Description

Fast Decals allows the game to render a large amount of flat decals - such as blood splatters and blob shadows - at a blazing fast speed, with a single draw call. It also provides an automatic aging mechanism, which slowly fades away old decals when new ones are created. Fast Decals was created mobile devices in mind and is best suited to games with a small world, such as tower defense games etc. Eventhough Fast Decals was created for rendering decals in 3d nothing stops you from using it for 2d tile/sprite rendering.

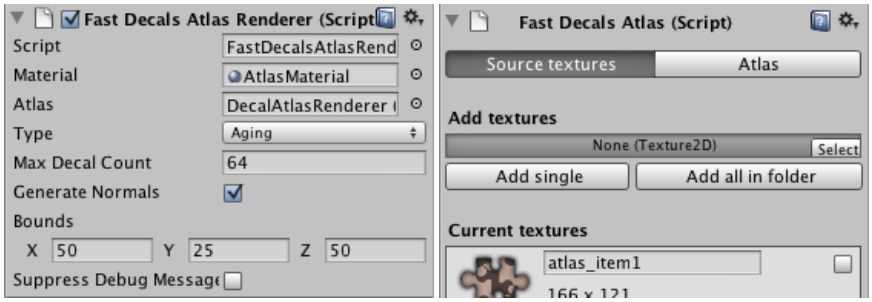
Version 1.5 now comes with built in texture atlas generation support so you are no longer required to create your own atlases. Just drag and drop your textures and the atlas will be generated automatically. However you can still use your own atlases the same way than before.

Renderers

There are two different renderers, FastDecalsRenderer and FastDecalsAtlasRenderer. The FastDecalsAtlasRenderer offers a simple way to create a texture atlas from a bunch of textures. Having your decals in a texture atlas is a requirement for fast rendering. If you want to create your texture atlases by hand you may use the FastDecalsRenderer instead.

Basic usage

1. Import the Fast Decals package.
2. Create an empty GameObject. Drag either FastDecalsRenderer or FastDecalsAtlasRenderer on the GameObject depending if you have created a texture atlas by hand or you want to use the automatic atlas generation tool.
3. Select desired rendering mode from Alpha, Degenerate, BlockCopy, ArrayCopy and Aging. Use Aging mode for static decals such as blood splatters and after explosion marks on the ground which may slowly fade away in time and Alpha, Degenerate or BlockCopy mode for dynamic stuff like blob shadows. You may read more about rendering modes in Rendering Modes chapter.
4. Define the material to be used for rendering the decals. For the shader you could use something from the Mobile / Particles group or create your own. For the Aging and Alpha modes the shader must support vertex colors. If your shader supports lighting you should set Generate Normals to true. It is recommended to add your decals in a texture atlas instead of creating many FastDecalsRenderers. You should only create a new renderer when you need to draw the decals with a different shader or when your atlas gets too big. In most cases you should set the offset to 0/0 and tiling to 1/1.
5. If you are using the **FastDecalsAtlasRenderer**: After dragging the FastDecalsAtlasRenderer to your GameObject you will notice that it will also have a component called FastDecalsAtlas which will contain the source data for your renderer. To create the texture atlas you should simply drag and drop your textures to the textures slot and click Add single to add it to the atlas. You may also add all textures inside the same folder by clicking the Add all in folder instead of dragging them all one by one. You may anytime add more items or remove items and the atlas will update automatically. You may also name the items to make them easier to access from your scripts. By default the name is stripped from the texture asset filename. By clicking the Atlas button on the component toolbar you will see a preview of the generated atlas and you may also change the atlas texture format, padding and it's maximum size.



1. If you are using the **FastDecalsRenderer**: If you are using a grid based texture atlas you can automatically create texture offsets/tiling data to your decal renderer by defining the horizontal and vertical tile count. When you draw a decal you can simply reference to the decal index. As an example you could set the horizontal and vertical tile count to 8 and use 1024x1024 atlas texture which would allow you to have 64 different decals of size 128x128. Incase you need even bigger atlases, please check your supported device list and their maximum texture size. Most of the newer device support 2048x2048 and some even 4096x4096. It is much better to have a bigger texture atlas than having many FastDecalsRenderers. Incase you want to use custom texture coordinates instead of the grid based, set the horizontal and vertical tile count to 1 and pass the texture offset and texture tiling when actually drawing the decals.

1. Define the maximum number of decals which you will not exceed. It probably does not matter much if you define 64 or 128 in performance but 5000 is a different story. Smaller is faster.
2. To actually draw a decal you should get a reference to your renderer and then call DrawDecal function in your Update function. Not in FixedUpdate since it might not be in sync with the rendering and not in LateUpdate since FastDecals will process the draw requests inside it.

```
    public void DrawDecal(Vector3 pos, float size, int index)
    public void DrawDecal(Vector3 pos, float size, int index, Quaternion rotation)
    public void DrawDecal(Vector3 pos, float size, int index, Vector3 normal)

    public void DrawDecal(Vector3 pos, float size, int index, Color32 color)
    public void DrawDecal(Vector3 pos, float size, int index, Color32 color, Quaternion rotation)
    public void DrawDecal(Vector3 pos, float size, int index, Color32 color, Vector3 normal)

    public void DrawDecal(Vector3 pos, float size, Vector2 textureOffset, Vector2 textureTiling)
    public void DrawDecal(Vector3 pos, float size, Vector2 textureOffset, Vector2 textureTiling, Quaternion rotation)
    public void DrawDecal(Vector3 pos, float size, Vector2 textureOffset, Vector2 textureTiling, Vector3 normal)

    public void DrawDecal(Vector3 pos, float size, Vector2 textureOffset, Vector2 textureTiling, Color32 color)
    public void DrawDecal(Vector3 pos, float size, Vector2 textureOffset, Vector2 textureTiling, Color32 color, Quaternio
n rotation)
    public void DrawDecal(Vector3 pos, float size, Vector2 textureOffset, Vector2 textureTiling, Color32 color, Vector3 n
ormal)

    public void DrawDecal(Vector3 pos, Vector2 size, int index)
    public void DrawDecal(Vector3 pos, Vector2 size, int index, Quaternion rotation)
    public void DrawDecal(Vector3 pos, Vector2 size, int index, Vector3 normal)

    public void DrawDecal(Vector3 pos, Vector2 size, int index, Color32 color)
    public void DrawDecal(Vector3 pos, Vector2 size, int index, Color32 color, Quaternion rotation)
    public void DrawDecal(Vector3 pos, Vector2 size, int index, Color32 color, Vector3 normal)

    public void DrawDecal(Vector3 pos, Vector2 size, Vector2 textureOffset, Vector2 textureTiling)
    public void DrawDecal(Vector3 pos, Vector2 size, Vector2 textureOffset, Vector2 textureTiling, Quaternion rotation)
    public void DrawDecal(Vector3 pos, Vector2 size, Vector2 textureOffset, Vector2 textureTiling, Vector3 normal)

    public void DrawDecal(Vector3 pos, Vector2 size, Vector2 textureOffset, Vector2 textureTiling, Color32 color)
    public void DrawDecal(Vector3 pos, Vector2 size, Vector2 textureOffset, Vector2 textureTiling, Color32 color, Quatern
```

```
ion rotation)
    public void DrawDecal(Vector3 pos, Vector2 size, Vector2 textureOffset, Vector2 textureTiling, Color32 color, Vector3
normal)
```

The Draw function will take the decal position, size and tile index as parameters. The size can be defined as a float when you want to want the decal to be a square or with Vector2 when you want to be able to stretch it. You may also rotate the decal by giving a rotation quaternion or by giving a normal the decal will be automatically aligned with it. When rotation or normal are not defined the decal will be aligned on to the ground and the y rotation is randomized.

When you use the FastDecalsRenderer and a grid based texture atlas you should give the requested tile index. For example when using 4x4 grid you may request index between 0 and 15. Incase you are using a none grid based texture atlas you can pass the texture offset and tiling as parameters. When you are using the FastDecalsAtlasRenderer you can retrieve the tile index by name from your FastDecalsAtlas.

The color is defined in Color32 instead of Color since Unity documentation say that it will perform better. For the Aging and the Alpha mode the shader should take the final alpha from the vertex colors. Fast Decals provides a couple of high performance example shaders inside the package.

Incase you want to suppress all Fast Decals log messages, just check the Suppress Log Messages check button on your FastDecalsRenderer. Log messages are automatically silenced in the release build.

Accessing the atlas items

When you are using the FastDecalsAtlasRenderer you will be also using FastDecalsAtlas. You can access atlas texture coordinates by texture name or texture index. When you are rendering a decal you can just pass the texture index to the FastDecalsAtlasRenderer instead of manually retrieving and passing the uv coordinates from the atlas. Your FastDecalsRenderer will contain a public reference to the atlas which also allows you to share atlas between multiple renderers.

```
public int GetIndexByName(string name)
public bool GetTexCoordsByIndex(int i, out Vector2 textureOffset, out Vector2 textureTiling)
public bool GetTexCoordsByName(string name, out Vector2 textureOffset, out Vector2 textureTiling)
public string GetNameByIndex(int i)
```

For best performance you should always cache the texture index when initializing your scene instead of requesting the index by name for each update call.

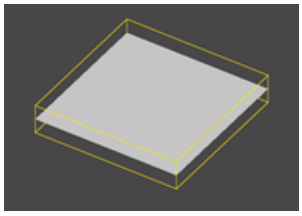
```
public class IndexCachingExample : MonoBehaviour {
    public FastDecalsAtlasRenderer atlasRenderer;
    private int mBlobShadowIndex;

    void Awake() {
        mBlobShadowIndex = atlasRenderer.atlas.GetIndexByName("BlobShadow");
    }

    void Update() {
        atlasRenderer.DrawDecal(transform.position + Vector3.up * 0.01f, 2.0f, mBlobShadowIndex);
    }
}
```

World Space and Local Space

By default the decals are always rendered in World Space. So even you change the position or rotation of your renderer it does not affect your decals. Sometimes you might want the decals to inherit parent position, scale and rotation. You can enable Local Space from a dropdown menu in your Fast Decals renderer inspector view.

Bounds and frustum culling

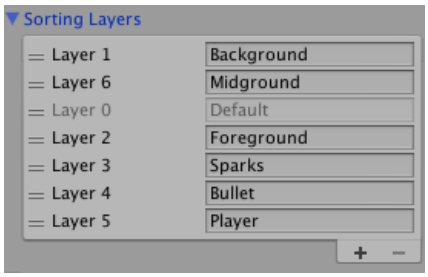
Like any other renderers in Unity, FastDecalsRenderer is only rendered when the renderer bounds are inside/intersecting the view/camera frustum. You may change the bounding box size in your FastDecalsRenderer inspector view. The position of the bounding box is always the position of the game object that you have attached your renderer on.



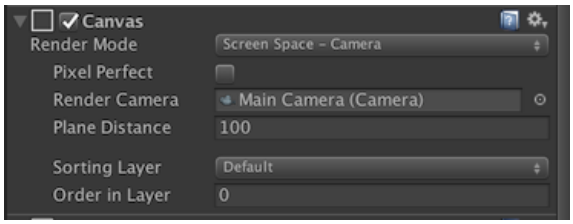
Depending of your game you may need to change the bounds of your renderer. For games with a small world you can simply set your renderer position to the center of your world and the bounds to be roughly the same size than your world. The current renderer bounds are shown with a yellow box inside the editor scene view. In some cases, like fps games and particle rendering you may need to manually update the bounds to be inside the camera view on every frame. This can be most easily achieved by simply parenting the renderer with the used camera.

Using Fast Decals in a 2d project

You can use Fast Decals together with Unity sprites/UI however you must keep in mind that Fast Decals is always drawn on Default sorting layer. Currently there is no way to change the Fast Decals sorting layer but you can re-order your sorting layers by dragging them in the sorting layers menu.



Another thing you probably need to change is your Canvas rendering mode. By default the Canvas uses Screen Space - Overlay mode and that causes everything to be drawn in front of Fast Decals. To make Fast Decals visible you must change the Canvas rendering mode to Screen Space - Camera and drag your camera to the camera field of the Canvas.



Rendering modes

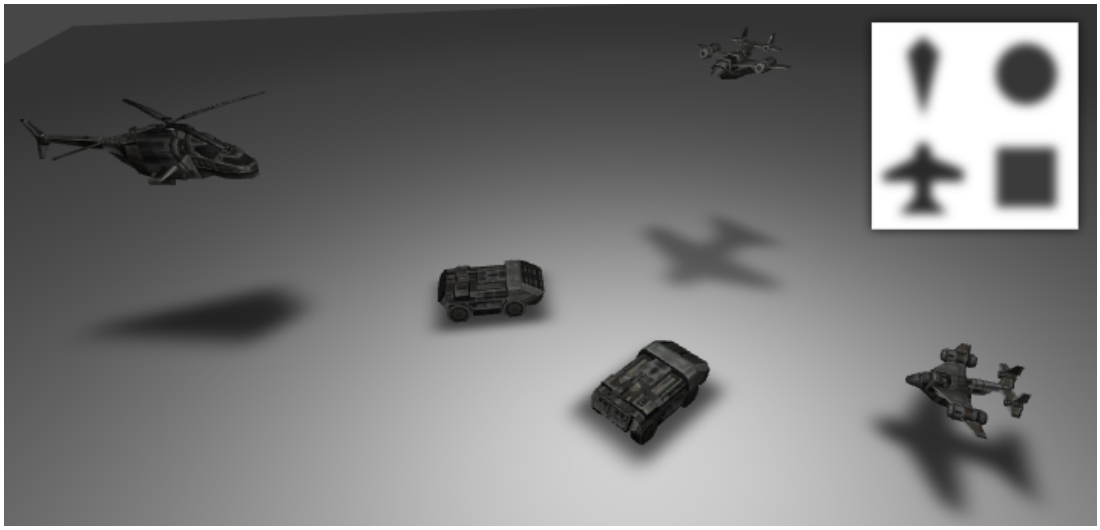
FastDecalsRenderer creates a single mesh from selected amount of decals. The mesh is dynamically changed. The rendering mode defines how the old/unvisible decals are hidden. Alpha, Degenerate, BlockCopy and ArrayCopy modes basically behave the same. They all draw decals with lifetime of one frame. BlockCopy and ArrayCopy just takes a bit more from the CPU but less from the GPU.

FIFO and Aging modes are persistent modes where the decal does not need to be redrawn every frame. The FIFO mode will just override the oldest decal when the maximum decal count has been reached. In the Aging mode the decals will slightly fade everytime a new decal is drawn. In both modes the decals can be cleared by calling the ClearDecals method.

The Degenerate mode is probably the fastest and it hides the old/unvisible decals by feeding incompatible triangle indices to the GPU which will cause the triangle to degenerate. In alpha mode the old decals are hidden using vertex alpha so the used shader must support it. In BlockCopy and ArrayCopy modes only the currently visible triangles are sent to the GPU. You may benchmark the modes to see which mode performs the best in your game. By using native code Fast Decals could be even faster but since you can only use managed code in Unity Indie this is how the magic happens now.

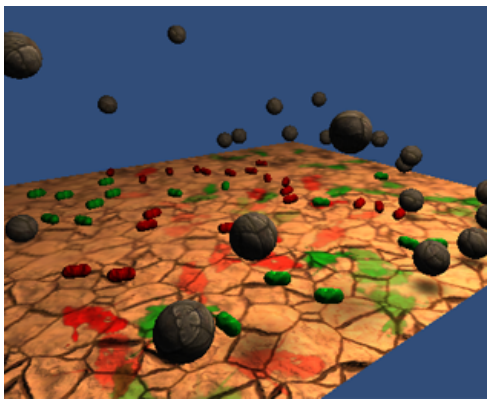
Another usage example

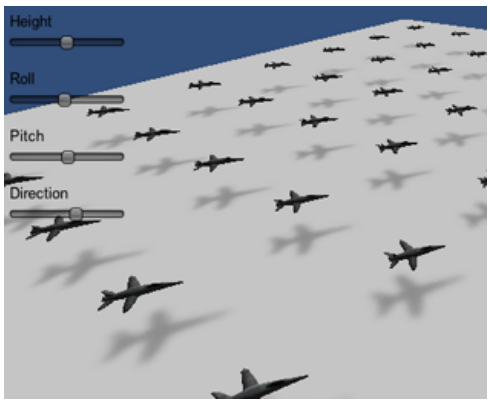
Faking shadows is a great way to boost your mobile game performance.

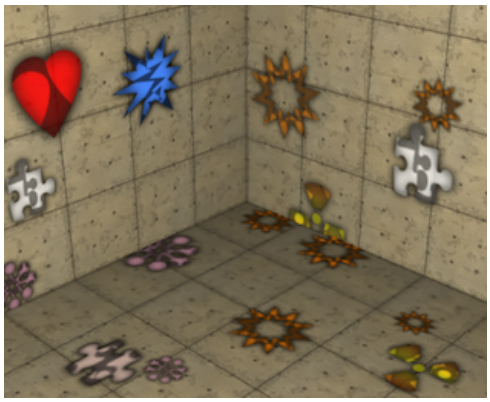


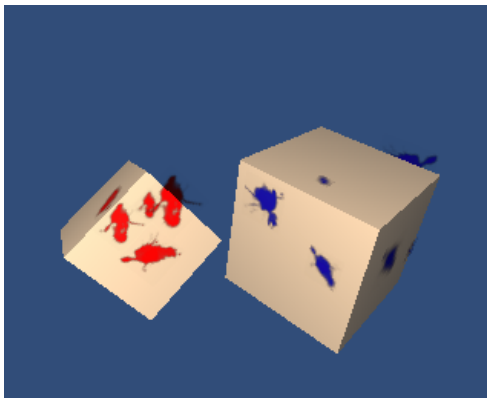
Provided example scenes

There are four different example scenes inside the Fast Decals package. Bounce scene demonstrates blood splatter rendering in Aging mode and blob shadow rendering Degenerate mode. FakeShadows scene shows an easy way to fake fighter jet shadows. The EasyAtlas scene demonstrates the new FastDecalsAtlasRenderer which will automatically create the texture atlas from provided textures. The LocalSpace scene shows how to draw decals on moving objects and how to share a texture atlas between multiple renderers.









Support

Incase you are having problems with Fast Decals, don't hesitate to contact **support@pmjo.org**