# Math 4997-3

## Lecture 4: N-Body simulations, Structs, Classes, and generic functions

Patrick Diehl

https://www.cct.lsu.edu/~pdiehl/teaching/2021/4997/

Reminder

$N$-body simulations

Structs

Generic programming

Summary

References

# Reminder

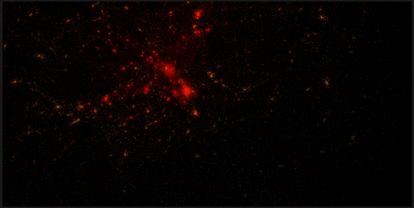
# Lecture 3

## What you should know from last lecture

- Iterators
- Lists
- Library algorithms
- Numerical limits
- Reading and Writing files

# $N$-body simulations

# *N*-body simulations[1]



The *N*-body problem is the physically problem of predicting the individual motions of a group of celestial objects interacting with each other gravitationally.

## Informal description:

Predict the interactive forces and true orbital motions for all future times of a group of celestial bodies. We assume that we have their quasi-steady orbital properties, e.g. instantaneous position, velocity and time.

# Recall: Vectors and basic operations

## Vectors

$$\mathbf{u} = (x, y, z) \in \mathbb{R}^3$$

1. Norm: $|\mathbf{u}| = \sqrt{x^2 + y^2 + z^2}$
2. Direction: $\frac{\mathbf{u}}{|\mathbf{u}|}$

## Inner product

$$\mathbf{u}_1 \circ \mathbf{u}_2 = x_1 x_2 + y_1 y_2 + z_1 z_2$$

## Cross product

$$\mathbf{u}_1 \times \mathbf{u}_2 = |\mathbf{u}_1||\mathbf{u}_2| sin(\theta)\mathbf{n}$$

where **n** is the normal vector perpendicular to the plane containing $\mathbf{u}_1$ and $\mathbf{u}_2$.

# Stepping back: Two-body problem

Let $m_i, m_j$ be the masses of two gravitational bodies at the positions $\mathbf{r}_i, \mathbf{r}_j \in \mathbb{R}^3$

## Three definitions:

1. The Law of Gravitation: The force of $m_i$ acting on $m_j$ is
   $\mathbf{F}_{ij} = G m_i m_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}$
2. The Calculus:
   2.1 The velocity of $m_i$ is $\mathbf{v}_i = \frac{d\mathbf{r}_i}{dt}$
   2.2 The acceleration of $m_i$ is $\mathbf{a}_i = \frac{d\mathbf{v}_i}{dt}$
3. The second Law of Mechanics:
   $\mathbf{F} = m\mathbf{a}$ (Force is equal mass times acceleration)

The universal constant of gravitation $G$ was estimated as $6.67408 \cdot 10^{-11} m^3 kg^{-1} s^{-2}$ in 2014 [8].

# Put all together: Equation of motion

Derivation for the first body:

$$\mathbf{F}_{ij} = Gm_i m_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}$$

$$m_i \mathbf{a}_i = Gm_i m_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_i - \mathbf{r}_j|^3}$$

$$\frac{d\mathbf{v}_i}{dt} = Gm_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}$$

$$\frac{d^2\mathbf{r}_i}{dt^2} = Gm_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}$$


For the second body follows: $\frac{d^2\mathbf{r}_j}{dt^2} = Gm_j \frac{\mathbf{r}_1 - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3}$

Note that we used Newton's law of universal gravitation [9].

# The $N$-body problem

## The force for body $m_i$

$$\mathbf{F}_i = \sum_{j=1,i\neq j}^{n} \mathbf{F}_{ij} = \sum_{j=1,,i\neq j}^{n} G m_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}$$

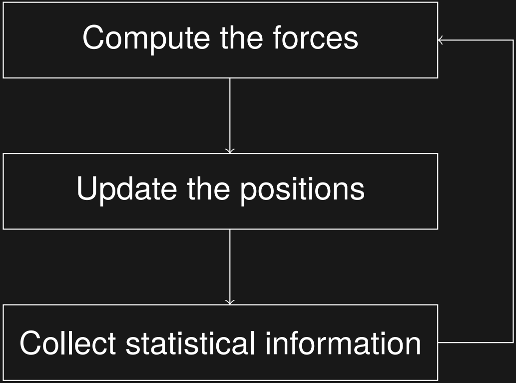
## Law of Conservation:

1. Linear Momentum: $\sum_{i=1}^{n} m_i \mathbf{v}_i = M_0$
2. Center of Mass: $\sum_{i=1}^{n} m_i \mathbf{r}_i = M_0 t + M_1$
3. Angular Momentum: $\sum_{i=1}^{n} m_i (\mathbf{r}_i \times \mathbf{v}_i) = \mathbf{c}$
4. Energy: T-U=h with

   $$T = \frac{1}{2} \sum_{i=1}^{n} m_i \mathbf{v}_i \circ \mathbf{v}_i, U = \sum_{i=1}^{n} \sum_{j=1}^{n} G \frac{m_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|}$$

More details: Simulations [2] and Astrophysics [1].

# Algorithm


Compute the forces
Update the positions
Collect statistical information

# Complexity of force computation

## Force computation: Direct sum

```
for(size_t i = 0; i < bodies.size(); i++)
for(size_t j = 0; j < bodies.size(); j++)
//Compute forces
```

### Advantage:

Robust, accurate, and completely general

### Disadvantage:

1. Computational cost per body $\mathcal{O}(n)$
2. Computational cost for all bodies $\mathcal{O}(n^2)$

Tree-based codes or the Barnes-Hut method [3] reduce the computational costs to $\mathcal{O}(n\log(n))$. More details [6].

# Update of positions

Assume we have computed the forces already, using the direct sum approach and now we want to compute the evolution of the system over the time $T$:

## Discretization in time:

- $\Delta t$ the uniform time step size
- $t_0$ the beginning of the evolution
- $T$ the final time of the evolution
- $k$ the time steps such that $k\Delta t = T$

Question: How can we compute the derivatives $dt$ and $dt^2$ of the velocity **v** and the acceleration **a** of a body?

# Finite difference and Euler method

## Finite difference

We can use a finite difference method to approximate the derivation by

$$u'(x) \approx \frac{u(x+h)-u(x)}{h}$$

## The Euler method

We use the finite difference scheme to approximate the derivations by

$$\mathbf{a}_i(t_k) = \frac{\mathbf{F}_i}{m_i} = \frac{\mathbf{v}_i(t_k) - \mathbf{v}_i(t_k - 1)}{\Delta t} \tag{1}$$

$$\mathbf{v}_i(t_k) = \frac{\mathbf{r}_i(t_{k+1}) - \mathbf{r}_i(t_k)}{\Delta t} \tag{2}$$

More details [10, 7, 5]

# Compute the velocity and updated position

## Velocity

$$\mathbf{v}_i(t_k) = \mathbf{v}_i(t_{k-1}) + \Delta t \frac{\mathbf{F}_i}{m_i} \text{ using (1)}$$

## Updated position

$$\mathbf{r}_i(t_{k+1}) = \mathbf{r}_{t_k} + \Delta t \mathbf{v}_i(t_k) \text{ using (2)}$$

Note that we used easy methods to update the positions and more sophisticated methods, *e.g.* Crank–Nicolson method [4], are available

# Structs

# Looking at the data structure[2]

For the $N$-body simulations, we need three dimensional vectors having

- $x$ Coordinate
- $y$ Coordinate
- $z$ Coordinate

```
struct vector {
double x;
double y;
double z;
};
```

## Initialization

```
struct vector v = {.x=1, .y=1, .z=1};
struct vector v1 = {1,1,1};
```

## Reading/Writing elements

```
std::cout << v.x << std:endl;
v.z=42;
```

[2] https://en.cppreference.com/w/c/language/struct

# Constructor[3]

## Assign initial values

```
struct A
{
    int x;
    A(int x = 1): x(x) {};
};
```

## A constructor has a

- Name `A`
- Arguments `int x = 1`
- Assignment `: x(x)`

Now `struct A a;` is equivalent to `struct A a = {1};`

3 https://en.cppreference.com/w/cpp/language/default_constructor

# Access specifiers[4]

```
struct A
{
    public:
    A(int x = 1): x(x) {};

    private:
    int x;
};
```

- `public` - The function and members have public access
- `private` - The function and members are only accessible within the struct

[4] https://en.cppreference.com/w/cpp/language/operator_member_access

# Access specifiers[5]: Example

```
struct A
{
    public:
    A(int x = 1): x(x) {};

    private:
    int x;
};

A a = A(10);
// Will not work since x is declared private
A.x = 1;
```

Solution: Providing `public` method to read and write the varibale `a`.

[5] https://en.cppreference.com/w/cpp/language/operator_member_access

# Access specifiers[6]: Access methods

```
struct A
{
    public:
    A(int x = 1): x(x) {};

    // So-called get method
    int getX(){ return x;}

    // So-called set method
    void setX(int value){ x = vlaue;}

    private:
    int x;
};
```

[6] https://en.cppreference.com/w/cpp/language/operator_member_access

# Functions[8]

## Compute the norm of the vector

```
#include <cmath>
struct vector2 {
double x , y , z;
vector2(double x = 0, double y=0, double z=0)
        : x(x) , y(y) ,z(z) {}
double norm(){ return std::sqrt(x*x+y*y+z*z);}
}
```

## Usage

```
struct vector2 v;
std::cout << v.norm() << std::endl;
```

`#include <cmath>`[7] provides mathematical expressions

[7] https://en.cppreference.com/w/cpp/header/cmath

[8] https://en.cppreference.com/w/cpp/language/functions

# Generic programming

# Why we need generic functions?

## Example

```
//Compute the sum of two double values
double add(double a, double b) {
return a + b;
}
//Compute the sum of two float values
float add(float a, float b) {
return a + b;
}
```

## Reasons:

- We have less redundant code
- The C++ standard library makes large usage of generic programming, *e.g.* `std::vector<double>`, `std::vector<float>`

# Function template[9]

## Writing a generic function:

```
template<typename T>
T add(T a, T b)
{
return a + b;
}
```

## Using the generic function:

```
std::cout << add<double>(2.0,1.0) << std::endl;
std::cout << add<int>(2,1) << std::endl;
std::cout << add<float>(2.0,1.0) << std::endl;
```

## Additional way to use the generic function:

```
std::cout << add(2,1) << std::endl;
```

[9] https://en.cppreference.com/w/cpp/language/function_template

# Generic structs[10]

## Writing a generic vector type

```
template<typename T>
struct vector {
T x;
T y;
T z;
};
```

## Using a generic vector type

```
struct vector<double> vd = {1.5,2.0,3.25};
struct vector<float> vf = {1.25,2.0,3.5};
struct vector<int> vi = {1,2,3};
```

[10] https://en.cppreference.com/w/cpp/language/templates

# Example

## Generic struct having functions

```
#include <cmath>

template<typename T>
struct vector {
T x , y , z;
vector( T x = 0, T y=0, T z=0)
        : x(x) , y(y) ,z(z) {};
T norm() { return std::sqrt(x*x+y*y+z*z);}
T cross(struct vector<T> b)
{return x*b.x+y*b.y+z*b.z;}
};
```

## What we need to define the vector data structure:

- Structs
- Generic functions

# Summary

# Summary

## After this lecture, you should know

- $N$-Body simulations
- Structs
- Generic programming (Templates)

## Further reading:

- C++ Lecture 2 - Template Programming[11]
- C++ Lecture 4 - Template Meta Programming[12]

[11] https://www.youtube.com/watch?v=iU3wsiJ5mts

[12] https://www.youtube.com/watch?v=6PWUByLZO0g

# References

# References I


[1] Sverre Aarseth, Christopher Tout, and Rosemary Mardling.
*The Cambridge N-body lectures*, volume 760.
Springer, 2008.

[2] Sverre J Aarseth.
*Gravitational N-body simulations: tools and algorithms*.
Cambridge University Press, 2003.

[3] Josh Barnes and Piet Hut.
A hierarchical o (n log n) force-calculation algorithm.
*nature*, 324(6096):446, 1986.

# References II

[4] John Crank and Phyllis Nicolson.
A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type.
In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 43, pages 50–67.
Cambridge University Press, 1947.

[5] Leonhard Euler.
*Institutionum calculi integralis*, volume 1.
impensis Academiae imperialis scientiarum, 1824.

[6] Donald Ervin Knuth.
*The art of computer programming: Fundamental Algorithms*, volume 1.
Pearson Education, 1968.

# References III


[7] Randall J LeVeque.
*Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, volume 98.
Siam, 2007.

[8] Peter J Mohr, David B Newell, and Barry N Taylor.
Codata recommended values of the fundamental physical constants: 2014.
*Journal of Physical and Chemical Reference Data*, 45(4):043102, 2016.

[9] Isaac Newton.
*Philosophiae naturalis principia mathematica*, volume 1.
G. Brookman, 1833.

# References IV


[10] John C Strikwerda.
*Finite difference schemes and partial differential equations*, volume 88.
Siam, 2004.