FluidFlow 2 Documentation

Christian Schott (mr3d.cs@gmail.com)

May 15, 2022

Contents

1 Introduction

- AssetStore: `https://u3d.as/1vef`
- All documentations:
  `https://drive.google.com/drive/folders/1YzdllLIZMiyoiQXxa5WZUSETRrSyIQAs`
- Videos: `https://youtube.com/playlist?list=PLH6XAuNbhBuukHLbeF1FId0-m2FzyJZiP`
- Forum:
  `https://forum.unity.com/threads/fluid-flow-2-realtime-fluid-simulation.678286/`

2 Frequently Asked Questions

Feel free to send me your questions, so I can add them here (mr3d.cs@gmail.com).

Does FluidFlow support URP, HDRP, etc.?
Yes, FluidFlow is largely independent of the rendering pipeline your project is using. However, for overlaying the fluid texture you will need a shader, supported by your rendering pipeline, which is able to overlay the fluid texture.

How can I increase the speed at which the fluid flows?
In each fluid simulation step, the fluid moves from one pixel to the neighboring pixels depending on gravity. Therefore, the maximum fluid speed is limited by one pixel per simulation step. The trivial way for altering the fluid speed is decreasing the fluid texture resolution, so each pixel is larger, or increasing the update rate of the fluid simulation. However, keep in mind, that the latter has a performance cost, so you should not go too far with this approach.

Does FluidFlow work on mobile platforms?
Yes, FluidFlow was successfully tested on a Google Pixel 3a, and should also be able to run on older devices. iOS based devices should also work, however, I am not able to test this myself. I would greatly appreciate it, if you share your experiences using FluidFlow on Apple devices.

How can I access/safe the generated textures?
You can access the internal texture, by accessing the Texture Channels property on your FF-Canvas. E.g. 'myCanvas.TextureChannels["_FluidTex"]'. As the texture data is stored inside a RenderTexture, it does only exist in GPU memory. If you want to access the texture data from C#, or save the texture, you will have to read it to CPU memory first. FluidFlow allows you to call the 'myRenderTexture.RequestReadback()' method, to simplify this process. Check the 'SaveTextureChannel()' function in 'Scripts/Extensions/FFCanvasExtensions.cs' for a more elaborate example.

3 Setup

A minimal example for setting up FluidFlow in your project:

- Add a FFCanvas component to an object in your scene.
- Add the renderer of the object you want to draw on, to the render targets of the FFCanvas, and set up the render target as described in section 4.1.1.
- Set which texture channels of the renderers you want to draw on, as described in section 4.1.2.

- Assign a material to your render targets, that supports drawing to the selected texture channel. FluidFlow provides two basic shaders (one created in shader graph and a surface shader) that support drawing to the '_MainTex', '_NormalTex', and '_FluidTex' texture properties.
- Create an empty gameobject close to the surface of the renderer, and add a 'SimpleDrawSphere' component. Set up the reference to the FFCanvas, set the desired texture channel name, and assign a FFBrush.
- If everything is set up properly, you should see all pixels within the set radius around this brush gameobject being colored, when the left mouse button is pressed, while the game is running.
- To add fluid simulation to the texture channel, add a FFSimulator component, and set the references to the FFCanvas and target texture channel.
- Make sure the precision of your texture channel is set to a floating-point type, and set the brush type of the FFBrush to fluid.
- At runtime, the texture channel should now flow, like a fluid, down sloped surfaces. Check the documentation of the FFSimulator for more information.

4 Components

For detailed information on each components' properties, please also refer to the tooltips when hovering over the inspector fields.

4.1 FFCanvas

The FFCanvas component is the main component of the asset. It manages the RenderTextures used for drawing, and sets up the renderers for displaying the textures.

In order for the drawing to work properly each mesh, that is being drawn on, requires a bijective UV map, meaning that there is a 1:1 mapping between the object's surface and its texture coordinates. Consequently, the UV islands must not overlap, and it is advised (and required for proper fluid simulation) that there is at least a 2 pixel distance between UV islands.

Lightmapping UVs fulfill these requirements. Unity can automatically generate such a secondary lightmap UVs, by enabling the 'Generate Lightmap UVs' option, in the models import settings. However, it has to be noted that the 'quality' of automatically generated lightmap UVs is lower than a set of handcrafted UVs, as UV seams might be in very prominent places of the model. FluidFlow can deal with UV seams (see FFSeamFixer), but it is impossible to completely hide all artifacts from UV seams. Especially for fluid simulations, UV seams can be noticeable, as fluid has to be teleported between UV islands (see FFSimulator).

4.1.1 Render Targets

A single FFCanvas can be used to draw on multiple renderers in a scene simultaneously. All renderers inside one FFCanvas share one texture for each texture channel. To ensure that the UV maps of the renderers do not overlap, a texture atlas is used, which splits the full resolution of the texture channel between all renderers. Currently, only equal sized atlas tiles are supported. Meaning that the possible number of atlas tiles increases quadratic (e.g. $1 \times 1 = 1$, $2 \times 2 = 4$, $3 \times 3 = 9$, ...). For example, when using 2 tiles, the texture has to be split into 4 equal sized tiles, so half of the texture remains unused.

Hint: click the 'Open Preview' button in the 'Advanced' section, for a preview of the internal texture atlas layout. This preview also marks overlapping areas of the UV layout with a red color.

When drawing the texture channels on the renderer, the shader of the renderer's material has to select the proper texture atlas tile for the given render target. Additional information about that, can be found in section 7.

For each render target, a subset of its sub-meshes can be selected. This allows excluding parts of renderer from the canvas.

Additionally, the UV set used for drawing can be selected for each renderer. When the UV0 of a renderer contains overlapping UV islands, its lightmap UV (UV1) can be used instead

However, using the UV1 of a renderer introduces a problem, as unity's tangent space is relative to the UV0 of the meshes. This causes problems for normal mapping and gravity calculations of the FFSimulator. FluidFlow solves this problem by saving a UV0-UV1 tangent space transformation matrix in the UV2 channel of the renderers meshes, when the render target is set to use the UV1 channel. For this purpose, the renderer's mesh is duplicated and swapped out on initialization of the FFCanvas, and the UV0-UV1 transformation is calculated for this duplicated mesh. For complex meshes, this can cause a considerable overhead when initializing the FFCanvas, therefore each render target has an optional FFModelCache field. This FFModelCache allows generating and caching this mesh with UV0-UV1 transformation once, before the game is started, and sharing this mesh between multiple FFCanvases.

Each render target can also specify, if it should be placed in its own texture atlas tile, or if it should be placed in the same tile as the render target before it. This might be useful in special cases where the UV sets of multiple meshes are non-overlapping, so they can be combined in a single tile, optimizing texture usage. In most cases, 'New Tile' should be used.

4.1.2 Texture Channels

Each FFCanvas can be used for drawing on multiple texture channels. Each texture channel corresponds to one RenderTexture with the specified resolution internally.

A texture channel specifies which texture property of the renderers' materials it controls. The texture property is linked by its name in the render target's materials (e.g. '_MainTex' or '_NormalTex'). By clicking on the button with the three dots next to the text field, a popup with all texture property names of the current render targets is shown.

When a single FFCanvas controls multiple Render Targets with different materials it can happen, that the texture property names of the materials do not match, even though they should correspond to the same Texture Channel. E.g. one might be called '_MainTex' and the other '_AlbedoTex'. In this case you can define '_AlbedoTex' as an alias for '_MainTex', by appending it to the Texture Property name, separated by a '|'. E.g. '_MainTex|_AlbedoTex'. The Texture Channel will still be identified just by '_MainTex'.

The format of the texture channel is specified by the number of components (1, 2, or 4), and their precision. The values in a fixed-point (FIXED8, FIXED16) texture are limited between 0-1, while floating-point (HALF, FLOAT) textures can also represent values > 1 and < 0. This is relevant for the fluid simulation, when a pixel should be able to store > 1 amount of fluid. The button, next to the precision field, provides a few presets for different use cases.

Each texture channel can be initialized in two main ways. With a single default color (BLACK, GRAY, WHITE, BUMP, RED), similar to defining a Texture2D property in a shader (`https://docs.unity3d.com/Manual/SL-Properties.html`), or by copying (COPY) the current contents from the texture property of renderers' materials.

When the texture channels are initialized, they are assigned corresponding textures of the renderers using MaterialPropertyBlocks (`https://docs.unity3d.com/ScriptReference/MaterialPropertyBlock.html`). So do not be confused, if the textures do not show up in the material's inspector fields, as they are directly assigned to the renderers.

4.1.3 Advanced

By enabling the 'Texture Border Padding' checkbox, one pixel of padding around each texture atlas tile is added. This can prevent color bleeding between atlas tiles, and allows proper UV seam stitching for the fluid simulation, when the seam lies on the texture border.

When the render targets are initialized, the FFCanvas automatically sets the corresponding atlas transformation vector on each material of the render targets. Additionally, a shader keyword is set, when the render target uses UV1. The default names for this property and keyword can be overwritten for each FFCanvas. This is needed, when a single renderer is used by multiple FFCanvases, as the atlas transformation and UV set of the render target will probably differ between them. Therefore, different shader properties and keyword names have to be used. However, for most applications, the default values will be sufficient.

4.2 FFSimulator

The FFSimulator component allows to add a gravity-based flowing effect to a texture channel of a FFCanvas. Texture channels used for fluid simulation are required to have four components (RGBA). Red, green, and blue for color, and alpha to store the fluid amount.

In order to improve performance, the FFSimulator can be set to halt simulation when currently no renderer is visible, or time out after the FFCanvas has not been changed for a set time.

As the fluid is simulated in texture space, the seams between UV islands pose a problem. FluidFlow handles this by teleporting pixel values between corresponding UV edges. In order for this to work, at least a single pixel wide space around each UV island is required. To link the UV seams together, the renderers' meshes need to be preprocessed. For complex meshes, this can add some overhead to the initialization of the FFSimulator. This processing step can also be precomputed in a FFModelCache, and assigned to the corresponding render target of the FFCanvas.

Also note, that texture channels linked to a FFSimulator do not require a FFSeamFixer for fixing artifacts from UV seams, as the FFSimulator deals with UV seams directly.

4.2.1 Gravity

The scene's gravity is baked into an internal flow map. In order to allow changes in the renderers rotation or skinned deformations to alter the flow of the fluid, the flow texture has to be re-baked. This can be done automatically in a set time step (CONTINUOUS, FIXED), or manually (CUSTOM). Updating the flow texture manually, via the 'UpdateGravity()' method, allows to only update the flow map when necessary, potentially improving performance.

When the render targets' materials contain a normal map texture property, it can be set to influence the generation of the flow map. This allows the fluid to react to small changes in the model's surface.

4.2.2 Fluid

Similar to the generation of the flow map, the fluid simulation can also be set to update in a set time step (CONTINUOUS, FIXED), or manually (CUSTOM) via the 'UpdateFluid()' method. When trying to achieve a constant flow of the fluid, the FIXED update mode should be used. For example, using an update interval of 0.025seconds equates to an update rate of 40Hz, which should result in a smooth flow of the fluid.

Additionally, it can be specified, how much fluid each pixel can retain. If a pixel contains more fluid than this specified amount, the excess is moved in the local gravity direction. Note that when the precision of the fluid texture is set to be a fixed-point value in the FFCanvas, and the fluid retain amount is > 1, there will never be any excess fluid to flow. Fluid simulation using fixed-point fluid textures is still possible, by using a fluid retain amount < 1. In order to make < 1 fluid values appear 'solid', the fluid amount can be scaled back up in the renderers' shaders. Floating-point fluid textures do not have this problem, but not all platforms support them (presumably only very old or mobile platforms).

4.2.3 Evaporation

Evaporation allows reducing the overall fluid amount of the entire fluid texture over time. The update time step can again be CONTINUOUS, FIXED, or CUSTOM (via the 'UpdateEvaporation()' method).

When timeout is enabled for the FFSimulator, an additional timeout can be set for the evaporation updates. This allows the more expensive fluid updates to stop, while the evaporation is still active.

4.3 FFSeamFixer

When drawing on a FFCanvas only pixel $> 50\%$ inside the UV islands are painted. Sampling this texture in a material later, will cause parts of these unpainted pixels being sampled, causing visible UV seams. FluidFlow fixes this problem by expanding the color inside the UV islands to the pixels surrounding the UV islands.

The FFSeamFixer component allows to specify which texture channels of a FFCanvas this seam fix should be applied to. In a specified update interval (CONTINUOUS, FIXED), or manually (CUSTOM) using the 'FixModifiedChannels()' method, it checks which of the specified texture channels have been modified since the last update and fixes them.

Optionally, parts of this fixing process can be cached in an internal texture to reduce future texture samples. However, this creates a single channel 8-bit render texture, which consumes additional video memory.

Note that texture channels used together with a FFSimulator do not require to be linked to a FFSeamFixer, as the FFSimulator already deals with the UV seams.

5 ScriptableObjects

5.1 FFModelCache

The FFModelCache scriptable object can be used to precalculate necessary initialization operations for models in the editor and cache the results for the initialization of the FFCanvas and FFSimulator. A FFModelCache can be created in the editor by selecting an imported model in unity's Project window and clicking 'Assets/Create/Fluid Flow/Model Cache'.

In the inspector of the FFModelCache, it can be specified which meshes of the targeted model should be cached, by adding new entries to the list. When the UV field of an entry is set to UV0 and the seam stitch mask is set to 'Nothing', the entry is ignored, as nothing has to be cached in this case. If any entry for a specific mesh has the UV1 flag set, a copy of this mesh with the UV0-UV1 transformations precalculated is generated and saved. Setting the 'Stitch Submesh Mask' to anything other than 'Nothing' causes the stitch data being generated for the specified mesh with the specified UV set. To generate stitch data for multiple subsets of submeshes, multiple list entries with the same mesh and UV set can be added.

The cached data from all loaded FFModelCache scriptable objects is loaded during the start of the game to a global cache manager internally. To ensure a FFModelCache is included in the build, it either has to be put in a 'Resources' folder in the project, or be referenced by an object in the scene. So in theory, a FFModelCache only has to be referenced once by a FFSimulator, and the cached data can still be accessed by all other FFSimulator components in the game. However, as a best practice, the FFModelCache should be referenced by all FFSimulator components which access the data, to make sure the FFModelCache is included in the final build.

In the 'Global Settings' section, an update of all FFModelCache objects in the project can be triggered. This checks for each FFModelCache if the model it targets has been modified, and if so recalculates the cache. This updating of the caches can also be scheduled each time before entering the play mode, by enabling the 'Auto Update Before Play' toggle. However, please note that this adds a small overhead to entering the play mode.

5.2 FFDecalSO

The FFDecalSO is a scriptable object wrapper around a FFDecal. It allows predefining a FFDecal in the editor for later use in-game. It can be created via 'Assets/Create/Fluid Flow/Decal'.

5.3 FFBrushSO

The FFBrushSO is a scriptable object wrapper around a FFBrush. It allows predefining a FFBrush in the editor for later use in-game. It can be created via 'Assets/Create/Fluid Flow/Brush'.

6 Drawing

In order to draw on a texture channel of a FFCanvas a set of extension methods for the FFCanvas are provided. There are currently two main approaches for drawing:

- **Decal projection**: a decal, defined by a FFDecal, is projected on the render targets' surface. The projection is defined by a projection-view matrix, similar to a camera, that describes a projection volume or frustum in world-space. (`https://docs.unity3d.com/ScriptReference/Matrix4x4.Perspective.html`). To project a decal on a FFCanvas, call the 'ProjectDecal()' extension method.
- **Brushes**: a brush is a 3D shape in the scene (e.g. a sphere) and the parts of the render targets' surface intersecting this shape are painted. How exactly the texture is painted is defined using a FFBrush. To draw a brush on a FFCanvas, call the 'DrawSphere()', 'DrawDisc()', or 'DrawCapsule()' extension methods.

For example code snippets on how these methods can be used, take a look at 'FluidFlow/Example/Scripts/DrawExamples.cs'.

6.1 FFDecal

A FFDecal allows drawing on multiple texture channels of a FFCanvas at once. Each decal channel is identified by the shader property name of the texture channel it affects. Additionally, setting the decal channel type to FLUID allows setting the fluid amount. Setting the type to NORMAL sets a flag internally, so normal maps are projected properly.

The FFDecal also has an optional mask field. When the mask texture is left empty, no mask is applied. If a mask texture is set, all decal channels are affected by it. The R, G, B, and A buttons allow specifying which channels of the mask texture will be used for calculating the mask value. The mask value for each pixel is calculated by adding the values from the selected color channels, and dividing by the count of selected channels. For example, when selecting the green and alpha channel, the mask value $v = \frac{Mask.g+Mask.a}{2}$.

6.2 FFBrush

A FFBrush can currently only be used for drawing a solid color (triplanar texture mapping might be added in the future). By setting the brush type to FLUID, the fluid amount added can be specified.

The drawing of a FFBrush is internally implemented as a distance function from an arbitrary 3d shape. When drawing a brush, a maximum distance is specified, and all pixels within this distance from the shape are colored. The fade value multiplied by this maximum distance defines the distance from the edge of the shape, where the intensity of the brush starts decreasing. And the intensity of the brush reaches zero, when the distance from the shape is equal to the maximum distance. The interpolation function is currently fixed to a smooth-step function, which has zero for its 1st- and 2nd-order derivatives at $x = 0$ and $x = 1$.

6.2.1 Custom Brush Shape

It is possible to add custom brush shapes, by creating a new method in the 'FluidFlow/Scripts/Draw/-BrushExtensions.cs'. This requires setting the required variables as global shader values, and writing a custom shader with the distance function of the shape.

The 'FluidFlow/Resources/InternalShader/Draw/CustomBrushTemplate.shader' provides a template for such a custom brush shader, which only requires you to fill in the distance function. To call this custom brush shader from C#, you can add a new material cache to the other draw shaders in the 'FluidFlow/Scripts/Internal/InternalShaders.cs'.

7 Custom Shaders

As the FFCanvas combines multiple render targets into a texture atlas, and some renderers might use the lightmapping UVs for drawing, a special material shader is required for displaying the generated textures on the models. Depending on the rendering pipeline used, unity's shader graph can be used, or a custom surface shader can be created. FluidFlow provides helper functions for both of these options. A set of custom shader graph nodes are defined in 'FluidFlow/Editor/CustomShaderGraphNodes', and the 'FluidFlow/Shaders/FFShaderUtil.cginc' provides useful helper functions for writing surface shaders. Take a look at the included shaders, for an example of how these helper functions can be utilized.

All shaders drawing texture channels from a FFCanvas require a Vector4 shader property, containing the atlas transformation values for the render target. The FFCanvas will automatically set up this atlas transformation value on initialization. For this to work properly, this transformation property has to be called '_FF_AtlasTransform', or the FFCanvas has to override the default shader property names. Additionally, the FFCanvas will set the 'FF_UV1' (if not overwritten) shader keyword on the materials, if the render target is set to use its secondary UV set.

7.1 Shader Graph

Ensure a Vector4 property, with the reference name '_FF_AtlasTransform' and default value $(0, 0, 1, 0)$, and a local multi compile boolean keyword 'FF_UV1' is defined.

Using the custom 'FF Atlas Transformed UV' node, and passing in the atlas transform vector, the texture coordinate for sampling a FFCanvas texture channel can be calculated. This automatically selects the proper UV set, depending on the 'FF_UV1' keyword, and transforms it to the proper atlas tile. In order to transform a texture coordinate to the atlas manually, use the 'FF Atlas Transform' node.

When sampling a fluid texture, use the custom 'FF Unpack Fluid' node, for easy access to the fluid's color, normal and fluid amount information. If the fluid texture is drawn using the secondary UV set, use the 'FF Transform Normal From UV1' to convert it back to UV0. This is necessary, because unity expects normal vectors to be relative to UV0 tangent space.

7.2 Surface Shader

The steps for creating a FluidFlow compatible surface shader:

- Add a vector material property named '_FF_AtlasTransform' with the default value $(0, 0, 1, 0)$. You can add a '[HideInInspector]' property flag, as you will not have to adjust this value manually.

- Include the 'FFShaderUtil.cginc'. Note that when your shader does not lie in the same directory as the 'FFShaderUtil.cginc', you will have to specify the relative path (e.g. '#include "../../FluidFlow/Shaders/FFShaderUtil.cginc"').
- When defining the surface shader pragma, you will have to add a vertex shader, as FluidFlow has to set up some internal values in the vertex shader (e.g. "#pragma surface surf Standard fullforwardshadows **vertex:vert**").
- Create a local multi compile shader variant for using the secondary UV set ("#pragma multi_compile_local __ FF_UV1").
- Add the 'FF_SURFACE_INPUT' macro to the surface shader's input struct.
- Define the vertex shader, and ensure the name matches the one defined in the surface shader pragma definition. The header of the vertex shader function should be similar to 'void vert(inout appdata_full v, out Input o)'. Inside the vertex function, call 'UNITY_INITIALIZE_OUTPUT(Input, o);' to initialize the input from unity's side. Additionally, call the 'FF_INITIALIZE_OUTPUT(v, o, _FF_AtlasTransform);' macro, to initialize the FluidFlow specific values. Notice that this requires the '_FF_AtlasTransform' value, so make sure to define 'float4 _FF_AtlasTransform;' in the 'CGPROGRAM' before the vertex function.

Inside the surface shader, you can now access the atlas transformed texture coordinate, via the 'FF_UV_NAME' field on the input struct. This field was initialized by the 'FF_INITIALIZE_OUTPUT' macro in the vertex shader.

When sampling a fluid texture, you can use 'FF_FLUID_COLOR' to access the color of the fluid, and 'FF_FLUID_AMOUNT' to access the fluid amount (these are just basic convenience defines to the RGB-/A-values of the sampled color).

Use the 'FFUnpackFluidNormal' function to calculate a normal vector for a fluid texture. When the fluid texture is painted to the secondary UV set of a model, use the 'FFTransformNormalFromUV1' function to convert it to UV0 tangent space. This is necessary, because unity expects normal vectors to be relative to UV0 tangent space. 'FF_TRANSFORM_NORMAL' is a convenience macro, that transforms the normal automatically when the 'FF_UV1' keyword is set, and does nothing otherwise.