



**Politechnika
Śląska**

SILESIA UNIVERSITY OF TECHNOLOGY

Faculty of Automatic control, Electronics and Informatics

Computer programming Gravitational force - final report

Student: Robert Okenczyc 298990
Instructor: Piotr Fabian
Year: 2022 (III sem)

Project topic:

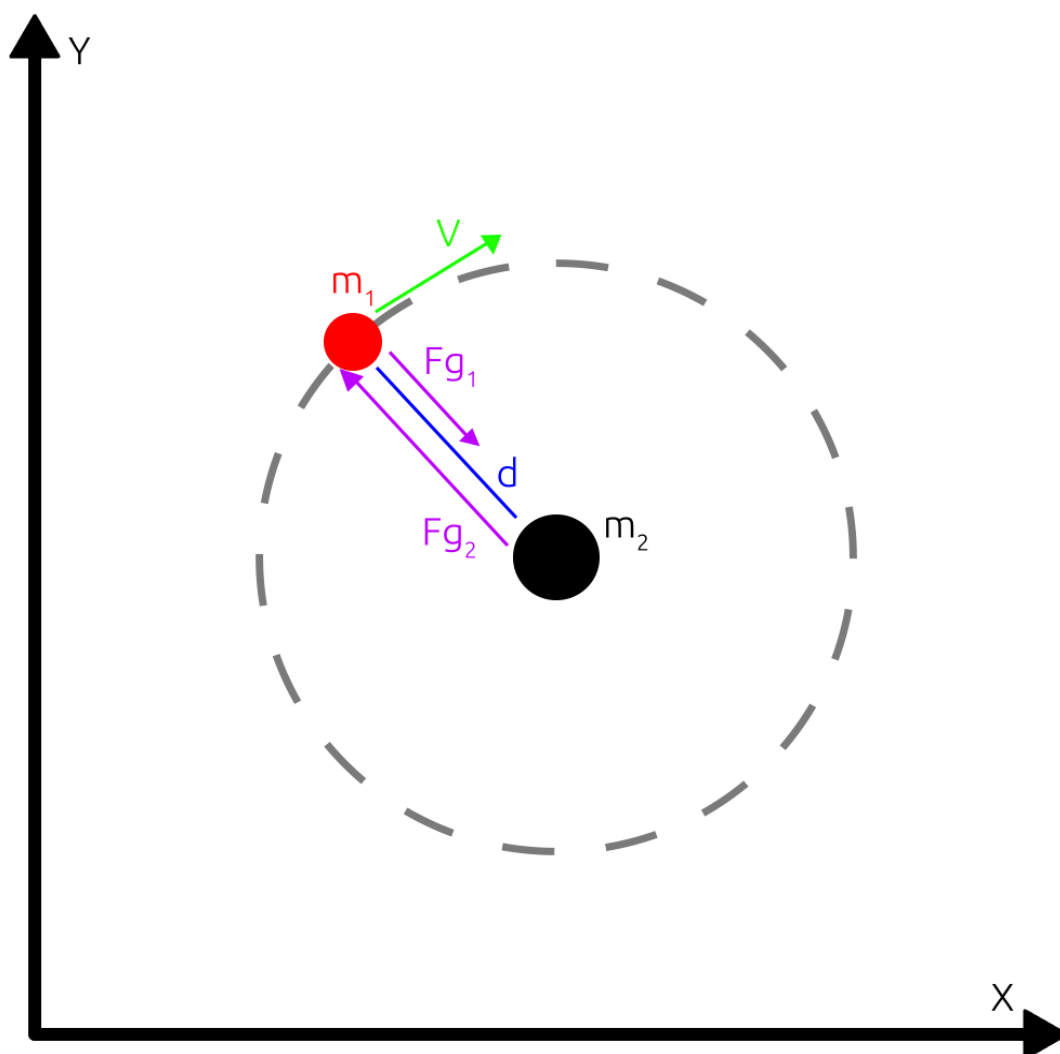
Gravitational Force - write a gravitation simulation program using graphics from Allegro 5

Brief description:

This library contains classes describing elements in 2D field, with parameters needed for proper gravity simulation and graphical expression (color and radius). Beside elements there are classes used for initialization of Allegro window and animating given data by 'frames' functionality.

Task Analysis:

The main problem of this task was to understand how time affects element position and calculating it using inner velocity and gravitational force from other elements. To know the distance to travel program uses Pythagorean theorem and velocity with gravitational force every frame.



Object with high mass are considered as stationary for visual purposes. On every objects forces are working with others objects. Meaning that, no matter where specific element is, every other elements are interfering its velocity, but in realistic manner. The larger distance between elements the weaker forces they are dealing with.

Externals:

This task did not ask for being external library, but I prefer to do it this way for code to be more readable and main program to be easy and basic. End user don't have to understand whole code to use it. Only things that may be need to code yourself are events. Events are handled by Allegro 5 library. In task example is (poorly) written event for making new element each time the mouse buttons are pressed. In library there are also some debugging functions for printing elements into console. Also in ready-to-run project there is functionality of loading cvs files. The formatting of data is as follows:

```
x_coordinate, y_coordinate, mass, radius, if_random_color, x_velocity, y_velocity  
x_coordinate, y_coordinate, mass, radius, if_random_color, x_velocity, y_velocity
```

example:

```
500, 500, 10000, 30, 1, 0, 0  
500, 380, 3000, 10, 1, 65, 0  
500, 800, 1000, 10, 1, -50, 0
```

Task example:

An example of using this library is here or in the documentation below. This example is well commented for programmers to understand.

```
#include <iostream>  
#include <string>  
#include <time.h>  
#include "gravitationLib.h"  
  
using namespace std;  
  
int main(int argc, char** argv)  
{  
    int seed = 1230;  
    srand(seed);  
  
    particleInit(50);  
  
    canvas Canvas1(1000, 1000, "Gravitational Force");  
  
    cout << "\nGenerating canvas\nPress Escape key to exit Allegro window\n";  
  
    while (Canvas1.endCondition()) {  
  
        if (Canvas1.event.type == ALLEGRO_EVENT_MOUSE_BUTTON_DOWN) {  
            addParticle(Canvas1.event.mouse.x, Canvas1.event.mouse.y);  
        }  
  
        if (Canvas1.FrameInit()) return 0;  
  
        particleDraw();  
        movement();  
        Canvas1.keyframe();  
    }  
    return 0;  
}
```

Testing:

This library was tested with different values, but first - why I have chosen double over float. Well answer to this questions is pretty simple - precision. Whenever I used float or approximate values after some time with transformations the figures bend and deform - be not accurate.

The input file algorithm was not tested properly and some bugs may happen.

There is not limit in random generated elements (beside obvious memory limit), but rendering more than 5000 elements could slow down simulation at the beginning. In some frames later probably some elements will be deleted (if elements touches other element the older one get deleted) so the hypothetical slow down will occur only with a few seconds.

Concusion:

This task was easy to understand and the physics behind this was not complicated. If I could rewrite some of code or add some more functionality I would probably prepare this program to use real units. With real units it would be possible to (approximately because of 2D space) simulate natural real cosmic object simulation, like solar system. For convenience I would make file handling in library and make it more resist for any user mis usage and more open for changes.