**Upgrading to Unity 5.2**

**升级到Unity 5.2**

**Global Illumination**

**全局照明**

UV packing for baked UVs not filling the 0–1 space (smaller or bigger) has been fixed. It makes the resolution assigned to each object work much more reliably if that object’s unwrap is not filling the 0–1 space and also when its bounds are non-square. Please review the resolution on your instances for baked lightmaps.

修复了对烘焙后的UV进行UV打没有处于0-1空间（更小或者更大）的问题。如果object的拆包没有处于0-1空间，或者它的边界不是一个方形，这会使得分配给每一个object的分辨率更可靠。请在检查烘焙贴图中实例对应的分辨率。

Shader variant stripping was fixed for realtime lightmaps. Now each lightmaps mode (non-directional, directional and directional specular) variant can be picked for baked and realtime GI separately. Please review your settings if you previously selected a specific lightmaps mode variant in the Graphics Settings to make that mode work for realtime lightmaps.

着色器变体（Shader variant）剥离在实时光照帖图中被修复了。现在每一个光照贴图模式（非定向，定向以及定向镜面）变体都可以分别选择烘焙和实时GI。请检查设置如果你之前在图像设置中选中了一个指定的光照贴图模式变体，来使得这个模式能够在实时光照贴图下工作。

Bounce scale has been changed from the arbitrary value of 0.7 to 1.0. The bounce is the product of the albedo and bounce scale. Artists should set real-life albedo values (the brightest non-metallic is snow with 0.9). This is our PBS reference [http://forum.unity3d.com/threads/official–5–0-pbr-calibration-charts.289416/](http://forum.unity3d.com/threads/official-5-0-pbr-calibration-charts.289416/)

反弹比例从任意值改为了0.7到1.0。 反照率和反弹比例决定了反弹行为。美术应该设置真实反照率的值（最明亮的非金属是雪的0.9）。这是我们的PBS参考 [http://forum.unity3d.com/threads/official–5–0-pbr-calibration-charts.289416/](http://forum.unity3d.com/threads/official-5-0-pbr-calibration-charts.289416/)

Since you should author physically correct albedo, it makes sense for us to set the scale close to 1. We already clamp albedo values in the meta pass, so the bounce scale should just be 1.0f.

Please note that if you choose to set albedo to 1.0 in a custom meta pass without clamping, then the scene can look like it’s exploding with light.

如果你设置了物理正确的反照率，那么比例（scale）设置为接近1会比较合理。我们已经在meta pass中固定了反照率的值，所以反弹比例就应该是1.0。注意如果你选择在自定义的meta pass中将反照率设置为1.0且没有固定，那么场景就看起来像是曝光一样。

**Shaders**

**着色器**

“Fixed Function” style shaders (the ones that use SetTexture, Lighting On etc.) internally get turned into actual shaders at shader import time now. Upside is that they now work on all platforms (previously did not work on consoles), and with more consistency. Also a lot of code and fixed function related inefficiencies got removed from runtime, making rendering a bit faster. Downside is, creating fixed function shaders at runtime - using new Material(fixedFunctionShaderString) - does not work anymore. That constructor was deprecated in Unity 5.1, and now in 5.2 it actually stopped working for fixed function shaders.

“固定功能”类型的着色器（那些使用SetTexture，LightingOn等的）现在在导入的时候内部会变成真的着色器。好处是他们现在在各个平台上都能工作（之前在控制台不能工作）并且更一致。很多代码以及的固定功能相关的低效元素也都在运行时移除了 -使用new Material(fixedFunctionShaderString)- 不再管用。它的构造器在Unity5.1中是不推荐的，而现在在5.2中则是完全停止了固定功能着色器的工作。

**Reflection Probes**

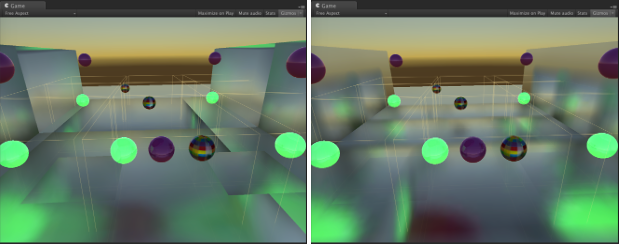
**反射探针**

We’ve changed how Reflection Probes are rendered when using Deferred Shading, in order to allow “screen space reflections” effects in the future. Short version is: in deferred shading, reflection probes are per-pixel instead of per-object now.

我们没有修改反射探针在延迟着色时的渲染方式，为了未来能够实现“屏幕空间反射”的效果。短期的版本是：在延迟着色中，反射探针是以像素而不是物体为单位的。

*Comparison of current behavior (reflection probes per object; in some cases hard to avoid harsh reflection transitions between large objects) and reflection probes per pixel (transitions much less visible; and they happen at probe boundaries not at object boundaries):*

比较现在的机制（以物体为单位的反射探针，在某些情况下很难避免在大型物体中粗糙的反射转换）和逐像素的反射探针（转换不太明显，并且只在探针的边界处会发生问题而不是物体的边界处）：



Before (5.0 and 5.1)

之前（5.0和5.1）

* Reflection probes are sampled during the G-buffer pass, in exactly the same way as in forward rendering. They are written into “emission” buffer together with light probes, lightmaps and emissive material parts.
* 反射探针在G-buffer通道中采样，这和正向渲染完全一样。他们和光照探针,光照贴图和放射性材质零件（emissive material parts）一起被写进emission缓冲区中。
* This meant you get one (or two, when probe blending is on) reflection probes per object.
* 这意味着每个对象都使您获得一个反射探针（或者是两个如果探针开启了混合的话）
* Reflections being together with emission/lightmaps in the same buffer means that doing SSRR “properly” is hard. SSRR provides reflections by itself (and falls back to reflection probes where it can’t), but it does not know which part of “emission buffer” color is coming from reflection probes.
* 反射行为和放射/光照贴图在同一个缓冲区中意味着很难适当地去执行SSRR(screen-space raytraced reflections屏幕空间光线追踪反射)。SSRR自身提供了反射（）但是他不知道从反探针中获取的是放射缓冲区颜色的哪一部分。

Now (5.2)

现在（5.2）

* When using deferred shading, do not sample reflection probes during G-buffer pass.
* 当使用延迟着色时，不再通过G-buffer通道来执行反射探针的采样。
* Instead, after the G-buffer is done, make a separate “deferred reflections” pass that draws reflection probes as boxes in screenspace; that output reflection information into a separate render target.
* 而是在G-buffer完成之后，生成一个独立的“延迟反射”通道来绘制反射探针，就像是屏幕空间中的箱子。
* [Future: SSRR effect will use this separate reflections buffer]
* 未来：SSRR效果会使用这个独立的反射缓冲区
* Combine reflections buffer & emission buffer at the end.
* 最后将反射缓冲区和放射缓冲区合并

What does this mean? *(everything below only affects deferred shading)*

这意味着什么（下面的东西仅仅影响延迟着色）

* **Reflection probes are no longer per-object**; they are effectively per-pixel. It is easier to have large objects affected by many reflection probes.
* **反射探针不再是以物体为单位的了**；他们是更有效的以像素为单位的。这会让大型物体被很多反射探针影响时更简单。
  + Also probes got a “blend distance” which defines how much space around the probe is used for blending into other probes.
  + Smaller probes “override” larger ones.
* **Reflection probe Renderer flags (probe blending, etc.)** are ignored; everything is affected by reflection probes in the same way (since it happens in screenspace now). This is very similar to how “receive shadows” flag is ignored in deferred shading.
* **反射探针渲染标识（探针混合等）**被忽略了；所有物体以同样的方式被反射探针影响（自从现在它出现在屏幕空间）。这和“接受阴影”标识在延迟着色中忽略非常相似。
* Your custom-written shaders that do deferred shading should mostly continue to work (in the worst case, they will be sampling a black reflection cubemap). If you do some totally crazy stuff in shaders, they might need to be updated to work with deferred shading in 5.2.
* 您自己撰写的执行延迟着色的着色器大部分都应该能继续工作（最糟糕的情况是他们会采样一个黑色的反射立方体映射（reflection cubemap））。如果您在着色器中做了一些非常疯狂的事情，他们可能需要更新以便能在5.2的延迟着色下工作。
* If you are using custom deferred shading light pass shader (with custom BRDF etc.), you’ll probably want to use custom deferred reflections shader too, with the same BRDF applied to reflection probes.
* 如果您正在使用自定义的延迟着色光照通道着色器（同时使用了自定义的BRDF（双向反射分布函数Bidirectional Reflectance Distribution Function）等），你可能会想要使用自定义的延迟反射着色器，将同样的BRDF应用到反射探针中。

**Shuriken**

**飞镖**

Particles are now generated in world space, which may require an update to any custom vertex shaders. This change was made in order to allow re-use of the particle buffers between each eye for VR.

粒子现在在世界空间中产生，这可能在任意自定义的顶点着色器需要一个更新。这个改动是为能使得VR双眼之间的粒子缓冲区能够复用。

Mesh particles now support the Texture Sheet Animation module. It’s worth checking that your existing effects do not have this enabled by accident, otherwise you may see a change in behaviour.

网格例子现在能够支持纹理表动画模块。检查你现有的效果没有误开启这一项，否则你可能会看到机制上的不同。

The Dampen parameter in the Limit Velocity over Lifetime module used to have a stronger effect at higher framerates. This has been fixed, and if your game is targetting 30fps, your old effects will be unaffected by this change. However, if your game targets a differnet FPS, you can update the Dampen value using this formula, to ensure your effect is unchaged in 5.2:

Limit Velocity over Lifetime模块下的Dampen参数会在更高的帧率时产生更强的效果。这已经被修复了，如果你的游戏设置为30FPS，你之前的效果不会受到这次改动的影响。然而，如果你的游戏设置了不一样的FPS，你可以这个公式来修改Dampen的值来确保你的效果在5.2中不会变化:

newDampen = 1.0f - pow(1.0f - oldDampen, targetFPS / 30.0f);

**Graphics (Other items)**

**图形（其他项）**

Material.CopyPropertiesFromMaterial now also copies shader keywords and render queue. If you were relying on them being not copied, you’ll have to change your code.

Material.CopyPropertiesFromMaterial现在也会拷贝着色器的关键字和渲染队列。如果您依赖于他们不被拷贝，您就需要修改一下您的代码。

SpeedTree materials now need to be re-generated as there are changes to SpeedTree built-in shaders. You could do so by selecting SpeedTree prefabs in your project and hit “Apply & Generate Materials” button. *Please be noted that by doing so your customisations to the generated material assets (if any) will be overwritten.*

SpeedTree材质现在需要重新生成因为他们被修改为SpeedTree内置着色器。你可以在项目中选择SpeedTree的预设并且点击“Apply & Generate Materials”按钮。请注意这么做的时候你生成的自定义的材质资源会被覆盖。

**UI**

**用户界面**

In 5.2 we have combined the shaders that text and normal UI element rendering users. A side effect of this is that if you specify a manual font texture in a 32bit format then the color channels will be honored. This means that black texture channels will result in black text where previously the text would be white (we only looked at the alpha). If you wish to use custom textures for your fonts do one of the following:

在5.2中我们将渲染文本和用户UI元素的着色器进行了合并。这么做的一个副作用是，如果你指定了一个32位格式的自定义字体纹理，那么颜色通道就会发生错误（honored）。这意味着之前的文本是白色的时候黑色的纹理通道会导致黑色的文本（我们只查看了alpha）。如果您希望您的字体使用自定义的纹理，就执行以下行为之一：

* 1. Change the import format of the texture to A8. This will only keep the alpha component and Unity will generate the text as white by default.
* 1. 修改纹理的导入格式为A8。这只保留了alpha通道而Uniy会默认生成白色的文字。
* 2. Specify a color / colors in the texture for Unity to use when rendering the text
* 2. 在Unity渲染文字时会使用到的纹理中指定一个或多个颜色。

**Multiplayer**

**多人**

The way the project identification is handled has changed in Unity 5.2, now the project is automatically registered and you do not need to manually enter an ID anywhere. There is a Multiplayer panel in the Services Window (open it with cloud icon in upper right corner) and in there you can find a deep link directly to the project on the website (Go to dashboard). When configured the Multiplayer configuration will appear here.

项目处理唯一标识的方式在5.2中作了修改，现在项目是自动注册的，你不需要在任何地方手动的输入ID。这是一个在Services窗口中的多人面板（通过右上角的cloud图标打开）在这里你可以发现一个直接链接到该项目的网页链接（前往仪表盘）。配置了多人配置时会在这里出现。

