2D Toolkit
User Manual

Version 1.10

Table of Contents

2D Toolkit.

Thank you for purchasing 2D Toolkit. 2D Toolkit is a set of tools to provide an efficient 2D sprite and text system which integrates seamlessly into the Unity environment. Among the features provided in 2D Toolkit are:

Sprites
- Fire and forget atlasing system. Set it up once, and it will be automatically regenerated when it needs to be.
- Automatically create multiple atlases when necessary.
- Automatically cut up sprites, eliminates empty space in textures and saves altas area and fill-rate.
- Supports arbitrarily sized sprites.
- Set up anchor points so your sprites rotate at the right points, and to line up animated sprites.
- Supports pixel perfect rendering on orthographic and perspective cameras.
- Efficient batching system for background static sprites. Create combined static sprite meshes seamlessly and efficiently.
- Scale sprites without breaking dynamic batching.
- As much data as possible is precomputed offline, runtime transformations are kept to a minimum.
- User friendly editor, creating and using sprites has never been simpler!

Sprite Animations
- Easily build large libraries of sprite animations for your projects.
- Supports arbitrarily sized sprites for animation.
- Sprites for animations can be sourced from any number of atlases.
- Easy to use in-game API.

Text
- Use bitmap fonts created in BMFont / Hiero.
- Tint and scale your text meshes without breaking dynamic batching.
- Supports vertical gradients.

Getting Started.

Sprite Collections.

Sprite collections form the basis of the 2d toolkit. They hold a list of textures and their associated attributes. The system handles creation of atlases and has various options for customization. Once created, any changes to your sprites will automatically rebuild the atlas.

1. Create a sprite collection by clicking on "Create > tk2d > Sprite Collection" in the **Project Window**. Alternatively you can find this menu in "Assets > Create > tk2d > Sprite Collection"
2. This will create a new SpriteCollection object in the project.
   Note: You should create a sprite collection within a folder. This will keep things nice and tidy.
3. You will now need to add your sprites to the sprite collection. You can do this by dragging textures to the Texture Refs parameter within the sprite collection.
4. Click Commit.
5. You are now ready to use your sprite collection.

Sprites.

1. Create a sprite by clicking on "Create > tk2d > Sprite" in the **Hierarchy Window**.
2. Your sprite now appears in the Hierarchy Window and in the viewport. It automatically picks a sprite id based on the information that is available.
3. Click on the sprite object in the Scene Tab or in the Hierarchy Window, and the Sprite editor inspector appears.

Static Sprite Batcher.

A static sprite batcher is a GameObject type which contains a composite of a few different sprites. This is generally used to reduce processing overheads for static backgrounds, whilst keeping the flexibility of combining individual sprites to build it.

1. Create a Static Sprite Batcher by clicking "Create > tk2d >Static Sprite Batcher" in the **Hierarchy Window**.

2. Create child sprites by clicking "Create > tk2d > Sprite"

   - Sprites within a static sprite batcher can only be sourced from one Sprite Collection. You will get an error message if you have sprites from different collections within the same batch.

3. Click the "Commit" button on the Static Sprite Batcher to build a composite mesh based on your child sprites.

4. Repeat the process by clicking on "Edit" to go into edit mode. While in edit mode, you will be able to move, color and position your sprites.

Sprite Animations.

Sprite animations hold a collection of named sprite animations which you can set up in the interface, or instantiate from script.

1. Create a sprite animation collection by clicking on "Create > tk2d > Sprite Animation" in the **Project Window**.
2. Click on the Add clip button to add an animation clip.
3. Set up the number of frames, and select your sprites for the animations.
4. Set up the wrap mode.

Animated Sprites.

An animated sprite is a special type of sprite which can play animations.

1. Create an animated sprite by clicking on "Create > tk2d > Animated Sprite" in the **Hierarchy Window**.
2. Select the appropriate default sprite by expanding the sprite foldout.
3. Select a default animation, and if necessary, tick the "Play Automatically" box to have the animation start playing as soon as the game starts. This is really convenient for looping background animations.

Fonts.

A Font imports a text BMFont exported from the excellent BMFont program (http://www.angelcode.com/products/bmfont/). You can also use Hiero (http://slick.cokeandcode.com/demos/hiero.jnlp) on Mac or Windows if you prefer.

2d Toolkit supports both xml and text font formats. The only limitation is that it only supports a single page of textures. The maximum ASCII character imported is set to 128 by default. You can increase this on the Font object if additional characters are necessary.

For best results, use the following settings:

> **BMFont**
> Export options: 32 bit, Channels A – outline, RGB – one (no outline) or glyph (when an outline has been set up).

> **Hiero**
> in Gylph Cache, Set page width and height so you end up with 1 page exactly. Increase padding around characters.

1. Create a Font by clicking on "Create > tk2d > Font" in the **Project Window**. This is ideally done in the same directory the font was exported to in the previous stage.
2. Create a new material with using the texture associated with the BMFont.
3. Assign the xml / text font, texture and material to the Font object.
4. Click Commit.

TextMeshes.

A TextMesh draws a custom string using a selected Font.

1. Create a TextMesh by clicking on "Create > tk2d > TextMesh" in the **Hierarchy Window**.
2. Select how many characters you wish to display on this TextMesh. Your string will be truncated if it exceeds this limit.

Congratulations, you have now mastered the basics of the 2d toolkit! Read on for some more advanced uses.

Everything in detail.

Sprite Collection
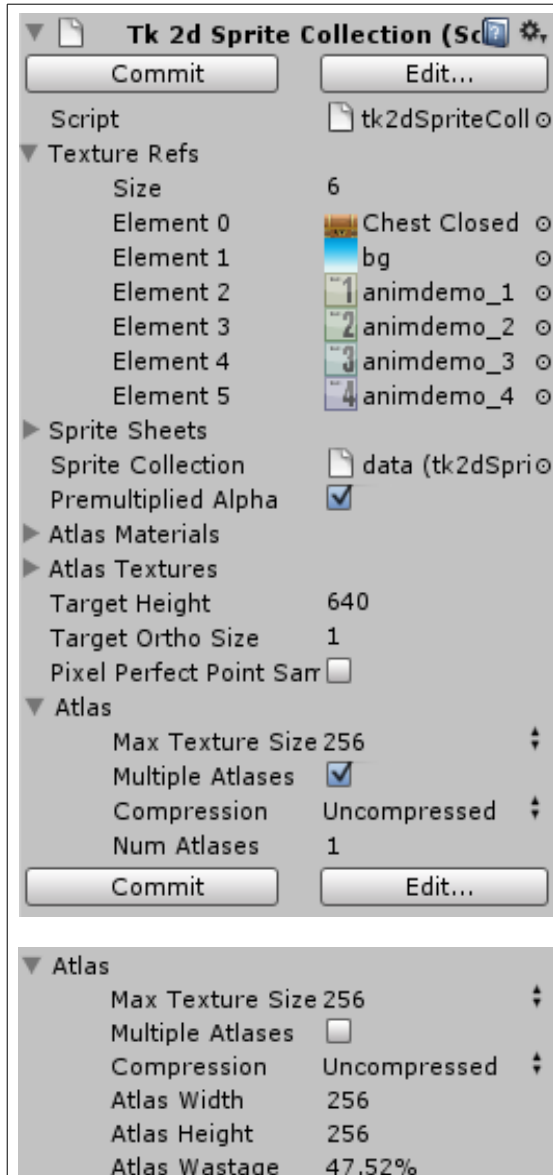
Texture refs – Contains all the textures used in building the atlas. You can delete textures by nulling out the entry in the list. You should NEVER reorder sprites within the list as this will break sprites already set up. Everything is handled by index in game to avoid slow string comparisons.

Sprite collection, Default material and Atlas texture – References to the in-game data. This will be automatically populated when Commit is clicked.

Premultiplied alpha – This is a rendering technique where the alpha is stored premultiplied in the atlas. This has the interesting side-effect of allowing additive and alpha blended sprites within the same sprite collection, and thus within the same draw call. In addition to that, alpha edge artefacts are generally reduced by having this ticked. The downside to having this ticked is that you will lose precision in very transparent textures. Untick this if you have a large number of very semi-transparent sprites.

Target height and Target Ortho Size – See Hints and Tips > Pixel Perfect Sprites and Fonts.

Commit – You will need to click this once when creating the sprite collection for the first time, or when adding or removing sprites from the collection.

Edit – Brings up the sprite collection editor. This will let you set up properties on your sprites.

Pixel Perfect Point Sampled – Disables padding around your textures, and sets the texture to Point Filtered.

Atlas category

Max Texture Size - Sets the maximum texture size allowed from this Sprite Collection. If the combined size exceeds this amount, the sprite collection will not be generated.

Multiple Atlases – Allows multiple atlases to be generated from this Sprite Collection. This is only recommended for animated sprites, as your draw calls can unexpectedly increase depending on the order of sprites in the atlases. When Multiple Atlases are enabled, you will not be able to use Dicing.

Compression – Sets the compression of the atlas texture.

Num Atlases – Read only field showing the number of atlases created.

Atlas Width, Height – Read only fields showing the current atlas width and height. The atlas size is cropped to the smallest possible size, so setting your Max Texture Size to 1024 is fine even if you only use 64x64 as your atlas will be cropped.

Atlas Wastage – Read only field showing how much of atlas space is wasted after trimming. If there is a large amount of wastage, you can put more sprites in the collection at no cost!

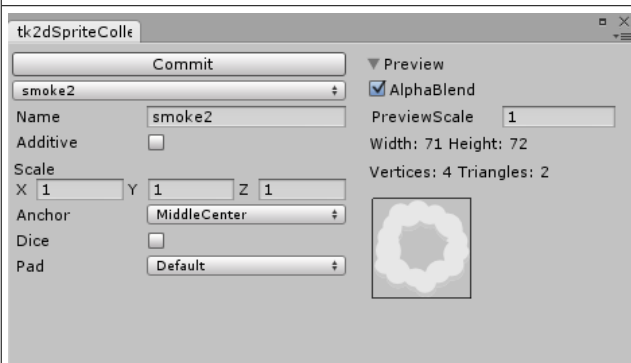|	
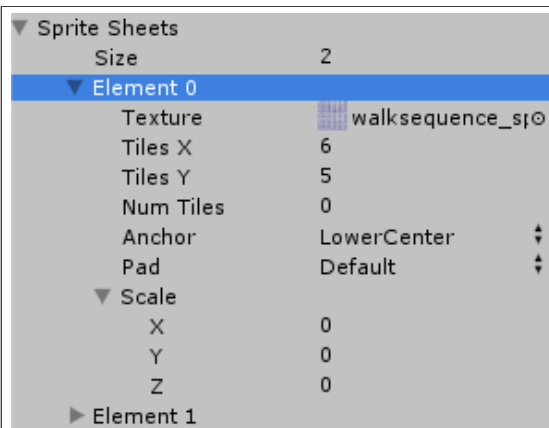
▼ Sprite Sheets
  Size   2
  ▼ Element 0
    Texture   walksequence_sp◉
    Tiles X   6
    Tiles Y   5
    Num Tiles   0
    Anchor   LowerCenter
    Pad   Default
    ▼ Scale
        X   0
        Y   0
        Z   0
  ▶ Element 1
	Version 1.06 introduces support for automatically splitting regularly spaced sprite-sheets. 2D Toolkit will split these up and atlas them like any other texture. These textures do NOT have to be power of two, but the dimension MUST divide perfectly with the number of divisions.
	
	To set these up, simply add entries to the SpriteSheets section of the Sprite Collection set up.
	
	Texture – the texture containing the sprite sheet.
	Tiles X – number of tiles across the texture
	Tiles Y – number of tiles vertically on the texture
	Num Tiles – number of tiles in the texture. 0 means create a full set of textures (i.e. Tiles X * Tiles Y). If you have less than a full atlas, enter the number of tiles here.
	Other parameters – The other parameters are directly transferred to the SpriteCollection, as described in the next section.

tk2dSpriteColle [Commit] ▼ Preview [smoke2 ▾] ☑ AlphaBlend Name smoke2 PreviewScale [1] Additive ☐ Width: 71 Height: 72 Scale Vertices: 4 Triangles: 2 X [1] Y [1] Z [1] Anchor [MiddleCenter ▾] Dice ☐ Pad [Default ▾]	**Name –** Name your sprites so you can easily find them. By default, they inherit the name of the source texture, but you may have multiple textures with the same name. You can also use the syntax "category1/category2/name" to get the drop down to group names together. **Additive –** When ticked, the sprite is treat as additive. This is useful for glow sprites and particles. **Scale –** A default scale to apply to the sprite. **Anchor –** Selects the default anchor point for the sprite. When the anchor point is set to Custom, you will be able to edit the actual anchor points. The anchor point will be represented by a flickering dot on the preview pane. **Dice –** Ticking this will allow you to dice your texture. This will create multiple polygons from your sprite and tightly pack them in the texture sheet, allowing you to use large sparse textures. Set up the dice width and height, while keeping an eye on the vertices and triangle count in the preview pane. Dicing a texture too much could result in a lot of polygon wastage. **Pad –** Override the default padding when you see padding related errors on your sprite. The options available are BlackZeroAlpha which is useful for when representing antialiased sprites, and Extend, which is useful for tightly packed tiles.

Sprite
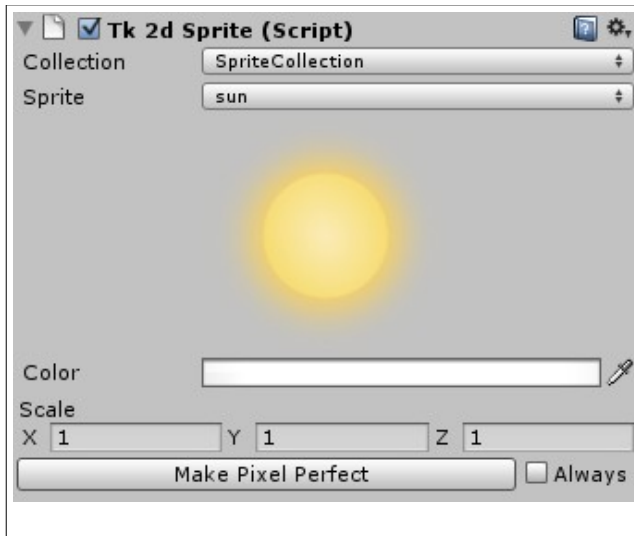
Collection & Sprite – Select a sprite collection, and a sprite from within it.
Color – This color is applied to the vertex colors, so you can use this in your custom shaders to do various things. The provided shaders will tint the texture by the vertex colors.
Scale – Change the scale on the sprite. This is the preferred way to change scale on your sprite, as opposed to changing the local scale on the transform. Changing scale here will not break dynamic batching.
Make Pixel Perfect – Clicking this will make the sprite draw pixel perfect for the current main camera, or if available, based on the Pixel Perfect Helper. If **Always** is selected, this sprite will resize itself to display 1:1 on startup. This will change the scale parameter.

Sprite Animation
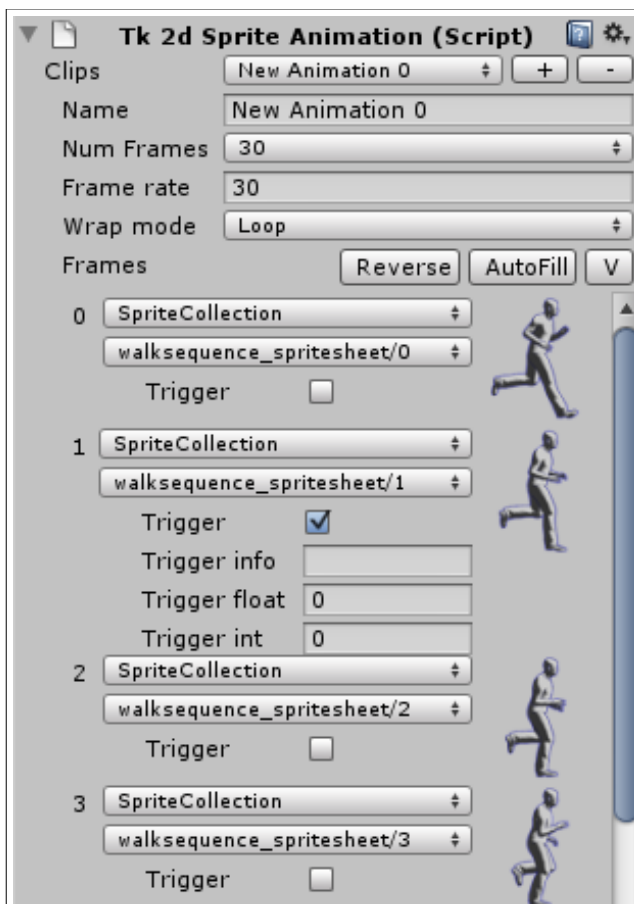
Clips – Lists the currently available animation clips. + adds a new clip and – removes the currently selected one.
Name – You should give your animation clips unique names.
Num Frames - Number of frames in the current animation clip.
Frame Rate – The default framerate for the clip.
Wrap Mode – Options are:
Loop – The animation is looped indefinitely.
LoopSection – A part of the animation is played once and a second section is looped indefinitely.
Once – The animation plays once and then stops on the last frame.
PingPong – The animation sequence is played forward, then in reverse. The last frames are only played once. Eg. 0 – 1 – 2 – 3 – 2 – 1 - …
Single – Simply switches to this sprite. This is useful for continuity in your code, and also for placeholders before you have any animations set up.

Reverse - Reverses all the frames in the currently selected clip.
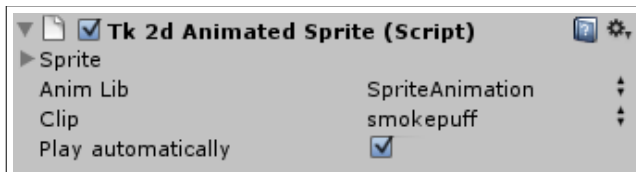**AutoFill –** Looks at the last frame, and looks for following sprite Ids. Eg. If your last frame is spritesheet/1, it will look for spritesheet/2, 3, 4 and so on until it can't find any more ids.
V/H – changes display direction of frames.

Trigger – When set, this frame will fire the FrameEvent callback.
Trigger info / float / int – Values to use as reference in the callback.

Animated Sprite

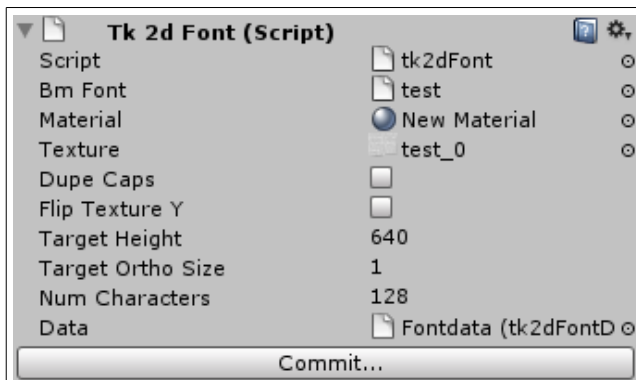|	
▼ ☑ Tk 2d Animated Sprite (Script) 🔲 ⚙︎ ▶ Sprite Anim Lib SpriteAnimation ↕ Clip smokepuff ↕ Play automatically ☑	The Sprite foldout displays a default sprite inspector when expanded. You can use this to set up a default sprite to display. **AnimLib –** Selects an animation library **Clip –** Selects a clip from the selected animation library **Play automatically –** Plays this clip automatically on start up.

Font

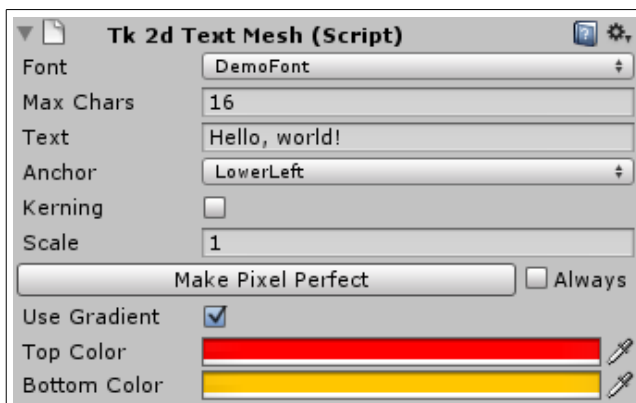|	
▼ Tk 2d Font (Script) 🔲 ⚙︎ Script 📄 tk2dFont ⊙ Bm Font 📄 test ⊙ Material ⚪ New Material ⊙ Texture test_0 ⊙ Dupe Caps ☐ Flip Texture Y ☐ Target Height 640 Target Ortho Size 1 Num Characters 128 Data 📄 Fontdata (tk2dFontD ⊙ Commit...	**BMFont –** Drop your BMFont here. The system supports both text and xml BMFonts. **Material –** Default material used by this BMFont. **Texture –** Texture created by BMFont. **Dupe Caps –** Tick this when you only have caps or lower case characters exported in your BMFont. This is useful when you want to maximize texture space, and don't need lower case / upper case characters. **Flip Texture Y –** Textures generated in Hiero need to be flipped. **Num Characters –** Total number of characters to import. For example, if you do not need a full ASCII set, you can set it to 128. **Target Height and Target Ortho Size -** See Hints and Tips > Pixel Perfect Sprites and Fonts. **Commit –** Commits changes to this Font object.

TextMesh

▼ Tk 2d Text Mesh (Script) 🔲 ⚙︎ Font DemoFont ↕ Max Chars 16 Text Hello, world! Anchor LowerLeft ↕ Kerning ☐ Scale 1 Make Pixel Perfect ☐ Always Use Gradient ☑ Top Color � red bar▌ 🖉 Bottom Color ▌full yellow bar▌ 🖉	**Font –** Select a font from the project. **Max Chars –** Number of character to allocate memory for. The TextMesh will not display more than this number of characters. It is not a good idea to change this while the game is running. **Text –** Default text **Anchor –** Anchor point for the text **Kerning –** Enable or disable kerning on this TextMesh. It is a lot quicker to have kerning disabled, so only turn it on when really needed. **Scale –** Scale the text mesh without breaking instancing. **Make Pixel Perfect -** Clicking this will make the sprite draw pixel perfect for the current main camera, or if available, based on the Pixel Perfect Helper. If **Always** is selected, this sprite will resize itself to display 1:1 on startup. This will change the scale parameter. **Use Gradient and Colors –** Color your text, or apply a vertical gradient on it.

Button

Note: The tk2dButton script is an example of how to create a button using 2D Toolkit. It is a functional button, but you will probably need to modify this script to work properly in your project. Check demo 6 – button and 3d sprites for them in action.

Tk 2d Button (Script)	
Script	tk2dButton
View Camera	None (Camera)
Button Down Sprite	button_down
Button Up Sprite	button_up
Button Pressed Sprite	button_up
Button Down Sound	None (Audio Clip)
Button Up Sound	None (Audio Clip)
Button Pressed Sound	demo_blip_so...
Target Object	Cube
Message Name	StopSpinning

In order to use this script, follow these instructions:
- Create a Sprite and select your initial button sprite.
- Add the tk2dButton behaviour to this Game Object.
- Camera doesn't need to be set if your button is a child of your GUI camera. Otherwise, point it to the camera which will display your button. If you only have one camera in the scene, you may leave this blank.
- Rename "Button Down Sprite" and the others to refect the sprite names you wish to use.

Button Down – Button is being clicked
Button Up – Button is no longer being clicked
Button Pressed – Button has been pressed.

Attach appropriate sounds if you wish, but you will need an AudioSource behaviour if you need it to play sound.

Target Object & Message Name – the target object to send the message.

Hints and Tips.

Pixel perfect sprites and fonts.

2d Toolkit attempts to create all sprites and text meshes pixel perfect by default. In order for it to do this, two variables need to be set up correctly on the Sprite Collection and the Font.

Target Height	Target device pixel height. For example, if your target resolution is 960x640, your Target Height should be 640.
Target Ortho Size	Target ortho size (as set up on camera). This corresponds directly to the "Size" parameter on your camera when "Projection" is set to "Orthographic". Leave it at 1 when using Perspective cameras.

By default, the sprites are created targeting a 640 high device (i.e. iPhone4 in landscape mode), with an ortho camera with an ortho size of 1.0. This should work perfectly even when switching from 960x640 → 480x320 as on the iPhone 3g & iPhone 3gs, as long as the texture is mip-mapped.

However, when designing an universal app for iPad in addition to the iPhone, the iPad version will not display pixel perfect automatically. This can be rectified in a few different ways, this is generally dependent on the type of game you are developing.

- Set up an object with a PixelPerfectHelper behaviour in the game. Set the appropriate target resolution (height) in the box. Eg. When targetting iPad, set the height to 768 / 1024 depending on whether the device is held in landscape or portrait mode. Any objects with "Always make pixel perfect" ticked will scale appropriately to display 1:1 on startup.

- Adjust the camera to correct appropriately. Eg. To scale from iPhone4 to iPad, you'll need to scale up the ortho size by 1.2.

- Use a completely different set of assets targeting the iPad resolution.

Positioning sprites.

The entire Unity mesh positioning tools are available to you when positioning sprites. Try using vertex snap mode to get sprites aligned closely, for instance, to build a tiled background. You can also use any of the alignment tools you can get on the Asset Store.

Multiple copies of the same sprite.

In certain cases, it may be necessary to create multiple copies of the same sprite within a sprite collection. For instance, you may require the same sprite with different default scales, or different anchor points. Each texture is only added once into the atlas, as long as dicing is not enabled on that texture, or other copies of the texture. The only overhead then is the vertex data, which is minimal compared to the overheads of storing the texture multiple times.

Changing the default sprite material.

You can override the default material used by the sprite. You can find the material by selecting the SpriteCollection and clicking on material.

The textures are being padded too much.

Each sprite is currently given 2 pixel padding, to work correctly when scaled up and down, and to work correctly when the resolution is halved for iPhone. If you do not have these constraints, or if you do not scale your sprites, you can change this by changing the pad variable in SpriteCollectionEditor. This is intentionally not exposed as a tweakable property.

Help! My textures are blocky and look compressed.

That is because they are compressed by default. Click on the sprite collection, then click on Atlas Texture to go to the texture used by this sprite collection. You can change the compression type here.

I don't like your implementation of Sprites / TextMeshes. I'd like to write my own.

Sure! Feel free to do so. The data from the back end is available for you to consume in any way you wish. Check tk2dSprite.cs and tk2dTextMesh.cs on how the data is used. As almost everything is precomputed, you can almost use the data as is!

How much performance do all the options cost?

A base sprite is simply a mesh in Unity, and it would be no different if you built the polygons in your favourite 3d program. You only incur a cost when you update the sprite or text mesh, i.e. change the sprite id, color, text, etc. Certain operations are more expensive than others and these are listed below:

Switching sprite ids with different vertex counts requires memory allocation. This is not a good idea at runtime. You will be shown a warning in the SpriteAnimation editor if you do this to alert you to the potential problem.

Changing maxChars on a TextMesh will reallocate memory. This is bad at runtime.

Switching sprite ids with the same vertex counts, changing scale or colors incur about the same cost. No allocation is performed here.

Changing scale & color is dependent on the number of vertices. You should avoid doing this at runtime on massively diced sprites.

I have a frame rate spike when I instantiate a large number of sprites.

The mesh on each sprite is built up at runtime. This means that even thought the data is very optimal and very few transformations are actually performed on this data, there is still a performance hit doing this. In order to get around this problem, you should preallocate and pool your objects at the start of the game. This is generally a good idea on iOS to avoid allocations & garbage collections even when not using this system.

How do I create a Sprite in code?

The sprite classes aren't meant to be instantiated in code. They use references to data blobs instead of the actual sprite collections, and this makes it quite nasty to implement. This is still possible, check tk2dSpriteEditor.DoCreateSpriteObject() for how a sprite is created in the interface.

We recommend you set up prefabs for the sprites you need to instantiate in game and instantiate them instead. That way, you can visually set up your sprite, and you'll be guaranteed everything is set up correctly when you instantiate it in game.

I'd like to switch sprites by name. The API seems awfully lacking.

This is by design. If you need to switch sprites by name, create a SpriteAnimation with as many clips as you need, each with a unique name. For each of these clips, set the wrap mode to "Single" and set up the sprite. You can now switch between sprites by calling Play("clipname"). As an added bonus, if you need to convert any of these to animations or if you need to switch to sprites from multiple collections, you are all set up to go!

Another way to switch sprites by name is shown in demo 6 – button and 3d sprites. Always cache the spriteIds, as string performance isn't very good on the mobile platforms.

I have created a texture which is 1024x1024, why is my atlas 2048x2048?

By default, 2d Toolkit will insert 2 pixels of padding around your texture to support automatic downsampling for iPhone 3g(s). This is also required when linear filtering is used to have correct edge seams. If you are drawing pixel art and are not going to have filtering enabled, tick "**Pixel Perfect Point Sampled**" in the Sprite Collection set-up. This will remove the padding around your texture, and change the filtering mode on your texture to Point.

I am setting .text and .color on my tk2dTextMesh, and nothing is changing. What gives?

You need to call Commit on the textmesh for your changes to be committed. The reason for this is you may change text and color and/or size in the same frame, and each of these will invalidate the mesh data, and by calling Commit yourself, you are saving the system from regenerating data multiple times.

What is .tk2d in my project window. Why does it keep coming back when I delete it?

.tk2d is a global index of all sprite collections, animations and fonts. This is used to speed up operations which need to find these objects. You should not delete this, as it is generated when it is missing.

Unity crashes when I click on a Sprite.

Prior to version 1.10, 2d Toolkit went through your project and loaded all prefabs to find 2d Toolkit objects. As you can imagine, this can be a very slow process when you have a large number of prefabs. 2d Toolkit 1.10 creates an index of all your 2d Toolkit objects the first time you use it.

Try to create a new scene, delete .tk2dIndex, and in this blank scene, create a new sprite. It should take a while to process if you have a large project, but it should succeed in creating the index. It should no longer crash when you click on a sprite, as it will now use the index instead of trying to find objects every single time.

API.

Below are the most commonly used member functions and properties.

tk2dSprite

color	Sets the color of a sprite
spriteId	Sets the spriteId. You can also use GetSpriteIdByName to get a spriteId by name. If you need to do this, it is better to use an Animated Sprite with clips set to "Single"
scale	Sets the scale on this sprite
MakePixelPerfect()	Sets the scale so the sprite displays pixel perfect. You will need to call this if you change the camera and need to update the sprite to display pixel perfect.

tk2dAnimatedSprite

Play(id) Play(string)	Plays an animation clip by name or id.
Pause() Resume()	Pauses and resumes the current animation.
Stop	Stops a currently playing animation.
animationCompleteDelegate	Set up an animationCompleteDelegate to receive notifications when an animation stops playing. This is very powerful when coupled with c# anonymous functions.
animationEventDelegate	Set up an animationEventDelegate to receive notifications when an animation frame has a trigger set.

tk2dTextMesh

Commit()	Commits all changes. You will need to call this after changing an of the members listed below.
text	Currently displayed text
color / color2 / useGradient	Color is the main color, or the top color when useGradient is enabled. Color2 is only used when useGradient is enabled, and represents the bottom color.
anchor	Change the anchor point for this text mesh
scale	Change the scale of this text mesh