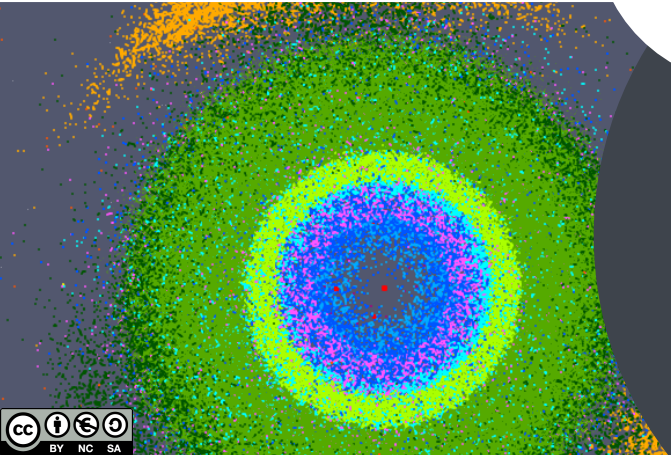
University of Stuttgart
Germany

IPVS

Alexander Van Craen,
Marcel Breyer,
Prof. Dr. Dirk Pflüger



Lab Course: Solar System Simulation

Who are we?

Alexander Van Craen

Alexander.Van-Craen@ipvs.uni-stuttgart.de

Marcel Breyer

Marcel.Breyer@ipvs.uni-stuttgart.de

Prof. Dr. Dirk Pflüger

Dirk.Pflueger@ipvs.uni-stuttgart.de

Learning Goals

- Implementation of a non-trivial project in a small group as well as an individual extension of it.
- Get a feeling for intra- and inter-node parallelization strategies and frameworks.
- Usage of established frameworks in the High-Performance Computing (HPC) community like OpenMP and MPI.
- Practical application of tree data structures.
- Working with C++ as one of the most widely used programming languages in HPC.
- Practical experiences with Git (using our GitLab instance; for more information about Git see Git HowTo) and Continuous Integration (CI) testing.
- Getting familiar with Linux, remote development, compute clusters, and cluster resource management software like SLURM.
- Usage of established programs in scientific computing (e.g., ParaView for visualization).

Content of the Lab Course

- Lab Course divided into multiple phases:

Group Formation	Independent formation of groups of up to 2. (Deadline: **02.11.2023**)
Phase 1	Get familiar with the n-body simulation and implement it using the tree-based Barnes-Hut algorithm on a multi-node system in the previously formed group. (Deadline: **04.12.2023**)
Phase 2	Extending your previous code with an **individual** project. (Deadline: **29.01.2024**)
Final Presentation	Present your results of phase 2 in a 15 min presentation. (Date: **TBA**)

Content of the Lab Course

- Lab Course divided into multiple phases:

Group Formation	Independent formation of groups of up to 2. (Deadline: **02.11.2023**)
Phase 1	Get familiar with the n-body simulation and implement it using the tree-based Barnes-Hut algorithm on a multi-node system in the previously formed group. (Deadline: **04.12.2023**)
Phase 2	Extending your previous code with an **individual** project. (Deadline: **29.01.2024**)
Final Presentation	Present your results of phase 2 in a 15 min presentation. (Date: **TBA**)

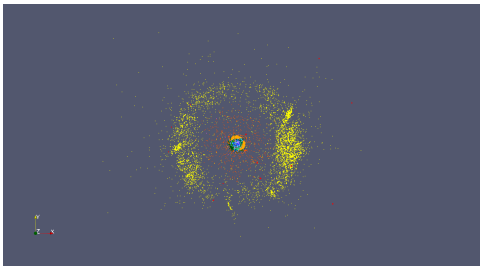
- Approval of phase 1 not later than the deadline.
  → Reaching phase 2 only after the successful acceptance of phase 1.
  → If you finish phase 1 early and let us know, you will have more time for phase 2!
- In addition, there will be two small competitions at the end of phase 1 (more on that later).

Group Formation

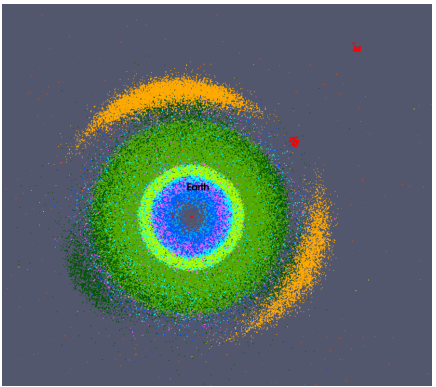
- Independent formation of groups of up to 2 (e.g., using the ILIAS forum).
- The first submission in ILIAS should consist of a simple text file containing all group members and a group name used for the competition and your repository in our GitLab.
- Deadline: **02.11.2023**
- If you did not find a group until 02.11.2023, contact us via email and we will assign you to a group.
- If you did not create an ILIAS submission and did not contact us via email, you do not count as a participant in this lab course!

Phase 1: Goal

Parallel, distributed n-body simulation using the tree-based Barnes-Hut algorithm of the planets, dwarf planets, moons, and asteroids in our solar system.



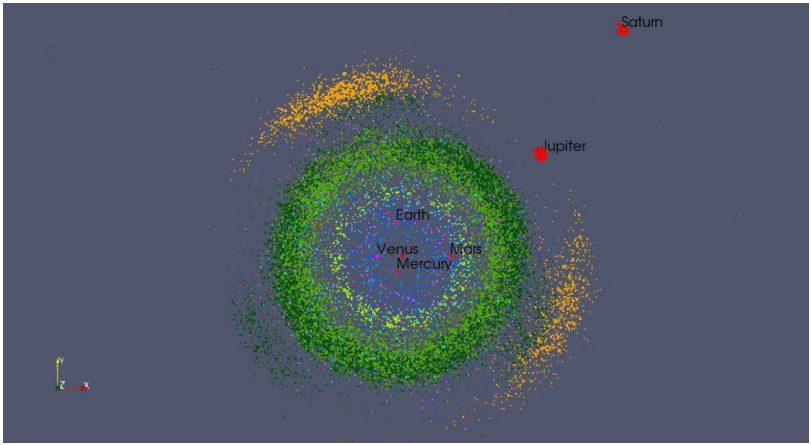
Our solar system, including outer dwarf planets and TransNeptunian Objects.



Our solar system zoomed in to Jupiter and the main asteroid belt, Earth highlighted.

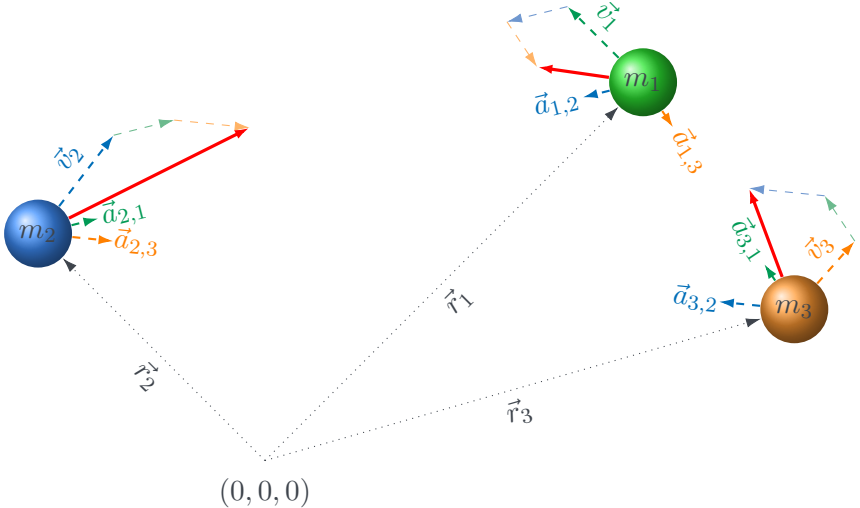
Phase 1: Goal

Simulation of our solar system: planets, dwarf planets, moons, and asteroids (larger than $10\,\mathrm{km}$).



Theory: Basic Idea of N-Body Simulations

The goal of an n-body simulation is to establish equations of motion for each body.



Theory: Naive Formulation of an N-Body Simulation Problem

Each body i with its mass m_i and position vector $\vec{r}_i$ experiences the force $\vec{a}_i$ from all other bodies according to Newton's law of universal gravitation:

$$\vec{a}_i = \sum_{i \neq j} G m_j \frac{\vec{r}_j - \vec{r}_i}{(\|\vec{r}_j - \vec{r}_i\|_2^2 + \epsilon^2)^{\frac{3}{2}}}$$

Theory: Naive Formulation of an N-Body Simulation Problem

Each body i with its mass m_i and position vector $\vec{r}_i$ experiences the force $\vec{a}_i$ from all other bodies according to Newton's law of universal gravitation:

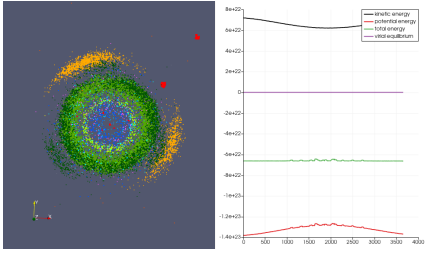
$$\vec{a}_i = \sum_{i \neq j} G m_j \frac{\vec{r}_j - \vec{r}_i}{(\|\vec{r}_j - \vec{r}_i\|_2^2 + \epsilon^2)^{\frac{3}{2}}}$$

- gravitational constant: $G = 6.674\,30 \times 10^{-11} \frac{\mathrm{m}^3}{\mathrm{kg} \cdot \mathrm{s}^2}$
- The physical units of G do not reflect the units used in the data sets.
- softening factor: $\epsilon = 1 \times 10^{-11}$ to prevent collisions between two bodies
- $\|\|_2^2$: squared Euclidean distance

Phase 1: Requirements - Scenario 1

Simulating our solar system with the provided planets and moons as well as all asteroids with a **given** `diameter` and `albedo` and where the **main-belt asteroids** (`MBA`) are further constraint with a diameter **greater or equal** than $10\,\text{km}$ must not take longer than $1\,\text{h}$ on our target system `sgscl1` using 8 compute nodes:

```
srun --exclusive -N 8 ./simulate --file scenario1.csv --dt 1h --t_end 12y --vs 2d --vs_dir sim_s1 --theta 1.05
```

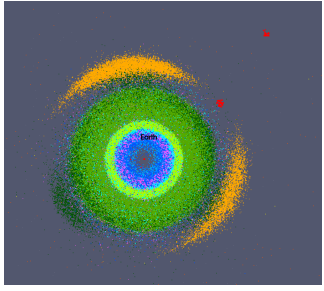


Simulation result of the inner solar system with different colors for the different orbit classes after $105\,192$ time steps with $19\,054$ bodies.

Phase 1: Requirements - Scenario 2

This time you have 30 min and again 8 compute nodes. Try to simulate as many planets, moons, and asteroids as possible for one Earth year!

```
srun --exclusive -N 8 ./simulate --file scenario2.csv --dt 1h --t_end 1y --vs 7d --vs_dir sim_s2 --theta 1.05
```



Example initial state with $1\,216\,869$ bodies (approximately the maximum number of asteroids in NASA JPL's Small-Body Database).

Phase 1: Simulation Data

- The data sets containing the planets and moons as well as the asteroids for scenario 1 can be found in ILIAS:
- For scenario 2 additional asteroid data can be queried using NASA Jet Propulsion Laboratory's (JPL) Small-Body Database.
- Be aware that some bodies may appear multiple times (e.g., Pluto and its moons) and you have to take care that these bodies are only used once in the actual simulation!
- **Note:** all data is given using the Keplerian orbital elements that must first be converted to Cartesian state vectors.
- **Attention:** The Sun/Sol (with the reference mass $1.988\,47 \times 10^{30}$ kg and epoch 2451544.5 JD) must always be **manually** added at the coordinate's origin since it is impossible to specify its orbital elements with respect to itself.
- All data must be saved and processed using double precision (FP64, `double`) floating point types.

Phase 1: Submission - 04.12.2023

- Submission via ILIAS.
- Scenario 1: animation of the 3D simulation with a reasonable temporal resolution.
- Scenario 2: your custom data set file; the name should contain the number of used bodies.
- Scenario 1 & 2: image of the last time step created via ParaView including an energy plot (kinetic, potential, and total energies) and `scenario1/2_save.csv` files with the end states (name, mass, positions, velocities) of the simulations.
- The ParaView specific `.pvd` and `.vtp` files must **not** be submitted!
- Diagrams with explanations regarding the runtimes (see next slide).
- One file, `submission.sh`, containing the following information (`${}` replaced):

```
#!/bin/sh
git clone ${REPOSITORY_URL} ${GROUP_NAME}
cd ${GROUP_NAME} || exit
git checkout ${COMMIT_HASH}
```

The commit must contain the code to be evaluated as well as a `README.md` file describing all necessary steps to build and run your code.

Phase 1: Performance Analysis

- Diagram of the runtimes of the code fixing the number of MPI nodes and OpenMP threads and using different numbers of bodies.
- Diagram of the runtimes of the code fixing the number of MPI nodes and varying the number of OpenMP threads (e.g., via `export OMP_NUM_THREADS=N`) for a fixed problem size.
- Diagram of the runtimes of the code fixing the number of OpenMP threads and varying the number of MPI nodes for a fixed problem size.
- Diagram displaying the runtime and the summed up distances to a very small θ using different values for θ.
- Additional: each diagram must contain a possible **explanation** for the displayed behavior!
- **Note**: use suitable scenarios for all diagrams.

Phase 1: Competitions

Two competitions:

- Which group produces the prettiest simulation animation for scenario 1?
  - How to best display the vastly different body sizes in ParaView?
  - Added body textures?
  - Camera movement?
- Which group can simulate the most bodies in the available time? (scenario 2)
  - Add your runtime and number of bodies to the Nextcloud table using the password `TODO`.
  - At the end of phase 1, the runtimes and number of bodies will be verified by us.
  - Real simulation without tricks (e.g., it is **not** allowed to simply output pre-calculated time steps!).

→ The winners of each competition will receive a small reward!

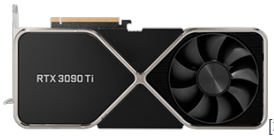
Phase 2: Goal

➔ Everyone can do a custom project on his own based on the code of phase 1.

Phase 2: Goal

HPX

STE||AR GROUP [1]



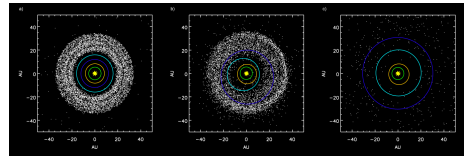
[2]

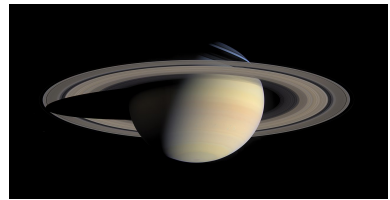
NVIDIA CUDA

SYCL

OpenCL

?


[3]


[4]

➔ Everyone can do a custom project on his own based on the code of phase 1.

Image Sources:

[1]: https://hpx-docs.stellar-group.org/latest/html/index.html
[2]: https://store.nvidia.com/de-de/geforce/store/?page=1&limit=9&locale=de-de
[3]: https://en.wikipedia.org/wiki/Nice_model
[4]: https://en.wikipedia.org/wiki/Saturn

Phase 2: Goal

Extend your code of phase 1 by an individual project, possible examples are:

- code acceleration using GPUs (e.g., via CUDA, OpenCL, Kokkos, or SYCL)
- replacing MPI with another framework for distributed computing like HPX
- add more physics (e.g., collision detection)
- implementing more efficient algorithms (e.g., Fast Multipole Methods)
- lifting the simplification that every node saves all bodies
  $\rightarrow$ each node only knows about a subset of all bodies (needs load balancing)
- new simulation scenarios (e.g., close-up simulation of Saturn including its rings, the Oort Cloud with Nemesis, the Nice model, or the "galaxy far, far away")
- ...

Restrictions: It must have something to do with our overall parallelism topic and it must be different from your group members' projects!

➔ Each project must be approved by us, i.e., simply send us an email with 2-3 sentences describing your idea before you start working on it.

Phase 2: Submission - 29.01.2024

- Submission via ILIAS.
- A `submission.sh` file as for phase 1 with adjusted placeholders and commit hash.
- A `.pdf` describing your project idea, implementation, and possible results on **at least** 5 pages excluding the title page and possible references
- All supplementary files for your individual project (e.g., simulation data sets for a new scenario or new runtime diagrams).
- The LaTeX template can be found in ILIAS.

Phase 2: Presentation - TBA

- 15-20 min presentation + 5 min Q&A per student.
- The slides must be uploaded in ILIAS.
- Phase 2 will be the crucial part for your grade where this presentation is a major part of! The results from phase 1 play only a minor role in grading!

- The slides should contain all your results of phase 2.
- They should also contain the problems you encountered during phase 2.

General Submission Guidelines

- For each phase **exactly** one `*.zip` or `*.tar.gz` archive must be uploaded to ILIAS with the content corresponding to the respective phase.
- PhasePhasenumber followed by the family name (of all group members), e.g., `Phase1_Name1_Name2_Name3.tar.gz`.
- Each file in the archive must contain the name (of **all** group members).
- The submission must be compilable and executable on a normal Linux command line on one of our machines, i.e., no IDE specific build scripts are allowed!

GitLab

- After applying for our hardware and receiving an account, this account can also be used to login to our GitLab server.
- For phase 1, you get one private repository per submission group, for phase 2, one fork of the phase 1 repository per student.
- Each repository **must** have one `README` file containing all necessary information on how to compile and execute your code.
- An example `C++ & CMake` repository with small MPI + OpenMP code snippets can also be found in our GitLab.

CI via GitLab

- You should at least have 10 useful test cases and a test coverage of at least $80\,\%$!
- Install a GitLab runner on your machine and register a runner for your repository/fork.
  - Install instructions for a GitLab runner.
  - GitLab runner as docker service as shown in the SSE lecture.
  - The runner should support the docker executor and should be able to run Linux images.
- Add a GitLab pipeline status badge to the project that is based on your pipeline and the main branch.

If you cannot add such a runner on your machine, please email one of us with the registration token and your GitLab username. We will then add a runner to your repository.

Allowed Software

- You are allowed to use any third-party library that does **not** solve a significant part of your project (like converting the orbital elements to state vectors, performing the actual simulation, writing the ParaView output, etc.).
- If you are not sure if you are allowed to use a library, feel free to contact us via email or ILIAS.
- Utility libraries, however, are allowed and we recommend the usage of the following:
  - any command line parser library (e.g., `cxxopts`)
  - the C++ formatting library `{fmt}`
- **Note:** you **must** provide installation instructions for every library you use! Or better use something like CMake's `FetchContent_Declare` to install it if possible automatically.
- Another way to install third-party libraries without the need for root privileges is `spack`.
- Any C++ standard from C++17 and upwards is allowed.

General Recommendations and Remarks

- Your simulation should include some sort of progress indication (e.g., a terminal output each $10\,\%$ of the simulation).
- Use a `C++` IDE (e.g., Visual Studio (Code) or CLion) and their tools like profiler, debugger, auto-formatting, linter, etc.
- Use an automated build system, preferable CMake.
- Document your code (for you **and** for us!), e.g., via Doxygen.
- The installation and usage of your program as well as its verification should be as easy as possible.
- Test your program on our target Linux system `sgscl1`.
- One of the main goals of this lab course is to encourage working independently, however, if questions arise, feel free to ask them in the ILIAS forum.
- The easiest way to determine the total runtime of your code is to use the Linux `time` utility:
  ```
  time ./simulate --file data.csv --dt 1h --t_end 1y --vs 2d --theta 1.05
  ```
- Start to work on phase 1 and phase 2 soon enough!
- Secret tip: MIT's "The Missing Semester of Your CS Education"

On a Final Note

The simulation is greatly simplified!

On a Final Note

The simulation is greatly simplified!

Many physical effects or properties are ignored or only roughly estimated:

- rotational forces, tidal forces, radiation, magnetic fields, etc.
- various value approximations, mainly for the asteroids' masses
- shape of the body (assumed to be spherical)
- ...

On a Final Note

The simulation is greatly simplified!

Many physical effects or properties are ignored or only roughly estimated:

- rotational forces, tidal forces, radiation, magnetic fields, etc.
- various value approximations, mainly for the asteroids' masses
- shape of the body (assumed to be spherical)
- ...

➔ Nevertheless, it is good enough for a stable simulation of the whole of our solar system including the sun, planets, dwarf planets, moons, and major asteroids!

Important Deadlines

25.10.2023	Kick-off
02.11.2023	Deadline Group Formation
04.12.2023	Deadline Phase 1
29.01.2024	Deadline Phase 2
TBA	Final Presentations

→ **Do not** forget to register for our lab course in C@MPUS!

Bonus Material

- Target System
- SLURM
- Theory
- Simulation Data
- Value Approximation
- Keplerian Orbital Elements
- Parallelization
- Visualization via ParaView
- Simulation
- Important Deadlines

Bonus Material: Target System

Target System: `sgscl1` - Hardware

`sgscl1` is a small cluster consisting of one login node and 14 compute nodes.
Attention: do not perform **any** calculations on the login node!

Available hardware per compute node:

- 1 Intel(R) Xeon(R) CPU E3-1585 v5 @ 3.50GHz
  - 4 physical cores
  - 8 hyper threads
- 31 GiB DDR4 RAM
- 400 GB `/scratch` space (**not shared** across nodes of the cluster)
  → If you want to use the `/scratch` space instead of your `$HOME` directory, you must create the `/scratch/FaPra/${GROUP_NAME}` custom directory and delete it afterwards!
- inter-node connection via 10 GB Ethernet

Target System: `sgscl1` - Connecting to the Cluster - 1

First login to our server:

1. `ssh ${USERNAME}@ipvslogin.informatik.uni-stuttgart.de`
2. You will be prompted to input your password.
3. Change your initial password: See `passwd` and follow the displayed instructions. This **must** be done during your first login.
4. Disconnect from the server by either typing `exit` or using `Strg + d`.

Target System: sgscl1 - Connecting to the Cluster - 1

First login to our server:

1. `ssh ${USERNAME}@ipvslogin.informatik.uni-stuttgart.de`
2. You will be prompted to input your password.
3. Change your initial password: See `passwd` and follow the displayed instructions. This **must** be done during your first login.
4. Disconnect from the server by either typing `exit` or using `Strg + d`.

Connecting to the sgscl1 cluster's login node using an ssh key:

1. Generate a new ssh key pair (if none already exists).
2. Copy your public ssh key to our login server via:
   `ssh-copy-id -i ~/.ssh/id_rsa ${USERNAME}@ipvslogin.informatik.uni-stuttgart.de`
   Note: you will be prompted to enter your passwords.

Target System: sgscl1 - Connecting to the Cluster - 2

3. Add the following lines to your `~/.ssh/config` file (on your local machine):

```
host ipvslogin
    ForwardAgent yes
    hostname ipvslogin.informatik.uni-stuttgart.de
    user ${USERNAME}
    IdentityFile ~/.ssh/id_rsa

host sgscl1
    ForwardAgent yes
    ProxyJump ipvslogin
    hostname
    user ${USERNAME}
    IdentityFile ~/.ssh/id_rsa
```

4. Now you can login to sgscl1 by simply using: `ssh sgscl1`

Bonus Material: SLURM

- Target System
  - Connecting to sgscl1
- **SLURM**
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- Simulation Data
- Value Approximation
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- Parallelization
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- Simulation
  - Validation
- Important Deadlines

SLURM Workload Manager

The most important SLURM commands are:

`sinfo`
: list available nodes including their operational state

`squeue`
: list running or pending jobs

`scancel JOB_ID`
: cancel to job identified by `JOB_ID` and remove it from the queue

`srun --pty bash`
: start an interactive session on one node

`srun -N X ./simulate [OPTIONS]`
: run your simulation on X compute nodes in parallel

`sbatch -N X -n X --wrap "./simulate [OPTIONS]"`
: queue your simulation on X compute nodes in parallel for future execution

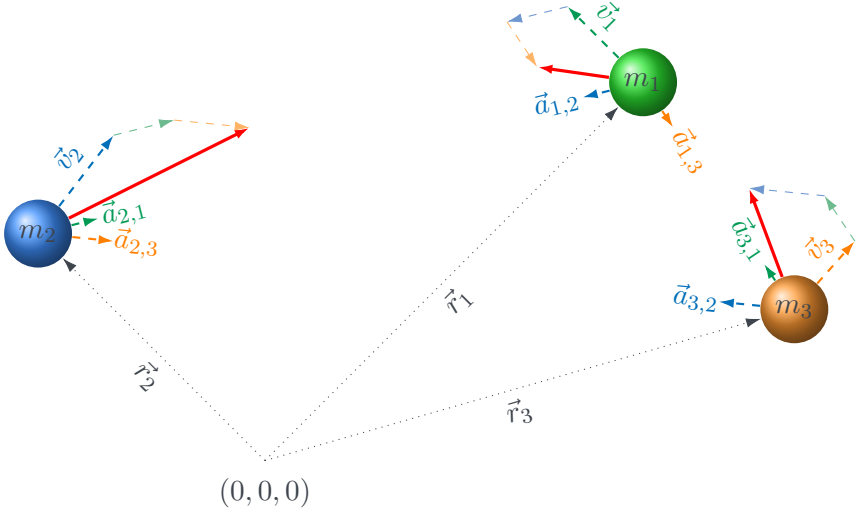
For more information, e.g., using `sbatch` with a script instead of `--wrap`, see the official SLURM homepage.

Bonus Material: Theory

- Target System
  - Connecting to sgscl1
- SLURM
- **Theory**
  - **Barnes-Hut**
  - **Time Stepping Method**
  - **Complete Simulation**
- Simulation Data
- Value Approximation
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- Parallelization
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- Simulation
  - Validation
- Important Deadlines

Theory: Basic Idea of N-Body Simulations

The goal of an n-body simulation is to establish equations of motion for each body.



Theory: Naive Formulation of an N-Body Simulation Problem

Each body i with its mass m_i and position vector $\vec{r}_i$ experiences the force $\vec{a}_i$ from all other bodies according to Newton's law of universal gravitation:

$$\vec{a}_i = \sum_{i \neq j} G m_j \frac{\vec{r}_j - \vec{r}_i}{(\|\vec{r}_j - \vec{r}_i\|_2^2 + \epsilon^2)^{\frac{3}{2}}}$$

Theory: Naive Formulation of an N-Body Simulation Problem

Each body i with its mass m_i and position vector $\vec{r}_i$ experiences the force $\vec{a}_i$ from all other bodies according to Newton's law of universal gravitation:

$$\vec{a}_i = \sum_{i \neq j} G m_j \frac{\vec{r}_j - \vec{r}_i}{(\|\vec{r}_j - \vec{r}_i\|_2^2 + \epsilon^2)^{\frac{3}{2}}}$$

- gravitational constant: $G = 6.674\,30 \times 10^{-11} \frac{\mathrm{m}^3}{\mathrm{kg \cdot s}^2}$
- The physical units of G do not reflect the units used in the data sets:
  - distance in astronomical units: $1\,\mathrm{AU} = 1.495\,978\,706\,91 \times 10^{11}\,\mathrm{m}$
  - time in days: $1\,\mathrm{d} = 86\,400\,\mathrm{s}$
- **Note:** the correctly scaled G should be calculate in your program!
- softening factor: $\epsilon = 1 \times 10^{-11}$ to prevent collisions between two bodies
- $\|\|_2^2$: squared Euclidean distance

Theory: Energy of the System

- kinetic energy: $E_K = \sum_i \frac{1}{2} m_i \|\vec{v}_i\|_2^2$
- potential energy: $E_P = -\sum_{i<j} \frac{G m_i m_j}{\|\vec{r}_j - \vec{r}_i\|_2}$
- sum of the energies in the system: $E_T = E_K + E_P$
- E_T must be "constant" in a stable simulation (conservation of energy)!

Theory: Energy of the System

- kinetic energy: $E_K = \sum_i \frac{1}{2} m_i \|\vec{v}_i\|_2^2$
- potential energy: $E_P = -\sum_{i<j} \frac{G m_i m_j}{\|\vec{r}_j - \vec{r}_i\|_2}$
- sum of the energies in the system: $E_T = E_K + E_P$
- E_T must be "constant" in a stable simulation (conservation of energy)!

The Virial Equilibrium:

$$\frac{2 \cdot E_K}{|E_P|} \approx 1.0$$

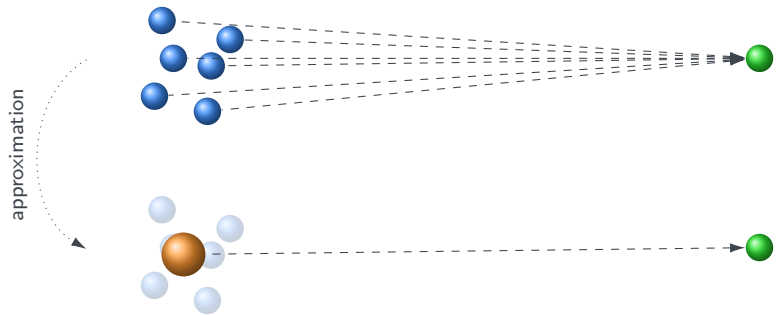
as a dynamic equilibrium state on a timescale comparable to a few times the typical time needed for a body to cross the system.

Question: What happens if the result is > 1.0 or < 1.0?

See: `http://www.scholarpedia.org/article/N-body_simulations_(gravitational)`

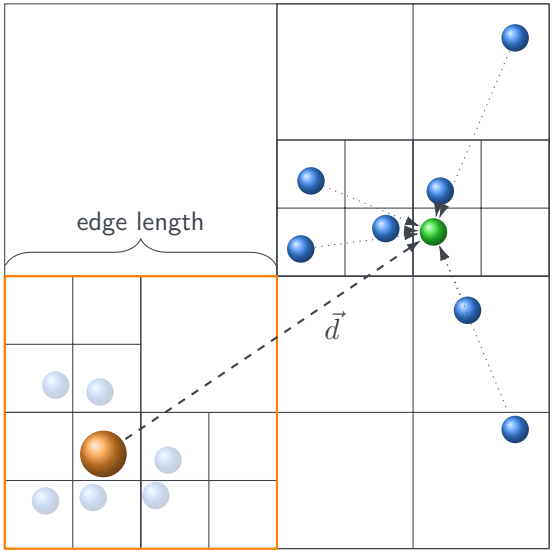
Theory: Barnes-Hut Algorithm to Speed Up the Simulation

Combine bodies that are far enough away from pseudo-bodies to reduce the number of necessary force calculations.



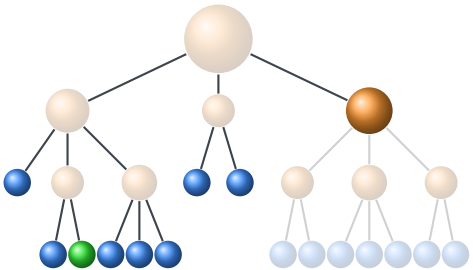
See: `Josh Barnes and Piet Hut: A hierarchical O(N log N) force-calculation algorithm (1986) (DOI: https://doi.org/10.1038/324446a0)`

Theory: Barnes-Hut - Example: Quadtree



Use pseudo-bodies to calculate the forces if:

$$\frac{\text{edge length}}{\left\|\vec{d}\right\|_2} < \theta$$



naive: 14 force calculations

Barnes-Hut: 8 force calculations

Theory: Barnes-Hut - Tree Node

Each node of the octree (extension of the quadtree for the 3-dimensional space) has to save the following information:

- edge length of the octree-octant
- sum of the masses of all bodies in the octant: $m_{oct} = \sum_i m_i$
- center of mass of all bodies i in the octant: $\vec{c}_{oct} = \frac{\sum_i m_i \vec{r}_i}{\sum_i m_i}$
- all children of the octant

Each leaf may only contain a single body!

Theory: Barnes-Hut - Update the Accelerations $\vec{a}_i$

```
1:  procedure UpdateAcceleration(body, node)
2:      calculate direction vector d⃗ between body and the center of mass of node
3:      if pseudo-body can be used ∨ node is leaf then
4:          update acceleration of a⃗_body
5:      else
6:          for each child of node do
7:              UpdateAcceleration(body, child)
8:          end for
9:      end if
10: end procedure
```

Question:

Is it also possible to speed up the force calculations using the Barnes-Hut tree?

Theory: Leapfrog Integration

The leapfrog integration, a 2nd order time step method, as a combination of the two variants of the symplectic Euler method (SE), is used to advance from the time step k to the next time step $k+1$:

$$(SE1)\begin{cases} \vec{v}^{k+\frac{1}{2}} &= \vec{v}^{k} + \vec{a}^{k}\frac{\Delta t}{2} \\ \vec{r}^{k+\frac{1}{2}} &= \vec{r}^{k} + \vec{v}^{k+\frac{1}{2}}\frac{\Delta t}{2} \end{cases}$$

$$(SE2)\begin{cases} \vec{r}^{k+1} &= \vec{r}^{k+\frac{1}{2}} + \vec{v}^{k+\frac{1}{2}}\frac{\Delta t}{2} \\ \vec{v}^{k+1} &= \vec{v}^{k+\frac{1}{2}} + \vec{a}^{k+1}\frac{\Delta t}{2} \end{cases}$$

See: `O. Buneman: Time-Reversible Difference Procedures (1967) (DOI: https://doi.org/10.1016/0021-9991(67)90056-3)`

Theory: A Complete Simulation

1: adjust initial velocities: $\vec{u}_i = \frac{\sum_j m_j \vec{v}_j}{\sum_j m_j}$; $\vec{v}_i = \vec{v}_i - \vec{u}_i$
2: update accelerations $\vec{a}_i$
3: visualize initial state
4: **while** simulation not terminated yet **do**
5: perform leapfrog integration step
6: **if** time step should be visualized **then**
7: visualize time step
8: **end if**
9: update time step
10: **end while**
11: Save final state of the simulation as `.csv` file (name + mass + positions + velocities)

Bonus Material: Simulation Data

- Target System
  - Connecting to sgscl1
- SLURM
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- **Simulation Data**
- Value Approximation
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- Parallelization
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- Simulation
  - Validation
- Important Deadlines

The Simulation Data

Two data sets given in ILIAS:

- Planets, dwarf planets, and (named) moons (feel free to add the data of the remaining moons if you can find them and let us know!) in our solar system.
  - → This data can be used for development, debugging, and verification.
- The asteroid data used in scenario 1.

The Simulation Data

Two data sets given in ILIAS:

- Planets, dwarf planets, and (named) moons (feel free to add the data of the remaining moons if you can find them and let us know!) in our solar system.
  - → This data can be used for development, debugging, and verification.
- The asteroid data used in scenario 1.

More asteroid data for scenario 2 can be downloaded using NASA Jet Propulsion Laboratory's (JPL) Small-Body Database:

- select desired orbit class(es)
- add orbit constraints: e.g., `albedo` and `diameter` must be defined (Physical Parameter Fields)
- select necessary Fields (for the name use the `IAU name` field)

This allows you to generate nearly arbitrarily large data sets using real-world data: $\approx 140\,000$ asteroids with defined `albedo` and `diameter` and over one million asteroids in total.

The Simulation Data - Format

You need to be able to parse the following `.csv` file:

```
e,a,i,om,w,ma,epoch,H,albedo,diameter,class,name,mass,central_body
0.2056302515978038,0.3870982252718477,7.005014362233553,48.33053877672862,29.12427943500334,
↪  172.7497133441682,2451544.5,,,,PLA,Mercury,3.301011e+23,Sun
.2227328427416296,1.4581505451557,10.82795835269297,304.2910556026917,178.9325148860407,
↪  358.8212586092838,2459800.5,10.31,0.25,16.84,AMO,Eros,,
```

`e,a,i,om,w,ma,epoch` : as part of the Keplerian orbital elements.

`H,albedo,diameter,mass` : to estimate the mass of a body if not given.

`class` : the orbital class of the body.

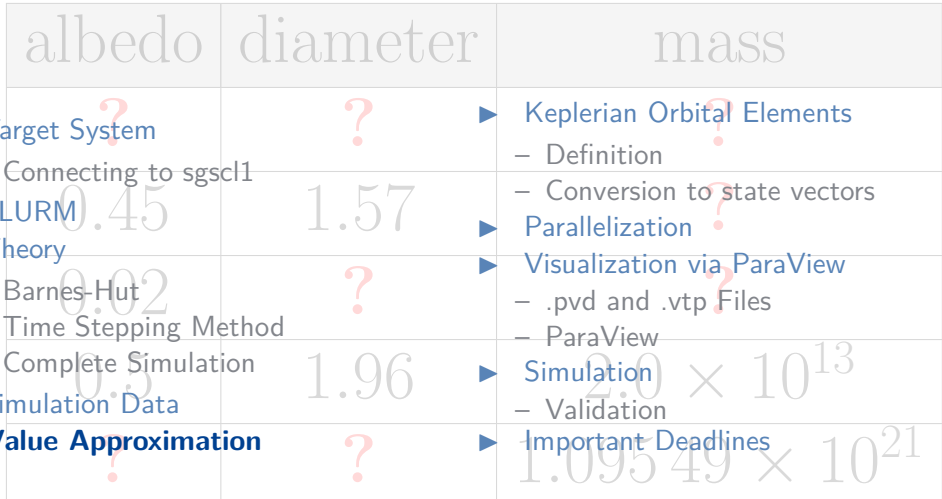
`name` : the IAU name of the body.

`central_body` : the name of the central body.

The Simulation Data - General Remarks

- Independent of the used JPL asteroid data set, the planets and moon data set must **always** be included in your simulation!
- Be aware that the angles are given in degrees and not in radians.
- Be aware that some bodies may appear multiple times (e.g., Pluto and its moons) and you have to ensure that these bodies are only used once in the actual simulation!
- All data must be saved and processed using double precision (FP64, `double`) floating point types.
- **Attention:** The Sun/Sol (with the reference mass $1.988\,47 \times 10^{30}$ kg and epoch 2451544.5 JD) must always be **manually** added at the coordinate's origin since it is impossible to specify its orbital elements with respect to itself.

Bonus Material: Value Approximation



- Target System
  - Connecting to sgscl1
- SLURM
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- Simulation Data
- **Value Approximation**
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- Parallelization
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- Simulation
  - Validation
- Important Deadlines

Value Approximation

Some values or columns may not be present and, therefore, must be approximated:

- the `mass` column does not exist in the JPL data sets: it must be estimated using the asteroid's `diameter` and geometric `albedo`.
- If no `albedo` is given, approximate it using the asteroid's orbit class.
- If no `diameter` is given, approximate it using the asteroid's absolute magnitude `H` and geometric `albedo`.
- The `central_body` column does not exist in the JPL data sets: by definition, the `central_body` of all asteroids is the Sun/Sol.

Value Approximation - Geometric Albedo

Approximate the assumed geometric `albedo` using random number distributions based on the asteroid's orbit class:

- AMO (Amor): $[0.450, 0.550]$
- APO (Apollo): $[0.450, 0.550]$
- ATE (Aten): $[0.450, 0.550]$
- IEO (Interior Earth Obj.): $[0.450, 0.550]$
- MCA (Mars-crossing): $[0.450, 0.550]$
- IMB (Inner Main-belt): $[0.030, 0.103]$
- MBA (Main-belt): $[0.097, 0.203]$
- OMB (Outer Main-belt): $[0.197, 0.5]$
- CEN (Centaur): $[0.450, 0.750]$
- TJN (Jupiter Trojans): $[0.188, 0.124]$
- TNO (TransNeptunian Obj.): $[0.022, 0.130]$
- AST (Asteroid): $[0.450, 0.550]$
- PAA (Parabolic): $[0.450, 0.550]$
- HYA (Hyperbolic): $[0.450, 0.550]$

Note: These are just very rough estimates, if you find more accurate estimates feel free to use them and let us know!

See: `https://pdssbn.astro.umd.edu/data_other/objclass.shtml`

Value Approximation - Geometric Albedo

Approximate the assumed geometric `albedo` using random number distributions based on the asteroid's orbit class:

- AMO (Amor): $[0.450, 0.550]$
- APO (Apollo): $[0.450, 0.550]$
- ATE (Aten): $[0.450, 0.550]$
- IEO (Interior Earth Obj.): $[0.450, 0.550]$
- MCA (Mars-crossing): $[0.450, 0.550]$
- IMB (Inner Main-belt): $[0.030, 0.103]$
- MBA (Main-belt): $[0.097, 0.203]$
- OMB (Outer Main-belt): $[0.197, 0.5]$
- CEN (Centaur): $[0.450, 0.750]$
- TJN (Jupiter Trojans): $[0.188, 0.124]$
- TNO (TransNeptunian Obj.): $[0.022, 0.130]$
- AST (Asteroid): $[0.450, 0.550]$
- PAA (Parabolic): $[0.450, 0.550]$
- HYA (Hyperbolic): $[0.450, 0.550]$

Note: These are just very rough estimates, if you find more accurate estimates feel free to use them and let us know!

We also added four new orbit classes (for a better ParaView output):

- STA (Star): the Sun/Sol
- PLA (Planets): Earth, Mars, ...
- DWA (Dwarf Planets): Ceres, Pluto, Eris, ...
- SAT (Satellites (Moons)): Luna, Titan, ...

See: `https://pdssbn.astro.umd.edu/data_other/objclass.shtml`

Value Approximation - Diameter

Approximate the diameter d of an asteroid given its assumed geometric `albedo` and absolute magnitude H using:

$$d = 1329 \cdot \texttt{albedo}^{-0.5} \cdot 10^{-0.2H} \quad [\text{km}]$$

See: B. Carry: Density of asteroids (2012)
(DOI: https://doi.org/10.1016/j.pss.2012.03.009)

Value Approximation - Mass

Approximate the density ρ of an asteroid given its assumed geometric `albedo` based on the three main asteroid categories:

C-type (chondrite): most common, probably consist of clay and silicate rocks;
assumption: $\texttt{albedo} < 0.1 \quad \rightarrow \quad \rho = 1.38\,\mathrm{g/cm^3}$

S-type ("stony"): made up of silicate materials and nickel-iron;
assumption: $0.1 \leq \texttt{albedo} \leq 0.2 \quad \rightarrow \quad \rho = 2.71\,\mathrm{g/cm^3}$

M-type (nickel-iron): metallic;
assumption: $\texttt{albedo} > 0.2 \quad \rightarrow \quad \rho = 5.32\,\mathrm{g/cm^3}$

See: `https://solarsystem.nasa.gov/asteroids-comets-and-meteors/asteroids/in-depth/`

Value Approximation - Mass

Approximate the density ρ of an asteroid given its assumed geometric `albedo` based on the three main asteroid categories:

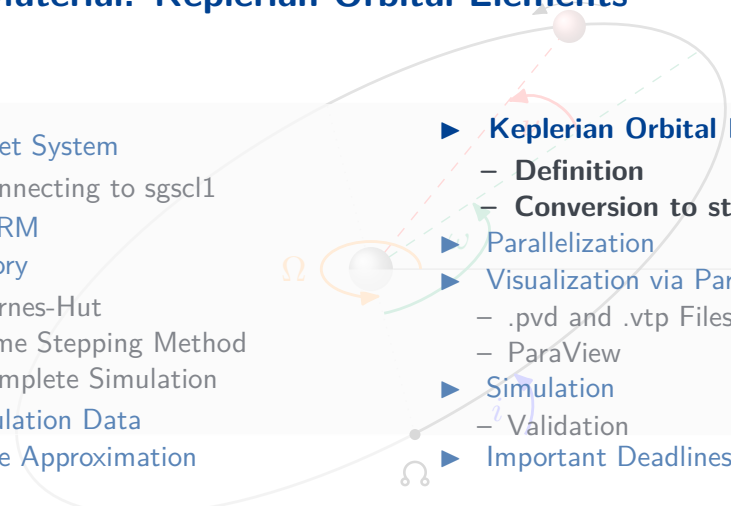
- C-type (chondrite): most common, probably consist of clay and silicate rocks;
  assumption: $\texttt{albedo} < 0.1 \quad \rightarrow \quad \rho = 1.38\,\mathrm{g/cm^3}$
- S-type ("stony"): made up of silicate materials and nickel-iron;
  assumption: $0.1 \leq \texttt{albedo} \leq 0.2 \quad \rightarrow \quad \rho = 2.71\,\mathrm{g/cm^3}$
- M-type (nickel-iron): metallic;
  assumption: $\texttt{albedo} > 0.2 \quad \rightarrow \quad \rho = 5.32\,\mathrm{g/cm^3}$

Approximate the mass of the asteroid using its presumed density ρ and its diameter d assuming a spherical shape via (**note:** pay attention to the physical units!):

$$m = \frac{4}{3}\pi \cdot \left(\frac{d}{2}\right)^3 \cdot \rho \qquad [\mathrm{kg}]$$

See: `https://solarsystem.nasa.gov/asteroids-comets-and-meteors/asteroids/in-depth/`

Bonus Material: Keplerian Orbital Elements



- Target System
  - Connecting to sgscl1
- SLURM
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- Simulation Data
- Value Approximation
- **Keplerian Orbital Elements**
  - **Definition**
  - **Conversion to state vectors**
- Parallelization
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- Simulation
  - Validation
- Important Deadlines

Phase 1.3: Orbital Elements to Orbital State Vectors

- For our n-body simulation, we need the two Cartesian vectors for each body: the position vector $\vec{r}$ and the velocity vector $\vec{v}$.
- They are also called *orbital state vectors*.

- In astronomy the Kepler orbit is used more often to describe the mechanics of a celestial body.
- They are also called Keplerian *orbital elements*.

Phase 1.3: Orbital Elements to Orbital State Vectors

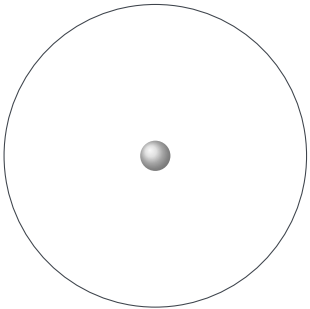
- For our n-body simulation, we need the two Cartesian vectors for each body: the position vector $\vec{r}$ and the velocity vector $\vec{v}$.
- They are also called *orbital state vectors*.

- In astronomy the Kepler orbit is used more often to describe the mechanics of a celestial body.
- They are also called Keplerian *orbital elements*.

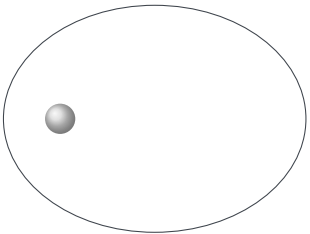
→ Since the data is given using the orbital elements, we must convert them to their respective orbital state vectors before we can use them in our simulation.

Orbital Elements - Visualization: Shape and Size of the Ellipsis

Orbital Elements - Visualization: Shape and Size of the Ellipsis



$e = 0$

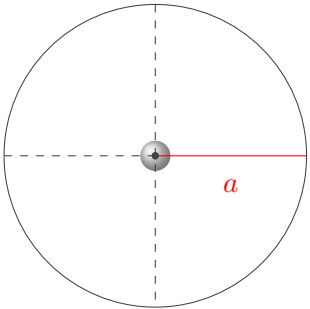


$e = 0.66$

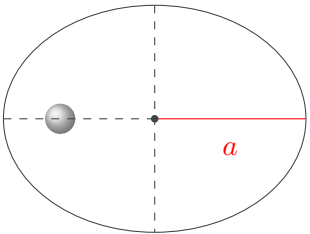
eccentricity e

shape of the ellipsis (elongation compared to a circle)

Orbital Elements - Visualization: Shape and Size of the Ellipsis



$e = 0$



$e = 0.66$

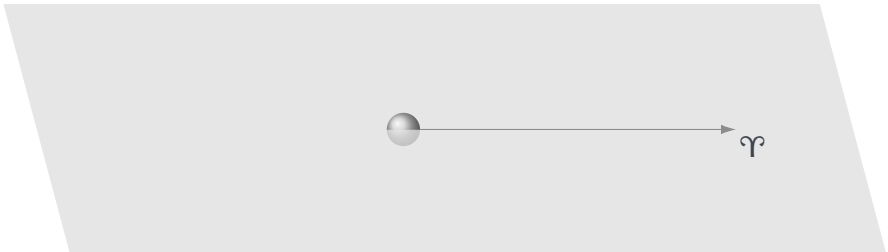
eccentricity e

shape of the ellipsis (elongation compared to a circle)

semi-major axis a [AU]

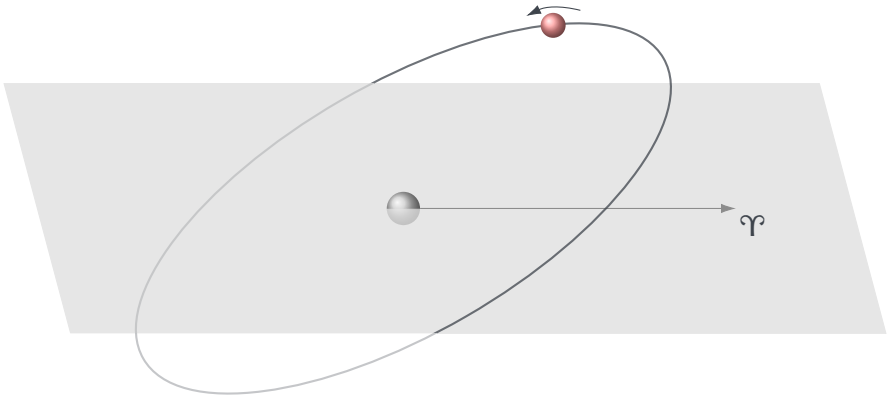
one-half of the largest diameter

Orbital Elements - Visualization

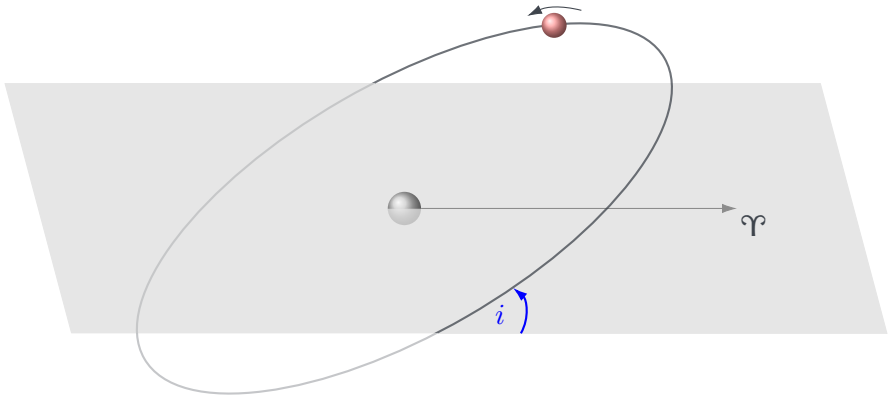


reference direction ♈:
for heliocentric orbits also known as “First Point of Aries”

Orbital Elements - Visualization



Orbital Elements - Visualization



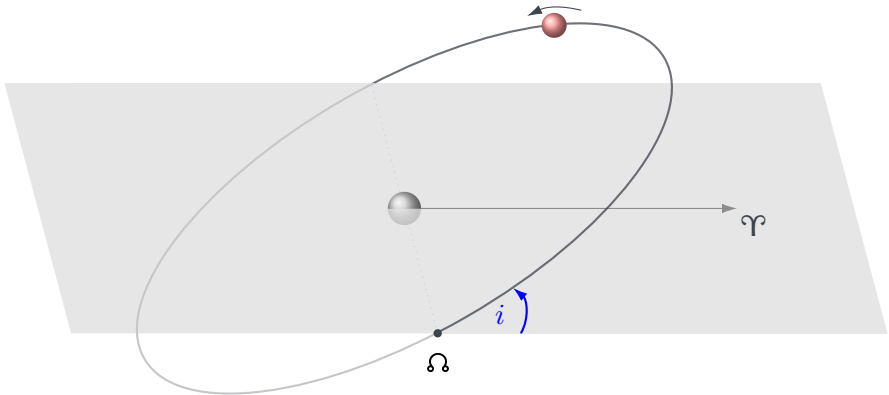
inclination i [rad]

vertical tilt of the ellipsis with respect to the reference plane, e.g.:

$0\,\text{rad} = \quad 0°$ orbital plane same as reference plane

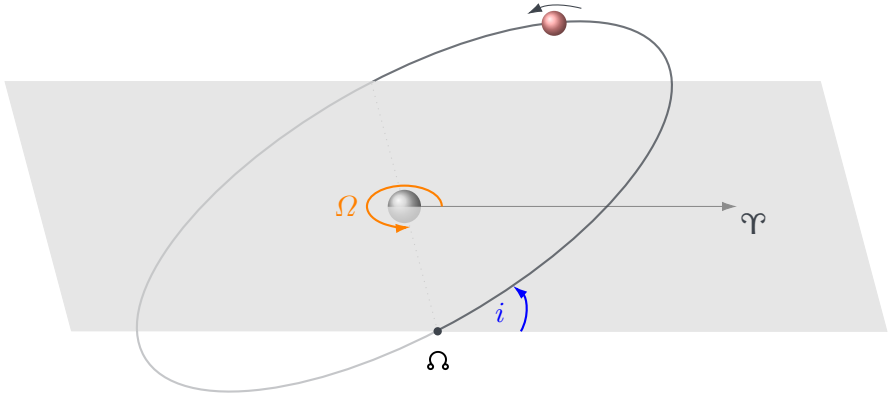
$1.5708\,\text{rad} = 90°$ orbital plane perpendicular to reference plane

Orbital Elements - Visualization



ascending node ☊:
point where the orbiting body moves **north** through the reference plane

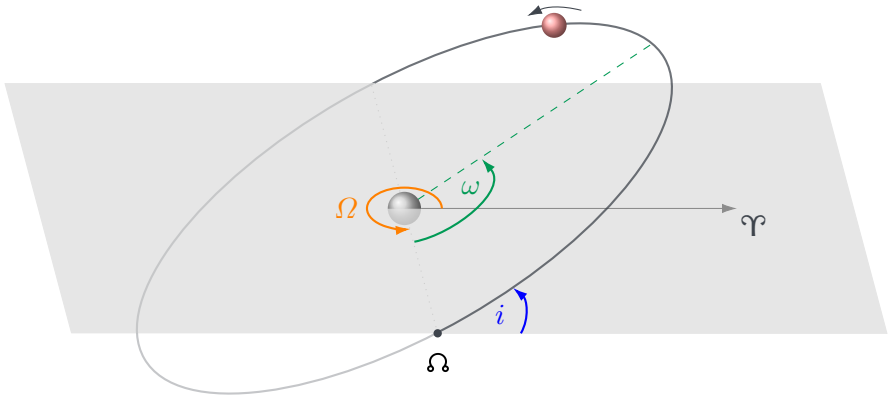
Orbital Elements - Visualization



longitude of the ascending node Ω [rad]

the angle from the reference direction to the direction of the ascending node; measured counterclockwise

Orbital Elements - Visualization

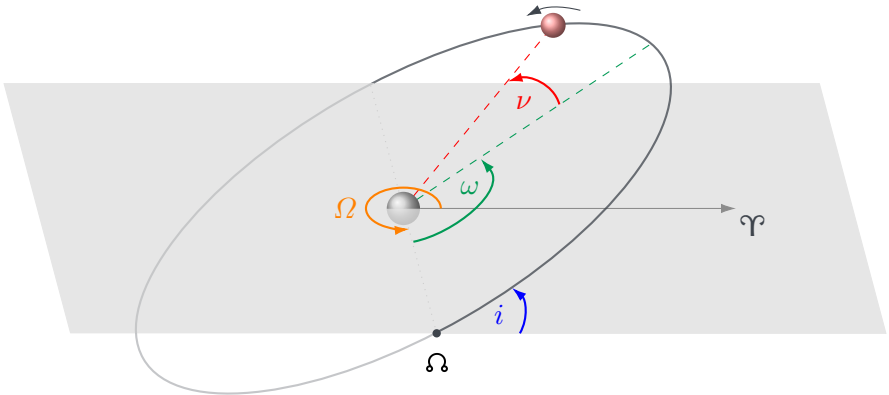


argument of periapsis ω [rad]

angle from the body's ascending node to its periapsis in the direction of motion, e.g.:

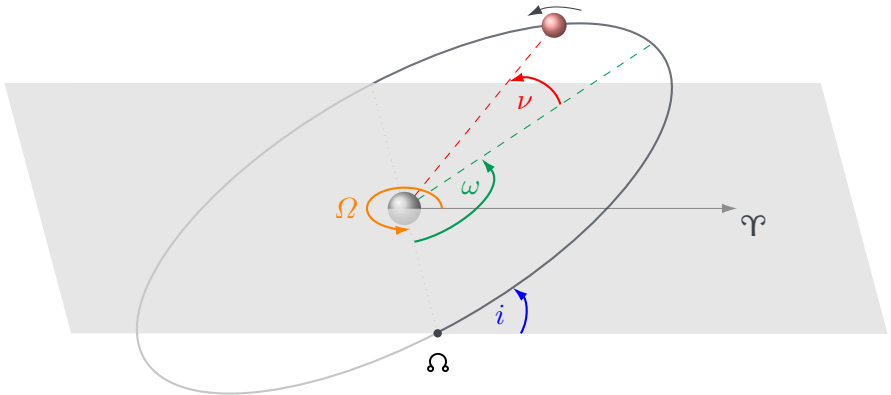
$0\,\text{rad}$ the orbiting body has its closest approach to the central body at the same moment it crosses the reference frame from south to north

Orbital Elements - Visualization



true anomaly ν [rad] at epoch t_0 [JD]:
angle between the direction of periapsis and the position of the orbiting body at epoch t_0

Orbital Elements - Visualization



mean anomaly M_0 [rad] at epoch t_0 [JD]

the angular distance from the pericenter which a fictitious body would have if it moved in a circular orbit, with constant speed, in the same orbital period as the actual body in its elliptical orbit

Orbital Elements - The Six Keplerian Elements (Overview)

eccentricity e
: shape of the ellipsis (elongation compared to a circle)

semi-major axis a [AU]
: one-half of the largest diameter

inclination i [rad]
: vertical tilt of the ellipsis with respect to the reference plane

longitude of the ascending node Ω [rad]
: the angle from the reference direction to the direction of the ascending node

argument of periapsis ω [rad]
: angle from the body's ascending node to its periapsis in the direction of motion

mean anomaly M_0 [rad] at epoch t_0 [JD]
: the angular distance from the pericenter which a fictitious body would have if it moved in a circular orbit, with constant speed, in the same orbital period as the actual body in its elliptical orbit

Orbital State Vectors - Cartesian Vectors

position $\vec{r}$ [AU]
: the position of the body in the reference frame, $\vec{r} \in \mathbb{R}^3$

velocity $\vec{v}$ $[\frac{\text{AU}}{\text{d}}]$
: the velocity of the body in the same reference frame, $\vec{v} \in \mathbb{R}^3$

Convert Orbital Elements to Orbital State Vectors - 1

1. Calculate $M(t)$:

$$M(t) = M_0 + (t - t_0)\sqrt{\frac{\mu}{a^3}}$$

where $\mu = GM$ is the standard gravitational parameter of the central body and t the Sun's/Sol's reference epoch at 2451544.5 JD (midnight on January 1, 2000)
(**note:** take care of the correct units!).

2. Solve Kepler's equation $M(t) = E(t) - e \sin E(t)$ for the eccentric anomaly $E(t)$ with a numerical method like Newton-Raphson:

$$E_0(t) = M(t)$$

$$E_{j+1}(t) = E_j(t) - \frac{E_j(t) - e \sin E_j(t) - M(t)}{1 - e \cos E_j(t)}$$

using a suitable stopping criterion (e.g., a fixed number of 30 iterations).

See: `https://downloads.rene-schwarz.com/download/M001-Keplerian_Orbit_Elements_to_Cartesian_State_Vectors.pdf`

Convert Orbital Elements to Orbital State Vectors - 2

3. Calculate the true anomaly $\nu(t)$:

$$\nu(t) = 2 \cdot \arctan 2 \left(\sqrt{1+e} \sin \frac{E(t)}{2}, \sqrt{1-e} \cos \frac{E(t)}{2} \right)$$

4. Calculate the distance to the central body:

$$r_c(t) = a(1 - e \cos E(t))$$

5. Calculate the position $\vec{o}(t)$ and velocity $\dot{\vec{o}}(t)$ vectors in the orbital frame:

$$\vec{o}(t) = r_c(t) \begin{pmatrix} \cos \nu(t) \\ \sin \nu(t) \\ 0 \end{pmatrix}, \quad \dot{\vec{o}}(t) = \frac{\sqrt{\mu a}}{r_c(t)} \begin{pmatrix} -\sin E(t) \\ \sqrt{1-e^2} \cos E(t) \\ 0 \end{pmatrix}$$

See: `https://downloads.rene-schwarz.com/download/M001-Keplerian_Orbit_Elements_to_Cartesian_State_Vectors.pdf`

Convert Orbital Elements to Orbital State Vectors - 3

6. Transform $\vec{o}(t)$ and $\dot{\vec{o}}(t)$ to the inertial frame in bodycentric rectangular coordinates $\vec{r}(t)$ and $\dot{\vec{r}}(t)$ using the matrix R:

$$R = \begin{pmatrix} \cos\omega\cos\Omega - \sin\omega\cos i\sin\Omega & -(\sin\omega\cos\Omega + \cos\omega\cos i\sin\Omega) & 0 \\ \cos\omega\sin\Omega + \sin\omega\cos i\cos\Omega & \cos\omega\cos i\cos\Omega - \sin\omega\sin\Omega & 0 \\ \sin\omega\sin i & \cos\omega\sin i & 0 \end{pmatrix}$$

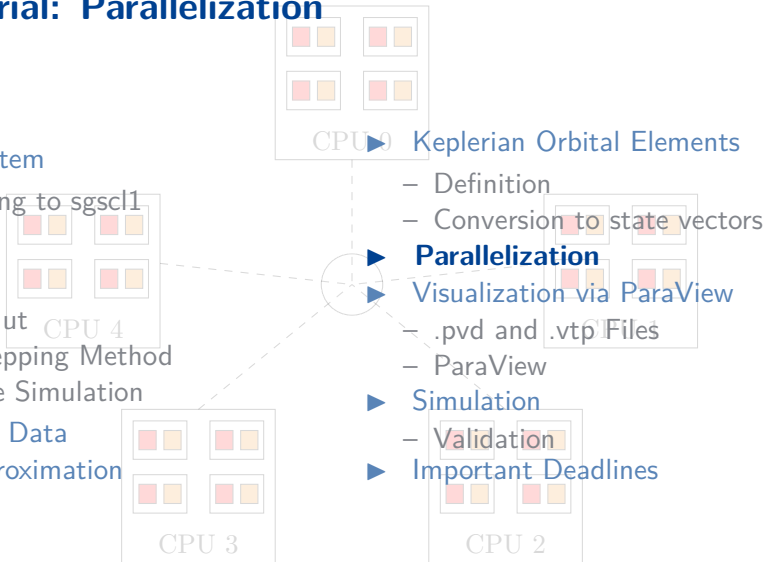
$$\vec{r}(t) = R \cdot \vec{o}(t), \quad \dot{\vec{r}}(t) = R \cdot \dot{\vec{o}}(t)$$

7. Update position and velocity to be relative to its central body cb:

$$\vec{r}(t) = \vec{r}(t) + \vec{r}_{cb}(t), \quad \dot{\vec{r}}(t) = \dot{\vec{r}}(t) + \dot{\vec{r}}_{cb}(t)$$

See: `https://downloads.rene-schwarz.com/download/M001-Keplerian_Orbit_Elements_to_Cartesian_State_Vectors.pdf`

Bonus Material: Parallelization



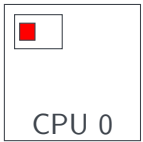
- Target System
  - Connecting to sgscl1
- SLURM
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- Simulation Data
- Value Approximation
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- **Parallelization**
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- Simulation
  - Validation
- Important Deadlines

A Complete Simulation

1: adjust initial velocities: $\vec{u}_i = \frac{\sum_j m_j \vec{v}_j}{\sum_j m_j}$; $\vec{v}_i = \vec{v}_i - \vec{u}_i$
2: update accelerations $\vec{a}_i$
3: visualize initial state
4: **while** simulation not terminated yet **do**
5: perform leapfrog integration step
6: **if** time step should be visualized **then**
7: visualize time step
8: **end if**
9: update time step
10: **end while**
11: Save final state of the simulation as `.csv` file (name + mass + positions + velocities)

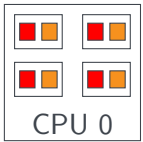
Question: What can and should be parallelized?

Parallelization



Normal:
1 core

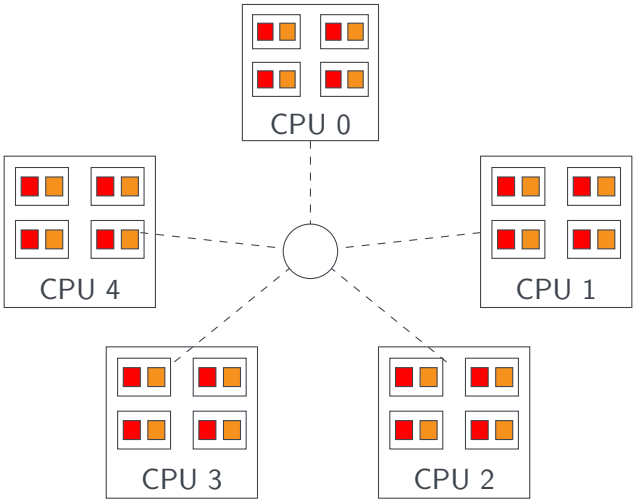
Parallelization



Normal:
1 core

OpenMP:
4 cores (8 HT)

Parallelization



Normal:
1 core

OpenMP:
4 cores (8 HT)

MPI + OpenMP:
20 cores (40 HT)
on 5 nodes

Parallelization Hints

- It is allowed and highly advised **to hold the data redundant on each node**.
- Tree refinements on one level and the ones below are independent of each other.
- **Hint**: start with a sequential implementation, but keep the parallelization in mind (especially when choosing the used data structures).
- All data structures that need to be communicated via MPI must be serialized in some way.
- Pointers can not easily point into another MPI rank's memory.
- Pointer addresses change if you copy a structure.
- A structure with relative offsets can be more easily merged together than a structure with absolute offsets.
- Can the Barnes-Hut tree construction also be parallelized?

Parallelization Hints

- It is allowed and highly advised **to hold the data redundant on each node**.
- Tree refinements on one level and the ones below are independent of each other.
- **Hint**: start with a sequential implementation, but keep the parallelization in mind (especially when choosing the used data structures).
- All data structures that need to be communicated via MPI must be serialized in some way.
- Pointers can not easily point into another MPI rank's memory.
- Pointer addresses change if you copy a structure.
- A structure with relative offsets can be more easily merged together than a structure with absolute offsets.
- Can the Barnes-Hut tree construction also be parallelized?

→ Think about **what** can be computed independently from each other!
→ Think about **how** these computations can be parallelized!
→ Think about **what** has to be communicated between different nodes!

Bonus Material: Visualization via ParaView

- Target System
  - Connecting to sgscl1
- SLURM
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- Simulation Data
- Value Approximation
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- Parallelization
- **Visualization via ParaView**
  - **.pvd and .vtp Files**
  - **ParaView**
- Simulation
  - Validation
- Important Deadlines

Visualization via ParaView

- Visualizing the data via ParaView.
- Create one `.vtp` file, containing information about the current simulation state (e.g., body mass, position, energies of the system, ...) for every visualization step on **every** MPI rank containing only the data it is responsible for.
- Create one `.pvd` file for the whole simulation on **one** MPI rank, referencing all `.vtp` files including the respective time step.

Example of a .vtp File with Two Bodies - 1

```
<?xml version="1.0"?>
<VTKFile type="PolyData" version="0.1" byte_order="LittleEndian" header_type="UInt64">
  <PolyData>
    <Piece NumberOfPoints="2" NumberOfVerts="2">
      <Points>
        <DataArray type="Float64" Name="position" NumberOfComponents="3" format="ascii">
                  0 0 0
                  -0.140728 -0.443901 -0.0233456
        </DataArray>
      </Points>
      <PointData>
        <DataArray type="Int32" Name="body_id" NumberOfComponents="1" format="ascii">
                  0
                  1
        </DataArray>
        <DataArray type="Float64" Name="velocity" NumberOfComponents="3" format="ascii">
                  5.37193e-06 -7.4078e-06 -9.4222e-08
                  0.0211745 -0.00710548 -0.00252296
        </DataArray>
        <DataArray type="Float64" Name="acceleration" NumberOfComponents="1" format="ascii">
                  9.15107e-06
                  0.022477
        </DataArray>
        <DataArray type="Float64" Name="mass" NumberOfComponents="1" format="ascii">
                  1.98847e+30
                  3.30101e+23
        </DataArray>
        <DataArray type="String" Name="name" NumberOfComponents="1" format="ascii">
                  83 117 110 0
                  77 101 114 99 117 114 121 0
        </DataArray>
```

Example of a `.vtp` File with Two Bodies - 2

```
      <DataArray type="Int32" Name="orbit_class" NumberOfComponents="1" format="ascii">
                  0
                  1
        </DataArray>
      </PointData>
      <Verts>
        <DataArray type="Int64" Name="offsets">
                  1 2
        </DataArray>
        <DataArray type="Int64" Name="connectivity">
                  0 1
        </DataArray>
      </Verts>
    </Piece>
    <FieldData>
      <DataArray type="Float64" Name="kinetic energy" NumberOfTuples="1" format="ascii">
                  7.18489e+22
      </DataArray>
      <DataArray type="Float64" Name="potential energy" NumberOfTuples="1" format="ascii">
                  -1.37988e+23
      </DataArray>
      <DataArray type="Float64" Name="total energy" NumberOfTuples="1" format="ascii">
                  -6.6139e+22
      </DataArray>
      <DataArray type="Float64" Name="virial equilibrium" NumberOfTuples="1" format="ascii">
                  1.04138
      </DataArray>
    </FieldData>
  </PolyData>
</VTKFile>
```

`.vtp` File General Remarks

- The `body_id` must be unique for each body **across all** MPI ranks.
- The `name` must be saved converting each character to its ASCII representation and adding a `NULL` terminator.

  Example:

S	u	n	
83	117	110	0

- The orbit class is an integer corresponding to one of the classes on the bonus material slide 24.
  → Devise a suitable mapping!
- The `offsets` and `connectivity` must start at `1` and `0` respectively on **each** MPI rank.

Example of a `.pvd` File

Note: `timestep` **does not** correspond to the current time step number but to the current simulation time (in earth days)!

Example with a visualization step width $vs = 5\,\mathrm{d}$ using two MPI ranks:

```
<?xml version="1.0"?>
<VTKFile type="Collection" version="0.1" byte_order="LittleEndian" compressor="vtkZLibDataCompressor">
  <Collection>
    <DataSet timestep="0.0" group="" part="0" file="time_series/0/sim.0.vtp"/>
    <DataSet timestep="0.0" group="" part="1" file="time_series/1/sim.0.vtp"/>
    <DataSet timestep="5.0" group="" part="0" file="time_series/0/sim.1.vtp"/>
    <DataSet timestep="5.0" group="" part="1" file="time_series/1/sim.1.vtp"/>
    <DataSet timestep="10.0" group="" part="0" file="time_series/0/sim.2.vtp"/>
    <DataSet timestep="10.0" group="" part="1" file="time_series/1/sim.2.vtp"/>
    <DataSet timestep="15.0" group="" part="0" file="time_series/0/sim.3.vtp"/>
    <DataSet timestep="15.0" group="" part="1" file="time_series/1/sim.3.vtp"/>
  </Collection>
</VTKFile>
```

ParaView Basics - 1

Most important ParaView views and options:

View → Properties used for general visualization properties

- Use `Points` as *Representation*.
- Use an appropriate *Point Size*.

View → Find Data used to extract data satisfying a specific property

- Can be used to create different groups for, e.g., different orbit classes (these groups can then be visualized with different colors or *Point Sizes*!).
- Use `name` in the *Point Labels* drop-down menu to print the body name during the simulation.
- Can also be used to search for a specific body (**Note:** unfortunately it is not possible to search by the body name in ParaView!).

View → Animation View used for more control over the simulation visualization

- Select a previously generated (via "Find Data") extraction.
- Select `Camera` and `Follow Data` in the two drop-downs and click the $+$.
- ParaView now follows the specified data during the simulation!

ParaView Basics - 2

Automate ParaView Tasks

- Start recording via *Tools* → *Start Trace*.
- Do all the stuff you want to be automated.
- Stop recording via *Tools* → *Stop Trace*.
- A new window opens where you can further customize the macro.
- Save the script as macro via *File* → *Save As Macro...*.
- The new macro can now be found under *Macros*.

Filters → Data Analysis → Plot Global Variables Over Time

used to display the energy plot

File → Save Screenshot...

used to save the **current** time step as an image

File → Save Animation...

used the save the **whole** simulation as a video

Bonus Material: Simulation

- Target System
  - Connecting to sgscl1
- SLURM
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- Simulation Data
- Value Approximation
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- Parallelization
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- **Simulation**
  - **Validation**
- Important Deadlines

Running the Simulation

The simulation must be executable on the command line as follows, whereby the command line parameters may be given in an arbitrary order!

```
./simulate --file data.csv --dt 1h --t_end 1y --vs 1d --vs_dir sim_out --theta 1.05
./simulate  --dt 1h --vs 1d --t_end 1y --theta 1.05 --file data.csv --vs_dir sim_out
```

`-file`	the used simulation data file
`-dt`	the time step width Δt
`-t_end`	the end time of the simulation
`-vs`	the visualization step width
`-vs_dir`	the top most output directory for all visualization files
`-theta`	the threshold in the Barnes-Hut algorithm

The above invocation simulates `data.csv` for one earth year with a resolution of one hour visualizing each day and writing all visualization files to the folder `sim_out/`.

Running the Simulation - Time Duration Parameters

For a more intuitive command line experience, the values of the three command line parameters `-dt`, `-t_end`, and `-vs` are positive, decimal numbers followed by one of `h` (hours), `d` (days), `m` (months), or `y` (years).

Examples for valid values are: $1\,\mathrm{h}$, $12\,\mathrm{d}$, $3.5\,\mathrm{m}$, or $1.25\,\mathrm{y}$.

Examples for invalid values are: $-2\,\mathrm{h}$, $60\,\mathrm{s}$, or $1.5\,\mathrm{hd}$.

Running the Simulation - Time Duration Parameters

For a more intuitive command line experience, the values of the three command line parameters `-dt`, `-t_end`, and `-vs` are positive, decimal numbers followed by one of `h` (hours), `d` (days), `m` (months), or `y` (years).

Examples for valid values are: $1\,\mathrm{h}$, $12\,\mathrm{d}$, $3.5\,\mathrm{m}$, or $1.25\,\mathrm{y}$.

Examples for invalid values are: $-2\,\mathrm{h}$, $60\,\mathrm{s}$, or $1.5\,\mathrm{hd}$.

Note: you should convert all values into earth days for your internal calculations:

- $1\,\mathrm{h} \equiv 0.041\,666\,7\,\mathrm{d}$
- $1\,\mathrm{m} \equiv 30.4167\,\mathrm{d}$
- $1\,\mathrm{y} \equiv 365.25\,\mathrm{d}$

Validation Hints for the Simulation

- Use the ParaView visualization to validate your simulation:
  - Is the simulation stable, i.e., do all planets and their moons have stable orbits?
  - Use the known orbital periods of objects:
    - Earth approx. $1\,\mathrm{y}$ ($365.25\,\mathrm{d}$)
    - Mars approx. $687\,\mathrm{d}$
    - Jupiter approx. $12\,\mathrm{y}$ ($4383\,\mathrm{d}$)
    - Europa (moon) approx. $85\,\mathrm{h}$ ($3.54\,\mathrm{d}$)
- Use the Virial Equilibrium and the laws of mass and energy conservation as additional mechanisms to detect errors.

Validation Hints for the Simulation

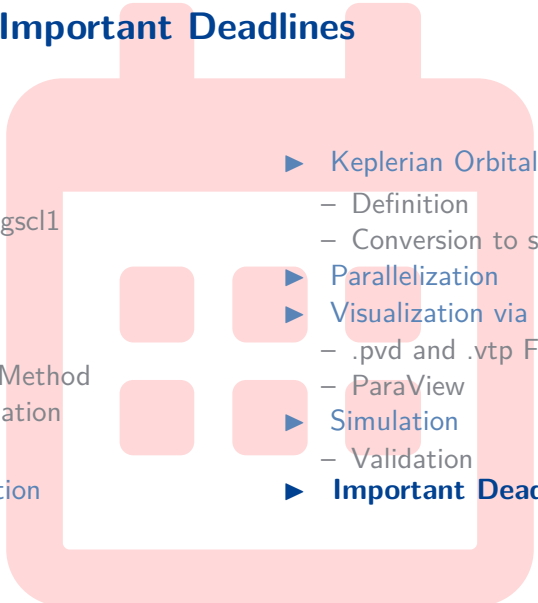
- Use the ParaView visualization to validate your simulation:
  - Is the simulation stable, i.e., do all planets and their moons have stable orbits?
  - Use the known orbital periods of objects:
    - Earth approx. $1\,\mathrm{y}$ ($365.25\,\mathrm{d}$)
    - Mars approx. $687\,\mathrm{d}$
    - Jupiter approx. $12\,\mathrm{y}$ ($4383\,\mathrm{d}$)
    - Europa (moon) approx. $85\,\mathrm{h}$ ($3.54\,\mathrm{d}$)
- Use the Virial Equilibrium and the laws of mass and energy conservation as additional mechanisms to detect errors.

Attention!:

It may happen that Pluto and its moons will not form a stable system, i.e., some moons (mainly Kerberos and Hydra) will be ejected from the system. This is most likely **not** a bug in your simulation, but the consequence of insufficient data from JPL's Horizons website.

For more information see:
S. Kenyon, B. Bromley: A Pluto-Charon Sonata IV. Improved Constraints on the Dynamical Behavior and Masses of the Small Satellites"(2022) (https://arxiv.org/pdf/2204.04226.pdf)

Bonus Material: Important Deadlines



- Target System
  - Connecting to sgscl1
- SLURM
- Theory
  - Barnes-Hut
  - Time Stepping Method
  - Complete Simulation
- Simulation Data
- Value Approximation
- Keplerian Orbital Elements
  - Definition
  - Conversion to state vectors
- Parallelization
- Visualization via ParaView
  - .pvd and .vtp Files
  - ParaView
- Simulation
  - Validation
- **Important Deadlines**

Important Deadlines

25.10.2023	Kick-off
02.11.2023	Deadline Group Formation
04.12.2023	Deadline Phase 1
29.01.2024	Deadline Phase 2
TBA	Final Presentations

→ **Do not** forget to register for our lab course in C@MPUS!