Advanced Waving Flag Shader for Unity (Double sided, Alpha shadow support)

>>> Summary

- Flag vertices animation via a shader (trigo. way)
- Advanced ShaderLab surface shading for transparency management
- Adding a shadow Pass (with alpha support)
- Double-sided surfaces methods (methods 1, 2 and 3)
- Provided Double-sided Materials Configuration Wizard (for 3rd method)

[VIDEO 3]

>>> Flag vertices animation via ShaderLab shader (Unity 3)

Ref : http://forum.unity3d.com/threads/19575-waving-flag-shader-feedback-please!

Starting from this point, I made a more advanced work. Here come informations about how to use vertices animation directly in a shader.

[CODE]

```
void computeWave (inout appdata_full v, inout v2f o)
{
        float sinOff=(v.vertex.x+v.vertex.y+v.vertex.z) * _WaveStrength;
        float t=-_Time*_WaveSpeed;
        float fx=v.texcoord.x;
        float fy=v.texcoord.x*v.texcoord.y;

        v.vertex.x+=sin(t*1.45+sinOff)*fx*0.5;
        v.vertex.y=(sin(t*3.12+sinOff)*fx*0.5-fy*0.9);
        v.vertex.z-=(sin(t*2.2+sinOff)*fx*0.2);
        o.pos = mul( UNITY_MATRIX_MVP, v.vertex );
        o.uv = v.texcoord;
}
```

Here's the original waving algorithm (by cboe and Seon) found at the link bellow.
I simply added a "_WaveStrength", parameter to get a flag more reactive, for strong windy weather.
[FIG 1]

>>> Advanced ShaderLab surface shading for transparency management

The “surface” shaders, available in Unity 3 are very simple and powerful (excepted that multi-passes are not allowed – will see it later in this doc.)

Anyway, they are the way I chose to accomplish this work.

[CODE]

```
Tags {
        "Queue"="Geometry"
        "IgnoreProjector"="True"
        "RenderType"="Transparent"
}

LOD 300

CGPROGRAM
                #pragma surface surf BlinnPhong alpha vertex:vert fullforwardshadows approxview
                #include "FlagWaveCG.cginc"

                half _Shininess;

                sampler2D _BumpMap;
                float _Parallax;

                struct Input {
                        float2 uv_MainTex;
                        float2 uv_BumpMap;
                };

                v2f vert (inout appdata_full v) {
                        v2f o;
                        computeWave(v, o);
                        return o;
                }

                void surf (Input IN, inout SurfaceOutput o) {
                        fixed4 tex = tex2D(_MainTex, IN.uv_MainTex);
                        o.Albedo = tex.rgb * _Color.rgb;
                        o.Gloss = tex.a;
                        o.Alpha = tex.a * _Color.a;
                        o.Specular = _Shininess;
                        o.Normal = UnpackNormal(tex2D(_BumpMap, IN.uv_BumpMap));
                }
ENDCG
```

Detail about the code bellow :

> Shader Tags :

- The render queue “Geometry” is important to keep the object rendering at the correct depth in render process (usually “Transparent” is used for transparent objects, but in our case we want the flag to be a full solid entity in our scene graph.
- The render type “Transparent”, is also important to enable holes in the surface (using the Alpha channel from the main texture)

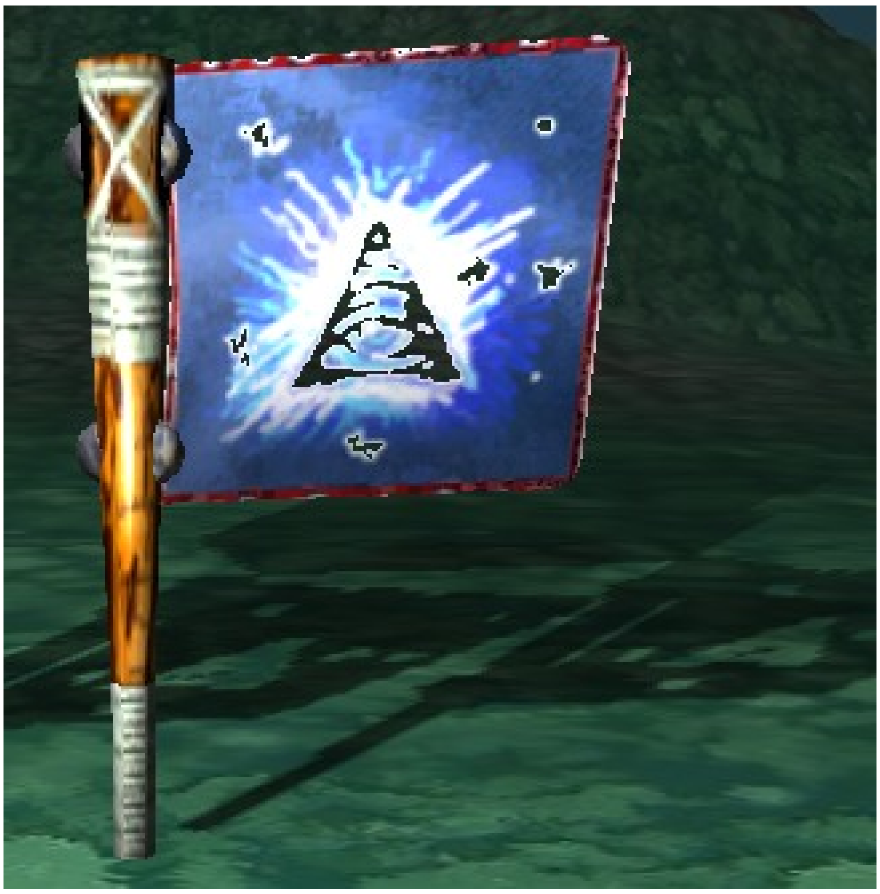
> “#pragma surface surf BlinnPhong alpha vertex:vert fullforwardshadows approxview” :

This line is quite complex but the most important here are :

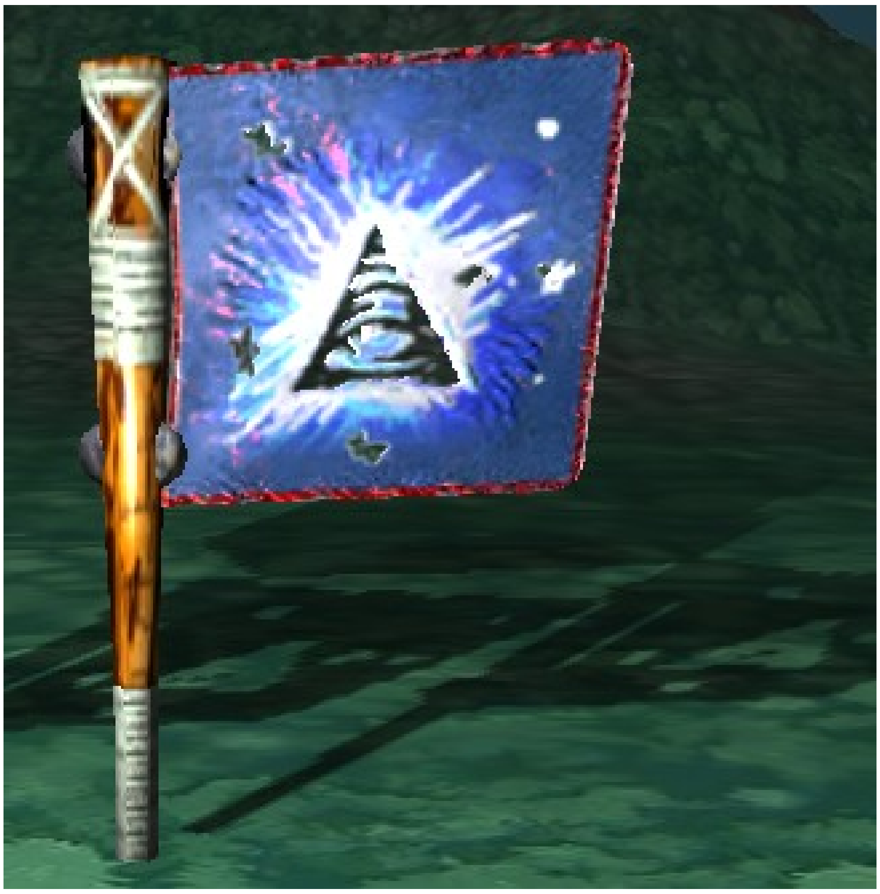
- The “alpha” statement : allows to get a nice rendering of the semi-transparent pixels (different from the simple “Cutoff” way, which only displays or not a pixel)

This needs to be coupled with the instruction “o.Alpha = tex.a * _Color.a;” (in the 'surf' function).

[FIG 2.1]



[FIG 2.2]



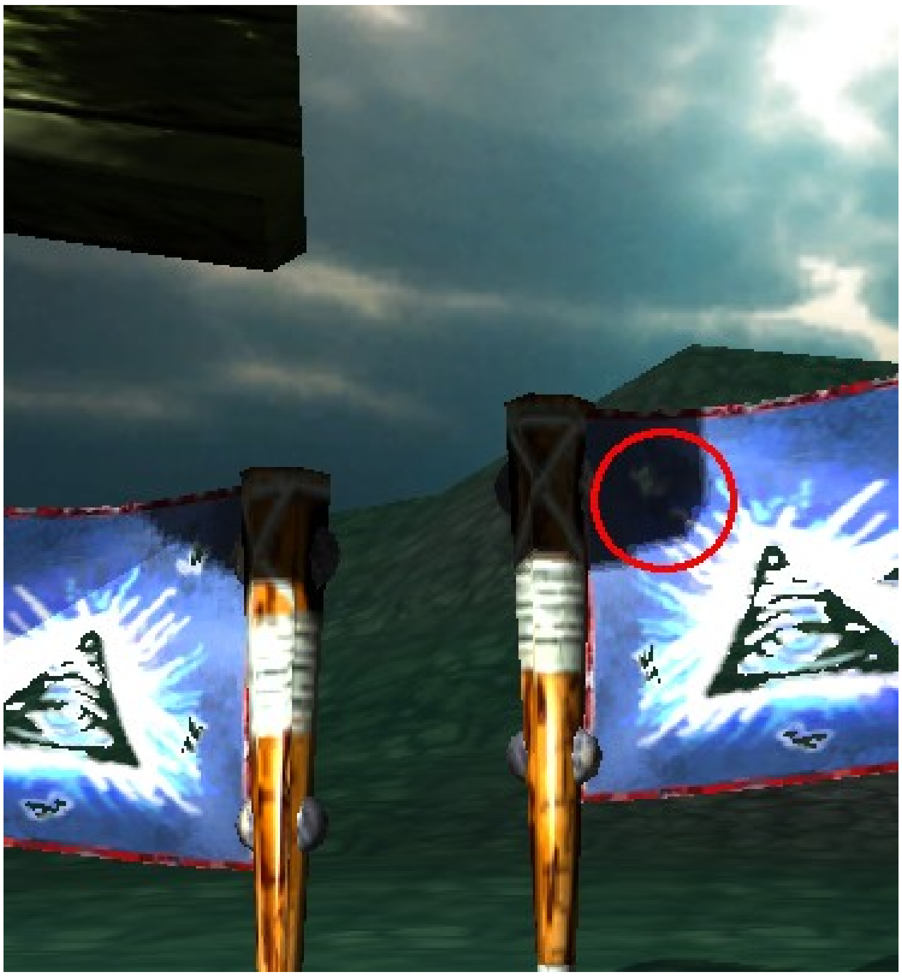
In [FIG 2.1], we can ostensibly see the aliasing where the alpha values should be smooth (as I made it smooth in the Alpha channel of my texture).
In [FIG 2.2], the result using the “alpha” statement.

Warning : This way of doing can cause issues when using the built-in “Skybox” system in Unity...
So be aware of the render queue offsets you use.

– The “vertex:vert” statement : where 'vert' is the function where to apply the vertices motion as described bellow ('computeWave' function).

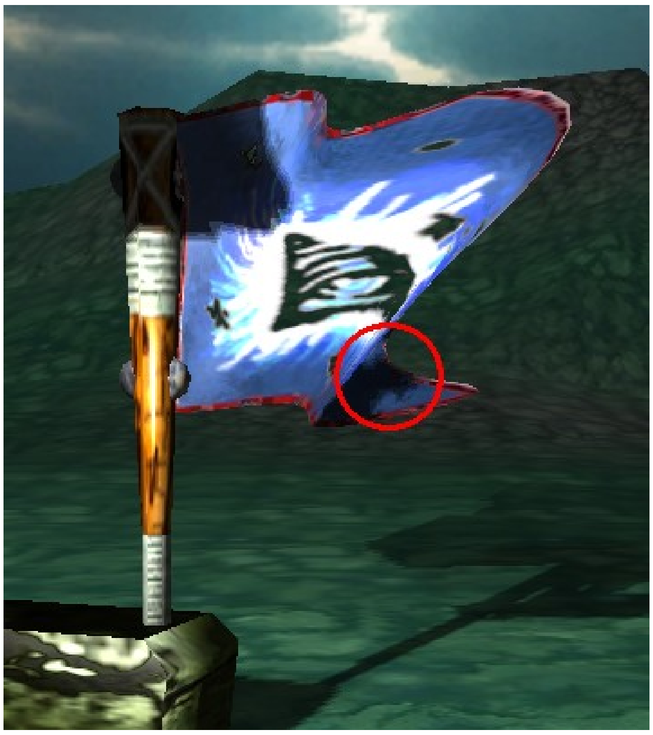
– The “ fullforwardshadows” statement : enables our flag to receive shadows even if “alpha” is On :

[FIG 3]



The flags are receiving shadows from the upper-left solid object.
Note : the projected shadow doesn't affect the surface transparency (red circled area)

Their also self-illuminated :
[FIG 4]



– The “approxview” statement : is only here to spare some computing instructions (when the camera is very close to the subject.

>>> Adding a shadow Pass (with alpha support)

For a more realistic use of this shading work-flow, the flag should also cast shadows, and those shadows must follow the flag waving animation.
To achieve this we need an additional pass in the shader :

[CODE]

```
Pass
{
                Name "ShadowCaster"
                Tags { "LightMode" = "ShadowCaster" }

                Fog {Mode Off}
                ZWrite On ZTest Less Cull Off
                Offset 1, 1

                CGPROGRAM
                #pragma exclude_renderers gles
                #pragma vertex vert
                #pragma fragment frag
                #pragma fragmentoption ARB_precision_hint_fastest
                #pragma multi_compile_shadowcaster
                #include "FlagWaveCG.cginc"

                v2f vert( appdata_full v )
                {
                        v2f o;
                        computeWave(v, o);
                        TRANSFER_SHADOW_CASTER(o)

                 return o;
                }

                float4 frag( v2f i ) : COLOR
                {
                        fixed4 texcol = tex2D( _MainTex, i.uv );
                        clip( texcol.a - _Cutoff );
                        SHADOW_CASTER_FRAGMENT(i)
                }
                ENDCG
}
```

In this pass, we first apply ('vert' function) our waving UV transformation with exactly the same function we used in the 'surface pass' ('computeWave' function), then we tell the render engine to cast shadows according to this redefined geometry ('TRANSFER_SHADOW_CASTER' instruction).

Finally, we want the cast shadow to take in account the alpha parts of the main texture : this is done in the 'frag' function, where the shadow is cut off accordingly to the "_MainTex" alpha channel ('SHADOW_CASTER_FRAGMENT' instruction).

Note : The clipping amount can be customized by externally modulating the "_Cutoff" parameter.

>>> Intermediate Conclusion :

Well, we're done with the first part of the job : having a nice realistic waving flag. So your done too if you don't need the back sides of the flag to be displayed. The next part of this document aims to show how to get a double-sided surface using different methods (which all have benefits and drawbacks, of course).

Here comes the full source code for this shader :

[CODE] / The shared '.cginc' include file : FlagWaveCG.cginc

```
#include "UnityCG.cginc"

float4 _Color;
sampler2D _MainTex;
fixed _Cutoff;
float _WaveSpeed;
float _WaveStrength;


struct v2f {
	V2F_SHADOW_CASTER;
	float2 uv : TEXCOORD1;
};


void computeWave (inout appdata_full v, inout v2f o)
{
	float sinOff=(v.vertex.x+v.vertex.y+v.vertex.z) * _WaveStrength;
	float t=-_Time*_WaveSpeed;
	float fx=v.texcoord.x;
	float fy=v.texcoord.x*v.texcoord.y;

	v.vertex.x+=sin(t*1.45+sinOff)*fx*0.5;
	v.vertex.y=(sin(t*3.12+sinOff)*fx*0.5-fy*0.9);
	v.vertex.z-=(sin(t*2.2+sinOff)*fx*0.2);
	o.pos = mul( UNITY_MATRIX_MVP, v.vertex );
	o.uv = v.texcoord;
}
```

[CODE] / The main shader program : FlagWave-Advanced.shader

```

// Original shader by cboe - Mar, 23, 2009
// Enhanced to 3 axis movement by Seon - Jan, 21, 2010
// Added _WaveSpeed by Eric5h5 - Jan, 26, 2010
// CHANGE LOG - Gauthier BOAGLIO (golgauth) / Klakos - May, 07, 2011 :
//		- Added Transparency support
//		- Added Spec and Normal mapping support
//		- Added Shadow casting support (+ Shadow Alpha and Shadow Alpha cutoff
support)
//				[Done in the "ShadowCaster" additional Pass]
//		- Added advanced double-sided rendering support
//		- Added _WaveStrength param
//
// Requirements: assumes you are using a subdivided plane created with X (width) * Z (height)
where Y is flat.
// Requirements: assumes UV as: left X (U0) is attatched to pole, and Top Z (V1) is at top of pole.
//
// See [ http://klakos.com/en/advanced-waving-flag-shader-for-unity-double-sided-alpha-shadow-
support-2/ ] for
// visuals and more informations


Shader "Selfmade/for-2sided/FlagWave Advanced Regular"
{

Properties
{
	// Ususal stuffs
	_Color ("Main Color", Color) = (1,1,1,1)
	_SpecColor ("Specular Color", Color) = (0.5, 0.5, 0.5, 0)
	_Shininess ("Shininess", Range (0.01, 1)) = 0.078125
	_MainTex ("Base (RGB) TransGloss (A)", 2D) = "white" {}

	// Bump stuffs
	//_Parallax ("Height", Range (0.005, 0.08)) = 0.02
	_BumpMap ("Normalmap", 2D) = "bump" {}
	//_ParallaxMap ("Heightmap (A)", 2D) = "black" {}

	// Shadow Stuff
	_Cutoff ("Shadow Alpha cutoff", Range(0.25,0.9)) = 1.0

	// Flag Stuffs
	_WaveSpeed ("Wave Speed", Range(0.0, 300.0)) = 50.0
	_WaveStrength ("Wave Strength", Range(0.0, 5.0)) = 1.0
}


SubShader
{
	Tags {
	"Queue"="Geometry"
	"IgnoreProjector"="True"
	"RenderType"="Transparent"}

	LOD 300
```

```
        Pass
        {
                        Name "ShadowCaster"
                        Tags { "LightMode" = "ShadowCaster" }

                        Fog {Mode Off}
                        ZWrite On ZTest Less Cull Off
                        Offset 1, 1

                        CGPROGRAM

                        #pragma exclude_renderers gles
                        #pragma vertex vert
                        #pragma fragment frag
                        #pragma fragmentoption ARB_precision_hint_fastest
                        #pragma multi_compile_shadowcaster
                        #include "FlagWaveCG.cginc"


                        v2f vert( appdata_full v )
                        {
                                v2f o;
                                computeWave(v, o);
                                TRANSFER_SHADOW_CASTER(o)

                                 return o;
                        }

                        float4 frag( v2f i ) : COLOR
                        {
                                fixed4 texcol = tex2D( _MainTex, i.uv );
                                clip( texcol.a - _Cutoff );
                                SHADOW_CASTER_FRAGMENT(i)
                        }
                        ENDCG

   }

//CULL Front

CGPROGRAM
                #pragma surface surf BlinnPhong alpha vertex:vert fullforwardshadows approxview
                #include "FlagWaveCG.cginc"

                half _Shininess;

                sampler2D _BumpMap;
                //sampler2D _ParallaxMap;
                float _Parallax;
```

```
		struct Input {
			float2 uv_MainTex;
			float2 uv_BumpMap;
			//float3 viewDir;
		};

		v2f vert (inout appdata_full v) {
			v2f o;
			computeWave(v, o);
			return o;
		}

		void surf (Input IN, inout SurfaceOutput o) {
			// Comment the next 4 following lines to get a standard bumped rendering
			// [Without Parallax usage, which can cause strange result on the back side of
the plane]
			/*half h = tex2D (_ParallaxMap, IN.uv_BumpMap).w;
			float2 offset = ParallaxOffset (h, _Parallax, IN.viewDir);
			IN.uv_MainTex += offset;
			IN.uv_BumpMap += offset;*/

			fixed4 tex = tex2D(_MainTex, IN.uv_MainTex);
			o.Albedo = tex.rgb * _Color.rgb;
			o.Gloss = tex.a;
			o.Alpha = tex.a * _Color.a;
			//clip(o.Alpha - _Cutoff);
			o.Specular = _Shininess;
			o.Normal = UnpackNormal(tex2D(_BumpMap, IN.uv_BumpMap));
		}
ENDCG
}

Fallback "Transparent/VertexLit"
}
```

Note : If you decide to use this shader “as is” and want to enable the parallax feature, you 'll have to be careful with the number of instructions called in the 'surf' function (the limit is 64). Adding some will inevitably force you to remove others (as this shader is already quite loaded – ex : “fullforwardshadows” is a bit heavy, this is the reason why I had to “approxview”).

>>> Double-sided surfaces methods (methods 1, 2 and 3)

> The “Doubled Mesh” Method :

This method is the most commonly used, because it is simple an efficient.
Just duplicate your plane mesh and, keeping the the same materials, just flip the normals of the duplicated one using your favourite 3D modelling tool (that is to say : Blender ;-).

Benefits : Easy
Drawbacks : Every time the mesh has to be modified, the second one has to be too : it can be

complicated an take a while.

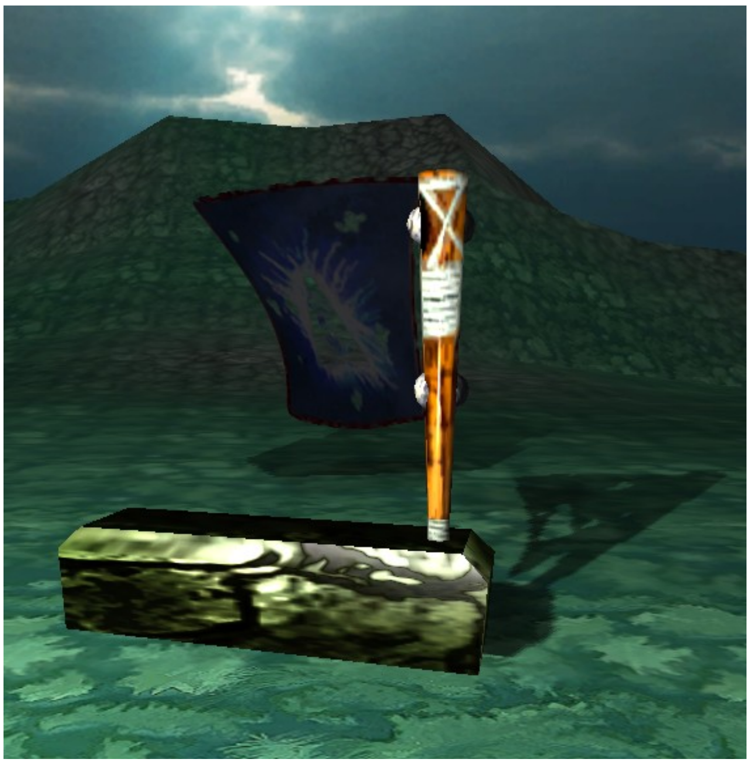
> The “CULL Off” Method :
That one is also very simple to achieve :
We just need to edit our previous shader and add the “CULL Off” instruction right before the “CGPROGRAM” statement, like this :

[CODE]

```
CULL Off

CGPROGRAM
			#pragma surface surf BlinnPhong alpha vertex:vert fullforwardshadows approxview
			#include "FlagWaveCG.cginc"
		.
		.
		.
```

The “CULL” statement can take 3 values :
- “Back” (the default) : hides backwards faces
- “Front” : only shows the backwards faces
- “Off” : displays all faces

[FIG 5]


Well, it works, but where is my Bump, Spec, and all the other stuffs ?

Benefits : Easy, included in the shader (so further mesh changes won't affect anything)

Drawbacks : The back faces are displayed, but they inherit from the front faces for lighting (in [FIG 5], it turns out that the flag is plunged into darkness X-). Excepted in some very specific cases, it isn't satisfying at all.

> The “Flip Normals Via Shader” Method :

The one I prefer !

Ref : http://forum.unity3d.com/threads/61882-U3-Add-2-sided-to-AlphaTest-Bumped.shader
[Big thanks to Daniel Brauer]

The aim of this method is to treat separately the front and back faces of the plane mesh. To achieve this and because we are working with surface shaders (remember that they don't support multi-passes*), we need to create two specific Shaders / Material for each kind of faces.

*Note : If they did, we could have used two passes (one for front faces et an other one for back faces) in one unique Shader/Material , instead of using two...

Step 1 :
Choosing any surface shader script (from Unity built-in shaders
[http://unity3d.com/support/resources/assets/built-in-shaders] or from yourselves).
Note : The “CULL” value must be “Back” (or no “CULL” statement at all). This Material / Shader must affect only front faces !

Step 2 :
Affecting it to a newly created Material in Unity, and setting up as desired.

Step 3 :
Adding this Material to the list of materials of the Renderer of the flag plane object.

Step 4 :
Creating the Shader for the backwards faces.
For this, simply editing the previous shader [let's call that one : the “Regular” shader] and save it with another name [let's call it : the “Backward” shader].

Then making some very few modifications :
We don't need to change the general behaviour of the shader : the back faces must have the same reactions to lights than the front ones. All we want is to make the back faces to be well oriented when we show them : the trick, here, is to invert the normals directions directly via the shader :

1- Adding a “CULL Front” directive (as explained earlier in the doc. for “CULL Off”)
2- Invert the normals at unpacking time, like follows (“minus” before “UnpackNormal” command) :

[CODE]

```
CULL Front

CGPROGRAM
        #pragma surface surf BlinnPhong alpha vertex:vert fullforwardshadows approxview
        .
        .
        .
```

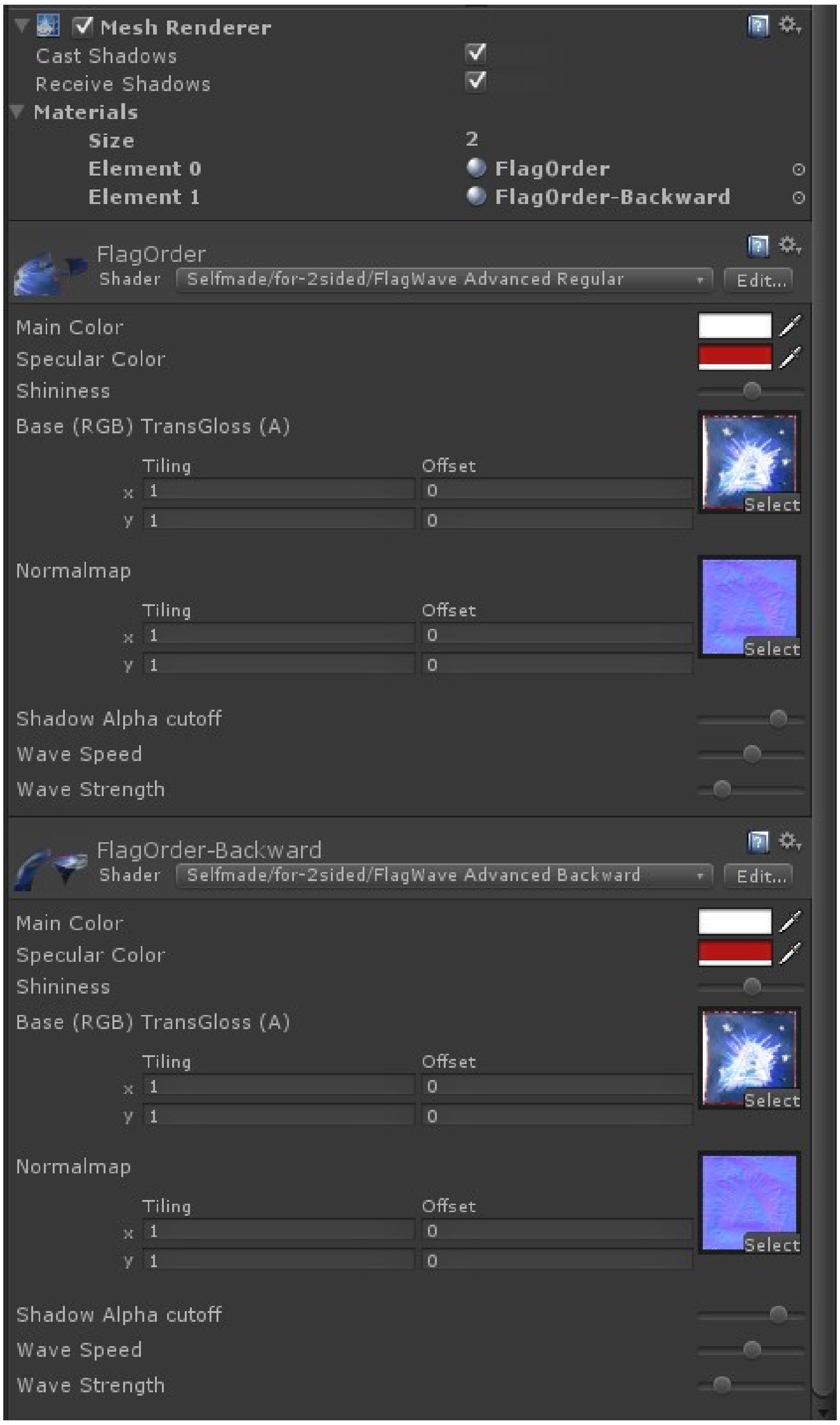
```
		void surf (Input IN, inout SurfaceOutput o) {
			o.Gloss = tex.a;
			o.Alpha = tex.a * _Color.a;
			.
			.
			.
			o.Specular = _Shininess;
			o.Normal = -UnpackNormal(tex2D(_BumpMap, IN.uv_BumpMap));
		}
ENDCG
```

Step 5 :
Affecting the “Backward” shader to a newly created Material, and setting it up the same as the “Regular” one.

Step 6 :
Adding this second Material to the list of materials of the Renderer of the flag plane object.

Step 7 :
You're done !
Turn the object in any position, move and add lights, modify the mesh. Everything should work like a charm...

[FIG 6]



Mesh Renderer
Cast Shadows ✓
Receive Shadows ✓
Materials
Size 2
Element 0 FlagOrder
Element 1 FlagOrder-Backward

FlagOrder
Shader Selfmade/for-2sided/FlagWave Advanced Regular

Main Color
Specular Color
Shininess
Base (RGB) TransGloss (A)

	Tiling	Offset
x	1	0
y	1	0

Normalmap

	Tiling	Offset
x	1	0
y	1	0

Shadow Alpha cutoff
Wave Speed
Wave Strength

FlagOrder-Backward
Shader Selfmade/for-2sided/FlagWave Advanced Backward

Main Color
Specular Color
Shininess
Base (RGB) TransGloss (A)

	Tiling	Offset
x	1	0
y	1	0

Normalmap

	Tiling	Offset
x	1	0
y	1	0

Shadow Alpha cutoff
Wave Speed
Wave Strength

Those final settings gave :
[VIDEO 4]

Benefits : Mostly, the fact that only one mesh is used and both sides are correctly displayed, without any loss of features (transparency, shadowing, ...) previously described in this document.

Drawbacks : Requires 2 materials (having the same settings) [See the following "Configuration Wizard" for easier way to configure and synchronize "Regular" => "Backward" materials]

>>> Provided Double-sided Materials Configuration Wizard (for 3rd method)

> Description :
This tool will help you to cover the steps said bellow. Mostly the need to synchronize BACK and FRONT faces materials.

> Prerequisites :

1- Naming conventions for shaders.

To start with this tool, you'll have to be aware of the fact that (for simplicity reasons), it is based on some names conventions :

– Front/Forward shaders Name has to end with “Regular” suffix
– Back/Backward shaders Name has to end with “Backward” suffix

Those conventions deal with the Shaders name (not the Materials name)

[CODE]

```
Shader "Selfmade/for-2sided/FlagWave Advanced Regular"
{

Properties
{
        // Usual stuffs
        _Color ("Main Color", Color) = (1,1,1,1)
        _SpecColor ("Specular Color", Color) = (0.5, 0.5, 0.5, 0)
        .
        .
        .
```

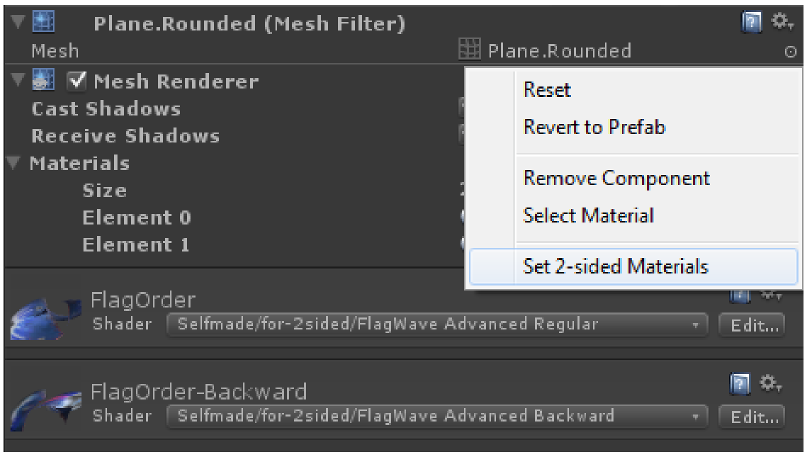
2- Material assets need to be made by hand and this tool will only help to set them up correctly.

3- The different shaders may have been previously created.

> Install :
Just drop the `“BackwardMaterialBuilder.cs”` (`ScriptableWizard`) script into the “Assets/Editor” directory of the Unity project.

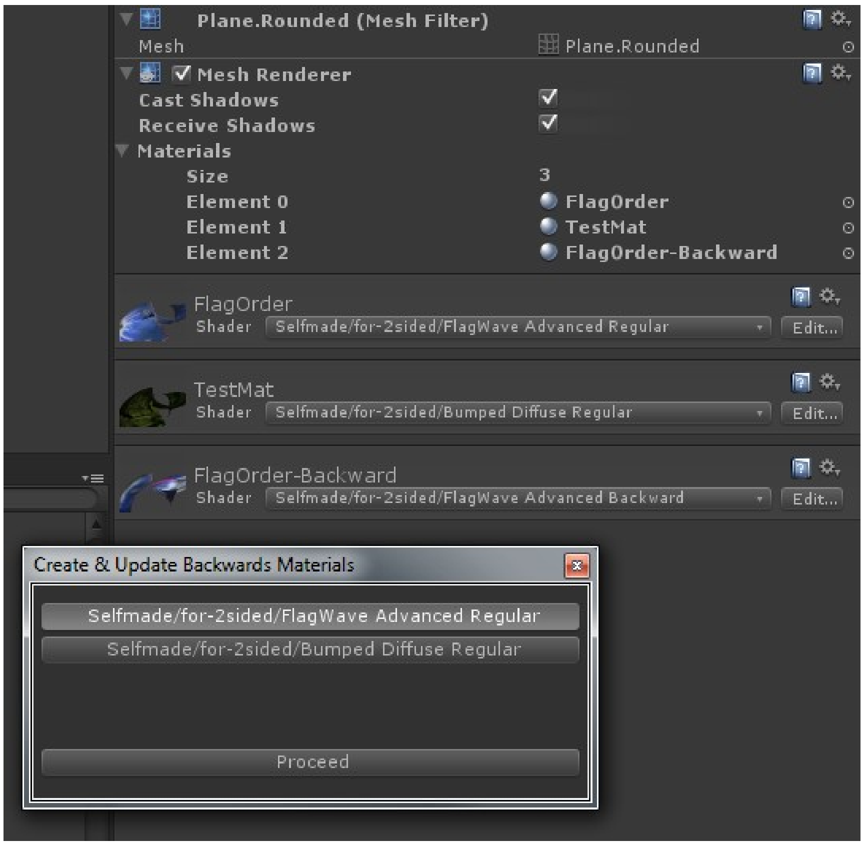
> Usage :
To access the script, select an object (in the Hierarchy or Project pane). Then right-click on the context menu icon of the Renderer (either Mesh or SkinnedMesh Renderer), and choose “Set 2-sided Materials” :

[FIG 7]



The related window displays a selectable list of all the “Regular” shaded materials from the Renderer materials list :

[FIG 8]


Plane.Rounded (Mesh Filter)
Mesh
Plane.Rounded
Mesh Renderer
Cast Shadows
Receive Shadows
Materials
Size
3
Element 0
FlagOrder
Element 1
TestMat
Element 2
FlagOrder-Backward
FlagOrder
Shader
Selfmade/for-2sided/FlagWave Advanced Regular
Edit...
TestMat
Shader
Selfmade/for-2sided/Bumped Diffuse Regular
Edit...
FlagOrder-Backward
Shader
Selfmade/for-2sided/FlagWave Advanced Backward
Edit...
Create & Update Backwards Materials
Selfmade/for-2sided/FlagWave Advanced Regular
Selfmade/for-2sided/Bumped Diffuse Regular
Proceed

Then select one of those Materials and press “Proceed”.

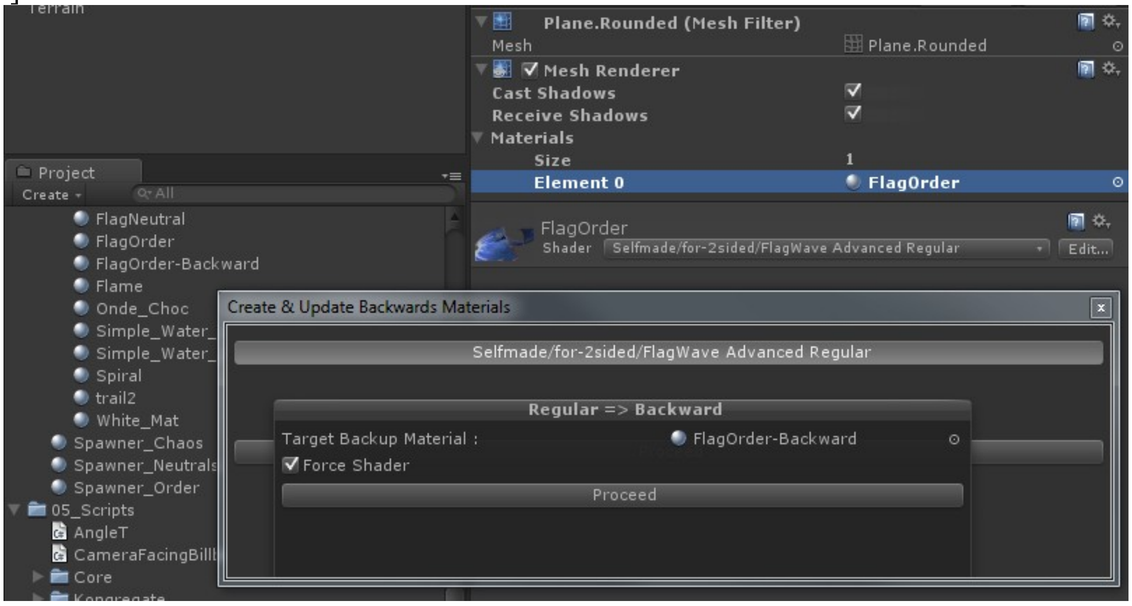
If the corresponding “Backward” Shader exists in you Project, 3 different things may happen :

- Synchronize props form “Regular” to “Backward”, if those Materials are defined already
- Create the “Backward” one if it doesn't exist (not setting the Shader)
- Erase or not the actual Shader of the current “Target Backup Material”

Then drop a hand made target Material of your choice from your “Assets” folder into the “Target Backup Material” field.

Option “Force Shader”, will set the right existing “Backward” Shader to the targeted Material.
If unchecked, the Shader of the original dropped Material will remain the same.
In many cases, this option has to be checked (but you'll lose all the current settings of the target Material).

[FIG 9]



Plane.Rounded (Mesh Filter)
Mesh: Plane.Rounded
Mesh Renderer
Cast Shadows: ✓
Receive Shadows: ✓
Materials
Size: 1
Element 0: FlagOrder

FlagOrder
Shader: Selfmade/for-2sided/FlagWave Advanced Regular

Create & Update Backwards Materials
Selfmade/for-2sided/FlagWave Advanced Regular
Regular => Backward
Target Backup Material : FlagOrder-Backward
✓ Force Shader
Proceed

Big Conclusion

Well, nothing more actually...
Hope this will help,
and don't forget to post feedbacks (I need you to make this doc. better)
Thanks !