

# SpriteBuncher

*A cross-platform, open source texture packing program.*



*The SpriteBuncher main interface (Mac version shown).*

## Quick-start steps:

- ≡ **Open a folder** of images using the button in the *Input* section of the application – or 'drag and drop' a folder onto any part of the application window.
- ≡ **Use the packing settings** to produce the desired layout on the sheet; the best method is generally *MaxRects*. A 2-pixel border and padding is recommended. The program will display a message if there is not enough space on the sheet.
- ≡ **Choose your output format** from the drop-down menu. Check which formats are compatible with your the target library or application.
- ≡ **Click the *Export* button** to generate the data and image output files. The program writes these to a `buncher` folder, inside the original image folder.

## **Introduction**

*SpriteBuncher* is texture/sprite packing application. In game development and other fields, using multiple small image files is usually an inefficient use of video texture memory. This can lead to increased rendering and loading time, or simply running out of available memory. The solution is to pack images onto a *sprite sheet* (sometimes called a *texture atlas*). By packing sprites efficiently, smaller texture sizes can be used, making optimum use of resources. A data file is then used to store the coordinates (and other information) for each individual sprite referenced on the sprite sheet(s).

Common sheet sizes are 512x512 or 1024x1024, *i.e.* square textures, using powers-of-two ('POT'). More powerful devices tend to support larger sizes, such as 2048x or even 4096x, and may also be optimised for non-square textures (check target device specifications). Lowering the output image *quality* is another way to reduce hardware requirements.

*SpriteBuncher* features advanced packing algorithms such as *MaxRects*, and options for cropping, rotation, extruding, and padding. These are described in more detail later. Data can be exported to a range of common output formats – *e.g.* for game engines such as *Cocos2d*, *LibGDX*, and *Unity* – or via generic XML, JSON, and plain text files.

When launched, the application will attempt to load the provided 'sample' folder of images. Most of the screen shots in this document will use this sample project.

*SpriteBuncher* is not only free, but is also open-source (GPLv3 license). This means that users are not limited to what the program can currently do, they can modify it to add new features or to support new platforms. Of course, reporting bugs and contributing changes back to the main project will benefit other users too.

The application is written using *Qt* and C++, with code documented using the *Doxygen* system.

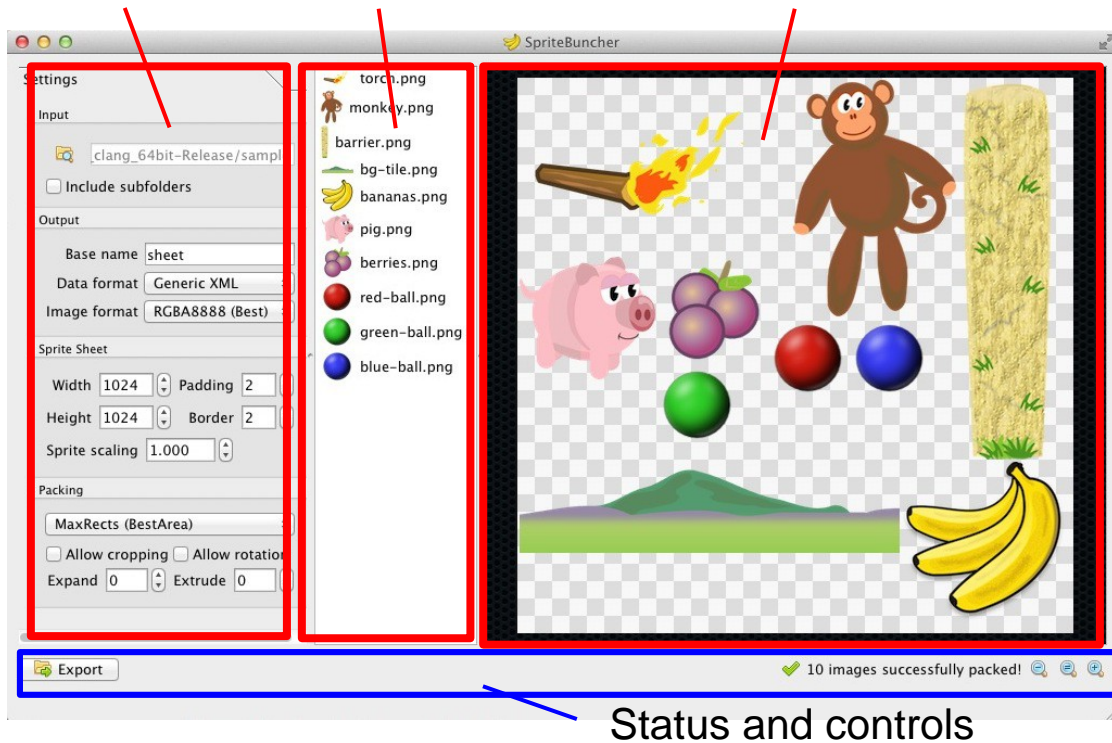
## User interface overview

The interface is split into 3 main sections, with a control/status area below:

Settings panel


Sprite list

Sprite sheet preview



Status and controls


The settings panel contains sections for input, output, sprite sheet parameters, and finally the packing options.

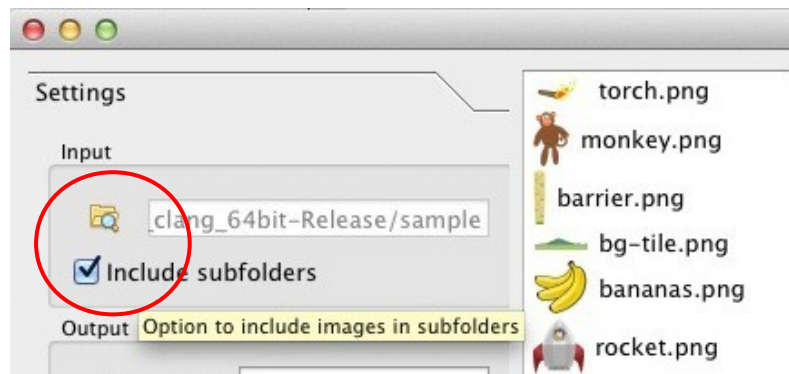
The sprite list view shows thumbnails for all images that were successfully loaded. The sprite sheet preview shows how the sheet will actually look based on the current settings, and will update automatically to reflect any changed settings. Use the zoom controls  to adjust the scale of the preview (this does not affect the output image size). On either sprite view, images can be clicked to select them.

The status area indicates how many sprites have packed successfully. If there are no issues, the *Export* button will become enabled, allowing you to write the output files.

At any time, hovering over an item on the interface will show a tooltip with a description of the feature. Note, you can drag the 'splitter' handles to adjust the relative sizes of the 3 main interface panes. Window geometry is stored between sessions.

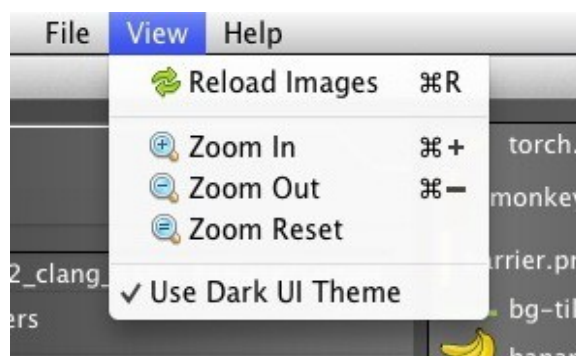
## Importing images

The current image folder path is displayed at the top of the settings panel. Use the  button to browse to a new folder, or simply 'drag and drop' a folder onto any part of the application window. *SpriteBuncher* will attempt to load all valid image files from the folder and, optionally, look inside any subfolders (the default behaviour is to not search subfolders).



For example, if you select the '*Include subfolders*' option for the sample project you will notice that the `rocket.png` image is now loaded (it was inside a subfolder called 'more'). The sprite sheet preview will then update and – if using the default packing settings – will indicate that there is no longer room to pack all the sprites in this case.

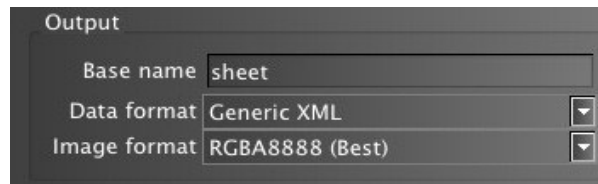
To force a reload of the current folder, e.g. to update any new or altered files, use the *Reload Images* option in the *View* menu:



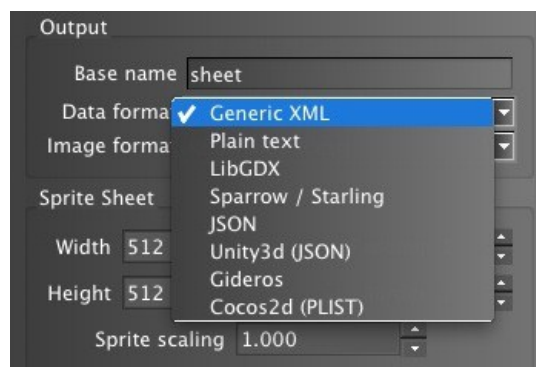
Currently, the application can not load individual images from different folders, it operates on a single location only.

## Output options

The output comprises a data file with its accompanying image file, named using the specified *base name*. Type the required name into the field provided (do not add a file extension).

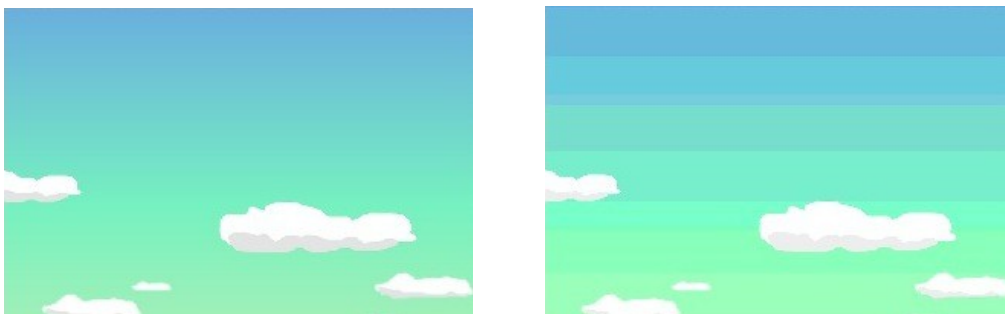


Select a target data format from the drop down list. If in doubt, consult the documentation for your target library or application. The simplest general output formats are *Generic XML* and *Plain text*.



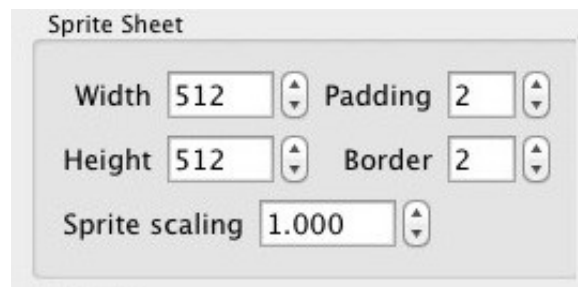
The default output image format is *RGBA8888*, which allocates 4 bytes of data per pixel (1 byte for each of red, green, and blue, and a further byte for alpha/transparency). This provides the best available quality; all other formats will sacrifice image quality for filesize.

Reduced colour-depth formats can produce noticeable colour banding, especially when using smooth gradients. Some formats save space by not storing any alpha channel at all.



*Reduced image quality can result in color-banding artefacts, as shown on the right.*

## Sprite sheet settings



*Default sprite sheet settings*

The default sheet size is 512x512, but this can be set to any size required using the width and height input boxes. In general, a square 'power-of-two' approach is best, but the program does not enforce this. The preview panel will update to display the new size, and attempt to pack the sprites again.

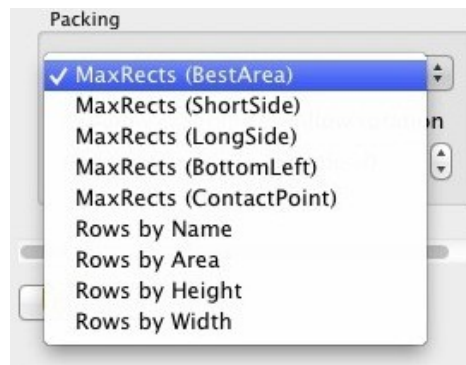
The *Border* option represents the gap around the entire sheet (not individual sprites), and is measured in pixels. Generally, a minimum gap of 2 pixels is best.

*Padding* defines the gap around each sprite, again in pixels. A value of at least 2 is recommended to avoid any 'bleeding' between neighbouring images when the final images are rendered from the sheet.

The final option in this section is for *sprite scaling*. This will scale all sprites by the chosen amount (default 1.0), without changing the dimensions of the sheet itself. This can be useful when exporting multiple sets of assets for different screen resolutions, or simply to scale down the original sizes. Scaling up generally produces poor results.

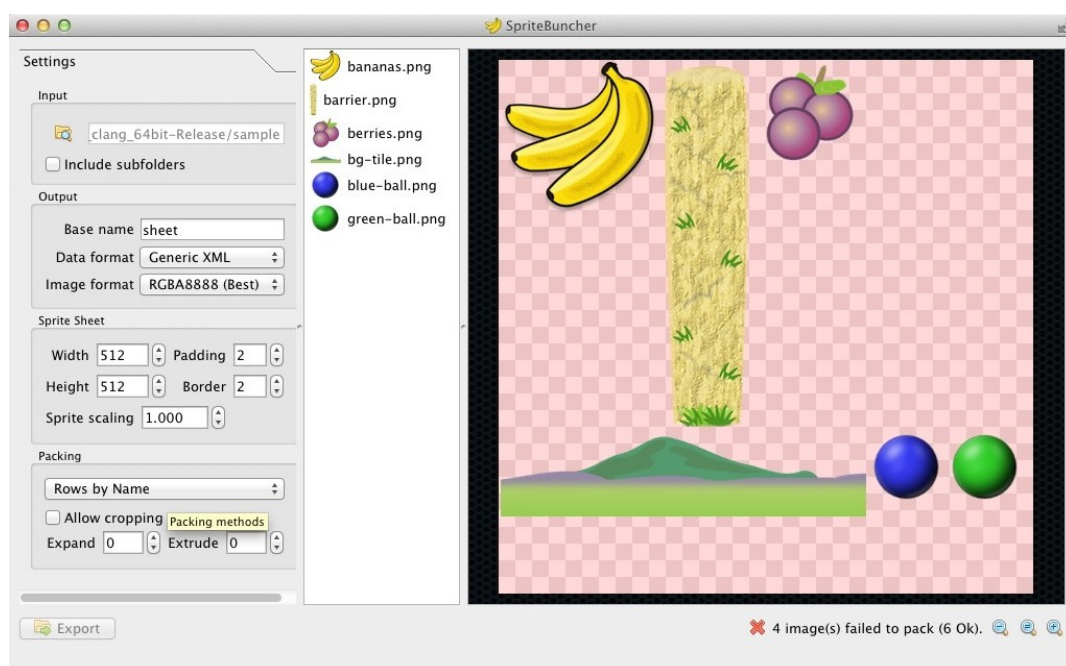


## Packing methods



*SpriteBuncher* currently offers 2 main packing methods, 'MaxRects' and 'Rows', each with a number of variations. *MaxRects* provides the best packing, and hence the most efficient use of available space. The different variations will change the way images are arranged, and may generate slightly better use of space depending on your image shapes and sizes – try each one and see the effect. 'BestArea' is the default.

'Rows' is a much simpler algorithm, where each image is placed along fixed rows, or 'shelves'. The rows have constant height. In some cases, you might prefer this arrangement, and it can be efficient if the images are of similar size. The images are packed in order of *name*, *area*, *height*, or *width*, depending on the option chosen. The sample project can not be packed onto a 512x512 sheet using the rows method, there is too much wasted space, as can be seen below:



*Attempting to pack the sample project using the 'Rows' method.*

## Packing options

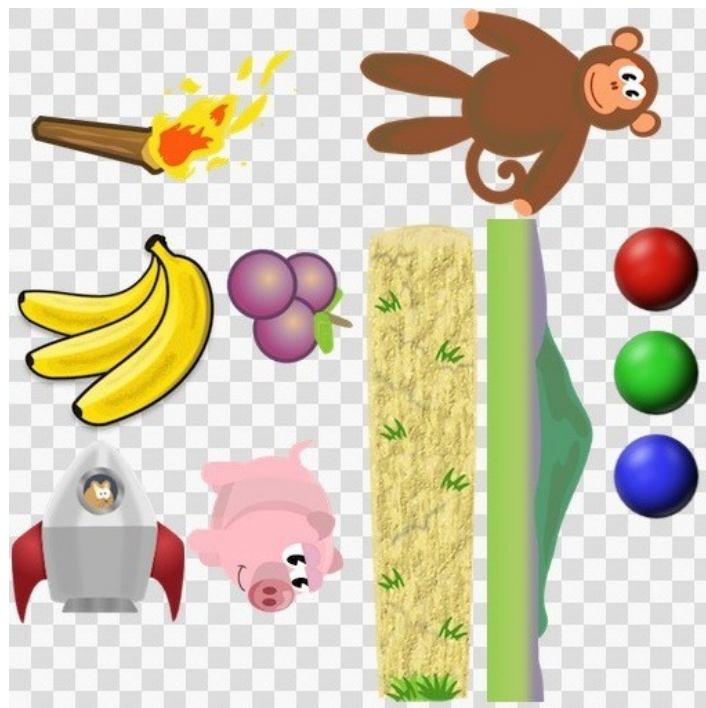
Under the packing method drop-down menu there are a number of further options.

*Allow Cropping* will analyse each sprite and reduce its size on the sheet by removing any transparent areas around the edges. The original files remain unchanged. For example, in the sample project the torch image has a large border which takes up a lot of space on the final sheet (hover over a preview image to display the tooltip information):



*Packing with and without cropping applied. Notice the size change.*

The *Allow Rotation* option (available for MaxRects only) will improve packing by rotating images by 90-degrees where required. *Not all libraries or applications can import this data* – check the documentation first. If the image set contains many wide or tall images this option can improve packing significantly:



*Packing with rotation enabled.*

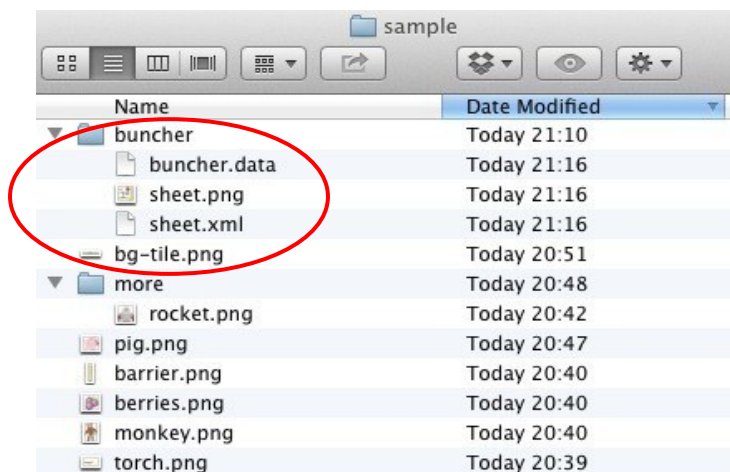


Increasing the value of the *Expand* setting will add the specified number of pixels onto the border around each sprite. This can be useful for adding borders without modifying the original images. It differs from padding in that it adds to the actual recorded size of each sprite, rather than adding space *between* them on the sheet. *Expand* can be used in conjunction with cropping (it is applied after the cropping is performed).

*Extrude* is an advanced option which duplicates the outermost 1-pixel border onto an area beyond the edges of each sprite. This can reduce edge artefacts, since the edges are no longer blended with transparent pixels (e.g. useful for tile sets). It does not add to the sprite size itself. If a sprite already has a transparent border then extrusion will have no visible effect. Extra padding is added automatically to ensure that the extruded area does not overlap with neighbouring sprites.

### Locating and managing output files

After selecting *Export* the application will show a dialog box to confirm that the files have been written. At this point you should be able to locate the new files inside the 'buncher' folder, within the original input folder:



*Output files appear inside the 'buncher' folder (Mac folder shown).*

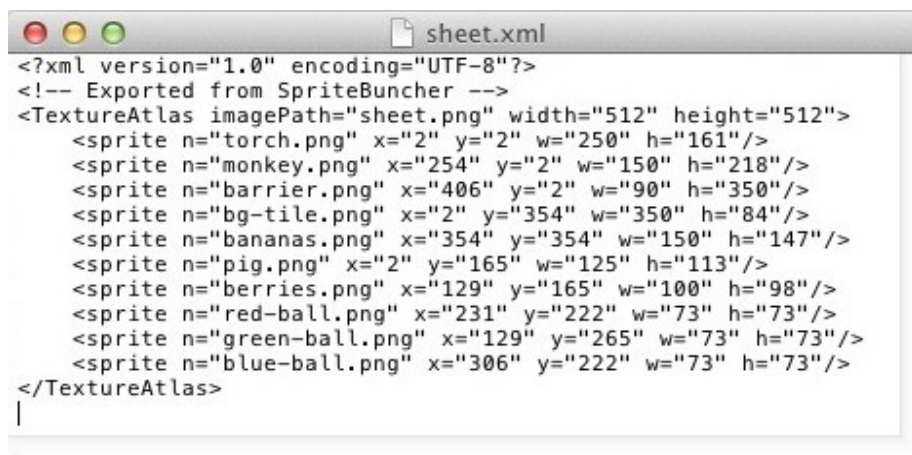
You can now move the image and coordinate files to a new location, should you wish. Note, the original input images are never changed or deleted by *SpriteBuncher*, but the output files are overwritten each time an *Export* is performed.

The `buncher.data` file is used internally by *SpriteBuncher* to store the settings that were used to create the output for this set of images. The file is automatically checked next time the folder is loaded, so the previous settings are retained. (Deleting the `buncher.data`

settings file will not cause a problem, but the program will revert to showing default options for this folder).

Note, the program knows not to include the 'buncher' folder when importing images from subfolders.

It can be useful to check that the output data looks reasonable, based on the settings used. An example of the Generic XML output for the sample project is shown below:



*Sample data output – Generic XML format*

*SpriteBuncher* – a texture packing program.

Copyright © 2014 Barry R Smith.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

*All other trademarks and copyrights are property of their respective owners.*

### **Acknowledgements:**

Qt GUI application framework, available at <http://qt-project.org>.

*MaxRects* - public domain code by Jukka Jylänki, see `maxrects/README.txt`.

Contains icons copyright of FatCow, <http://www.fatcow.com>, licensed under Creative Commons Attribution 3.0 License, <http://creativecommons.org/licenses/by/3.0/us>.