On this page

Introduction to ProSprite

This tutorial will quickly take you through the basics of using ProSprite.

Download

First, download the ProSprite unity package from Github. If you're reading the offline documentation, *you've probably already done that.*

Open Samples

ProSprite comes with two different samples to test:

- A 'Main Demo', showcasing available features.
- An 'Extreme Demo', which fills the screen with layer on layer of high-res circles to show ProSprite's efficiency.

Try opening both of these samples, and looking around to see how things work. All of the code is layed out in the scripts folder of each Sample.

Create your first sprite

Setting up the script

While ProSprite is designed to be as easy to use as possible, **you will need to create your own scripts to take full advantage.** First, create an empty GameObject and add a new script to it. Typically, scripts inherit from MonoBehavior, but in order to use ProSprite, you need to have your class inherit from ProSprite. Just change the `MonoBehavior` label in the class declaration to `ProSprite`. ProSprite itself inherits from MonoBehavior, so you will still be able to use your script with objects.

If you've followed these instructions properly, you should have the error: "'MyScript' does not implement inherited abstract member 'ProSprite.Render()'". To fix this, replace `void Update()` with `override protected void Render`. The Render() function is called just before the camera renders a frame. This is where you will use ProSprite to place imagery on screen. Be sure to replace the comment `Update is called once per frame` with `Render is called before the camera renders a frame` You can also just remove the comment altogether, if that's your preference. Your script should look like this:

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

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

    }

    // Render is called before the camera renders a frame
    override protected void Render()
    {

    }
}
```

Initialize ProSprite with Setup()

Before we do anything, we must use the Setup function. Setup instantiates all needed shaders and creates all of the assets needed to render the ProSprite. **Without calling Setup(), it is impossible to render anything.** Keep this in mind. The Setup function takes in a width and a height parameter, for the width and height of your sprite. Try to make this as small as you can, and the larger your ProSprite is, the more resources it takes to draw anything. For this tutorial, we'll use a size of 64 by 64. To do so, simply type out the function `Setup(64, 64)`.

Draw a Circle

Finally, it's time to actually draw something to the screen. The simplest function available is DrawCircles(). This function takes an array of circles and draws them to the screen. Circles are a struct with three properties: position, radius, and color. We'll declare an array of circles. To do this, type out, at the stop of the Render function:

```
Circle[] myCircleArray = new Circle[] { new Circle(center, 256, 0) }
```

This constructs a Circle array called "myCircleArray," and fills it with one circle, with a position of `center`, a radius of 256, and a color of 0. Some notes on this:

- The "center" property is inherited from the ProSprite class, and it returns a Vector2 corresponding to the position of the center of the texture. It's useful when you want to draw something at the center of the screen.
- The radius of a circle is actually the square of what the radius of the circle will be in pixels when it is drawn. Here, we use a radius of 256, so the circle will have a radius of 16 (the square root of 25), meaning it will have a diameter of 32, filling the 32 x 32 canvas.
- The color of the circle is the index of a color in the palette. We will talk about how to change the palette later in the tutorial, but ProSprite comes loaded with 5 colors, so you won't have to worry about this right now.

Finally, call the DrawCircles function using myCircleArray as an argument. This should give you the following script:

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

public class MyScript : ProSprite
{
    // Start is called before the first frame update
    void Start()
    {
        Setup(32, 32);
    }

    // Render is called before the camera renders a frame
    override protected void Render()
    {
        Circle[] myCircleArray = new Circle[] { new Circle(center, 256, 0) };
```

```
        DrawCircles(myCircleArray);
    }
}
```

Look at your finished sprite

Finally, go back to Unity and hit play to look at the finished product. If your object doesn't appear, make sure it is in the camera's field of view.

Next Steps

Move things around

To actually get the ProSprite moving, go into the script again and externalize the radius variable by declaring a public int `radius`, setting it to 256, and plugging it into the circle constructor in myCircleArray. You should end up with this:

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

public class MyScript : ProSprite
{
    public int radius = 256;

    // Start is called before the first frame update
    void Start()
    {
        Setup(32, 32);
    }

    // Render is called before the camera renders a frame
    override protected void Render()
    {
        Circle[] myCircleArray = new Circle[] { new Circle(center, radius, 0) };
        DrawCircles(myCircleArray);
    }
}
```

Return to unity, click play, select the object with your script, and change the radius property of the circle. You should notice the size of the circle change. Radius values that are not perfect squares will draw circles that appear more or less round. experiment with different values between 225 and 256 (15 and 16 squared) to see this effect.

Change the palette

To change the color pallete that ProSprite takes from, go into the Project Settings and select "ProSprite Pallete." From there, you can change the colors available. *These settings will not effect the game during runtime, and will only take effect after you leave the settings menu and re-compile.*

Look more into the demos

The demos will show case other features, like Stroke(), Curve(), and more.

Edit this page