Augmented Reality – introductory tutorial

Part 1: Setting up your AR scene

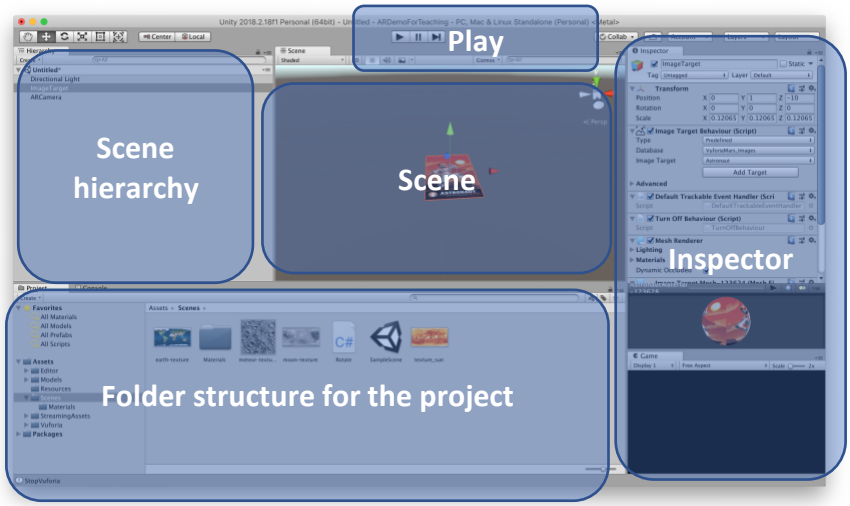


Figure 0. This figure shows an overview of the Unity developer environment.

- A video guide for setting up an AR environment in Unity
- Vuforia's guide to getting started with AR in Unity
- Unity's guide to registering as Vuforia developer

Quick start

1) Enable AR via File -> Build Settings -> Player Settings. It might be necessary to download and install Vuforia Support. In that case, you will need to restart Unity (see Figure 1).
2) Remove Main Camera from the scene
3) Add AR camera via GameObject -> Vuforia -> AR Camera
4) Add image target via GameObject -> Vuforia -> Image

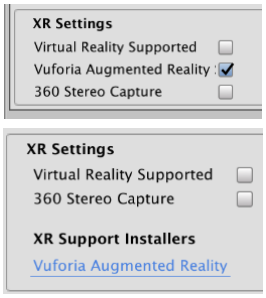
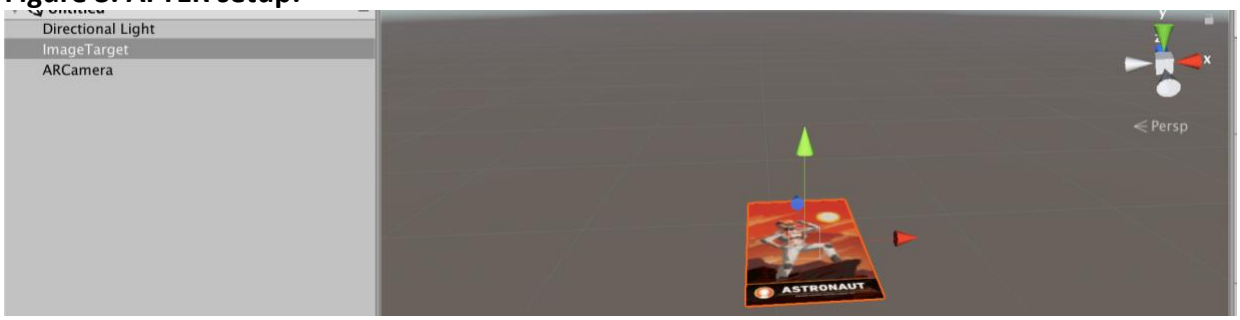


Figure 1. Enable AR

Figure 2. BEFORE setup:

Figure 3. AFTER setup:



Vuforia Configurations can be found via Window -> Vuforia Configuration. Here, you add your license, choose how many targets can simultaneously be tracked, or which camera hardware to use when running the Scene in the editor (e.g., built-in or an external camera)

Part 2: Your first simple AR app

Before you start, print this sheet including four standard image targets.
In this tutorial, we will use the image targets called ‘Astronaut’ and ‘Oxygen’.

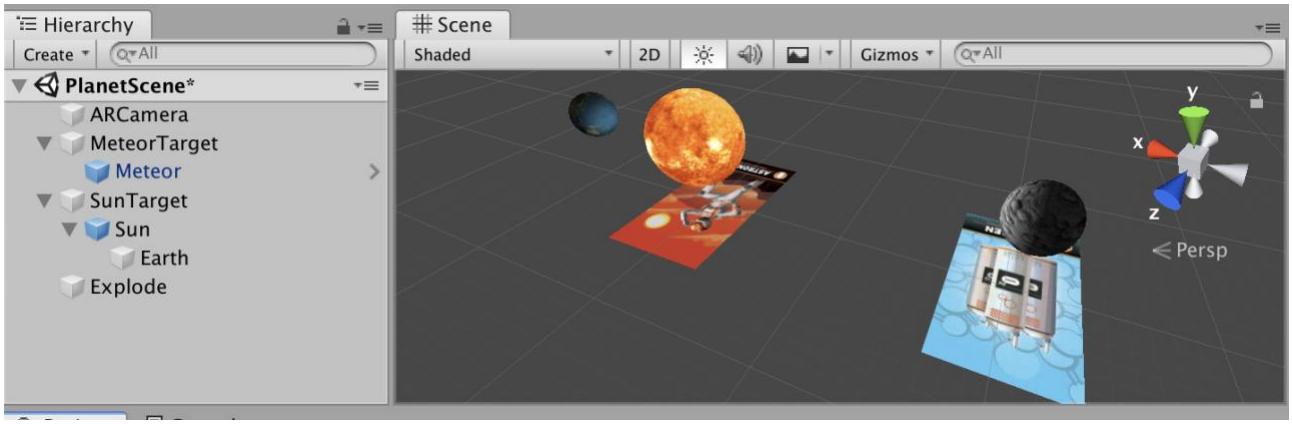


Figure 4. This figure shows how the scene (right) and scene hierarchy (left) should look in the end.

1) Create a Sun object showing on an image target
    a. Rename ‘ImageTarget’ to ‘SunTarget’
    b. Add a sphere (via GameObject -> 3D Object -> Sphere) and name it ‘Sun’.
    c. Make ’Sun’ child of ‘SunTarget’. Making an object ’child’ of another refers to nesting it in the hierarchy (see Figure 4).
    d. Hit play button to see if the Sun object appears in the camera frame when you point your real camera on the printed image target. Yay! First AR check point!

2) Create realistic Sun material
    a. Here are textures for Earth, Moon, Meteor and Sun.
    b. Add a texture to the material for the Sun object via drag-n-drop of texture file into the scene (first drag file into the Assets folder, then drag from this folder into the scene).
    c. Change from the “Standard” shader to use the ”Unlit/Texture” shader. This will provide a good simulation of the Sun (which primarily emits light, and has no visible reflection).
    d. Setup lighting in the scene to only have light coming from the Sun
        i. Remove the standard ”Directional Light” object in the scene.
        ii. Add a point light component to the Sun sphere: Select the Sun object. In the inspector, click the button ‘Add Component’. Search for ‘Light’ and click it to add as component to the Sun. It will be Point Light as default.

3) Create realistic Earth material
    a. Drag-n-drop the earth texture to the Sphere.
    b. Using ‘Standard’ shader, experiment with manipulation of light on the object, e.g., reflection (Albedo) and emitted light from the object itself (Emission). Observe how this affects the object.

4) Make Sun and Earth object rotate
    a. Create a script by selecting Sun object and clicking “Add Component” and click on “New Script”. Name the script ”Rotate”.
    b. Open the script file in your preferred editor (Visual Studio has good integration with Unity) and add a line of code in the Update() function corresponding to Figure 5.
    c. Experiment with different values for the 3 axes and observe the effect.
    d. Add yet another Sphere as child of ‘Sun’ called ’Earth’.
    e. Attach the same script to the Earth object, so that both Sun and Earth rotate.

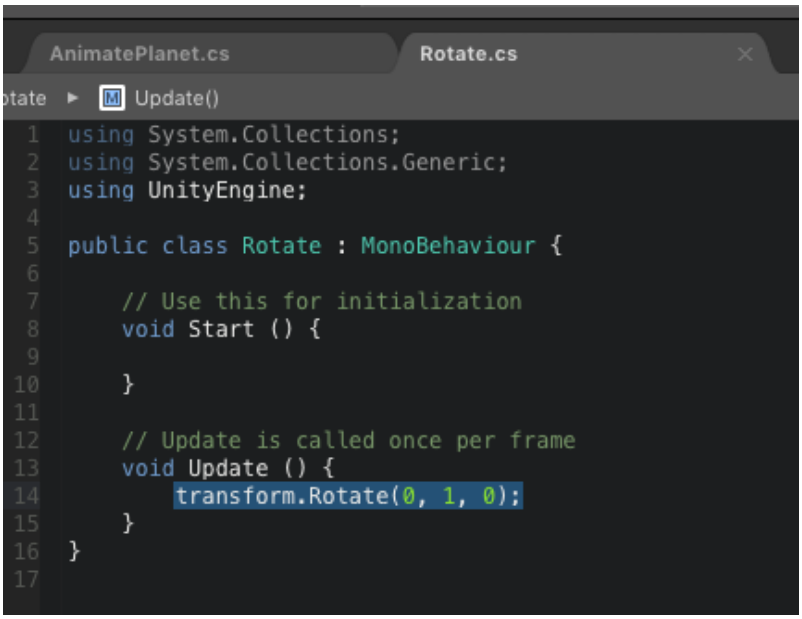


Figure 5. This line of code in the script’s update function makes the Sphere object rotate around itself on the Y axis.

1) Create a Meteor target
    a. Add another image target (named ’MeteorTarget’) and Sphere (named ‘Meteor’) as child. Style the Meteor like Earth, but using the meteor texture from the link.
    b. In the Inspector – change the ’Image Target’ field from ’Astronaut’ to ’Drone’ in order for the Vuforia system to detect your other printed image.

2) Remove ambient environment light from the scene
    a. Go to Window -> Rendering -> Lighting Settings.
    b. Set it up like in Figure 6.

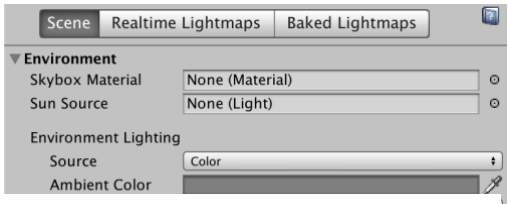


Figure 6. Remove ambient lighting

3) Play the scene again and see the finished result. Move the printouts corresponding to ’SunTarget’ and ’MeteorTarget’ in relation to each other. Notice how the light emitted from the Sun reflects dynamically from the Earth and Meteor, depending on how they are positioned in relation to the Sun.

4) Pat yourself on the back! You just made your first AR application ;)

Part 3: Your first AR exercise

Your first exercise is to implement an explosion when earth and the meteor collide. In this exercise, it is important that the collision happens in the exact intersection point between the edges of the two spheres. The result should look like in Figure 7.
This exercise is split into three smaller exercises (A, B and C), described on the next pages.
Tip: In this exercise, you will find the documentation for C# scripting useful. The illustration in Figure 8 provides some hints on how to solve the tasks.

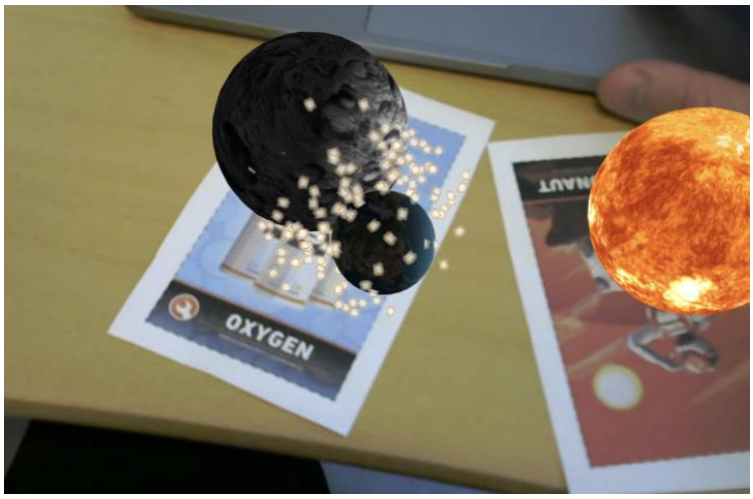


Figure 7. The explosion is a particle system that should be triggered in the position of the collision point as illustrated in the figure.

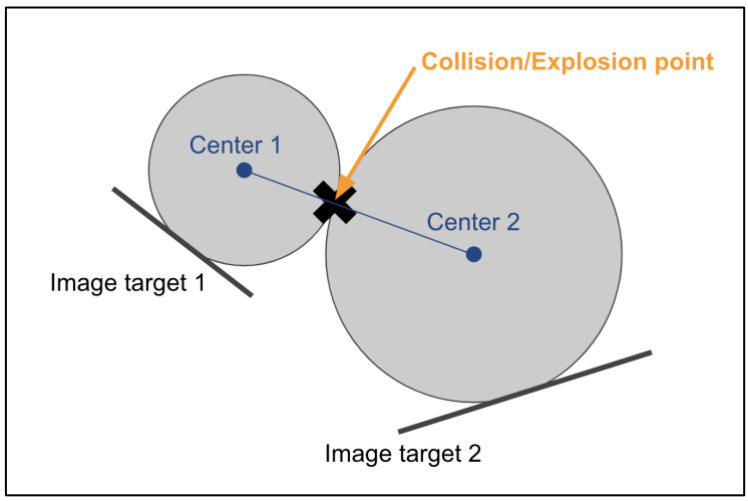


Figure 8. This figure illustrates that the collision point is the point along the vector between center 1 and 2 that indicates where the edges of the spheres meet.

Exercise A) Detect whether the spheres collide
Figure 9 shows what you need to set up before getting started.

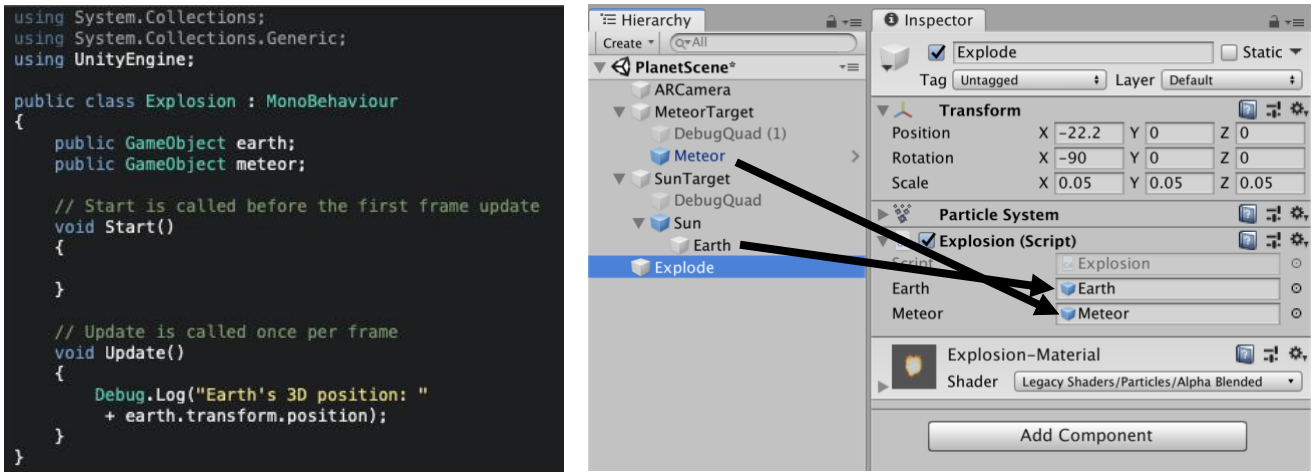


Figure 9. Create an empty game object (GameObject -> Create Empty) and name it “Explode”. Attach to it a new script called “Explosion”. Modify the new script to have the above code. Then drag and drop the sphere objects (earth and meteor) from the hierarchy into the script like above.

Now, all the information you need in your script in order to calculate whether the two spheres collide is the **position** and the **radius** of the spheres (of earth and meteor).
You can access the **position** of earth in world space via `earth.transform.position`.
You can access the **radius** of the sphere by accessing `earth.transform.lossyScale` (its diameter in world space) and dividing by 2.

You first have to calculate the distance between the two center points using:
`Vector3.Distance(Vector3 a, Vector3 b)`
(Just google “Vector3 Distance Unity” and you will find the documentation. Or you can use the link above to the scripting documentation on Unity, which also has a search bar.)

You then have to calculate the minimum distance between the two spheres using their radiuses.

You can then compare these values to see if the distance between spheres is below their minimum distance using the `<` (“less than”) operator.

You can print the result to the console using something like the following:

```
bool isColliding = distance < minDistance;
Debug.Log(“Do earth and meteor collide?” + isColliding);
```

Exercise B) Calculate the exact collision point

In this final exercise, you need to calculate the position for which the explosion should trigger.

To more easily see if you have done this correct, you can do the following:
Delete the empty object from exercise A. Use a basic sphere object (GameObject -> 3D Object -> Sphere) and attach the Explosion script to this object instead. Call the GameObject "DebugSphere".

Now you can calculate the unit (normalized) vector pointing from earth to meteor (hint: use `Vector3.Normalize`). The two lines of code will have the following form:

```
Vector3 fromTo = Vector3.Normalize(meteorPosition - earthPosition);
Vector3 collisionPoint = earthPosition + earthRadius * fromTo;
transform.position = collisionpoint;
```

When you run the app, you should see the DebugSphere appear in the collision point.

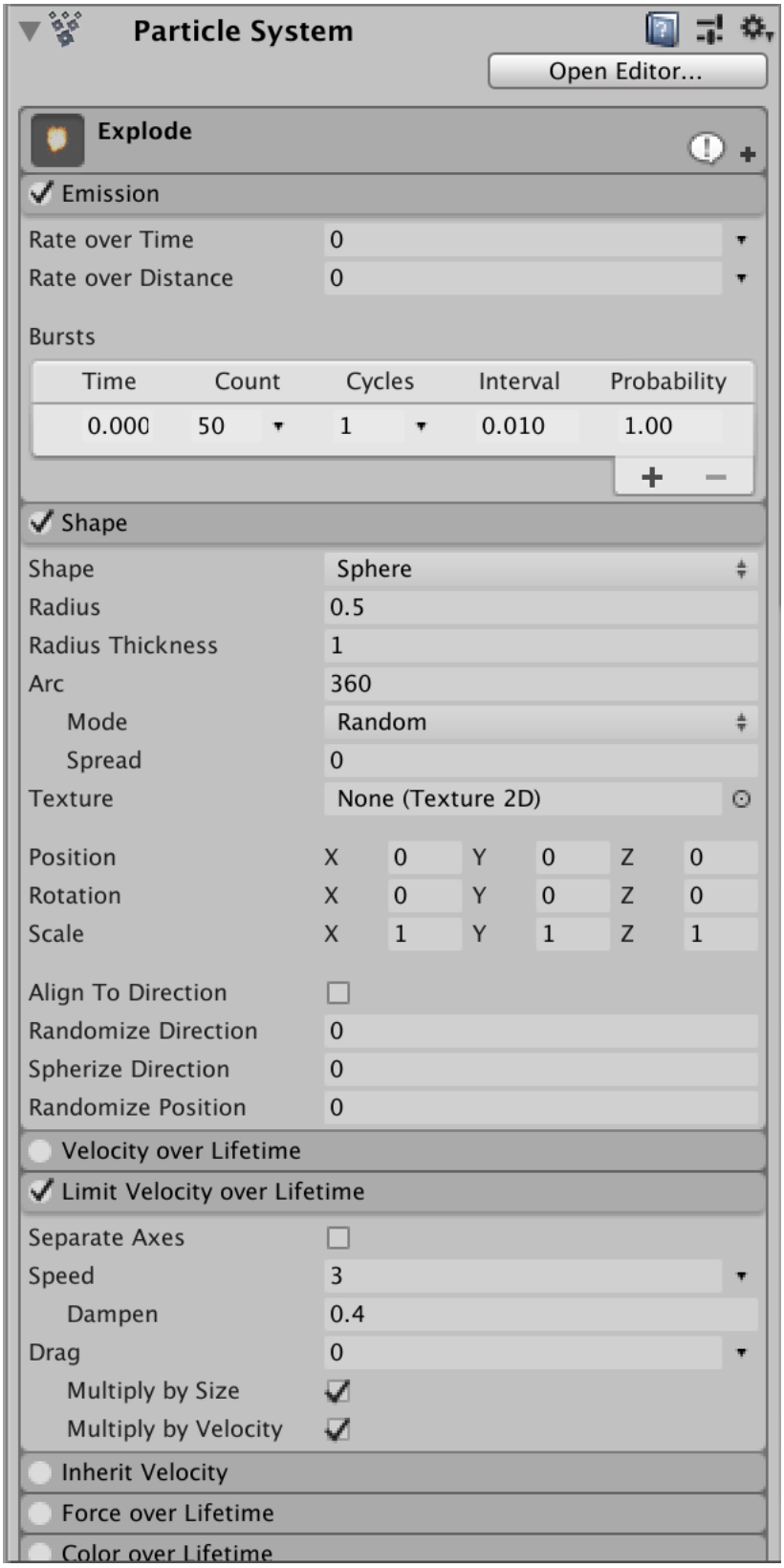
Exercise C) Trigger an explosion with a particle system

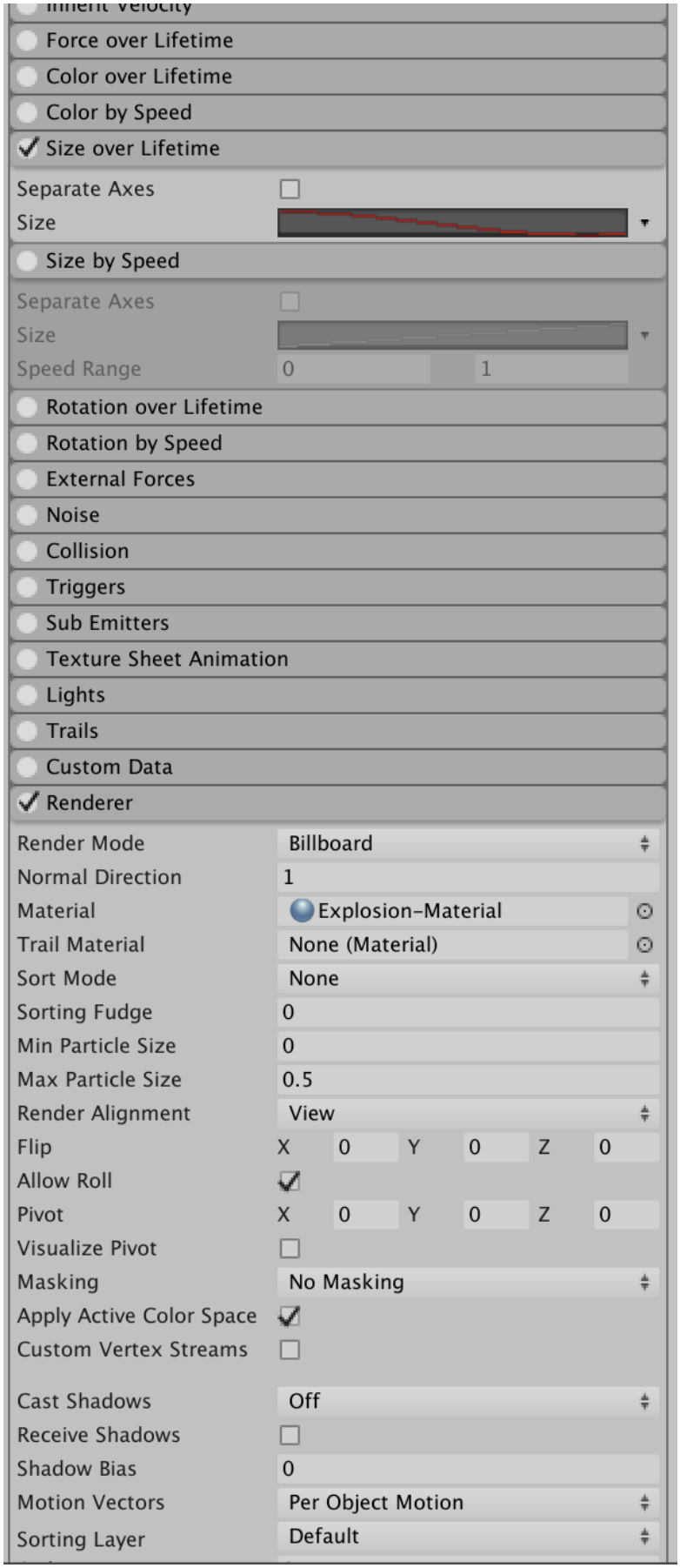
You have now calculated *whether* the two spheres collide, and *at which position* they collide. You just need a bit more Unity magic to make it look really cool!

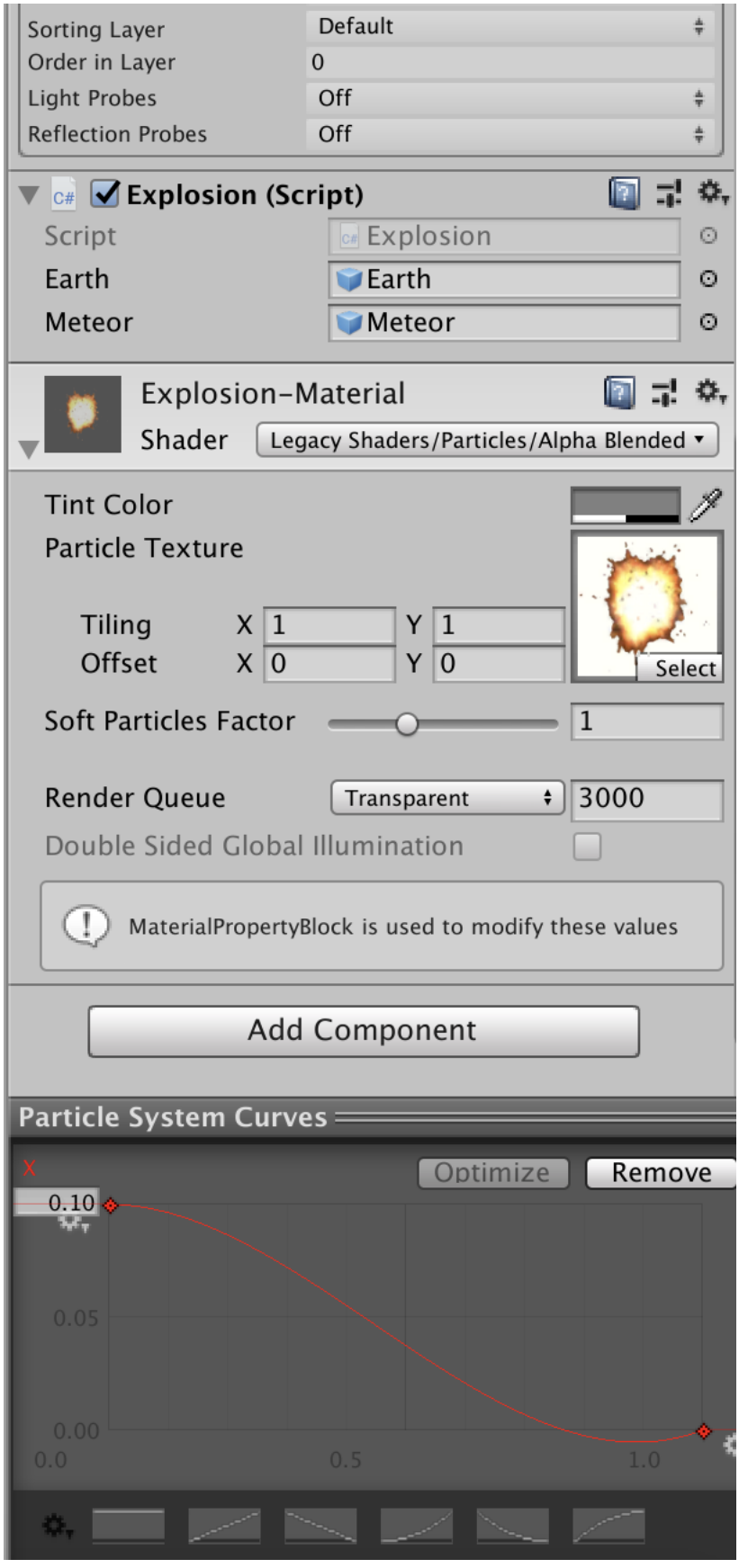
Instead of the "debugging" sphere, you can trigger an explosion when the meteor hits earth. Here is a simple tutorial for triggering a particle system explosion.

Figure 10. On the last three pages, you will see the panel of a working configuration for an explosion particle system could look. (you see it as a mini-overview to the right.

In the end, you need to create a material that looks like "Explosion-Material" (as seen in the very bottom of this panel). Attached to the Material, you need an explosion texture. Here is a link to one for download.


Particle System
Open Editor...
Explode
Emission
Rate over Time 0
Rate over Distance 0
Bursts
Time Count Cycles Interval Probability
0.000 50 1 0.010 1.00
Shape
Shape Sphere
Radius 0.5
Radius Thickness 1
Arc 360
Mode Random
Spread 0
Texture None (Texture 2D)
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Align To Direction
Randomize Direction 0
Spherize Direction 0
Randomize Position 0
Velocity over Lifetime
Limit Velocity over Lifetime
Separate Axes
Speed 3
Dampen 0.4
Drag 0
Multiply by Size
Multiply by Velocity
Inherit Velocity
Force over Lifetime
Color over Lifetime


Inherit Velocity
Force over Lifetime
Color over Lifetime
Color by Speed
Size over Lifetime
Separate Axes
Size
Size by Speed
Separate Axes
Size
Speed Range
0
1
Rotation over Lifetime
Rotation by Speed
External Forces
Noise
Collision
Triggers
Sub Emitters
Texture Sheet Animation
Lights
Trails
Custom Data
Renderer
Render Mode
Billboard
Normal Direction
1
Material
Explosion-Material
Trail Material
None (Material)
Sort Mode
None
Sorting Fudge
0
Min Particle Size
0
Max Particle Size
0.5
Render Alignment
View
Flip
X 0 Y 0 Z 0
Allow Roll
Pivot
X 0 Y 0 Z 0
Visualize Pivot
Masking
No Masking
Apply Active Color Space
Custom Vertex Streams
Cast Shadows
Off
Receive Shadows
Shadow Bias
0
Motion Vectors
Per Object Motion
Sorting Layer
Default


Sorting Layer
Default
Order in Layer
0
Light Probes
Off
Reflection Probes
Off
Explosion (Script)
Script
Explosion
Earth
Earth
Meteor
Meteor
Explosion-Material
Shader
Legacy Shaders/Particles/Alpha Blended
Tint Color
Particle Texture
Tiling
X 1
Y 1
Offset
X 0
Y 0
Select
Soft Particles Factor
1
Render Queue
Transparent
3000
Double Sided Global Illumination
MaterialPropertyBlock is used to modify these values
Add Component
Particle System Curves
X
Optimize
Remove
0.10
0.05
0.00
0.0
0.5
1.0