Discretized models of Radially Pulsating Stars

Gábor Kovács [1] [2]

[1]MTA CSFK, Konkoly Observatory

[2]Eötvös Loránd University,
Institute of Geography and Earth Sciences,
Department of Astronomy

22 June 2018

Outline

General description of stars

. . . so simple a thing as a star. – A. S. Eddington

We can describe it as (first approximation):

- A spherically symmetric object
- Composed by ionized gas
- In hydro-static and thermodynamic equilibrium
- Inside its core thermonuclear processes takes place
- No rotating, changing, moving etc.
- No magnetic field.

So we only need five equation for description.

Pulsating stars

But there are stars which change their luminosity, the variable stars. Many processes can modify the amount of light coming from a star: eclipsing, gravitational-lensing, pulsation, etc.
Two types of pulsation:

- Radial pulsation: the outer layers of the star oscillating only radially.
- Non-radial oscillation: e. g. 5 min oscillation of the sun.

Now we only discuss the radial case.

Process of pulsation

There are known physical processes that are able to maintain the pulsation (otherwise it would be damped). One of these processes is the κ process, which related to the partially ionized hydrogen and helium layers of the stars.

In nutshell: The opacity changes with the ionization rate of these layers, thus they absorb more energy, which is released later by the pulsation.

Stars with these properties: RR Lyraes, Cepheids, W Virs, BL Hers, RV Taus.

Importance of pulsating stars

Common property:

Period-Luminosity Relation

If we measure the apparent magnitude of the star, and know the period of pulsation, we can determine the absolute magnitude and thus the distance of the star.
This is why we need to understand these "standard candles".

Modelling the pulsation

When we model the pulsation, we radially split the star into layers, and compute these processes:

- The equation of motion: the pressure at the to border of the layer defines the movement of that layer
- Radiation: this is the main inducing effect of the pulsation
- Equation of state: The pressure of the gas depends on its temperature
- Convection: this dissipates a part of the energy

This is a very complex problem, but to demonstrate it we can simplify to a chain of springs by neglecting the difficult physical processes above, and by replacing the layers by given masses and the pressure by linear springs.

General Solution

We have a classic finite linear chain:

$$m_j \frac{d^2 u_j}{dt^2} = (u_{j+1} - u_j) D_{i+j} - (u_j - u_{j-1}) D_j$$

Let's assume that all m and D are the same. And let $\omega^2 = \frac{D}{m}$.
Now we get:

$$\frac{d^2 u_j}{dt^2} = \omega^2 u_{j-1} - 2\omega^2 u_j + \omega^2 u_{j+1}$$

Now substitute: $u_j(t) = A_j \exp(i\omega' t + \varphi)$ Thus:

$$-A_j \omega'^2 = \omega^2 A_{j-1} - 2\omega^2 A_j + \omega^2 A_{j+1}$$

General Solution

The previous equation:

$$-A_j\omega'^2 = \omega^2 A_{j-1} - 2\omega^2 A_j + \omega^2 A_{j+1}$$

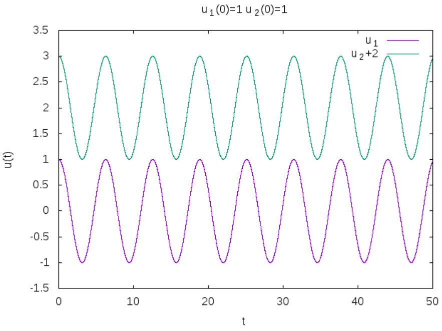
Can be written in the form of:

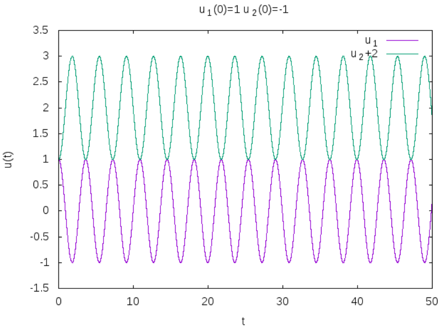
$$\begin{pmatrix} \ddots & \ddots & \ddots & \ddots & \ddots & & & & & \\ & 0 & \omega^2 & -2\omega^2 & \omega^2 & 0 & & & & \\ & & 0 & \omega^2 & -2\omega^2 & \omega^2 & 0 & & & \\ & & & 0 & \omega^2 & -2\omega^2 & \omega^2 & 0 & & \\ & & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} \vdots \\ A_{j-1} \\ A_j \\ A_{j+1} \\ \vdots \end{pmatrix} = \omega'^2 \mathbf{A}$$

Which is an eigen-value problem.
The general solution is: $u_j(t) = \sum_{k=0}^{N} A_k \exp(i\omega'_k t)$

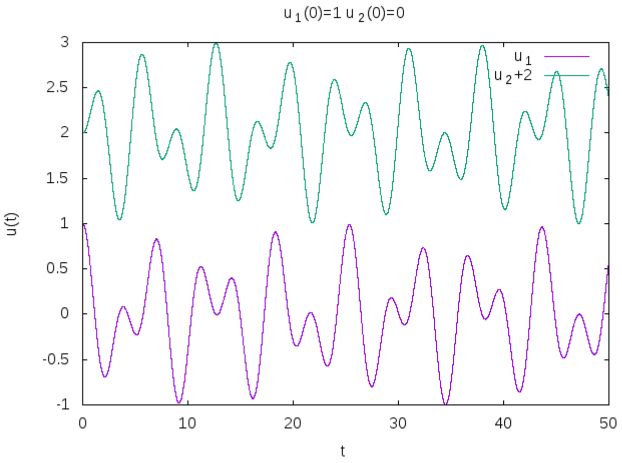
Eigenvalues and the modes of movement

These eigenvalues describe independent movements that the masses can do. The full motion of a given mass is the sum of these. They are called modes. Always there are as many modes as degrees of freedom. E. g. for two masses:





Eigenvalues and the Modes of Movement



Solution with parallel processing

We can solve the original equation directly:

$$m_j \frac{d^2 u_j}{dt^2} = (u_{j+1} - u_j) D_{i+j} - (u_j - u_{j-1}) D_j$$

In each step we only need the position of the two neighbouring masses.
Hence this computation can be easily made parallel.

We can compute each individual motion on different threads.
In the case of real stellar modelling we have thousands of layers, thus parallel computing can make a significant decrease in executing time.

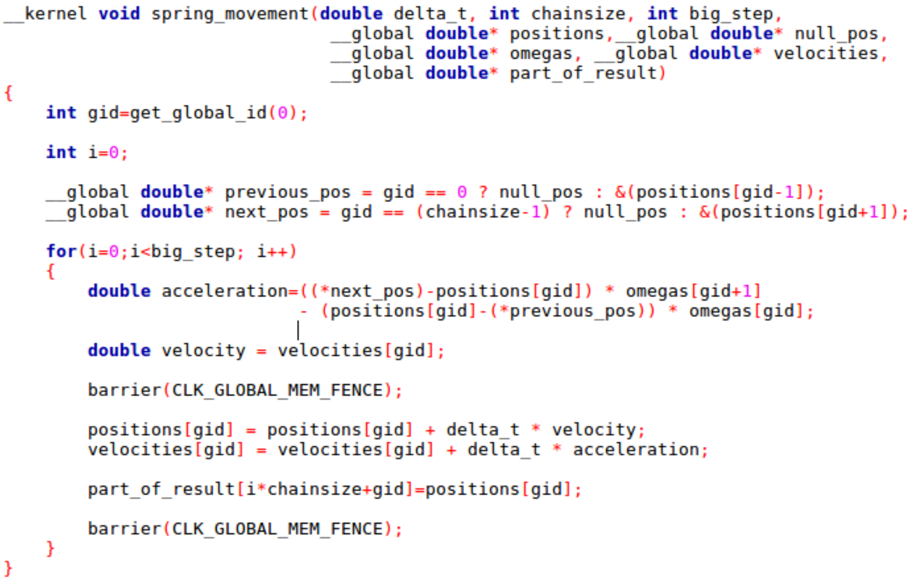
Example Solution

The program can be coded onto an OpenCL API.

The client side kernel should contain the equation of motion, and the integrator (It is necessary to synchronize the work-items between the integrator and the motion step, otherwise we can end up calculating an acceleration and using positions that are not from the same time step).

The output could be a larger buffer, so we don't need to copy the output back to the host side at each time step, saving time.

Example Solution



```
__kernel void spring_movement(double delta_t, int chainsize, int big_step,
                              __global double* positions,__global double* null_pos,
                              __global double* omegas, __global double* velocities,
                              __global double* part_of_result)
{
    int gid=get_global_id(0);

    int i=0;

    __global double* previous_pos = gid == 0 ? null_pos : &(positions[gid-1]);
    __global double* next_pos = gid == (chainsize-1) ? null_pos : &(positions[gid+1]);

    for(i=0;i<big_step; i++)
    {
        double acceleration=((*next_pos)-positions[gid]) * omegas[gid+1]
                           - (positions[gid]-(*previous_pos)) * omegas[gid];

        double velocity = velocities[gid];

        barrier(CLK_GLOBAL_MEM_FENCE);

        positions[gid] = positions[gid] + delta_t * velocity;
        velocities[gid] = velocities[gid] + delta_t * acceleration;

        part_of_result[i*chainsize+gid]=positions[gid];

        barrier(CLK_GLOBAL_MEM_FENCE);
    }
}
```

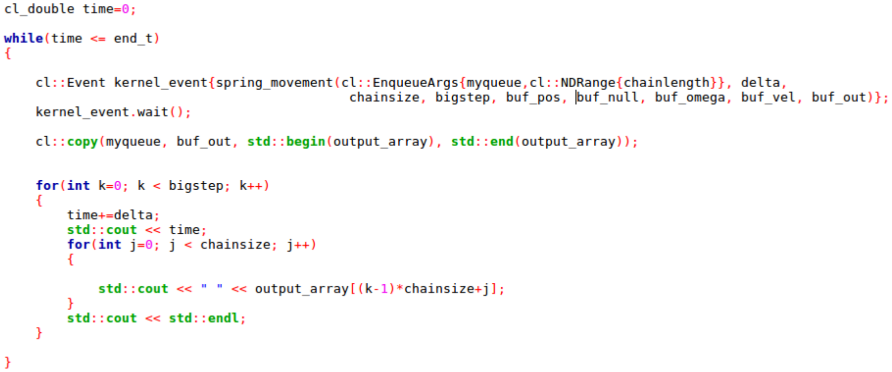
Example Solution

```cpp
auto spring_movement=cl::KernelFunctor<cl_double, cl_int, cl_int,cl::Buffer,cl::Buffer,
                                     cl::Buffer,cl::Buffer,cl::Buffer>(program, "spring_movement");

cl::Buffer buf_pos{context, std::begin(starting_positions), std::end(starting_positions), false};
cl::Buffer buf_null{context, std::begin(null_pos), std::end(null_pos), true };
cl::Buffer buf_omega{context, std::begin(omegas), std::end(omegas), true};
cl::Buffer buf_out{context, std::begin(output_array), std::end(output_array), false};
cl::Buffer buf_vel{context, std::begin(velocities), std::end(velocities), false};

cl::copy(myqueue, std::cbegin(starting_positions), std::cend(starting_positions), buf_pos);
cl::copy(myqueue, std::cbegin(velocities), std::cend(velocities), buf_vel);
cl::copy(myqueue, std::cbegin(null_pos), std::cend(null_pos), buf_null);
cl::copy(myqueue, std::cbegin(omegas), std::cend(omegas), buf_omega);
cl::copy(myqueue, std::cbegin(output_array), std::cend(output_array), buf_out);
```

Example Solution



```
cl_double time=0;

while(time <= end_t)
{

    cl::Event kernel_event{spring_movement(cl::EnqueueArgs{myqueue,cl::NDRange{chainlength}}, delta,
                                        chainsize, bigstep, buf_pos, buf_null, buf_omega, buf_vel, buf_out)};
    kernel_event.wait();

    cl::copy(myqueue, buf_out, std::begin(output_array), std::end(output_array));


    for(int k=0; k < bigstep; k++)
    {
        time+=delta;
        std::cout << time;
        for(int j=0; j < chainsize; j++)
        {

            std::cout << " " << output_array[(k-1)*chainsize+j];
        }
        std::cout << std::endl;
    }

}
```

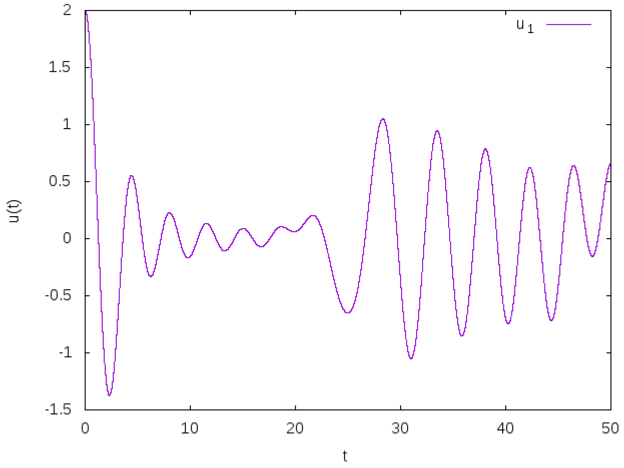
Example Solution

The following example was calculated for a system of ten springs and ten masses (The last mass had only one spring, this case is closer to the case of pulsating stars.). We used 500 000 time steps. (From 0 to 50, with 0.0001 steps).

The execution took approximately half minute without any optimization. The serial calculation took more than one minute.

Although this is not the proper utilization of the device, and we neglected almost every optimization, we turns out that using the GPU is much faster.

Chain of ten springs with open end



Summary

- The pulsation of stars can be modelled by a chain of springs as a first approximation.
- In the linear case, we can compute an exact solution, but the original problem is non-linear, thus we have to solve the differential equations
- This can be done in parallel way.

- Outlook
  - The state-of-the-art pulsation codes are still not parallelized.
  - Another problem is that most of them is written in Fortran language, thus improving them is difficult.