Force Field

Version 2.0

http://www.forge3d.com
support@forge3d.com

Contents

Introduction

Thank you for purchasing Force Field!

This guide describes the features of the Force Field integration in Unity3D. A basic understanding of the Unity3D engine, as well as C# programming language is assumed. Having basic knowledge of the shader and visual effects design in Unity3D may be advantageous.

For more information please visit www.forge3d.com

If you have any questions, suggestions, comments or feature request please do not hesitate to contact us at support@forge3d.com

Brief overview

Force Field is a shader based solution that allows you to render protective shield visual effect on top of any complex mesh surface. The communication between your game and the Force Field shader is done through the Force Field Controller script component. To display the effect the controller component requires you to supply field surface coordinate for each new impact by providing world space coordinates to the public OnHit method. It is possible to provide hit spot size and initial alpha value as a second and third parameter. It is also required to have a Unity3D Collider sharing the exact same mesh as the field surface used by the Mesh Renderer. Rigid body collisions can also be used to supply impact coordinates through the Unity3D events passing into Force Field Controller script. Please note that collision events are only sent if one of the colliders also has a non-kinematic rigidbody attached.

Differences between Desktop and Mobile version

Force Field ships with separate shader and script best suited for Mobile platforms. Due to platform limitations the Mobile version has a different set of features.

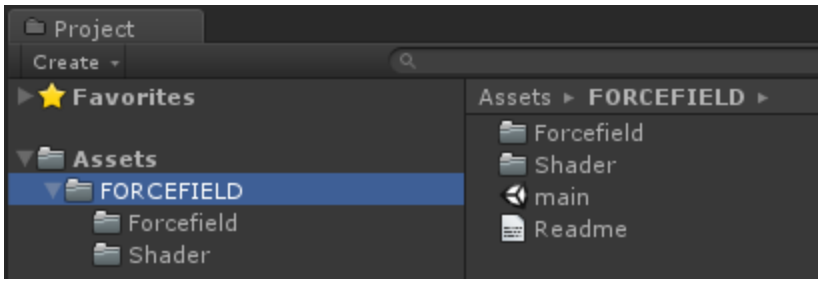
Mobile limitations on Force Field Mobile include:

- Amount of simultaneous impact points limited to 6
- No impact spot size and alpha control from script
- Outer mask has been removed
- ‘Sparks’ texture effect has been removed
- Rigid body collisions limited to ‘OnCollisionEnter’ only

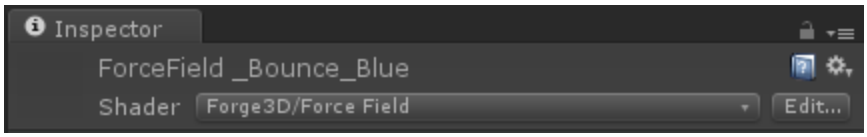
Upgrading from previous versions

Upgrading from previous version of Force Field is simple and done in four steps:

1. In Unity > File > New Scene
2. Remove old FORCEFIELD folder including all of its content

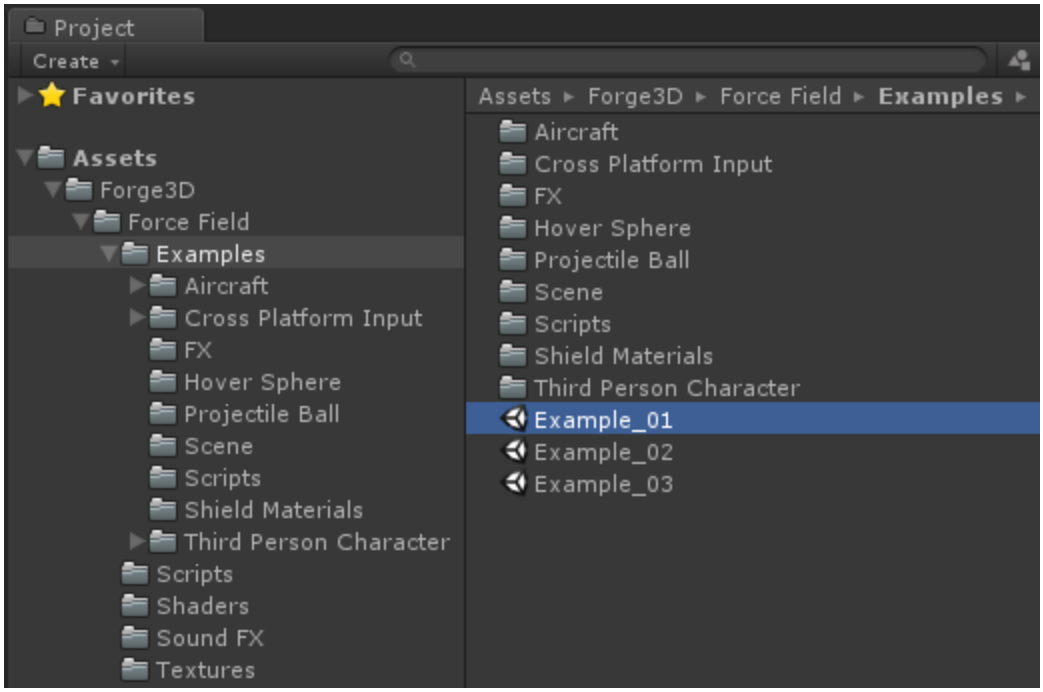


3. Import updated Force Field 2.0 Unity Package
4. Reassign your existing Force Field materials to use an updated shader



Getting started with the examples

The examples provide a good starting point for you. They are located in *Assets\Forge3D\Force Field\Examples* and the required components are already configured.



Example_01 - This shows basic setup and various Force Field types.

Example_02 - This scene is showing the example of complex mesh setup.

Example_03 - This example demonstrates Force Field rigid body collisions.

Example scene controls:

Left mouse button - Throw object

Right mouse button - Shoot

W S A D - Character move

Space - Character jump

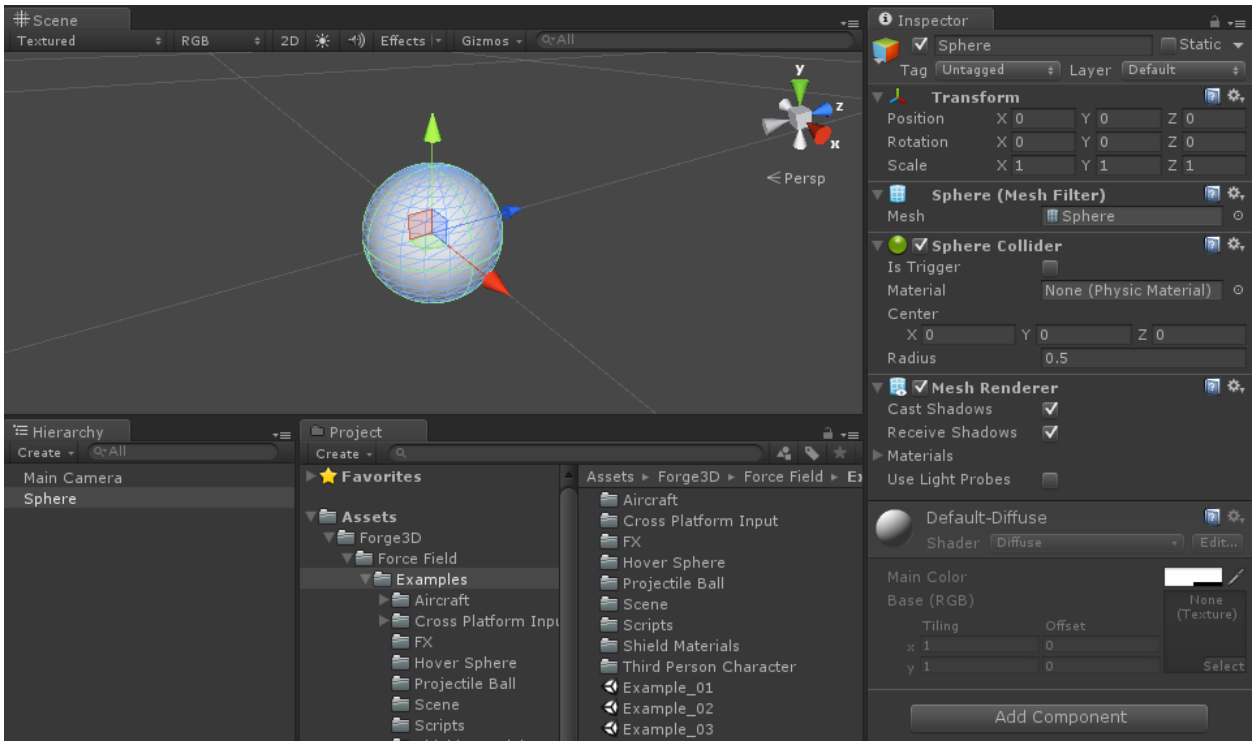
Force Field Basics

The easiest way to understand the basics is to create your first Force Field object. Lets get started with a new scene:

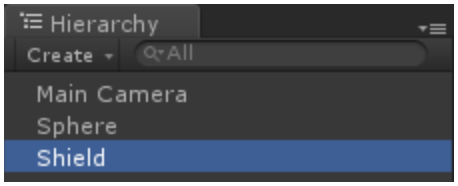
1. In Unity > File > New Scene

Now we need one object to display Force Field effect and another to render a surface it covers. Lets create a simple sphere:

2. Create new sphere game object using Unity menu GameObject -> Create Other -> Sphere and place it in front of camera.



3. Duplicate the Sphere game object using Edit -> Duplicate menu and rename it to 'Shield'. Select Sphere game object and remove Sphere Collider component.

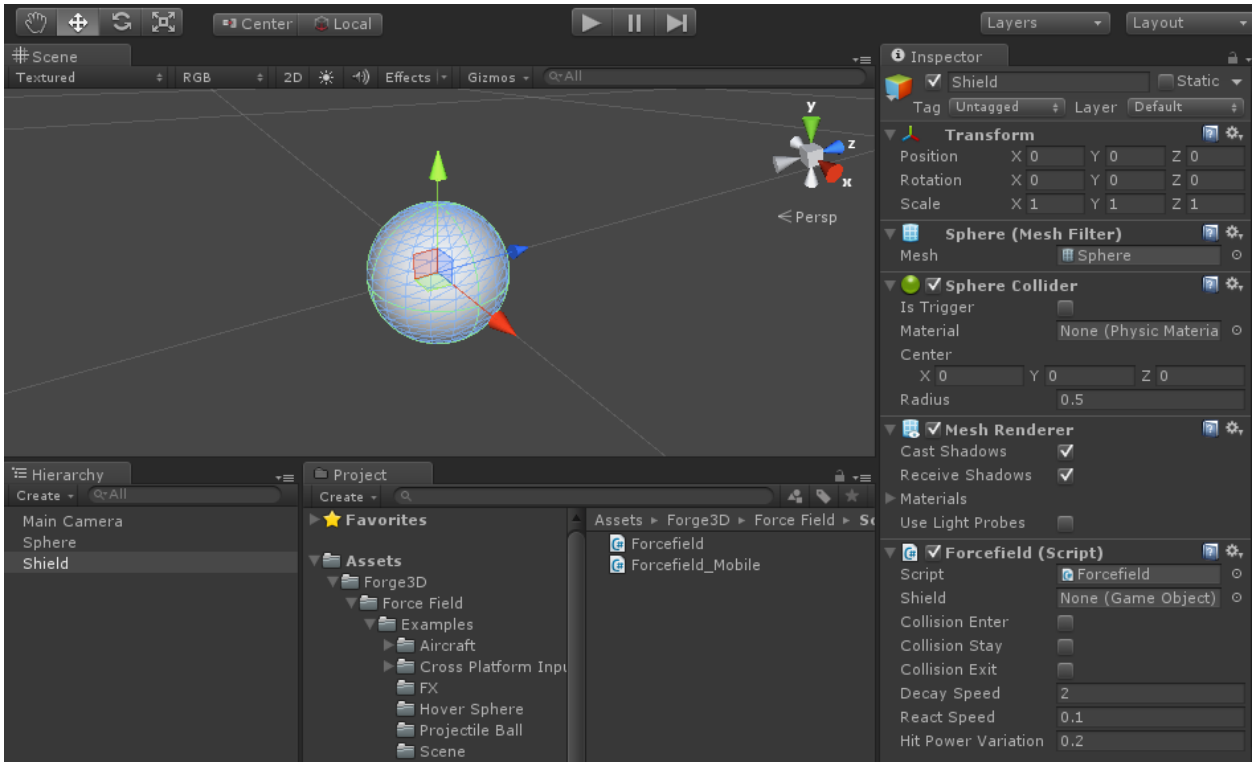


Later we will use 'Mesh Offset' in shader to push Shield outwards to prevent overlapping with the sphere.

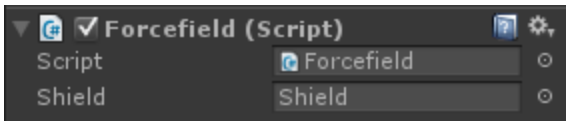
Next thing we require is a controller script. After importing Force Field package into your project, you can find it located in ***Assets\Forge3D\Force Field\Scripts*** folder.

Let's add *Force Field Controller* script to the *Shield* and point which game object will be used as a Force Field surface:

4. Add Forcefield.cs script to the Shield game object from Assets\Forge3D\Force Field\Scripts folder by dragging it on top.

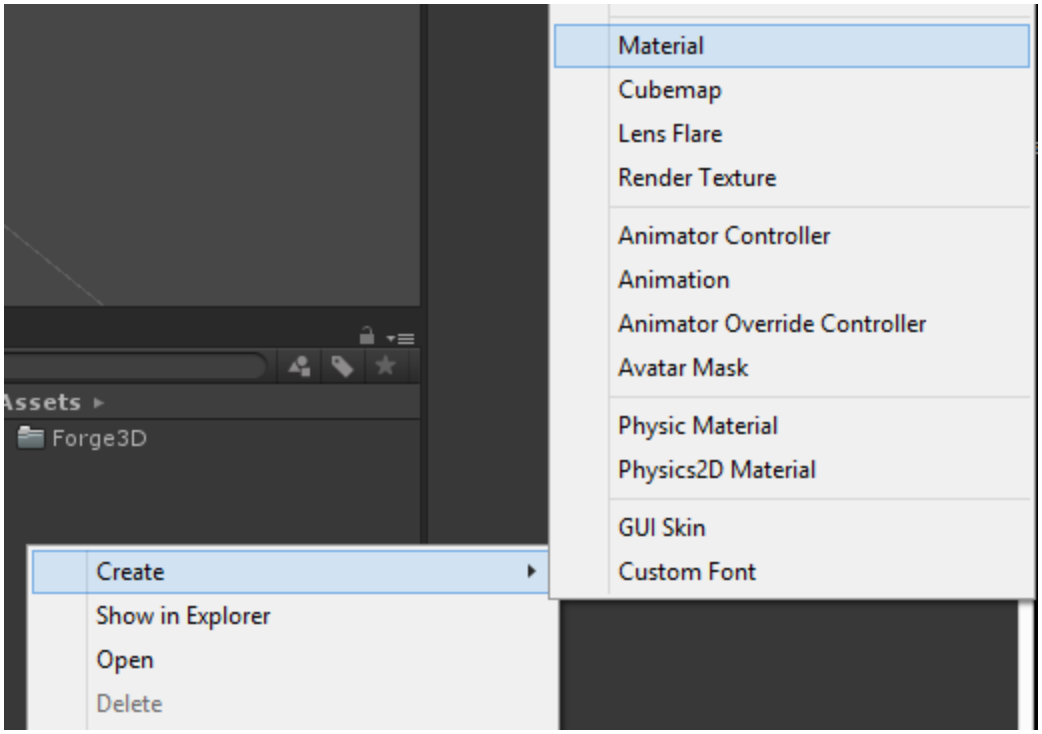


5. Select Shield game object in the Hierarchy panel and drag it on top of the empty 'Field' property in the Inspector panel.

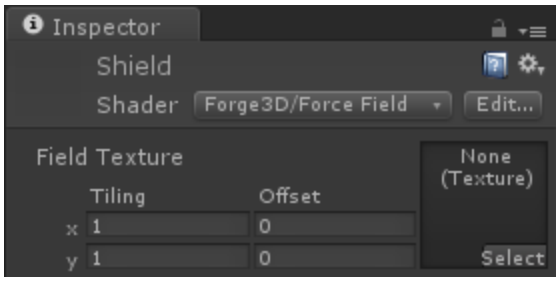


It's time to create a new material and assign it to the Shield. We will use special shader that will be rendering the effect:

6. Create new Material in Project panel. Right Click -> Create -> Material and name it 'Shield'.

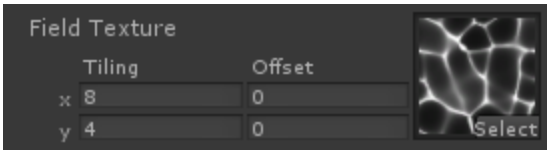


7. Select Shield material and assign Forge3D/Force Field shader in the drop down menu.



Force Field ships with a set of textures to help you create various kinds of shield effect. Let's assign one of the textures into the material:

8. Assign 'field_charge_003' texture to the 'Field Texture' property from Assets\Forge3D\Force Field\Textures and set texture tiling to 8 on X and 4 on Y.



8 to 4 tiling helps us fill the spherical mesh in a more efficient way. Different scale setup in your project may require you to assign different values.

Let's setup a color gradation to the effect and the alpha value which is used as a multiplier for both impact zones:

9. Set 'Inner Mask Tint' RGBA color values to (R: 130, G: 220, B: 255, A: 40)

10. Set 'Outer Mask Tint' RGBA color values to (R: 0, G: 150, B: 255, A: 40).

'Mesh Offset' is used to push Shield outwards along normals. Since we are using duplicated sphere we have to set a small positive value:

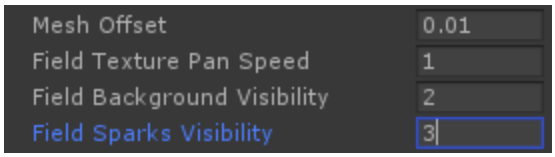
11. Set 'Mesh Offset' value to 0.01

Now it's time to set general texture visibility at the moment of impact:

12. Set 'Field Background Visibility' value to 2

'Sparks' are the special effect made by using texture mask with animated offset. Let's set it's visibility as well:

13. Set 'Field Sparks Visibility' value to 3



The inner mask is displayed at the point of impact and fades inwards at the speed controlled by *'Decay'* parameter from the Force Field Controller script. *Offset* sets the size of the initial mask. Setting bigger values will make mask appear smaller. *Feather* controls how smooth or sharp the mask should appear:

14. Set 'Inner Mask Offset' value to 8
15. Set 'Inner Mask Feather' value to 8

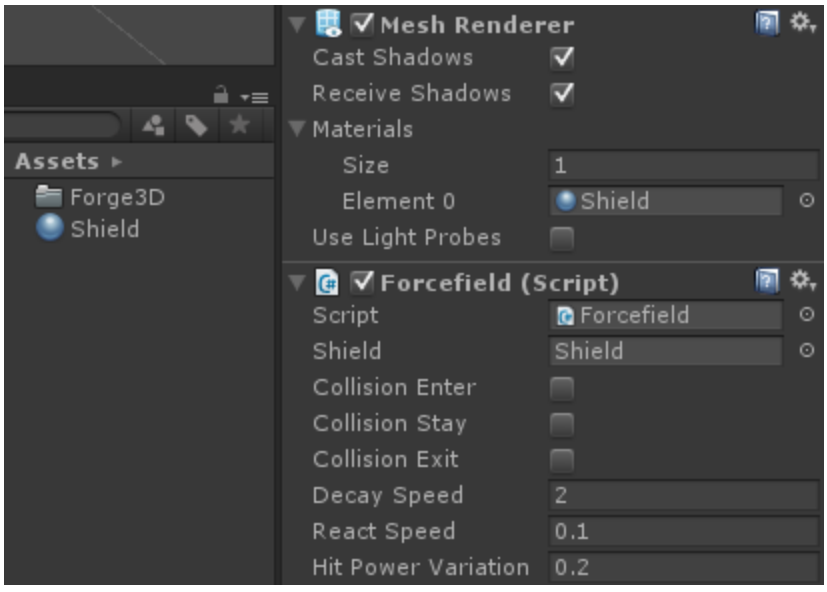
The outer mask is displayed at the impact point, however it fades outwards and appears more like a blast wave:

16. Set 'Outer Mask Offset' value to 4
17. Set 'Outer Mask Feather' value to 4

** Some corrections may be required depending on the scale setup in your project.*

Now we assign the material to the Shield Mesh Renderer:

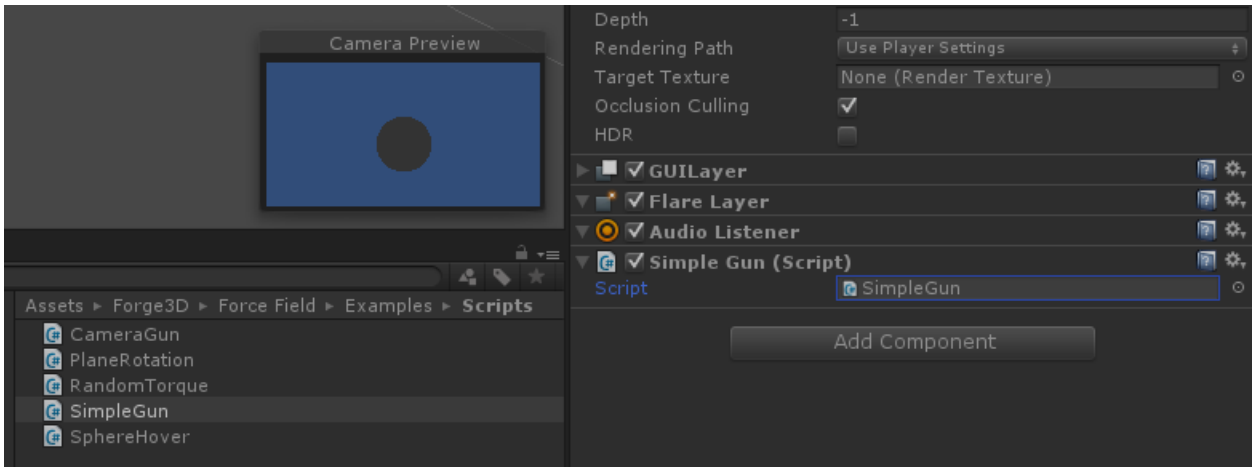
18. Assign Shield material to Shield game object replacing 'Default-Diffuse' in the Mesh Renderer.



Now that we have our object setup we need to provide it with some data to activate the field. The script we have attached earlier is responsible for getting this data and transmitting it into the shader.

To simplify things we will use a SimlpeGun script provided inside ***Assets\Forge3D\Force Field\Examples\Scripts*** folder. This script simulates a projectile traveling from camera through a screen point.

19. Drag SimpleGun script from Assets\Forge3D\Force Field\Examples\Scripts on top of the Main Camera.



Let's examine SimpleGun script in more detail. Each frame we will test for mouse click. If successful we will test for colliders along the ray going from the camera through a screen point. If any collider were hit during the test, the script will store collision details in a special RaycastHit structure. This will allow us to access collider's game object and its components. No matter what we hit we will try to use GetComponent and access Force Field script attached. Then we will check if it was successful and if it is the case we will send hit coordinates from the RaycastHit structure along with the random hit power into the Force Field.

Our key point of interest would be ***public void OnHit(Vector3 hitPoint, float hitPower = 0.0f, float hitAlpha = 1.0f)*** method.

Vector3 hitPoint - World space coordinates of the impact point required to render Force Field effect. Such coordinates can be generated by raycasting against the collider or by receiving collision data through rigid body collision events.

float hitPower **-** Impact spot size modifier. Use this value if you like to modify default impact spot size. ***hitPower*** is added to the '***Inner Mask Offset***' shader's parameter.

float hitAlpha **-** Is a starting alpha value of the impact spot in range of 0.0 to 1.0 which is interpolated towards 0 with the ***'Decay'*** parameter set in the Force Field script.

** hitPower and hitAlpha are can be omitted so the default values will be used with the OnHit call.*

```
// Update is called once per frame
0 references
void Update()
{
    // Activate on Left mouse button
    if (Input.GetMouseButtonDown(0))
    {
        // Returns a ray going from camera through a screen point
        RaycastHit hit;
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

        // Casts a ray against colliders in the scene
        if (Physics.Raycast(ray, out hit, 500.0f))
        {
            // Get Forcefield script component
            Forcefield ffHit = hit.transform.gameObject.GetComponent<Forcefield>();

            // Generate random hit power value and call Force Field script if successful
            if (ffHit != null)
            {
                float hitPower = Random.Range(-7.0f, 1.0f);
                ffHit.OnHit(hit.point, hitPower);
            }
        }
    }
}
```

The code demonstrated in this script may not be efficient in terms of optimisation, however it does it's job well on showing you the basics. In a real world example you would probably store precached references to all the Force Fields in your scene and avoid unnecessary calls like GetComponent from running each Update.

This concludes Force Field Basics section. Press Play in the editor and test the Force Field in the game view.

Using Force Field on Complex Surfaces

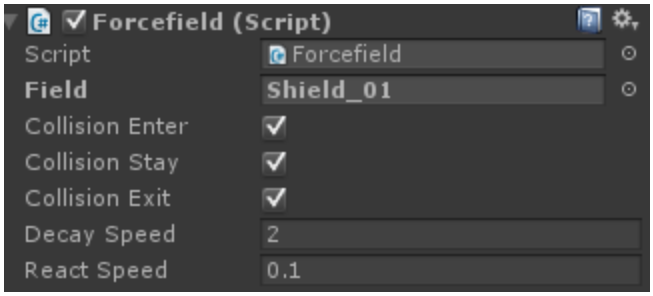
Force Field is very efficient at rendering on complex surfaces. However, there are few important requirements you should be aware of:

- **Complex model should be solid meaning it's vertices are welded together in 3D modelling package.**
- **Complex model should be using a single material assigned to the whole mesh in 3D modelling package.**
- **Complex model should be UV unwrapped.**

** Requirements may not apply to built in Unity primitives.*

Force Field Controller

Force Field Controller script is key component which allows you to control the shader behaviour by using its core OnHit method as well as to control the decay speed of the effect. It is also possible to set the activation time at which new impacts are registered.



'Decay Speed' **- Is the speed at which new impact points are faded. The bigger the value the faster the decay. Setting it to zero will stop effect from fading.**

'React Speed' **- Is the time gap in milliseconds which allows you to control how often the Force Field will register new impacts. Setting zero value will remove any limitations while setting value to 1 will register a new effect only once per second.**

Rigid body collisions

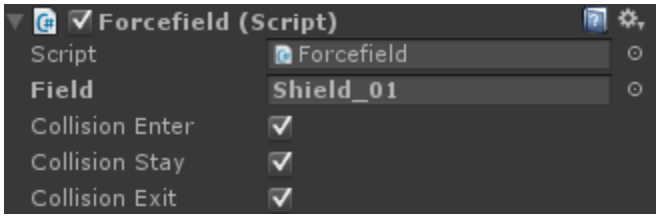
Force Field is capable of handling rigid body collisions events sent by Unity engine in the following methods:

OnCollisionEnter, OnCollisionStay and ***OnCollisionExit***

```
// COLLISIONS EVENTS
0 references
void OnCollisionEnter(Collision collisionInfo)
{
    if(CollisionEnter)
        foreach (ContactPoint contact in collisionInfo.contacts)
            OnHit(contact.point);
}
```

The collision data from each contact point is sent to the shader using *OnHit* method. Actually this is very similar to the raycasting you have already seen in the previous section of this manual.

You can control the behaviour of the Force Field with *Collision Enter*, *Collision Stay* and *Collision Exit* boolean flags:



To get more information about this events please refer to the Unity manual:
http://docs.unity3d.com/ScriptReference/Rigidbody.html

**** It is important to note that collision events are only sent if one of the colliders also has a non-kinematic rigidbody attached.***

** If there is a reason not to use Rigid Body vs Force FIeld collisions in your project it is best to comment out corresponding sections in Forcefield.cs script to avoid unnecessary calculations.*

Modifying number of impact points

Force Field is using 24 simultaneous impact points in Desktop version and 6 in Mobile version. If there is a reason you would like to alter the number of impact points processed by the Force Field please follow this steps:

1. Open Force Field Controller script (Forcefield.cs) and change the interpolator value.

```
public class Forcefield : MonoBehaviour {

    // Force Field component cache variables
    private Material mat;
    private MeshFilter mesh;

    // Number of controllable interpolators (impact points)
    private int interpolators = 24;
```

2. Open Force Field Shader (Forcefield.shader) and find the line starting with #include "UnityCG.cging" like shown in the picture:

```
41                  #include "UnityCG.cginc"
42
43                  fixed4 _Pos_0;
44                  fixed4 _Pos_1;
45                  fixed4 _Pos_2;
46                  fixed4 _Pos_3;
47                  fixed4 _Pos_4;
```

3. You will see two list of *fixed4 _Pos_… and fixed _Pow_...* variable declarations both starting with 0 index and ending with 23. Modify this two list so they both correspond to the amount of impact points you previously set within Forcefield.cs script. For example, if you changed *interpolators* value from 24 (default value) to 8, you should have two lists of shader variables both starting from 0 and ending with 7.

** Don't be surprised by the amount of variables it is required to modify by hand. Unfortunately it is not possible in Unity to send an array into shader directly and this seems to be the only available option.*

4. Scroll down and find a section of code that describes the number of impact points inside the fragment program within the shader and modify the value from 24 (by default) to the amount you like:

```
142                 /// FRAGMENT
143                 fixed4 frag (v2f i) : COLOR
144                 {
145                     int interpolators = 24;
146                     fixed4 pos[24];
147                     fixed power[24];
```

5. Modify two lists of *pos[]* and *power[]* array assignments to correspond to the number impact points you set earlier:

```
152                     pos[0] = _Pos_0;
153                     pos[1] = _Pos_1;
154                     pos[2] = _Pos_2;
```

```
177                     power[0] = _Pow_0;
178                     power[1] = _Pow_1;
179                     power[2] = _Pow_2;
```

6. Now go to the very first line starting with *Shader "Forge3D/Force FIeld"* and change string caption by adding new number of impact points to the end. For example *Shader "Forge3D/Force FIeld 8"*

```
1   Shader "Forge3D/Force Field 8"
2   {
3       Properties
4       {
```

7. Save your shader modification by giving a different name and switch back to Unity.

If everything went well Unity won't tell you a word about successful shader compilation. Now go to your Shield material and change the shader from the dropdown list to your modified version.

** It its not recommended to increase number of impact points in Mobile shader due to platform limitations and shader registers limit.*