Instituto Politécnico Nacional
Escuela Superior de Cómputo
Computer Graphics
2 Introduction to Unity
Malagón Baeza Alan Adrian
Rosas Trigueros Jorge Luis
17/04/2023
24/04/2023

Theoretical framework

When it comes to creating 3D animations, there are a variety of software tools available to artists and designers. One such tool is Unity, which has become a popular choice for game developers and animators alike. Unity is a powerful game engine that can also be used for creating animations and other interactive media. For those who are new to Unity and 3D animation, it can be a daunting task to figure out where to start. In this introduction, we will cover the basics of using Unity for animating 3D objects using transforms and vectors.

Firstly, let's define what transforms and vectors are. Transforms are a fundamental concept in Unity that refers to the position, rotation, and scale of an object in 3D space. Every object in Unity has a transform component that determines its location, orientation, and size. Vectors, on the other hand, are mathematical constructs that represent direction and magnitude. They are used extensively in Unity for movement, rotation, and scaling of objects.

To start animating in Unity, you'll need to create a new project and import your 3D object files into the scene. Unity supports a variety of file formats, including .fbx, .obj, and .3ds. Once you've imported your object, you can use the Inspector panel to view and modify its transform properties, such as position, rotation, and scale.

One of the most powerful features of Unity is the ability to use scripting to create complex animations. By using scripts, you can manipulate the transform and vector properties of objects in real-time, creating dynamic animations that respond to user input or other events.

In summary, Unity is a powerful tool for creating 3D animations that utilizes the concepts of transforms and vectors. Whether you're new to Unity or an experienced animator, the software provides a variety of tools and techniques that can be used to create stunning and engaging animations. With a little bit of practice and experimentation, you can create animations that bring your 3D objects to life.

Material and equipment

We will use C# (Visual Studio 2019), and Unity Hub 3.4.1 (Editor version: 2021.3.19f1). A personal computer with AMD Ryzen 5 5600H and 16 GB of RAM.

Practice development

Sphere

This is a C# script for a cylinder object in Unity game engine.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

The above lines include necessary libraries to access Unity API and generic collections in the code.

```
public class CylinderScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }
```

This is a class definition for the script that inherits from the MonoBehaviour class. The Start() function is called once when the script is first loaded.

```
    public Vector3 origin = new Vector3(0, 0, 0);
    public Vector3 myAxis = new Vector3 (1,1,0);
```

These are two public Vector3 variables that can be modified in the Unity Editor. origin defines the center of rotation for the cylinder, and myAxis defines the axis around which the cylinder rotates.

```
    // Update is called once per frame
    void Update()
    {
        transform.RotateAround(origin, myAxis, 180 * Time.deltaTime);
        //transform.Rotate(Vector3.up,180*Time.deltaTime);
    }
```

The Update() function is called once per frame and updates the rotation of the cylinder. The RotateAround() method rotates the object around a specified point (origin) and axis (myAxis) by a specified angle (180 * Time.deltaTime), which is scaled by the time since the last frame to ensure smooth rotation. The alternative Rotate() method is commented out and can be used instead to rotate the object around a fixed axis (Vector3.up in this case).

Overall, this script provides a simple way to make a cylinder object rotate continuously in a game or simulation.

Finally a beach ball texture was added to the sphere.

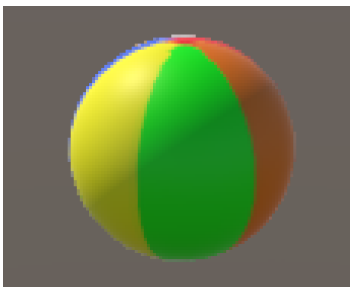

Figure 1. Sphere before animation

Figure 2. Sphere after animation

Cube

This is a C# script for a cube object in Unity game engine. Here's a breakdown of the code:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

The above lines include necessary libraries to access Unity API and generic collections in the code.

```
public class CubeScript : MonoBehaviour
{
    public float speed = 20.0f; // Speed of cube movement
    public float distance = 10.0f; // Distance between each vertex of the
square

    private Vector3[] vertices; // Array of vertices of the square
    private int currentVertex = 0; // Current vertex index that the cube
is moving towards

    void Start()
    {
        // Set up an array of vertices of the square
        vertices = new Vector3[4];
        vertices[0] = new Vector3(-5.0f, 0.0f, -5.0f);
        vertices[1] = new Vector3(5.0f, 0.0f, -5.0f);
        vertices[2] = new Vector3(5.0f, 0.0f, 5.0f);
        vertices[3] = new Vector3(-5.0f, 0.0f, 5.0f);
    }
```

This is a class definition for the script that inherits from the MonoBehaviour class. It includes public variables `speed` and `distance` which define the speed of cube movement and the distance between each vertex of the square. It also includes private variables `vertices` and `currentVertex` which represent an array of vertices of the square and the current vertex index that the cube is moving towards.

The `Start()` function is called once when the script is first loaded. In this case, it sets up an array of `vertices` for the square by assigning each vertex a position in 3D space.

```
    void Update()
    {
        // Get the position of the current vertex that the cube is moving
towards
        Vector3 targetPosition = vertices[currentVertex];

        // Move the cube towards the target position
        transform.position = Vector3.MoveTowards(transform.position,
targetPosition, speed * Time.deltaTime);

        // If the cube reaches the target position, move to the next
vertex
        if (transform.position == targetPosition)
        {
```

```
            currentVertex = (currentVertex + 1) % vertices.Length;
        }

        // Rotate the cube as it moves
        transform.Rotate(new Vector3(0, 1, 0), 50 * Time.deltaTime);
    }
```

The `Update()` function is called once per frame and updates the movement and rotation of the cube. First, it gets the position of the `currentVertex` that the cube is moving towards from the `vertices` array. Then, it moves the cube towards the `targetPosition` using the `MoveTowards()` method, which calculates the next position of the cube based on the current position, target position, and speed. If the cube reaches the `targetPosition`, it moves to the next vertex by incrementing `currentVertex` and using the modulo operator to loop back to the first vertex when it reaches the end of the array.

Lastly, the cube is rotated as it moves using the `Rotate()` method, which rotates the cube around the Y-axis by a fixed angle (`50 * Time.deltaTime` in this case).

Overall, this script provides a simple way to make a cube object move and rotate along a square path in a game or simulation.

Finally a Minecraft Slime Block texture was added to the cube.

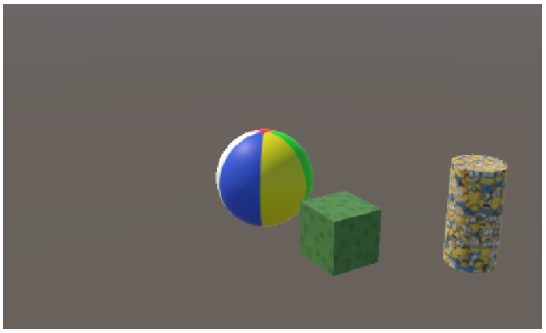

Figure 3. Cube before animation

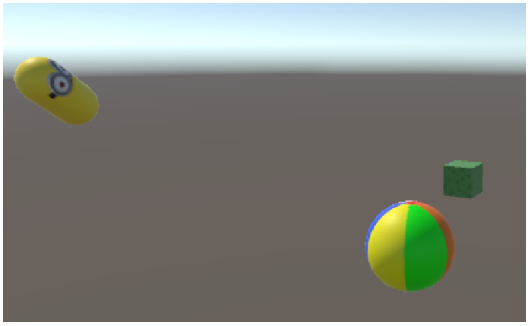

Figure 4. Cube after animation

Cylinder

This is a script written in C# for a cylinder object in Unity.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CylinderScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    public Vector3 origin = new Vector3(0, 0, 0);
    public Vector3 myAxis = new Vector3 (1,1,0);

    // Update is called once per frame
    void Update()
    {
        transform.RotateAround(origin, myAxis, 180 *
Time.deltaTime);
        //transform.Rotate(Vector3.up,180*Time.deltaTime);
    }
}
```

It contains a public Vector3 variable named "origin" which has a default value of (0,0,0), and a public Vector3 variable named "myAxis" which has a default value of (1,1,0). These variables are used to define the axis around which the cylinder will rotate.

The Update() function is called once per frame and rotates the cylinder around the "origin" point using the "myAxis" vector. The amount of rotation is determined by the product of 180 degrees and the time since the last frame (Time.deltaTime).
There is also an empty Start() function, which is called once when the script is initialized.

Finally a Minions texture was added to the cube.

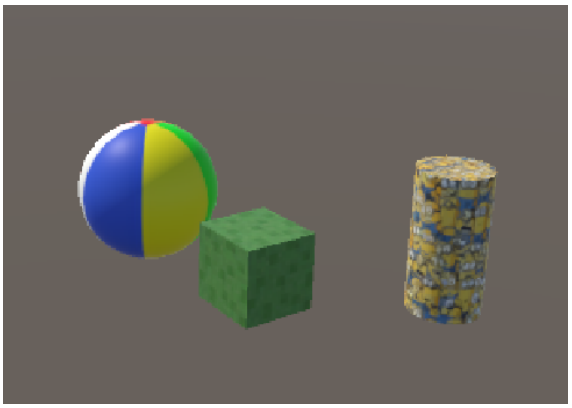

Figure 5. Cylinder before animation

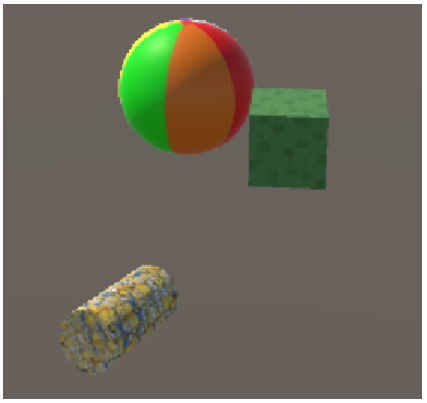

Figure 6. Cylinder after animation

Plane

This is a C# script for a plane object in Unity.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlaneScript : MonoBehaviour
{
    public float speed = 1.0f;      // velocidad de movimiento
    public float amplitude = 1.0f;  // amplitud de la onda

    private float startTime;        // tiempo de inicio del
movimiento
    private Vector3 startPos;       // posición de inicio del
movimiento

    void Start()
    {
        // guarda la posición de inicio y tiempo de inicio
        startTime = Time.time;
        startPos = transform.position;
    }

    void Update()
    {
        // calcula la nueva posición del objeto
        float time = (Time.time - startTime) % (2 * Mathf.PI);  //
tiempo desde el inicio del movimiento
```

```
        float x = time * speed;                                    //
componente x de la posición
        float y = amplitude * Mathf.Sin(time);                     //
componente y de la posición
        Vector3 newPos = startPos + new Vector3(x, y, 0.0f);      //
nueva posición del objeto

        // calcula la rotación del objeto
        Vector3 direction = new Vector3(speed, amplitude *
Mathf.Cos(time), 0.0f).normalized;
        transform.rotation = Quaternion.LookRotation(direction,
Vector3.up);

        // mueve el objeto a la nueva posición
        transform.position = newPos;
    }
}
```

It contains two public variables, "speed" and "amplitude", which determine the speed of movement and the amplitude of the wave, respectively.

The Start() function initializes two private variables, "startTime" and "startPos", which store the time of the start of the movement and the initial position of the plane, respectively.

The Update() function is called once per frame and calculates the new position of the plane based on time elapsed since the start of the movement, "speed", and "amplitude". The new position is calculated using the formula for a sine wave in 2D space. The function also calculates the rotation of the plane based on the direction of movement and sets the new position and rotation of the plane accordingly.

The rotation is calculated using the direction vector, which is a normalized vector pointing in the direction of the movement, and the up vector of the world, Vector3.up.

Overall, this script creates an object that moves in a sine wave pattern and rotates to face the direction of movement.

Finally a Magic Carpet texture was added to the cube.

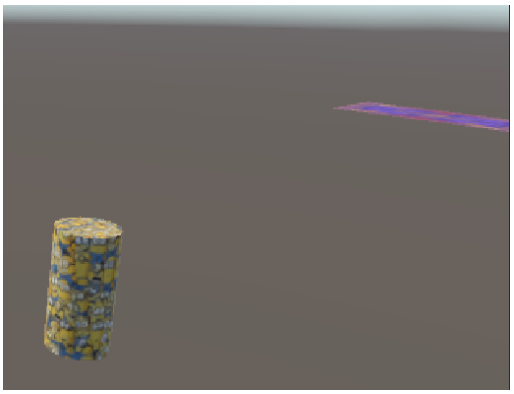

Figure 7. Plane before animation

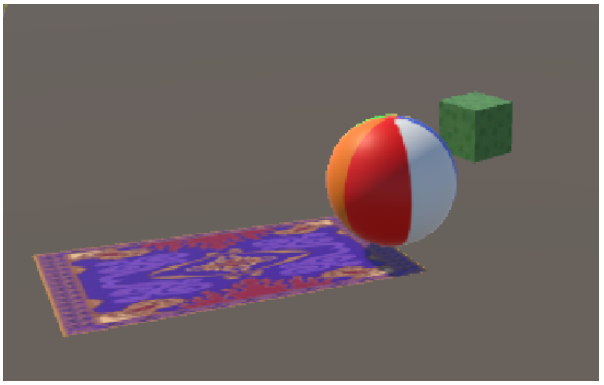

Figure 8. Plane after animation

Capsule

This script defines the behavior of a capsule in the Unity engine.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CapsuleScript : MonoBehaviour
{
    public float minScale = 1f;
    public float maxScale = 5f;
    public float scaleSpeed = 1f;
    public float rotationSpeed = 1f;

    void Start() {
    }

    void Update()
    {
        float scale = Mathf.PingPong(Time.time * scaleSpeed, maxScale -
minScale) + minScale;
        transform.localScale = new Vector3(scale, scale, scale);

        if (Random.value > 0.5f)
        {
            transform.Rotate(Vector3.up, 45f * rotationSpeed *
Time.deltaTime);
        }
        else
        {
            transform.Rotate(Vector3.forward, 45f * rotationSpeed *
Time.deltaTime);
        }
    }
```

```
}
```

In the `Start()` method, there is no code.

In the `Update()` method, the capsule's scale is set to a value that oscillates between `minScale` and `maxScale` using `Mathf.PingPong()` function. The scale oscillation speed is controlled by `scaleSpeed`.

The capsule's rotation is also updated in the `Update()` method. The `Random.value` function generates a random value between 0 and 1, and if it is greater than 0.5f, the capsule rotates around the `Vector3.up` axis, otherwise, it rotates around the `Vector3.forward` axis. The rotation speed is controlled by `rotationSpeed`.

Overall, this script creates a capsule that continuously changes its size and rotates randomly around different axes.

Figure 9. Capsule before animation

Figure 10. Capsule after animation

Conclusions and recommendations

In the previous codes, we can see different examples of how to manipulate 3D objects in Unity using C# scripts. Each script has its own purpose, whether it is to move a cube or a plane, rotate a cylinder or a capsule, and make them more dynamic with various effects such as scaling, rotation, and color change.

Textures and other assets can also be added to enhance the appearance of 3D objects in Unity. This can be done by dragging and dropping image files into the Unity Editor, creating a material, and applying it to the object. Additionally, many assets are available on the Unity Asset Store, including textures, models, animations, and sound effects, which can save time and effort in creating 3D scenes.

Overall, the codes presented here demonstrate some of the basic techniques used in manipulating 3D objects in Unity, and the importance of adding assets to enhance their appearance. With these techniques and assets, developers can create immersive and visually appealing 3D environments.

References

- Unity Technologies. (n.d.). Unity Manual: Materials. Retrieved from https://docs.unity3d.com/Manual/Materials.html
- Unity Technologies. (n.d.). Unity Asset Store. Retrieved from https://assetstore.unity.com/
- Unity Technologies. (n.d.). Transform. Retrieved from https://docs.unity3d.com/ScriptReference/Transform.html
- Robin Wood. (n.d.). Ball maps texture page. Retrieved April 25, 2023, from https://www.robinwood.com/Catalog/FreeStuff/Textures/TexturePages/BallMaps.html