DH2323
Computer Graphics and Interaction
Lab 2: Transformations and Particle Systems

Viktor Meyer viktorme@kth.se

June 15, 2020

1 Introduction to the Unity Game Engine

1.1 General

Unity is one the most popular game engines and is widely employed within the games industry. With the designation of being a game engine comes a lot of features. This means that developers using the engine are not required to build everything from the ground up, but can conveniently use existing functionality. Examples of existing functionality include: renderer, physics, scene hierarchy, high-level scripting and interactive editor.

1.2 Project structure

When working with the Unity Editor everything is organized by a specific project structure. A project generally contains Assets that are used during run-time, these can include everything from 3d-models to scripts. It is common to organize the folder structure in a Unity project into: resources, scenes and scripts. Out of these three, the resource folder is special since it allows assets to be loaded on demand from scripts using *Resources.Load()*.

1.3 Scene hierarchies

A project in Unity can contain multiple scenes where each scene contains a number of game objects. Game objects allow a developer to specify hierarchical relationships between objects. Game objects can contain components/scripts that modify the state of the game objects that they are attached to. The transform component is an example of this, most notably it contains information about the position of the game object. Since game objects can be ordered hierarchically there may be parent-child relationships. The position of a parent object will for example affect all of its children's positions.

1.4 Update loop

Understanding the Unity update loop is critical when working with the game engine. All scripts that work with the update loop derive from the *MonoBehaviour* class. Deriving from the MonoBehaviour class allows scripts to receive callbacks from Unity's update loop. The most frequently used callbacks are: Awake(), Start(), Update() and FixedUpdate(). Awake() can be used to initialize variables or state and is always executed before Start(). Start() is called before the first Update() is executed. Update() is called every frame and most commonly used for game logic. FixedUpdate() is frame-rate independent and runs at the same rate as the physics system, any physics related logic should usually run in this function.

2 Tank

2.1 Tank movement

Unity provides a basic input layer that can listen to keyboard or mouse input. It is possible to access various input mappings by using *Input.GetAxis()*. It makes sense to use the *Vertical* and *Horizontal* axis for tank forward+backward and turning respectively. Reading Input values should be done every frame for responsiveness which is why it is a good idea to do it in the Update() function.

With the input read, the next step is to move the tank which involves the physics system. This means logic should ideally happen inside FixedUpdate(). The physics system provides an easy to use API where it is possible to move rigidbodies with MoveRigidbody(). The MoveRigidbody() function takes a Vector3 as an arugment and moves the rigidbody to the specified position. It is easy to see that it is possible to use the previously read input to create a movement vector for the tank and actually move it. When moving the tank it is essential to scale the vector with Time.fixedDeltaTime so that movement is independent from frame-rate. **Note that the lab example uses Time.deltaTime instead of Time.fixedDeltaTime which defeats the purpose of executing inside FixedUpdate(). It is not good practice to use Time.deltaTime inside FixedUpdate() since it by definition depends on frame-rate**

Turning the tank is achieved in similar fashion to moving the tank. The horizontal input axis is used create a rotation quaternion. The rotation is then passed to the physics system that handles the rest when called with MoveRotation().

2.2 Tank wheel animation

TankController.cs contains a list of all the tanks wheels. To animate the wheels they need to rotate in proportion to the tank movement. This can be achieved by taking a constant wheel rotation speed factor and multiplying it with the current input values. This calculation is then multiplied by Time.fixedDeltaTime to

decouple it from the rendering frame-rate. Finally all wheel transforms are rotated using Rotate() in local space. Local space is used since the wheels should rotate relative to the rest of the tank.

2.3 Tank turret animation

The tank turret should naturally be aimed at the position of the end-users mouse position. Unity provides a function ScreenPointToRay() that takes a 2d position from screen-space and returns a world-space Ray. This ray can then be cast using the physics systems Raycast() to return an intersection point of the ray in the scene. To compute a direction vector from the tank to the aimed position, it is simply a matter of subtracting the ray point with the current tank position. The direction vector can be vertically rotated which is undesirable, to prevent this the direction vectors y-component should be zeroed. With the direction vector calculated, the last step is to calculate a quaternion with LookRotation() and apply it to the tank.

3 Particle system

3.1 Fire flame

Particle systems can be used to simulate various effects such as flames, rain, snow, splashes and explosions in Unity. The main advantage of particle systems is that they are easy to work with and provide a lot of tweakable settings. The settings can be used to create interesting behaviours of the particle systems. Notable settings include: duration, speed, lifetime, color and shape. Below is an example of a particle system that attempts to look like a flame.

3.2 Shooter

The end-user can fire an AIM-9 SIDEWINDER missile at a target in the scene. The target listens for collisions using the physics callback OnTriggerEnter which spawns a new particle system that resembles an explosion.

The particle system is the shape of a sphere since that is the shape that best fits an explosion. In addition to this the colors are animated from yellow to orange to make the effect more vibrant. The particle system is also set to *one-shot* since explosions are only fired once and then discarded. To make the effect more dramatic all particles are emitted very rapidly to convey explosiveness. The velocity is also modified to be very high at the beginning and then fall-off as time progresses. Particles start out small and increase over time for effect.