1 st Project

Mandatory, groups of 2 people (preferred) or individually. Deadline: most probably before Christmas, oral exam will follow.Weight: approximately 25% of the final grade.

A) Implement Unity script(s) in C # (https://docs.unity3d.com/Manual/ScriptingSection.html) that simulate particle systems with the following behaviors:

1. Particles which are continuously created from a certain position in space, with random (direction & magnitude) initial velocities, moving under the effect of gravity and a wind force $\boldsymbol{f}_{wind} = k_w \boldsymbol{v_w}$ and bouncing (coefficient of restitution ε =0.7)on the walls of a transparent (invisible) axis aligned cube that surrounds them. See something similar (simpler) at https :// processing . org / examples / simpleparticlesystem . html . You **do not need** to implement the cube as an object, simply take into account the positions of its faces in a simple collision detection (that you will implement, see below) so that particles bounce on them. Wind speed $\boldsymbol{v}_w$ is constant everywhere and (optionally) can change over time.
2. Seek & pursue steering behaviors in 2D : A target particle moves along a straight line on a plane (without dynamics, its motion can, for example, be implemented through translation). 2 particles start their movement from some point of the plane; one seeks and the other pursues the target-particle (see below). The camera is placed somewhere above the plane and "looks" towards it, so that we have a top - down view. The animation stops after some time.
3. Arrive steering behavior in 2D : We define a stationary target particle on a plane. From some point of the plane a particle starts its motion so as to arrive at the target particle(see below). The camera is placed somewhere above the plane and "looks" towards it, so that we have a top - down view. The animation stops after some time e.g. when the moving particle reaches the target. Also add to (2) above a particle that arrives at the moving target particle.
4. Bird flocking (see end of text for references and code)
5. **Optional** (+1 point) : Particles (specific number) which are initially located together at the center of a transparent sphere (see below) and repulsive forces inversely proportional to their distance are exerted on them. Similar to (1), it is not necessary to implement a sphere object but simply take it into account during collision detection and response.
6. **Optional** (possibly additional points): additional steering behaviors , a target particle in (2) that the user moves with the mouse, etc.

Comments / notes

- The script (s) should calculate the position and velocity of each particle by simple Euler integration. based on the exerted forces (or accelerations in the case of steering behaviors).
- They should also check for possible collisions with the cube (case 1) and the sphere (optional case 5) by comparing the position of the particle with the boundaries of the cube (e.g. check the x coordinate of the particle with the position of the cube's faces that are perpendicular to x , or alternatively use the methodology presented on the slides) or compare its distance from the center of the sphere with its radius.
- In the event of a collision, the component of velocity that is perpendicular to the collision surface should change.
- Suggestion for calculating time and point of collision: assume linear motion with constant speed, as in the problem we solved in class.
- In the case of the sphere (optional case 5) the point of collision is the point on the linear particle trajectory that satisfies the sphere equation (its distance from the center is equal

to the radius). This results in a 2nd degree equation with respect to time t , from which we can find the collision time (of the 2 possible solutions of the equation, we keep the appropriate one).

- After position and time of collision as well as post-collision velocity have been calculated, perform (from the collision position) an Euler step with Δt equal to the time that the particle traveled after the collision and render the particle in the evaluated position. You do not have to render the particle at the time of collision.
- The steering behaviors seek, pursue, arrive are described on slides "Game Ai movement 2", slides 13-48 & 71-73. The pseudocode calculates the acceleration of the particle that seeks / pursues another one. This acceleration is used (instead of force / mass which we use as an acceleration term in classical dynamics) to calculate its new speed. In the case of pursue assume that the velocity of the target particle is known.
- In all cases ignore collisions between particles.

Alternatively, the above can be implemented in a) Unreal Engine with blueprints or C ++ programming or b) C and OpenGL . You can optionally use the ready-made particle systems provided by Unity but **only in addition** to your own scripts. You **shall not** complete this assignment only with the particle systems of Unity and the functionalities offered by the ParticleSystem class and collision handling.

B) Build a scene in Unity as described in http :// web . Stanford . edu / class / cs 2 4 8/ assignment / assignment 1. pdf and place some of your constructions. The scene has to have the following features

- Scene contains geometry added via the Unity editor
- Scene incorporates at least one texture
- Scene incorporates at least two different types of light (eg a directional light and a point source)
- Scene includes at least one object imported from another source (eg made in Blender, downloaded online, etc.)

Bird flocking (you can "borrow" code but you have to understand it)
https :// threejs . org / e xamples /? q = bird # webgl _ gpgpu _ birds
https :// processing . org / examples / flocking . html
https :// dl . acc . org / doi /10.1145/37402.37406
https :// dl . acc . org / doi /10.1145/37402.37406

Useful stuff

- Unity tutorials and manual
- Unity intro: http://web.stanford.edu/class/cs248/pdf/CS248_Lecture_2.pptx
- Particle systems: http://web.stanford.ed u /class/cs248/pdf/class_08_particle_systems.pdf