

UNI2D v2.1 REFERENCE



SUMMARY

This document is a complete and in-depth overview of Uni2D v2.1 plug-in.

WHAT'S NEW IN UNI2D?

v2.1

SKINNING

- **Physics Skinning**

The skinning can now be use to deform the collider (Skeletal Smooth Binding > Physics Skin Mode)

- **CPU Skinning**

Added skinning mode for the render mesh (Skeletal Smooth Binding > Render Skin Mode). The CPU modes can be used to batch skinned sprites. Manual update mode is an optimised way of using skeletal deformed sprite without the runtime cost of skinning

- Added “Only Border” option for 3D physics
- Added physics collider Subdivision option (allows smoothest deformation when skinned)

2D

- 2D physics mode (Physics > Dimension). Use Box2D instead of PhysX
- Added Sorting Layer and Order in Layer to the Uni2DSprite

SPRITE

- Added none uniform Sprite Scale

v2.0

SPRITE

- Multi-selection inspector
- Interactive inspector (no more «Generate» buttons)
- Added texture-to-mesh & grid mesh generators
- Sprite editor window stripped down

ANIMATION (NEW FEATURE)

- Brand new animation clip inspector
- Embedded animation clip preview
- Hover-to-preview feature

ATLASING

- Multi-atlasing
- New texture packing algorithm

SKELETAL ANIMATION (NEW FEATURE)

- Pose & animate bone editor
- Smooth binding: easy setup

GENERAL IMPROVEMENTS

- Improved workflow
- Improved consistency
- Improved usability
- Improved responsiveness
- Improved speed, fixed slowdowns and bottlenecks
- Improved memory usage
- Tons of bugs fixed

LET'S BEGIN

Uni2D adds new handy features to Unity which will boost your 2D workflow and productivity. One of the most useful is the sprite creation: with Uni2D, you can create sprites in a flash by dragging textures from the project window and dropping them to the scene view. Uni2D will automatically create sprites or physics sprites from the dragged textures. This section will show you how to setup Uni2D as well as the main features and principles of the plugin.

WHAT IS A SPRITE?

A sprite is a plane mesh with a mapped texture. A sprite is usually perpendicular to a camera. The most simple sprite is a rectangle composed of 2 triangles, but it can be more complex. In Unity, a sprite is a 3D object with no thickness.

WHAT IS A PHYSICS SPRITE?

A physics sprite is a sprite with a physics collider. It can collide to other physics objects and can react accordingly. The physics sprite representation is separated from the rendering one, so the physics shape doesn't usually match exactly the sprite shape. Uni2D can generate a physics collider automatically from the sprite texture.

WHAT IS AN ANIMATION CLIP?

An animation clip is a set of textures, called frames, to display on a sprite at a regular rate to create the illusion of animation. No magic science behind that, an animation clip is basically a film stock.

WHAT IS AN ATLAS?

A texture atlas is a large image containing a collection of smaller sub-images, each of which is a texture for some part of a 2D / 3D object.

When something needs to be rendered on the screen, there's a «draw call» behind it. For various reasons (rendering state changes for example), a draw call is a slow operation and one should keep the draw call count low to have a good frame rate. On low end mobile devices, approximatively 30 to 40 draw calls begin to be critical, depending on the device and the scene to render.

Fortunately, in some conditions, draw calls can be batched. Unity does that automatically. Atlasing is one way to help draw call batching: by using one big texture (the atlas), various sprites can be batched since they share the same texture.

WHAT IS A SKELETAL ANIMATION?

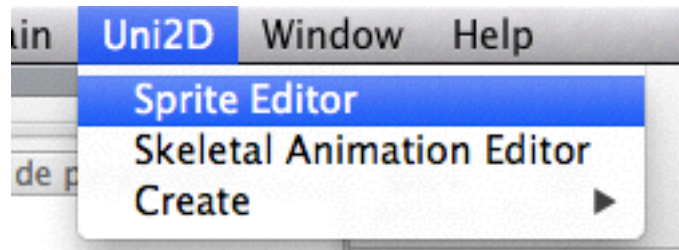
A skeletal animation is a method to animate a 2D / 3D object as a hierarchy of bones which forms a skeleton, pretty much like the human one. By moving a bone, the mesh is deformed depending on the bone influence. This technique is really intuitive and allows you to animate an object like an action figure.

SEAMLESS UPDATE

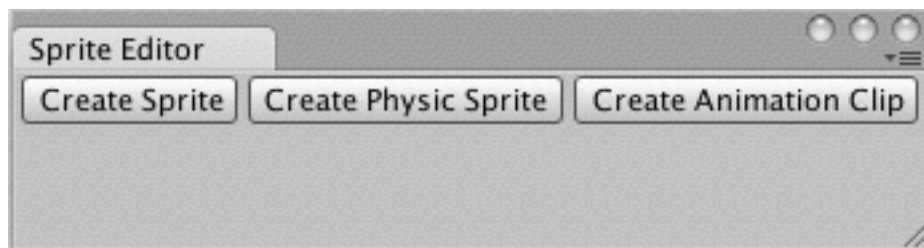
If one of your textures has changed and is used by an Uni2D asset, like a sprite or an animation clip, this asset will be automatically updated. This feature offers a very comfortable workflow since you can edit a texture with your favorite graphic editor and see the results few seconds later in Unity! Not satisfied? Undo the change and everything will be reverted, automatically. Easy!

ENOUGH THEORY!











Well, let's begin! First, go to the Unity menu bar and click [Uni2D > Sprite Editor](#).



This will open the [Sprite Editor](#). As long as this window is open, the drag'n'drop feature is enabled, so don't close it too quickly!



The sprite drag'n'drop feature is now enabled. Several sprite creation shortcuts are handled by Uni2D.

| SHORTCUT | DESCRIPTION |
|---|---|
| Texture drag'n'drop | Creates a new sprite from the texture. |
|  | |
| ALT + Texture drag'n'drop | Creates a new physics sprite from the texture. |
|  /  +  | |
| SHIFT + Textures drag'n'drop | Creates a new animated sprite from the textures (sorted by name). An animation clip is also created. |
|  +  | |
| ALT + SHIFT + Textures drag'n'drop | Creates a new animated sprite with physics from the textures (sorted by name). The physics is computed from the first animation frame texture. An animation clip is also created. |
|  /  +  +  | |

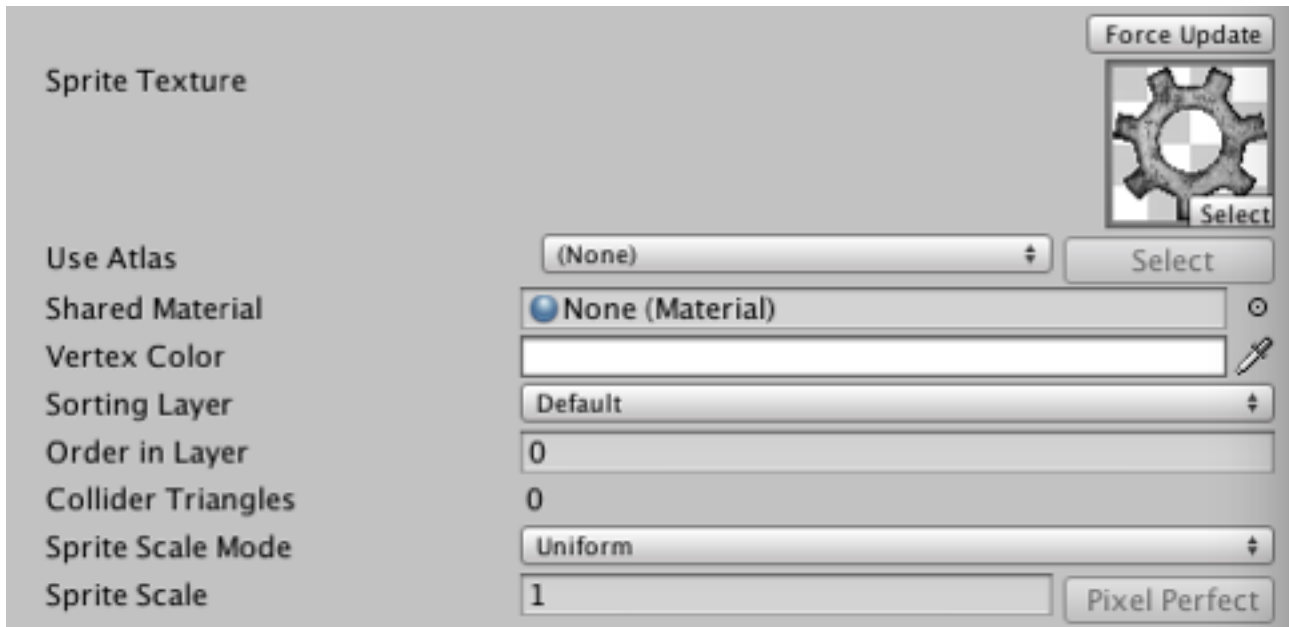
But there's more: as you can see, the sprite editor window contains 3 buttons to create Uni2D assets. Select one or more textures to create the assets of your choice:

| BUTTON | DESCRIPTION |
|-----------------------|--|
| Create sprite | Creates a sprite for each selected texture. Does exactly the same as drag'n'dropping the textures to the scene view. |
| Create physics sprite | Creates a physics sprite for each selected texture. In addition of the sprites, Uni2D computes physics meshes automatically from the texture shape! You can obtain exactly the same results by pressing ALT while performing the drag'n'drop |
| Create animation clip | Creates an animation clip asset from the selected textures. The frames are sorted by texture names. One cool shortcut is to press SHIFT while performing the drag'n'drop: Uni2D will create an animated sprite. Notice you can also combine the physics feature by pressing ALT and SHIFT to obtain an animated physics sprite! The physics mesh is generated from the first texture |

SPRITE


SPRITE COMPONENT AND INSPECTOR (UNI2DSPRITE)

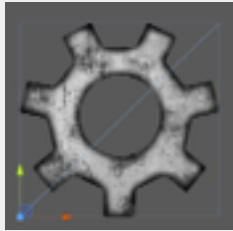



The Uni2DSprite component inspector displays all the setting you can tweak to shape your sprites to your liking. The inspector supports multi-selection edition and can save you a lot of precious time.



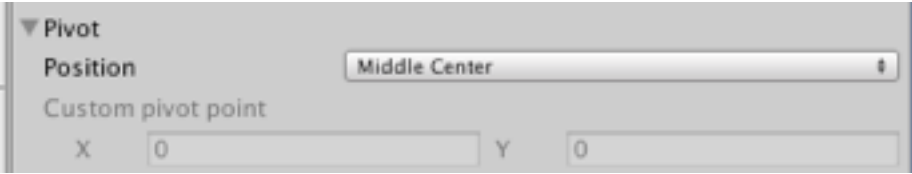
SPRITE INSPECTOR OVERVIEW

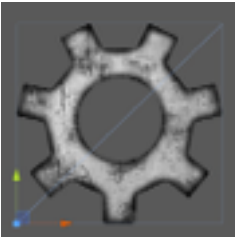
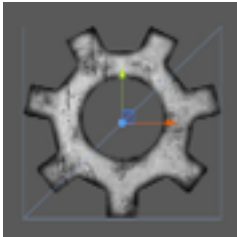

Sprite section

| SETTING | DESCRIPTION | |
|-----------------|---|---|
| Sprite texture | The texture used by the sprite. |  |
| Use atlas | The atlas in which the texture is contained. The atlases already containing the sprite texture are listed below, all the other available atlases are listed in the «All atlases» sub-menu. You can also create a new atlas containing the sprite texture right away by clicking «Create a new atlas...». See Atlasing section for more details. | |
| Shared Material | By default, if not atlased, the sprite has its own unique material. If you want to share a material between several sprites (for optimization and/or ease of use) set this field to the material of your choice. | |

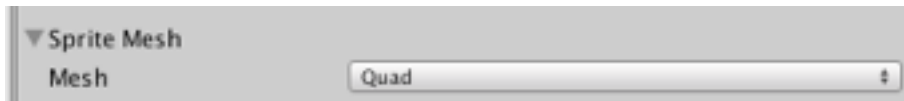
| SETTING | DESCRIPTION | | |
|--------------------|---|--|---|
| Vertex color | The sprite mesh tint. This property is also accessible from script at runtime (see API section for more details). | White  | Red  |
| Sorting Layer | The layer used to define this sprite's overlay priority during rendering. | | |
| Order In Layer | The overlay priority of this sprite within its layer. Lower numbers are rendered first and subsequent numbers overlay those below. | | |
| Collider triangles | Generated physics meshes triangle count summary, for information purpose. Try to keep this number low when creating physics sprites. | | |
| Sprite scale mode | Select either Uniform scale (one number to control the sprite square size) or Not Uniform scale (two number to control each sprite dimension) | | |
| Sprite scale | The base scale used to generate the sprite. Uni2D uses the texture width and height as base sizes then applies a 0,01 factor to them. The default scale 1,0 and means a 512 x 512 texture will produce in Unity a 5,12 x 5,12 sprite, 0,5 will give a 2,56 x 2,56 sprite and so on. | Scale 1  | Scale 0.5  |

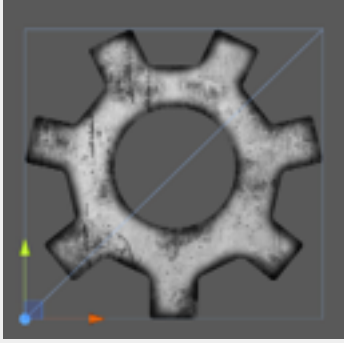
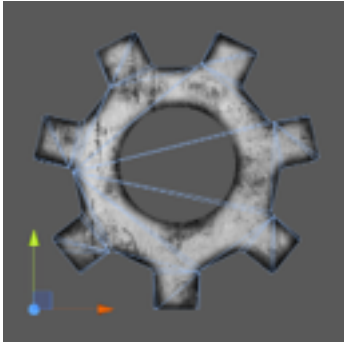
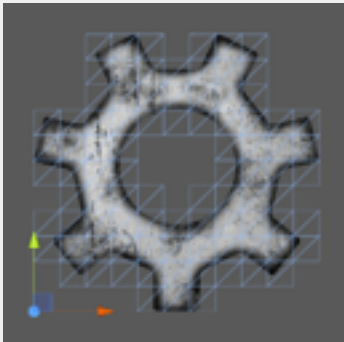
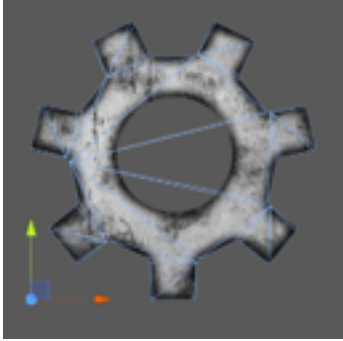
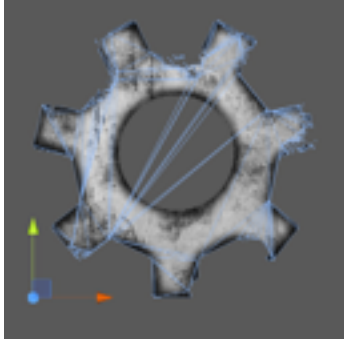
Pivot section

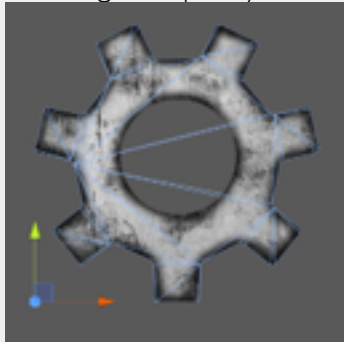
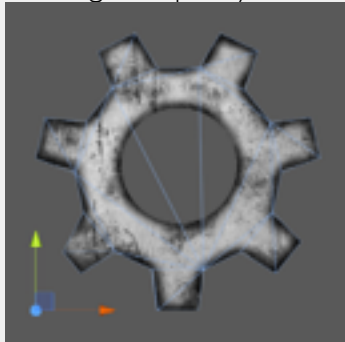
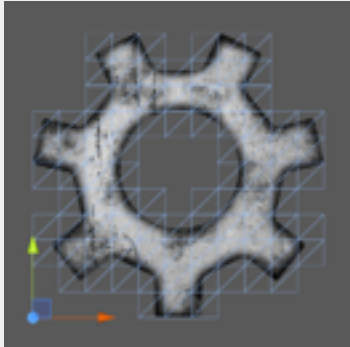
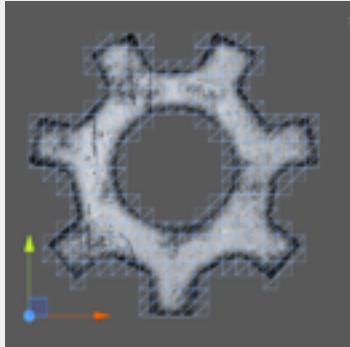


| SETTING | DESCRIPTION | | |
|----------|--|--|---|
| Position | Predefined position of the pivot. Default is Middle Center. | Bottom left | Middle center |
| | |  |  |
| Custom | Use this to place the pivot to your liking. | Custom (X:350,Y:450)  | |

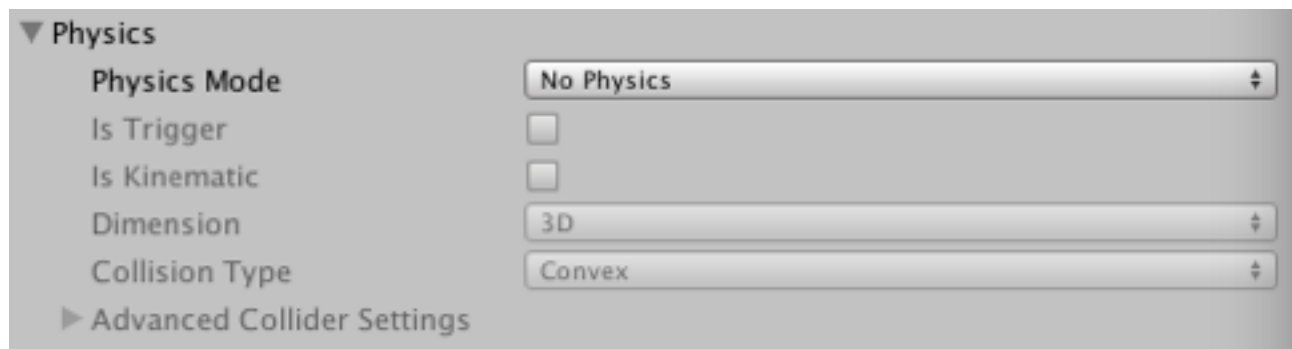
Sprite Mesh section

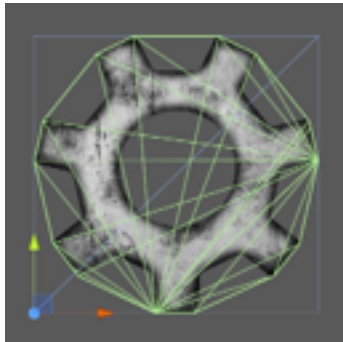
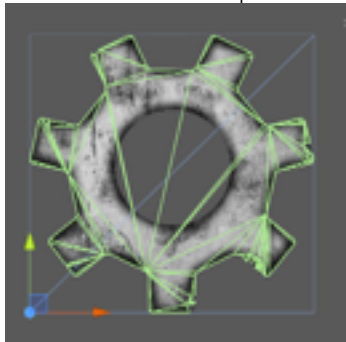


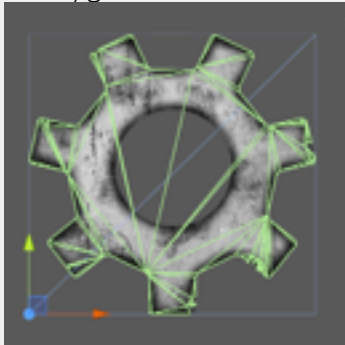
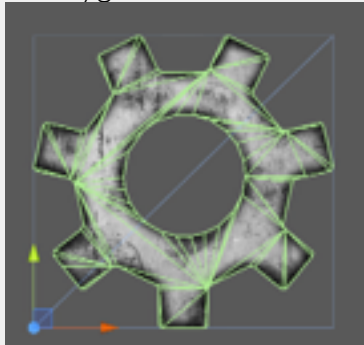
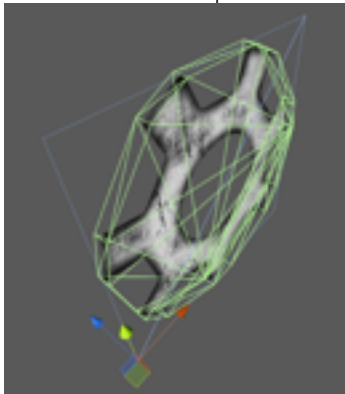
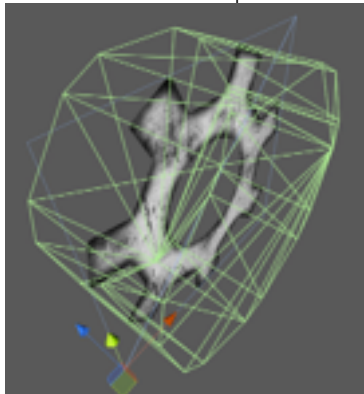
| SETTING | DESCRIPTION | | |
|--|--|---|---|
| Mesh | The mesh used to render the sprite. | | |
| | Quad: generates a simple quad mesh. |  | |
| | Texture-to-mesh: generates a mesh with the shape of the texture. Useful to save fillrate. |  | |
| | Grid: generates a grid mesh. Gives good results when skinning a sprite, also saves some fillrate. |  | |
| Alpha cut off (texture-to-mesh & grid only) | Texture pixels with or a equal alpha value than this threshold will be considered when generating the mesh. Zero value means all pixels are considered and results in a quad mesh. | Alpha cut off 0.1  | Alpha cut off 0.95  |

| SETTING | DESCRIPTION | | |
|---|--|---|--|
| Edge simplicity (texture-to-mesh only) | Use this setting to simplify the generated mesh topology. A low value means a very defined mesh, a high value means a very simple shape. Useful to reduce the triangle count. | Edge simplicity 5  | Edge simplicity 50  |
| Use physics settings (texture-to-mesh only) | Tick this checkbox if you want your sprite mesh to use the same settings as your physics mesh. | | |
| Horizontal / Vertical subdivs (grid only) | Number of horizontal and vertical subdivisions to create in the grid. Increase these values to create beautiful deformations on skinned sprites. No horizontal and vertical subdivisions result in a quad. | Sub divs H:10 V:10  | Sub divs H:20 V:20  |

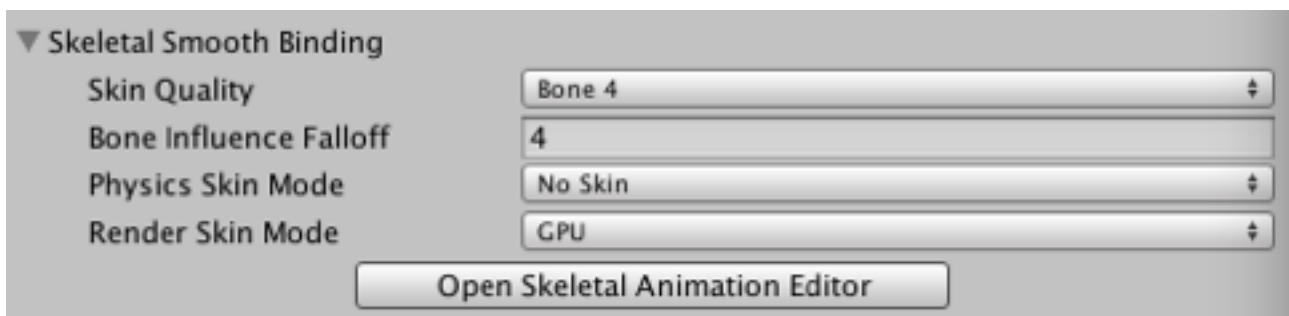
Physics section



| SETTING | DESCRIPTION | | |
|--|--|--|---|
| Physics mode | <p>The physics to apply to the sprite:</p> <ul style="list-style-type: none"> • No Physics: the sprite has no physics • Static: a mesh collider is attached to the sprite • Dynamic: a mesh collider and a rigidbody are attached to the sprite. | | |
| Is Trigger | If enabled, this Collider is used for triggering events, and is ignored by the physics engine. | | |
| Is Kinematic (dynamic physics mode only) | If enabled, the object will not be driven by the physics engine, and can only be manipulated by its Transform. This is useful for moving platforms or if you want to animate a Rigidbody that has a HingeJoint attached. | | |
| Dimension | <p>Choose between 2D or 3D physics.</p> <p>There are actually two separate physics engines in Unity, one for 3D physics and one for 2D physics. The main concepts are identical between the two engines (except for the extra dimension in 3D) but they are implemented with different components. So for example, there is Rigidbody component for 3D physics and an analogous Rigidbody 2D for 2D physics.</p> | | |
| Collision type | <p>The shape of the physics meshes to generate. Compound type is composed of several colliders and allows concave-concave collisions at the expense of speed.</p> | <p>Convex</p>  | <p>Concave / Compound</p>  |
| Alpha cut off | Same as sprite mesh additional options, but for physics meshes. | | |

| SETTING | DESCRIPTION | | |
|---|---|---|--|
| Edge simplicity | Same as sprite mesh additional options, but for physics meshes. | | |
| Subdivide | Enable this to subdivide each edge. This can be useful when the collider is bent by bones. | | |
| Subdivision Count | Number of time each edge are subdivided when Subdivide is enabled. | | |
| Polygonize holes (concave & compound collision types only) | Turn it on to consider holes when generating physics meshes. | Polygonize holes OFF  | Polygonize holes ON  |
| Extrusion depth | All generated physics meshes have a thickness. Use this setting to tune it. | Extrusion depth 0.5  | Extrusion depth 5  |
| Only Border (3D physics only) | When on 3D mode, turn it on to only construct the border of the mesh and not fill its interior. | | |

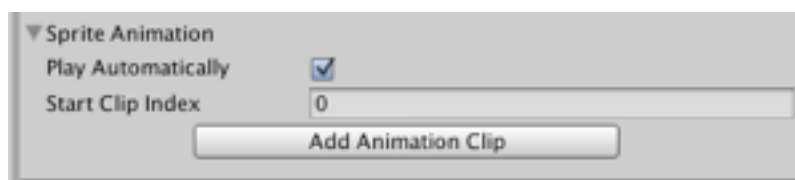
Skeletal smooth binding section



| SETTING | DESCRIPTION |
|------------------------|--|
| Skin quality | Number of bone influences to consider for each vertex. Greater values provide better results but are more computationally intensive. The «Auto» value uses Unity's current platform default quality described in your project build settings. |
| Bone influence falloff | Tune this value to set the influence area of your bones. Bone influence is computed in function of the distance between a bone and a vertex. Low values means less falloff, so big influence areas, high values means great falloffs, so small influence areas. |
| Physics skin mode | <ul style="list-style-type: none">• No Skin: The bones have no influence on the physics collider• Manual Update: Moving the bones deform the physics collider in editor but not in play mode. However, you can update manually the physics skin by script. (If mySprite is a Uni2D Sprite call mySprite.Skinning.UpdatePhysicsSkinning())• Auto Update: Moving the bones deform the physics collider in editor and in play mode. Note : Only update when bones move, so really inexpensive when bones stay still. |
| Render skin mode | <ul style="list-style-type: none">• GPU: Use the default Unity skin deformation. Deformation computation is more optimised as it can be executed on the GPU. But the computation occur every frame so it can be overkill for static bent object. An other side effect is the impossibility to batch two skinned mesh even when they have the same material, resulting in two draw call instead of one.• Manual Update CPU: Moving the bones deform the render mesh in editor but not in play mode. However, you can update manually the render skin by script. (If mySprite is a Uni2D Sprite call mySprite.Skinning.UpdateRenderMeshSkinning()). Good for static bent object as there is no runtime computation to keep the deformed mesh. The deformed meshes can also be batched.• Auto Update CPU: Moving the bones deform the render mesh in editor and in play mode. Can be very expensive if there is too much vertex or bones but the render meshes can be batched. Note : Only update when bones move, so really inexpensive when bones stay still. |

| SETTING | DESCRIPTION |
|--------------------------------|---|
| Open skeletal animation editor | Click this button to open the Skeletal Animation Editor. More details in the dedicated section. |

Animation section



| SETTING | DESCRIPTION |
|--------------------|---|
| Play automatically | Tick this checkbox to play the default clip after sprite initialization. |
| Start clip index | The index of the first animation clip to play. |
| Animation clips | The animation clips attached to the sprite. More details in the Animation Clip section |
| Add animation clip | Click this button to attach an animation clip to the sprite. You can also create a new animation clip and attach it to the sprite by clicking on «Create a new animation clip...» |

TEXTURE ATLAS

WHY SHOULD I USE TEXTURE ATLASES?

Using atlases is a widespread good practice and has plenty of benefits¹. When something needs to be rendered on the screen, there's a «draw call» behind it. For various reasons (rendering state changes for example), a draw call is a slow operation and one should keep the draw call count low to have a good frame rate. On mobile devices, approximatively 30 to 40 draw calls begin to be critical, depending on the device and the scene to render.

Fortunately, in some conditions, draw calls can be batched. Unity does that automatically. Atlasing is one way to help draw call batching: by using one big texture (the atlas), various sprites can be batched since they share the same texture.

There's plenty of ways to create a new Uni2D atlas. The easiest way is through the project window, by clicking on the «Create» button and choosing «Uni2D > Texture Atlas». Select the atlas to display the atlas inspector.

ATLAS INSPECTOR

Core settings section



| SETTING | DESCRIPTION |
|--------------------|---|
| Material override | For each atlas texture, a new material is generated. You can override this material by selecting a material of your choice. Leave this field to None to use the default Uni2D material. |
| Maximum atlas size | The maximum size for an atlas texture. Once the maximum size is reached, another atlas texture of this maximum size is created if needed. |
| Padding | Amount of padding pixels to add between each atlased texture borders. Useful when you want to remove artifacts when using texture filtering. |

¹ What is a sprite sheet? — <http://www.codeandweb.com/what-is-a-sprite-sheet>

Textures section

This section lists the contained (i.e. atlased) textures. Simply drag'n'drop your textures here to atlas them.

Output section



This section is purely informational and shows the generated data for this atlas. Click on the displayed asset names to highlight them in the project view.

Actions section



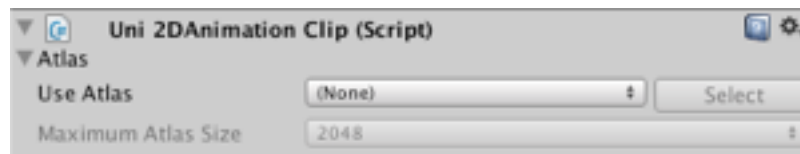
| SETTING | DESCRIPTION |
|--------------------------|--|
| Apply | Click to apply the settings. Since generating an atlas can be a long process, the atlas is not regenerated at each setting change. |
| Revert | Click to cancel your changes. |
| Force atlas regeneration | Click to force the atlas clip to be fully regenerated from scratch. |

ANIMATION CLIP

An animation clip is a reusable set of frames to be displayed by a sprite at a given rate. Each frame contains a texture and an optional event

ANIMATION CLIP INSPECTOR

Atlas section



| SETTING | DESCRIPTION |
|--------------------|--|
| Use atlas | The global atlas for this animation clip. Use this to set the same atlas for all animation clip frames. Leave it to None to keep the ability to choose a specific atlas for each animation clip frame. |
| Maximum atlas size | The maximum size of the global atlas. Only available if a global atlas is set. |

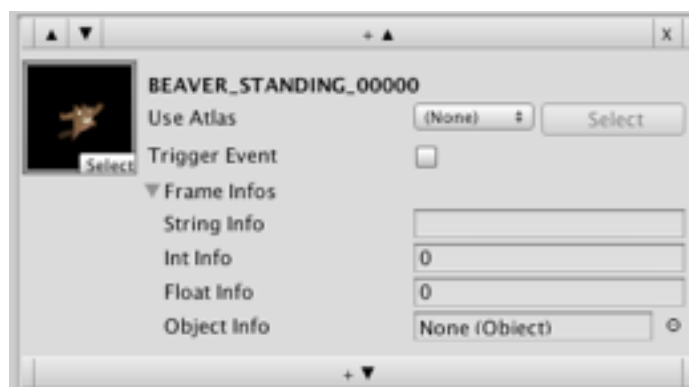
Clip header section



| SETTING | DESCRIPTION |
|------------------------|--|
| Animation clip preview | Hover the texture to play a quick preview of the animation clip. |
| Frame rate | The frame rate of the animation clip, in frame per second unit (FPS). If you want to reverse the animation, you can set this to a negative number. |

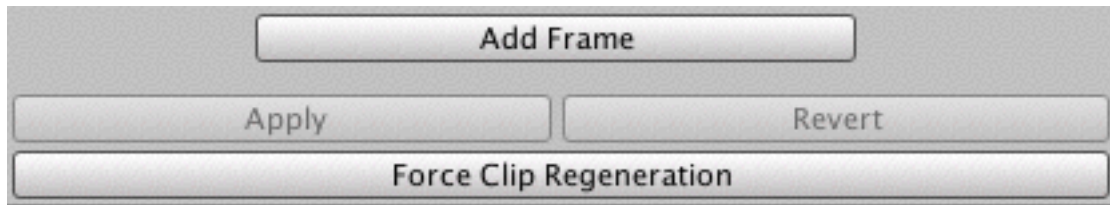
| SETTING | DESCRIPTION |
|-----------|--|
| Wrap mode | <p>This setting describes how the animation clip will loop. This settings is similar to Unity's built-in wrap modes.</p> <ul style="list-style-type: none"> • Loop: plays the animation from first to last frame then returns to first frame and so on. • Ping-Pong: plays the animation from first to last frame then last to first frame indefinitely. • Once: plays the animation once then stop it and returns to the sprite frame. It's the main frame of the sprite and not necessarily the first frame of the animation. • Clamp Forever: plays the animation once from first to last frame then stops at last frame. |

Frame section



| SETTING | DESCRIPTION |
|---------------|---|
| ▲ / ▼ | Click these buttons to move the frame up or down respectively. |
| +▲ / +▼ | Click these buttons to add a above or below this frame respectively. |
| × | Click this button to delete this frame |
| Frame texture | The texture to use for this frame. None is a valid value. |
| Use atlas | The atlas to use for the frame texture. Available only if no global atlas specified. |
| Trigger event | Tick this checkbox if you want Uni2D to trigger an event when this frame is displayed. More infos in the Animation API section. |
| Frame infos | Custom infos to attach to this frame. Typically used as event datas. You can specify a string, an int, a float and a Unity object. More infos in the Animation API section. |

Actions section



| SETTING | DESCRIPTION |
|-------------------------|--|
| Add frame | Adds a new frame at the end of the animation clip |
| Apply | Click to apply the settings. Since generating the animation clip can be a long process, the animation clip is not regenerated at each setting change. Note applying settings can imply an atlas rebuild! |
| Revert | Click to cancel your changes. |
| Force clip regeneration | Click to force the animation clip to be fully regenerated from scratch. Implies atlases to be rebuilt. |

Embedded player section



In this sub-window, you can preview the animation clip with more playback controls.

ANIMATION API

Uni2D provides some basic methods to manage your animation. The animation API can be used in JavaScript.

The `Uni2DSprite` class has a `SpriteAnimation` property which returns an `Uni2DSpriteAnimation` object reference. This object controls the animation of the sprite.

- **bool** `playAutomatically`: specify if the animation will whether or not be automatically played after sprite awake
- **Uni2DSprite** `Sprite` (read-only): the sprite this object is referencing to.
- **Uni2DAnimationClip.WrapMode** `WrapMode`: how the animation will loop (see Animation Clip inspector).
- **float** `FrameRate`: overrides the animation clip frame rate. Use this to modify the way the animation is played. This does not change the speed with which the animation time advances. It allows the animation to be shorter or longer. It's a good practice to negate the frame rate to reverse the animation. The animation will then be played from last to first.
- **int** `FrameCount`: how many frames the current animation contains. This number depends on the current animation clip AND the current wrap mode. For example, an animation clip of 10 frames with a ping-pong wrap mode will have a frame count of 18.
- **int** `FrameIndex`: the index of the currently drawn frame.
- **Uni2DAnimationFrame** `Frame` (read-only): the current frame.
- **float** `Time`: the current playing time of the animation between 0,0 and the animation length.
- **float** `NormalizedTime`: the current playing time of the animation, normalized (between 0,0 and 1,0)
- **float** `Speed`: use the speed property to control the animation playing speed. This will make the animation time goes more slowly or quickly by a factor you choose. You can set it to 0,0 if you want to freeze it, -1,0 to come back in time etc.
- **float** `Length` (read-only): the length of the animation clip currently played.
- **string** `Name` (read-only): the name of the animation clip currently played.
- **int** `CurrentClipIndex` (read-only): the index of the animation clip currently played.
- **Uni2DAnimationClip** `Clip` (read-only): the animation clip currently played.
- **int** `ClipCount` (read-only): how many animation clips are attached to the sprite.
- **bool** `Paused`: pauses the animation if set to true, resumes the animation if set to false.
- **void** `Pause()`: pauses the animation.
- **void** `Resume()`: resumes the animation.
- **void** `Play()`: plays the currently selected animation from the beginning (the default selected animation clip is defined by the Start Clip Index value in the sprite inspector).
- **void** `Play(string clipName)`: plays the clip with name clipName from the beginning. It must be attached to the sprite beforehand or the method will do nothing.
- **void** `Play(int clipIndex)`: plays the animation clip with the index clipIndex from the beginning. Does nothing if no clip with such index.
- **int** `GetClipIndexByName(string clipName)`: returns the index of the animation clip with the name clipName. Returns -1 if no such animation clip name.
- **void** `Stop(bool resetToMainFrame = true)`: stops the current animation. If the optional argument resetToMainFrame is set to false (**true** by default), the sprite won't display its main texture but will keep the current animation frame texture.

ANIMATION EVENTS

There are two types of events that could be triggered by an animation: end animations events and frame animation events.

End Animation Event

Automatically triggered at the end of an animation or a loop.

Callback signature:

```
void OnAnimationEndEvent( Uni2DAnimationEvent a_oAnimationEvent )
```

Delegate: `Uni2DSpriteAnimation.onAnimationEndEvent`

Example: `sprite.spriteAnimation.onAnimationEndEvent += myCallback;`

Frame Animation Event

Triggered for each frame with `triggerEvent` param set to `true`.

Callback signature:

```
void OnAnimationFrameEvent( Uni2DAnimationFrameEvent a_oAnimationFrameEvent );
```

Delegate: `Uni2DSpriteAnimation.onAnimationFrameEvent`.

Example: `sprite.spriteAnimation.onAnimationFrameEvent += myCallback;`

The given `Uni2DAnimationFrameEvent` object allows you to access the frame that fired the event. It's worth noting that user-defined frame infos can be accessed from this event. For example, `a_oAnimationFrameEvent.frame.eventInfos.stringInfo`.

SKELETAL ANIMATION

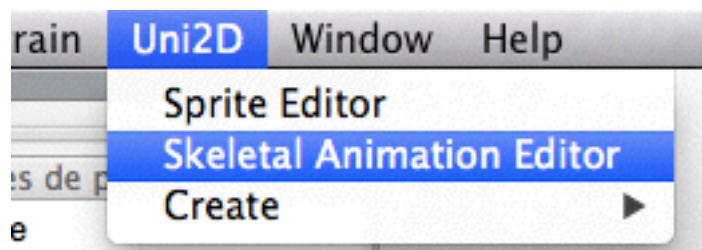
Uni2D v2.0 introduces a new skinning feature: you can now add bones to your sprites and animate them in no time thanks to the embedded skeletal animation editor. To do so, your sprite requires a short setup step first, achievable from the posing mode.

Tip

To obtain good results, we strongly recommend you to use a well tessellated grid render mesh.

SKELETAL ANIMATION EDITOR WINDOW




So, begin by opening the skeletal animation editor from the Uni2D menu > Skeletal Animation Editor. You can also open it from the sprite inspector, by clicking the Open «Skeletal Animation Editor» button.











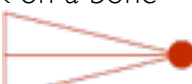







Posing mode

In this mode, one can edit the sprite setup by creating bones. A bone influences vertices in its surroundings so its movements also affect the vertices. The more a vertex is near a bone, the more the vertex is influenced by the bone. A single vertex can be influenced by several bones at once (4 at most in Unity) and the final vertex position is the pondered average of the bones' influence (for example, 30% bone #1, 40% bone #2, 20% bone #3, 10% bone #4).

Posing mode commands

| ACTION | DESCRIPTION |
|---|--|
| Left click on a bone  | Selects the bone. |
| Left click on empty space  | <ul style="list-style-type: none">• Resets selection when a bone is selected.• Creates a bone when no bone is selected. |
| Drag on bone articulation inner disc  | Create a new child bone to this bone. |

| ACTION | DESCRIPTION |
|---|--|
| Drag on bone articulation outer disc  +  | Moves the bone. |
| ALT + drag on bone articulation outer disc  /  +  +  | Moves the bone and its hierarchy. |
| Left click on last created bone, while creating new bones  +  /  | Cancels / exits the creation mode. |
| Right click on a bone  +  /  | Breaks the bone. |
| Right click on a bone articulation  +  | Deletes the bone and its children. |
| Right click while creating a new bone  | Cancels / exits the creation mode. |
| Backspace while creating a bone chain | Deletes the latest created bone in the chain. |
| Escape  | <ul style="list-style-type: none"> •Unselects current bone •Exits posing mode if nothing selected. |

ANIM MODE

After setuping the sprite, switch to the anim mode. Anim mode allows you to handle the bones of your skeleton naturally. *The typical workflow when working in anim mode is to open the Unity animation window, to start recording a new animation and to move your bones along the animation timeline.*

There's no such things as «translations» when speaking of skeletal animation. In theory, a bone should be the same length all over its animation cycle life. Only rotations matter. In practice, translations are used in very particular cases to fake scaling, like when a chameleon sticks its tongue out. For convenience, Uni2D does not handle such error prone cases and will perform rotations only.

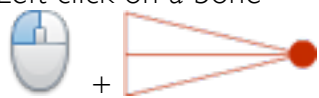

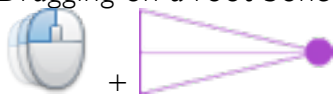
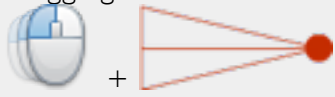

Tip

you can however bypass Uni2D's skeletal animation editor restrictions by manipulating manually the bones' transforms in the scene hierarchy. Only the **Uni2DBone** component is needed by Uni2D to keep working properly.

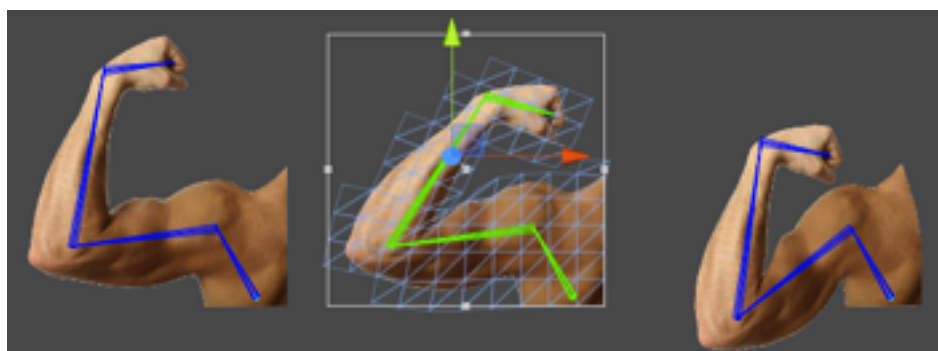
Warning

If you look at the scene graph hierarchy, you should notice that Uni2D generates «fake bones». These fake bones allow you to move independently two children bones of a same parent. They are handled automatically by Uni2D and should not be removed manually.

Anim mode commands

| ACTION | DESCRIPTION |
|---|---|
| Left click on a bone  | Selects the bone |
| Left click on empty space  | Resets selection when a bone is selected |
| Dragging on a root bone  | Moves the bone along sprite plane. |
| Dragging on a child bone  | Rotates the bone. |
| Escape  | <ul style="list-style-type: none">• Resets selection when a bone is selected.• Exits anim mode when no bone is selected. |

Happy animating!

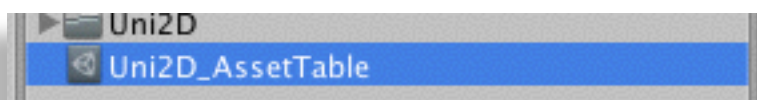


UNI2D ASSET TABLE

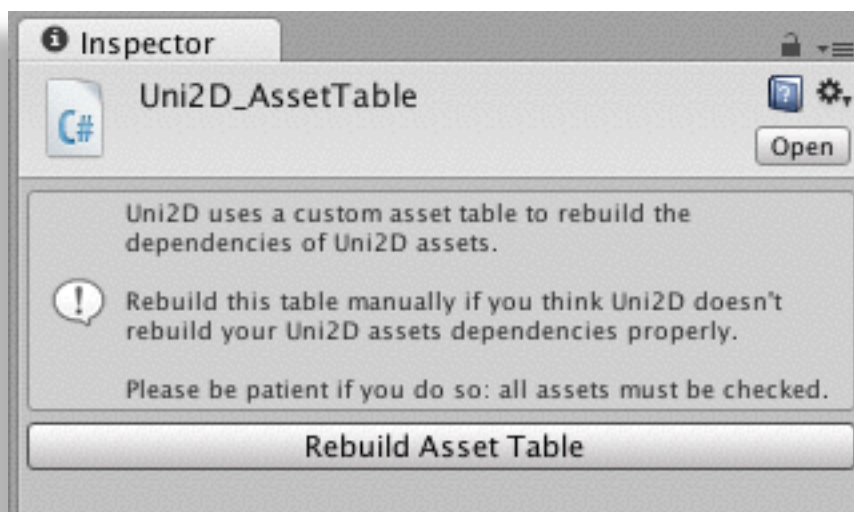
Uni2D builds a custom table to keep reference of your assets GUIDs. This table is used to update assets depending on other assets. For example, when you reimport a texture, Uni2D will check this table to know if an atlas needs to be updated.

Just in case of, you perform a full rebuild of this table if you suspect something is going wrong with Uni2D, like an empty atlas list in the sprite inspector.

First, select the Uni2D asset table. It should be named *Uni2D_AssetTable*.



Then, in the inspector panel, click the button *Rebuild Asset Table*.



To rebuild the table, Uni2D must check all your assets. No reimport is performed so it's a fairly quick operation, but it can take a long time if you have many assets (~45 seconds for 500 assets).

FREQUENTLY ASKED QUESTIONS

Q: Does Uni2D require a Unity Pro license?

A: No, a free license is enough.

Q: Does Uni2D support prefabs?

A: Yes but with limitations: you can create and instantiate prefabs but the prefab connection is lost when the instance is created.

Q: Can I generate physics sprites from a random texture at runtime?

A: Short answer: no. Long answer: yes but not in the shipped version of Uni2D. Uni2D needs to read the texture content itself, which implies the texture to be marked as readable and uses more memory.

Moreover, the original texture size is needed, which is only accessible by importing first the texture as a non-power-of-two texture (NPOT) with a sufficient maximum size. Such overrides uses again more memory and may not be intended by the users, so the original texture import settings are restored when the texture-to-mesh process is over.

To keep a good workflow in the editor, Uni2D does this process automatically behind users' back. Extending this workflow to a runtime use would be quite hard and was not planned for this version.

The texture-to-mesh process itself counts a little in the generation time but remains often time consuming for an online use (~10ms to 80ms, depending on the texture size and complexity). Uni2D is shipped with its full source code. You have all the keys to improve it and to build your very own tool. Feel free to adapt it to your needs.