spine-unity 运行时文档

许可证须知

将官方的 Spine Runtime 整合到你的应用程序中需要有效的 Spine 许可证,但我们欢迎你对 Spine Runtimes 进行评估。

入门篇

安装

要在你的 Unity 项目中使用 spine-unity 运行时，你需要:

1. 下载并安装 Unity.
2. 在 Unity Editor 创建一个空白的新项目.
3. 下载最新的 spine-unity 组件包. 你也可以如下所述方法通过 Git 获得最新版.
4. 导入这个 Unity 组件包 (双击它就可以让 Unity 打开).

通过 Git 而非 unity 组件包获取最新运行时:

1. 克隆 spine-runtimes Git 库.
2. 把 `spine-runtimes/spine-unity/Assets/` 里的文件拷贝到你项目里的 `Assets/` 文件夹里.
3. 把 `spine-runtimes/spine-csharp/src` 文件夹拷贝到你项目里的 `Assets/Spine/Runtime/spine-csharp` 文件夹中.

2D Toolkit 兼容性

spine-unity 支持 2D Toolkit ,可以使用 TK2D 的 texture atlas 系统渲染 Spine 骨架 (skeletons)。要启用 2D Toolkit 兼容性,请通过 `Preferences` via `Edit -> Preferences...`打开 Unity 的偏好设置,在 Spine 部分选择 `Define TK2D` - `Enable`.

可选的 UPM 插件包

spine-unity 运行时无需额外的插件即可工作。一些可选的功能,如 timeline 或 URP 支持,是通过单独的 Unity Package Manager(UPM)扩展包提供的。

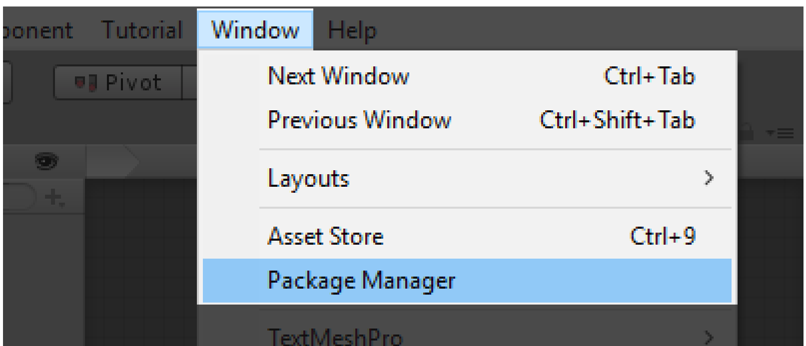
为什么使用独立的插件包

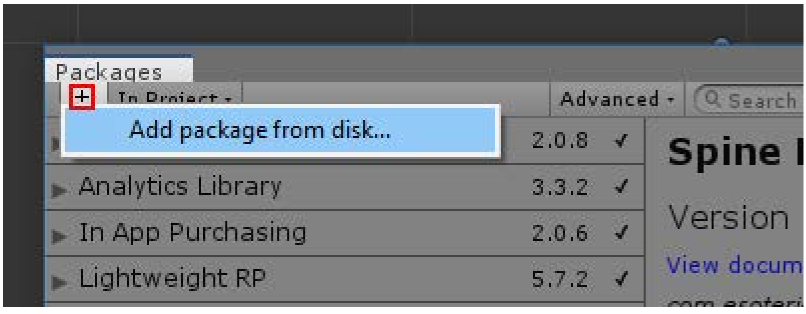
Unity 已将大量的可选插件迁移至新的 Unity Package Manager(UPM)生态系统。例如,其基于 Universal Render Pipeline 的着色器文件也是包含在 Universal RP 包中通过 UPM 提供,而不是直接包含在每个新的 Unity 项目里。

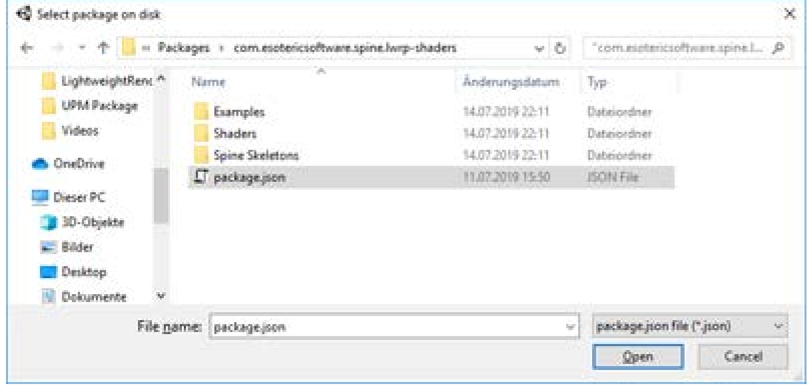
在 spine-unity 运行时中包含 Universal RP Spine shaders 会导致错误信息的混乱并引入额外配置。把 Universal RP Spine shaders 打包为 UPM 包,可自动解决这种 Unity 的 Universal RP 包尚未引入项目中的依赖性,使这种附加功能更容易使用。

安装

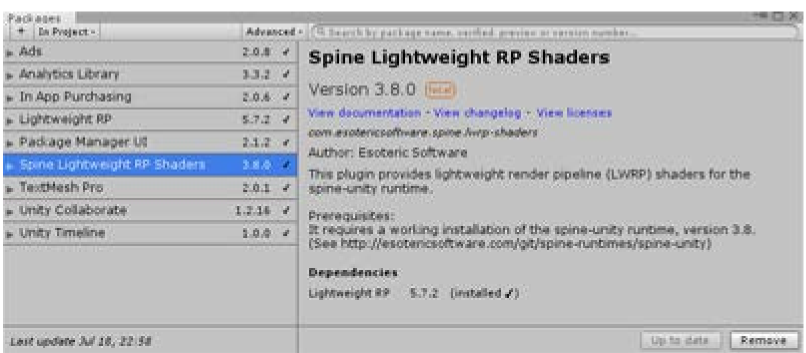
1. 通过下载页面下载所需的 UPM 包,或者在 Git 仓库的 spine-unity/Modules 子目录下找到.
2. 如果你的 Unity 项目已经打开,你可以:
    - a) 关闭 Unity 或者
    - b) 关闭任何包含 Spine 组件的场景(例如,通过打开一个新的空白场景).
3. 在解压或克隆后,你可以通过以下两种方式之一使用该软件包:
    - a) **复制到你的项目中**
      把包的内容复制到你的项目中的 Packages 目录。Unity 会自动加载它.
    - b) **用 UPM 包管理器导入**
      把包的内容复制到 Assets 目录以外的任何地方,然后在 Unity 中打开包管理器(`Window > Package Manager`),选择 +图标,选择从 `Add package from disk...`,定位并选择 `package.json` 文件.



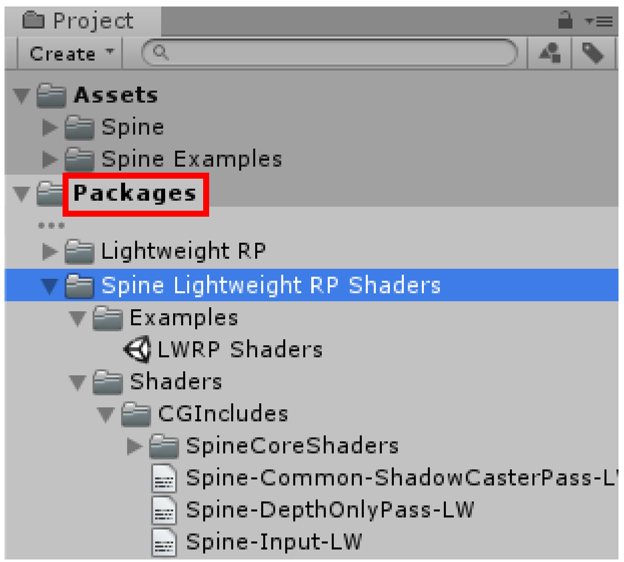




现在,包管理器窗口将列出一个 `Spine Lightweight RP Shaders` 条目:



在 `Project` 面板中,你也可以在包下找到一个 `Spine Lightweight RP Shaders` 条目。:



如果该条目没有在 `Project` 面板中列出,你可能需要关闭并重新打开 Unity.

示例场景

要探索 spine-unity 运行时的附带示例场景你可以:

1. 下载并安装 [Unity](#).
2. 在 Unity 编辑器中创建一个新的空项目.
3. 下载[最新的 spine-unity 组件包](#). 你也可以如下所述方法通过 Git 获得最新版.
4. 导入这个 Unity 组件包 (双击它就可以让 Unity 打开).
5. 在 Unity 编辑器中打开项目,然后在 `Project` 面板中查看 `Spine Examples/Getting Started` 文件夹中的不同示例场景。每个示例场景都有关于如何运行的文字说明和描述.

Getting latest changes via Git instead of the unitypackage

1. Clone the [spine-runtimes Git repository](#).

2. Copy the contents of `spine-runtimes/spine-unity/Assets/` to your project's `Assets/` folder.
3. Copy the folder `spine-runtimes/spine-csharp/src` to your project's `Assets/Spine/Runtime/spine-csharp` folder.

You can inspect and modify the C# code of both the samples and the spine-unity runtime by opening the project in Unity Editor and selecting `Assets -> Open C# Project`. For additional information on the example scenes, please see section Example Scenes.

Updating the spine-unity Runtime

Before updating your project' spine-unity runtime, please consult our guide on Spine editor and runtime version management.

Note: Json and binary skeleton data files exported from Spine 3.8 or earlier will **not** be readable by the spine-unity 4.0 runtime! The skeleton data files need to be re-exported using Spine 4.0.

If you have many projects, we suggest automating exporting your project files as described here.

For example, we use this script to export all the Spine example projects and to create texture atlases: export.sh

Please consult the following upgrade guides when updating from spine-unity 3.6, 3.7 or 3.8 to newer versions:

- spine-unity 3.6 to 3.7 Upgrade-Guide
- spine-unity 3.7 to 3.8 Upgrade-Guide
- spine-unity 3.8 to 4.0 Upgrade-Guide

It is recommended to perform the following steps to prevent potential problems:

1. As with Unity updates, it is always recommended that you back up your whole Unity project before performing an update.
2. Always check with your Lead Programmer and Technical Artist before updating your Spine runtime. Spine runtimes are source-available and designed to be user-modifiable based on varying project needs. Your project's Spine runtime may have been modified by your programmer. In this case, updating to the latest runtime also requires reapplying those modifications to the new version of the runtime.
3. Read the `CHANGELOG.md` file included in the downloaded unitypackage or on github. You can find the necessary documentation here when obsolete methods have been replaced with new counterparts.

Once you are sure you want to update to the latest spine-unity runtime:

1. Get the latest spine-unity runtime by downloading the latest spine-unity unitypackage. Alternatively you can update by pulling the latest changes from the spine-runtimes Git repository via Git as described below.

2. Close the Unity Editor and Visual Studio/Xcode.
3. When upgrading to a different major or minor version (e.g. from 3.7 to 3.8), delete the previous spine-unity installation directories `Assets/Spine` and `Assets/Spine Examples` from your project.
4. Open the project in the Unity Editor. In case you removed the previous spine-unity installation, ignore any logged errors.
5. Import the unitypackage (you can double-click on it and Unity will open it).
6. When upgrading to a different major or minor version, re-import the skeleton assets by selecting `right-click - Reimport All` in the Project panel. Alternatively to not re-import all assets, you can locate the folder containing your Spine skeleton assets in the Project panel that you want to re-import, right-click on the folder, then select `Reimport`.

Getting latest changes via Git instead of the unitypackage

1. Get the latest spine-unity runtime by pulling the latest changes from the spine-runtimes Git repository.
2. When upgrading to a different major or minor version (e.g. from 3.7 to 3.8), delete the previous spine-unity installation directories `Assets/Spine` and `Assets/Spine Examples` from your project.
3. Copy the contents of `spine-runtimes/spine-unity/Assets/` to your project's `Assets/` folder.
4. Copy the folder `spine-runtimes/spine-csharp/src` to your project's `Assets/Spine/Runtime/spine-csharp` folder.

Note: The spine-unity runtime is based on the generic spine-csharp runtime. Make sure to watch changes to both the spine-unity and spine-csharp runtime on GitHub.

更新 UPM 包插件

当升级一个可选 UMP 包插件时:

- 与更新 Spine Unity 运行时原则相同.
- 如上所述,建议在执行更新前备份你的整个 Unity 项目.

就地更新(通过 .zip 文件或 git 更新)

1. 如果你打开了你的 Unity 项目,建议 a)关闭 Unity 或 b)关闭任何包含 Spine 组件的场景(例如,通过打开一个新的空场景).
2. 将新的 UPM 包的 zip 文件或 git 目录的内容复制到现有的目录下。根据你安装 UPM 包的不同方式,目录可能是你项目中的 `project_root/Packages/package_name` 目录,或者是 `Assets` 目录外的任意目录,你通过 `Add package from disk...`来加载它.
3. 如果你已经关闭了 Unity,在 Unity 中再次打开你的项目.
4. Unity 将导入新的资产并显示一个加载进度条.

在 Unity 中编码

如果你不熟悉 C#编程和 Unity 的使用，我们建议先看看官方的 Unity 教程． 界面基础 和 编程 是很好主题．请注意,动画主题并不直接适用于 spine-unity,因为 Spine 提供了自己的动画工作流程。.

使用 spine-unity 运行时

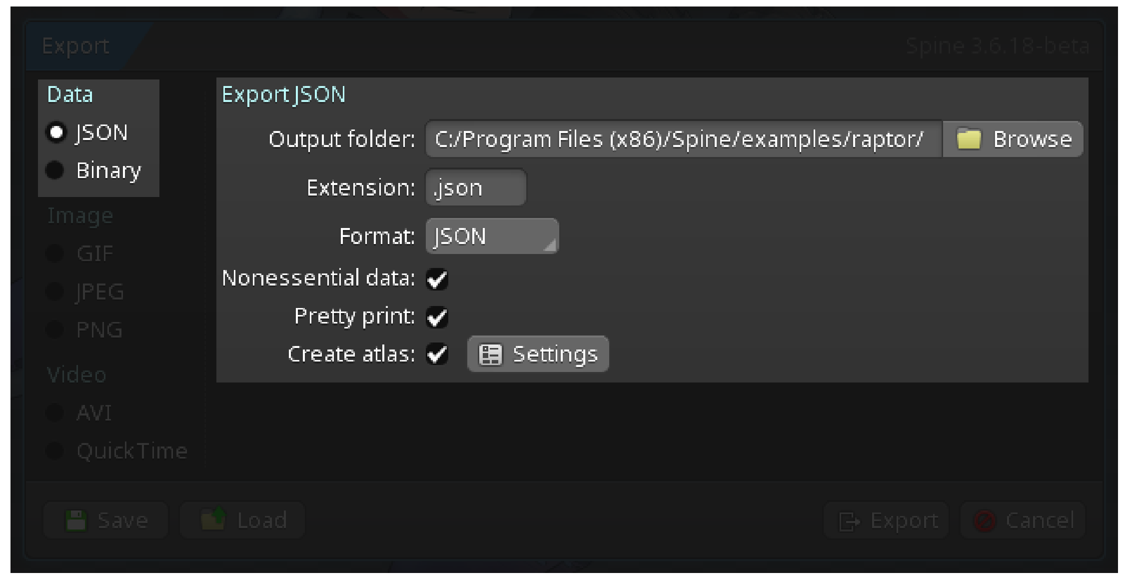
概览

spine-unity 运行时是一个 Unity 插件,用于回放和操纵 Spine 创建的动画. spine-unity 运行时是用 C#编写基于通用的 spine-csharp runtime 的运行时. spine-unity 运行时包装了 spine-csharp 的结构体和函数并将它们作为 Unity 组件暴露. 此外,spine-unity 运行时导入从 Spine Editor 导出的文件,并将其存储在自定义的 Unity 资产类型中.

你可以查阅 Spine Runtimes Guide,了解 Spine Runtime 架构的详细概述.

Asset 管理

为 Unity 导出 Spine 资产



导出指南

你可以在《Spine 用户指南》中找到完整的指示,说明如何来:

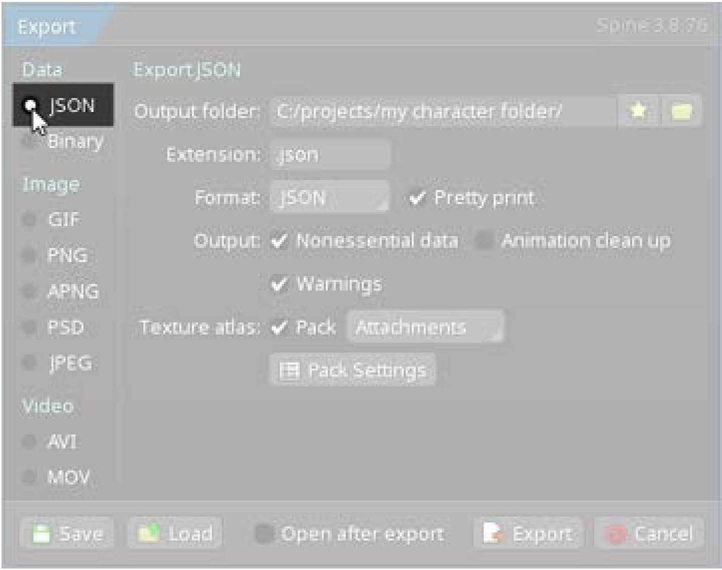
1. 导出 skeleton & animation 数据
2. 导出包含 skeleton 图像的 texture atlases

Spine 导出到 Unity

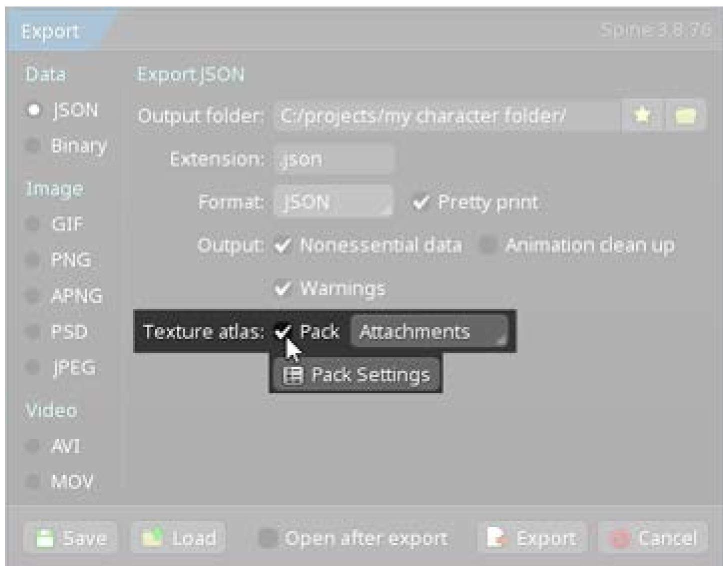
下面的步骤展示了为 Unity 导出 Spine 资产的一个简单方法.

1. 在你创建了你的 skeleton 和 animations 后，单击 `Spine Menu>Export...` (`CTRL+E`). 打开 **Export window**.

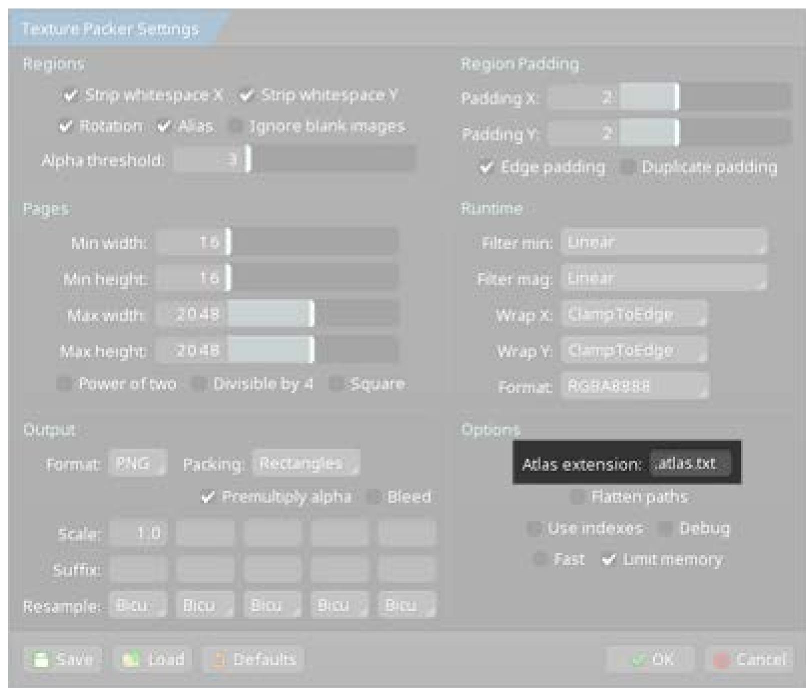
2. 选择导出窗口左上角的 `JSON` .



3. 勾选 `Texture Atlas` `Pack` . (也推荐新手们勾上 `Nonessential data` 和 `Pretty print`). 确保 `Animation cleanup` 没有启用,否则与 pose 对应的 keys 将不会被导出 .

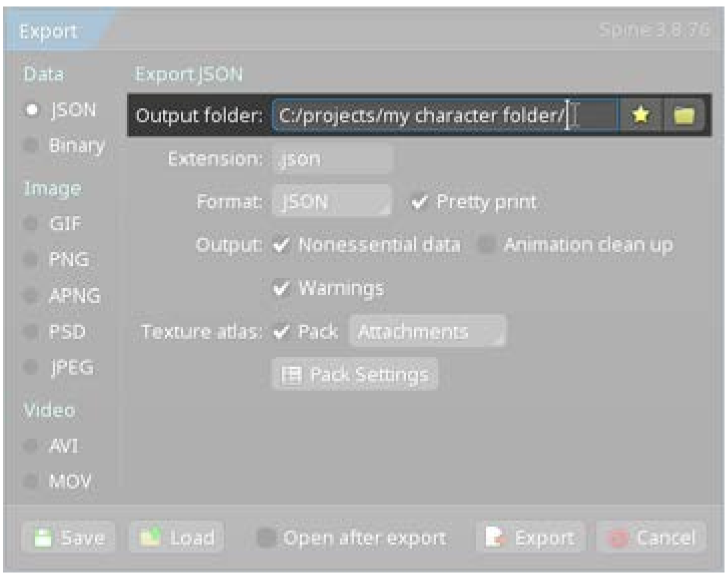


   1. 单击 `Pack` 复选框下面的 `Pack Settings`. 会打开 **Texture Packer Settings** 窗口.
   2. 左下角的 `Atlas extension` 文本框内容要设置为`.atlas.txt`.

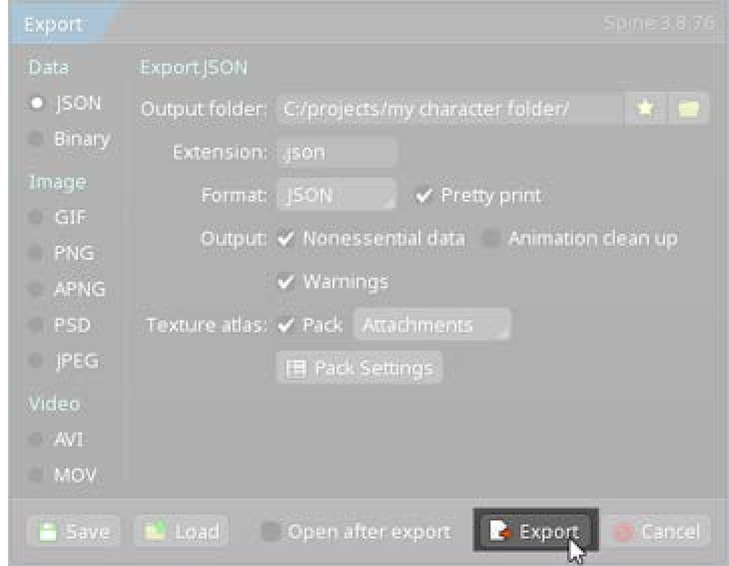


   3. 你已经完成了材质打包器的设置。点击"OK "即可关闭.

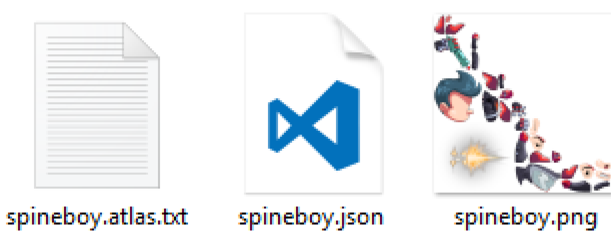
4. 在 **Export window** 中选择输出文件夹的位置. (推荐创建一个新的空文件夹来存放.)



5. 点击 `Export`.



6. 将会输出三个如下文件:



   1. `skeleton-name.json` or `skeleton-name.skel.bytes`, 包含了 skeleton 和 animation 数据.
   2. `skeleton-name.atlas.txt`,包含了 texture atlas 的信息.
   3. 一个或多个 `.png` 文件, 每个文件代表包含了 skeleton 所使用的图片包 texture atlas 中的一页.

注意: 对于 **2D Toolkit** 用户来说, 步骤 3(打包一个`.png` 和`.atlas.txt`)是没有必要的. 相反你需要在 SkeletonDataAsset 中填入适当字段,以指定对 `tk2dSpriteCollectionData` 的引用。关于如何启用 2D 工具包支持,请参考本页面的 Installation 部分.

用于 Unity 二进制导出

以二进制格式而不是 JSON 格式导出,会使文件体积更小,加载更快.

下面的步骤展示了为 spine-unity 导出二进制 Spine 资产的一个简单方法.

1. 在导出窗口的左上角选择 `Binary` 而非 `JSON`.
2. `Extension` 选为 `.skel.bytes`.

Note: sspine-unity 无法加载扩展名为`.skel` 的文件. 务必使用扩展名`.skel.bytes`.

高级- 单 Texture Atlas 文件导出和 SkeletonGraphic

一般来说,建议尽可能使用单 texture(单页)atlas,以减少由额外的子网格导致的绘制调用次数。对于 SkeletonGraphic 来说尤其如此。由于 Unity 所使用的 `CanvasRenderer` 的限制,SkeletonGraphic 在默认情况下被限制为单 texture。你可以在 `SkeletonGraphic` 的组件检查器中启用 `Advanced - Multiple CanvasRenderers`,为每个子网格生成一个子 `CanvasRenderer` `GameObject` 来突破 texture 限制。出于性能方面的考虑,最好尽可能地避免这样做。这意味着 UI 中使用的 Skeletons 应被打包成一个单 texture(单页)atlas,而非多页 atlas。

如果它们不适合单页 atlas,你可以按文件夹分组打包纹理 atlas 页。这样你可以确保每个皮肤只需要一个 atlas 页。

当图像被放置在各自的文件夹中时,你可以通过以下步骤导出 skeleton:

1. 按 `Ctrl+E` 或在下拉菜单中选择 `Export...`
2. 启用 `Texture Atlas Pack`,并选择 `Image Folder`,而不是右侧的 `Attachments`.
3. (可选步骤) 确保 `Pack Settings` 的 `Options` 右下 方 `Flatten Paths` 和 `Combine Subdirectories` 没有启用(默认设置如此).
4. 点击导出.

高级- Premultiplied 和 Straight Alpha Export

Spine 的 Texture Packer 对于如何导出 texture atlas 有两种不同工作流::

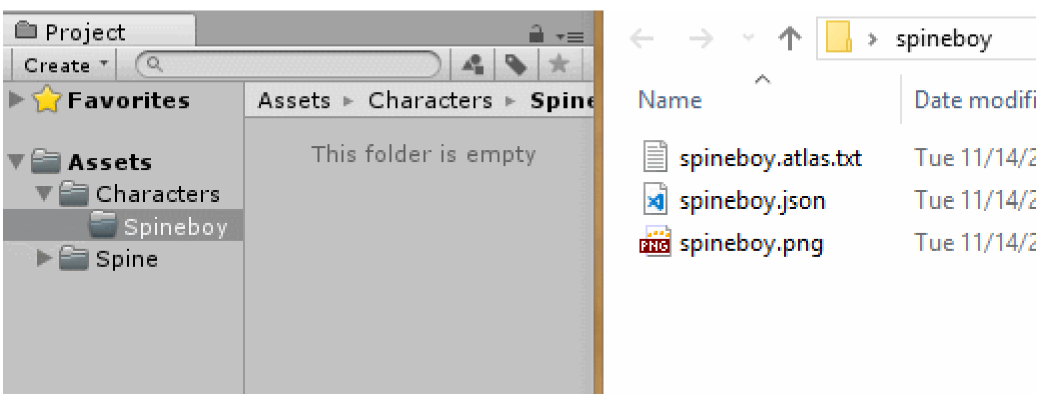
1. Premultiplied alpha (默认为此, 在 Gamma 色彩空间中 premultiplied)
2. Straight alpha

Premultiplied alpha 工作流程比 straight alpha 有一些优势,其中两个优势是: 没有额外的加性混合附件的绘制调用(draw call)和更好的 mip-map 生成。

正确地匹配导出和导入设置是非常重要的,请参阅 "高级- Premultiplied 和 Straight Alpha Import "一节,了解 Unity 中的正确导入设置。

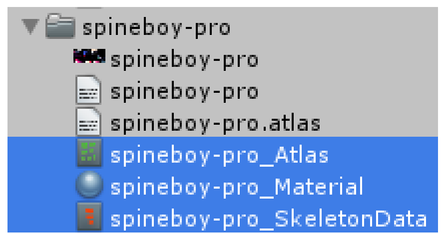
在 Unity 中导入 Spine 资产

1. 在你的 Unity 编辑器中打开 Unity 项目. 它应该已经有一个正常运行的 spine-unity 运行时,如 安装部分所述.
2. 打开导出文件的文件夹. (`.json`, `.atlas.txt` 和 `.png`)
3. 将导出的文件(或包含它们的文件夹)复制到你的项目的 Assets 文件夹中你想要的子文件夹. 你可以通过从资源管理器/搜索器窗口将导出的文件拖到 Unity 的项目面板中你想要的文件夹中来完成这一操作.



spine-unity 运行时在检测到添加的文件后会自动生成必要的额外 Unity 资产.

以下资产均为自动生成:



1. **_Atlas** 文件是 texture atlas 文件(`.atlas.txt`). 它包含对 material 和`.atlas.txt` 文件的引用.
2. **_Material** 是每个 texture atlas 页 (`.png`). 它包含对着色器和 `.png` texture 的引用.
3. **_SkeletonData** 存储了 skeleton 数据(`.json`, `.skel.bytes`). 它包含对 `.json` 或 `.skel.bytes` 文件以及生成的 **_Atlas** 资产的引用. 它还为 skeleton 提供了自定义的导入和动画设置,详见 Skeleton Data Asset 部分.

高级- Premultiplied 和 Straight Alpha 导入

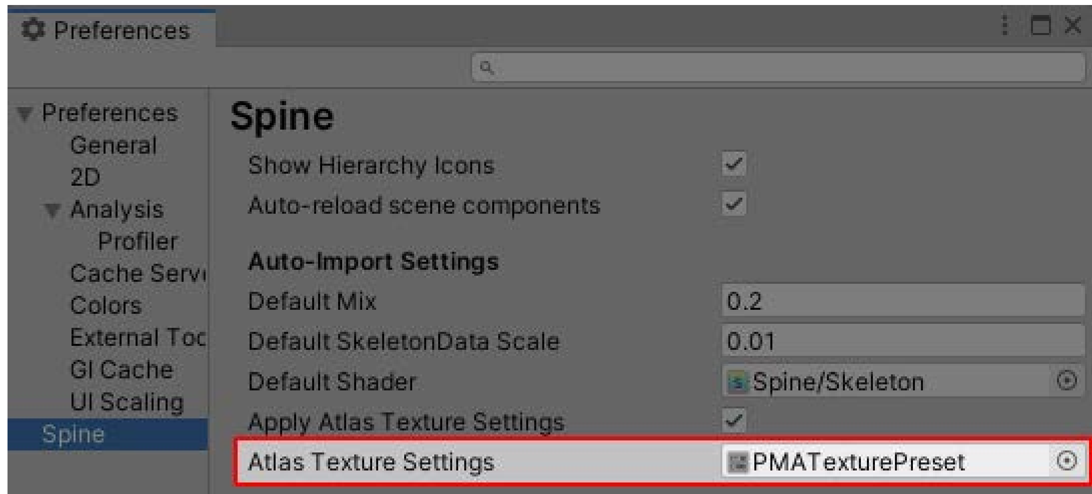
如前文 Advanced - Premultiplied vs Straight Alpha Export 所述, Spine 提供了两种 texture 导出的基本工作流:

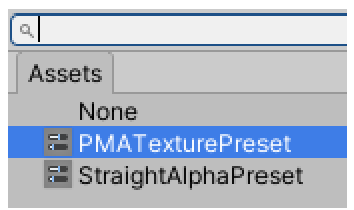
1. Premultiplied alpha (默认为此，在 Gamma 色彩空间中 premultiplied)
2. Straight alpha

用于正确导入的 `Atlas Texture Settings` 设置参数

重要提示: 非常重要的是,只要在 atlas texture 导出时启用 `Premultiply alpha` 设置,Unity 中 Material 的 Straight Alpha Texture 参数和 Texture 的 `Alpha Is Transparency` 设置都要禁用，反之亦然.

spine-unity 运行时在 Unity 的首选项窗口中提供了一个 `Spine` preferences 部分,可以通过 `Preferences` 下的 `Edit - Preferences` 访问。它提供了一个 *Atlas Texture* 设置参数,在新导入的 Atlas textures 中自动应用适当的 texture 和 materia 导入设置.



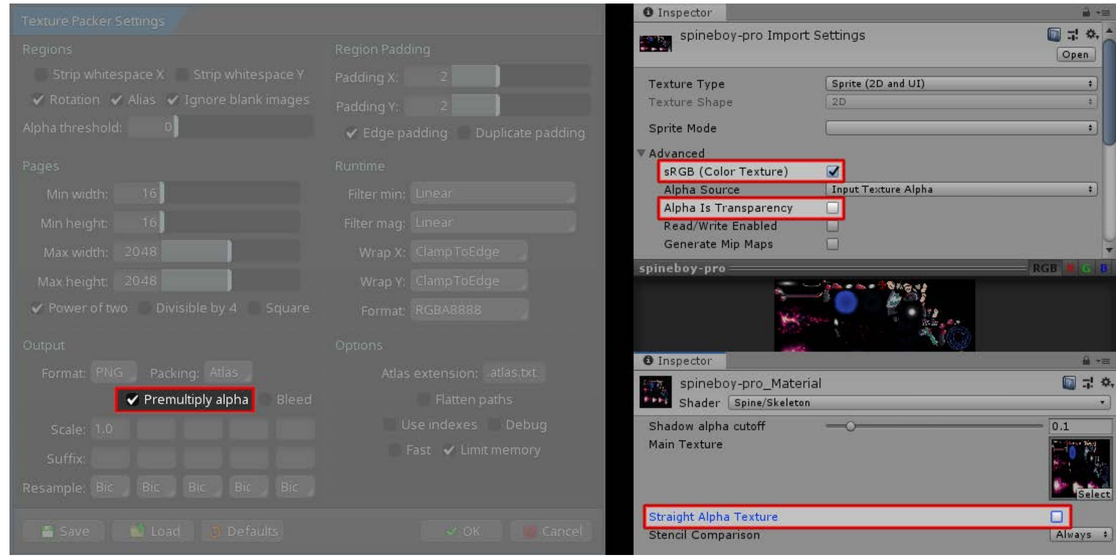


当从 Spine 导出带有启用了 `Premultiply alpha`(默认为此)的 atlas textures 时，可直接使用 `PMATexturePreset` 预设．如果你禁用了 `Premultiply alpha`，则应选择 `StraightAlphaTexturePreset` 预设．你也可以新建自己的 `TextureImporter` `Preset` 资产并赋值在此处.

每当你看到透明区域周围有深色的边框,或者图片附件周围有彩色条带,这很可能是因为导入设置不正确导致的.

正确的 Texture 打包器导出和 Texture & Material 导入设置:

1. Premultiplied Alpha

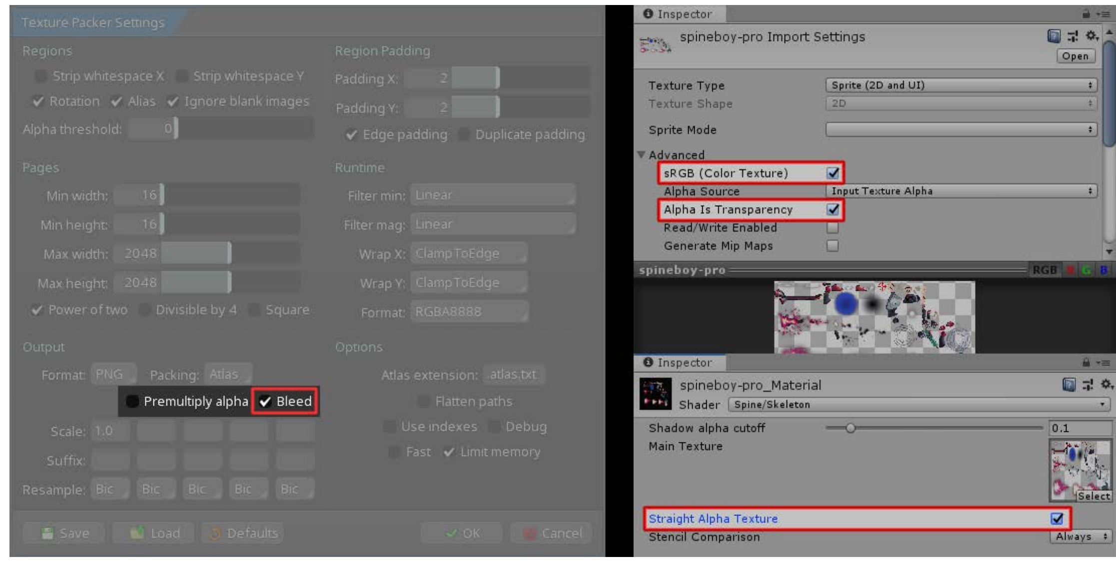


Texture 打包器启用 `Premultiply alpha`,

Unity Texture 设置启用了 `sRGB (Color Texture)` 且禁用了 `Alpha Is Transparency` ,

Unity Material 参数中禁用了 `Straight Alpha Texture` .

2. Straight Alpha



Texture 打包器禁用了 `Premultiply alpha`, 启用了 `Bleed`,

Unity Texture 设置启用了 `sRGB (Color Texture)` 且启用了 `Alpha Is Transparency`,

Unity Material 参数中启用了 `Straight Alpha Texture` .

texture packer 的默认设置是使用 `Premultiply alpha`. spine 运行时中的所有 Spine 着色器也都默认使用 `Premultiply alpha` 工作流, 其 `Straight Alpha Texture` 参数都是默认禁用的.

然而在某些情况下你可能想使用 straight alpha 工作流. 比如:

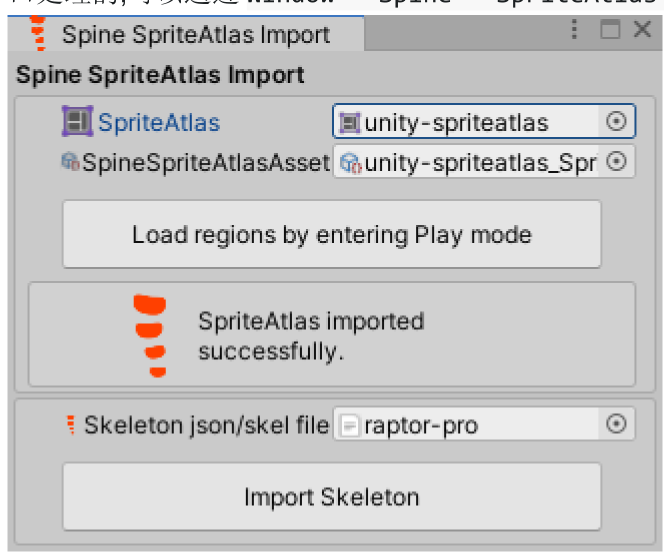
1. 当使用了**线性色彩空间**，你就**必须使用 straight alpha**.
   在导出时,premultiplication 是在 Gamma 色彩空间中进行的,在导入时转换为线性空间时将导致色彩边界错误。当在 material 里检测到这种设置组合时,你会收到一个警告日志消息.
2. 当你想使用非 Spine 着色器时.
   一般的着色器会假设作用于 straight alpha texture,这将导致图像附件周围出现错误的黑色边框.

在切换到 straight alpha 工作流时,请确保对所有 textures 和 materials 进行如上配置。 你可以通过 `Project Settings - Player - Other Settings - Color Space` 来检查或修改当前的颜色空间.

高级- 将 Unity SpriteAtlas 作为 Atlas Provider

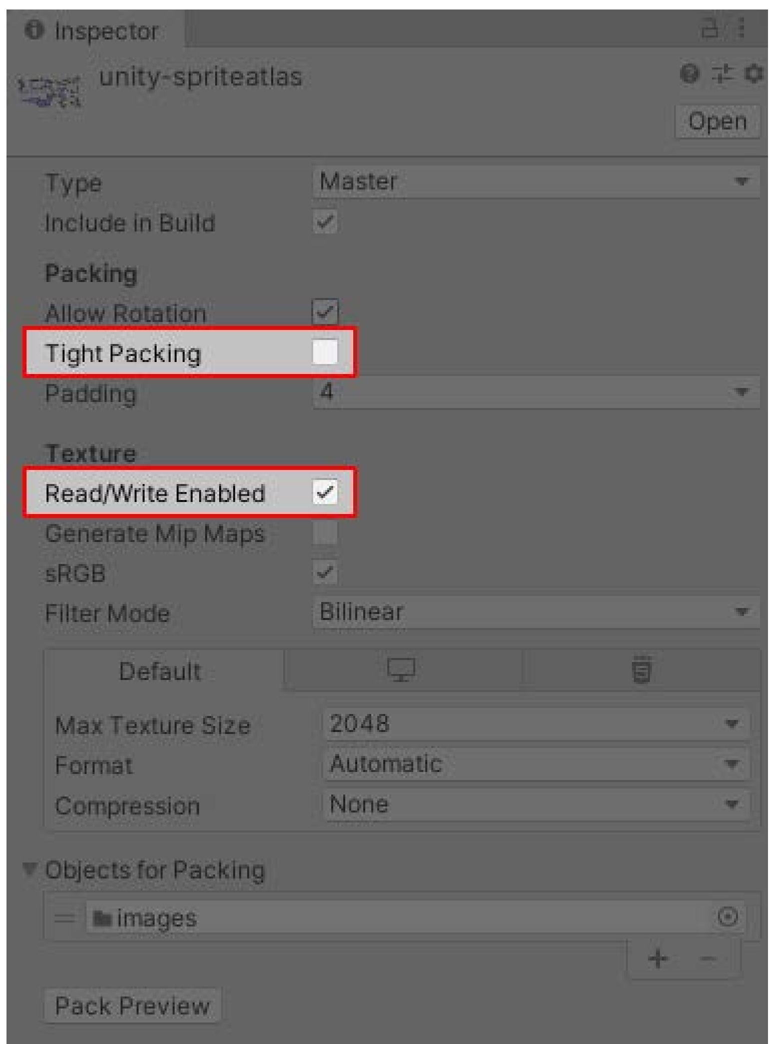
注意: 我们推荐遵循正常的 Spine 工作流,使用 Spine 创建的 sprite atlases 而非 Unity 的 `Sprite Atlas` 资产。Spine atlases 提供了卓越的包装,支持多 atlas 页且健壮性更佳。只有当你不能使用正常的 Spine 工作流时,再使用 Unity 的 Sprite Atlas 资产作为 atlas provider。你还可以在运行时从单个附件重新打包 Spine 的 sprite atlases。

你可以使用 Unity 的 `SpriteAtlas` 作为 atlas provider 以替代携带 skeleton 数据文件的 `.atlas.txt` 和 `.png` 文件. 导入是通过一个特殊的 `Spine SpriteAtlas Import` 工具窗口处理的,可以通过 `Window - Spine - SpriteAtlas Import` 来访问.

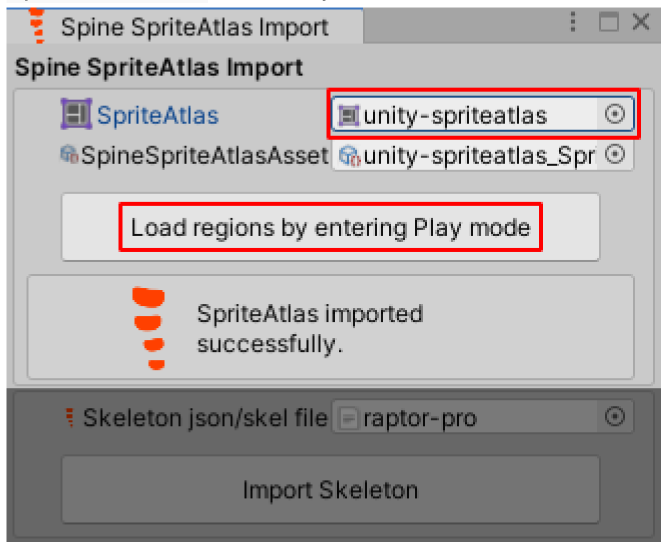


准备 `Sprite Atlas` 的步骤如下:

1. 通过 `Assets - Create - Sprite Atlas` 创建 `Sprite Atlas` .
2. 在 `Sprite Atlas` Inspector 中,将包含所需 Sprites 的文件夹作为附件添加到 `Objects for Packing` 中.
3. (a) 在 Unity 2018.2 以前的版本中，请手动禁用 `Tight Packing` 并启用 `Read/Write Enabled`.
   (b) 在 Unity 2018.2 及后续版本中，这些参数会被自动设置.



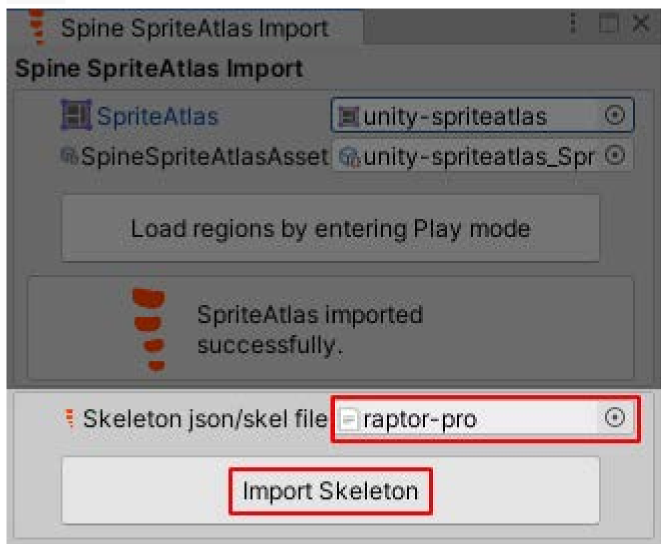
4. 在新的 `Spine SpriteAtlas Import` 窗口中把 `Sprite Atlas` 赋值给 `Sprite Atlas` 属性.Unity 会自动生成附加资产.



5. 点击 `Load regions by entering Play mode` 进入游玩模式然后退出该模式, 这样可以加载区域信息. 现在你的 `Sprite Atlas` 就可作为 Spine atlas 使用了.

要在 `.json` 或 `.skel.bytes` skeleton 资产上使用 atlas:

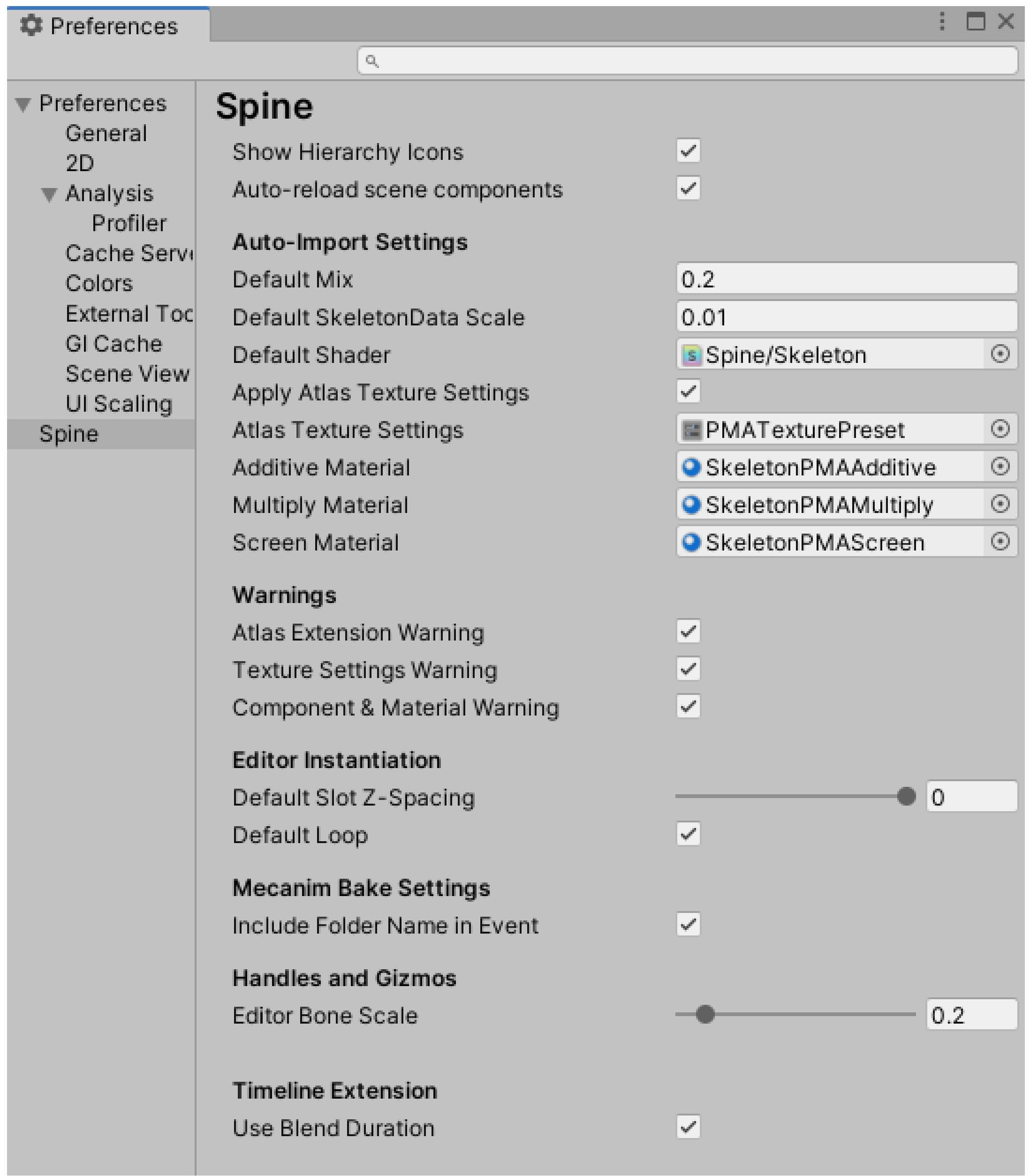
1. 将 `.json` 或 `.skel.bytes` 与新建的 atlas 资产放在同一目录下.
2. 将其赋值给 `Spine SpriteAtlas Import` 窗口中的 `Skeleton json/skel file` 属性



3. 点击 `Import Skeleton`，生成使用 `Sprite Atlas` 资产的_`SkeletonData` 文件.

Spine 首选项

spine-unity 运行时在 Unity 的首选项窗口中添加了一个 `Spine` 页面，可通过 `Edit - Preferences` 访问. 在这里,你可以设置 skeleton 导入和实例化时使用的默认值,以及自定义 spine-unity 运行时的外观和更新行为.



- *Show Hierarchy Icons.* 层次结构面板中带有 Spine 组件的 `GameObjects` 旁显示相关图标.
- *Auto-reload scene components.* 每当场景中的 skeleton 组件的 SkeletonDataAsset 被修改时,都会自动重新加载。当你的场景有大量的 SkeletonRenderer 或 SkeletonGraphic 组件时,自动加载操作可能很慢.
- *Auto-Import Settings*
  - *Default Mix.* 设置新导入的 SkeletonDataAssets 的 Default Mix Duration.
  - *Default SkeletonData Scale.* 设置新导入的 SkeletonDataAssets 的默认 `Scale` 值.
  - *Default Shader.* 设置新导入的 skeleton atlas textures 创建 materials 时使用的默认着色器.

        - *Apply Atlas Texture Settings*. 对下方指定的 texture 导入器应用参考 `Atlas Texture Settings`.
        - *Atlas Texture Settings*. 在新导入的 atlas textures 和 materials 上应用选定的 texture 导入设置。当从 Spine 导出 atlas textures 并启用 `Premultiply alpha` (默认为此)时,你可以将设置保存在 `PMATexturePreset` 中。如果你已经禁用了 `remultiply alpha`,请将其设置为 `StraightAlphaTexturePreset`。你也可以自己创建 `TextureImporter` `Preset` 资产并将之赋值至此.
        - *Additive Material*. 设置 `Additive` 槽混合模式的 Material 模板. 具体参见 `SkeletonData` 混合模式 Materials.
        - *Multiply Material*. 设置 `Multiply` 槽混合模式的 Material 模板. 具体参见 `SkeletonData` 混合模式 Materials.
        - *Screen Material*. 设置 `Screen` 槽混合模式的 Material 模板. 具体参见 `SkeletonData` 混合模式 Materials.
- *Warnings*
    - *Atlas Extension Warning*. 每当发现一个`.atlas` 文件时,均在日志中给出警告和建议.
    - *Texture Settings Warning*. 每当检测到 texture 导入设置可能导致不佳效果(例如白边伪影)时,均在日志中给出警告和建议.
- *Editor Instantiation*
    - *Default Slot Z-Spacing*. 设置新实例化的 SkeletonRenderer 或 SkeletonGraphic 组件的默认 Z Spacing 参数.
    - *Default Loop*. 设置新实例化的 SkeletonRenderer 或 SkeletonGraphic 组件的默认 `Loop` 参数.
- *Mecanim Bake Settings*
    - *Include Folder Name in Event*. 当启用该项时,Mecanim 事件将调用名为"`FolderNameEventName` "的方法; 当禁用时则调用"`EventName` ".
- *Handles and Gizmos*
    - *Editor Bone Scale*. 设置场景视图中显示的骨骼和 gizmo 元素的尺寸.
- *Timeline Extension* - relevant for the Timeline Extension UPM Package
    - *Use Blend Duration*. 设置新创建的 `Spine Animation State Clips` 的默认 `Use Blend Duration` 参数.

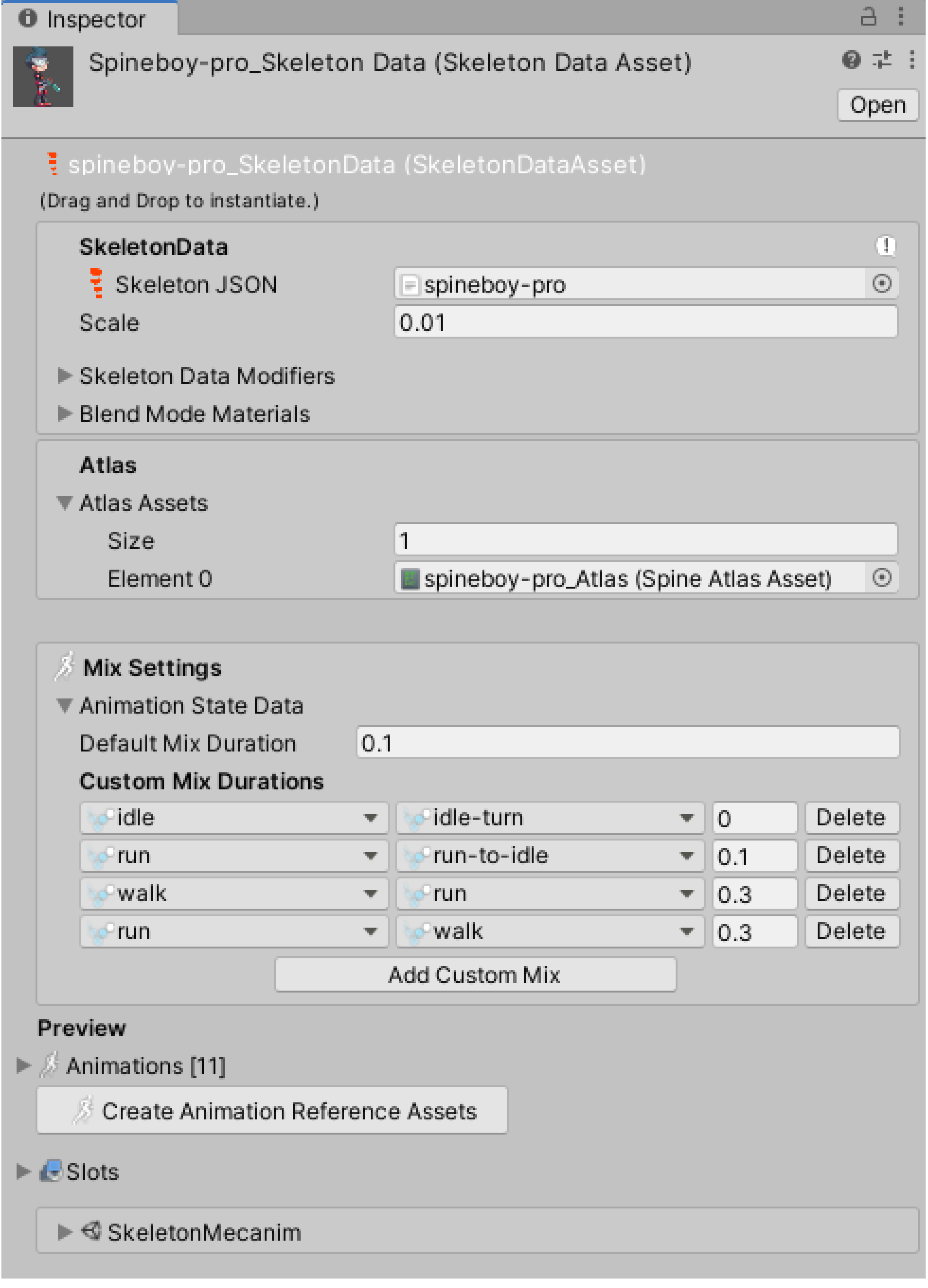
更新 Spine 资产

在开发过程中,可能经常需要更新你的 Spine skeleton 数据和 texture atlas 文件. 直接覆盖这些文件可以完成更新 (`.json`, `.skel.bytes`, `.atlas.txt`, `.png`). 从 Spine 编辑器中重新导出资产,并将再次导出的文件复制到你的项目的 `Assets` 文件夹的子文件夹中, 覆盖现有文件即可.
Unity 将检测这些文件的变化,并自动重新导入修改后的资产。重新导入后,所有以前导入的 Spine 资产的引用都是完整并使用最新数据的.

注意: Unity 有时不能识别文件的变化。在这种情况下,在 Unity 的项目面板中找到你想重新导入的包含 Spine 资产的文件夹,右击该文件夹,然后从上下文菜单中选择 `Reimport`.

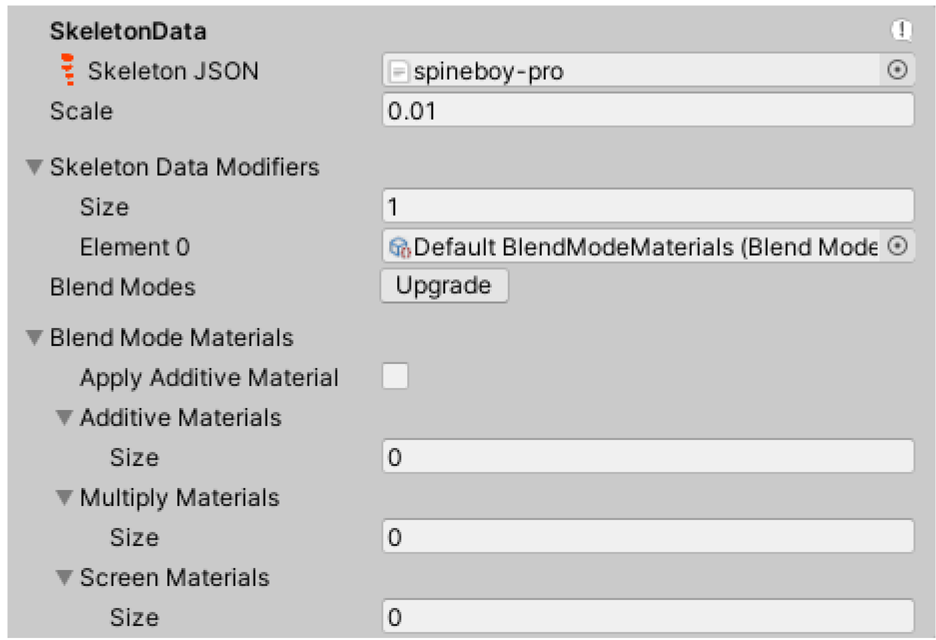
Skeleton Data 资产

skeleton data 资产 (后缀名为`_SkeletonData`) 储了关于 skeleton 层次结构、槽位、绘制顺序、动画和其他构成 skeleton 的数据信息。由 spine-unity 运行时提供的其他组件引用并共享 skeleton data 资产,以实现 skeleton 的动画及显示。
要检查或修改一个 skeleton data 资产,在 Unity 的项目面板中选择它。检查器面板将显示该 skeleton 数据的所有属性以供查看和修改.



Skeleton Data

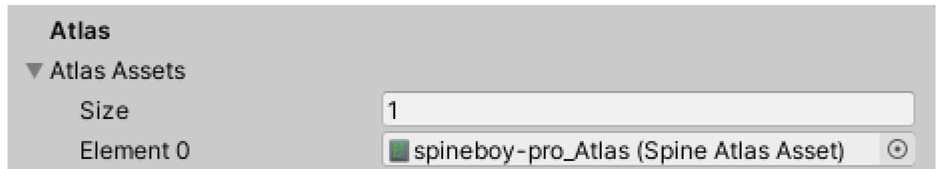
SkeletonData 包含了 skeleton 的通用导入设置.



- *Scale*. 设置你自定义的导入缩放值,它会影响所有引用该数据资产的 skeleton 实例。改变该值将立即影响该 skeleton 的所有实例.
  注意：当你想精确匹配如 32px 的图像到 1 个游戏单位时(在 Spine 中没有缩放附件图像),可以把这个 `Scale` 参数设置为 `1/px_per_unit`。所以对于 `32px/unit` 的情况,可以将 `Scale` 设置为 `1/32 = 0.03125`。
- *SkeletonData Modifiers*. 在完成从`.json` 或 `.skel.bytes` 文件的加载后,为 skeleton data 资产进行额外处理提供了一种方法。请参考下文的 "SkeletonData 修改器资产"部分,以了解更多信息。
- *Blend Modes - Upgrade*. 将过时的 `BlendModeMaterialAsset` 升级为下方可显示的原生的 Blend Mode Materials 属性.
- *Blend Mode Materials*. 具有特殊 blend modes 的 Skeleton 槽位需要额外的 materials。这些 materials 在导入时会自动设置,除非使用了旧版 Unity 和 spine-unity 的 `BlendModeMaterialAsset` 。建议通过上面的 `Blend Modes - Upgrade` 按钮升级该 `BlendModeMaterialAsset` ，这样可以防止在较新的 Unity 版本中出现问题。你可以在 Spine Preferences 定制每个 blend modes 的 material 模板.
  - *Apply Additive Material*. 启用后,也将为 `Additive` 混合模式的槽位生成材质。当使用 Straight alpha 工作流时,启用这个参数。当使用 PMA 工作流程时则无需启用,因为 `Normal` 和 `Additive` 槽位可以使用同样的 PMA material 绘制
  - *Additive Materials*, *Multiply Materials*, *Screen Materials*. 这些列表显示每个 blend mod 当前使用的 blend mode materials.

Atlas

skeleton 使用 Atlas 引用来解析导出的图像名称，该名称是对应用以渲染的图像区域的引用.



在导入时,每个 Atlas 资产(以_**Atlas** 结尾)将会被自动地一一填入 `Atlas Assets` 资产数组中

如果 spine-unity 未能自动填入所有需要的 atlas 资产,你可将 `Atlas Assets` `Size` 改为所需的 atlas 资产数量,并手动按 `Element0` - `ElementN` 将所需的 atlas 资产填入.

Mix Settings

The skeleton data asset 资产中可以设置 animation mix times.

Mix Settings
Animation State Data
Default Mix Duration 0.1
Custom Mix Durations

idle	idle-turn	0	Delete
run	run-to-idle	0.1	Delete
walk	run	0.3	Delete
run	walk	0.3	Delete

Add Custom Mix

Preview

`Default Mix Duration` 设置动画的默认混合时长,单位是秒.

可以通过点击资产的 `Custom Mix Durations` 部分的 `Add Custom Mix` 按钮来定义某两个动画的混合,并设置特定的混合时长,这将会覆盖上方的默认混合时长值.

使用 skeleton data 资产的组件(如 skeleton 动画组件)在播放动画时会使用这些混合时间设置.

Preview

Skeleton Data 资产的预览页面中可以检查资产中包含的所有骨骼(bone)、槽位、动画、皮肤和事件.

Preview

Animations [11]

Setup Pose

Name	Duration
aim	0.000s
death	4.933s
hoverboard	1.000s
idle	1.667s
idle-turn	0.267s
jump	1.333s
portal	3.600s
run	0.800s
run-to-idle	0.267s
shoot	0.633s
walk	0.867s

Create Animation Reference Assets

Slots

Show Attachments　default (83 attachments)

muzzle-ring4
- muzzle-ring

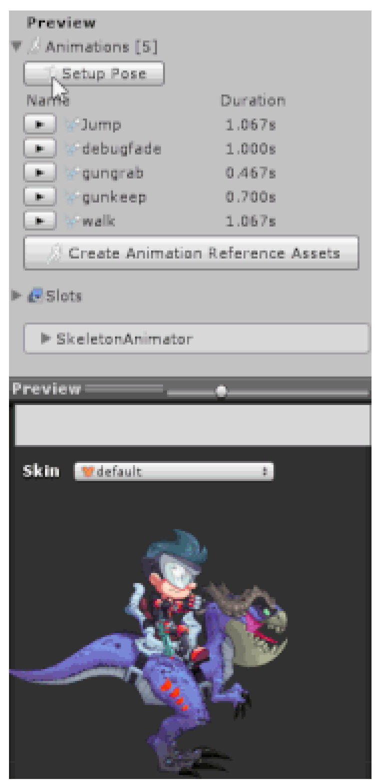
muzzle-ring3
- muzzle-ring

muzzle-ring2
- muzzle-ring

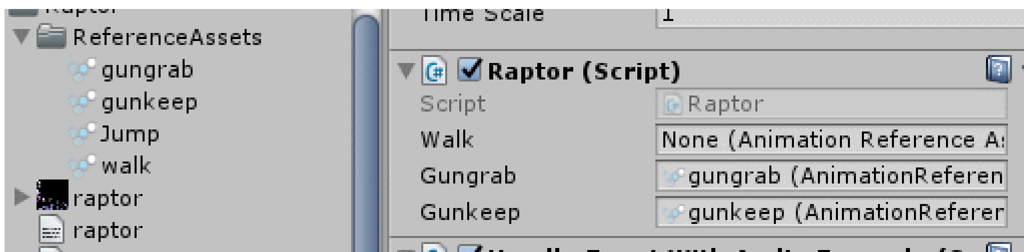
muzzle-ring
- muzzle-ring

muzzle-glow
- muzzle-glow

你可以通过每个动画左边的播放按钮来回放动画,并在 `Slots` 部分的 `Show Attachments` 来查看对应槽位的实时更新。所有时间轴事件以紫色标记显示。在播放时将鼠标悬停在标记上,就能显示事件名称.

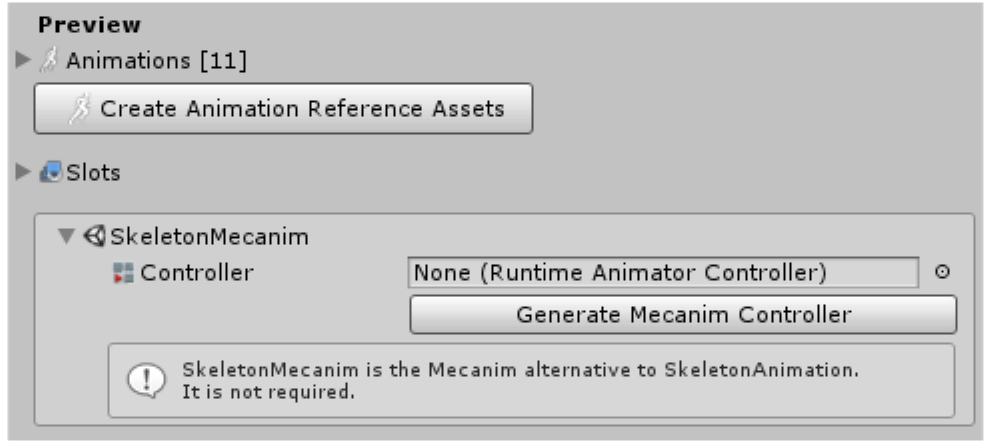


点击 `Create Animation Reference Assets` 按钮即可为 skeleton 的所有动画生成引用资产. 一个 `AnimationReferenceAsset` 就像一个引用了 `Spine.Animation` 的 Unity 资产, 它也可以在检查器中赋值给组件属性.



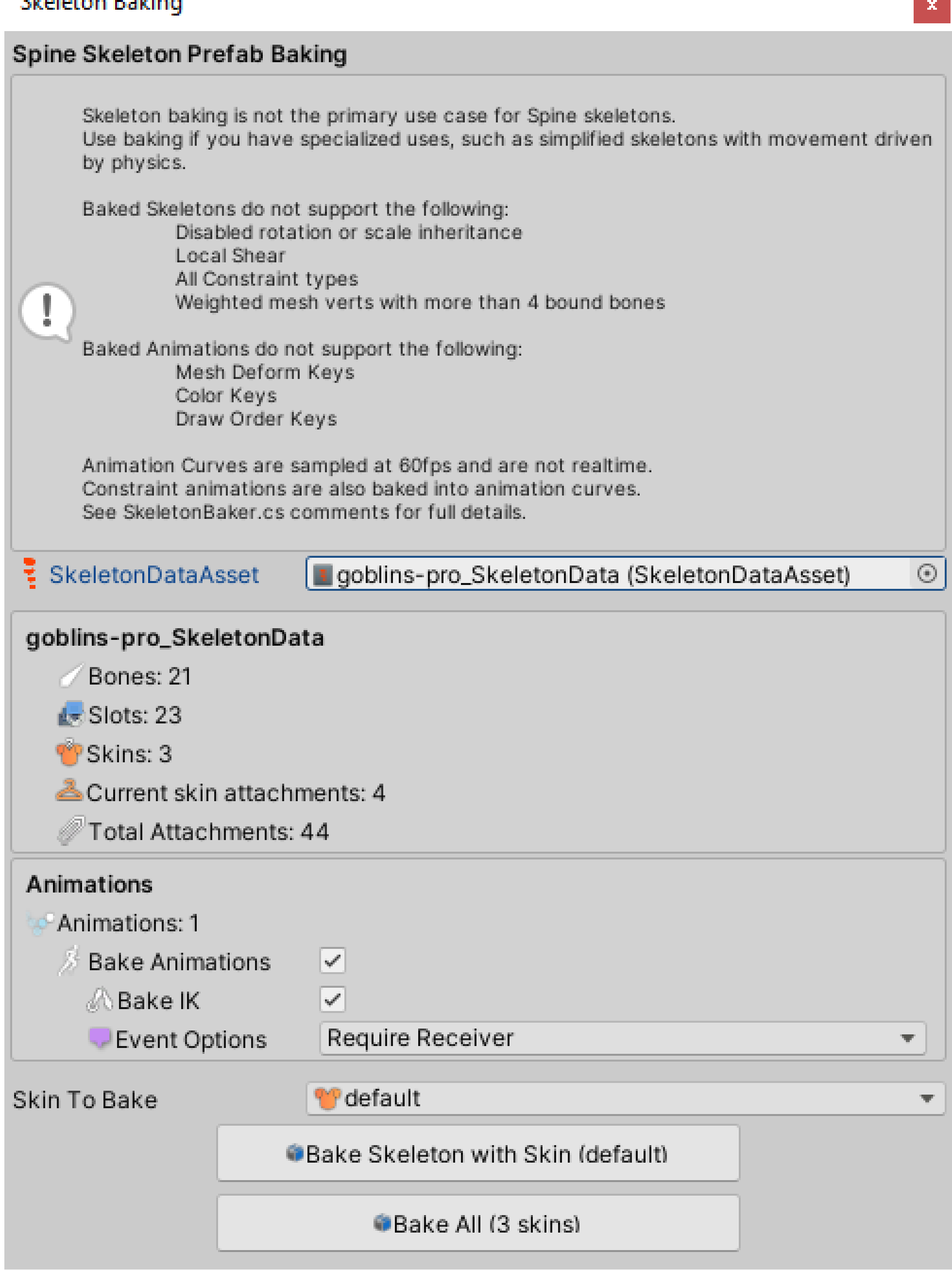
Skeleton Mecanim

如果你想使用 Unity 的 Mecanim 动画系统不是 Spine 的默认动画系统, 你可以通过 `Generate Mecanim Controller` 按钮生成并赋值一个 Mecanim 控制器.



Skeleton Baking

注意: 烘焙是一个专用工具,在 spine-unity 中**不推荐通过这种方式**来使用 Skeletons! 它不适用于 SkeletonMecanim, SkeletonAnimation 或 SkeletonGraphic (UI)组件! 它将把一个 Skeleton 烘焙成一个不太灵活的，包含带有 `MeshRenderers` 的, `Transforms` 层次结构固定的 Prefab，兼容 animation clips。Spine 的大量功能在 Unity 的动画系统中无法识别无法转换，导致这些功能在烘焙后就被丢弃.



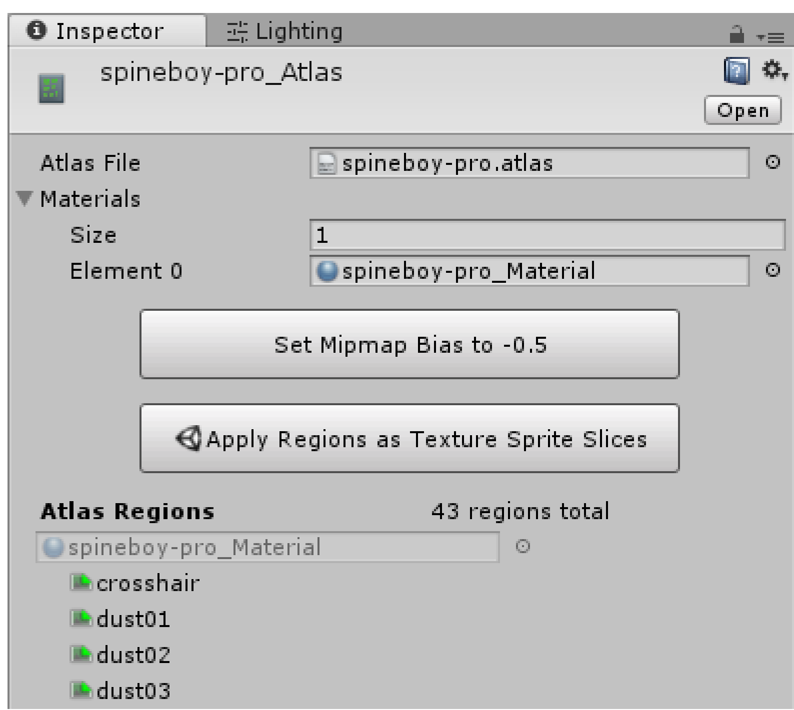
点击 `SkeletonDataAsset` 检查器右上方的齿轮图标选择 `Skeleton Baking` 就能打开 `Skeleton Baking Window`.

在 SkeletonBaker.cs中可以找到支持和不支持功能的详细列表.

注意: 烘焙没有使用最近新增的 Unity 2D 动画系统,而是采用了基于 3D `MeshRenderer` 的动画系统.

Texture Atlas 资产

texture atlas 资产包含了 skeleton 所使用的图像的信息,即一个图像存储在哪个 texture atlas 页上,以及它在 texture atlas 页上的 UV texture 坐标.



你可以通过双击 `Materials` 数组中的 material 资产来查看 texture atlas 页的 material.
注意: 你可以修改 texture atlas 资产所引用的 material 和 texture。在修改 texture 的时候,要确保 UV texture 坐标是有效的.
`Set Mipmap Bias to -0.5` 按钮是为高级用户准备的,它可以修正在 atlas texture 上启用 `Generate Mip Maps` 时的模糊问题.
你可以按下 `Apply Regions as Texture Sprite Slices` 按钮为 atlas 中每个图像元素生成 sprites。生成的 sprites 引用了 texture atlas 图像(`png` 文件)的区域,且可以作为任何 Unity sprite 资产使用.

SkeletonData 修改器资产

SkeletonData 修改器在完成从`.json` 或 `.skel.bytes` 文件的加载后,为 skeleton data 资产进行额外处理提供了一种方法.
SkeletonDataAsset's 检查器提供了一个 `Skeleton Data Modifiers` 列表,可以在此添加修改器资产.

tank-pro SkeletonData (SkeletonDataAsset)
(Drag and Drop to instantiate.)
SkeletonData
Skeleton JSON tank-pro
Scale 0.01
Skeleton Data Modifiers
Size 1
Element 0 Default BlendModeMaterials (BlendMode
Atlas

内置的 SkeletonData 修改器 - BlendModeMaterialsAsset *(已废弃)*

Note: spine-unity 现在为每个 `SkeletonDataAsset` 提供了**原生槽位混合模式** `Additive`, `Multiply` 和 `Screen`,并在新导入的 skeleto 资产时可自动设置。`BlendModeMaterialAssets` 现在已经废弃,被原生的 `SkeletonDataAsset` 属性所取代。`SkeletonDataAsset` 检查器提供了一个新的 `lend Modes - Upgrade` 按钮,用以将废弃的 `BlendModeMaterialAsset` 升级为原生的混合模式属性。这个升级将在 Unity 2020.1 及其后续版本的导入和重导入的资产中自动执行,以避免这些 Unity 版本中出现 BlendModeMaterialAsset 导致的问题. spine-unity 4.0 及更高版本将自动执行这一升级,与 Unity 版本无关.

`BlendModeMaterialsAsset` is a SkeletonData modifier asset class included in spine-unity. It holds references to materials that can be used to render attachments within slots that have the `Additive`, `Multiply` and `Screen` blend modes assigned to them in the Spine editor.

The Material references stored in BlendModeMaterials assets are used as templates to generate new Materials that use the appropriate texture needed by the loaded attachments.

The spine-unity runtime comes packaged with a ready-to-use `BlendModeMaterialsAsset` named `Default BlendModeMaterials`. Using this included asset allows the attachments in slots with special blend modes to use the included default `Multiply` and `Screen` shaders: `Spine/Blend Modes/Skeleton PMA Multiply` and `Spine/Blend Modes/Skeleton PMA Screen`.

If you need to use different Materials or shaders or Materials with different settings, you can create new `BlendModeMaterialsAssets` using `Create -> Spine -> SkeletonData Modifiers -> Blend Mode Materials`. Then assign your Material templates to the created asset.

Writing a custom SkeletonDataModifierAsset class

You can write your own custom `SkeletonDataModifierAsset` class to add additional processing to skeleton data assets after loading from a `.json` or `.skel.bytes` file. `SkeletonDataModifierAsset` is an abstract `ScriptableObject` class where you can derive your own classes from.

1. Create a new class derived from `SkeletonDataModifierAsset` and implement the `void Apply (SkeletonData skeletonData)` method. Add the `CreateAssetMenu` class attribute to list an entry for your class in the `Asset -> Create` menu.
   C#
   ```
   [[CreateAssetMenu(menuName = "TopMenu/Submenu/SubSubmenu", order = 200)]
   public class BlendModeMaterialsAsset : SkeletonDataModifierAsset {

    public override void Apply (SkeletonData skeletonData) {
       ...
    }
   }
   ```
2. Create an instance of your new class by selecting your desired folder in the Project panel and selecting your newly created `Asset -> Create` menu entry. Assign the created asset at an element of the `Skeleton Data Modifiers` list of your SkeletonData asset.

`Apply(skeletonData)` will be called after loading data from the `.json` or `.skel.bytes` file has been completed.

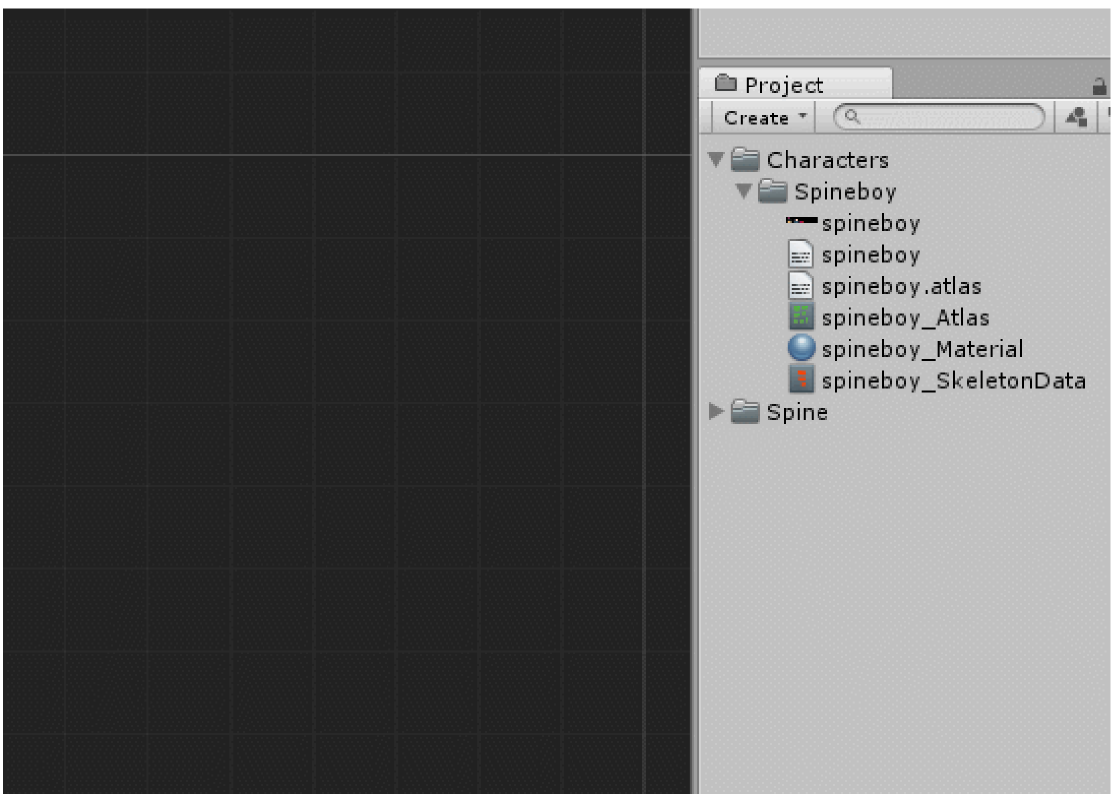
主要组件

spine-unity 运行时提供了一组显示、动画、跟踪和修改从 Spine 导出的 skeletons 的组件。这些组件将引用按上文所述导入的 skeleton data 和 texture atlas 资产.

将 Skeleton 添加到场景中

在 Unity 项目中快速添加 Spine 的骨架 skeleton，需要:

1. 按前文所述来导入 skeleton data 和 texture atlas.
2. 将 **_SkeletonData** 资产拖入 Scene 视图或 Hierarchy 面板然后选择 `SkeletonAnimation`. 这将实例化一个带有所需 Spine 组件的新 GameObject.



注意: 作为步骤 2 的另一个实现方法是从头开始创建这个 GameObject:

1. 用 `GameObject -> Create Empty` 菜单创建一个空的新 GameObject.
2. 在检查器中选择 GameObject 然后点击 `Add Component`, 然后选择 `SkeletonAnimation`. 这会自动为 GameObject 添加 `MeshRenderer` 和 `MeshFilter` 组件.
3. 在 SkeletonAnimation 组件中, 把所需的 **_SkeletonData** 资产赋值给 `Skeleton Data Asset` 属性.

注意: 为防在场景视图中只能看见无图附加的 skeleton 骨骼(bones), 你可能需要在皮肤的 `Initial Skin` 属性中切换为 `default` 以外的皮肤.

你现在可以使用组件的 C# API 来控制 skeleton 动画, 对动画触发的事件做出反应等等. 更多细节请参考下文的组件文档.

SkeletonAnimation 的一种替代方案 - SkeletonGraphic (UI)和 SkeletonMecanim

将 skeleton 实例化 SkeletonAnimation 是在 Unity 中使用 Spine skeleton 的推荐方法, 因为它提供了三种方法中最为完整的功能.

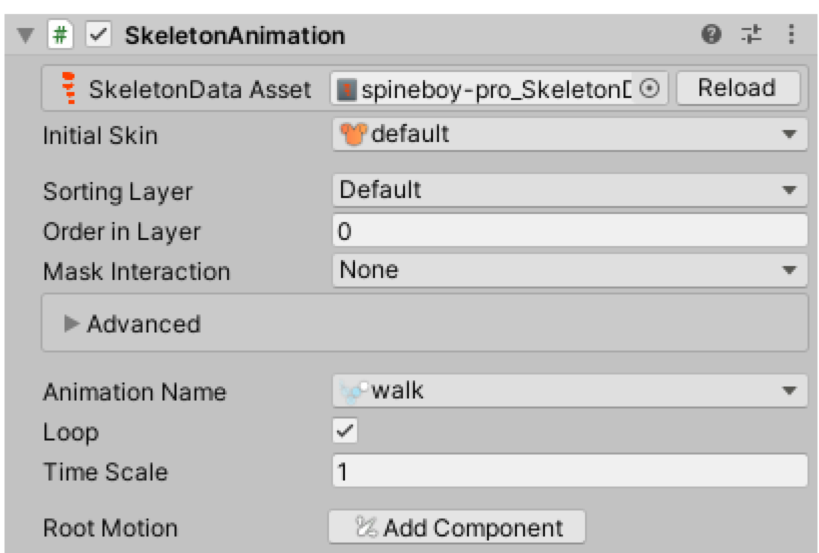
三种实例化 skeleton 的方法为:

1. **SkeletonAnimation** – 使用 Spine 定制的动画和事件系统, 提供最高的可定制性. 渲染器使用 `MeshRenderer`, 像 Unity sprite 一样与 `SpriteMask` 等遮罩进行交互. *推荐在 Unity 中以这种方式使用 Spine skeleton.*
2. **SkeletonGraphic (UI)** - 与 Unity `Canvas` 一起作为 UI 元素使用。像内置的 Unity UI 元素一样渲染并与 UI 遮罩(如 `RectMask2D`)交互。动画和事件行为与 SkeletonAnimation 相同

3. **SkeletonMecanim** – 使用 Unity 的 Mecanim 动画和事件系统来开始、混合和过渡动画。与 SkeletonAnimation 相比,提供的动画混合和过渡选项较少。当使用 `SkeletonMecanim` 时,将不能保证动画过渡看起来和 Spine Editor 中的预览一致.

SkeletonAnimation 组件

`SkeletonAnimation` 组件是三种在 Unity 中使用 Spine skeleton 的方式之一。这些三种方法是: SkeletonAnimation, SkeletonMecanim 和 SkeletonGraphic (UI)
SkeletonAnimation 组件是 spine-unity 运行时的核心。通过它你可以将一个 Spine skeleton 添加到一个 GameObject 上,播放动画、对动画事件作出反应等等.



设置 Skeleton Data
SkeletonAnimation 组件需要一个对 skeleton data 资产的引用,它可以从中获得 skeleton 的骨骼层次、槽位等信息。
如果您通过拖放将一个 skeleton 添加到一个场景中,skeleton data 资产会被自动赋值。如果你有一个已经设置好的 GameObject,而突然想把 skeleto 改成一个不同的资产,你可以通过检查器属性手动执行更改。

要设置或改变 skeleton data 则需要:

1. 选中 SkeletonAnimation 的 GameObject
2. 检查器中的 `SkeletonData Asset` 属性处赋值一个_**SkeletonData** 资产.

设置初始皮肤和动画
SkeletonAnimation 检查器公开暴露了以下参数:

1. *Initial Skin.* 该皮肤将在动画开始时显示。注意：如果你看到的 skeleton 没有附加任何图像,可以切换到默认以外的皮肤来显示皮肤.
2. *Animation Name.* 该动画将在开始时播放.
3. *Loop.* 定义初始动画是循环播放还是只播放一次.

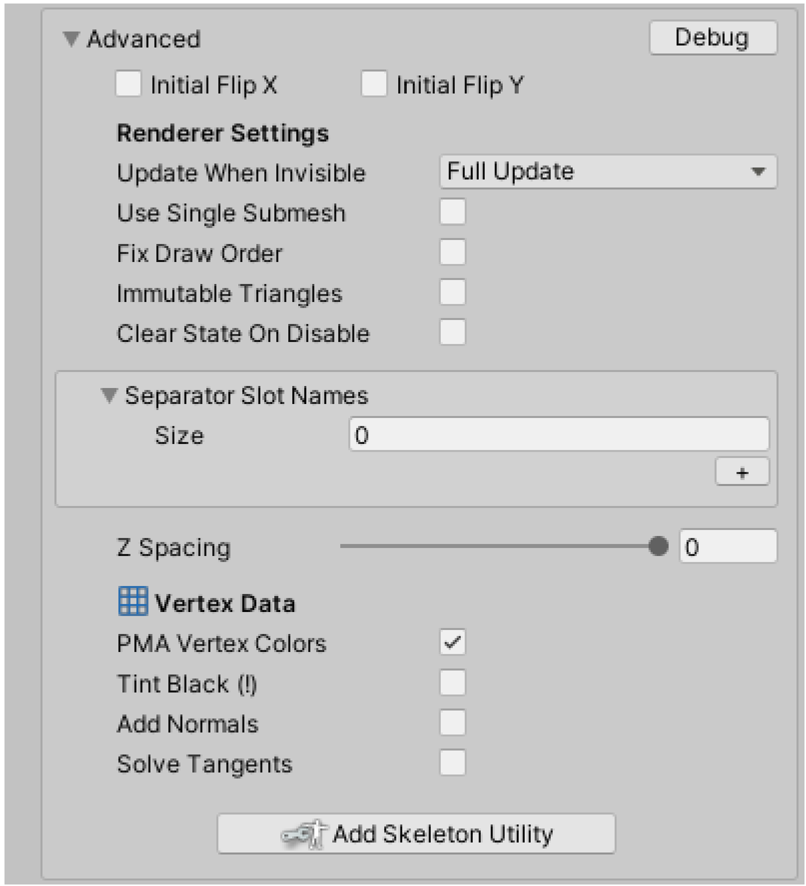
4. *Time Scale*. 设置时间比例来减慢或加快动画的播放速度.

启用 Root Motion

`SkeletonAnimation` 和 SkeletonGraphic (UI) 组件的 root motion 是通过一个单独的 SkeletonRootMotion 组件实现的的。 `SkeletonAnimation` 检查器中的 `Root Motion` `Add Component` 按钮可以快速添加合适的组件到 skeleton GameObject 上.

设置高级参数

展开 SkeletonAnimation 检查器的 `Advanced` 部分就能显示高级配置参数.

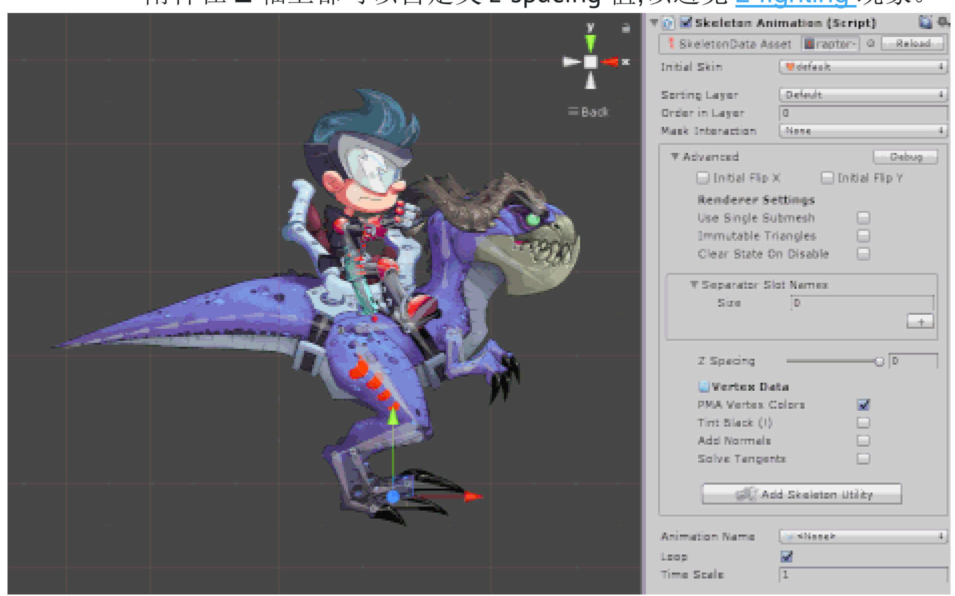


SkeletonAnimation 公开暴露了以下高级参数:

- *Initial Flip X, Initial Flip Y*. 这两个选项在开始时水平或垂直地翻转 skeleton。他们会把翻转后的 skeleton 的 *ScaleX* 和 *ScaleY* 置为*-1* .
- *Update When Invisible*. 当 MeshRenderer 不可见时的更新模式。当网格再度可见时,更新模式会自动重置为 `UpdateMode.FullUpdate`.
- *Use single submesh*. 该选项选项用于简化子网格的生成,它假设 skeleton 只使用了一个 Material 且只需一个子网格。这将禁用多种 materials、渲染分离(render separation)和自定义槽位 materials.
- *Fix Draw Order*.
  仅适用于使用 3 个以上子网格的情况(顺序交错的多个 materials,例如 "A B A")。如果为 "true",MaterialPropertyBlocks 会被赋值到每个 material 中,以防止 LWRP 等渲染器对子网格进行主动(aggressive) batching, 从而导致绘制顺序错

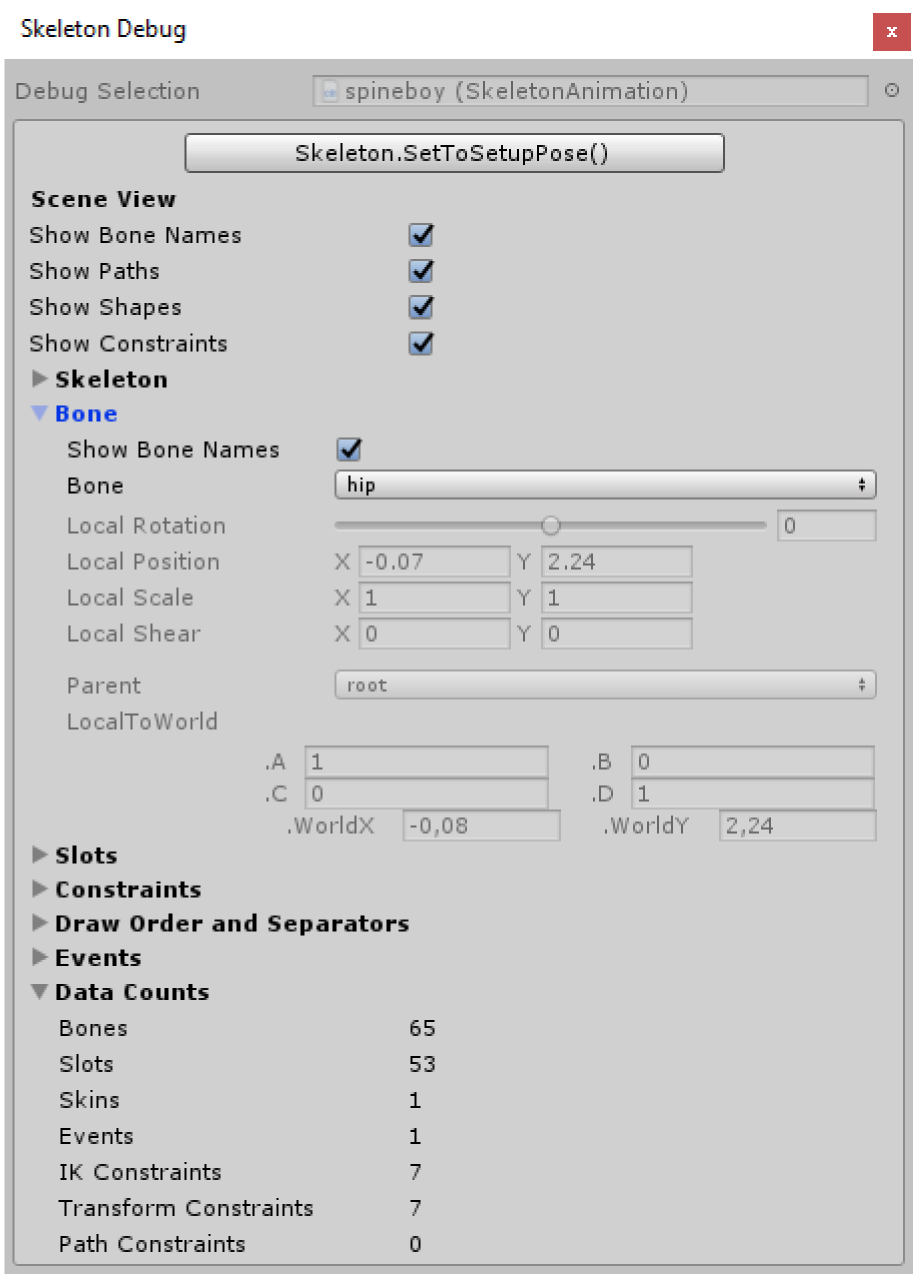
误(例如,"A1 B A2 "变为"A1 A2 B")。当绘制正确时,可以禁用该参数来避免额外的性能开销.

- *Immutable triangles.* 启用这个参数可以优化附件图片可见性固定不变时 skeletons 的渲染。若启用则三角形将不会被更新。如果 skeleton 的附件未使用 swapping 或 hiding,或者未使用绘制顺序键,则应启用此选项来进行优化。否则设置为 false 可能会导致渲染时出现错误.
- *Clear State on Disable.* 当该组件或其 GameObject 被禁用时,清除渲染和 skeleton 的状态。这可以避免它们在再次启用时之前的状态被保留。当 pooling skeleton 时,将其设置为"true "可能会有所裨益。
- *Separator Slot Names.* 设置渲染分离时的槽位名称。`SkeletonRenderSeparator` 等组件将使用该选项,以便在不同的 GameObjects 上用两个独立的渲染器来渲染 skeleton。
- *Z-Spacing.* skeleton 渲染器组件在 X/Y 平面上对附件图片从后往前渲染。每个附件在 Z 轴上都可以自定义 z-spacing 值,以避免 z-fighting 现象。

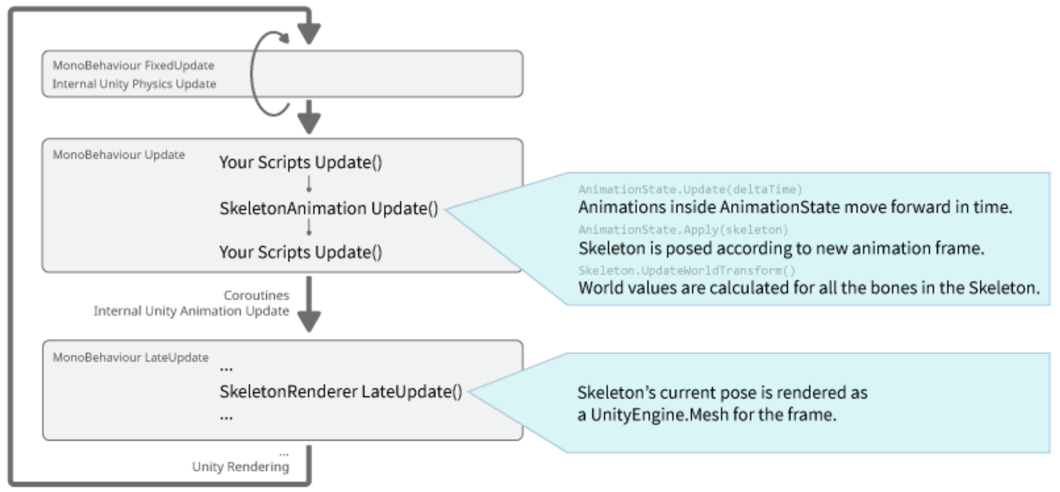


- *PMA Vertex Colors.* 将顶点色彩 RGB 值与顶点色彩 alpha 值相乘。如果用于渲染的着色器是 Spine 着色器(即使使用了 `traight Alpha Texture`)或者使用了 PMA additive 混合模式 `Blend One OneMinusSrcAlpha` 的第三方着色器,则应启用该参数。对于使用常规混合模式 `Blend SrcAlpha OneMinusSrcAlpha` 的普通着色器,则禁用此参数。当禁用时,它可以防止 additive 槽位的 single-batch 渲染,并可能增加绘制调用的次数.
- *Tint Black (!).* 为网格添加黑色 tint 顶点数据。黑色 tinting 要求着色器将 UV2 和 UV3 解释为黑色 tint。你可以使用附带的 `Spine/Skeleton Tint Black` 着色器。如果你只需要 tint 整个骨架而非个别部分,建议使用 `Spine/Skeleton Tint` 着色器,这样效率更高,并且可以通过 MaterialPropertyBlock 改变或在动画中使用`_Black` material 属性。更多信息请参见着色器部分。为了保证对多个 tint skeletons 的不同 batching 能力,建议使用 Skeleton.R .G .B .A 进行 tint.

- *Add Normals*. 启用后网格生成器会在输出网格中添加法线。如果你的着色器需要顶点法线请启用之。为了更佳性能和更低的内存占用,推荐使用可以接收(assume)所需法线的着色器,比如 `Spine/Skeleton Lit` 着色器。注意,`Spine/Sprite` 着色器也可以被配置为接收 `Fixed Normal`
- *Solve Tangents*. 一些通常用于应用法线贴图的 lit 着色器需要顶点切线(tangents)。启用后,每一帧都会计算切线并添加进输出网格.
- *Add Skeleton Utility*. 该按钮可以用来添加一个用于跟踪或覆盖骨骼(bone)位置的 `SkeletonUtility` 组件到 GameObject 中。更多信息请参见 SkeletonUtility .
- *Debug*. 有时你可能想在游戏运行时了解槽位当前的颜色或骨骼比例。按下 *Debug* 按钮可以打开为此目的而创建的 *Skeleton Debug* 窗口。你可以在这里检查 skeletion 当前的骨骼状态、槽位、约束、绘制顺序、事件和统计信息.



生命周期



在 SkeletonAnimation 组件中, AnimationState 保存了所有当前播放的和队列中的动画状态。每一次 `Update`, AnimationState 都会被更新,从而使动画在时间上向前推进。然后,新一帧作为一个新的 pose 被应用于 Skeleton 上.

你的脚本可以在 SkeletonAnimation 的 `Update` 之前或之后运行。如果你的代码在 SkeletonAnimation 的更新之前获取 Skeleton 或骨骼的值,你的代码会读取上一帧而不是当前帧的值.

该组件将事件回调委托暴露为属性,这样就能在计算所有骨骼的世界 transforms 之前和之后介入该生命周期。你可以绑定这些委托来修改骨骼的位置和其他方面,而无需关注角色和组件的更新顺序.

SkeletonAnimation Update 回调:

- `SkeletonAnimation.BeforeApply` 事件在应用该帧动画之前被引发。当你想在动画应用到 skeleton 上之前改变 skeleton 的状态时，请使用这个回调.
- `SkeletonAnimation.UpdateLocal` 事件在该帧动画更新完成并应用于 skeleton 的局部值之后被引发。如果你需要读取或修改骨骼的局部值,请使用它.
- `SkeletonAnimation.UpdateComplete` 在计算完 Skeleton 中所有骨骼的世界值后被引发。SkeletonAnimation 在这之后不会在 Update 中做进一步的操作。如果你只需要读取骨骼的世界值,就使用它。如果有代码在 SkeletonAnimation 的 Update 之后修改了这些值,这些值可能仍会发生变化.
- `SkeletonAnimation.UpdateWorld` 是在计算了 Skeleton 中所有骨骼的世界值后引发的。若订阅了该事件则将二次调用 `skeleton.UpdateWorldTransform`。如果你的 skeleton 复杂或正在执行其他计算,那这将是多余甚至是浪费的行为。如果需要根据骨骼的世界值来修改骨骼的局部值,请使用该事件。它在 Unity 代码中实现自定义约束很有用。
- `OnMeshAndMaterialsUpdated` 在 `LateUpdate()`结束后,在网格和所有 materials 都被更新后引发.

```
1. // your delegate method
2. void AfterUpdateComplete (ISkeletonAnimation anim) {
3.    // this is called after animation updates have been completed.
4. }
5.
6. // register your delegate method
7. void Start() {
8.    skeletonAnimation.UpdateComplete -= AfterUpdateComplete;
9.    skeletonAnimation.UpdateComplete += AfterUpdateComplete;
10. }
```

另一种做法是改变脚本的执行顺序,在 SkeletonAnimation 的 Update 方法之后运行.

关于 Unity 中 MonoBehaviour 的生命周期的更多信息请见: docs.unity3d.com/Manual/ExecutionOrder

C#

通过代码与 skeleton 互动需要访问 SkeletonAnimation 组件。如同其他 Unity 组件一样,建议对引用查询一次后将其存储起来以便复用.

```
1. ...
2. using Spine.Unity;
3.
4. public class YourComponent : MonoBehaviour {
5.
6.    SkeletonAnimation skeletonAnimation;
7.    Spine.AnimationState animationState;
8.    Skeleton skeleton;
9.
10.   void Awake () {
11.      skeletonAnimation = GetComponent<SkeletonAnimation>();
12.      skeleton = skeletonAnimation.Skeleton;
13.      //skeletonAnimation.Initialize(false); // when not accessing skeletonAnimation.Skeleton,
14.                                  // use Initialize(false) to ensure everything is loaded.
15.      animationState = skeletonAnimation.AnimationState;
16.   }
```

Skeleton

SkeletonAnimation 组件通过 `SkeletonAnimation.Skeleton` 属性提供对底层 skeleton 的访问。一个 Skeleton 存储了对一个 skeleton data 资产的引用,而 skeleton data 又引用了一个或多个 atlas 资产。

通过 Skeleton 可以设置皮肤、附件、重置骨骼为 setup pose、比例以及翻转整个 skeleton.

设置附件

设置一个附件需要提供槽位和附件名称:

C#

```
bool success = skeletonAnimation.Skeleton.SetAttachment("slotName", "attachmentName");
```

C#

```
// using properties
[SpineSlot] public string slotProperty = "slotName";
[SpineAttachment] public string attachmentProperty = "attachmentName";
...
bool success = skeletonAnimation.Skeleton.SetAttachment(slotProperty, attachmentProperty);
```

请注意,上述代码中的`[SpineSlot]` 和 `[SpineAttachment]` 是 本节中描述的 String Property Attributes.

重置为 Setup Pose

程序性动画有时需要将骨骼和/或槽位重置为其 setup pose.

C#

```
skeleton.SetToSetupPose();
skeleton.SetBonesToSetupPose();
skeleton.SetSlotsToSetupPose();
```

设置皮肤

一个 Spine skeleton 可以有多个定义了哪个附件在哪个槽位上的皮肤。skeleton 组件提供了一个简单的方法来切换皮肤。

C#

```
bool success = skeletonAnimation.Skeleton.SetSkin("skinName");
```

C#

```
// using properties
[SpineSkin] public string skinProperty = "skinName";
...
```

```
4.  bool success = skeletonAnimation.Skeleton.SetSkin(skinProperty);
```

组合皮肤

可以组合 Spine 皮肤,比如从一个个衣物皮肤组成一个完整的角色皮肤。更多细节请参见新的皮肤 API 文档.

C#

```
1.  var skeleton = skeletonAnimation.Skeleton;
2.  var skeletonData = skeleton.data;
3.  var mixAndMatchSkin = new Skin("custom-girl");
4.  mixAndMatchSkin.AddSkin(skeletonData.FindSkin("skin-base"));
5.  mixAndMatchSkin.AddSkin(skeletonData.FindSkin("nose/short"));
6.  mixAndMatchSkin.AddSkin(skeletonData.FindSkin("eyelids/girly"));
7.  mixAndMatchSkin.AddSkin(skeletonData.FindSkin("eyes/violet"));
8.  mixAndMatchSkin.AddSkin(skeletonData.FindSkin("hair/brown"));
9.  mixAndMatchSkin.AddSkin(skeletonData.FindSkin("clothes/hoodie-orange"));
10. mixAndMatchSkin.AddSkin(skeletonData.FindSkin("legs/pants-jeans"));
11. mixAndMatchSkin.AddSkin(skeletonData.FindSkin("accessories/bag"));
12. mixAndMatchSkin.AddSkin(skeletonData.FindSkin("accessories/hat-red-
    yellow"));
13. skeleton.SetSkin(mixAndMatchSkin);
14. skeleton.SetSlotsToSetupPose();
```

运行时重打包

在组合皮肤时会有多个 material 累积. 会引发额外的绘制调用. 使用 `Skin.GetRepackedSkin()` 方法可以在运行时将收集到的皮肤中使用过的 texture 区域合并为一个 texture.

C#

```
1.  using Spine.Unity.AttachmentTools;
2.
3.  // Create a repacked skin.
4.  Skin repackedSkin = collectedSkin.GetRepackedSkin("Repacked skin", skeletonA
    nimation.SkeletonDataAsset.atlasAssets[0].PrimaryMaterial, out runtimeMateri
    al, out runtimeAtlas);
5.  collectedSkin.Clear();
6.
7.  // Use the repacked skin.
8.  skeletonAnimation.Skeleton.Skin = repackedSkin;
9.  skeletonAnimation.Skeleton.SetSlotsToSetupPose();
10. skeletonAnimation.AnimationState.Apply(skeletonAnimation.Skeleton);
11.
12. // You can optionally clear the cache after multiple repack operations.
13. AtlasUtilities.ClearCache();
```

重要提示: 如果重打包失败或产生意外问题,可能源于以下原因:

1. 禁用了读/写。你可能需要在并入重打包 texture 的源 textures 上设置 `Read/Write Enabled` 参数.
2. 启用了压缩: 需要确保源 texture 的 Texture 导入设置中 `Compression` 为 `None`,而不是 `Normal Quality`.
3. 质量层使用了 half/quarter 分辨率的 textures: 这个是 Unity 的 bug,当使用 half/quarter 分辨率的 texture 时会复制错误区域。确保项目设置中的所有质量层都使用全分辨率 texture.
4. 源 texture 不是 2 次 texture 但 Unity 将其放大为了最接近次数：a)从 Spine 导出时启用 Pack Settings 中的 `Power of two`,或者 b)确保 Unity 的 Atlas Texture 导入设置中将 `Non-Power of Two` 置为 `None`.

可以查看示例场景 `Spine Examples/Other Examples/Mix and Match` 和 `Spine Examples/Other Examples/Mix and Match Equip` 以及 `MixAndMatch.cs` 示例脚本来加深理解。

高级 – 使用法线贴图的运行时重打包

你也可以同时重打包主 texture 的法线贴图和其他附加 texture 层。只需将 `int[] additionalTexturePropertyIDsToCopy = new int[] { Shader.PropertyToID("_BumpMap") };`传入为 `GetRepackedSkin()` 的参数,就能重打包主 texture 和法线贴图层.

C#

```csharp
Material runtimeMaterial;
Texture2D runtimeAtlas;
Texture2D[] additionalOutputTextures = null;
int[] additionalTexturePropertyIDsToCopy = new int[] { Shader.PropertyToID("_BumpMap") };
Skin repackedSkin = prevSkin.GetRepackedSkin("Repacked skin", skeletonAnimation.SkeletonDataAsset.atlasAssets[0].PrimaryMaterial, out runtimeMaterial, out runtimeAtlas,
 additionalTexturePropertyIDsToCopy : additionalTexturePropertyIDsToCopy, additionalOutputTextures : additionalOutputTextures);

// Use the repacked skin.
skeletonAnimation.Skeleton.Skin = repackedSkin;
skeletonAnimation.Skeleton.SetSlotsToSetupPose();
skeletonAnimation.AnimationState.Apply(skeletonAnimation.Skeleton);
```

注意: 通常情况下,法线贴图属性被命名为`"_BumpMap"`。当使用自定义着色器时,一定要使用相应的属性名。这个名字是着色器中的属性名,而非检查器中`"Normal Map"`标签里显示的字符串。

改变 Skeleton 比例和翻转 Skeleton

垂直或水平翻转一个 skeleton 可以复用动画,例如一个面向左边的行走动画可以回放为右朝向.

C#

```
1. bool isFlippedX = skeleton.ScaleX < 0;
2. skeleton.ScaleX = -1;
3. bool isFlippedY = skeleton.ScaleY < 0;
4. skeleton.ScaleY = -1;
5.
6. skeleton.ScaleX = -skeleton.ScaleX; // toggle flip x state
```

手动获取和设置骨骼 Transforms

注意: 该做法仅适用于非常特殊的场合. Spine 的 BoneFollower 和 SkeletonUtilityBone 组件是一种与骨骼交互的更简单方式

通过 Skeleton 可以设置和获取骨骼的 transform 值,以便实现 IK 地形跟随或者让其他角色和组件(如粒子系统)跟随 skeleton 中的骨骼。

注意: 要确保通过订阅 `SkeletonAnimation.UpdateWorld` 委托在更新世界 transform 的生命周期中获取和应用了新的骨骼位置。否则你的修改在读取时可能会晚一帧,或者在设置时被动画覆盖.

C#

```
1. Bone bone = skeletonAnimation.skeleton.FindBone("boneName");
2. Vector3 worldPosition = bone.GetWorldPosition(skeletonAnimation.transform);

3. // note: when using SkeletonGraphic, all values need to be scaled by the par
   ent Canvas.referencePixelsPerUnit.
4.
5. Vector3 position = ...;
6. bone.SetPositionSkeletonSpace(position);
7.
8. Quaternion worldRotationQuaternion = bone.GetQuaternion();
```

动画 - AnimationState

生命周期

SkeletonAnimation 组件实现了 `Update` 方法,它根据 delta 时间更新底层 AnimationState ,将 `AnimationState` 应用于 skeleton 上,并更新 skeleton 上所有骨骼的世界 transforms. skeleton 动画组件通过 `SkeletonAnimation.AnimationState` 属性暴露了 AnimationState API。本节假定你熟悉轨道、轨道项、混合时间或动画队列等概念,具体可见通用 Spine 运行时指南中的 Applying Animations 一节所述.

时间比例(Scale)

你可以设置 skeleton 动画组件的时间比例,以减慢或加快动画的播放速度。用于推进动画的 delta 时间会乘上时间比例来决定动画播放速度,例如时间比例为 0.5 时,动画会减慢到正常速度的一半,时间比例为 2 时,动画会加速为正常速度的两倍。

C#

```
1. float timeScale = skeletonAnimation.timeScale;
2. skeletonAnimation.timeScale = 0.5f;
```

设置动画

要设置一个动画,需要确定轨道索引(index)、动画名称和是否循环动画:

C#

```
1. TrackEntry entry = skeletonAnimation.AnimationState.SetAnimation(trackIndex,
    "walk", true);
```

C#

```
1. // using properties
2. [SpineAnimation] public string animationProperty = "walk";
3. ...
4. TrackEntry entry = skeletonAnimation.AnimationState.SetAnimation(trackIndex,
    animationProperty, true);
```

你也可以用传入 AnimationReferenceAsset 作为参数而非动画名字符串.

C#

```
1. // using AnimationReferenceAsset
2. public AnimationReferenceAsset animationReferenceAsset; // assign a generate
   d AnimationReferenceAsset to this property
3. ...
4. TrackEntry entry = skeletonAnimation.AnimationState.SetAnimation(trackIndex,
    animationReferenceAsset, true);
```

动画队列

要排队动画,需要提供轨道索引、动画名称、是否循环播放该动画,以及该动画在轨道上开始播放的延迟时间(以秒计).

C#

```
TrackEntry entry = skeletonAnimation.AnimationState.AddAnimation(trackIndex, "run", true, 2);
```

C#

```
// using properties
[SpineAnimation] public string animationProperty = "run";
...
TrackEntry entry = skeletonAnimation.AnimationState.AddAnimation(trackIndex, animationProperty, true, 2);
```

设置和队列空动画，清空轨道

skeleton 动画组件还提供了设置空动画、队列空动画、清除一个或全部轨道的方法。这些功能需求都与上文的队列动画的方法和参数类似.

C#

```
TrackEntry entry = skeletonAnimation.AnimationState.SetEmptyAnimation(trackIndex, mixDuration);
entry = skeletonAnimation.AnimationState.AddEmptyAnimation(trackIndex, mixDuration, delay);
skeletonAnimation.AnimationState.ClearTrack(trackIndex);
skeletonAnimation.AnimationState.ClearTracks();
```

轨道条目(Entry)

所有的方法都会返回一个 TrackEntry 对象,可以通过它来进一步定制具体动画的回放,以及定制绑定到轨道条目的具体事件的委托。详见下文的*处理 AnimationState 事件*一节。

注意：返回的轨道条目只在底层动画状态中删除了对应动画之前有效。Unity 的垃圾收集器会自动释放它们。在接收到轨道条目的销毁事件(dispose event)后,它就无法再被存储或访问.

C#

```
TrackEntry entry = ...
entry.EventThreshold = 2;
float trackEnd = entry.TrackEnd;
```

处理 AnimationStates 事件

底层的 `AnimationState` 回放动画时,将引发下列事件来通知监听器(listener):

1. 动画**已开始(started)**.
2. 动画**被中断(interrupted)**，例如清空一条轨道或设置了一个新的动画时.
3. 动画无中断地播放**已完成(completed)**，如果是循环动画，该事件可能会多次出现.
4. 动画**已停止(ended)**.
5. 动画和其所处的 `TrackEntry` **已被销毁(disposed)**.

6. 触发了用户定义的**事件(event)**.

注意:当设置了一个新动画时,如果中断了之前的动画,将不会引发完成事件,而会引发中断和停止事件。

skeleton 动画组件提供了 C#代码可以绑定的委托,以便对全部轨道上全部队列中的全部动画的事件做出反应。监听器也可以只与特定 `TrackEntry` 的委托绑定。所以你可以注册到例如 `AnimationState.Complete` 事件来处理任何后续的动画 `Complete` 事件的事件回调(event callbacks),或者注册到 `TrackEntry.Complete` 事件来处理由某个入队动画引发的 `Complete` 事件.

C#

在处理 `AnimationState` 事件的类中添加欲监听事件的委托:

C#

```
1. SkeletonAnimation skeletonAnimation;
2. Spine.AnimationState animationState;
3.
4. void Awake () {
5.    skeletonAnimation = GetComponent<SkeletonAnimation>();
6.    animationState = skeletonAnimation.AnimationState;
7.
8.    // registering for events raised by any animation
9.    animationState.Start += OnSpineAnimationStart;
10.   animationState.Interrupt += OnSpineAnimationInterrupt;
11.   animationState.End += OnSpineAnimationEnd;
12.   animationState.Dispose += OnSpineAnimationDispose;
13.   animationState.Complete += OnSpineAnimationComplete;
14.
15.   animationState.Event += OnUserDefinedEvent;
16.
17.   // registering for events raised by a single animation track entry
18.   Spine.TrackEntry trackEntry = animationState.SetAnimation(trackIndex, "wa
lk", true);
19.   trackEntry.Start += OnSpineAnimationStart;
20.   trackEntry.Interrupt += OnSpineAnimationInterrupt;
21.   trackEntry.End += OnSpineAnimationEnd;
22.   trackEntry.Dispose += OnSpineAnimationDispose;
23.   trackEntry.Complete += OnSpineAnimationComplete;
24.   trackEntry.Event += OnUserDefinedEvent;
25. }
26.
27. public void OnSpineAnimationStart(TrackEntry trackEntry) {
28.   // Add your implementation code here to react to start events
29. }
30. public void OnSpineAnimationInterrupt(TrackEntry trackEntry) {
31.   // Add your implementation code here to react to interrupt events
```

```
32. }
33. public void OnSpineAnimationEnd(TrackEntry trackEntry) {
34.    // Add your implementation code here to react to end events
35. }
36. public void OnSpineAnimationDispose(TrackEntry trackEntry) {
37.    // Add your implementation code here to react to dispose events
38. }
39. public void OnSpineAnimationComplete(TrackEntry trackEntry) {
40.    // Add your implementation code here to react to complete events
41. }
42.
43.
44. string targetEventName = "targetEvent";
45. string targetEventNameInFolder = "eventFolderName/targetEvent";
46.
47. public void OnUserDefinedEvent(Spine.TrackEntry trackEntry, Spine.Event e) {

48.
49.    if (e.Data.Name == targetEventName) {
50.       // Add your implementation code here to react to user defined event
51.    }
52. }
53.
54. // you can cache event data to save the string comparison
55. Spine.EventData targetEventData;
56. void Start () {
57.    targetEventData = skeletonAnimation.Skeleton.Data.FindEvent(targetEventNa
    me);
58. }
59. public void OnUserDefinedEvent(Spine.TrackEntry trackEntry, Spine.Event e) {

60.
61.    if (e.Data == targetEventData) {
62.       // Add your implementation code here to react to user defined event
63.    }
64. }
```

更多详情请参考 Spine API 文档 .

Coroutine Yield 命令

spine-unity 运行时提供了几个用于 Unity Coroutines 的 yield 指令。如果你是 Unity Coroutines 的新手,Coroutine 教程和 Coroutine 文档是值得一看的.

以下是可供使用的 yield 指令:

1. **WaitForSpineAnimation**. 等待，直到 `Spine.TrackEntry` 引发一个具体事件.

C#

```
1. var track = skeletonAnimation.state.SetAnimation(0, "interruptible", false);
2. var completeOrEnd = WaitForSpineAnimation.AnimationEventTypes.Complete |
3.                               WaitForSpineAnimation.AnimationEventTypes.End;
4. yield return new WaitForSpineAnimation(track, completeOrEnd);
```

2. **WaitForSpineAnimationComplete**. 等待，直到 `Spine.TrackEntry` 引发一个 `Complete` 事件.

C#

```
1. var track = skeletonAnimation.state.SetAnimation(0, "talk", false);
2. yield return new WaitForSpineAnimationComplete(track);
```

3. **WaitForSpineAnimationEnd**. 等待，直到 `Spine.TrackEntry` 引发一个 `End` 事件.

C#

```
1. var track = skeletonAnimation.state.SetAnimation(0, "talk", false);
2. yield return new WaitForSpineAnimationEnd(track);
```

4. **WaitForSpineEvent**. 等待，直到 `Spine.AnimationState` 引发了一个自定义的 `Spine.Event` 事件(在 Spine 编辑器中命名).

C#

```
1. yield return new WaitForSpineEvent(skeletonAnimation.state, "spawn bullet");
2. // You can also pass a Spine.Event's Spine.EventData reference.
3. Spine.EventData spawnBulletEvent; // cached in e.g. Start()
4. ..
5. yield return new WaitForSpineEvent(skeletonAnimation.state, spawnBulletEvent);
```

注意: 和 Unity 内置的 yield 指令一样, spine-unity 的 yield 的实例也可以重复使用，以避免额外的内存分配.

教程页

你可以在这里找到关于 spine-unity 事件的教程.

在编码中使用字符串属性特性(String Property Attributes)

在检查器中手动输入动画名称等字符并不方便。因此，spine-unity 为提供了一种将字符串参数作为下拉(popup)字段的方式。可以在 `string` 属性前加上下文列出的属性特性(property attribute)，以将其自动转为一个可下拉选择选择的字段，比如下拉列出一个 skeleton 上的所有可用动画。如果你在 Spine 组件中看到过某个字段，那么也可以在你自定义的组件中通过属性特性下拉相同字段。下表展示了可用的属性特性:

C#

```
1. [SpineBone] public string bone;
2. [SpineSlot] public string slot;
3. [SpineAttachment] public string attachment;
4. [SpineSkin] public string skin;
5. [SpineAnimation] public string animation;
6. [SpineEvent] public string event;
7. [SpineIkConstraint] public string ikConstraint;
8. [SpineTransformConstraint] public string transformConstraint;
9. [SpinePathConstraint] public string pathConstraint;
```

可以查看 spine-unity 包中附带的 example scenes 在用的字符串属性特性.

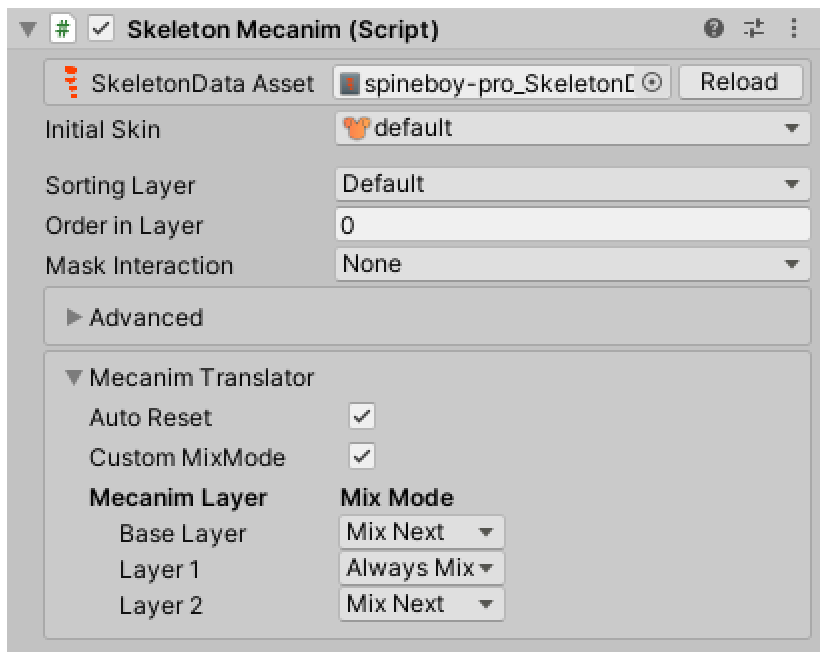
SkeletonMecanim 组件

`SkeletonMecanim` 组件是在 Unity 中使用 Spine skeleton 的三种方式之一: SkeletonAnimation, SkeletonMecanim 和 SkeletonGraphic (UI)。

`SkeletonMecanim` 组件是 SkeletonAnimation 组件的 Mecanim 版本，它使用 Unity 的 Mecanim 动画系统而非 Spine 的动画系统。与 SkeletonAnimation 组件一样，你可以将 Spine skeleton 附加到一个 GameObject 上，为它制作动画，处理动画事件等等。相较于 SkeletonAnimation ，它有一些限制和额外的要求:

注意：与 SkeletonAnimation 组件相比，如果要从前一动画的时间轴状态平滑地过渡(动画混合), `SkeletonMecanim` 额外要求后续动画的时间轴首帧为关键帧。更多信息请参见下文的需要的额外关键帧一节。

注意：`TrackEntry.AttachmentThreshold` 和类似的混合阈值功能在 `SkeletonMecanim` 上不可用.



`SkeletonMecanim`组件提供的参数与 SkeletonAnimation 组件类似，请查阅 SkeletonAnimation 部分以了解更多信息.

需要额外关键帧

为了平滑地将一个时间轴状态(如骨骼旋转)从一个动画 mix out 到下一个动画，后一个动画在 setup pose 时，需要在首帧添加一个关键帧。否则前一个动画的时间轴状态不会被清除。这是 SkeletonMec `SkeletonMecanim` anim 相较于 SkeletonAnimation 的一个缺点

小示例：一个 `idle` 动画应平滑地跟在前一个 `jump` 动画后。假设 `jump` 动画结束时，骨骼 `bone1` 和 `bone2` 的关键帧在非 setup-pose 位置上，那么为了正确地 mix out 到 `jump` 状态，必须在 `idle` 动画的首帧上为 `bone1` 和 `bone2` 添加关键帧(setup pose 或其他你自定义的 pose).

`Auto Reset` 这一参数会重置动画状态，但会在动画过渡结束时粗糙地进行 mix out，无法实现平滑的过渡效果.

另外，在将 skeleton 导出为`.json` 或`.skel.bytes` 时，一定要禁用 `Animation cleanup`，否则将无法导出 setup pose 对应的关键帧!

动画混合控制的参数

`SkeletonMecanim` 还暴露了以下参数:

- *Mecanim Translator*
  - *Auto Reset*. 当其置为 `true`，在一个动画结束时，skeleton 状态会 mix out 为 setup pose。当一个动画改变了附件的可见性状态时，这就特别重要了：当 mix out 时，附件可见性将变为 setup pose 状态，否则当前的附件状态将保持到另一个动画的关键帧再度设置这个附件.
  - *Custom MixMode*. 当禁用时会根据图层的混合模式使用推荐的 `MixMode` 。而当启用时，会在下方显示 `Mix Modes` 设置让你单独为每个 Mecanim 图层指定 `Mix Modes`.

- *Mix Modes*. 当上文的 `Custom MixMode` 设置启用时显示。这个设置决定了连续动画和跨图层动画间的混合模式.
    - Mix Next *(推荐用于* `Base Layer` *和* `Override` *图层)*
      应用于前一条轨道，然后按照 Mecanim 过渡权重 mix in 到下一条轨道.
    - Always Mix *(推荐用于* `Additive` *图层)*
      淡出前一条轨道(当 `Auto Reset` 启用时可能是淡出到 setup pose)，并使用 Mecanim 过渡权重 mix in 到下一条轨道。请注意，在基础图层(base layer)上使用时可能会造成动画出现非预期的浸润(dipping)效果.
    - Hard *(以前叫* `Spine Style`*)*
      立即应用于下一条轨道.

`Auto Reset` 和图层 `Mix Mode` 参数的结果

当动画过渡时有四个 pose -- `current state last frame`、`setup pose`、`previous clip pose` 和 `new clip pose` -- 其组合结果如下::

1. 开始于 `current state last frame` *(或者在 SkeletonMecanim 的更新前对改帧的修改)*.
2. 应用 `setup pose`:
    - 当 `Auto Reset` 启用, `setup pose` 会取代 `current state last frame`.
    - 当 `Auto Reset` 禁用, `setup pose` 将不参与动画混合.
3. 应用 `previous clip pose`:
    - 当混合模式置为 `Always Mix` 时 , `previous clip pose` 会与当前状态混合 (所以当 `Auto Reset` 启用时会与 `setup pose` 混合).
    - 当混合模式置为 `Hard` 或 `Mix Next` 时, `previous clip pose` 将按 100%权重应用, 完全覆盖当前状态 (所以它会覆盖 `setup pose`).
4. 应用 `new clip pose`:
    - 当混合模式被置为 `Always Mix` 或 `Mix Next` 时，`new clip pose` 将与当前状态混合。因此启用 `Auto Reset` 的 `Always Mix` 是对 `setup pose`、`previous clip pose` 和 `new clip pose` 三者的混合
    - 当混合模式被置为 `Hard` 时, the `new clip pose` 将按 100%权重应用, 完全覆盖已应用的 pose.

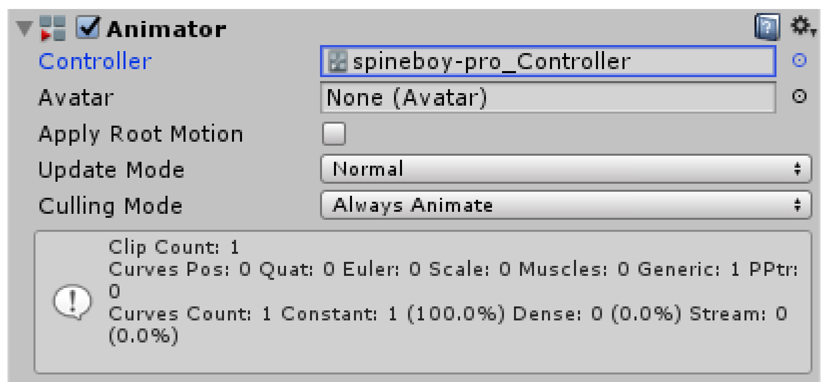
下表显示了当前一剪辑片段(Clip) `P` 和新的剪辑片段 `N` 都修改了相同的时间轴值时的情况，例如相同的骨骼旋转。 `S` 代表启用了 `Auto Reset` 时的 setup pose，如果禁用了 `Auto Reset`，则代表当前状态(例如前一帧)。变量 `w` 表示过渡权重(过渡开始时为 0，过渡结束时为 1)。每个图层的(推荐)混合模式以粗体标出. The default (recommended) mix mode at each layer blend mode is highlighted in bold.(blend mode 和 mix mode 的区别是什么?该如何翻译?)

	Always Mix	Mix Next	Hard
Base Layer	lerp(lerp(S, P, 1-w), N, w)	**lerp(P, N, w)**	N
Override	lerp(lerp(lower_layers_result, P, (1-w) * layer_weight), N, w * layer_weight)	**lerp(** **lerp(lower_layers_result,** **P, layer_weight),** **N, w * layer_weight)**	lerp(lower N, layer_weig
Additive	**lower_layers_result +** **layer_weight * lerp(P, N, w))**	counts as Always Mix	lower_laye layer_weig

缩写	含义
S	**S**etup pose
P	**P**revious clip pose
N	**N**ew clip pose
w	Transition **w**eight
lerp(a, b, weight)	**L**inear int**erp**olation from a to b by weight.

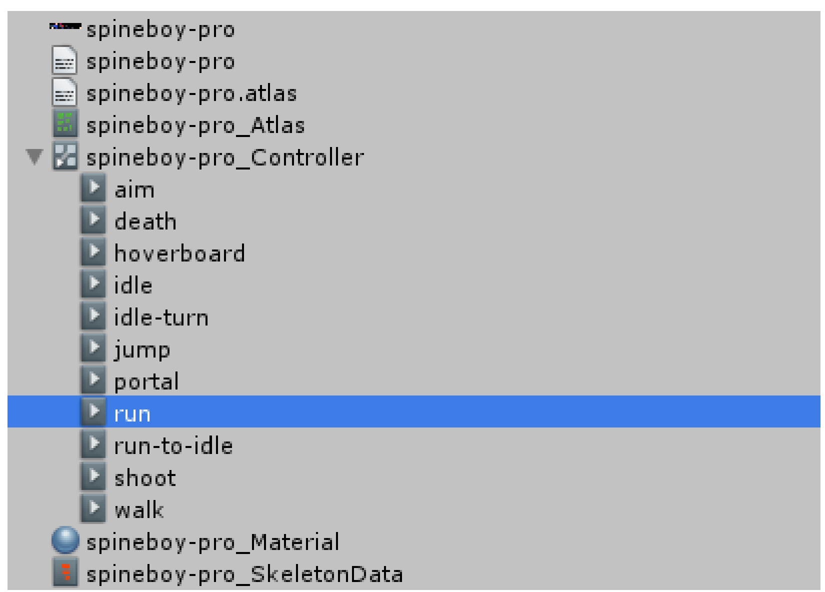
SkeletonMecanim 的控制器(Controller)和动画器(Animator)

`SkeletonMecanim` 组件使用控制器(Controller)资产的方式和 Unity Mecanim 的 `Animator` 组件一样. 当通过拖放将一个 skeleton 实例化为 `SkeletonMecanim` 时会自动生成并分配控制器资产.



注意: 当启用 `Apply Root Motion` 时，SkeletonMecanimRootMotion 组件会自动添加到 `SkeletonMecanim` GameObject 中.

你可以将 Spine 动画拖放到 Animator 面板来向控制器的动画状态机添加动画。添加后的动画会显示在控制器资产下.



在 SkeletonDataAsset 中设置的 Mix duration 值将被 `SkeletonMecanim` 忽略。而 Mecanim 是通过 Animator 面板的过渡时间来设置的。

SkeletonMecanim 事件

当使用 `SkeletonMecanim` 时，事件被存储在每个 `AnimationClip` 中，并像其他 Unity 动画事件一样被引发。例如，如果你在 Spine 中把事件命名为" `Footstep`"，那么需要在 `SkeletonMecanim` 的 GameObject 上添加一个 `MonoBehaviour` 脚本，使用名为 `Footstep()`的方法来处理该事件。当在 Spine 中使用文件夹时，方法名则是文件夹名和动画名的拼接。例如，若前文中的事件被放在一个名为 `Foldername` 的文件夹中时，方法名将为 `FoldernameFootstep()`.

C#

```
1. public class YourComponentReceivingEvents : MonoBehaviour {
2.    // to capture event "Footstep" when it's placed outside of folders
3.    void Footstep() {
4.       Debug.Log("Footstep event received");
5.    }
6.
7.    // to capture event "Footstep" when it's placed in a folder named "Folder
   name"
8.    void FoldernameFootstep () {
9.       Debug.Log("Footstep (in folder Foldername) received");
10.    }
11. }
```

关于 Unity Mecanim 事件的更多信息，请参见 Unity's Documentation on Animation Events.

SkeletonGraphic 组件

`SkeletonGraphic` 组件是 Unity 组件是三种在 Unity 中使用 Spine skeleton 的方式之一。这些三种方法是: SkeletonAnimation, SkeletonMecanim 和 SkeletonGraphic (UI).

The `SkeletonGraphic` 组件是 SkeletonAnimation 组件的平替，它使用 Unity 的 UI 系统进行布局、渲染和遮罩交互。与 SkeletonAnimation 组件一样，它允许你将 Spine skeleton 添加到 GameObject 中，将其制成动画，对动画事件做出反应等等.

为何要用这一 UI 组件

Unity UI(UnityEngine.UI)使用 `Canvas` 和 `CanvasRenderers` 系统来分类和管理其可渲染对象。内置的可渲染的 UI 组件——如 `Text`、`Image` 和 `RawImage`——都依赖于 `CanvasRenderer` 来正常工作。将 `MeshRenderers` (如默认的 Cube 对象)或 `SpriteRenderers` (如 Sprite)等对象置入 UI，Unity 将不会在 `Canvas` 中渲染它们。`SkeletonAnimation` 使用的是 `MeshRenderer`，因此行为方式也是一样的。因此，spine-unity 运行时提供了 `SkeletonAnimation` 的平替组件 `SkeletonGraphic`，它是使用 `CanvasRenderer` 组件进行渲染的 `UnityEngine.UI.MaskableGraphic` 的子类。

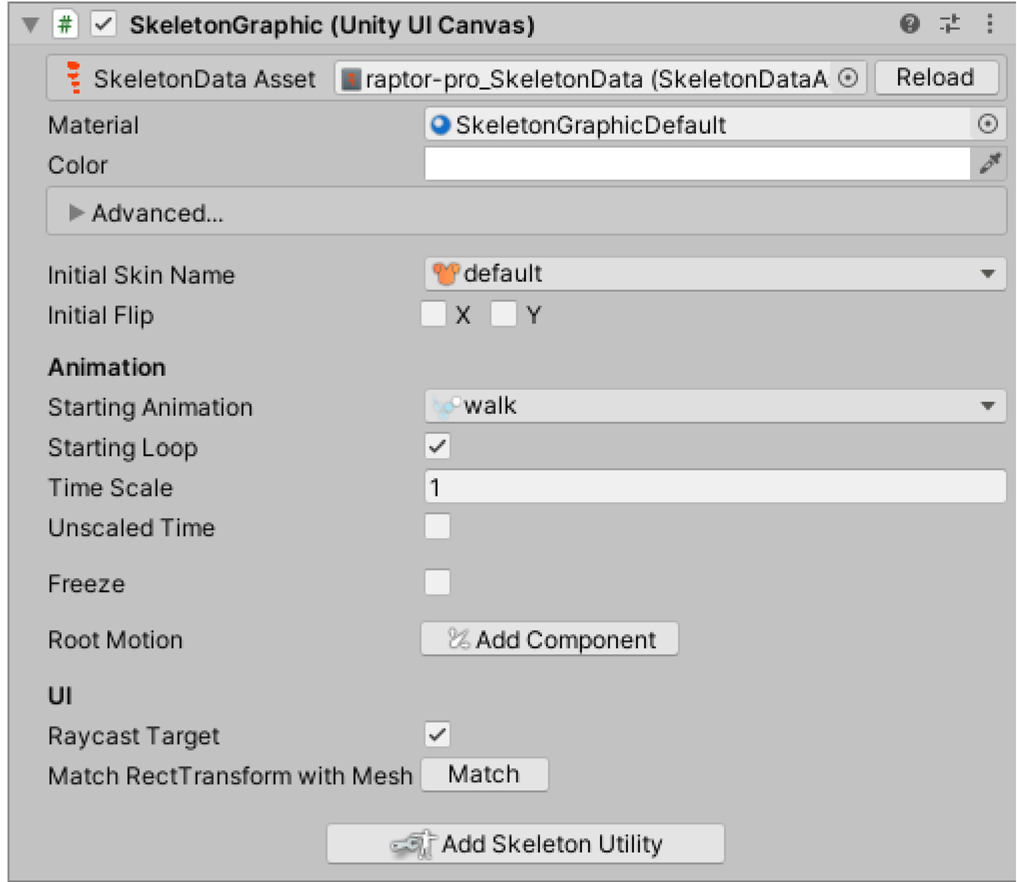
重要提示: 由于 Unity `CanvasRenderer` 的限制，SkeletonGraphic 默认情况下限制为单一 texture。你可以在 `SkeletonGraphic` 组件的检查器中启用 `Advanced - Multiple CanvasRenderers` 来为每个子网格生成一个子 `CanvasRenderer` `GameObject` 以突破 texture 限制。出于性能方面的考虑，最好在尽可能避免这样做。这意味着 UI 中使用的 Skeletons 应被打包成一个单 texture(单页)atlas 而非多页 atlas。关于如何将附件打包成单页 atlas texture，请参见高级 – 单页 Atlas Texture 导出与 SkeletonGraphic 部分.

调整 RectTransform 区间来实现正确可见性

`SkeletonGraphic` 的可见性是通过其 RectTransform 的边界决定的. 当 Skeleton 通过拖放实例化为 `Canvas` GameObject 的子对象时，RectTransform 边界会自动匹配于 initial pose。你也可以点击 `Match RectTransform with Mesh` 按钮手动将 RectTransform 置为适合当前 pose 的边界。重要的是，RectTransform 的边界不能小于网格，否则如 `RectMask2D` 这样的组件会当作 RectTransform 完全位于网格外部而在绘制时忽略 skeleton，即使部分应被渲染的网格仍在 RectTransform 内部。当五种 Transform 模式之一的 RectTransform 工具处于激活状态时，当前的 RectTransform 边界会显示在场景视图中。

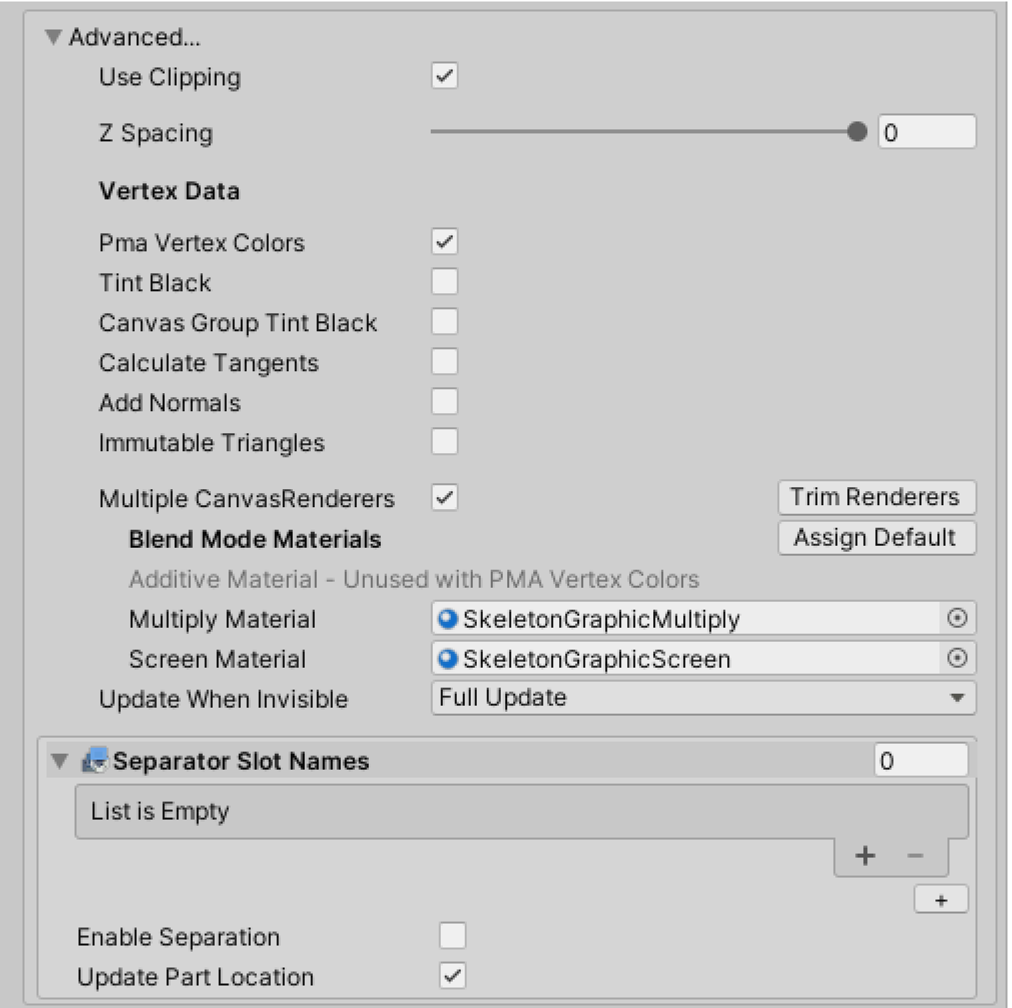
参数

`SkeletonGraphic` 提供了与 SkeletonAnimation 组件类似的参数，请参考 SkeletonAnimation 部分以获得更多信息.



`SkeletonGraphic` 检查器暴露了以下额外参数:

- *Unscaled Time*. 当设置为 `true` 时，更新将按照 Time.unscaledDeltaTime 而非 Time.deltaTime 进行。这对独立于慢动作效果的动画 UI 元素很有用.
- *Freeze*. 当置为 `true` 时，`SkeletonGraphic` 将不再被更新.
- *Match RectTransform with Mesh* 你可以通过点击 `Match` 按钮使 `SkeletonGraphic` 的 RectTransform 匹配其当前 pose。重要的是，RectTransform 的边界不能小于网格，否则如 `RectMask2D` 这样的组件会当作 RectTransform 完全位于网格外部而在绘制时忽略 skeleton，即使部分应被渲染的网格仍在 RectTransform 内部.



- *高级参数*
  - *Use Clipping.* 当置为 `false` 时，所有 [Spine clipping attachments]() 都会被忽略.
  - *Canvas Group Tint Black.* 只在使用 `SkeletonGraphic Tint Black` 着色器时应被启用。当 `Tint Black` 被禁用时不产生效果。在 `CanvasGroup` 下的 `SkeletonGraphic` 槽位中使用 `Additive` 混合模式时启用之。启用后，`Additive` Alpha 值将存储在 `uv2.g` 而非 `color.a`，以便跟踪 `CanvasGroup` 对 `color.a` 的修改.
  - *Multiple CanvasRenderers.* 当置为 `true`，`SkeletonGraphic` 不再使用单个 `CanvasRenderer`，而是为每个绘制调用(submesh)自动创建所需的子 `CanvasRenderer` GameObjects。这可以用来突破[单txture 限制]()，但会产生额外的性能开销.
    - *Blend Mode Materials.* 允许为每个槽位的 Blend 模式使用不同的 SkeletonGraphic materials。这里只能使用 `"Spine/SkeletonGraphic *"`或其他与 CanvasRenderer 兼容的 material.

        - 选择 `Assign Default` 来设置默认的混合模式 `SkeletonGraphicAdditive`, `SkeletonGraphicMultiply` and `SkeletonGraphicScreen` 的 materials. (这里的原文我不太明白)
          注意: 确保 `SkeletonDataAsset` 有一个正确的 `Blend Mode Materials` 设置，因为 `SkeletonGraphic` 的 blend 模式 material 的依赖于 `SkeletonDataAsset` 的 material。当启用 `PMA Vertex Colors` 时，`Additive Material` 将会被忽略.
    - *Update When Invisible.* 设置当 MeshRenderer 不可见时使用何种更新模式。当网格再次可见时更新模式会自动置为 `UpdateMode.FullUpdate`.
    - *Separator Slot Names.* 设置当 *Enable Separation* 置为 `true` 时渲染分离位置的槽位名称。关于渲染分离的信息，请参考 SkeletonRenderSeparator 章节，但注意，使用 `SkeletonGraphic` 时无需额外组件.
    - *Enable Separation.* 渲染分离可以在此处直接启用，无需任何额外组件(如 `SkeletonRenderSeparator` 或用于 `SkeletonRenderer` 的 `SkeletonPartsRenderer` 组件)。启用后将自动创建额外的分离部件 GameObjects，也会附加相应的 `CanvasRenderer` GameObjects 到层级架构中。分离部件的 GameObjects 可以根据你的要求在层次结构中移动并重新设置层级关系，以便保证在 `Canvas` 中的绘制顺序.
    - *Update Part Location.* 当启用时，将更新分离部件 GameObject 的位置以匹配 `SkeletonGraphic` 的位置。这在将部件 re-parenting 到不同的 GameObject 时会有帮助.

示例场景

可以浏览示例场景 Spine Examples/Getting Started/6 Skeleton Graphic 来了解基本用法.
可以在 `Spine Examples/Other Examples/SkeletonRenderSeparator` 中找到一个高级示例场景，它展示了如何设置分离器(separator)槽位和修改绘制顺序.

SkeletonRenderer 组件

`SkeletonRenderer` 组件负责绘制 skeleton 的当前状态。它是 SkeletonAnimation 和 SkeletonMecanim 的基类.
注意: 多数时候均应使用 SkeletonAnimation、SkeletonMecanim 或 SkeletonGraphic (UI)。这些组件提供了丰富的动画控制方法。该组件只有在手动应用无过渡动画(如 UI 仪表元素中)时可能有用.
渲染是通过 `MeshRenderer` 组件中更新 procedural 网格进行的。该组件使用由 SkeletonDataAsset 引用的 texture atlas 资产来查找绘制 skeleton 附件所需的 textures 和 materials。你可以查阅 SkeletonAnimation 文档部分以了解更多信息。

也可以查看示例场景 `Spine Examples/Other Examples/SpineGauge` 来了解如何直接使用 `SkeletonRenderer` 组件.

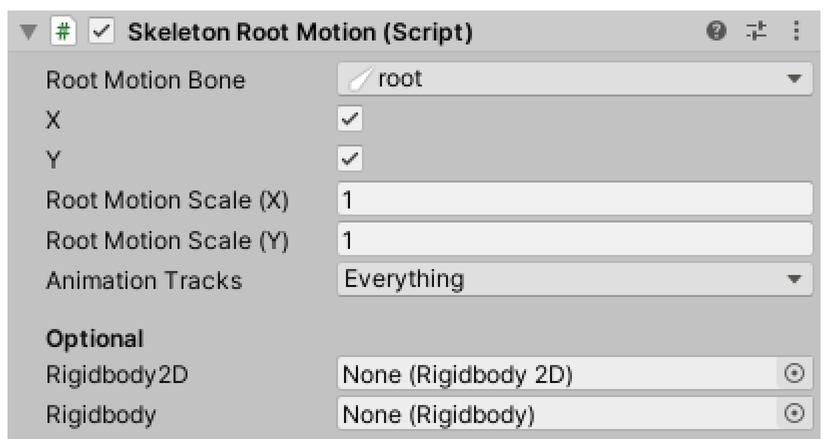
.

Utility 组件

SkeletonRootMotion

spine-unity 的三种 Spine skeleton 组件均支持 root motion。`SkeletonRootMotion` 组件可以附加到 SkeletonAnimation 和 SkeletonGraphic (UI) GameObjects 上，而且为 SkeletonMecanim 单独提供了 SkeletonMecanimRootMotion 组件。附加这个 root motion 组件类似于在 Unity Mecanim `Animator` 组件上启用 `Apply Root Motion` 参数。当启用时，会根据动画中选定的 `Root Motion Bone` 的运动来驱动角色位置。

注意： SkeletonMecanimRootMotion 组件仅用于 SkeletonMecanim 对象。当它用于 `SkeletonAnimation` 或 `SkeletonGraphic (UI)`组件时 `SkeletonRootMotion` 将无法正常工作.



参数

- *Root Motion Bone.* 其运动用于 root motion 的目标骨骼.
- *X.* 启用后，沿局部 X 轴的运动被用于 root motion.
- *Y.* 启用后，沿局部 Y 轴的运动被用于 root motion.
- *Root Motion Scale (X).* 应用于水平 root motion delta 的比例。可用于 delta 补偿，例如将一个跳跃动画的位移拉伸到所需的距离.
- *Root Motion Scale (Y)* 应用于垂直 root motion delta 的比例。可用于 delta 补偿，例如将一个跳跃动画的位移拉伸到所需的高度.
- *Animation Tracks.* 可以指定哪些动画轨道应包含在 root motion 的计算中.

可选参数

- *Rigidbody2D.* 当添加了 `Rigidbody2D` 组件时，运动将通过物理运动应用于给定的 `Rigidbody2D` 而非 `Transform` 组件.
- *Rigidbody.* 当添加了 `Rigidbody` 组件时，运动将通过物理运动应用于给定的 `Rigidbody` 而非 `Transform` 组件.

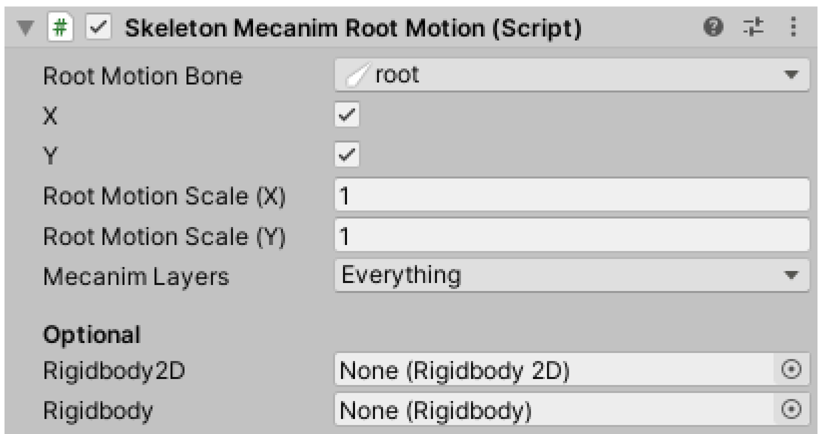
注意: `SkeletonRootMotion` 类提供了 `AdjustRootMotionToDistance()`和其他方法来进行 Delta 补偿。Delta 补偿可以用于特定场景(例如拉伸一个跳跃动画到一个给定的距离)。Root

motion 可以在动画开始时调整，也可以通过 `skeletonRootMotion.AdjustRootMotionToDistance(targetPosition - transform.position, trackIndex);`在某一帧上调整.

SkeletonMecanimRootMotion

这个组件是 SkeletonRootMotion 组件的 SkeletonMecanim 版，与 SkeletonMecanim 组件一起使用.

当 Unity Mecanim `Animator` 的 `Apply Root Motion` 参数被启用时，`SkeletonMecanimRootMotion` 组件会自动添加到 skeleton 的 GameObject 上。要移除 `SkeletonMecanimRootMotion` 组件，必须首先禁用 Animator 的 `Apply Root Motion` 参数.



参数

- *Root Motion Bone.* 其运动用于 root motion 的目标骨骼.
- *X.* 启用后，沿局部 X 轴的运动被用于 root motion.
- *Y.* 启用后，沿局部 Y 轴的运动被用于 root motion.
- *Root Motion Scale (X).* 应用于水平 root motion delta 的比例。可用于 delta 补偿，例如将一个跳跃动画的位移拉伸到所需的距离.
- *Root Motion Scale (Y) .* 应用于垂直 root motion delta 的比例。可用于 delta 补偿，例如将一个跳跃动画的位移拉伸到所需的高度.
- *Mecanim Layers.* 允许你指定哪些 Mecanim 图层应包括在 root motion 计算中.

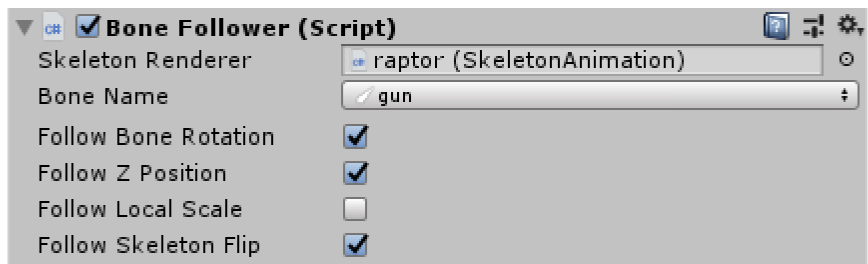
可选参数

- *Rigidbody2D.* 当添加了 `Rigidbody2D` 组件时，运动将通过物理运动应用于给定的 `Rigidbody2D` 而非 `Transform` 组件.
- *Rigidbody.* 当添加了 `Rigidbody` 组件时，运动将通过物理运动应用于给定的 `Rigidbody` 而非 `Transform` 组件.

注意: `SkeletonMecanimRootMotion` 类提供了 `AdjustRootMotionToDistance()`和其他方法来进行 Delta 补偿。Delta 补偿可以用于特定场景(例如拉伸一个跳跃动画到一个给定的距离)。Root motion 可以在动画开始时调整，也可以通过 `skeletonRootMotion.AdjustRootMotionToDistance(targetPosition - transform.position, trackIndex);`在某一帧上调整.

BoneFollower

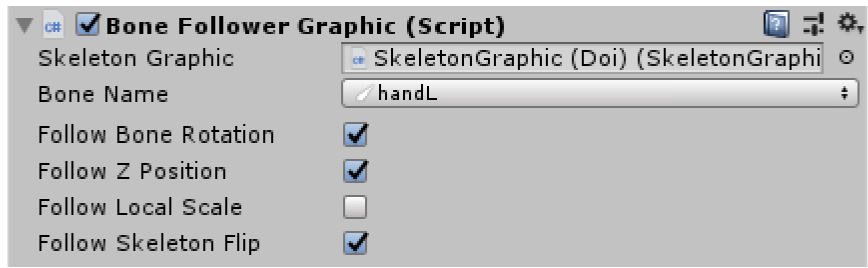
这个组件引用一个 SkeletonAnimation 组件的骨骼，并在每次 `Update` 时将自己的 transform 设置为该骨骼的 transform.



注意: SkeletonGraphic 对象有专用的 BoneFollowerGraphic 组件.
与 SkeletonUtilityBone 组件相反，`BoneFollower` 可以作为一个单独的 GameObject 使用，没有任何父级骨骼对象.
使用它可以让粒子系统等对象跟随 skeleton 上的某个骨骼.
你可以在示例场景 Spine Examples/Getting Started/4 Object Oriented Sample 中查看如何设置 `BoneFollower` 组件.

BoneFollowerGraphic

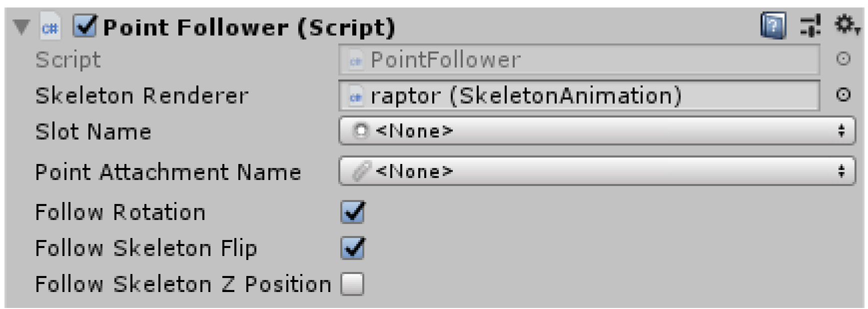
这个组件是 Bone Follower 组件的一个 SkeletonGraphic 版本，和 SkeletonGraphic 组件一起使用.



与 SkeletonUtilityBone 组件不同的是，`BoneFollowerGraphic` 可以作为一个单独的 GameObject 使用，没有任何父级骨骼对象.
使用它可以让粒子系统等对象跟随 skeleton 上的某个骨骼.
你可以在示例场景 Spine Examples/Getting Started/6 Skeleton Graphic 中查看如何设置 `BoneFollowerGraphic` 组件.

PointFollower

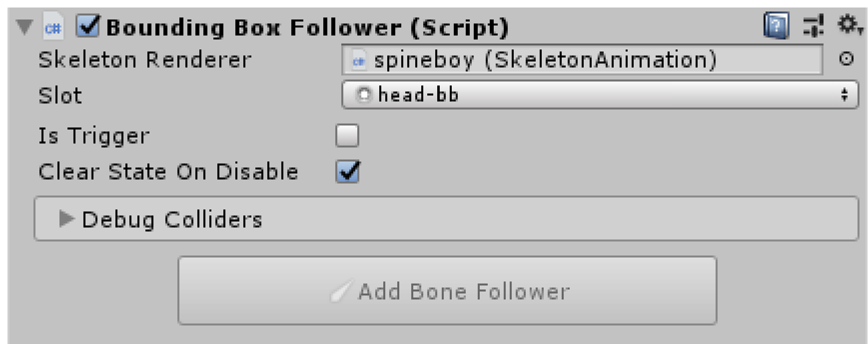
这个组件类似于 Bone Follower 组件，但是它跟随的是一个 PointAttachment 而非一个骨骼.



与 SkeletonUtilityBone 组件不同的是，`PointFollower` 可以作为一个单独的 GameObject 使用，没有任何父级骨骼对象.

BoundingBoxFollower

这个组件用于在 skeleton 的槽位中匹配一个边界盒(bounding box)。它会获取边界形状并将其传递给 `PolygonCollider2D` ，并在每一帧启用或禁用它以匹配当前动画.



注意：骨骼位置不会被自动跟踪，这就是为什么它通常与 `BoneFollower` 组件一起使用。你可以使用 `BoundingBoxFollower` 检查器中的 `Add Bone Follower` 按钮来创建和设置一个 `BoneFollower` 组件.

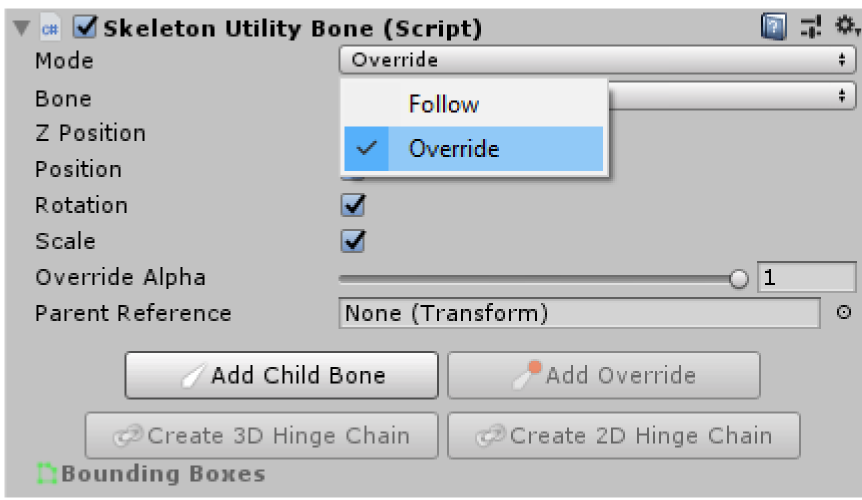
注意：顶点变形动画（随着时间的推移, Bounding Box 的顶点也会移动的动画）不会被跟随，它只关注初始形状

Vertex deformation animation (moving vertices of a Bounding Box over time in an animation) is not followed along, it only covers the initial shape.

更多信息请参见 Bone Follower.

SkeletonUtilityBone

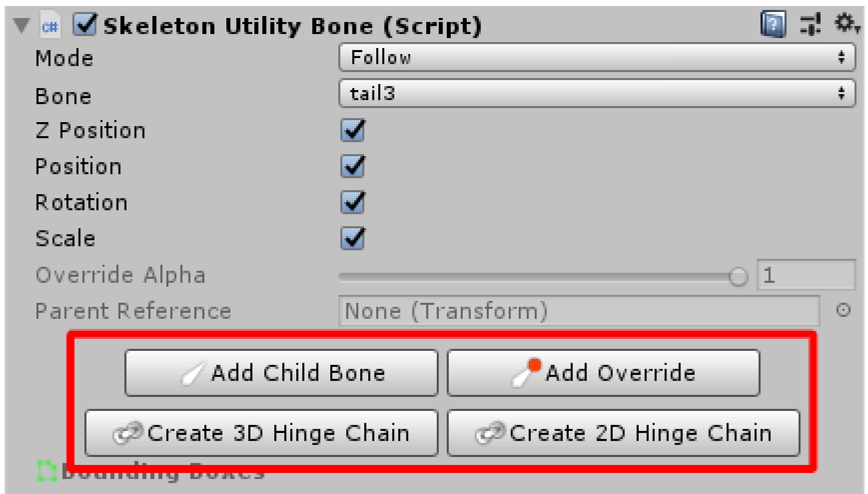
有时你需要在运行时以编码的方式修改骨骼位置，以以便对物理运动或用户输入做出反应.



SkeletonUtilityBone 组件提供了一个方便的接口，让 GameObjects 跟随一个骨骼位置，通过手动或 2D/3D 物理来覆盖一个骨骼的位置。它可以被配置为*跟随(follow)*本地骨骼位置或在每次 `Update` 时*覆盖(override)*它。当设置为 `Override` 时，该组件将在 SkeletonAnimation 组件更新世界 transform 之前设置骨骼位置.

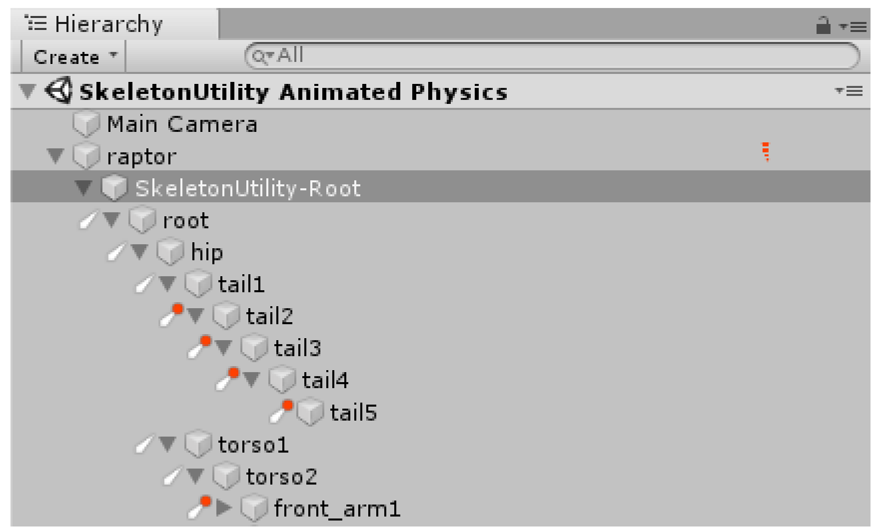
重要提示: `SkeletonUtilityBone` 使用局部 transform 值。它依赖于 `SkeletonUtilityBone` GameObjects 的层次结构，所以反映了 skeleton 的骨骼层次结构。快速创建 `SkeletonUtilityBone` 层次结构的推荐方法是使用下文描述的 `SkeletonUtility` 组件.

`SkeletonUtilityBone` 检查器还提供了一个接口来（选择性或递归性地）创建额外的子骨骼或创建一个 2D 和 3D 铰链(hinge chain).



一旦你创建了 `SkeletonUtilityBones` 的层次结构，层次结构面板会在 `SkeletonUtilityBone` GameObject 旁边显示不同的图标，取决于它的设置为

- `Follow`:
- `Override`:



示例用例

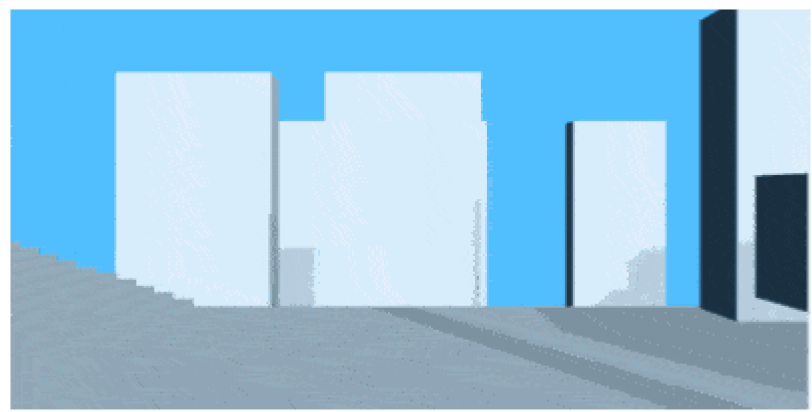
在 `Override` 模式下使用 `SkeletonUtilityBone`，例如让用户拖动 skeleton 上的一块骨头.

如果一个 GameObject 只跟随一个骨骼的位置，你可以使用 BoneFollower 组件来节省资源.

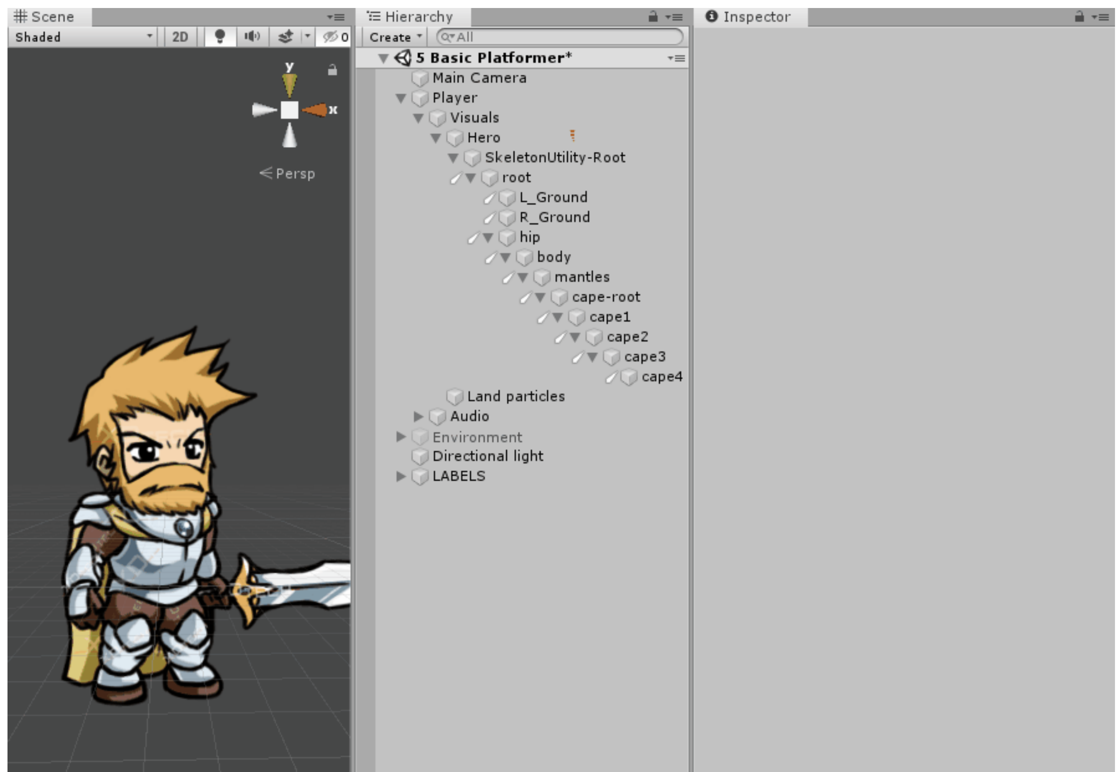
示例场景

你可以在 `Spine Examples/Other Examples/SkeletonUtility Animated Physics` 中找到一个演示 `SkeletonUtilityBone` 用法的示例场景。它展示了 `SkeletonUtilityBones` 节点是如何配置为 `Follow` 骨骼位置，成为 `Override` 节点的必要父层节点的.

创建 2D 和 3D 的物理铰链(Hinge Chains)



你可能想为你的角色的披风添加物理效果，让角色拖拽重物或者给它加个流星锤。spine-unity 运行时可以从现有的 SkeletonUtilityBone 层次结构（见创建 SkeletonUtilityBones 的层次结构）中生成 HingeJoint 或 HingeJoint2D 元素的 physics rig.

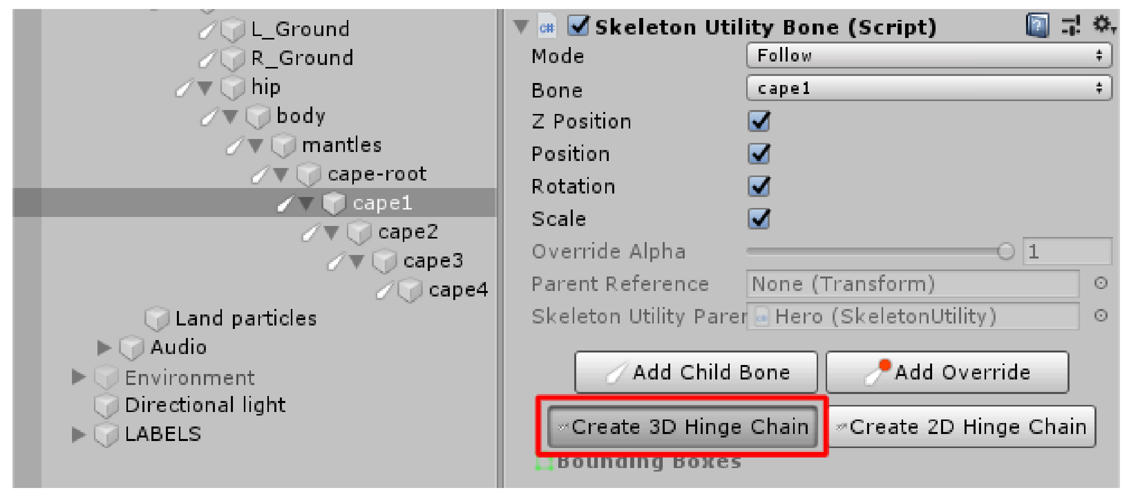


选择第一个 `SkeletonUtilityBone` 链元素，在检查器中选择 `Create 3D Hinge Chain` 或 `Create 2D Hinge Chain` 来生成 physics rig。被选中的元素和其全部 `SkeletonUtilityBone` 子元素都会转换为一个铰链。然后你可以调整 Rigidbody 的阻力(drag)和质量(mass)参数来调整效果。提高阻力值将使 Rigidbody 移动得更慢，可以创造出物体沉重或被空气阻碍的效果.

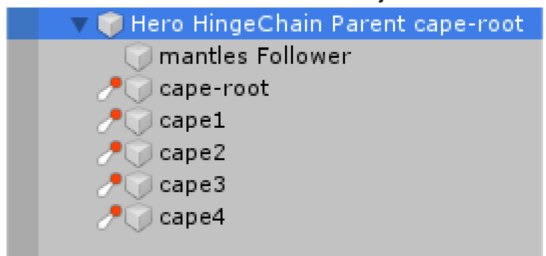
请注意，链根节点不再以 skeleton 的骨骼为父节点，而是放置在场景的最高层级。这是 Unity 中使用动量的一个要求。不要把链根节点再置入 skeleton 骨骼的层级中，否则链元素将不再受到 skeleton 运动的影响!

3D 铰链

1. 创建一个普通的 SkeletonUtilityBone 层级结构.
2. 在场景面板中选中第一个链元素，在检查器中选择 `Create 3D Hinge Chain` 来创建 3D 铰链 rig.



3. 这将从先前的父级（在本例中为 `cape-root`）移除链 GameObjects，并在场景的最高层放置一个全新的 `HingeChain Parent` GameObject。如上文所述，千万不要将这个 GameObject 重新放到到 skeleton 下面去!

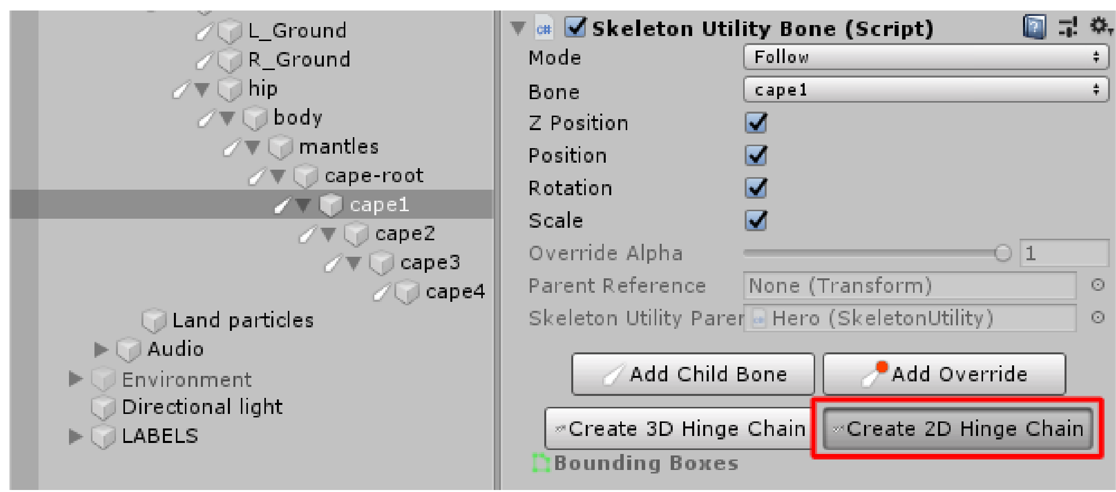


4. 调整链元素 Rigidbody 的阻力和质量参数来调整效果.

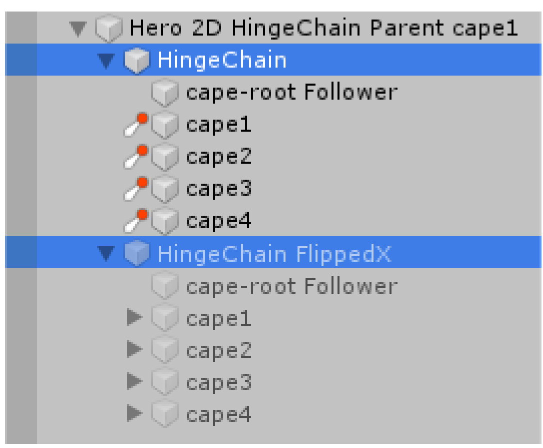
当 skeleton 翻转时，`HingeChain Parent` GameObject 将自动旋转 180 度来把铰链调整到翻转后的骨骼位置上.

2D 铰链

1. 创建一个普通的 SkeletonUtilityBone 层级结构.
2. 在场景面板中选中第一个链元素，在检查器中选择 `Create 2D Hinge Chain` 来创建 2D 铰链 rig.



3. 这将从先前的父级（在本例中为 `cape-root`）移除链 GameObjects，并在场景的最高层放置一个全新的 `HingeChain Parent` GameObject。如上文所述，千万不要将这个 GameObject 重新放到到 skeleton 下面去!



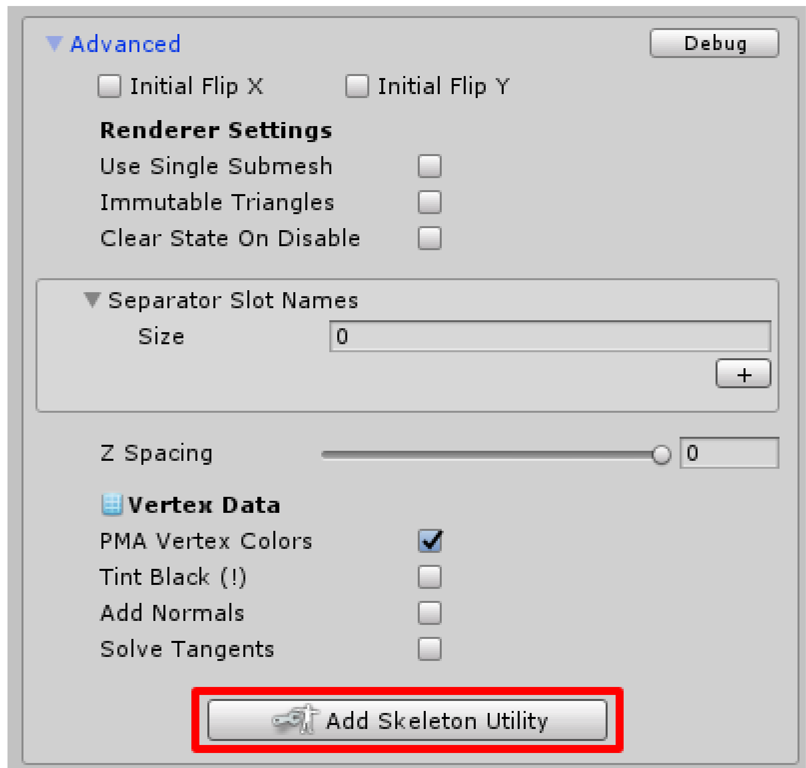
4. 调整链元素 Rigidbody2D 的阻力和质量参数来调整效果.

要注意这个 GameObject 包含两个子对象: `Hinge Chain` 和 `Hinge Chain FlippedX`。当翻转 skeleton 时，这些 GameObjects 将自动启用和停用，它们代表了各自朝向的铰链动画.

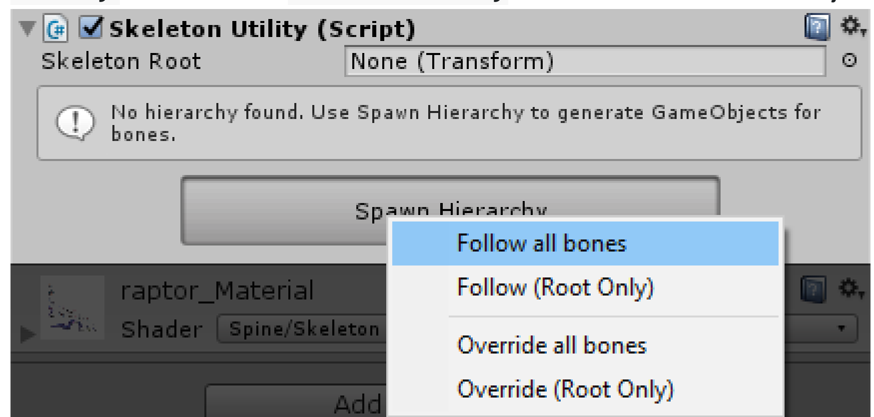
SkeletonUtility

创建 SkeletonUtilityBones 的层次结构

`SkeletonUtility` 组件提供了一个快速创建 `SkeletonUtilityBone` GameObjects 层次结构的方法，该层级结构它反映了 skeleton 的骨骼层次结构.



要添加一个 `SkeletonUtility` 组件: 选中 SkeletonAnimation 组件，在检查器中展开 `Advanced` 部分并点击 `Add Skeleton Utility`。一旦创建了该组件，`Add Skeleton Utility` 按钮将消失，`SkeletonUtility` 组件将被添加到 GameObject 中.

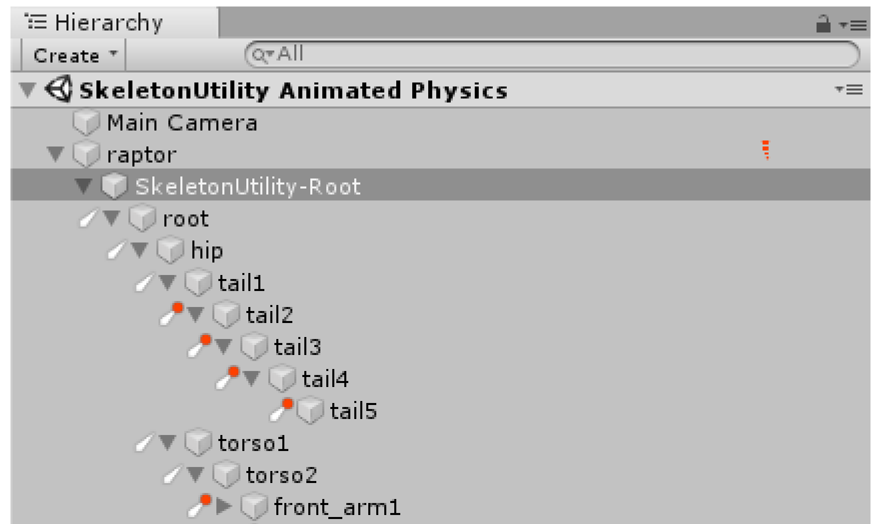


`SkeletonUtility` 组件提供了一个 `Spawn Hierarchy` 按钮，点击后会出现以下选项:

1. `Follow all bones` 为层次结构中的所有骨骼创建 `SkeletonUtilityBone` GameObject，设置为 `Follow` 模式

2. `Follow (Root Only)` 只创建 root `SkeletonUtilityBone` GameObject，设置为 `Follow` 模式
3. `Override all bones` 为层次结构中的所有骨骼创建 `SkeletonUtilityBone` GameObjects，设置为 `Override` 模式
4. `Override (Root Only)` 只创建 root `SkeletonUtilityBone` GameObject，设置为 `Override` 模式

然后可以单独配置每个 `SkeletonUtilityBone`，以便在必要时覆盖 skeleton 的骨骼位置.



注意: 你可以在以后通过 `SkeletonUtilityBone` 检查器添加额外的 `SkeletonUtilityBone` GameObjects, `Spawn Hierarchy` 功能可以做好大致的铺垫。你也可以删除多余的 `SkeletonUtilityBone` GameObjects 来节省资源。只需记住一定要保持父子关系不变，所以不要在层次结构里删除 GameObjects 或改变它们的父对象.

SkeletonUtilityConstraint

```
public abstract class SkeletonUtilityConstraint : MonoBehaviour {

    public abstract void DoUpdate ();
}
```

C#

它是 skeleton utility 约束子类的基类。它在父类 `SkeletonUtility` 处自动注册并进行相应的更新。

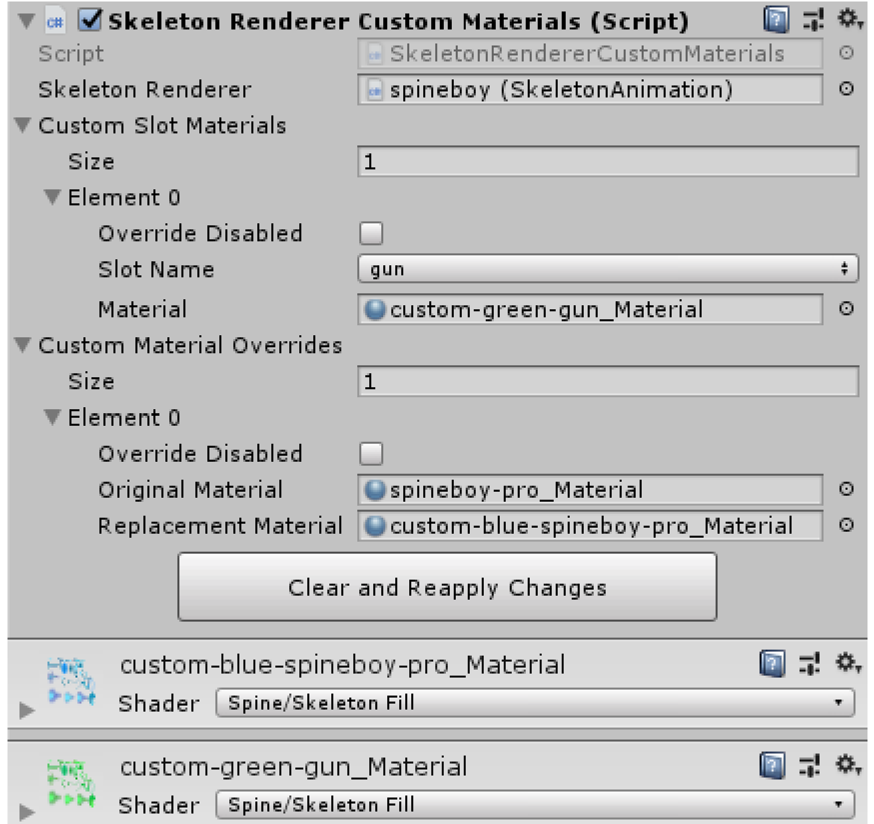
关于如何编写你自己的约束类，请参见示例约束类 `SkeletonUtilityGroundConstraint` 和 `SkeletonUtilityEyeConstraint`.

示例场景

spine-unity 运行时附带了演示上述约束的示例场景。你可以在 `Spine Examples/Other Examples/SkeletonUtility GroundConstraint` 和 `Spine Examples/Other Examples/SkeletonUtility Eyes` 中找到它们.

SkeletonRendererCustomMaterials

你可能想覆盖某个 skeleton 实例甚至某个槽位的 material。该组件提供了一个检查器接口，用于为 SkeletonRenderer（包括子类 SkeletonAnimation 和 SkeletonMecanim）设置自定义材质覆盖.

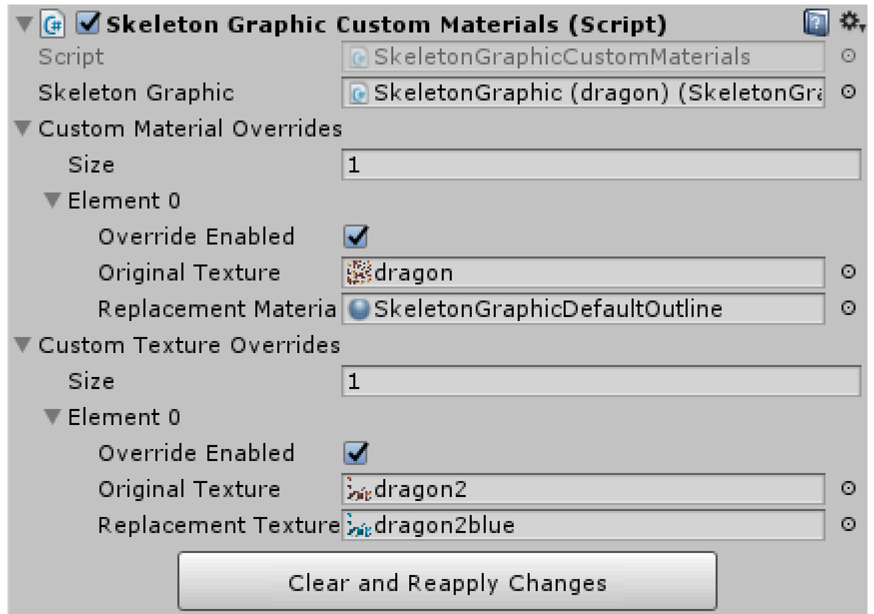


右击 SkeletonRenderer （或子类 SkeletonAnimation 和 SkeletonMecanim），并选择 `Add Basic Serialized Custom Materials` 来将此组件添加到渲染器中。在 `Custom Slot Materials` 数组中添加条目可覆盖某个槽位的 material，或 在 `Custom Material Overrides` 数组中添加条目，可以将整个 skeleton 的当前 material 替换为另一 material。请确保禁用 `Override Disabled`，这样才能启用各条目中设置的 material 覆盖.

注意: 该组件设计初衷里不包括通过代码进行交互。要通过代码为你的 SkeletonRenderer 动态地设置 material，应直接通过 `SkeletonRenderer.CustomMaterialOverride` 来访问 material 覆盖数组，或者 `SkeletonRenderer.CustomSlotMaterials` 来访问槽位 material 覆盖.

SkeletonGraphicCustomMaterials

该组件为 SkeletonGraphic 专用，是 SkeletonRendererCustomMaterials 的 SkeletonGraphic 版本。该组件也提供了检查器接口，用于为 SkeletonGraphic 设定自定义 material 和 texture 覆盖.



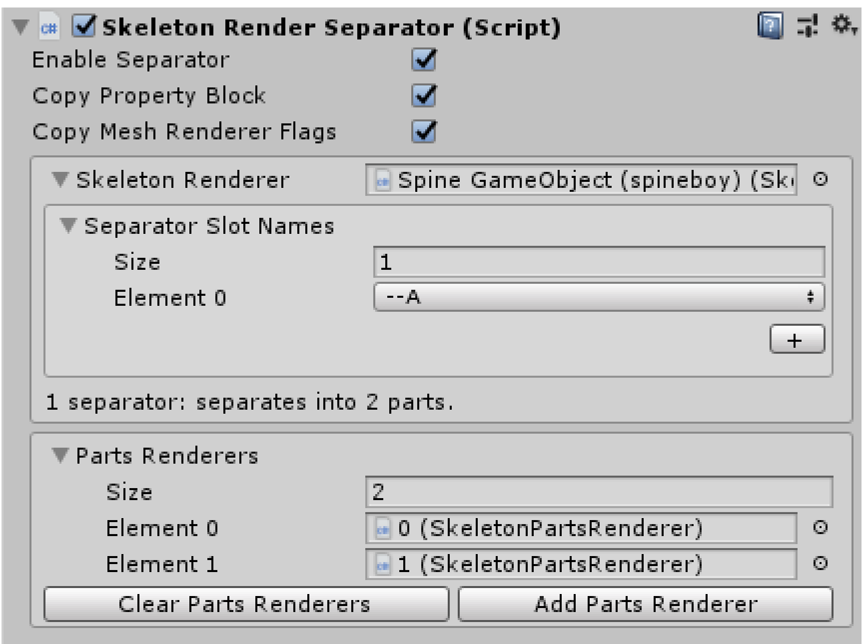
你可以右击 SkeletonGraphic 并选择 `Add Basic Serialized Custom Materials` 来将这个组件添加到 GameObject 中。在 `Custom Texture Overrides` 数组中添加条目可以给整个 skeleton 替换一种 texture。在 `Custom Material Overrides` 数组中添加条目，可以用另一种 material 替换原始（替换前的）texture 中使用的 material。请确保启用 `Override Enabled` 以启用各条目中的 texture 或 material 覆盖.

注意: 该组件设计初衷里不包括通过代码进行交互。要通过代码为你的 SkeletonGraphic 动态设置 material，应直接通过 `SkeletonGraphic.CustomMaterialOverride` 来访问 material 覆盖数组，或通过 `SkeletonGraphic.CustomTextureOverride` 来访问 texture 覆盖.

SkeletonRenderSeparator

你可能想在角色的各个部分之间显示其他的 GameObjects，例如：让你的角色爬到一棵树上，在树干前面显示一条腿在树干后面显示另一条。`SkeletonRenderSeparator` 组件让你可以将 SkeletonRenderer （或其子类 SkeletonAnimation 和 SkeletonMecanim）分为两个或更多的 `SkeletonPartsRenderers`，并可自定义其图层顺序.

注意: SkeletonGraphic 组件直接在 `SkeletonGraphic` 检查器的 `Advanced` 部分提供了渲染分离功能，它不无需任何额外组件.

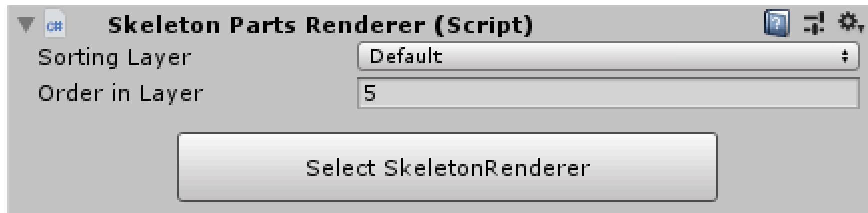


注意: 通常情况下，Spine 渲染器组件会使用一个渲染器来显示整个 skeleton 的网格。然而这会导致无法在其部件(parts)之间插入其他 `UnityEngine.Renderers`（`SpriteRenderer`, `MeshRenderer`, `ParticleSystem` 等等）组件.

设置步骤

1. **确保 Skeleton 的绘制顺序.** 找出你想用哪个槽位来将 Skeleton 渲染分为多个部件。为方便起见，在导出 skeleton 之前应清晰地标注该槽位.
2. **添加 SkeletonRenderSeparator 组件** 选中 Spine GameObject。在检查器中右键点击你的 SkeletonAnimation 或 SkeletonRenderer 。选择 `Add Skeleton Render Separator`。这将把 `SkeletonRenderSeparator` 组件添加到 GameObject 中.
3. **分配 Separator 槽位** 检查器现在会显示一个警告，因为 separator 列表是空的。在 `Separator Slot Names` 下选择所需的槽位。你可以点击" +"按钮来添加额外的 separator 槽位.
   注意: 这个字段在 SkeletonRenderer （或 SkeletonAnimation 和 SkeletonMecanim 子类）组件处被序列化，`SkeletonRenderSeparator` 只是为它提供了接口.
4. **添加部件(Parts)渲染器** 检查器现在会显示一个警告：你没有足够的部件渲染器.
   点击 `Add the missing renderers (n)`按钮，用 `SkeletonPartsRenderer` 组件创建所需的 GameObjects。这些 GameObjects 将被自动分配到上面的 `Parts Renderers` 列表中.
   注意: `SkeletonRenderSeparator` 会根据当前的绘制顺序来检测当前所需的部件渲染器的数量。如果在运行时修改了绘制顺序，可能会发现渲染器需要更多的部件渲染器。在这种情况下，可以通过点击 `Add Parts Renderer` 按钮，手动添加一到两个额外的部件渲染器.

5. **设置 `Sorting Layer` 和 `Order in Layer`** 每个 `SkeletonPartsRenderers` 都在检查器中提供了一个 `Sorting Layer` 和 `Order in Layer` 属性。你可以在每个 `SkeletonPartsRenderer` 中设置顺序属性值，值更大的部件渲染器越靠前.



注意: `SkeletonPartsRenderer` GameObjects 不必为 Spine GameObject 的子对象。`SkeletonRenderSeparator` 会保持引用，所以你可以根据需要组织层级结构.

示例场景

你可以在 `Spine Examples/Other Examples/SkeletonRenderSeparator` 中找到关于 `SkeletonPartsRenderer` 和 `SkeletonRenderSeparator` 用法的示例场景 .

C#

启用和禁用

默认情况下，`SkeletonRenderSeparator` 将禁用 SkeletonRenderer 并接管其网格渲染任务。同样，如果你禁用 `SkeletonRenderSeparator`，SkeletonRenderer 将重新接管渲染工作.

你可以像其他组件一样启用和禁用 `SkeletonRenderSeparator`：

C#

```
1. skeletonRenderSeparator.enabled = true; // separation is enabled.
2. skeletonRenderSeparator.enabled = false; // separation is disabled.
```

改变分离点

分离点并不存储在 `SkeletonRenderSeparator` 中。

它是由 SkeletonRenderer （或其子类 SkeletonAnimation 和 SkeletonMecanim）中的 separator 槽位定义的。如果你想在运行时操纵 separator 槽位，你可以访问 `SkeletonRenderer.separatorSlots` 列表，并像一般列表一样通过调用 `Add`、`Remove` 或 `Clear` 来操纵它.

C#

```
1. Spine.Slot mySlot = skeletonAnimation.Skeleton.FindSlot("MY SPECIAL SLOT");

2. skeletonAnimation.separatorSlots.Clear();
3. skeletonAnimation.separatorSlots.Add(mySlot);
```

在运行时添加 SkeletonRenderSeparator

你可以使用静态方法 `SkeletonRenderSeparator.AddToSkeletonRenderer` 来添加和初始化一个新的 `SkeletonRenderSeparator` 组件.

C#

```
1. SkeletonAnimation skeletonAnimation = GetComponent<SkeletonAnimation>();
2. skeletonAnimation.SeparatorSlots.Add(mySlot); // see above
3.
4. // Add the SkeletonRenderSeparator.
5. SkeletonRenderSeparator skeletonRenderSeparator = SkeletonRenderSeparator.AddToSkeletonRenderer(skeletonAnimation);
```

默认情况下，它将添加当前必要的 `SkeletonPartsRenderers`。它也为高级用例提供了一些可选的参数，详情请查看编码文档.

渲染

着色器

spine-unity 运行时包含多种不同着色器。默认情况下，一个新导入的 skeleton material 用的是 `Spine/Skeleton` 着色器。你也可以通过 Material 的着色器参数来更改着色器。下文为 Spine 着色器列表..

注意: Spine 着色器还不支持*延迟着色渲染路径*.

1. **Spine/Skeleton** *(默认着色器)*
   Unlit 透明着色器. 不写 z-buffer.
2. **Spine/Skeleton Graphic** *(SkeletonGraphic 的默认着色器)*
   SkeletonGraphic 用的 Unlit 透明着色器。不写 Z-buffer。当与 `CanvasGroup` 一起使用时，不支持 `Additive` blend 模式. 如果有这类需求应使用

`Spine/Skeleton Graphic Tint Black`代替。由于 CanvasRenderer 的限制，只能使用单 texture.

3. **Spine/Skeleton Lit**

   Simple lit 透明着色器，不支持法线贴图. 不写 z-buffer.

4. **Spine/Skeleton Lit ZWrite**

   Simple lit 透明着色器，不支持法线贴图. 写 z-buffer.

5. **Spine/Skeleton Fill**

   可自定义颜色叠加(color overlay)的 Unlit 透明着色器. 不写 z-buffer.

   `FillColor` 决定叠加的颜色, `FillPhase` 决定叠加强度.

6. **Spine/Skeleton Tint**

   可定制双色 tint 的 Unlit 透明着色器，将深色与浅色的 tinting 独立，称为 tint black。不写 z-buffer.

   texture 的浅色被染成 `Tint Color`，texture 的深色被染成 `Black Point`。与普通的多色混合(multiply color blending)相比，tinted 的 texture 比其原颜色更亮。当把 `Tint Color` 和 B `Black Point` 设置为同一颜色时，得到的是一个纯色叠加。当把 `Tint Color` 设置为黑色 `Black Point` 设置为白色时，texture 则会反色.

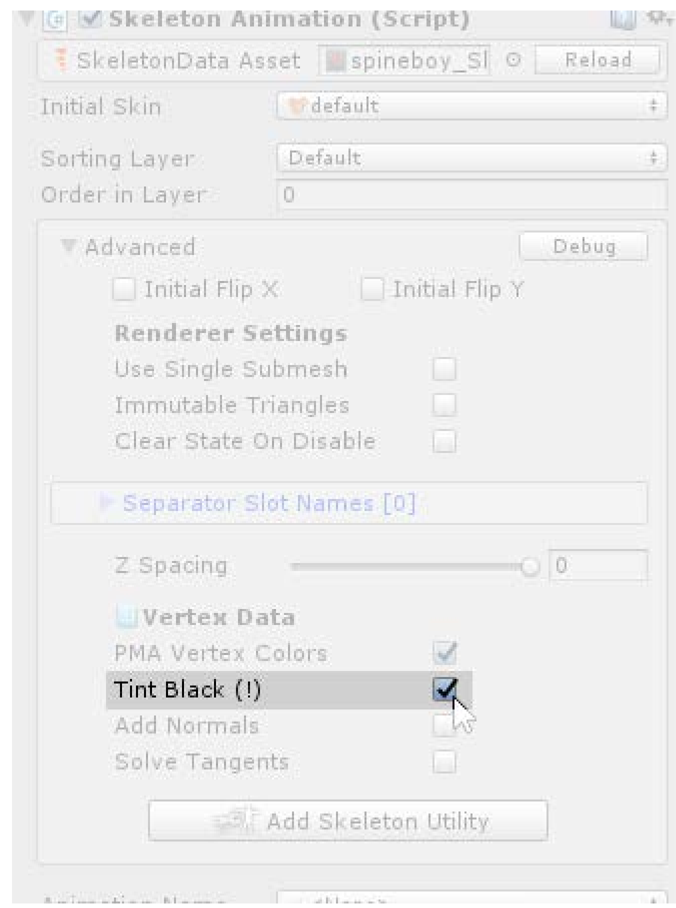
7. **Spine/Skeleton Tint Black**

   可为动画单槽位 tint black的 unlit 透明着色器。不写 z-buffer.

   Spine 为槽位提供了 Tint Black 功能，可以动画 black tint.

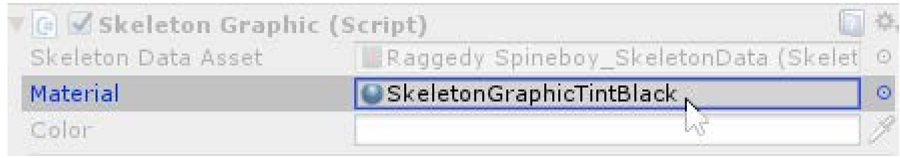
   需要额外的设置步骤（用于 tint color 顶点数据）:

- 在 SkeletonAnimation 的检查器中的 `Advanced` 部分启用 `Tint Black`:

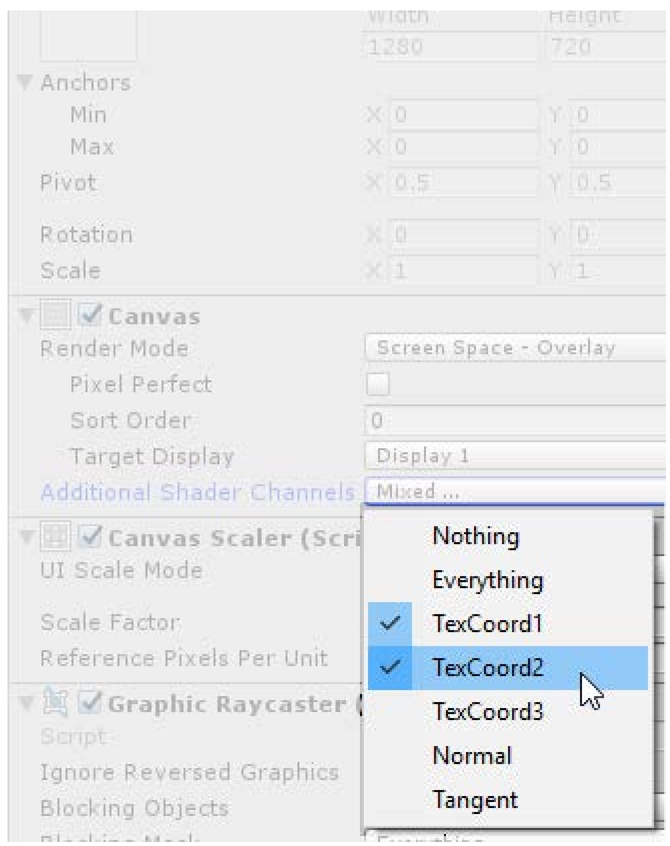


8. **Spine/Skeleton Tint Black Additive**
   可为动画单槽位 tint black的 unlit 透明着色器。使用 additive blend 模式。不写 z-buffer.
9. **Spine/SkeletonGraphic Tint Black**
   `Spine/Skeleton Tint Black` 着色器的 SkeletonGraphic 版本。当与 `CanvasGroup` 一起使用时支持 `Additive` blend 模式.
   需要额外的设置步骤（用于 tint color 顶点数据）:

- 在 SkeletonAnimation 的检查器中的 `Advanced` 部分启用 `Tint Black`.
- 将 SkeletonGraphic 的 material 设置为放在 `Spine/Runtime/spine-unity/Materials` 文件夹中的 `SkeletonGraphicTintBlack` material..



- 选择父 Canvas 并在 `Additional Shader Channels` 中启用 `TexCoord1` 和 `TexCoord2`



在 `CanvasGroup` 中设置 `Additive` blend 模式的步骤:

- 在 [SkeletonGraphic] 检查器的 `Advanced` 部分中启用 `Canvas Group Tint Black`.
- 在着色器中启用 `CanvasGroup Compatible`.

10. **Spine/Sprite**

    复杂的可配置着色器，支持比 `Spine/Skeleton Lit` 着色器更高级的照明。可以在实例场景 `Spine Examples/Other Examples/Sprite Shaders` 中找到关于 `Spine/Sprite/Vertex Lit` 着色器的演示.

    - **Spine/Sprite/Unlit**
      unlit 着色器，包含可配置的 blend 模式、覆盖颜色、色调(hue)、饱和度(saturation)和亮度(brightness)调整。可配置为写入 z-buffer。支持 fog.
    - **Spine/Sprite/Vertex Lit**
      可配置 blend 模式的复杂 vertex-lit 着色器.
      支持法线贴图、次级漫反射(secondary albedo)、金属性(metallic)和自发光(emission)贴图.
      用于 cel-shaded look 的可配置 color ramp 和基于法线的边缘照明(rim lighting).
      可配置颜色叠加、色调、饱和度和亮度调整.
      可配置为写入 z-buffer。支持 fog.

- **Spine/Sprite/Pixel Lit**
  `Spine/Sprite/Vertex Lit` 着色器的 Pixel-lit 版。这个着色器总是写入 z-buffer（启用了 `ZWrite`，因为它使用了 `ForwardAdd` pass）.

11. **Spine/Special**
    - **Spine/Special/Skeleton Grayscale**
      可定制灰度渲染强度的 Unlit 透明着色器。不写 z-buffer.
    - **Spine/Special/Skeleton Ghost**
      SkeletonGhost 组件使用的特殊着色器，用于跟踪(trail)渲染.
12. **Spine/Blend Modes**
    适用于在 Spine 编辑器中的槽位中配置了 `Additive`、`Multiply` 和 `Screen` blend 模式的情况。建议使用 BlendModeMaterials SkeletonData Modifier 资产，在导入时自动设置 blend 模式 material.
    - **Spine/Blend Modes/Skeleton PMA Additive**
      Unlit 透明着色器。使用 additive blend 模式。不写 z-buffer.
    - **Spine/Blend Modes/Skeleton PMA Multiply**
      Unlit 透明着色器。使用 multiply blend 模式。不写 z-buffer.
    - **Spine/Blend Modes/Skeleton PMA Screen**
      Unlit 透明着色器。使用 screen blend 模式。不写 z-buffer.
13. **Spine/Outline**
    上述所有着色器都提供了一个 `Outline` 参数，可用它来切换不同版本的 `Spine/Outline` 着色器，它用于 i 在 skeleton 周围绘制一个额外的彩色轮廓。你可以在示例场景 `Spine Examples/Other Examples/Outline Shaders` 中找到 `Spine/Outline` 着色器的演示.
    - **Spine/Outline/OutlineOnly-ZWrite** 一个只渲染轮廓的特殊 two-pass 着色器。写入 z-buffer，以便在重叠(overlapping)附件上实现合适的轮廓遮蔽(outline occlusion).

URP 着色器- UPM 插件包

通用渲染管线（URP）着色器包含于一个单独的 UPM（Unity Package Manager）包。关于如何下载和安装 UPM 包，请参见可选 UPM 插件包一节；关于如何更新扩展 UPM 包，请参见更新 UPM 插件包。

URP Shaders UPM 包提供了专门为 Unity 通用渲染管线构建的着色器，包括 2D 渲染器功能.

注意: Spine URP 着色器尚未支持新添加的 URP *延迟渲染路径*.

1. **Universal Render Pipeline/2D/Spine/Skeleton Lit**
   `Spine/Skeleton Lit` shader 着色器的通用 2D 渲染器版.
2. **Universal Render Pipeline/2D/Spine/Sprite**
   `Spine/Sprite/Vertex Lit` 和 `Pixel Lit` 着色器的通用 2D 渲染器版.
3. **Universal Render Pipeline/Spine/Skeleton**
   `Spine/Skeleton` 着色器的 URP 版.

4. **Universal Render Pipeline/Spine/Skeleton Lit**
   `Spine/Skeleton Lit` 着色器的 URP 版.
5. **Universal Render Pipeline/Spine/Sprite**
   `Spine/Sprite/Vertex Lit` 和 `Pixel Lit` 着色器的 URP 版.
6. **Universal Render Pipeline/Spine/Outline/Skeleton-OutlineOnly**
   `Spine/Outline` 着色器的 URP 版。URP 不允许每个着色器有多个 pass，所以它需要单独 material。你可以考虑 `RenderExistingMesh` 组件，详见示例场景 `Outline Shaders URP`.

这些着色器可以附加于 material，并且会受到 `Project Settings - Graphics` 中设置的 `UniversalRenderPipelineAsset` 的影响.
你可以示例场景 `com.esotericsoftware.spine.URP-shaders-3.8/Examples/URP Shaders.unity` 中找到 URP 着色器的演示.

LWRP Shaders - Extension UPM Package

轻量渲染管线（LWRP）着色器包含于一个单独的 UPM（Unity Package Manager）包。关于如何下载和安装 UPM 包，请参见可选 UPM 插件包一节；关于如何更新扩展 UPM 包，请参见更新 UPM 插件包..
LWRP Shaders UPM 包提供了 Unity 的轻量渲染管线专用的着色器:

1. **Lightweight Render Pipeline/Spine/Skeleton**
   `Spine/Skeleton` 着色器的 LWRP 版.
2. **Lightweight Render Pipeline/Spine/Skeleton Lit**
   `Spine/Skeleton Lit` 着色器的 LWRP 版.
3. **Lightweight Render Pipeline/Spine/Sprite**
   `Spine/Sprite/Vertex Lit` and `Pixel Lit` 着色器的 LWRP 版.

这些着色器可以附加于 material，并且会受到 `Project Settings - Graphics` 中设置的 `LightweightRenderPipelineAsset` 的影响.
你可以示例场景 `com.esotericsoftware.spine.lwrp-shaders-3.8/Examples/LWRP Shaders.unity` 中找到 LWRP 着色器的演示.

Shader Graph

目前还没有官方的 Shader Graph Spine 着色器或着色器节点可用。请注意，当你从 Spine 导出 texture 且使用 Straight alpha 时，你可以使用任何非 Spine 着色器。如果你想在其中复刻 Spine 着色器专用功能，请参见论坛上的帖子[1], [2]。如果你有任何问题，欢迎随时来 Unity 分论坛上发帖.

Amplify Shader Editor

虽然官方没有为 Amplify 着色器编辑器提供着色器模板，但用户 `Hana` 已经在论坛上热情地分享了他的模板代码.

编写自己的着色器

请先熟悉一下为 Unity 编写自定义着色器的一般方法。特别是教程: 顶点和片元(fragment)编程提供了一个很好的概述，让你更容易理解 spine-unity 着色器的各个部分在做什么.

从现有的 spine-unity 着色器开始编写自己的着色器

强烈建议从现有的 spine-unity 着色器副本着手。然后，你可以逐步修改一个已可用的着色器，调整它来达到你想要的效果。例如，你可以在着色器返回最终颜色结果之前添加自

己的颜色处理代码。下面的代码展示了一个简短的示例，说明了如何创建一个添加了灰度(grayscale)功能的修改版 `SkeletonGraphic` 着色器:

```
1. Properties
2. {
3.     _GrayIntensity("Intensity", Range(0, 1)) = 1 // this line was added to provide a Material property
4.     [..]
5. }
```

```
1. sampler2D _MainTex;
2. float _GrayIntensity; // this parameter was added
3. ..
4. fixed4 frag (VertexOutput IN) : SV_Target
5. {
6.     ..
7.     color.rgb = lerp(color.rgb, dot(color.rgb, float3(0.3, 0.59, 0.11)), _GrayIntensity); // this line was added
8.     return color;
9. }
```

使用非 Spine 着色器或可视化着色器编辑器时的附加说明

请务必考虑以下 spine-unity 着色器 shaders 和其他非 Spine 着色器间的区别::

1. 在渲染任何 Spine skeleton 时，必须设置 `Cull Off` 来**禁用 Backface culling**
2. Spine 着色器通常不需要法线，因此在使用 lit 着色器时，可能需要在组件处启用 `Advanced - Add Normals`.
3. Spine 着色器通常不需要 tangents，因此在使用法线贴图时，可能需要在组件中启用 `Advanced - Solve Tangents`.
4. Spine 着色器默认使用 Premultiply Alpha textures，所以要么
   a) 将 atlas textures 导出为 Straight alpha，或者:
   b) 将着色器的 blend 模式改为 PMA blend 模式 `Blend One OneMinusSrcAlpha`.
5. Spine 顶点颜色通常是 PMA 顶点颜色。当使用透明槽位或 `Additive` 槽位时，你可以选择:
   a) 将着色器的 blend 模式改为 PMA blend 模式 `Blend One OneMinusSrcAlpha`，并使用 PMA atlas textures，或者
   b) 在组件中禁用 `Advanced - PMA Vertex Colors`（此时不渲染 `Additive` 槽位）.

UI 和非 UI 着色器的一般准则是:

1. 不要在 `SkeletonAnimation` 或 `SkeletonMecanim` 上使用 UI 着色器.
2. 不要在 `SkeletonGraphic` 上使用非 UI 着色器.

对 `Spine/Skeleton` 着色器的解析

下文提供了对 [`Spine/Skeleton` 着色器]的解析。在导入 Spine texture atlas 后生成 Material 时，默认会使用该着色器。`Spine/Skeleton` 着色器是相当精简和典型的，它具有以下特点:

- Premultiply Alpha (PMA) blending
- No depth buffer writing
- No lighting
- No backface culling
- No fog
- 通过乘以顶点颜色来对 texture 进行 tint
- 可选择使用 straight alpha 而非 PMA textures
- 有一个 " `ShadowCaster` " pass，因此它可以投射实时阴影
- Material 属性:
    - _MainTex "Main Texture"
    - _StraightAlphaInput "Straight Alpha Texture"
    - _Cutoff "Shadow alpha cutoff"
    - 高级参数:
        - _StencilRef "Stencil Reference"
        - _StencilComp "Stencil Comparison"
        - Outline parameters (_OutlineWidth "Outline Width", etc.)

详细解释:

- **Premultiply Alpha (PMA) blending**

  hlsl

  ```
  Blend One OneMinusSrcAlpha
  ```

  (Spine-Skeleton.shader:25)

  Blending 的本质是 `result_rgba = frag_output_rgba * src_factor + framebuffer_rgba * dst_factor`.

  非标准的 `Blend One OneMinusSrcAlpha` PMA blend 模式可以让 `Additive` blend 模式的槽位和 `Normal` blend 模式槽位在单次渲染 pass 中同时绘制。这是通过在上面这行代码的 `SrcFactor` 中使用值 `One`（而非值 `SrcAlpha`）实现的。这将无修改的 `frag_output_rgba` 值加到被 `OneMinusSrcAlpha` 加权的 `framebuffer_rgba` 上:

  a) 对于 `Normal` blending，片元着色器将 `RGB` 乘以 `A`，且 `A` 保持不变.

b) 对于 `Additive` blending，`RGB` 不乘 alpha，且 `A` 被置为 0，接收 `result_rgba = frag_output_rgba + (1-0) * framebuffer_rgba`.

当在 `SkeletonRenderer` 或 `SkeletonGraphic` 组件上启用 Advanced - `PMA Vertex Colors` 时，`Normal` 和 `Additive` 槽位混合模式将作为顶点颜色被隐式地传入着色器:

hlsl
```
struct VertexInput {
    float4 vertexColor : COLOR
}
```

(Spine-Skeleton.shader:47)

当 PMA 顶点颜色乘以采样出的 PMA texture 颜色时，会自动应用槽位的 `Normal` 或 `Additive` blend 模式:

hlsl
```
return (texColor * i.vertexColor);
```

(Spine-Skeleton.shader:71)

所以得在你自己的着色器中适当地支持 `Normal` 和 `Additive` PMA blend 模式:

1. 将 blend 函数定义为 `Blend One OneMinusSrcAlpha`
2. 将 texture 颜色与顶点颜色相乘
3. 在组件中启用 Advanced - `PMA Vertex Colors`.

如果你想使用一个使用标准 blend 模式 `Blend SrcAlpha OneMinusSrcAlpha` 的着色器，且不需要 `Additive` 槽位，则需要确保从 Spine 导出的 atlas textures 为 Straight alpha.

- **No depth buffer writing**

  hlsl
  ```
  ZWrite Off
  ```

  (Spine-Skeleton.shader:24)

  典型的 alpha-blended 2D sprite 着色器不写 depth buffer。透明物体将根据 `Camera.transparencySortMode` 以从后向前绘制，而不依赖 depth buffer 进行深度排序。`Spine/Skeleton` 与 Unity 自己的 `Sprites/Default` 着色器均有这一特性.

  如果你想使用带有深度写入的着色器，请确保在你的 `SkeletonRenderer` 或 `SkeletonGraphic` 组件中将 `Advanced - Z-Spacing` 置为 0 以外的值，以防止 Z-Fighting 现象，特别是当你用了光照时。请注意，使用 depth buffer 可能会在半透明区域周围造成意外结果，比如边缘锯齿(aliasing effects on edges).

- **No lighting**

  `Spine/Skeleton` 着色器不受放置在场景中的任何灯光的影响，且将始终以 100%的 `texColor * i.vertexColor` 强度(intensity)进行渲染.

  为了在着色器中使用光照，建议从一个可用的 lit 着色器开始修改，来得到你所需的副本。简单地将 `Lighting Off` 行改为 `Lighting On` 并不能达到预期的效果，你需要在你的顶点着色器（对于每顶点照明(per-vertex lighting)）或片元着

色器函数（对于每像素照明(per-pixel lighting)）中计算光照，来分别改变颜色强度。还要注意的是，URP, URP-2D 和标准管线着色器都使用了不同的光找计算步骤，所以参考着色器代码时一定要选择对应的着色器.

- **No backface culling**

  hlsl

  ```
  Cull Off
  ```

  [(Spine-Skeleton.shader:23)]

  渲染 Spine skeleton 的唯一要求是**禁用 backface culling**，特别是对于 2D 着色器来说.

  因为负向缩放(scaled negatively)Spine 网格的部件，或是翻转 skeleton 方向时，会导致某些三角形变得不可见，所以大多数 3D 着色器都启用了 backface culling.

- **No fog**

  `Spine/Skeleton` 着色器不受 fog 的影响.

  在着色器上启用 fog 需要在着色器代码中添加一些额外的顶点参数和函数调用。例如 UnityCG.cginc 的代码:

  hlsl

  ```
  multi_compile_fog Will compile fog variants.
  UNITY_FOG_COORDS(texcoordindex) Declares the fog data interpolator.
  UNITY_TRANSFER_FOG(outputStruct,clipspacePos) Outputs fog data from the vertex
  shader.
  UNITY_APPLY_FOG(fogData,col) Applies fog to color "col". Automatically applies
  black fog when in forward-additive pass.
  Can also use UNITY_APPLY_FOG_COLOR to supply your own fog color.
  ```

  你可以查阅 `Spine/Sprite/Unlit` 着色器来了解如何在你的着色器中应用 fog:

  hlsl

  ```
  #pragma multi_compile_fog
  ```

  [(SpritesUnlit.shader:70)]

  hlsl

  ```
  UNITY_FOG_COORDS(1) // to declare it at the free attribute TEXCOORD1
  ```

  [(SpriteUnlit.cginc:L24)]

  hlsl

  ```
  UNITY_TRANSFER_FOG(output,output.pos);
  ```

  [(SpriteUnlit.cginc:46)]

- **Uses vertex colors to tint the texture via multiply**

  请参见前文的 *Premultiply Alpha (PMA) blending.*

- **Optionally uses straight-alpha instead of PMA textures**

  由于 `Spine/Skeleton` 着色器的 blend 模式一直被设置为 PMA blending，所以没有 premultiplied alpha 颜色的输入 texture 需要在采样后被转换为 PMA 颜色。下面几行代码实现了该功能:

```
// bool Material parameter, enables the _STRAIGHT_ALPHA_INPUT shader keyword when enabled
[Toggle(_STRAIGHT_ALPHA_INPUT)] _StraightAlphaInput("Straight Alpha Texture", Int) = 0
..
// compiles the shader in two variants so that shader keywords can switch between both variants
#pragma shader_feature _ _STRAIGHT_ALPHA_INPUT
..
// when enabled, multiply texture rgb values by the texture alpha value.
#if defined(_STRAIGHT_ALPHA_INPUT)
texColor.rgb *= texColor.a;
#endif
```

- **有`"ShadowCaster"` pass 因此可以投射实时阴影**
  带有 `Tags { "LightMode"="ShadowCaster" }`的第二 pass 将被 `LightMode` 自动识别为阴影投射(shadow caster) pass。`ShadowCaster` pass 不写入任何 RGB 颜色，而是将深度信息写入阴影缓冲区(shadow buffer)。因此，它必须设置为 `ZWrite On`。由于不能写入半透明的深度，可以把片元写入深度缓冲区，也可以丢弃片元(这时不投射阴影)。这是通过调用阈值(thresholding)函数完成的:
  hlsl
  ```
  clip(texcol.a * i.uvAndAlpha.a - _Cutoff);
  ```
  (Spine-Skeleton.shader:111)
  这里的`_Cutoff` Material 参数定义了 alpha 的阈值，若 x<0 则 `clip(x)`丢弃片元.
- **Material 属性:**
  1. _MainTex "Main Texture"
     主纹理.
  2. _StraightAlphaInput "Straight Alpha Texture"
     参见上文"*使用 straight-alpha 而非 PMA textures* ".
  3. _Cutoff "Shadow alpha cutoff"
     参见上文"*有`"ShadowCaster"` pass 因此可以投射实时阴影*".
  4. 高级参数:
     - _StencilRef "Stencil Reference"
       用于与遮罩交互.
     - _StencilComp "Stencil Comparison"
       用于与遮罩交互，将根据 `Mask Interaction` 属性由 `SkeletonRenderer` 或 `SkeletonGraphic` 组件来设置.

- Outline parameters (_OutlineWidth "Outline Width", etc.) 在切换到 `Spine/Outline/Skeleton` 类的 outline 着色器时使用，在普通的 `Spine/Skeleton` 这类非 outline 着色器中不使用.

Materials

MeshRenderer 的 material 数组是由 SkeletonRenderer 逐帧管理，依赖于当前所用的附件和包含这些附件的 AtlasAssets.

Note: 直接修改 material 数组是无效的，因为它们会再接下来的 `LateUpdate()` 调用中被覆盖掉，所以应使用 SkeletonRendererCustomMaterials 或 SkeletonGraphicCustomMaterials 组件来覆盖材料。也可以在_*Atlas* 资产处配置不同的 material 来改变所有实例的 material。修改_*Atlas* 资产之后，都需要在 `SkeletonRenderer` 组件的 `SkeletonData Asset` 参数处点击 `Reload` 来重新加载使用了新的 atlas materials 的 skeleton.

Material 切换(Switching)和绘制调用

如果附件分布在多个 atlas 页上，比如 material A 和 material B，运行时会根据需要 material 的绘制顺序来设置 material 数组.

若需求顺序为:

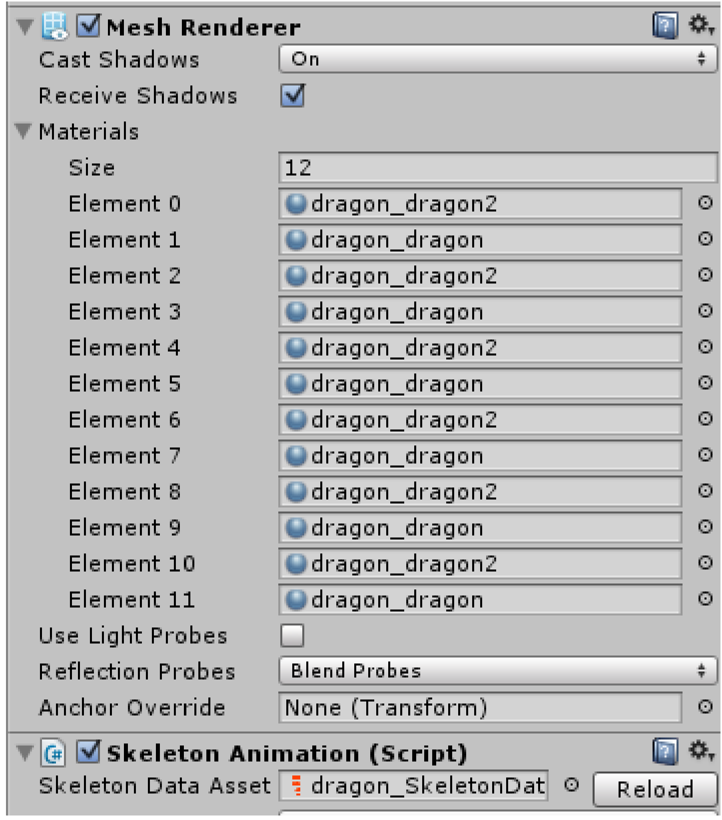
1. A 的附件
2. A 的附件
3. B 的附件
4. A 的附件

则生成的 material 数组为:

1. Material A (满足需求 1 和 2)
2. Material B (满足需求 3)
3. Material A (满足需求 4)

Material 数组中的每一个 material 都对应着一次绘制调用。因此大量的 material 切换会削弱性能表现.

Dragon 示例展示了这种有很多绘制调用的用例:



因此，建议将附件打包到尽可能少的 atlas 页中，并根据绘制顺序将附件分组置入 atlas 页以防止多余的 material 切换。请参阅 Spine Texture Packer: Folder Structure 了解如何在你的 Spine atlas 中编排 atlas 区域.

Changing Materials Per Instance

Note: 直接修改 SkeletonRenderer 中的 material 数组是无效的，因为它们会再接下来的 `LateUpdate()`调用中被覆盖掉。如果下面的方法都不合适你，可以使用 `SkeletonAnimation.OnMeshAndMaterialsUpdated` 手动地逐帧覆盖 `MeshRenderer.Materials`。这个回调会当 atlas materials 分配完毕后在 `LateUpdate()`的结尾处调用.

CustomMaterialOverride 和 CustomSlotMaterial

SkeletonRenderer 让你可以覆盖(override)特定槽位上的 materials，或者覆盖生成的(resulting) materials.

要在运行时替换 SkeletonRenderer 的一个实例的 material，可以使用 `SkeletonRenderer.CustomMaterialOverride`:

C#

```
1. skeletonAnimation.CustomMaterialOverride[originalMaterial] = newMaterial; // to enable the replacement.
2. skeletonAnimation.CustomMaterialOverride.Remove(originalMaterial); // to disable that replacement.
```

要替换某个槽位上的 material，可以使用 `SkeletonRenderer.CustomSlotMaterial`:

C#

```
1. skeletonAnimation.CustomSlotMaterial[slot] = newMaterial; // to enable the replacement.
2. skeletonAnimation.CustomSlotMaterial.Remove(slot); // to disable that replacement.
```

Tinting Skeletons while retaining batching

在 skeleton 实例上使用不同的 `Materials` 或 `MaterialProeprtyBlocks` 会破坏 batching。如果你只需要 tint 某个 skeleton 实例且不修改其他 material 属性，可以使用 `Skeleton.R .G .B .A` 颜色属性。为了使 tinting 生效，必须在 SkeletonRenderer 检查器中启用 `Advanced - PMA Vertex Colors` .

C#

```
1. public Color color = Color.white;
2. ...
3. skeleton = GetComponent<SkeletonRenderer>().Skeleton;
4. ...
5. skeleton.R = color.r;
6. skeleton.G = color.g;
7. skeleton.B = color.b;
8. skeleton.A = color.a;
```

这些 skeleton 颜色值将置为顶点颜色且不改变 material 属性.

这也同样适用于 tinting 某个附件:

C#

```
1. slot = skeleton.FindSlot(slotname);
2. ...
3. slot.R = slotColor.r;
4. slot.G = slotColor.g;
5. slot.B = slotColor.b;
6. slot.A = slotColor.a;
```

Note: 当你的动画修改附件的颜色值时，一定要使用诸如 SkeletonAnimation.UpdateComplete 的回调来在动画应用后设置全部槽位的颜色值.

MaterialPropertyBlocks

These two Spineboys share the same Material, but use MaterialPropertyBlock to have different tints.

可以使用 [Renderer.SetPropertyBlock] 来覆盖单个 `MeshRenderer` 的 material 属性值.

C#

```
1. MaterialPropertyBlock mpb = new MaterialPropertyBlock();
2. mpb.SetColor("_FillColor", Color.red); // "_FillColor" is a named property o
   n the used shader.
3. mpb.SetFloat("_FillPhase", 1.0f); // "_FillPhase" is another named property
   on the used shader.
4. GetComponent<MeshRenderer>().SetPropertyBlock(mpb);
5.
6. // to deactivate the override again:
7. MaterialPropertyBlock mpb = this.cachedMaterialPropertyBlock; // assuming yo
   u had cached the MaterialPropertyBlock
8. mpb.Clear();
9. GetComponent<Renderer>().SetPropertyBlock(mpb);
```

Note: 在 `MaterialPropertyBlock` 中使用的参数名，如`_FillColor` 或`_FillPhase`，须与对应的着色器变量名匹配。请注意，着色器变量名不是显示在检查器中的名称，例如 `Fill Color` 和 `Fill Phase`。要查看着色器的参数名，你可以打开`.shader` 文件（通过点击 material 的齿轮图标，选择 `Edit Shader`），看看最上面的 `Properties { .. }` ,那里有全部参数的列表。在如下这行参数中，**最左边的名字**就是参数名`_FillColor`:

```
_FillColor ("Fill Color", Color) = (1,1,1,1)
^^^^^^^^^^
```

着色器变量名通常以_字符开头，且从不包含任何空格。它旁边的字符串`"Fill Color"`就是在检查器中显示的名称.

你可以在示例场景 `Spine Examples/Other Examples/Per Instance Material Properties` 中找到对每个实例 material 属性的演示.

优化提示:

- 使用具有不同 Material 值的 Renderer.SetPropertyBlock 会破坏渲染器之间的 batching。当 MaterialPropertyBlock 的参数相等时(例如所有的 tint 颜色都设置为绿色)那么渲染器间将进行 batching.
- 每当你改变或增加 MaterialPropertyBlock 的属性值时，你都需要调用 `SetPropertyBlock` 。但是你可以把 MaterialPropertyBlock 作为你的类的成员，这样你就不必在改变某个属性值时不得不实例化出一个新的.
- 当你需要经常设置某个属性时，你可以使用静态方法: `Shader.PropertyToID(string)`来缓存该属性的 int ID，而不是用 MaterialPropertyBlock 的 setter 的字符串重载.

透明度(Transparency)与绘制顺序

所有的 spine-unity 着色器都使用 alpha blending 来干净地绘制附件边缘的半透明过渡。如果没有 alpha blending（使用了一个硬的透明度阈值），会产生尖锐的狗牙状的轮廓，类似于锯齿(aliasing artifacts)效果.

然而 alpha blending 有个老问题，就是无法自动地对 z-buffer 进行深度排序。所以三角形需要按照从后到前进行渲染，将各部件堆叠绘制。每个 SkeletonRenderer 都会生成其对应网格，按照 Spine 中定义的槽位绘制顺序三角形。在单一网格内，一个绘制调用即可将部件有序的 skeleton 正确绘制.
在网格之间，spine-unity 利用 Unity 的许多渲染顺序系统来决定哪个网格应该在哪个上面。按照标准的 spine-unity 设置，整个 skeleton 网格的渲染顺序由以下多个因素决定:

1. 相机深度(depth) *与多机位设置有关.*
2. **Material.renderQueue.** *当它设置后将覆盖着色器的* `Queue` *标签.*
3. **着色器的 `Queue` 标签.** *与其他 sprites 一样，在* `Spine` *着色器中默认为* `"Transparent"` *队列.*
4. **Sorting Group 组件.** *当该组件置于 MeshRenderer 的 GameObject 或任何一个父 GameObject 上时.*
5. **渲染器 SortingLayer 的 SortingOrder within a layer.**
6. **相机距离(distance)** *可以把相机配置为*使用平面(planar)距离(正交相机)还是透视距离(透视相机).

如果一个场景的渲染器处于相同的 sorting layer，顺序相同且着色器 `Queue` 标签也相同，那么你可以通过与摄像机的距离来控制 Spine GameObjects 的排序。请注意这里的摄像机提供了 transparencySortMode 属性.

Sorting Layer 和 Order in Layer

SkeletonRenderer （或子类 SkeletonAnimation 和 SkeletonMecanim）的检查器提供了 `Sorting Layer` 和 `Order in Layer` 属性，它们本质上是对 `MeshRenderer` 的 sortingLayerID 和 sortingOrder 属性的修改。这些属性是存储在 `MeshRenderer` 内的，而非 `SkeletonRenderer` 内.
你可以通过代码访问这些属性:

C#

```
GetComponent<MeshRenderer>().sortingOrder = 1; // Change the Order in Layer to 1.
```

防止排序错误

在使用正交相机时，特别容易出现多页 atlas 的 skeleton 排序错误的情况。这个问题可以通过在 skeleton 的 GameObject 中添加一个 Sorting Group 组件来解决。另一个解决方法是将摄像机请为旋转一下，比如，将摄像机 transform 的旋转 Y 值置为 0.001.

在 Skeleton 的部件间渲染对象

你可能想在角色的各个部分之间显示其他的 GameObjects，例如：让你的角色爬到一棵树上，在树干前面显示一条腿在树干后面显示另一条. spine-unity 提供了一个 SkeletonRenderSeparator 组件，用于将一个 skeleton 分割成多个部件.

淡入和淡出 Skeleton

当降低 skeleton 的 alpha 值使其半透明时，alpha blending 将导致 skeleton 后面的部分露出来。这是一个常见的问题，因为每次绘制三角形时才会应用透明度.

解决该问题的一个办法是使用一个临时的 RenderTexture。可以在正常的不透明度下将整个角色渲染到 RenderTexture 上，然后按所需的淡化不透明度(fade opacity)将 RenderTexture 的内容绘制到场景中.
注意，这只是实现淡出效果的多种方法之一。可能还有其他更简单的解决方案，比如用纯色来渐进地 tinting skeleton，或减小比例(scale)等等。现在的 2D 游戏可以承担较大的开销，因为 RenderTextures 是一种开销很大的方案，在过去很少使用.

示例组件

spine-unity 附带了额外的示例组件，展示了一些高级用例的解决方案。下文列出了一些最重要的示例组件.

SkeletonRagdoll

如果需要把一个动画 skeleton 变成一个类似木偶的布娃娃，例如在角色死亡时模拟物理跌落。这可以通过 `SkeletonRagdoll` 和 `SkeletonRagdoll2D` 示例组件来实现，它们提供了一个简单的接口来在 SkeletonRenderer （或子类 SkeletonAnimation 和 SkeletonMecanim）上创建布娃娃物理组件.
你可以在示例场景 `Spine Examples/Other Examples/SkeletonUtility Ragdoll` 中可以找到对 `SkeletonRagdoll2D` 组件的演示.

SkeletonGhost

用于在角色处渲染一个运动轨迹(motion-trail)或运动模糊(motion-blur)效果来模拟速度或力量。`SkeletonGhost` 组件可以附加在 SkeletonRenderer （或子类 SkeletonAnimation 和 SkeletonMecanim）上，使用可定制的 material 来多次绘制 skeleton.

SkeletonUtilityKinematicShadow

用于需要让 skeleton 有些惯性或对其他 skeleton 的运动做出反应，比如让披风以一种更真实的方式跟随角色运动。这可以通过 `SkeletonUtilityKinematicShadow` 组件来实现。它允许铰链(hinge chains)从父 transform 位置的变化或无关的刚体中继承到速度矢量.
你可以在实例场景 `Spine Examples/Other Examples/SkeletonUtility Animated Physics` 中找到 `SkeletonUtilityKinematicShadow` 组件的演示.

RenderExistingMesh

若想通过在不同位置多次渲染同一个动画 skeleton 的副本来节省性能，比如渲染一大群可以重复的 skeleton。或者你想用 URP 着色器比如 `Universal Render Pipeline/Spine/Outline/Skeleton-OutlineOnly` 在选中网格后在其身后再次渲染一个网格。该组件可用于再次渲染一个已经被动画化和更新的 skeleton 网格，以节省动画和计算网格的开销.

Example Scenes

spine-unity 运行时包含示例场景，以演示重要组件和 C# API 在一般场景下的使用。你可以在 `Spine Examples` 顶层目录中找到它们。每个场景都包括一个描述文本，可以帮助你快速探索和理解相关部分

如果你是第一次使用 spine-unity 运行时，强烈建议至少查看一下 `Spine Examples/Getting Started` 中的示例场景

.

Spine Examples / Getting Started

`Spine Examples/Getting Started` 中的示例场景演示了基本组件及基本用例.

1 The Spine GameObject

该场景演示了 SkeletonAnimation 组件和提供必要数据的 SkeletonDataAsset 引用.

2 Controlling Animation

该场景演示了基本的动画 C# API 代码的使用——开始动画以及对动画事件做出响应.

当按下 Play 键时，Spineboy 将依次播放 walk、run、idle 和 turn(转身)等动画。脚步(Footstep)事件将触发对应的音效.

你可以检查附在 `spineboy` GameObject 上的示例脚本 `SpineBeginnerTwo` 组件。它演示了 `SkeletonAnimation.AnimationState.SetAnimation()` 和 `SkeletonAnimation.AnimationState.AddAnimation()` 的用法。脚本 `HandleEventWithAudioExample` 附在 sound GameObject 上，演示了你如何通过在 `SkeletonAnimation.AnimationState.Event` 注册委托来回调自己的事件方法.

3 Controlling Animation Continued

该场景演示了如何通过使用多个动画轨道同时播放动画。还展示了如何使用 `AnimationReferenceAssets` 来替代动画名字符串的使用.

当按下 Play 键时，将开始循环播放 `walk` 动画。同时，在其 timeframe 内还会同时播放次要动画 `gungrab` 和 `gunkeep` .

你可以检查附在 `raptor Skeleton` GameObject 上的示例脚本 `Raptor` 。它展示了如何在一个组件上暴露 AnimationReferenceAsset 属性，并通过调用 `SkeletonAnimation.AnimationState.SetAnimation()` 方法在轨道 0 和 1 上将其作为动画来赋值.

4 Object Oriented Sample

该场景演示了如何按照 Model-View-Controller 的面向对象的软件设计模式来设置一个平台角色。请注意，虽然这种设置可能不会完美适配你的项目，但它将启发你的一些游戏设计思路，说明如何将用户输入、游戏逻辑和可视化部分分离成不同的组件.

当按下 Play 键时，你可以用 WASD（移动）、空格键（跳跃）和鼠标输入（瞄准和射击）控制 Spineboy 角色。另外，你也可以通过 XBOX 手柄来控制它.

你可以检查附在 `PLAYER INPUT` GameObject 上，作为 `controller` 的示例脚本 `SpineboyBeginnerInput`。它修改了附在 `PLAYER Spineboy` GameObject 上代表了 `model` 的 `SpineboyBeginnerModel` 组件的状态。而状态的可视化是由 `SpineboyBeginnerView` 组件执行的，它附在了 `VIEW Spineboy` GameObject 上，作为 `view`。它将启动附加于同一个 GameObject 上的 SkeletonAnimation 组件上的各个动画.

5 Basic Platformer

该场景演示了一个带动画的典型平台游戏用例，包含如跳、跑、下落、落地的动画，并配有粒子和声音效果。它还展示了在 Unity 中如何用 Spine 网格来投射阴影.

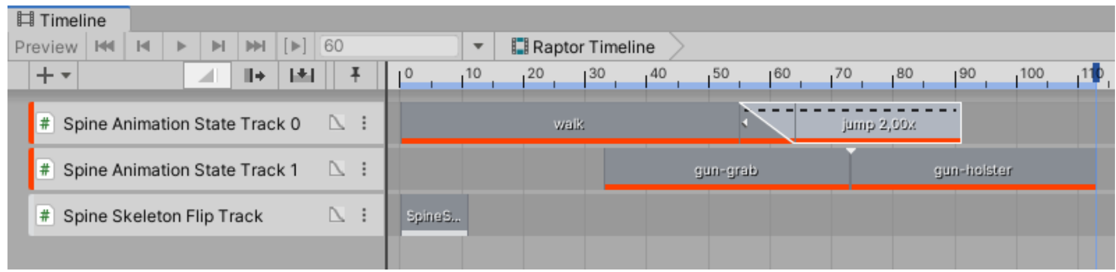
Note: 如果没有看到任何投射出的阴影，请通过 `Edit - Preferences - Quality - Shadows` 来启用阴影.
当按下 Play 键时，你可以用 WASD（移动）和空格键（跳跃）控制英雄角色。或者你也可以通过 XBOX 手柄来控制它.

可以检查附在 `Player` GameObject 上的示例脚本 `BasicPlatformerController` 。它展示了如何使用 Unity input 来改变角色的状态，这些状态保存在一个新建的 `CharacterState` 属性中。当状态改变时，用 `SkeletonAnimationHandleExample` 示例脚本来过渡到新动画，而 `HeroEffectsHandlerExample` 示例脚本则用来播放声音和产生粒子系统.

6 SkeletonGraphic

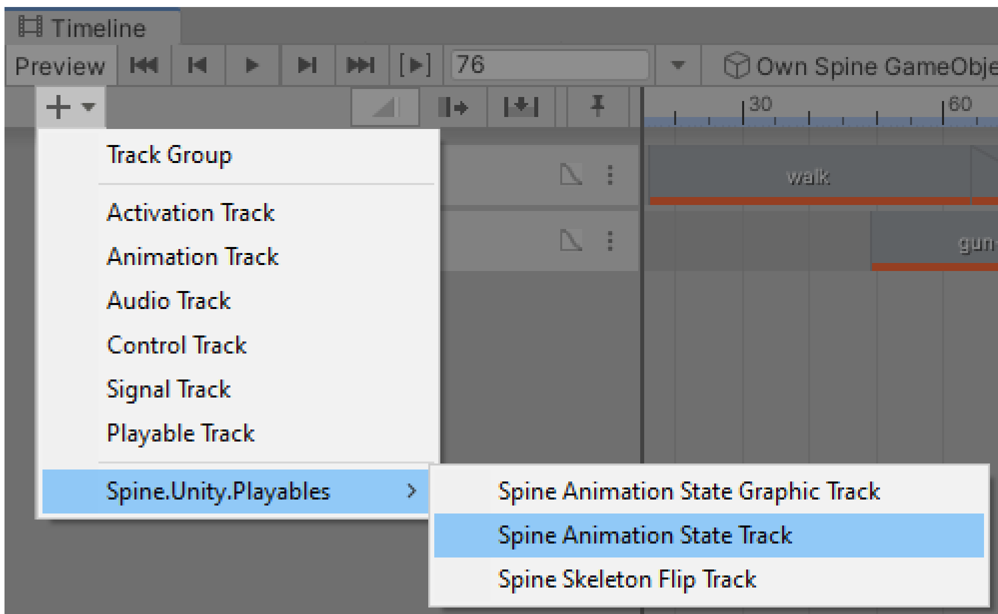
该场景演示了 SkeletonGraphic 组件的使用以及如何将其集成到现有的 Unity UI 中。它还展示了如何使用 `BoneFollowerGraphic` 组件来附加文本标签以跟随骨骼位置。可以在 `Detached BoneFollowerGraphic` 和 `Child BoneFollowerGraphic` GameObjects 中找到它们.
当按下 Play 键时，你会看到一个基于 Canvas 的用户界面，Doi 和 Spineboy 被置入一个可滚动的面板中。两者都像 SkeletonAnimation 组件一样循环播放动画，它们同时也是用户界面的一部分.

Timeline Extension UPM Package



时间线（Timeline）插件作为一个单独的 UPM（Unity Package Manager）包提供。关于如何下载和安装该包，请参见可选 UPM 插件包一节，关于如何更新 UPM 包，请参见 更新 UPM 插件包一节.

Spine-Unity Timeline Playables

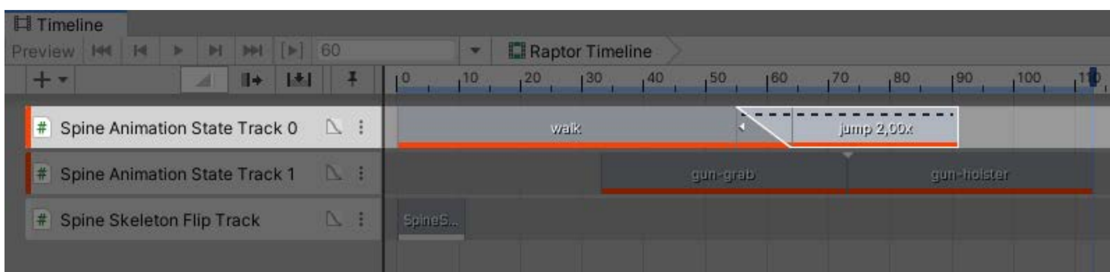


Spine Timeline 目前提供三种类型的时间线 Playables:

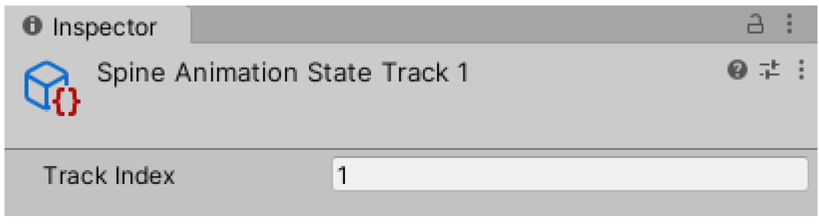
- `Spine AnimationState Track` *(用于* `SkeletonAnimation`*)*,
- `Spine AnimationState Graphic Track` *(用于* `SkeletonGraphic`*)*,
- `Spine Skeleton Flip Track` *(用于* `SkeletonAnimation` *和* `SkeletonGraphic`*)*.

限制: 目前只支持 SkeletonAnimation 和 SkeletonGraphic 。时间线插件目前不支持 SkeletonMecanim.

Spine AnimationState Track



该类型轨道可用于给 `SkeletonAnimation` 或 `SkeletonGraphic` 的 AnimationState 设置动画。`Spine AnimationState Track` 用于 `SkeletonAnimation`，`Spine AnimationState Graphic Track` 则用于 `SkeletonGraphic`.

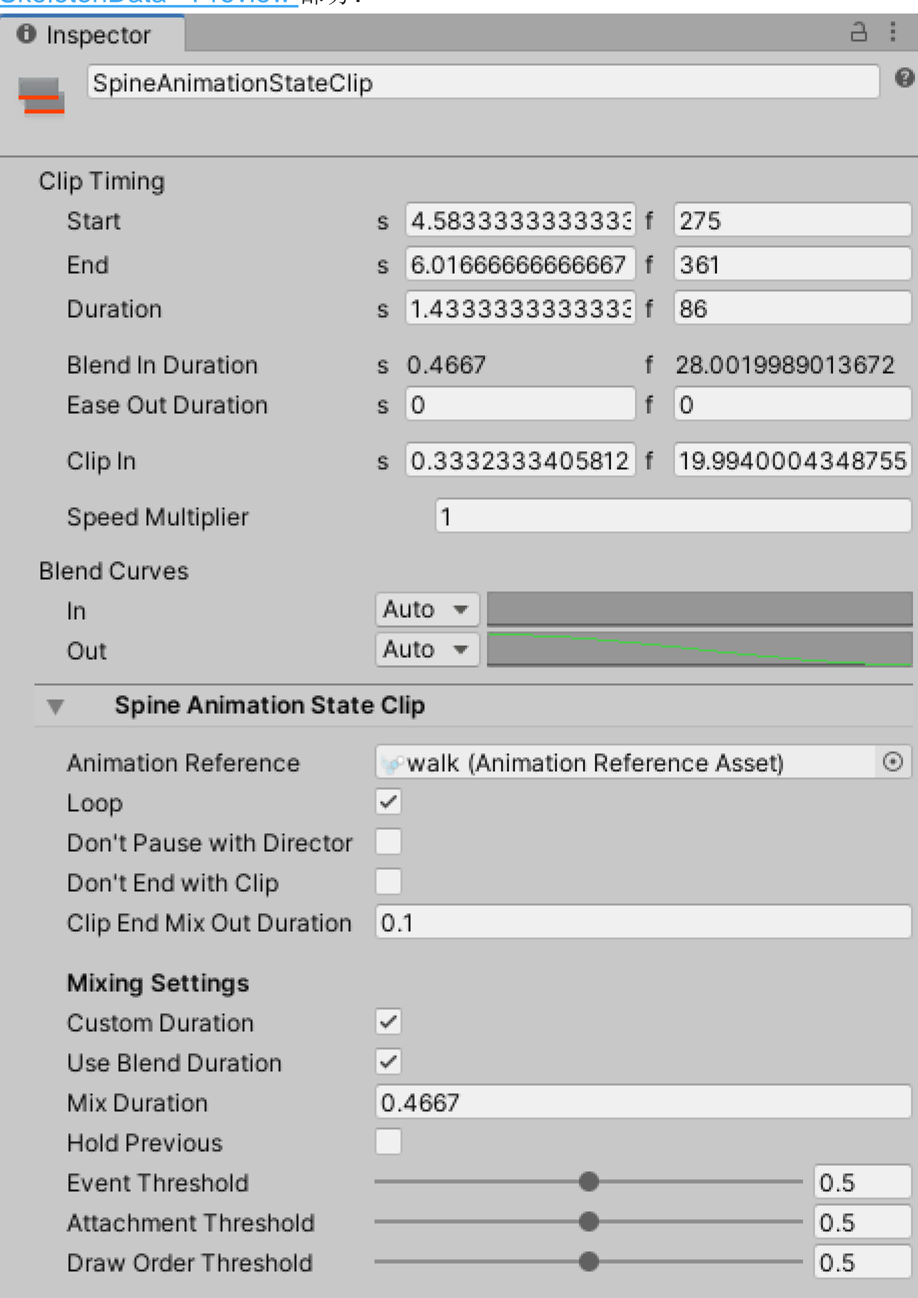


参数:

- *Track Index.* 要设置动画的 AnimationState 轨道索引。当使用多个时间线轨道时，不要忘记相应地设置这个值.
  重要提示: 目前需要对时间线轨道进行排序，基本轨道在顶部，叠加轨道在下方，否则在编辑器预览时将显示错误的结果.

Spine Animation State Clip

可以通过将 `AnimationReferenceAsset` 拖拽到时间线轨道上，来将 `Spine Animation State Clip` 添加到 `Spine AnimationState Track` （或 `Spine AnimationState Graphic Track`）。关于如何给 `SkeletonDataAsset` 生成 `AnimationReferenceAssets`，请参见 SkeletonData - Preview 部分.



参数:

Clip Timing

- *Clip In.* 播放此动画时初始本地开始时间的偏移量。也可以通过拖动 clip 的左边缘来调整.
- *Blend In Duration.* 启用 `Use Blend Duration` 和 `Custom duration` 时使用的 Blend 过渡时长。可以通过把 clip 叠到前一个 clip 来调整，在过渡处会出现一个交叉渐变的三角形.
- *Speed Multiplier.* 播放速度的倍率。当设置为 2.0 时，将以两倍速播放动画，当设置为 0.5 时，以半速播放.

Spine Animation State Clip

- *Don't Pause with Director.* 若置为 true，当 Director 暂停时，动画将继续播放.
- *Don't End with Clip.* 通常情况下，当时间轴上的 clip 后面的时间轴为空时，会在轨道上使用空动画填充。若此参数置为"true"则继续播放 clip 的动画.
- *Clip End Mix Out Duration.* 当 `Don't End with Clip` 为 false 时，如果 clip 后面为空白或停止，就会按 `MixDuration` 设置一个空动画。当该值小于 0 时，clip 则会被暂停.

Mixing Setting

- *Custom duration.* 当启用时，下面的 `Mix Duration` 值将作用于上一个动画到当前动画的过渡。当禁用时，它将使用该动画在 SkeletonData 资产处设置的 `Mix Duration` 值.
- *Use Blend Duration.* 当启用时，`Mix Duration` 值将与时间线 clip 的过渡时长" Ease In Duration"同步。启用此值后，把 clip 叠到前一个 clip 来调整过渡时长，会在过渡处会出现一个交叉渐变的三角形.
- *Mix Duration.* 当启用 *Custom duration* 时，这个 mix 时长将用于从上一个动画到当前动画的过渡.
- *Event Threshold.* 参见 TrackEntry.EventThreshold.
- *Attachment Threshold.* 参见 TrackEntry.AttachmentThreshold.
- *Draw Order Threshold.* 参见 TrackEntry.DrawOrderThreshold.

Ignored Parameters

- *Ease Out Duration, Blend Curves.* 该参数被忽略，不产生任何影响.

使用方法

1. 将 `SkeletonAnimationPlayableHandle` 组件添加到你的 SkeletonAnimation GameObject，如果是 `SkeletonGraphic` 则添加 `SkeletonGraphicPlayableHandle` .
2. 在当前的 Unity Playable Director 中，在 Unity Timeline 窗口中，右击左边的一个空白区域，选择 `Spine.Unity.Playables` - `Spine Animation State Track`.
3. 把 SkeletonAnimation 或 SkeletonGraphic GameObject 拖到新的 Spine AnimationState Track 的空引用属性上.
4. 要在轨道上添加动画，可以香处理普通动画 clip 一样，将各自的 `AnimationReferenceAsset` 拖入 clips 视图（时间线面板的右侧部分）.

请参考 spine-unity 运行时文档的 Preview 部分，了解如何为每个动画创建 `AnimationReferenceAsset`.

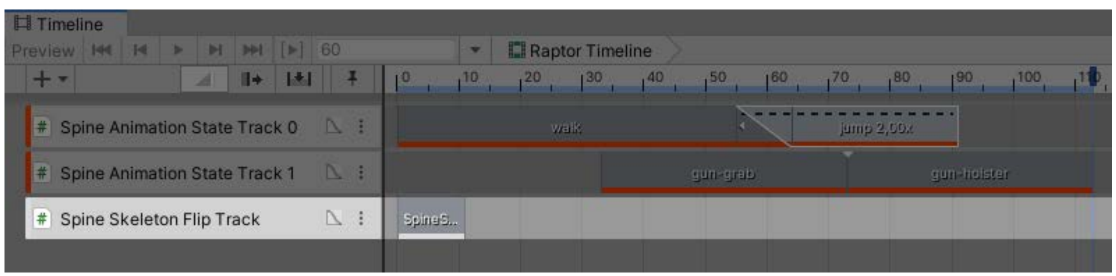
注意: 你可以使用复制功能（`CTRL/CMD` + `D`）在 clips 视图中复制选定的 clips.

轨道行为

- 将在每个 clip 的开头根据 `AnimationReferenceAsset` 调用 `AnimationState.SetAnimation()`.
- 从 Timeline 4.0 版本开始，Clip 时长变得很重要.

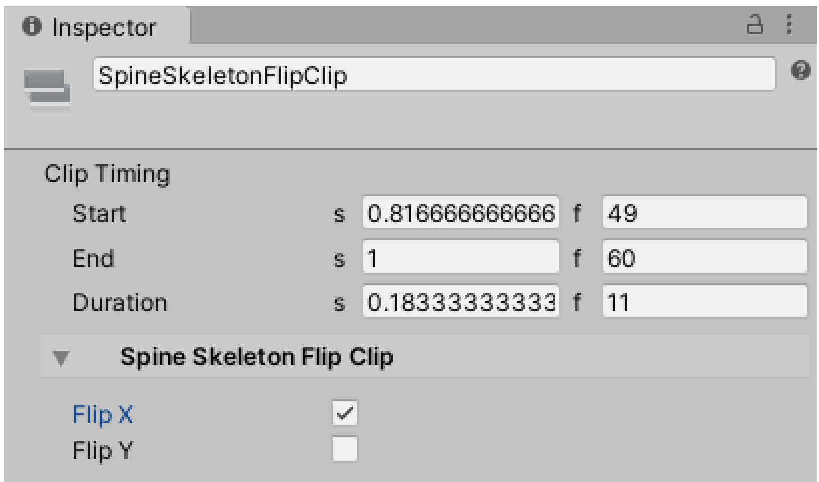
  注意: 3.8 版认为是 Clip 时长不重要。当时间轴上的 clip 后没有其他内容时也不会清除动画.
- 空动画: 如果一个 clip 没有赋值 `AnimationReferenceAsset`，它将调用 `SetEmptyAnimation` 作为代替.
- 错误处理: 如果动画没有找到其 `AnimationReferenceAsset`，它将不产生任何行为（前一动画将继续正常播放）.
- 在 Timeline 开始播放之前，正在播放的动画将不会被打断，直到第一个 clip 开始播放.
- 在 clip 结束和 Timeline 结束时发生的事情取决于 clip 设置。当 `Don't End with Clip` 启用时，在 clip 结束不产生任何行为。这意味着最后一个 clip 的 `SetAnimation` 调用的效果将持续到 AnimationState 发出其他调用。如果 `Don't End with Clip` 禁用，它将按 `Clip End Mix Out Duration` 设置的时长 mix out 到空动画，如果 `Clip End Mix Out Duration` 小于 0 时则暂停播放.
- Edit 模式预览的 mix 可能看起来与 Play 模式的 mix 不同。效果以 Play 模式下的结果为准。请参阅 Spine AnimationState Track 中关于预览多个重叠轨道时正确排序轨道的说明.

Spine Skeleton Flip Track



此轨道类型可用于翻转 `SkeletonAnimation` 或 `SkeletonGraphic` 的 skeleton.

Spine Skeleton Flip Clip



参数

- *Flip X.* 在 clip 内沿 X 轴翻转 skeleton.
- *Flip Y.* 在 clip 内沿 Y 轴翻转 skeleton.

使用方法

1. 将 `SkeletonAnimationPlayableHandle` 组件添加到你的 SkeletonAnimation GameObject，如果使用的是 `SkeletonGraphic`，则添加 `SkeletonGraphicPlayableHandle`.
2. 在当前的 Unity Playable Director 中，在 Unity Timeline 窗口中，右击左边的一个空白区域，选择 `Spine.Unity.Playables` - `Spine Skeleton Flip Track`.
3. 将 SkeletonAnimation 或 SkeletonGraphic GameObject 拖到新的 Spine Skeleton Flip Track 的空引用属性上.
4. 在 timeline dopesheet 中的空行上点击右键，选择 `Add Spine Skeleton Flip Clip Clip`.
5. 调整新 clip 的开始和结束时间，在检查器顶部为它命名，并选择所需的 FlipX 和 FlipY 值.

轨道行为

- 指定的 skeleton 翻转值将用于每个轨道时长内的每一帧.
- 在 timeline 结束时，轨道将把 skeleton 翻转恢复到它该时间线播放开始时记录的翻转值.

已知问题

- 控制台可能会记录一个不正确的、无害的错误: `DrivenPropertyManager has failed to register property "m_Script" of object "Spine GameObject (spineboy-pro)" with driver "" because the property doesn't exist.`.
  这是一个已知的 Unity 问题。请在此了解更多信息： https://forum.unity.com/threads/default-playables-text-switcher-track-error.502903/

暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻
暂时先翻译到这儿吧，后面都是半机翻

运行时术语表

Spine 官方运行时中的常用术语.

Atlas 图集也被称为纹理图集，存储纹理的指定区域。Spine 可以执行纹理打包以创建图集，也可以使用 Texture Packer Pro 等外部工具（大多数运行时使用 "libgdx "图集格式）

An atlas, also known as a texture atlas, stores named regions of a texture. Spine can perform texture packing to create an atlas, or external tools such as Texture Packer Pro can be used (most runtimes use the “libgdx” atlas format).

Animation 一个动画存储一个时间线的列表。每条时间线都存储着键，每个键都有一个时间和一个或多个值。当动画被应用时，时间线使用这些键来操作骨架、触发事件等。动画不存储任何状态
An animation stores a list of timelines. Each timeline stores keys, each of which have a time and one or more values. When the animation is applied, the timelines use the keys to manipulate the skeleton, fire events, etc. The animation does not store any state.

AnimationState 动画状态是一个方便的类，它持有将一个或多个动画应用于骨架的状态。它有一个 "轨道 "的概念，其索引从零开始。每条轨道的动画在每一帧中依次被应用，允许动画被应用在彼此的上面。每个轨道都可以有动画排队，以便以后播放。当当前的动画发生变化时，动画状态还可以处理动画之间的混合（交叉渐变）
An animation state is a convenience class that holds the state for applying one or more animations to a skeleton. It has the notion of “tracks” which are indexed starting at zero. The animation for each track is applied in sequence each frame, allowing animations to be applied on top of each other. Each track can have animations queued for later playback. Animation state also handles mixing (crossfading) between animations when the current animation changes.

Attachment 附着物是通过放置在一个槽上而附着在骨骼上的东西。例如，一个纹理区域或边界框
An attachment is something that is attached to a bone by being placed on a slot. For example, a texture region or bounding box.

AttachmentLoader 附件加载器被 SkeletonJson 用来创建附件。这是一个钩子，可以提供你自己的附件实现，例如进行懒惰加载。最常见的是，附件加载器被用来定制区域附件的图片来源
An attachment loader is used by SkeletonJson to create attachments. This is a hook to provide your own attachment implementations, eg to do lazy loading. Most commonly an attachment loader is used to customize where the images come from for region attachments.

Bone 一个骨骼有一个局部变换（SRT），子骨骼会继承它。一个骨骼也有一个世界变换，它是所有父骨骼变换与本地变换的组合。世界变换使用根骨骼所定义的相同坐标系
A bone has a local transform (SRT) which child bones inherit. A bone also has a world transform, which is the combination of all parent bone transforms with the local transform. The world transform uses the same coordinate system the root bone is defined in.

Bounding box attachment 有一个多边形的附件，用于执行撞击检测、物理模拟等
An attachment that has a polygon for performing hit detection, physics simulation, etc.

Draw order 绘制顺序是一个骨架上的槽的列表。该列表的顺序是每个槽的附件应该被绘制的顺序，从后到前

Draw order is a list of slots on a skeleton. The order of the list is the order each slot’s attachment should be drawn, from back to front.

Mixing 混合也被称为交叉渐变，是通过在当前姿势和动画的姿势之间进行线性混合来应用一个动画
Mixing, also known as crossfading, is applying an animation by blending linearly between the current pose and the pose for the animation.

Region attachment 一个有纹理区域和偏移量 SRT 的附件，SRT 用来定位区域相对于附件的骨骼 An attachment that has a texture region and an offset SRT, which is used to position the region relative to the attachment’s bone.

Slot 插槽是骨骼上的一个占位符。一个槽可以有一个附件，也可以没有附件。它也有一个颜色，以及它的附件被改变后所经过的时间
A slot is a placeholder on a bone. A slot can have either a single attachment or no attachment at all. It also has a color and the time elapsed since its attachment was changed.

Skeleton 骨架持有一个骨架的状态。这包括当前的姿势、骨骼、槽、绘制顺序等
A skeleton holds the state of a skeleton. This includes the current pose, bones, slots, draw order, etc.

SkeletonBounds 骨架界线是一个方便的类，用于使用当前附加的界线盒附件对骨架进行撞击检测
A skeleton bounds is a convenience class for performing hit detection for a skeleton using the currently attached bounding box attachments.

SkeletonData 骨架数据包含骨架信息（绑定姿势的骨骼、槽、绘制顺序、附件、皮肤等）和动画，但不保存任何状态。多个骨架可以共享相同的骨架数据
The skeleton data contains the skeleton information (bind pose bones, slots, draw order, attachments, skins, etc) and animations but does not hold any state. Multiple skeletons can share the same skeleton data.

SkeletonJson 骨架 JSON 从 JSON 中加载一个 SkeletonData

The skeleton JSON loads a SkeletonData from JSON.
SkeletonRenderer The skeleton renderer iterates over the slots in the draw order for a skeleton and knows how to render various attachments.

Skin 皮肤是一个地图，其中键是槽和名称，值是附件。皮肤是一个间接的层次，它允许使用一个槽和一个非特定的名字来找到特定的附件。例如，一个皮

肤可能有一个[slot:head,name:head]的键，值是[attachment:redHead]。另一个

皮肤可能有[attachment:greenHead]作为同一键。非特定的名称在动画中使用，

使改变附件的动画能够与具有不同附件的骨架重复使用
A skin is a map where the key is a slot and name and the value is an attachment. A skin is a level of indirection which allows specific attachments to be found using a slot and a non-specific name. For example, a skin might have a key of [slot:head,name:head] with a value of [attachment:redHead]. Another skin might have [attachment:greenHead] for the same key. The non-specific name is used in animations, enabling animations that change attachments to be reused with skeletons that have different attachments.

SRT 缩放、旋转和平移。也被称为 "变换"

Scale, rotation, and translation. Also known as the “transform”.

Transform 缩放、旋转和平移

The scale, rotation, and translation.