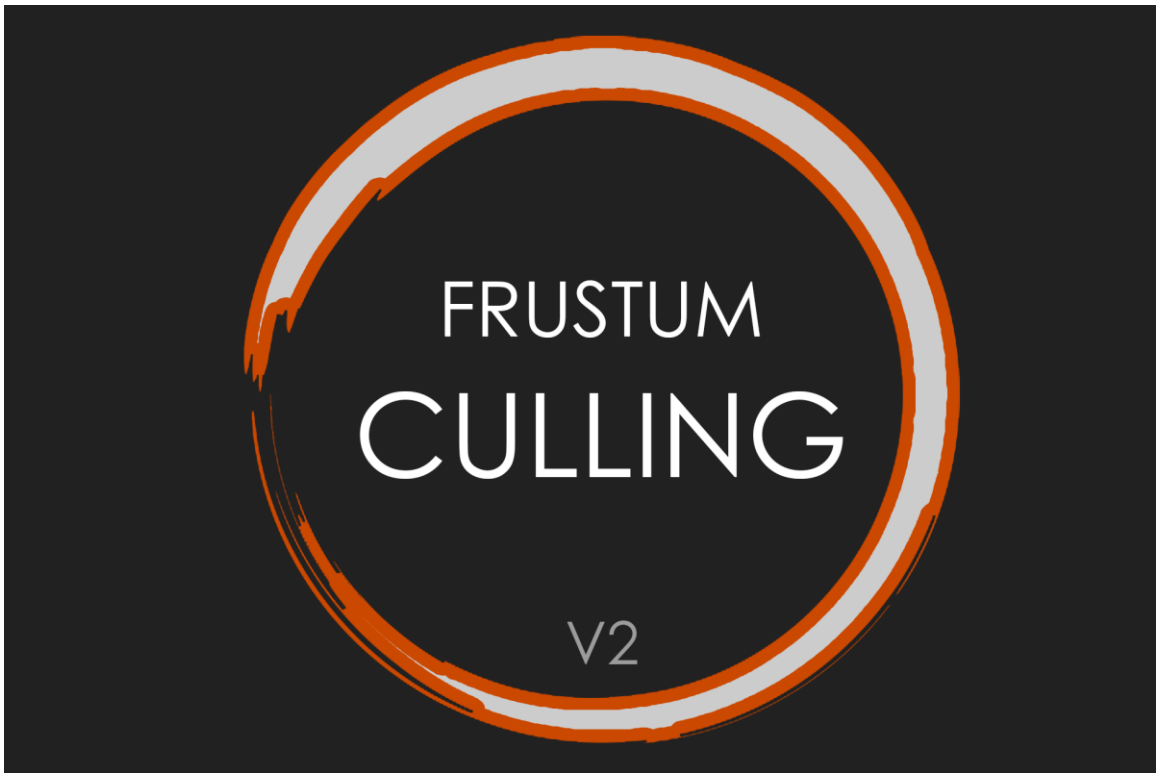


FRUSTUM CULLING DOCUMENTATION

For support

Feel free to email me at ***pathiralgames@gmail.com***

or message me in ***Discord***

Find more packages from Pathiral here

ABOUT

Improve performance by disabling game objects completely that're out of the camera's view (frustum) or distance.

P.S: Unity already does *Occlusion Culling* for static objects but it disables the renderers only. You'll need a custom solution to completely disable the object. This is where this solution comes in. Extremely helpful to disable objects that are very costly to have on but you need them enabled when in view. Things like an AI standing still and playing animations, small volumetric lights, objects with highly intensive scripts, etc…

The rule of thumb is:

- use Unity's default occlusion culling with static gameobjects
- use Frustum Culling with dynamic gameobjects
- use Frustum Culling for static gameobjects with many components and scripts with *Update(), etc…*
- use Frustum Culling with particle systems

GETTING STARTED : *(Tutorial Video)*

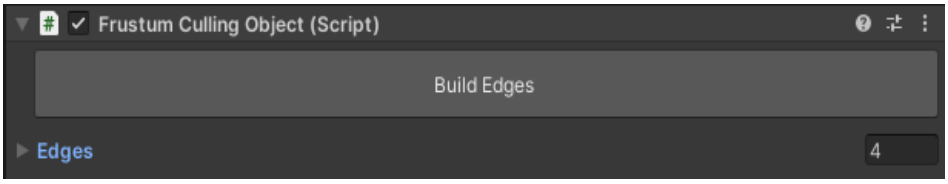
1. Create an empty game object in your scene and add the FrustumCulling component.

2. Leave the default values of FrustumCulling properties as is.

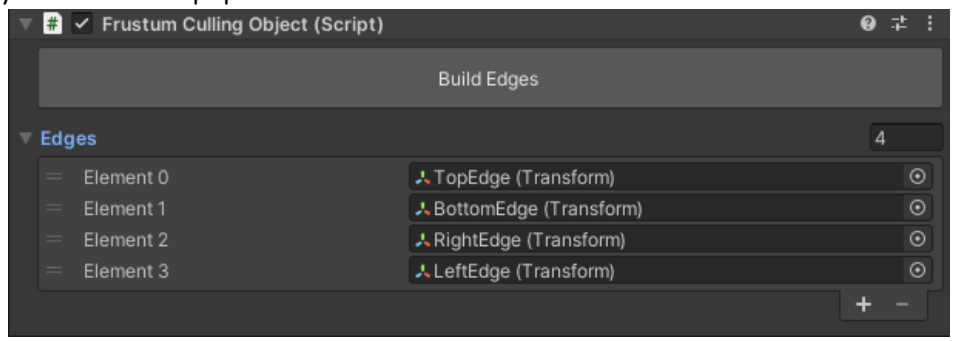
3. Create a Cube and place it in front of the game camera. This will be the object we'll test on. When out of view, we'll make this cube disable.

4. On that cube add the component FrustumCullingObject. Any object we want to cull, we need to add to it this script **WHERE THE MESH RENDERER & MESH FILTER IS**. This is very important.

5. After adding the script click on Build Edges button:



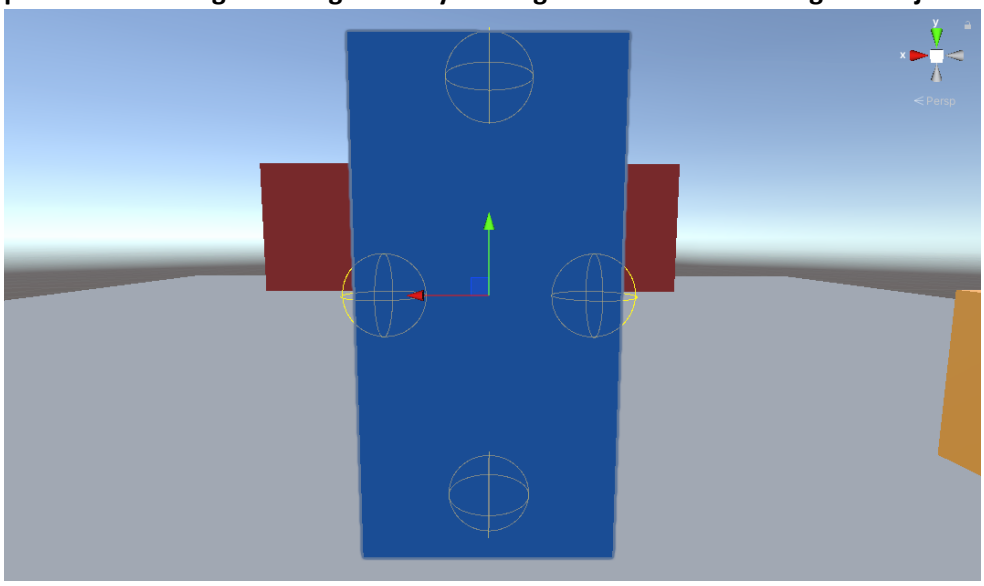
6. After clicking on the button you'll notice 2 things have happened. First thing is the Edges array has now been populated as so:



7. Second thing you should have noticed is that the hierarchal structure of the cube has been changed. It is now a parent to 4 other game objects. These 4 game objects represent edges. Each one is an edge.

8. Selecting the parent and opening the scene view, you'll find the generated edges have been placed as yellow spheres on each side of the cube. **Be careful though, automatic placement of edges aren't always accurate and sometimes you'll have to manually**

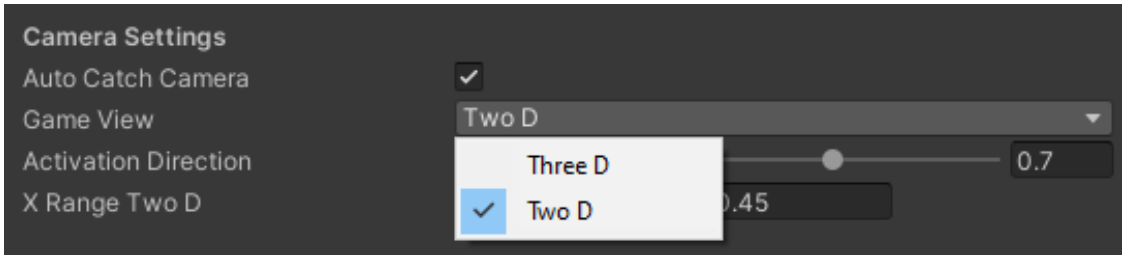
place them after generating them by moving each individual child game object.



9. You have now set the edges to your cube. Now start the game and move your camera away from the object. Keep your eye on the hierarchy or open the scene view side-by-side. You'll see the cube gets disabled when it's out of camera view.

10. That's it!

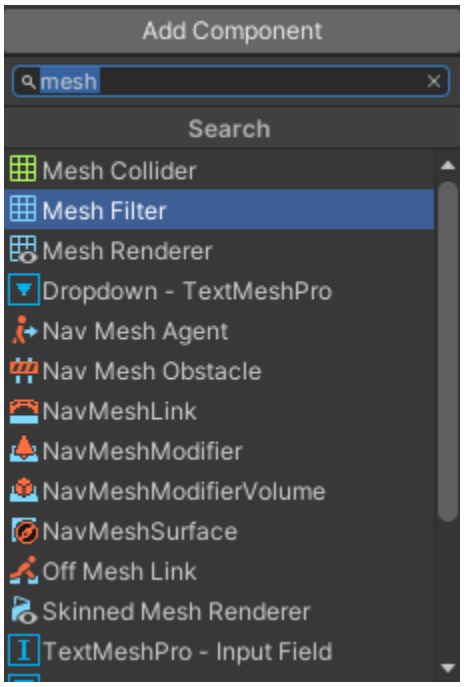
If your game is 2D or 2.5D then set **Game View** property to **Two D**. If it's first or third person then naturally keep it at **Three D**.



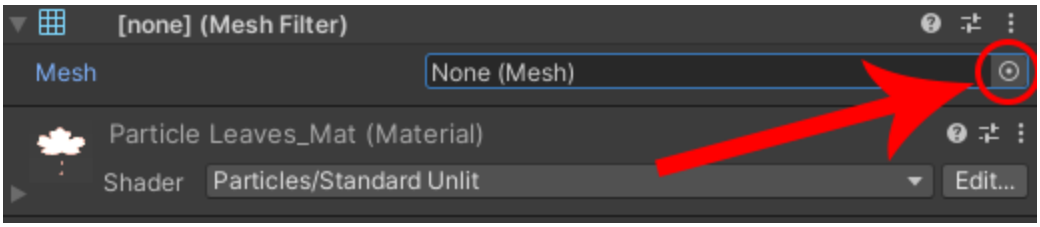
Using with Particle Systems

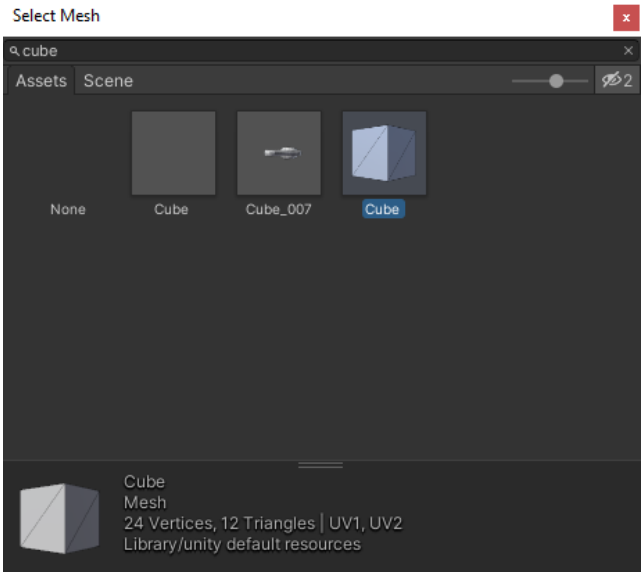
Probably Frustum Culling won't work as is with your particle systems. Luckily, it's extremely easy to make it work!

Frustum Culling needs a Mesh filter component to work and build the edges. So first things first is adding a **Mesh Filter** component.



After adding the mesh filter, we need an actual mesh so we can build the edges. Simply, click on the circle on the right-hand side, search for cube and select the default cube.





From here on out so you can continue as is and add the Frustum Culling Object component, click on Build Edges, set the edges to encompass your entire particles radius and that's it! **There's nothing more you need to do with the cube mesh it's just a quick way for building the edges.**

Properties

Auto Catch Camera : on *Start()*, the system will automatically get the main camera to use for culling. The camera needs to have the tag ***MainCamera*** in order to be found.

Main Cam : If auto catch camera is set to false, you can manually set the camera you want for culling in this property.

Game View : This changes the culling calculations based on the game view. If your game is 2D or 2.5D then set it to **Two D**. If it's first or third person then naturally keep it at **Three D**.

Activation Direction : this is the angle to any edge that when met the culling will occur or object gets enabled. As a rule of thumb, this only be played with if you're having problems with culling (objects activating at incorrect or late angles). Otherwise, leave it alone.

X Range Two D : This property sets the range on the X axis each built edge needs to be in order to cull or enable. It's better to leave this property alone unless you're having problems then you can tweak. The **X** should take a negative value while the **Y** a positive value.

Run Every Frames : the amount of frames to pass before running a pass of frustum culling. The higher the number, the more performant but less accurate. The lower the number, the more accurate but may affect performance.

Resting Frames : The amount of frames to rest before jumping to the next object in the loop. This helps spread the looping functionality across several frames, drastically improving performance.

Cull In Scene : If set to true, culling will appear in the scene view. Meaning you can see the culling normally as it happens in the scene. But having this enabled may have a

minor impact on accuracy. On the other hand, if set to false, the game objects will not be culled **if they're visible in the scene view** and culling will only appear if the game view **only** is selected and you will notice the culling by the dimming (disabling) of the game objects in the hierarchy window. **In game build, this isn't taken into account so you have nothing to worry about whether it's true or false. It won't be considered.**

Disable Root Object : If enabled, the parent game object and subsequently all of it's children will be disabled when culled. If disabled, only the child with the edges will be disabled when culled.

Distance Culling : If enabled, distance culling will be taken into account. Meaning the system won't only cull when objects are out of view but also when they're out of distance. Objects that are to be culled by distance won't be culled unless they get out of view first.

Distance To Cull : The distance which when exceeded the game object will be culled.

Prioritize Distance Culling : First let's understand how distance culling works in it's default state. *Objects that are to be culled by distance won't be culled unless they get out of view first.* If this is set to true, then the object will be culled even if it's not out of view yet. Same thing in it's re-appearance. This means game object pop ups in your game.

Distance Culling Only : As the name insinuates. If enabled, no frustum culling calculations will take place but rather, the culling will be made according to distance only.

PUBLIC APIs

These are public methods you can call programmatically.

[FrustumCulling component]

Add(GameObject) : This is automatically called by the FrustumCullingObject script on *Start()* to add the current game object to the list of objects to cull by the system. If for some reason you need to call it manually, there you go. *The FrustumCullingObject needs to be already added to the passed game object.*

Used:

```
FrustumCulling fcScript = GetComponent<FrustumCulling>();

fcScript.Add(gameObject);
```

Remove(GameObject) : If you would like for some reason to remove an object from the list of objects to cull, you can call this.

Used:

```
FrustumCulling fcScript = GetComponent<FrustumCulling>();

fcScript.Remove(gameObject);
```

[FrustumCullingObject component]

BuildEdges() : Call this method to build the edges dynamically

GetEdges() : Will return an **array of type Transform** of the 4 edges.

CheckIfEdgesBuilt() : Returns a bool to whether the edges have been built or not.