Cone Cast Script Documentation

February 2025

version 1.0.0

Contents

1 About

Cone Cast Script is a simple asset that lets you *cast a cone* that is approximated with raycasts (and optional sphere casts) in a cone shape. Use it in your projects as you see fit.

2 Dependencies

The code itself has no dependencies.

The **demo scene** needs shader graph only to be able to present on all render pipelines with a single material.

3 Details

3.1 How it works

The cone cast works as follows:

1. a raycast forward is cast to see if there is anything right ahead of the origin of the cone
2. depending on *subvidision*, a number of rings are formed, along whom lines are shot, forming a circle
3. the origin of the rings can be offset with *near clip distance*
4. after all the lines and optional spheres are shot, the `RaycastHit`s are collected
5. `ConeCast` returns the closest target out of all that was hit
6. `ConeCastAll` returns all targets that were hit

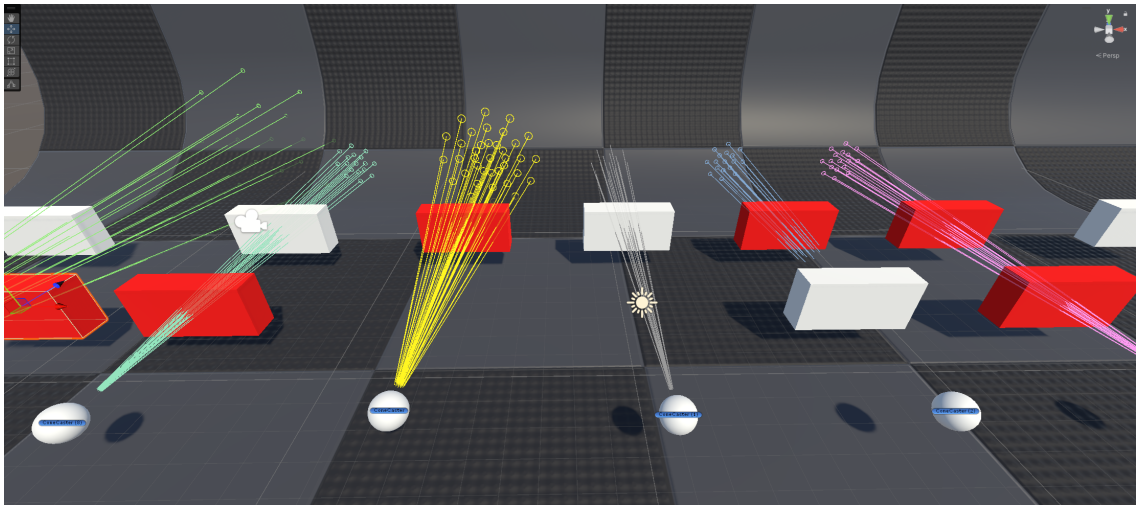

Figure 1: Cone Cast Example

3.2 Code example

`ConeCast` returns a bool that is true if it hit anything, or false if it didn't. The `RaycastHit` is set in a similar fashion to Unity's API.
`ConeCastAll` returns all `RaycastHit`s, or null if nothing was hit.

```
public class Example : MonoBehaviour {
    public void BlockingConeCast() {
        bool isAnyHit = ConePhysics.ConeCast(
            hit: out var hit,
            origin: transform.position,
            direction: transform.forward,
            coneAngle: coneAngle,
            subdivision: subdivision,
            nearClipDistance: nearClipDistance,
            farClipDistance: farClipDistance,
            layerMask: layers,
            useExtraSpheres: useExtraSpheres,
            extraSphereRadius: extraSphereRadius,
            queryTriggerInteraction: QueryTriggerInteraction.Collide,
            visualize: visualize,
            drawColor: drawColor);
    }

    public void PassThroughConeCast() {
        RaycastHit[] results = ConePhysics.ConeCastAll(
            origin: transform.position,
            direction: transform.forward,
            coneAngle: coneAngle,
            subdivision: subdivision,
            nearClipDistance: nearClipDistance,
            farClipDistance: farClipDistance,
            layerMask: layers,
            useExtraSpheres: useExtraSpheres,
            extraSphereRadius: extraSphereRadius,
            queryTriggerInteraction: QueryTriggerInteraction.Collide,
            visualize: visualize,
            drawColor: drawColor);
    }
}
```

3.3 Parameters

- hit is the `RaycastHit` result, if anything was hit.
- origin is the start point of the cone
- direction is the direction vector the cone is facing
- coneAngle is the angle, or spread, of the cone
- subdivision is how dense the lines are that make up the cone. if you increase this, the increase will be squared so make sure to use a number as low as you can get away with.
- nearClipDistance is an offset in the cone direction where the raycasts start
- farClipDistance is how far the lines shoot
- layerMask sets the layers that are considered

- useExtraSpheres enables shooting sphere casts on top of the lines. this makes the cone less precise but more optimal as less lines can be enough
- extraSphereRadius is the radius of those extra spheres
- queryTriggerInteraction sets how the lines should behave with trigger colliders
- visualize enables debug lines to draw (and a circle at the end if spheres are enabled)
- drawColor is the color of the visualization