Ultimate Fog of War: Documentation

Document Version 1.6 (Click for Latest Version)

Ultimate Fog of War 1.6

Table of Contents

Update Notes

Version 1.6

Shader Updates (Unity 2017.1 and later)

The implementation of the ULTIMATE FOG OF WAR Fog calculation is now smarter and shorter. It required less Shaders and is sorted better. The Conversion Guide has been updated.

Multi Hardblocker + ID

It is now possible to place multiple Hardblockers at once (e.g.: 3x3, 4x4, 2x5) and give them an ID. 0 is unblocked, 1-255 are ID's.

Faction Management

Each Faction can now reveal multiple other factions. This is handy if you want to create and change teams. You can choose if you want to Update them in the background while it is not being revealed or not.

Editor improvements

The Editor is now faster and more responsive.

Important!

All future updates will only be provided for the latest version of Unity.

Version 1.3

Standard Shader

ULTIMATE FOG OF WAR now comes with a permutation of the Standard Shader for Horizontal/ Terrain games. It works just the same as the normal Standard Shader. Pick it from:

UltimateFogOfWar/Standard or

UltimateFogOfWar/Standard (Specular Setup)

Grid Scale

The Grid, defining the visibility of the Map can now be scaled, to change its density.

Origin

The Origin of the Terrain or the Canvas can be changed.

Version 1.2

Vertical Mode

You can now decide if you want to make a Vertical or Horizontal game. If you choose the new option (Vertical), you can automatically create a Canvas that is set to the exact size of the Fog frustum. Unity's sprite shaders for UI have been adapted and can be picked from the Shader selection. In Case you choose not to use sprites, it is possible to use any geometry (See Vertical template Shaders).

Vision Modifiers

This Version introduces Vision Modifiers to ULTIMATE FOG OF WAR. Vision Modifiers are Vision blockers, that block a Revealer from seeing inside the defined area, just like tall grass in DOTA or LOL and Smoke Screens in Starcraft II. Vision Modifiers become see through to a Revealer, as soon as it enters the defined patch.

Every patch has an ID to for more control.

Debug Options

A new Set of Debug options is now available. You can now get detailed information of the ResistenceMap per patch.

Particle Shaders

A new Set of Particle Shaders is now available, both for Vertical and Horizontal games.

ResistanceMap Tools

It is now Possible to change Hardblockers, Height and Vision Modifiers in the Editor. (See Modifier Options)

Version 1.1

Visibility testing

The Fog map generation has changed from having a single Fog map to having one Fog map per Faction. It is now possible to define the current Faction to be displayed. Only this Faction's Fog Map will be blurred. The other Factions are still updated, though.

According to this, it is now possible to do all necessary calculations on one instance (Server). In Unity 2017.1 and later

IsSelfRevealed(Vector3 _Position) has been replaced with IsPositionRevealedByFaction(Vector3 _Position, int _RevealingFaction).

Saving the state of the Fog

It is now possible to Save the current state of the fog and load it again. This can be done in 2 Ways:

1. Use FogOfWarManager.SaveFogMaps() & FogOfWarManager.LoadFogMaps(), to save and load automatically.
2. Use **FogOfWar.GetFogMapData & FogOfWar.SetFogMapData** to implement your own save algorithm.

Please Note: **FogOfWarManager.SaveFogMaps** & **FogOfWarManager.LoadFogMaps** are **not** available for **WebGL** builds.

Shader fixes

-

Improved efficiency.

-

Description

ULTIMATE FOG OF WAR is a grid based Fog of War solution, with Field of View Blockers, based on multiple options:

1. Static Blockers
2. Height Blockers
3. Dynamic Blockers
4. Vision Modifiers (Tall grass)

The First two are saved in a Resistence Map in the colour channels R&G. The Dynamic Blockers are saved in a separate Bufferer and can be added and removed at Runtime. Vision Modifiers usually have a corresponding Effect/ Prefab that can be placed in Edit Mode and is saved in the Resistence Map.

ULTIMATE FOG OF WAR handles all necessary calculations per Scene in the Fog of war Manager Component for you. If you want to Use the Unity 3D Terrain, there is a tool for you to set it up properly. Click on Fog of War > **New Terrain** in the tool bar and everything will be set up for you:

1. Creating Terrain
2. Creating a Resistence map
3. Setting up the Fog of War (Buffers, Threads, settings)

All Unity Build-in Shaders related to Terrain have been adapted to work with ULTIMATE FOG OF WAR. It is very simple and with only a few lines of code, you can change your Shader to work with Fog of War, too.

Features

Toolbar

1. New Terrain (Fog of War > New Fog Of War Manager)

Opens a “Fog Of War Editor prompt” (see below). This creates a new Terrain or Canvas and a Fog of War Manager and saves a Resistence Map to a specified location. You can start designing immediately.

2. New Revealer (Fog of War > New Revealer)

This will add a Revealer for Debugging purposes to your Scene.

3. New Blocker (Fog of War > New Blocker)

This will add a Dynamic Blocker for Debugging purposes to your Scene.

New Fog Of War Manager Prompt

In this prompt you can create a new Terrain or a new Canvas. After that, you can immediately start designing your level, map etc.

Step 1

Map Size:

Choose a Map that matches your Level. You can Pick map sizes between 16 and 512 for both axis individually. This cannot be changed later!

Overshoot:

This defines how far (in world units) the Terrain will be extended to all sides. This can also be done manually later, but should be done at the start because it might lead to data loss in the Terrain.

Border Map:

This will create a static border around the map in order to prevent Revealers from looking outside the level.

Fog Opacity:

1. Revealed: The opacity of the fog while being revealed.

2. Covered: This patch has been discovered before but at the moment is not being revealed.

3. Unrevealed: This patch has never been seen by a Revealer.

Other Options:

1. Material: Pick the Shader you want to use for this Terrain (Diffuse, Standard, or Specular). This will automatically pick the correct Fog of War Shader. (Terrain Mode only!)

2. Blur Fog of War: If you want a better Fog quality, enable this feature. For performance optimized games or pixelated Fog, turn this off.

3. Filter: Set this to Point if you want to make the Fog pixelated.

Step 2

In this step, you can define where you want to save your Materials and the Resistance Map. This is the Map used in the Terrain and can be modified using Code or Photoshop (or any other image tool). In case you want to create only the Fog of War Manager and create the Terrain or the Canvas on your own, un-tick the “Create Terrain” / “Create Canvas” Box.

Fog of War Manager

The Fog of War Manager takes care of the fog of War. Every scene needs only one and supports only one Fog of War map.

Preview tab

In the preview Tab you can see the actual values used for the Field of View Calculations. Click on the tab to display the information you want to see: Static, Height, Modifiers and the Fog. If you have multiple factions, you can switch between them to display its Fog of War.

Note that you have to play the game to see the Fog of War Map.

Level Tab

In the following tabs, the Fog of War Manager is referencing:

1. The Terrain/ Canvas
2. The Path to the Resistance Map
3. The Resistance Map

Usually you don't have to change anything to these three value, but ULTIMATE FOG OF WAR can only work while this data is provided.

Scaling & Origin:

Scaling can be enabled and changed to edit the Grid's density. This is useful, if your game has a bigger or smaller Scale. You can also optionally change the Origin of your Terrain (Bottom left Corner of the Grid).

Level Dimensions:

This can only be changed on initialization because it would lead to data loss if changed. (Width and Depth of the Map)

Cover Speed: The Speed at which an uncovered patch is covered. Bright -> Dark

Reveal Speed: The Speed at which a covered patch is revealed. Dark -> Bright

Min Max Height:

Here is the definition of the height of your Terrain. The maximum

precision is 256 different steps and is defined by the Resistence Map and NOT by the actual y-position. You can also Analyze the Terrain here and save it as to the Resistence Map. (Terrain Mode only)

Fog opacities:

The Fog opacities can be changed here as well.

1. Revealed: The opacity of the fog while being revealed.
2. Covered: This patch has been discovered before but at the moment is not being revealed.
3. Unrevealed: This patch has never been seen by a Revealer.

Factions Tab

In this Tab you can add or remove factions. You can also define which faction is this client belongs to. This is important when this Client is also a server and handles visibility for other Clients.

Blur Options

Define the settings for the Fog RenderTexture. Also controls the style of the fog (Color and animated effect).

Upsample

How many times should the Fog texture size be up scaled for blurring? This increases the quality of the Fog. Level Size = 128x128, Upscale 2x, Blur texture size = 256 x 256.

Blur Fog

Turn Blur On/Off.

Filter Mode

Set the Filtering for the RenderTexture.

Modifier Options Tab

Open this Tab to start Editing the Resistance Map. You can Change Hard blockers, Height, Modifiers (Grass) or delete Modifiers.

Revert All

Reverts all Modifier Changes.

Save

Save all Changes to your Resistance Map.

You have to close this Tab in order to stop Editing the Resistance Map!

Diagnostic and Performance Tab:

Use Threading:

If enabled, all Field of View calculations will be done asynchronously to save performance. This feature is not supported on Unity Web GL Export because Threading is not supported.

Automatically Update:

This feature allows you to limit the Rate of Updates per second for performance optimizations. The Fog still fades every frame.

If you need to Update manually, you can Update the fog once by calling CalculateOnce(). To use this, disable "Automatically Update" and set the Updates per Second to 0.

Reveal Blockers:

Increase vision +1 if the last patch is a blocker.

Show Guides:

Display the Debug Mode in the Scene View

Fog Effect (2017.1 and later):

Color: Pick “Color”, to define a color of the Fog. Bright colors will make the Fog invisible! The Animated Fog option creates an effect by sampling multiple layers of noise and overlaying it on top of the Fog. You can find the algorithm here:

FogOfWarMath.cginc

Fog effects require the FoWAnimatedFog or FoWColor keyword.

Fog of War Blur

This Component will be added to the Fog of War Manager automatically if Blur Fog is enabled. It is an adaption of the mobile blur standard asset, adapted to work with any RenderTexture.

Downsample: This will reduce the size of the Fog RenderTexture before bluring to improve performance.

Blur Size: The Size/ Range of the Blur.

Blur Iterations: Number of iterations to increase quality.

Blur Type: Style of the Blur.

Blur Shader: Shader used for blurring.

Shader adaption guide (5.4)

This guide covers the adaption of surface and CG Shaders for ULTIMATE FOG OF WAR. You can also work with the template Shaders to get right into designing your Shader without worrying about the Fog. You can find those here: 'Assets\FogOfWar\Resources\Shaders\Terrain\Templates'

This guide assumes that the naming of the input and output structs are the same as Unity 3D standard built-in shaders.

Surface Shaders

Step 1: Include FogOfWarMath.cginc

```
#include "UnityCG.cginc"
```

Make sure that you get the path correct or else Unity can't find the file.

Step 2:

Add the global variables used to sample the fog:

```
uniform sampler2D _FogOfWar;
uniform float _LevelWidth;
uniform float _LevelHeight;
```

Step 3:

Add the World Position (float3 worldPos;) to the shader's input struct. It should look like this:

```
struct Input {
        float2 uv_MainTex;
        float3 worldPos;
};
```

Step 4:

Go to the Shaders surface function (Usually called 'surf') and add the following lines, if you are making Horizontal Shader:

```
float2 UV = float2(
        Remap(IN.worldPos.x, 0, _LevelWidth, 0, 1),
        Remap(IN.worldPos.z, 0, _LevelHeight, 0, 1)
);


half4 Fog = tex2D(_FogOfWar, UV);
```

Or Add these lines if you are making a Vertical Shader:

```
float2 UV = float2(
        Remap(i.worldPos.x, 0, _LevelWidth, 0, 1),
        Remap(i.worldPos.y, 0, _LevelHeight, 0, 1)
);

half4 Fog = tex2D(_FogOfWar, UV);
```

The first part creates UV coordinates based on the vertex world position. The second part samples the global Fog texture. We now know how bright this position is supposed to be. Now all we have to do is multiply this value with the final color, metallic and smoothness. The Blurred value is stored only in the green channel of the texture:

```
o.Albedo = c.rgb * Fog.g;
o.Metallic = _Metallic * Fog.g;
o.Smoothness = _Glossiness * Fog.g;
```

Shader adaption guide (2017.1 and later)

The latest update features an update to the way ULTIMATE FOG OF WAR is implemented. Follow this guide to convert Shaders if you are using Unity 2017.1 or later. Template: "UFoW/Template/UltimateFogOfWarStandard_TEMPLATE"

Surface Shaders

Step 1: Include FogOfWarMath.cginc

```
#include "UnityCG.cginc"
```

Make sure that you get the path correct or else Unity can't find the file.

Step 2:

Include the Shader keywords for effects and vertical mode:

```
#pragma multi_compile __  VerticalMode
#pragma multi_compile __ FoWColor FoWAnimatedFog
```

Step 3:

Add the World Position (float3 worldPos;) to the shader's input struct. It should look like this:

```
struct Input {
        float2 uv_MainTex;
        float3 worldPos;
};
```

Step 4:

Go to the Shaders surface function (Usually called 'surf') and add the following line to get the color of the Fog:

```
fixed4 FoW = FoWIntensity(IN.worldPos);
```

Then Multiply the result with the Albedo output:

```
o.Albedo = c.rgb * FoW.rgb;
```

CG Vertex and Fragment Shaders (5.4)

Step 1: Include FogOfWarMath.cginc

```
#include "UnityCG.cginc"
```

Make sure that you get the path correct or else Unity can't find the file.

Step 2:

Add the global variables used to sample the fog:

```
uniform sampler2D _FogOfWar;
uniform float _LevelWidth;
uniform float _LevelHeight;
```

Step 3:

Add the World Position (float3 worldPos;) to the shader's output struct. It should look like this:

```
struct v2f
{
	float2 uv : TEXCOORD0;
	UNITY_FOG_COORDS(1)
	float4 vertex : SV_POSITION;
	float3 worldPos : TEXCOORD1;//FOG OF WAR
};
```

Step 4:

Calcualte the World Position in the Vertex function (usually named 'vert'):

```
o.worldPos = mul(_Object2World, v.vertex);//FOG OF WAR
```

Step 5:

Go to the Shaders fragment function (Usually called "frag") and add the following lines, if you are making Horizontal Shader:

```
float2 UV = float2(
	Remap(i.worldPos.x, 0, _LevelWidth, 0, 1),
	Remap(i.worldPos.z, 0, _LevelHeight, 0, 1)
);


half4 Fog = tex2D(_FogOfWar, UV);
```

Or Add these lines if you are making a Vertical Shader:

```
float2 UV = float2(
        Remap(i.worldPos.x, 0, _LevelWidth, 0, 1),
        Remap(i.worldPos.y, 0, _LevelHeight, 0, 1)
);

half4 Fog = tex2D(_FogOfWar, UV);
```

The first part creates UV coordinates based on the vertex world position. The second part samples the global Fog texture. We now know how bright this position is supposed to be. Now all we have to do is multiply this value with the final colour. The Blurred value is stored only in the green channel of the texture. *The R-Channel stores the current visibility without fading/ blurring.*

```
        return col * Fog.g;
```

CG Vertex and Fragment Shaders (2017.1 and later)

Step 1: Include FogOfWarMath.cginc

```
#include "UnityCG.cginc"
```

Make sure that you get the path correct or else Unity can't find the file.

Step 2:

Include the Shader keywords for effects and vertical mode:

```
#pragma multi_compile __  VerticalMode
#pragma multi_compile __ FoWColor FoWAnimatedFog
```

Step 3:

Add the World Position (float3 worldPos;) to the shader's output struct. It should look like this:

```
struct v2f
{
        float4 vertex : SV_POSITION;
        float2 uv : TEXCOORD0;
        UNITY_FOG_COORDS(1)
        float3 worldPos : TEXCOORD2;//FOG OF WAR
};
```

Step 4:

Calcualte the World Position in the Vertex function (usually named 'vert'):

```
o.worldPos = mul(unity_ObjectToWorld, v.vertex);//FOG OF WAR
```

Step 5:

Go to the Shaders fragment function (Usually called "frag") and add the following line, to get the color value of the fog.

```
fixed4 FoW = FoWIntensity(i.worldPos);
```

Finally, add this line to add the Fog to the output color. Make sure, that you add this at the end to be sure, that it does not get lost during other shader calculations.

```
return col * Fog.g;
```

ULTIMATE FOG OF WAR - API

FogOfWarManager.cs

Public Variables

Int LevelWidth	Map width on X-Axis
Int LevelHeight	Map width on Z-Axis
Texture2D Resistence	Resistence Texture used to calculate the Fog of War. R: Static Blockers, G: LevelHeight, B&A: Free for expansions.
Vector2 StartEndHeight	The Minimum and Maximum Height of the Map. This is used to define if a Revealer is blocked from seeing a patch.
Color RevealOpacities	Fog opacity for three states: R = Revealed, G = Covered, B = Undiscovered.
Bool ShowBlocker	Should a vision blocking patch be revealed?! This does +1 Vision on all Blockers.
Int RevealFaction	All Revealers that belong to the Faction defined here, will reveal the Map. Negative values will reveal all Revealers. This is useful for Replays.
Float RevealSpeed	The speed at which the Fog will disappear when revealed.
Float CoverSpeed	The Speed at which the Fog will reappear if not revealed anymore.
FogOfWar.FogQuality BlurFog	Can be disabled to optimize performance.
Int UpSample	The Size of the Blur Texture. The Size of this RenderTexture is equal to the level size multiplied by this value. This increases the blur quality.

Bool UseThreading	Should the Fog be revealed asynchronously? If you deactivate this, all calculations are done in the main Thread. *Threading is not supported by Unity on WebGL*.
FilterMode FogTextureFilterMode	Unity texture filter mode. This allows to filter the Fog. Set this to 'FilterMode.Point' for a pixelated style.

Public Functions

void InitializeMaps()	Before you can use the Fog, call this function to set up the Fog of War Manager. This is only needed, if you create a map from code.
void ShowFaction(int Faction)	Change the revealing Faction. See RevealFaction.
void SetResistenceMapData(Color[] ResistenceMapData)	This sets a Buffer manually if you create a map procedurally from code.
void SaveFogMaps(string _SaveName)	Save the current state of the fog at this location: **Application.persistenDataPath+"/saves/"+_SaveName** .
void LoadFogMaps(string _SaveName)	Load a state from this location: **Application.persistenDataPath+"/saves/"+_SaveName** .
void UpdateOnce(string _SaveName)	This function calculates the fog if "Automatically Update" is disabled.
public FogOfWar.Players GetFactionViewMask(int _Faction)	Get the Reveal Mask of a Faction
public void SetFactionViewMask(int _Faction, FogOfWar.Players _Mask)	Set the Reveal Mask of a Faction

public void ChangeHardBlockerTo(Vector3 _Position, int _Width, int _Height, int ID)

Add a Hard Blocker to the Resistance Map. The Position is mapped to a position on the Grid. Width and height define how many patches of the grid are being set. the Position is in the center. If necessary, you can give a blocker an ID. 0 is no blocker. 1-255 are blocking ID's.

FogOfWar.cs

Public Enums

FogEffect	This allows you to change the Fog appearance. *See Fog Effect.*
MapSizes	Used in dropdown to define the Map size. You can change this if you need different map sizes, but keep in mind that it needs to be power of two!
FogQuality	Blur fog or do nothing.

Public Variables

-

Public Functions

Bool IsPositionRevealedByFaction(Vector3 Position, int Faction)

Is this position revealed by Faction? The position will be floored to the nearest int value.

void RegisterRevealer(Revealer Revealer)

Register a Revealer OnEnable. For Example when a Unit was added to the Game.

void UnRegisterRevealer(Revealer Revealer)

Unregister a Revealer OnDisable. For Example when a Unit was destroyed.

void RegisterVisionBlocker(GameObject gameObject)

Register a Vision Blocker OnEnable.

void UnRegisterVisionBlocker(GameObject gameObject)

Unregister a Vision Blocker OnDisable.

Color[] GetFogMapData(int _Faction)

Get the current state of the Fog. Useful for creating a custom save file.

void SetFogMapData(int _Faction, Color[] _Data)

Set a Fog state from custom Data.

Revealer.cs

Public Variables

float VisionRange	How far can this Revealer see? *Add +0.75 to the distance to make the view circle smoother.*
FogOfWar.Players Faction	The Faction of this Unit. This is used to determine if it reveals or not.
float UpVision	How far up can this Unit see? A Patch will be revealed, if the height of the patch this Revealer is standing on + upvision is bigger than the patch that is being tested.
GameObject RevealerObject	The GameObject (and Transform) of the Revealer.

Public Functions

-

Performance and Limitations

The Following variables have an impact on performance:

1. Level Size

Make the Level as big as necessary, but as small as possible.

2. Amount of Revealers

3. Vision Range

The bigger the vision range, the more calculations the FoV solver has to do.

4. Dynamic Blockers

5. Amount of Factions

The more factions you have and the more factions are revealed in the background have an impact on performance. Turn reveal in Background off (Factions tab) to save performance if possible.

The FoV Solver is currently confined to X&Z Axis / X&Y Axis.

Code Example(s)

The package contains examples that you can use to make your own game. If you have questions or suggestions about UFoW and it's usability, please contact support@maxproude.com.

ChangeFaction.cs

CreateFoWManagerAtRuntime.cs

ExampleUnit.cs

EcampleVisionBlocker.cs