cpu time (about 10 hours on a GTX 580). In order to speed up the calculation time, mass scaling was done with the star shell particles. Figure 29 shows that scaling the star's corona mass is a way to accelerate the simulation time. For example, an increase in particle mass by a factor of 100 decreases the nova explosion period by 100. Using this approach, the 100,000 year nova period in figure 25 could be simulated.

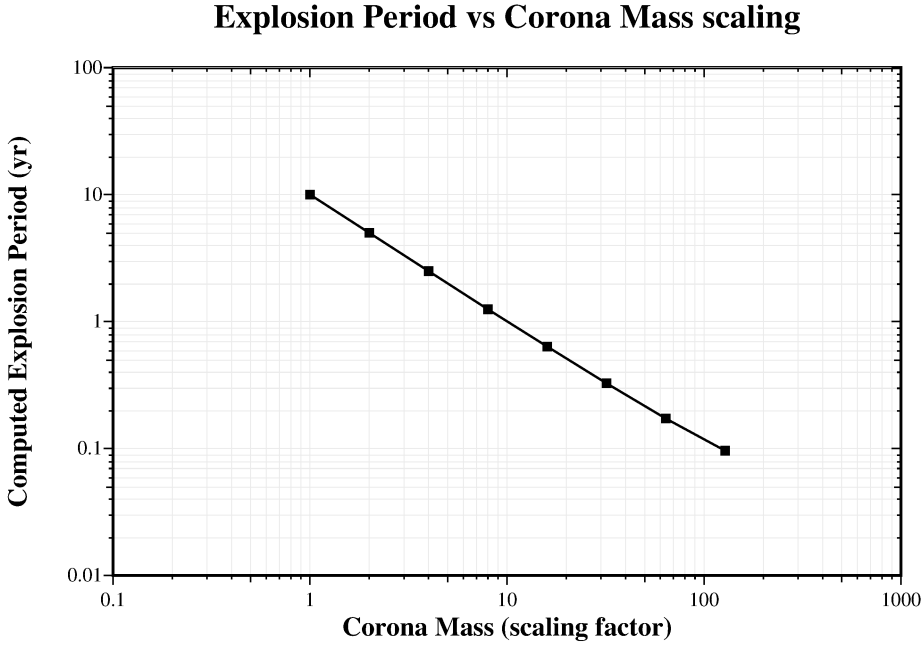


Figure 29: Explosion period vs star corona mass scaling factor. A value of 1 is the realistic case.

By changing the K_{eff} scaling, the particles will receive greater "kicks" from the Langevin thermostat force (equation 1).[4] The star shell radius will increase as a result of the larger thermostat force. This can be seen below, in figure 30. The green curve is with a scaling factor of 2, the black with a factor of 1, and red with a factor of 0.5. The normal average radius of U Scorpii is around 2.4 Solar Radii, which is shown with the black curve as the optimal K_{eff}.

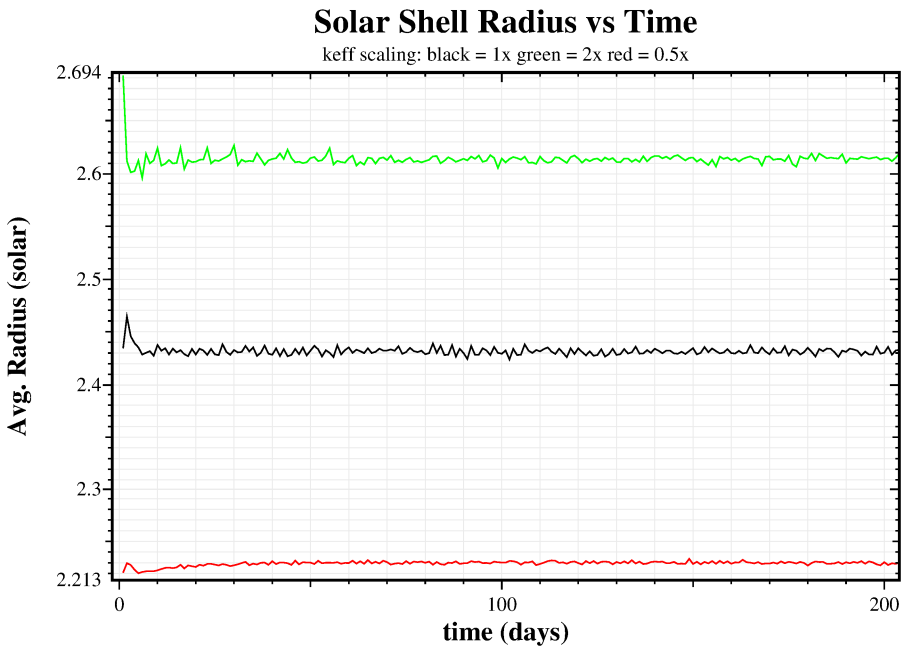


Figure 30: The average shell radius over time, shown with 3 values of K_{eff} scaling

A major issue was observed in that when the particle count changed, the accretion rates and explosion periods were changing, which resulted in a different K_{eff} value. To solve this issue, an automatic scaling conversion was included to scale the K_{eff} value depending on the particle mass. Figure 31 shows that as the particle count on the star is increased, the same light curve is produced. The black curve is with 30,000 particles (the base case), green with 60,000, and red with 130,000. All the light curves are essentially on top of each other, showing that the simulation results are stable as the particle counts are changed.

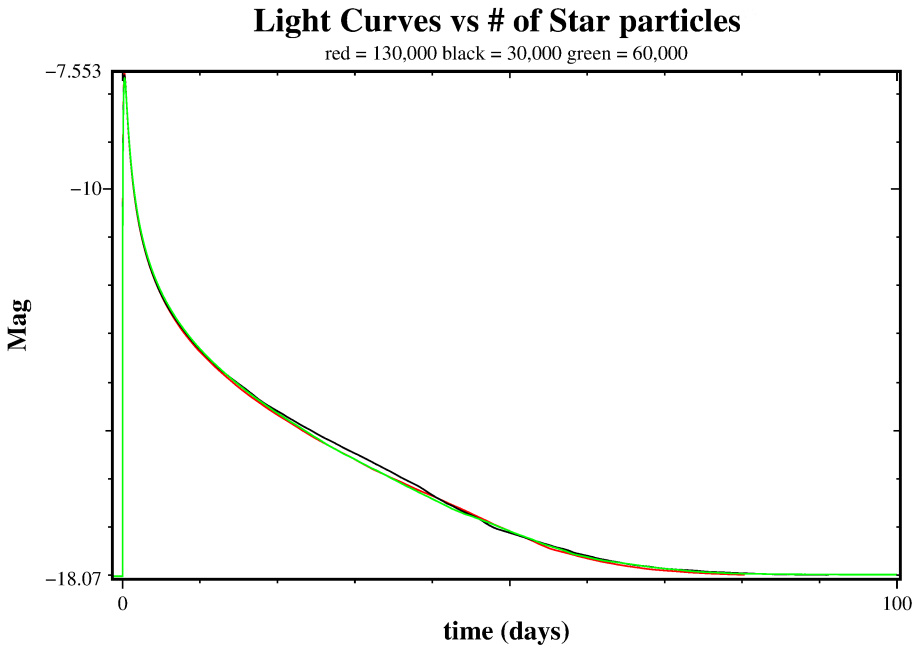


Figure 31: The light curves with different particle counts on the star shell.

With the particle recycling method, the number of particles that are active on the white dwarf surface is greatly reduced. An issue with reducing the amount of particles on the white dwarf, is that the computed light curve may not be as accurate as it should. Different cases were run to give different particle counts at the time of explosion (this was achieved by changing the $wd_{recycle}$ parameter). The red curve shows the exploded shell with about 1,300 particles. This curve produces a much lower light curve than the black curve (6,900 particles) and green curve (30,000) particles. While the particle recycling method is absolutely necessary to achieve nova explosions, recycling too much can cause bad results. A exploded shell with a particle count of greater than about 3,000 will give good results.

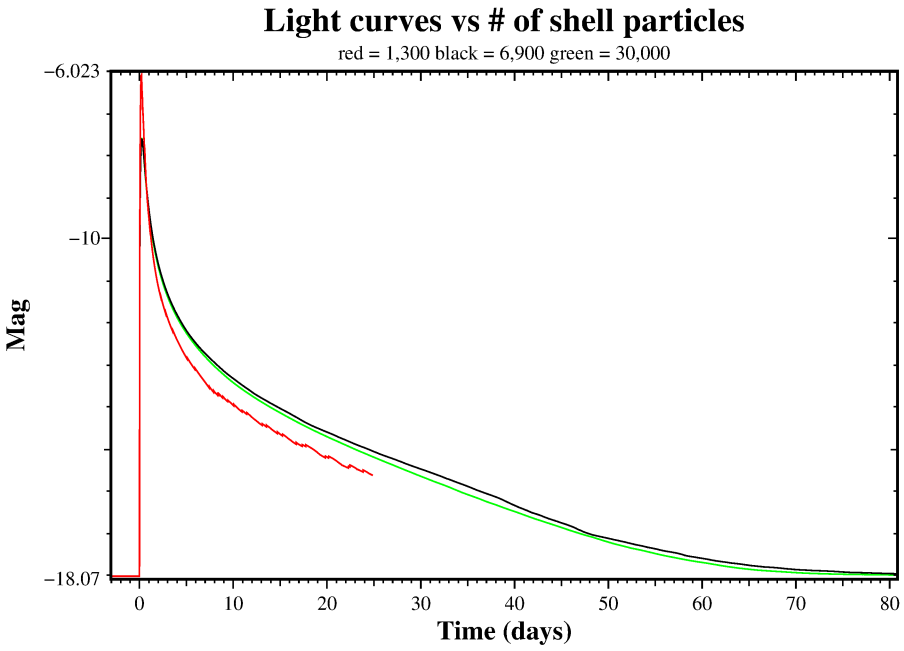


Figure 32: Light curve with different WD shell particle counts. Anything less than 3,000 gives bad results

The effects of using different time steps (dt) where also studied. Figure 33 plots the total energy of all the particles divided by the initial energy ($E0$) versus time for three different time steps. The base case $dt = 1s$ is plotted in blue, $dt = 0.5s$ in red and $dt = 2.0s$ in green. In all three cases, the ratio of $E/E0$ remains close to one on average indicating that energy is conserved.

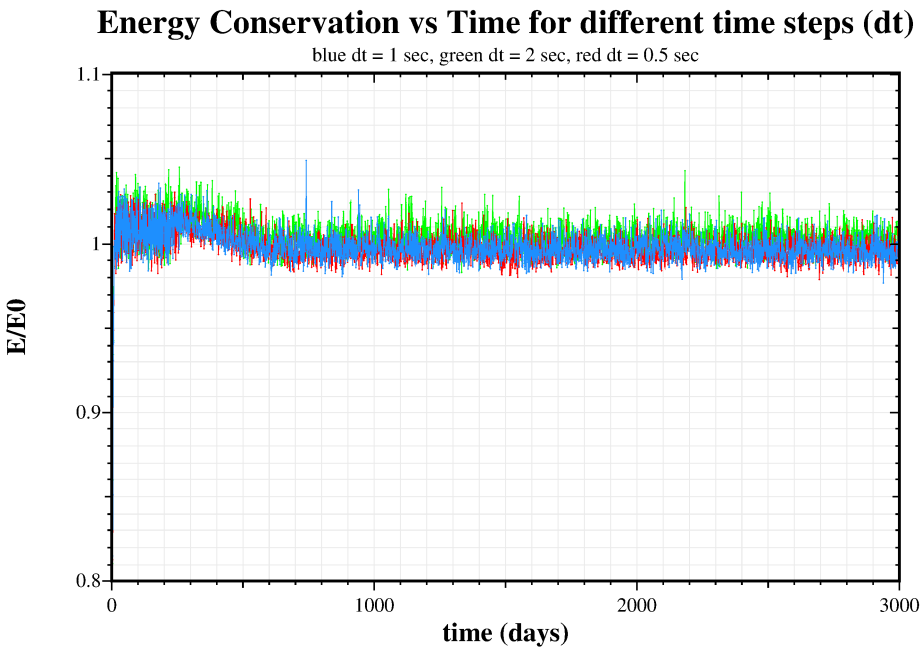


Figure 33: Conservation of energy vs time for three different time steps

The effects of different dt values on the light curve is plotted in figure 34. The base case $dt = 1s$ is plotted in black, $dt = 0.5s$ in red, $dt = 0.25s$ in blue, and $dt = 2.0s$ in green. The light curve for $dt = 2s$ (green) is very different from the other three. The other three have different peaks but have mostly the same shape and fall-off.

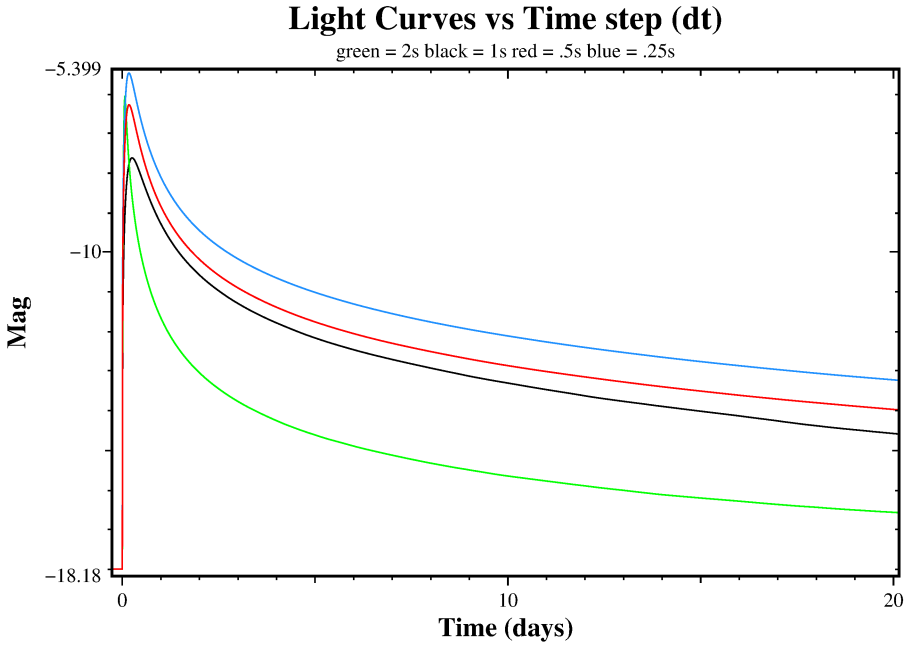


Figure 34: Effects of the time step dt on the computed light curve

The effects of the Langevin K_{eff} and friction coefficient on the accretion rate are plotted in figures 35 and 36. The accretion rate increases with increasing K_{eff} up to about $K_{eff} = 2$ after which it remains nearly constant. The nominal K_{eff} scaling factor is 1. This plot also shows that the accretion rate versus K_{eff} is nearly the same for both 30,000 (black) and 60,000 (red) shell particles. Figure 36 shows that the accretion rate decreases as the friction coefficient is increased (since the increased friction makes it harder for the particles to leave the star's shell). The nominal value is 0.001. The optimal values for K_{eff} and the friction coefficient were determined by varying them to give the desired star radius (see figure 30) and reasonable particle counts leaving the star's shell.

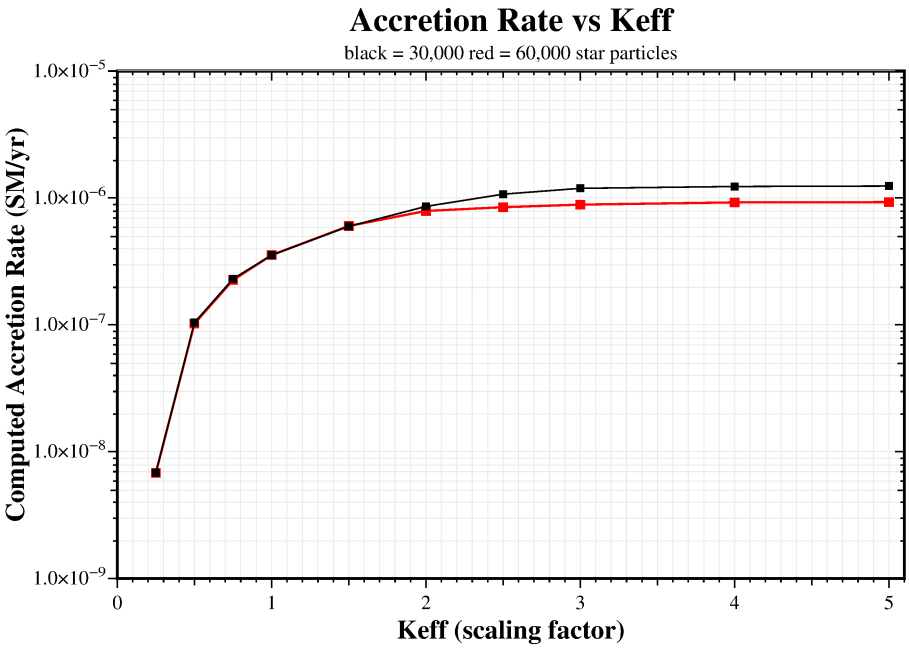


Figure 35: Accretion rate versus K_{eff} scaling for 30K (black) and 60K (red) shell particles

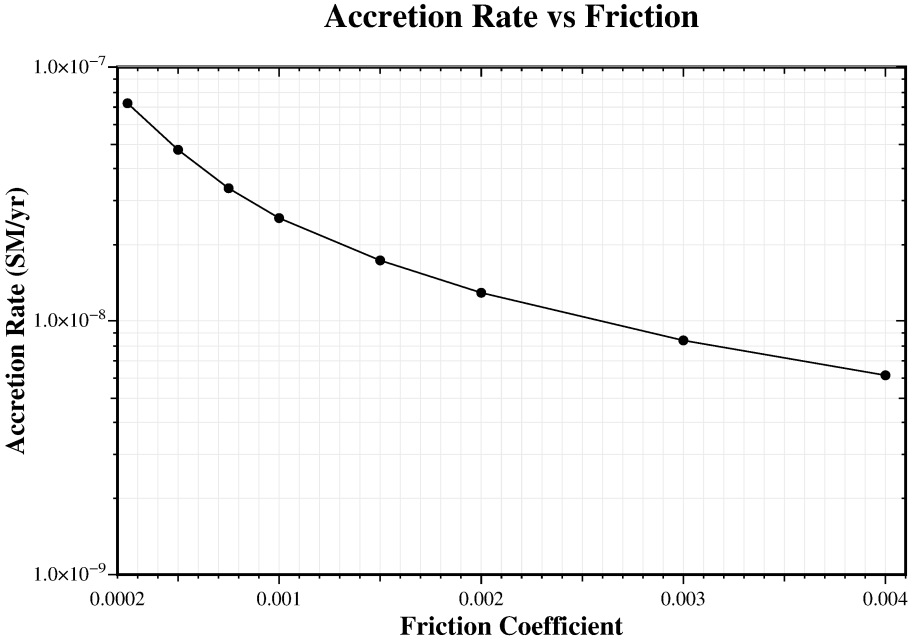


Figure 36: Accretion rate versus friction coefficient

B Summary of Uploaded Code

MPI and OpenCL Nova simulation code (used for all simulation results):

File Name	Description
main/giant.cpp	Host code.
main/inputs.cfg	Configuration file, parameter input deck
main/functions.cpp main/functions.h	Contains misc. subroutines
main/Timer.cpp main/Timer.h	AMD Timer library
main/kernels/core_local.cl	OpenCL Kernel to compute all forces

Post processing program:

File Name	Description
process/process/process.c	Main program
process/lib/pdefaults	Program configuration file
process/lib/2d_header.dat process/lib/3d_header.dat	2D PlotMTV header 3D PlotMTV header
process/lib/main.c process/lib/main.h	Contains misc. functions, directory tree structures, memory allocation and printing routines.
process/lib/ParaView.c process/lib/ParaView.h	Functions to export .vtk, .xyz, and .csv files
process/lib/Plotmtv.c process/lib/Plotmtv.h	Functions to export colored .mtv files and 2D slices

MLS Particle Based CPU Code:

File Name	Description
cpu_mls/giant.cpp	Host CPU MLS code.
cpu_mls/Makefile	Makefile used to compile
cpu_mls/MLSnn.cpp cpu_mls/MLSnn.h	Nearest neighbor version of MLS methods
cpu_mls/functionsnn.cpp cpu_mls/functionsnn.h	Contains misc. subroutines
cpu_mls/matrix.cpp cpu_mls/matrix.h	Matrix functions, matrix-matrix multiply
cpu_mls/neighbor.cpp cpu_mls/neihgbor.h	Various versions of a nearest neighbor search

MLS Particle Based GPU Code:

File Name	Description
gpu_mls/giant.cpp	Host GPU MLS code.
gpu_mls/Makefile	Makefile used to compile
gpu_mls/inputs.cfg	Configuration file for parameters
gpu_mls/MLSnn.cpp gpu_mls/MLSnn.h	Nearest neighbor version of MLS methods
gpu_mls/functionsnn.cpp gpu_mls/functionsnn.h	Contains misc. subroutines
gpu_mls/matrix.cpp gpu_mls/matrix.h	Matrix functions, matrix-matrix multiply
gpu_mls/neighbor.cpp gpu_mls/neighbor.h	Various versions of a nearest neighbor search
gpu_mls/Timer.cpp gpu_mls/Timer.h	AMD Timer library
gpu_mls/kernels/Amatrix.cl	OpenCL Kernel - Constructs MLS A matrix
gpu_mls/kernels/DAmatrix.cl	OpenCL Kernel - Constructs derivatives of the MLS A matrix
gpu_mls/kernels/MLS.cl	OpenCL Kernel - Computes MLS equations and forces

References

[1] M.F. Bode. "The Outbursts of Classical and Recurrent Novas". In: *Astron. Nachr.* 331 (2010), pp. 160–168.

[2] K. Moore and L. Bildsten. "Circumstellar Shell Formation in Symbiotic Recurrent Novae". In: *The Astrophysical Journal* 761 (2012), pp. 182–188.

[3] K. Shen and L. Bildsten. "The Effect of Composition on Nova Ignitions". In: *The Astrophysical Journal* 692 (2009), pp. 324–334.

[4] P. H. Hunenberger. "Thermostat Algorithms for Molecular Dynamics Simulations". In: *Adv. Polym. Sci.* 173 (2005), pp. 105–149.

[5] W. B. Langdon. *A Fast High Quality Pseudo Random Number Generator for nVidia CUDA*.

[6] H. Wendland. "Piecewise polynomial, positive definite and compactly supported radial basis functions of minimal degree". In: *Advances in Computational Mathematics* 4 (1995), pp. 389–396.

[7] W. H. Press et al. *Numerical Recipes: The Art of Scientific Computing*. Vol. Third Edition. Cambridge University Press, 2007.

[8] R. J. Strope, B. E. Schaefer, and A. A. Henden. "Catalog of 93 Nova Light Curves: Classification and Properties". In: *The Astronomical Journal* 140 (2010), pp. 34–62.

[9] *OpenCL - The open standard for parallel programming of heterogeneous systems*. URL: `www.khronos.org/opencl/`.

[10] AMD. *CodeXL*. 2014. URL: `developer.amd.com/tools-and-sdks/heterogeneous-computing/codexl/`.

[11] *Web pages for MPI Routines*. URL: `www.mpich.org/static/docs/v3.1/www3/`.

[12] The Khronos Group. *OpenCL Reference Pages*. 2010. URL: `www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/`.

[13] Chris Lomont. *Fast Inverse Square Root*. URL: `www.lomont.org/Software/..%5CMath/Papers/2003/InvSqrt.pdf`.

[14] J. J. Monaghan. *A refined particle method for astrophysical problems*. 1985, pp. 135–143.

[15] G. A. Dilts. "Moving-Least-Squares-Particle Hydrodynamics-I. Consistency and Stability". In: *Int. J. Num. Meth. Eng.* 44 (1999), pp. 1115–1155.

[16] Andrew Nealen. *An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation*. TU Darmstadt.

[17] T.D. Thoroughgood et al. "The Mass of the White Dwarf in the Recurrent Nova U Scorpii". In: *Monthly Notices of the Royal Astronomical Society* 327 (2001), pp. 1323–1334.

[18] J.J. Drake and S. Orlando. "The Early Blast Wave of the 2010 Explosion of U Scorpii". In: *The Astrophysical Journal Letters* 720 (2010), pp. L195–L200.

[19] American Association of Variable Star Observers. *Light Curve data for 2010 U Scorpii*. 2010.

[20] I. Hachisu and M. Kato. "A Universal Decline Law of Classical Novae". In: *The Astrophysical Journal* 167 (2006), p. 59.

[21] D. Prialink and A. Kovetz. "An Extended Grid of Multicycle Nova Evolution Models". In: *The Astrophysical Journal* 445 (1995), p. 789.

[22] Helena Uthas, Christian Knigge, and Danny Steeghs. *The orbital period and system parameters of the recurrent nova T Pyx*. 2010, pp. 237–246.

[23] Bradley E. Schaefer, Ashley Pagnotta, and Michael M. Shara. *The Nova Shell and Evolution of the Recurrent Nova T Pyxidis*. 2010, pp. 381–402.

[24] R. Walder, D. Folini, and S.N. Shore. "3D Simulations of RS Oph: From Accretion to Nova Blast". In: *Astronomy and Astrophysics* 484 (2008), pp. L9–L12.

[25] Martin Burtscher and Keshav Pingali. *An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm*. 2011.