# PNG2ObjV3.py Manual

A Simple program, poorly written to take a PNG Image and convert it to simple 3D and 2D files, for use in your 3D Print and crafting projects.


| | |
|---|---|
| Author: | Jason Brooks |
| Version: | 0.1c |
| Review Date: | TBC |
| Release Date: | TBD |
| Websites: | www.muckypaws.com<br>www.muckypawslabs.com<br>www.github.com/muckypaws |

**1 INTRODUCTION ..... 4**

1.1 WHERE TO FIND THE OFFICIAL CODE REPOSITORY ..... 4
1.2 REQUIREMENTS ..... 4

**2 FEATURES ..... 6**

**3 PARAMETER TABLE ..... 7**

**4 SHARED PARAMETERS ..... 9**

4.1 -OUTFILE OR --OUTFILENAME ..... 9
*4.1.1 Usage ..... 9*
4.2 -LC OR --LISTCOLOURS ..... 9
*4.2.1 Usage ..... 9*
4.3 -EL OR --EXCLUDELIST ..... 11
*4.3.1 Usage ..... 11*
4.4 -PC OR --PROCESSCOLOURS ..... 12
*4.4.1 Usage ..... 12*
*4.4.2 Multicolour Images ..... 12*
4.5 -AC OR --ALPHACUTOFF ..... 13
*4.5.1 Usage ..... 13*

**5 WAVEFRONT OBJ FILE CREATION ..... 14**

5.1 FLAT ..... 14
5.2 TOWERED ..... 15
5.3 LAYERED ..... 16
5.4 DEFAULT OPERATION ..... 17
*5.4.1 Usage ..... 17*
5.5 -M OR --MTL ..... 18
*5.5.1 An example Material File ..... 18*
*5.5.2 Usage ..... 18*
5.6 -NF OR --NOFRAME ..... 19
*5.6.1 Usage ..... 19*
5.7 -DB OR --DEBUG ..... 19
*5.7.1 Usage ..... 19*
5.8 -S, --SORT, -RS AND --REVERSESORT ..... 20
*5.8.1 Unsorted - Default ..... 20*
*5.8.2 Sorted ..... 21*
*5.8.3 Reverse Sort, -rs or --reversesort ..... 22*
5.9 -BGH OR --BACKGROUNDHEIGHT ..... 23
*5.9.1 Usage ..... 23*
5.10 -LM OR --LAYERMULTIPLIER ..... 23
5.11 -NL OR –ONLY ..... 24
*5.11.1 Usage ..... 24*
5.12 -SW, --SPRITEWIDTH, -SH AND --SPRITEHEIGHT ..... 25
*5.12.1 Usage ..... 25*
5.13 -J OR --JOINT ..... 26
*5.13.1 Usage ..... 26*
5.14 -MW, --MAXWIDTH, -MH, --MAXHEIGHT, -MD AND --MAXDEPTH ..... 27
*5.14.1 Usage ..... 27*
5.15 -SZ OR --STARTZ ..... 29
*5.15.1 Usage ..... 29*
5.16 -ILD OR --INITIALLAYERDEPTH ..... 30
*5.16.1 Usage ..... 30*

**6 PARAMETRIC OBJECT GENERATION ..... 31**

**7 SVG FILE CREATION ..... 33**

**8 THE FOLLOWING DESCRIBES THE PROGRAMS SVG SPECIFIC PARAMETERS AND THEIR USAGE. ..... 35**

8.1 Basic Operation ..... 35
8.2 Command Syntax - All defaults ..... 35
*8.2.1 Usage ..... 35*
*8.2.2 Usage ..... 36*
8.3 Parameter Information ..... 36
8.4 -svgpw or --svg_pixel_width ..... 37
*8.4.1 Usage ..... 37*
8.5 -svgph or --svg_pixel_height ..... 37
*8.5.1 Usage ..... 37*
8.6 -svgrpx or --svg_radius_percent_x ..... 38
*8.6.1 Usage ..... 38*
8.7 -svgrpy or --svg_radius_percent_y ..... 39
*8.7.1 Usage ..... 39*
8.8 -illusion or --illusion ..... 40
*8.8.1 Usage ..... 40*
8.9 -f400 or --frame400 ..... 41
*8.9.1 Usage ..... 41*
8.10 -outline or --outlineOnly ..... 42
*8.10.1 Usage ..... 42*
*8.10.2 Usage ..... 42*
*8.10.3 Combined with F400 Parameter ..... 44*
*8.10.4 Usage ..... 44*
8.11 -svgaddpng or --svgaddpng ..... 46
*8.11.1 Usage ..... 46*
*8.11.2 Combined with -illusion or -f400 ..... 47*
8.12 -svgopen ..... 47
*8.12.1 Usage ..... 47*
8.13 -ict or --illusioncolourtable ..... 48
*8.13.1 Usage ..... 48*
8.14 -urc or --userealcolours ..... 49
*8.14.1 Usage ..... 49*
8.15 -ugc or --usegridcolours ..... 49
*8.15.1 Usage ..... 49*
8.16 -mb or --minimumborder ..... 50
*8.16.1 Usage ..... 50*
8.17 -mgw or --minimumgridwidth ..... 51
*8.17.1 Usage ..... 51*
8.18 -mgh or --minimumgridheight ..... 52
*8.18.1 Usage ..... 52*
8.19 -ilc or --illusioncircle ..... 53
*8.19.1 Usage ..... 53*
8.20 -cset or --colourset ..... 54
*8.20.1 Usage ..... 54*
8.21 -stmax or --maxstreak ..... 55
*8.21.1 Usage ..... 55*

# 1 Introduction

Welcome to this simple program I created when needing an easy way to convert PNG files to 3D Waveform OBJ formats or 2D SVG Files for import and manipulation in either CAD, 3D Printing Slicer software or other software supporting the OBJ and SVG file formats.

A few years ago, I was converting images by hand in CAD and quickly discovered the errors that were easily introduced for larger and complex images. I knocked something up that was crude and got the job done, whilst having little knowledge of Python.

Why Python? My thinking was to produce a program that was platform agnostic, and could potentially be used on any device that implemented Python3, as opposed to OS specific.

The programs gotten out of hand, and now performs a number of functions to images in an overly simplistic manner.

That said, the code needs quite some work to refactor from its early incarnation as something quick and simple to something more properly structured and class oriented. If you're a python coder and want to help tidy the code and able to convert to classes, optimise or add features, pull the project and submit a pull request when you're done.

## 1.1 Where to find the official code repository

The official source of my little project can be found here: -

https://github.com/muckypaws/PNG2OBJ

## 1.2 Requirements

Python3 V3.8.5 has been used to develop and test this.

https://www.python.org/downloads/

## 1.3 Installation Instructions

To get started with PNG2ObjV3, follow these steps:

### 1.3.1 Clone the Repository

Open your terminal and run:

```
git clone https://github.com/muckypaws/PNG2OBJ.git
cd PNG2OBJ
```

### 1.3.2 Create a Python Virtual Environment

Use the following command to create an isolated Python environment:

```
python3 -m venv venv
```

### 1.3.3 Activate the Virtual Environment

**macOS/Linux:**

```
source venv/bin/activate
```

**Windows (Command Prompt):**

```
venv\Scripts\activate.bat
```

**Windows (PowerShell):**

```
venv\Scripts\Activate.ps1
```

**Install Required Packages**

With the environment active, install dependencies:

```
pip install -r requirements.txt
```

**Run the Program**

You can now execute the script, for example:

python3 PNG2ObjV3.py -lc your_image.png

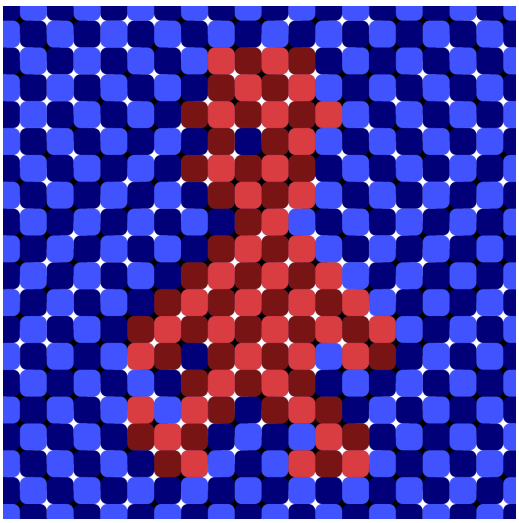
This command lists the colours found in your image and confirms the environment is working correctly.

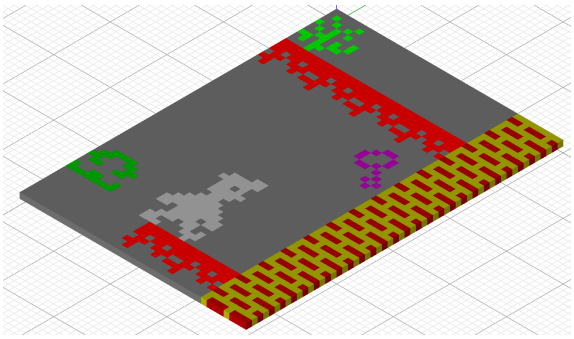
**Tip:**

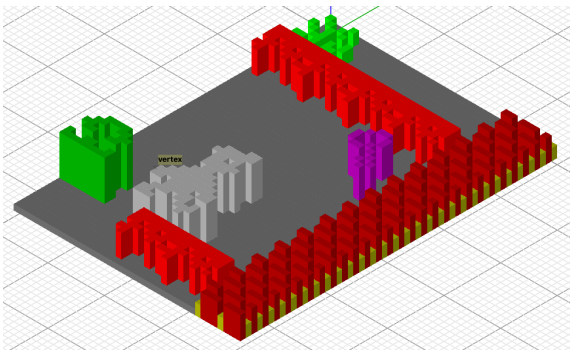
If you plan to create SVG files and want them to open automatically after generation, make sure your system has a default application associated with .svg files. This can be software like Affinity Designer, Adobe Illustrator, Inkscape, or a modern web browser.

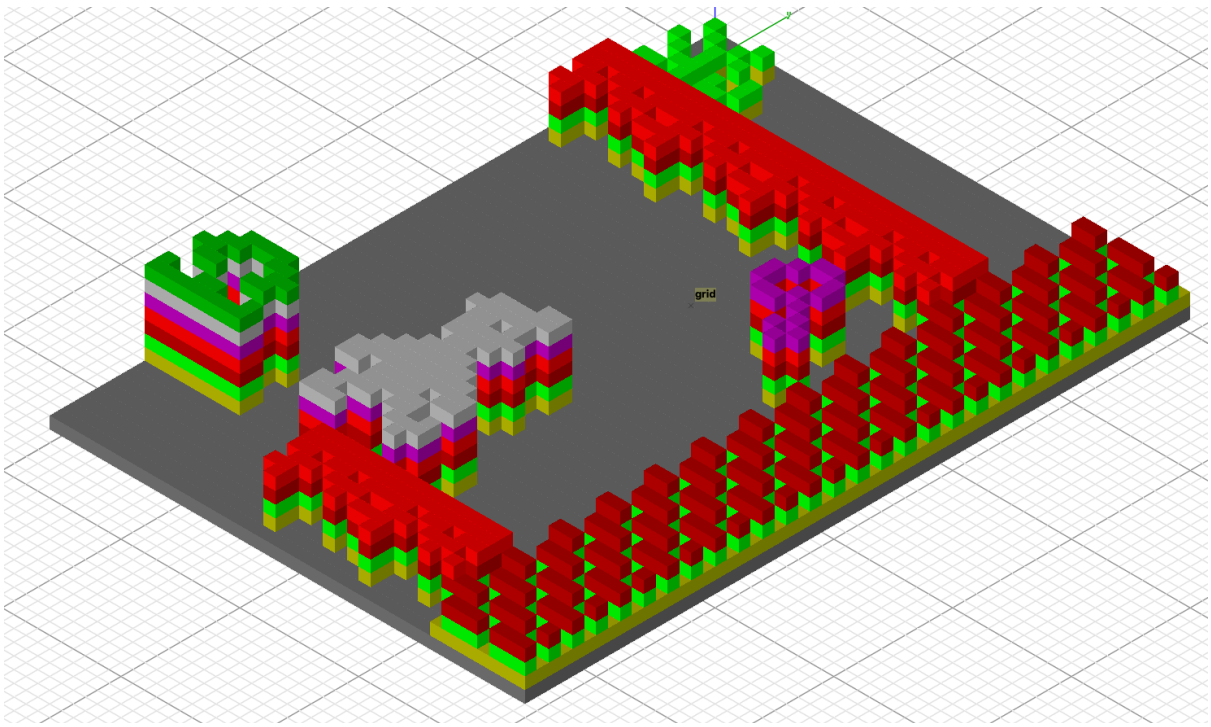
# 2 Features

The program started life as a simple PNG to 3D OBJ converter, though recently I've been playing with creating 2D optical illusions and added some limited SVG creation functionality too.





# 3 Parameter Table

| Wavefront OBJ File Creation | | | | |
|---|---|---|---|---|
| **Short** | **Long** | **2D** | **3D** | **Description** |
| -outfile | --outfilename | ✅ | ✅ | Output Filename for processed file |
| -m | --mtl | | ✅ | Create the Material File .MTL Associated with the OBJ File. |
| -lc | --listColours | ✅ | ✅ | List all the colours found in the PNG Image |
| -el | --excludelist | ✅ | ✅ | Add a list of colours to exclude from the processing of the PNG Image. |
| -ac | --alphacutoff | ✅ | ✅ | Set the Alpha Byte Cut off if present (0-255) 0 = Transparent, 255 = Fully Opaque. |
| -db | --debug | | ✅ | Create a minimal ASCII File used to help debugging issues, not for general use. |
| -s | --sort | | ✅ | Sort the colours found in the image in order of lowest pixel colour count to highest. |
| -rs | --reversesort | | ✅ | Sort the colours found in the image in reverse order, highest colour pixel count to lowest. |
| -pc | --processcolours | ✅ | ✅ | A list of colours to process from the Image. |
| -bgh | --backgroundheight | | ✅ | |
| -lm | --layermultiplier | | ✅ | Largely redundant now, used to increase/decrease the height of each layer. |
| -nl | --nextlayeronly | | ✅ | Process all colours requested with Z at the same position for all colours, though all colour objects will be grouped and in separate files. |
| -sw | --spriteWidth | | ✅ | When processing PNG Images with Multiple Sprites in multiples of x Pixels. |
| -sh | --spriteHeight | | ✅ | When processing PNG Images with Multiple Sprites in multiples of x Pixels. |
| -j | --joint | | ✅ | Adds a joint to stranded pixels for 3D Printing |
| -mw | --maxwidth | | ✅ | Set the Maximum Width in millimetres for the OBJ file to be created. |
| -mh | --maxheight | | ✅ | Set the Maximum Height in millimetres for the OBJ file to be created. |
| -md | --maxdepth | | ✅ | Set the Maximum Deptg in millimetres for the OBJ file to be created. |
| -ild | --initialLayerDepth | | ✅ | Set the depth of the first layer in mm to be created. |
| -nf | --noframe | | ✅ | Don't generate a bounding frame. |
| -sz | --startZ | | ✅ | Initial Z Height starting position. |

## Mutually Exclusive Flags (Only One Can be Enabled)

| | | | | |
|---|---|---|---|---|
| -fl | --flat | | ✅ | Create a Single OBJ File with all colour information. |
| | --layered | | ✅ | Create a layered OBJ File with one colour per layer. |
| | --tower | | ✅ | Create an OBJ File with colours at different heights based on processing order. |

## Parametric Tools – Coming Soon

| | | | | |
|---|---|---|---|---|
| -pt | --parametricTest | | ✅ | Experimental don't use. |
| -ptn | --parametricFilename | | ✅ | Experimental still under development, don't use. |

## SVG Specific Parameters

| Short | Long | 2D | 3D | Description |
|---|---|---|---|---|
| -svg | --svg | ✅ | | Create an SVG File. |
| -svgpw | --svg_pixel_width | ✅ | | Request width of each pixel in millimetres. |
| -svgph | --svg_pixel_height | ✅ | | Request height of each pixel in millimetres. |
| -svgrpx | --svg_radius_percent_x | ✅ | | The corner radius of each rectangle in percent (100% = Semi Circle) -- X Axis. |
| -svgrpy | --svg_radius_percent_y | ✅ | | The corner radius of each rectangle in percent (100% = Semi Circle) -- Y Axis. |
| -illusion | --illusion | ✅ | | Create the Optical Illusion Effect. |
| -f400 | --frame400 | ✅ | | Create an SVG Template for Cutting machines. |
| -outline | --outlineOnly | ✅ | | Provide an Outline only for guides when creating. |
| -svgaddpng | --svgaddpng | ✅ | | Add the PNG to the illusion output. Only needed when creating optical illusions. |
| -svgopen | | ✅ | | Try to open the created file once written to disk using the systems default application. |
| -ict | --illusioncolourtable | ✅ | | Create a Table for the colours used in the illusion. |
| -mb | --minimumborder | ✅ | | Add a minimum border to the PNG Image in pixels. |
| -mgw | --minimumgridwidth | ✅ | | Set the minimum grid Width for the SVG File. |
| -mgh | --minimumgridheight | ✅ | | Set the minimum grid Height for the SVG File. |
| -ilc | --illusioncircle | ✅ | | Default is the create the illusion with diagonal lines, use this flag to create circular negative space. |
| -cset | --colourset | ✅ | | Select one of the predefined colour sets for the Optical Illusion. |
| -stmax | --maxstreak | ✅ | | The maximum streak of negative/positive space pixels in the optical illusion -- default 3. |

## Mutually Exclusive Parameters -- Select only One.

| | | | | |
|---|---|---|---|---|
| -urc | --userealcolours | ✅ | | Use the PNG Real Colours in the Optical Illusion. |
| -ugc | --usegridcolours | ✅ | | Use the grid colours in place of the PNG Image |

# 4 Shared Parameters

The following parameters are common to both 2D and 3D file creation tools.

## 4.1 -outfile or --outfilename

By default the program will create a filename the same as the PNG Image with an appropriate extension (I.e. OBJ, MTL or SVG)

```
./PNG2ObjV3.py JSWStrideSingle.png
./PNG2ObjV3.py -svg -f400 -outline -svgaddpng JSWStrideSingle.png
```

Will create file JSWStrideSingle.obj or JSWStrideSingle.svg in the same folder/directory the PNG Image is located.

This may not always be desirable, you can specify a different filename using the **-outfile** parameter

### 4.1.1 Usage

```
./PNG2ObjV3.py JSWStrideSingle.png -outfile "./Test/Mytest"
./PNG2ObjV3.py -svg -f400 -outline -svgaddpng JSWStrideSingle.png -outfile "./Test/MySVGTest"
```

## 4.2 -lc or --listColours

Quite simply, this parameter provides basic information about the PNG image loaded and the Colours detected.

Not particularly useful if you have a fully photographic image, as this program is really all about processing few colour images like retro computer screens.

The output enumerates the Hex Colour of the Pixels found with a count. The order will be presented in reverse order, Most number of colour pixels to the lowest.

### 4.2.1 Usage

```
./PNG2ObjV3.py -lc JSWStrideSingle.png

2024-09-22 20:05:04.443583: Loaded PNG file: JSWStrideSingle.png

    Image Information :
    -------------------

           Image File : JSWStrideSingle
       PNG Image Size : 10px (width), 16px (height)
   Number of Channels : 4

     Joints Requested : False
 Create Material File : False
 Reverse Sort Colours : False
Create Flat File Only : True
         Alpha Cutoff : 128
Background Multiplier : 1.00
     Layer Multiplier : 1.00
       Object Start Z : 0.00mm
 Object Max Depth (Z) : 0.00mm
     Each Layer Depth : 1.00mm
     Background Frame : True

  Pixel Width/Height : 0.01mm x 0.01mm


         Layer Order :
         -------------
```

```
        Colour Code : Number of Pixels
        -----------   ----------------

            #bdbdbd : 85
            #000000 : 75

              Found : 2 Colours
```

Using the Multicolour PNG Image PlatformWilly.png, we find the following: -

```
./PNG2ObjV3.py -lc PlatformWilly.png

2024-09-22 20:23:56.387464: Loaded PNG file: PlatformWilly.png

    Image Information :
    -------------------

           Image File : PlatformWilly
      PNG Image Size : 40px (width), 56px (height)
   Number of Channels : 4

     Joints Requested : False
 Create Material File : False
 Reverse Sort Colours : False
Create Flat File Only : True
         Alpha Cutoff : 128
Background Multiplier : 1.00
     Layer Multiplier : 1.00
       Object Start Z : 0.00mm
 Object Max Depth (Z) : 0.00mm
     Each Layer Depth : 1.00mm
     Background Frame : True

  Pixel Width/Height : 0.01mm x 0.01mm


        Layer Order :
        -------------

        Colour Code : Number of Pixels
        -----------   ----------------

            #000000 : 1452
            #bdbf00 : 280
            #ff0000 : 192
            #bf0000 : 168
            #bdbdbd : 82
            #00bf00 : 28
            #00ff00 : 23
            #bf00bf : 15
              Found : 8 Colours
```

## 4.3 -el or --excludeList

When processing an Image, you may wish to exclude a series of colours. For example, if a PNG of sprites has Black (#000000) as the background colour you may want to remove this from you final processed file.

In addition, you can exclude a number of colours

### 4.3.1 Usage

```
./PNG2ObjV3.py JSWStrideSingle.png -el #000000
```

You'll notice that I've included the filename first in this command. Why?

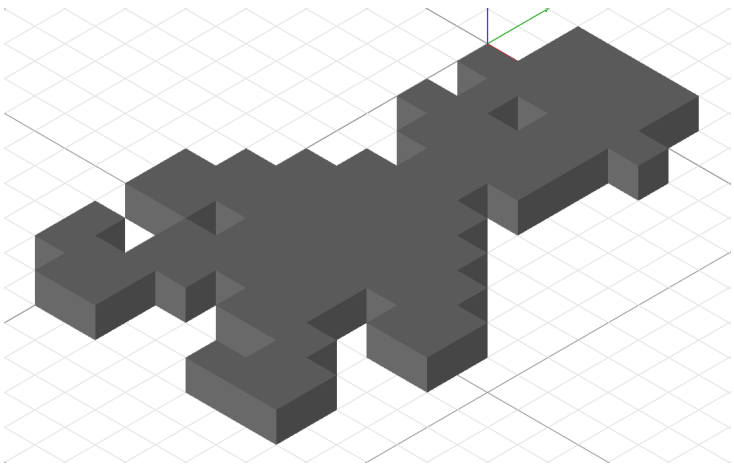
The command

```
./PNG2ObjV3.py -el #000000 JSWStrideSingle.png
```

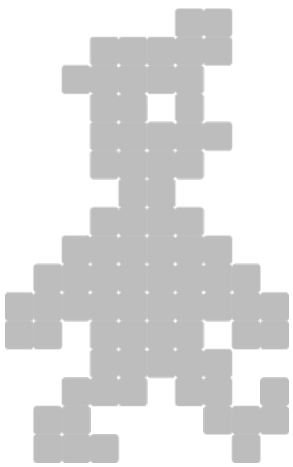
Results in an error, simply because Python interprets the filename as part of the **-EL** Parameter list. Its usually good practice to add the filename directly after the command.

Preferred examples :-

```
./PNG2ObjV3.py JSWStrideSingle.png -el #000000

./PNG2ObjV3.py PlatformWilly.png -el #000000 #bdbf00 #ff0000 #bf0000 #00bf00 #00ff00 #bf00bf

./PNG2ObjV3.py -el #000000 #bdbf00 #ff0000 #bf0000 #00bf00 #00ff00 #bf00bf -svg PlatformWilly.png
```



**3D OBJ**



**SVG**

## 4.4 -pc or --processColours

If we have a complex image with multiple colours, instead of defining all the colours to exclude, we can set the colours to include or **Process Only.**

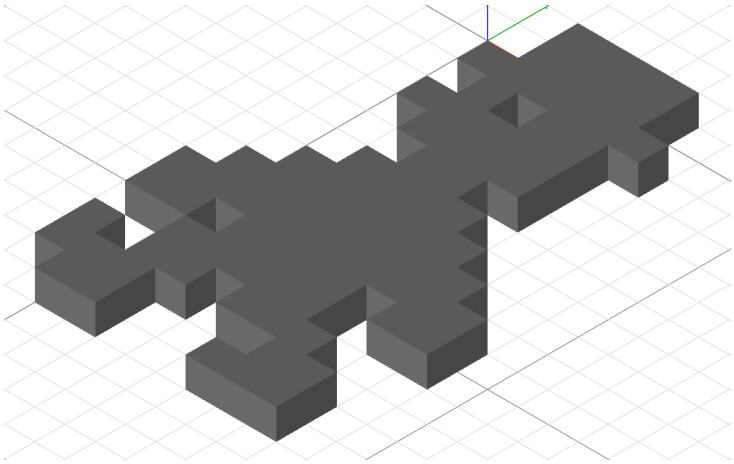
### 4.4.1 Usage

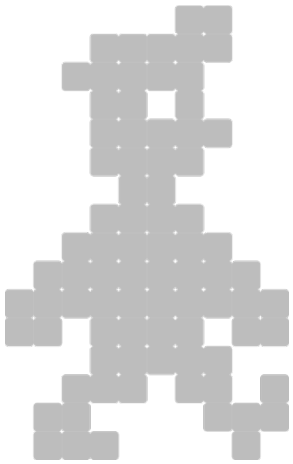
```
./PNG2ObjV3.py JSWStrideSingle.png -pc #bdbdbd
```

You'll notice that I've included the filename first in this command.  Why?

```
./PNG2ObjV3.py JSWStrideSingle.png -pc #bdbdbd

./PNG2ObjV3.py PlatformWilly.png -pc #bdbdbd

./PNG2ObjV3.py -pc #bdbdbd -svg PlatformWilly.png
```

Results in an error, simply because Python interprets the filename as part of the **-PC** Parameter list.





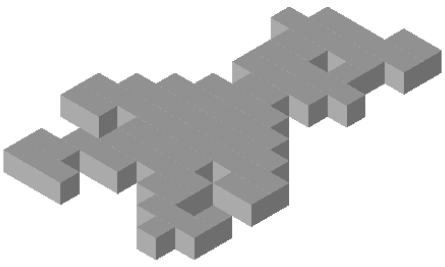
### 4.4.2 Multicolour Images

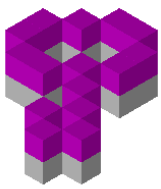
Using the PlatformWilly.png image, what if we only want to include Willy and the collectible key pixels?

We know Willy is off-white colour #bdbdbd and the Magenta coloured key is #bf00bf

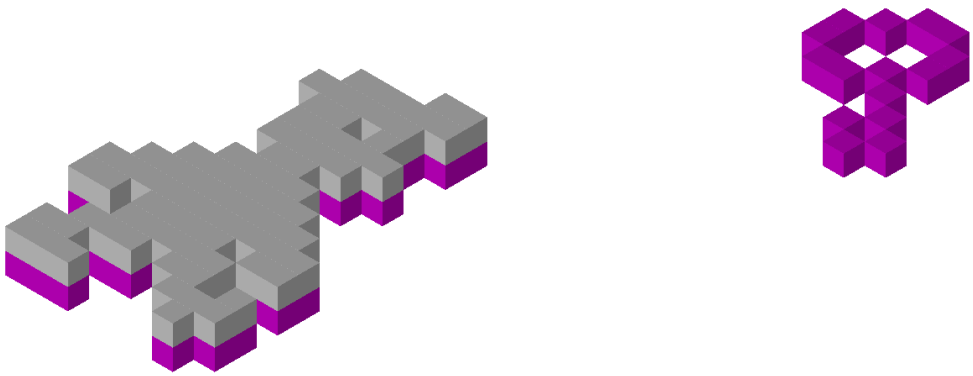
We could provide an exclusion list, for the remaining six colours, or we use a process colour list.

```
./PNG2ObjV3.py PlatformWilly.png -pc #bdbdbd #bf00bf -layered --mtl
```





```
./PNG2ObjV3.py PlatformWilly.png -pc #bf00bf #bdbdbd --layered --mtl
```



The image will be processed in the order of the colours, which for layered objects is important to get the best effect. In this case willy is twice as deep than the collectible key.

## 4.5 -ac or --alphaCutoff

Many PNG Images have a fourth channel (Alpha Channel) present, which describes the opacity of a pixel in an image. The value of which is 0 (Fully Transparent) to 255 (Fully Opaque) and in-between.

Although a pixel is transparent it still contains colour information, even if it's Black #000000. We don't want to include this in processing.

Default: 254

By default, level 254 is set, thereby only selecting fully opaque pixels when processing image information. However if you wish to change the threshold, use this parameter.

### 4.5.1 Usage

```
./PNG2ObjV3.py -pc #bdbdbd -ac 128 PlatformWilly.png
```

Will select and include pixels that have an Alpha Channel byte of 128 and above.

# 5 Wavefront OBJ File Creation

When I started writing this program, the main goal was to reduce the error from creating 3D Models in CAD by hand. For anything but the simplest of sprites, resulted in errors, usually picked up after 3D Printing.

I knocked something up quickly. It was crude but worked for my needs at the time. New functionality has been added over a period of time.

The key issue is the initial program was kinda flawed in how I implemented the Wavefront Object data, defaulting to 10mm x 10mm x 10mm per pixel.

This is an exercise in poor design, the program grew, the code is messy. It needs to be rewritten using Python Classes, and optimised using Python tricks. Which may happen when time permits. Though being made available on my GitHub repo, hopefully someone with better Python Skills can help.

What would also be awesome is if anyone can add a front end to this, either HTML or Python GUI to make this more user friendly…

A guy can dream right?

The downside to the OBJ file format is there's no default or descriptor for measurement units, and so an arbitrary unit is used to provide scale on various parameters. When importing models into your slicing software, you may need to specify millimetres or inches depending on your preference. Throughout I will refer to measurements in millimetres.

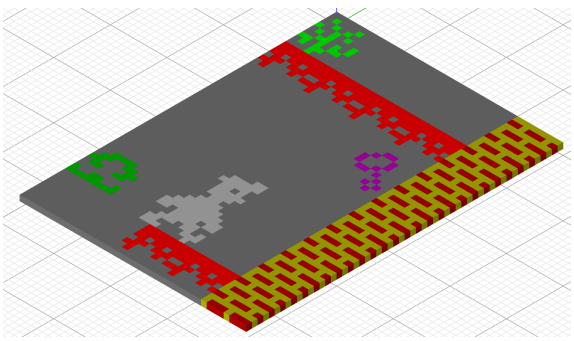
The program will handle multi-colours which can result in three different types of 3D Objects.

**Flat, Layered or Towered.**

Essentially

## 5.1 FLAT

Flat OBJ files contain all pixels and colour information in one file with all pixels/colours being the same level/height.

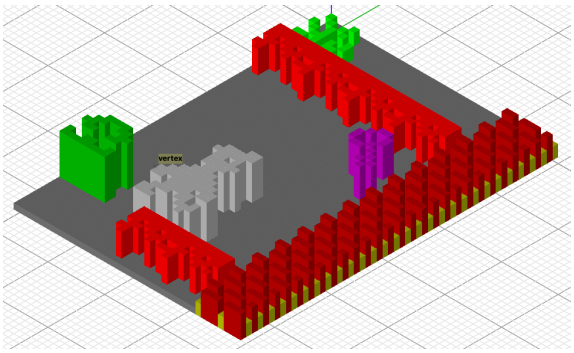


## 5.2 Towered

Towered OBJ files contain all pixels and colour information in one file, pixels of differing colour will have different height information.

This results in improved 3D Renderings giving basic depth, though when 3D colour printing will result in the most amount of filament swaps and increase production costs.

The height of each colour can be ordered to suit the scene, though you may need to use some editing tricks in photoshop or equivalent to get the best out of this feature.

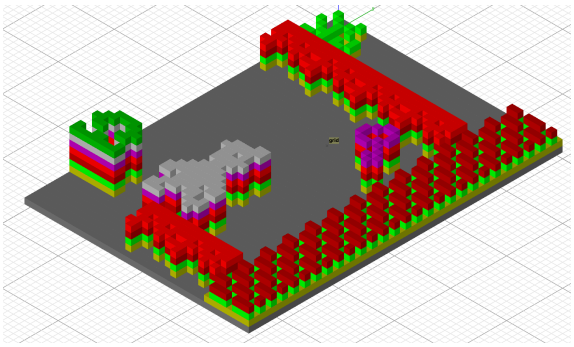



**Example of a Tower Style 3D Print**

This style resulted in nearly 200 filament changes over 5mm height. Produces higher quality colour models, at the expense of longer print times and increased filament wastage.

## 5.3 Layered

Layered OBJ files contain all pixels and colour information in multiple files. The colour order is decided at processing time. Each lower layer supports a higher layer, which is useful for where you wish to save production costs and quality isn't necessarily needed to be high. This results in filament swaps for each colour change, resulting in lower wastage and costs, and can easily be used on single filament printers where you can swap filament out for each colour change.





**Examples of Layered 3D Print**

## 5.4 Default Operation

By default the program will generate a Flat 3D Single File without any colour information.

### 5.4.1 Usage

```
./PNG2ObjV3.py ./JSWStrideSingle.png
```

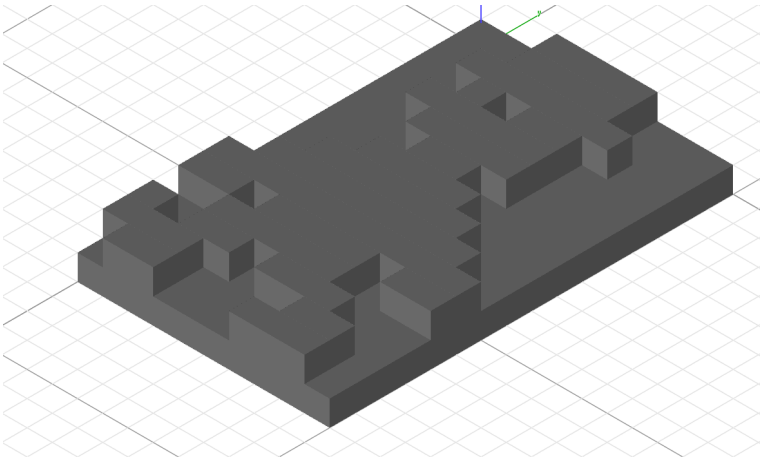
Results in two files created

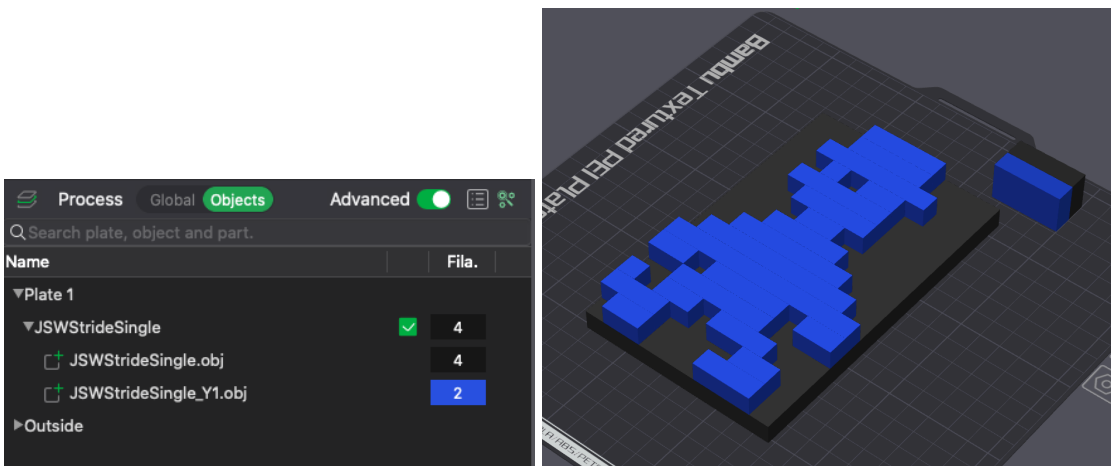
```
JSWStrideSingle.obj
JSWStrideSingle_Y1.obj
```

**JSWStrideSingle.obj** is a simple 3D Rectangle that forms the overall length and width of the PNG Object Processed and van be used as a simple base, frame or background.

**JSWStrideSingle_Y1.obj** contains the 3D model from all the PNG colour information discovered when processing the image. I've deliberately used the PNG with Opacity enabled in this example to ensure we get our game character.

This is the simplest of designs, you can for example import both output files into your slicer, setting each object to appropriate colours, such as colour black for the background and the Y1 Object to a colour of your choosing.






**Imported into Slicer Software**

## 5.5 -m or --mtl

Creating the OBJ File is the first part, though to help you visualise this in other software, you will probably want to add a material file. This file describes each of the Objects colours, each of the OBJ files created will reference this file to help an external program render the object correctly on screen.

In addition, some slicers now will try to colour match the material data with filaments loaded in your printer.

### 5.5.1 An example Material File

```
# Created with PNG2ObjV3.PY (C) Jason Brooks
# See www.muckypaws.com and www.muckypawslabs.com
# https://github.com/muckypaws/PNG2OBJ


newmtl 0
Kd 0.741176 0.741176 0.741176
Ks 0.000000 0.000000 0.000000
Tf 0.000000 0.000000 0.000000
illum 4
d 1.0
Ns 100
sharpness 100
Ni 1

#
# Total Material Colours Created: 1
#
```

### 5.5.2 Usage

```
./PNG2ObjV3.py ./JSWStrideSingle.png --mtl
```

Results in three files being created, the background and the PNG Object.

```
JSWStrideSingle.mtl
JSWStrideSingle.obj
JSWStrideSingle_Y1.obj
```

Of course the more colour information contained within the PNG image, the more material information is provided in the file.

```
./PNG2ObjV3.py PlatformWilly.png -outfile TestMultiColours --mtl
```

Creates a material file with eight entries. Experiment with PNG Images and check the files created out.

## 5.6 -nf or --noframe

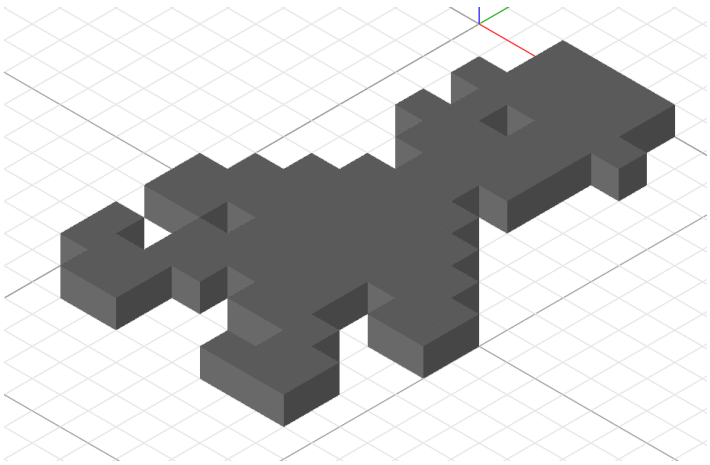
By default, the program adds an OBJ file with a simple bounding rectangle covering the entire width and height of the PNG image processed. The purpose is to be used as a background or place holder to layer additional objects on top of.

Essentially acting as a backing board, especially for stranded pixels that didn't have anything to hold on to.

This may not always be desirable, as we can see Jet Set willy is capable of being printed on his own without additional support. So you can either ignore/delete the frame file or simply not created it by adding the **-nf** option.

### 5.6.1 Usage

```
./PNG2ObjV3.py JSWStrideSingle.png -outfile Test -nf
```



## 5.7 -db or --debug

This option is for internal debugging/development purposes.

Whilst processing the PNG image a TXT file is created with a map of which pixels were captured/processed by the program, and behaves differently depending on the options selected.

```
   ****
   ****
  ******
   * **
  *****
   ****
    **
   ****
  ******
 ********
**********
** **** **
  *****
* ** ***
***    **
 **   ***
```

### 5.7.1 Usage

```
./PNG2ObjV3.py JSWStrideSingle.png -outfile Test -nf -db
```

## 5.8 -s, --sort, -rs and --reversesort

This is useful when using the --layered or --Towered options. By default, the program creates a processing colour order list based on when each colour was identified in the image, working top left to top right and down through the image file.

When requesting a Flat file, this is largely unimportant.

However, when requesting a layered file, in general the most of a colour will be at the bottom of the print with the least number of colours at the top, potentially saving on printing costs.

If we take the image PlatformWilly.png, the program finds the colours in this order.

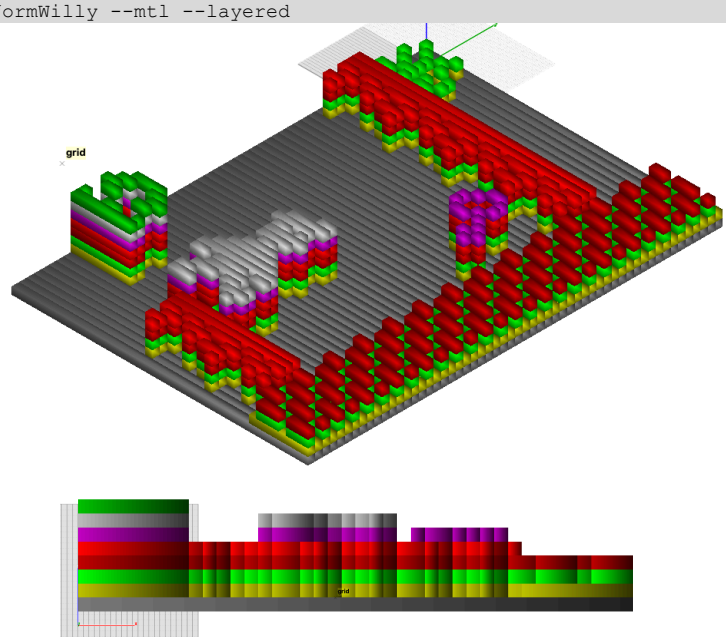
```
         Colour Code : Number of Pixels
         -----------   ----------------

             #000000 : 1452
             #bdbf00 : 280
             #ff0000 : 192
             #bf0000 : 168
             #bdbdbd : 82
             #00bf00 : 28
             #00ff00 : 23
             #bf00bf : 15

               Found : 8 Colours
Layer Order:
#000000
#bdbf00
#00ff00
#bf0000
#ff0000
#bf00bf
#bdbdbd
#00bf00
```

The layer order appears in the order each colour was identified.

### 5.8.1 Unsorted - Default

Using the Unsorted Option worked better than expected, since the greatest number of pixels (Black) was detected first.

```
./PNG2ObjV3.py PlatformWilly --mtl --layered
```



The next colour Yellow #bdbf00 at the top edge of the wall was detected (The mortar for the bricks), though the Red Pixels #bf0000 are on Layer 4, which gives a distorted view of the bricks (Less mortar).

The previous layer of colour is printed to support the upper layer until the image/scene is complete. For an image with lots of colour, this becomes unsustainable.

### 5.8.2 Sorted

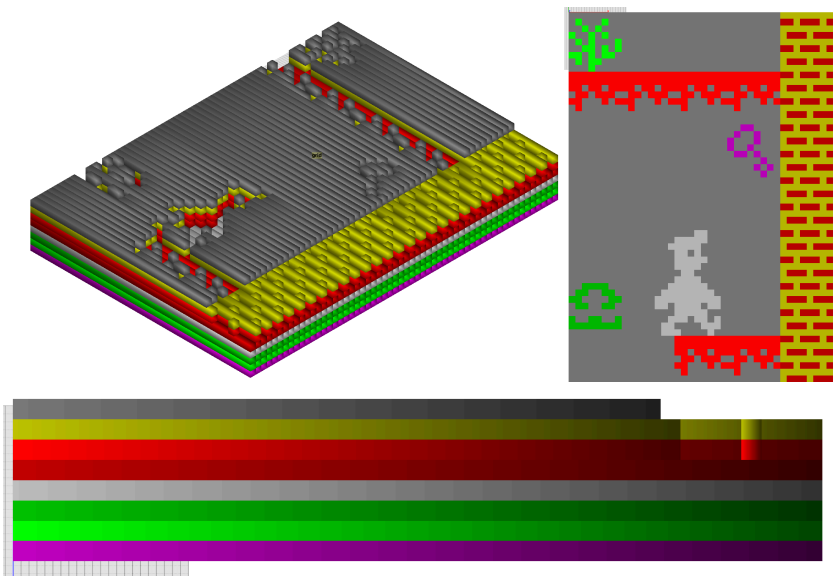
To demostrate this, we'll sort the colours, **-s** sorts in lowest to highest order first

```
./PNG2ObjV3.py PlatformWilly --mtl -layered -s
```

```
Layer Order:
#bf00bf
#00ff00
#00bf00
#bdbdbd
#bf0000
#ff0000
#bdbf00
#000000
```

Based on the pixel count above we get the order list above.

Which results in this 3D Object…



**Why?**

Black is the last colour to process (Highest number of pixels) which happened to be the last layer. As a result, all previous layer colours had to be printed to ensure Black had support on the top.

### 5.8.3 Reverse Sort, -rs or --reversesort

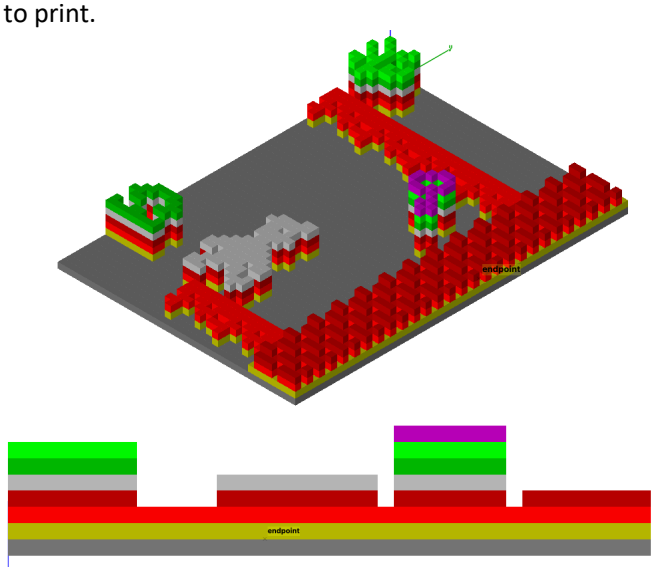
What if we reverse sort?

```
./PNG2ObjV3.py PlatformWilly --mtl -layered -rs
```

Results in the following order, most number of coloured pixels to the least.

```
Layer Order:
#000000
#bdbf00
#ff0000
#bf0000
#bdbdbd
#00bf00
#00ff00
#bf00bf
```

This time taking the greatest number of pixels to layer first building the image upwards, we see an improved set of objects to print.




However, I would probably recommend sticking to lower colour images to use the layering technique, i.e. Amstrad CPC Mode 1 (Three colour mode). Like the example below.

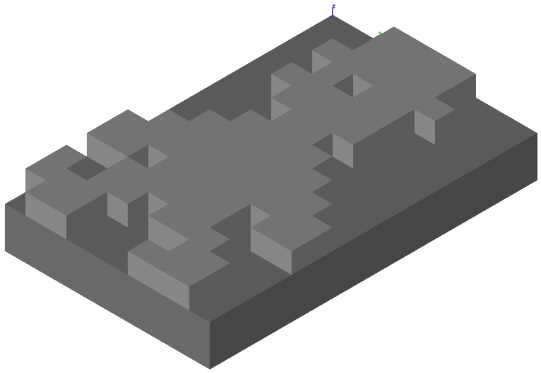

You can argue that this would probably look ok printed. With a creative approach, you can improve this 3D Print, by choosing the order of colours, check out the parameter **-ProcessColours or -PC**

## 5.9 -bgh or --backgroundheight

This is option may be removed in a future release as it may now be rather redundant. The goal was originally to determine a multiplier to the frame/background object created for building a final OBJ file.

### 5.9.1 Usage

```
./PNG2ObjV3.py JSWStrideSingle --mtl -layered -bgh 2.0
```



**Combined Object with Background**

**Side View of Object**

The example shows the background Object is now twice as high as the previously set. This can now have finer controller over using the **--InitialLayerDepth** parameter.

## 5.10 -lm or --layermultiplier

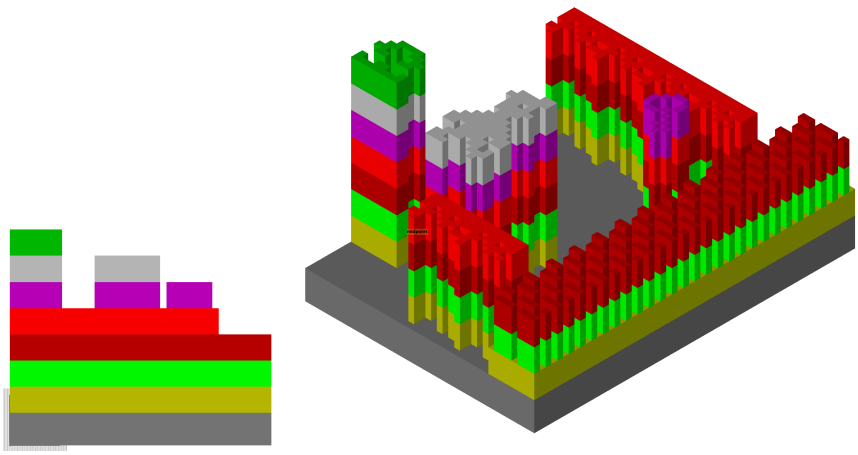
This parameter has been superseded by **--maximumdepth** parameter as this is more useful.ef

This parameter acts as a layer multiplier for each layer of colour information processed.

**Usage**

```
./PNG2ObjV3.py PlatformWilly --mtl -layered -lm 4.0
```

Results in creating layers 4 x the height of the default layer height.



## 5.11 -nl or –only

This parameter processes each colour in turn when combined with –layered creating individual OBJ Files that when combined produce a Flat image as below.

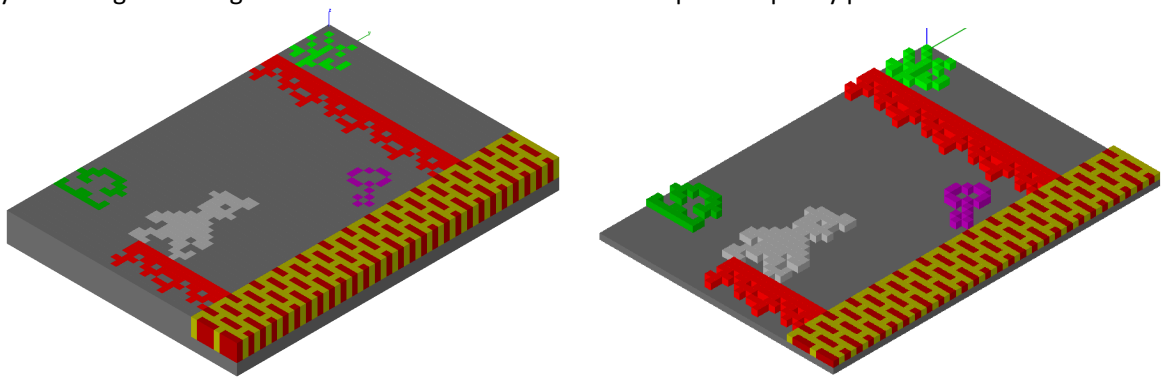
| Filename | Description |
|---|---|
| PlatformWilly.mtl | Material File (Colour Information) |
| PlatformWilly.obj | Background/Baseplate |
| PlatformWilly_Y1#000000.obj | OBJ File for colour Black Pixels |
| PlatformWilly_Y1#00bf00.obj | OBJ File for colour Dark Green Pixels |
| PlatformWilly_Y1#00ff00.obj | OBJ File for colour Green Pixels |
| PlatformWilly_Y1#bdbdbd.obj | OBJ File for colour Off White Pixels |
| PlatformWilly_Y1#bdbf00.obj | OBJ File for colour Off Yellow Pixels |
| PlatformWilly_Y1#bf0000.obj | OBJ File for colour Dark Red Pixels |
| PlatformWilly_Y1#bf00bf.obj | OBJ File for colour Magenta Pixels |
| PlatformWilly_Y1#ff0000.obj | OBJ File for colour Red Pixels |

### 5.11.1 Usage

```
./PNG2ObjV3.py PlatformWilly --mtl -nl --layered
./PNG2ObjV3.py PlatformWilly --mtl -nl --layered -el #000000
```

Why not use the default **--flat?** Flat will do the job, though when you load it up into your slicer you'll need to find and paint each pixel individually. This method produces a flat image as shown but with all the colour information your slicer needs to match filament on multi colour print machines.

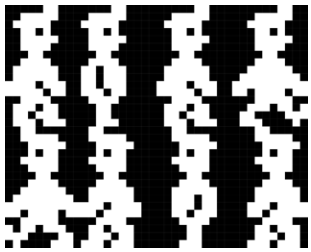
By excluding the background colour #000000 results in an improved quality print too.



## 5.12 -sw, --spritewidth, -sh and --spriteheight

This when used correctly is a power tool. Developers love sprite sheets for game development and usually aligned on byte boundaries. Though for this example I've deliberately aligned Miner Willy on 10 pixels width by 16 pixels height boundaries to illustrate the commands use.

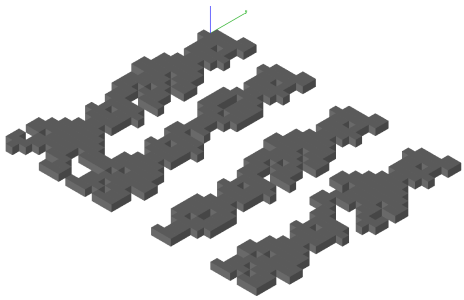
The file **SpriteSheet.png** looks something like this: -



If we use our standard command to convert this to 3D

```
./PNG2ObjV3.py SpriteSheet.png -el #000000
```

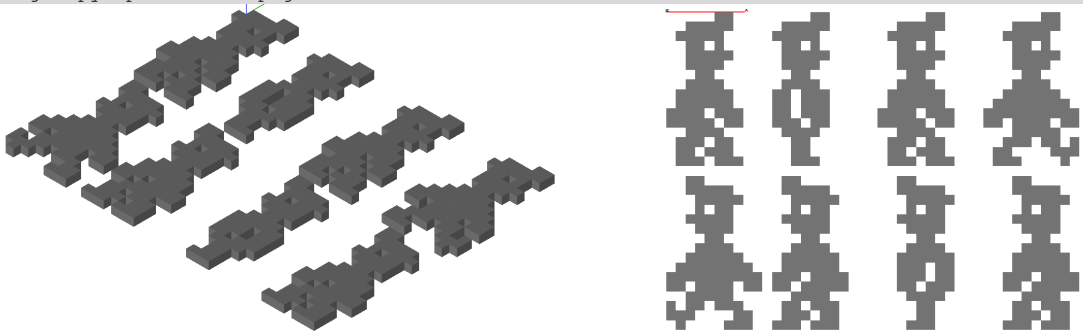
We end up with the following, resulting in each Miner Willy being fused into the next where pixels from one frame join the other. That's fine for a scene you don't want to change, but… using the power of the Sprite Width command, we can set a boundary for each animation frame.



We know each animation is 10 pixels wide, and 16 pixels high.

### 5.12.1 Usage

```
./PNG2ObjV3.py SpriteSheet.png -el #000000 -sw 10 -sh 16
```



How cool is that? Each animation frame has been separated out, meaning in your slicer you can split the objects simply, or load into TinkerCad or other CAD Programs for further processing. For this to work, all sprites on the sprite sheet must be aligned to the same dimensions X by Y.

## 5.13 -j or --joint

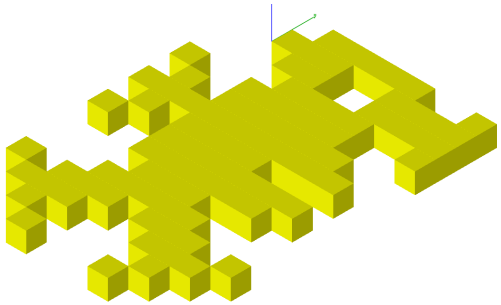
Some images/sprites have a dithered effect, and if you're planning on printing them, it's highly likely they'll break easily.

Lets take **Phil.png**, the name I've given the mining robot who guards the central cavern.



Our cute guardian converts to 3D as simply as other examples

```
./PNG2ObjV3.py Phil.png -el #000000
```

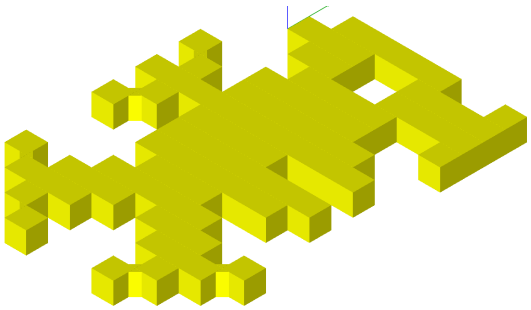


And he looks great, will work well in 3D Software, games etc. Printing Phil, will result in very weak points on the clockwork key and his feet since there are diagonal pixels that have little to no support.

In testing a few years back I'd add support joints to print him. The program will do this automatically with the **--joints** parameter where these are needed.

### 5.13.1 Usage

```
./PNG2ObjV3.py Phil.png -el #000000 -mtl --joints
```





You can see from the results the model makes for a more stable 3D Print.

## 5.14 -mw, --maxwidth, -mh, --maxheight, -md and --maxdepth

These parameters are quite powerful when designing prints to a specific size. The Wavefront OBJ File format doesn't have a descriptor for units, i.e., Inches, millimetres etc. hence the values you plug in here are arbitrary though this value will default to your slicer settings. In my case millimetres.

I tend to refer to these in milimeters.

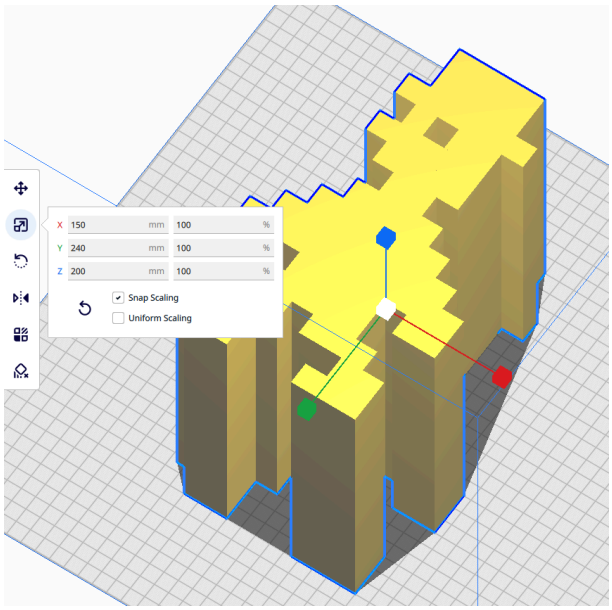
The values passed will size your final object to meet the maximum dimension you request, and will be used in combination with **--InitialLayerDepth** parameter.

Lets Take WillyStrideSingle.png, currently when processed the size is approx. 10 x 16 mm when loaded into your slicer.

For example, you want to print the model to the maximum size of your 3D Printers bed, which is say, 240mm width(X) x 240mm depth(Y) x 200mm height (Z). Now that's a huge Willy… (Sorry folks I tried to restrain myself).

### 5.14.1 Usage

```
./PNG2ObjV3.py JSWStrideSingle.png -mw 240.0 -mh 240.0 -md 200.0
```



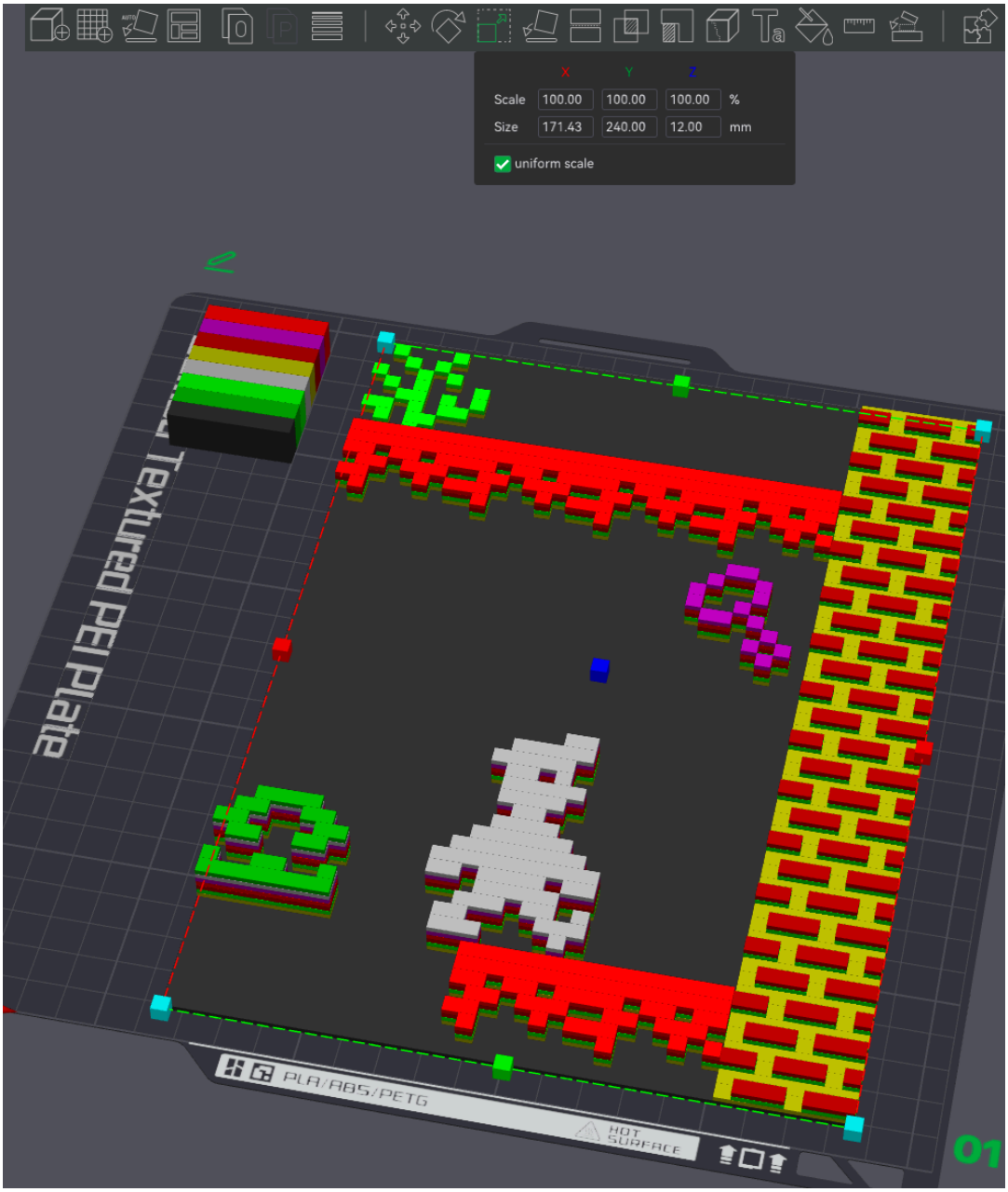

Ok, cool but we could easily have resized this in the slicing software? For single colour simple models, sure I agree, it's a personal choice.

When you have a more complex scene such as **PlatformWilly.png** additional effort may be required to resize each layer of the model. This is useful when processing multiple images in batch or shell script.

Using the following command: -

```
./PNG2ObjV3.py PlatformWilly.png --mtl -el #000000 -mw 240.0 -mh 240.0 -md 12.0
```

Produces the results with the models perfectly resized to the ratio they were designed.




It's worth noting that not all parameters need to be supplied. They can be used independently.

You can specify just the maximum width or height if there's no constraints to consider. I recommend experimenting with these parameters.

## 5.15 -sz or --startZ

The purpose of this is to set the initial Z position of the 3D Object when starting the conversion process. Why not Zero?

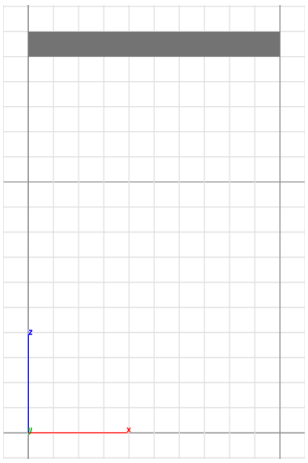
I produce a number of retro inspired plaques where the base plate is fixed, the design doesn't change, however the PNG Image to be converted, I want to start at a known fixed Z-Position meaning that dragging the base file and the converted file into the Slicing software means minimal effort as all models are aligned on the build plate.

Equally, producing QR Code Coasters are made easier by setting the correct height on the template coaster OBJ File.

### 5.15.1 Usage

```
./PNG2ObjV3.py JSWStrideSingle.png --mtl -el #000000 -nf -sz 15.0
```

Results in a converted object who's start position is at 15.0



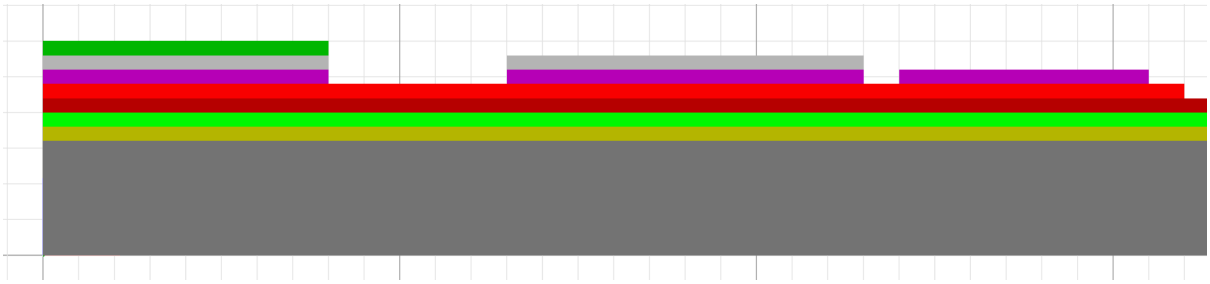
## 5.16 -ild or --initialLayerDepth

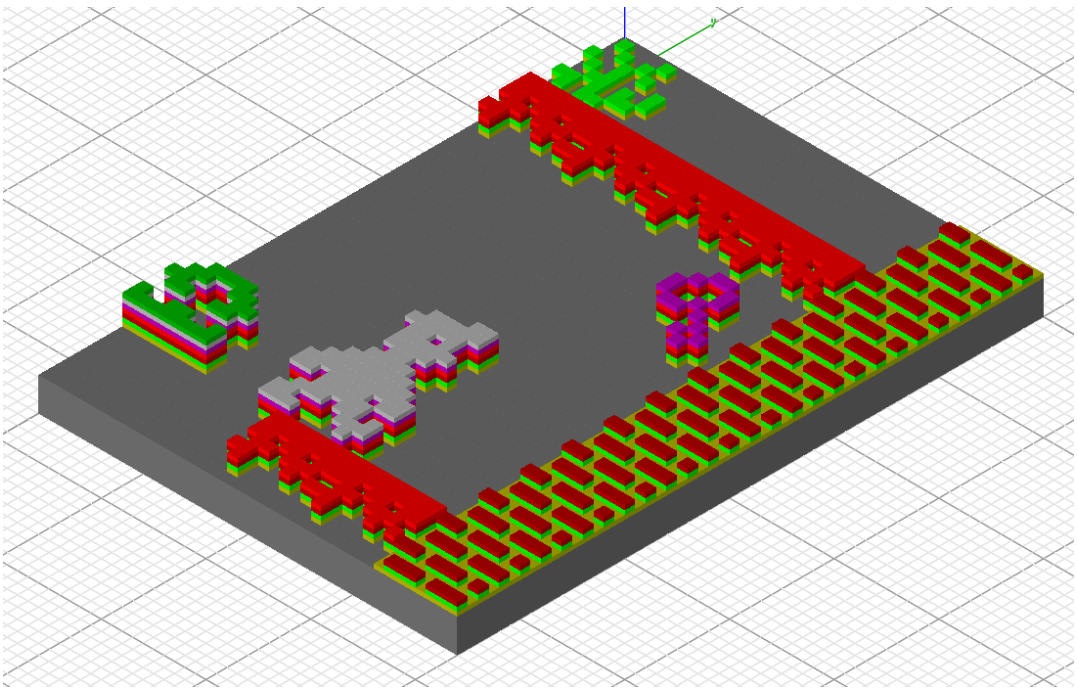
By default, the program created a bounding frame/plate for objects to sit on. When building my plaque plates for example, the first layer may want to be thicker or thinner depending on the design.

### 5.16.1 Usage

```
./PNG2ObjV3.py PlatformWilly.png --mtl -md 6.0 -layered -ild 3.2
```

Processes all colours and sets the initial layer to 3.2mm Depth, remaining colours are set to ensure each layer is equal and all total to 6mm.

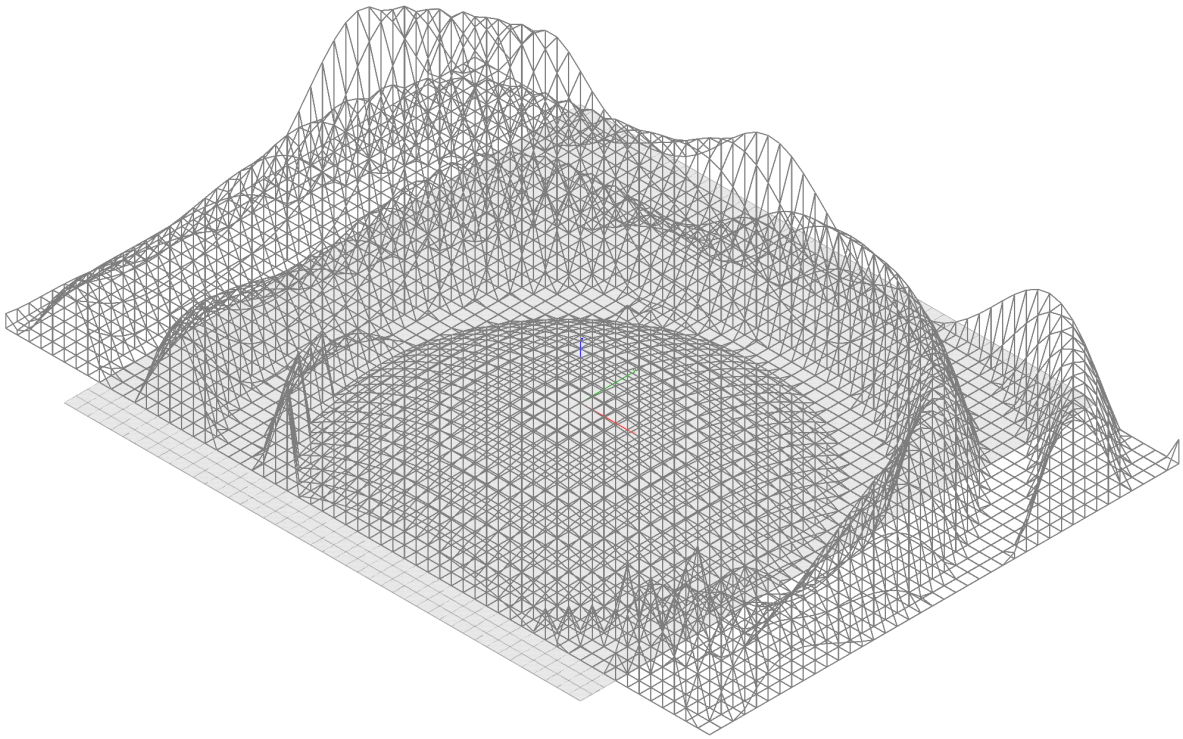


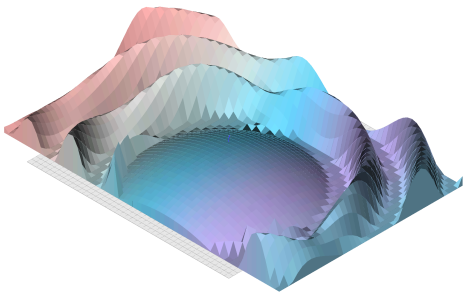


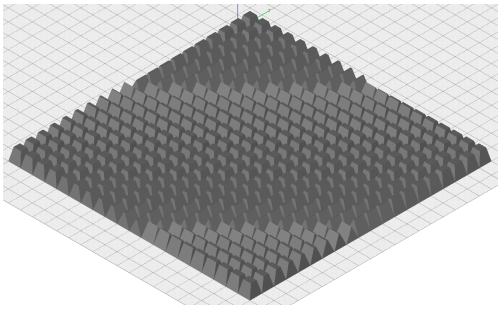
# 6 Parametric Object Generation

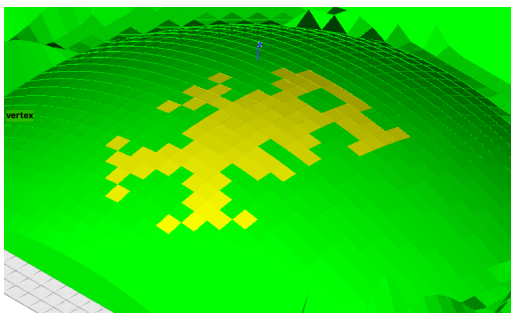
This feature is still under development, here's some very basic examples of what I'm working on…

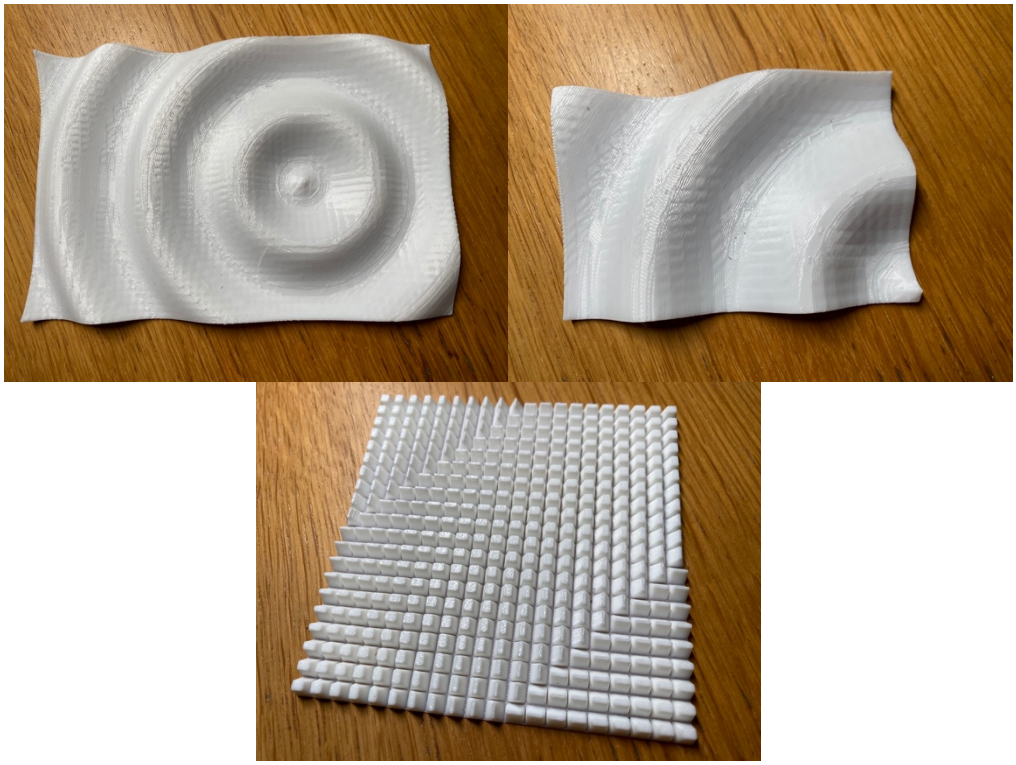
Applying Sine waves to objects and manipulating 3D Objects for 3D Printing purpose. It will make sense when ready for release.











# 7 SVG File Creation

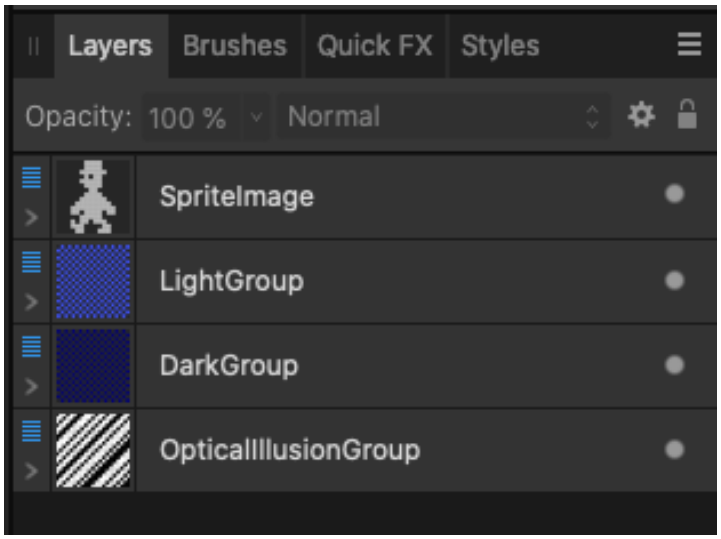
Creating a basic simple SVG file based on a PNG image is straight forward using this command line program.

The program is very simple in design and currently there's no processing implemented to combine adjacent pixels into joined shapes. If this is important you can achieve this in your favourite art program supporting SVG File formats. I use Affinity Designer from Serif and Adobe Illustrator.

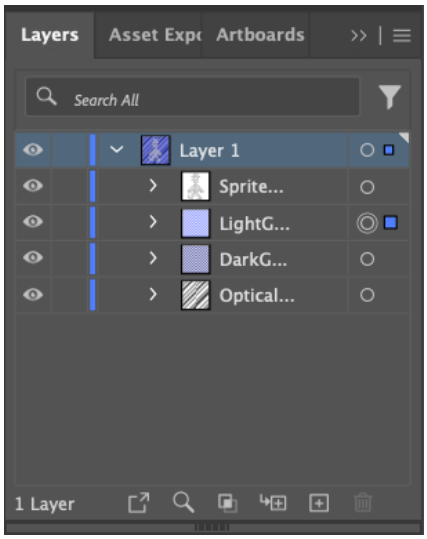
https://affinity.serif.com/ or https://www.adobe.com

There are many free programs available, for example, GIMP https://www.gimp.org or Inkscape https://inkscape.org for post processing of the SVG Data created, there's many more solutions available.

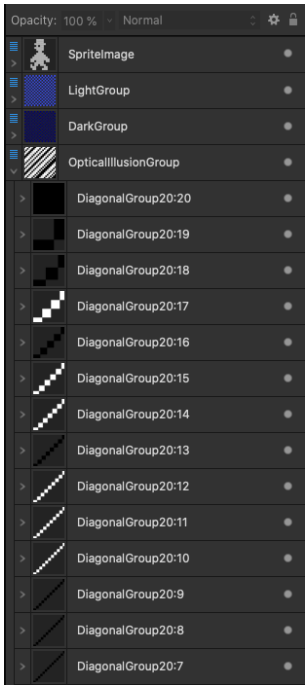
For your convenience in post processing, SVG Elements are grouped for easier post processing/editing. Even the diagonal optical illusion elements are grouped by co-ordinates enabling you to make simple changes to the illusion effect.



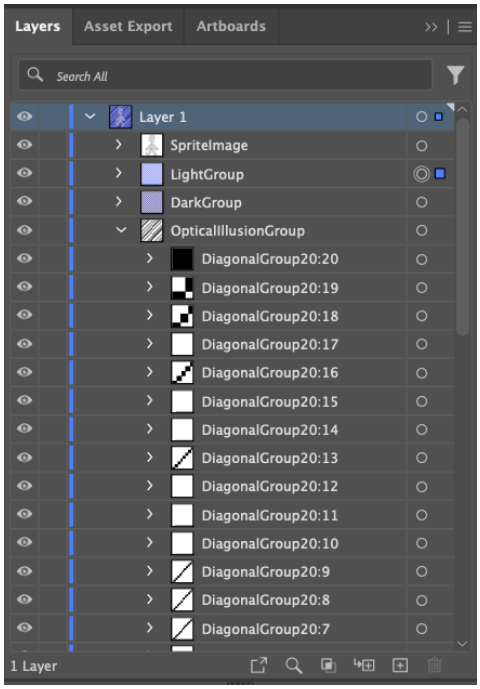

**Affinity Designer**




**Adobe Illustrator**




**Affinity Designer**




**Adobe Illustrator**

# 8 The following describes the programs SVG Specific parameters and their usage.

You will need to have Python3 installed on your system.

## 8.1 Basic Operation

Start the python program using: -

```
python3 ./PNG2ObjV3.py -svg PNGFilename.png
```

or on Linux/macOS with Python3 installed

```
./PNG2ObjV3.py -svg PNGFilename.png
```

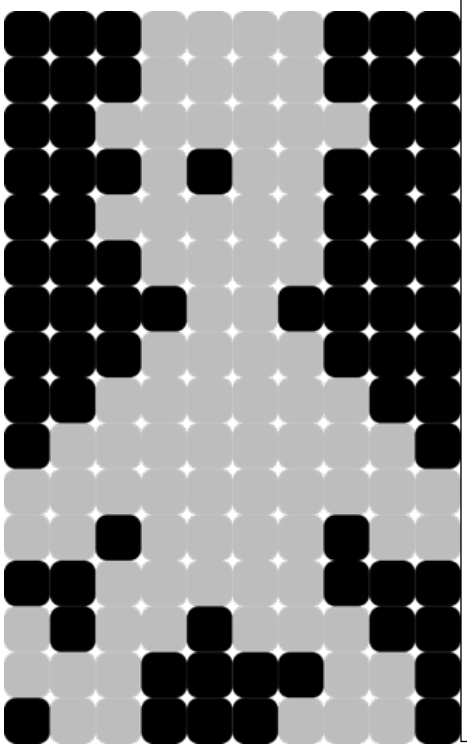
The program was developed and tested on macOS, Ubuntu Linux and Raspberry Pi OS with Python 3.8.5 installed and suggest this to be the minimum required.

## 8.2 Command Syntax - All defaults

### 8.2.1 Usage

```
python3 ./PNG2ObjV3.py -svg ./JSWStride.png
```

Creates the file **JSWStride.svg** from the PNG Image **JSWStride.png** using all the defaults, rounded corner rectangle. Produces the image below.



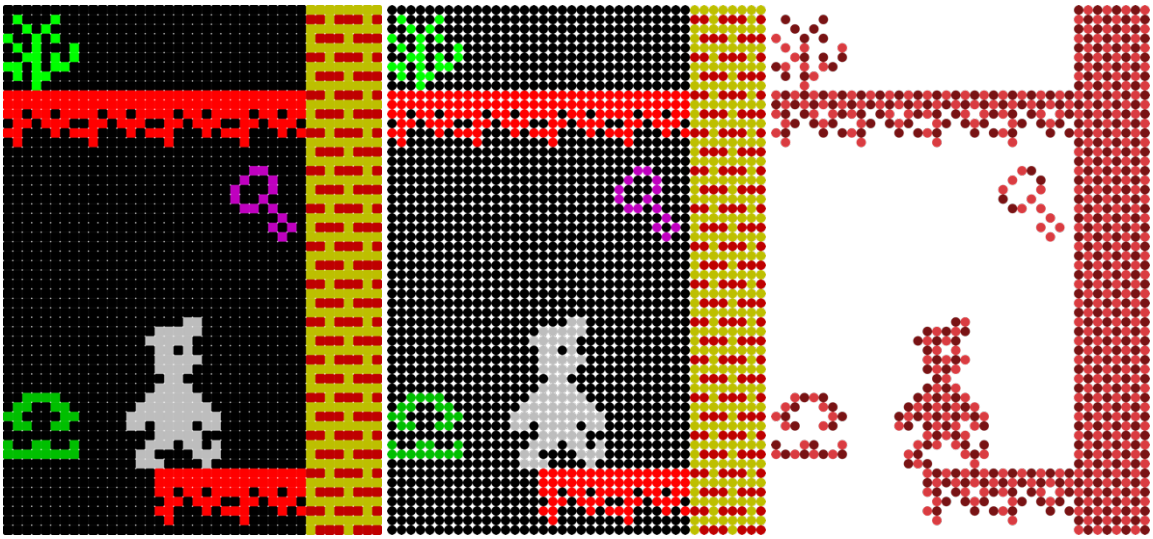
```
./PNG2ObjV3.py -svg JSWStride.png

2024-09-20 16:11:56.410450: Loaded PNG file: JSWStride.png

    Image Information :
    -------------------

          Image File : JSWStride
      PNG Image Size : 10px (width), 16px (height)
   Number of Channels : 3

     Joints Requested : False
 Create Material File : False
 Reverse Sort Colours : False
Create Flat File Only : True
         Alpha Cutoff : 128

            Requested :
   Pixel Width/Height : 20.00mm x 20.00mm
          Rect Radius : 25.00% x 25.00%


              SVG Parameters :
              ---------------

   SVG Pixel Width Requested : 20.0mm
  SVG Pixel Height Requested : 20.0mm
         SVG Corner Radius X : 25.0%
         SVG Corner Radius Y : 25.0%
              Add loaded PNG : False
        Use PNG Real Colours : True
              Create Outline : False
            Output Filename : SVGTestFile.svg
Open SVG File after creation : True

Creating SVG File from PNG:
     Total PNG Pixels : 160

Written file: SVGTestFile.svg
```

### 8.2.2 Usage

```
python3 ./PNG2ObjV3.py -svg ./PlatformWilly.png
```

```
python3 ./PNG2ObjV3.py -svg -svgrpx 100.0 ./PlatformWilly.png
```

```
python3 ./PNG2ObjV3.py -svg -svgrpx 100.0 -ugc -el #000000 ./PlatformWilly.png
```



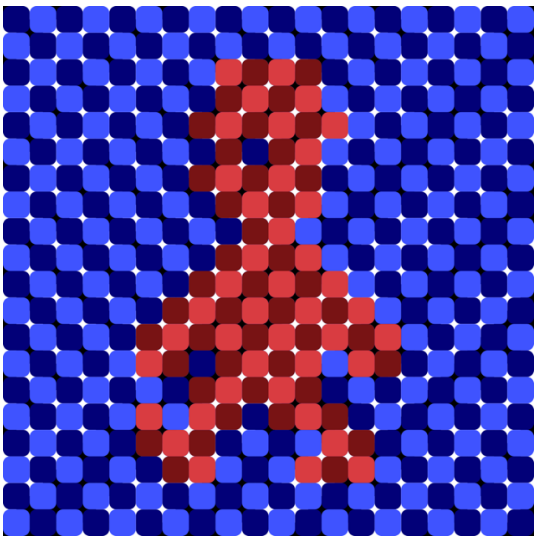
## 8.3 Parameter Information

This describes each of the parameters in turn and their defaults, you can combine many of these parameters to create interesting designs. Examples are provided below; some parameters may need careful ordering when filenames are required due to the way the command line is parsed. The filename of the PNG file to process must always be the final value passed to the program.

Many parameters have short and long version names for convenience.

```
./PNG2ObjV3.py -svg -svgopen -outfile ./mySVGFile.svg -el "#000000" -illusion -ict "#D93C41"
"#781314" "#3F53FF" "#020078" -ugc -svgaddpng JSWStride.png
```

Creates :-



## 8.4 -svgpw or --svg_pixel_width

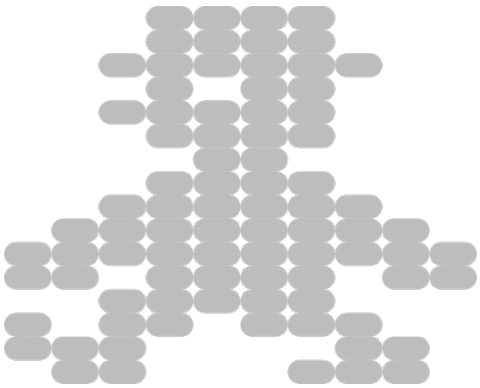
Define the height in millimetres of each SVG Objects that are created.

**Default: 20.0mm per pixel.**

### 8.4.1 Usage

Request the Pixel Width set to 40.0 mm per pixel.

```
./PNG2ObjV3.py -svg -svgopen -svgpw 40.0  JSWStrideSingle.png
```



## 8.5 -svgph or --svg_pixel_height

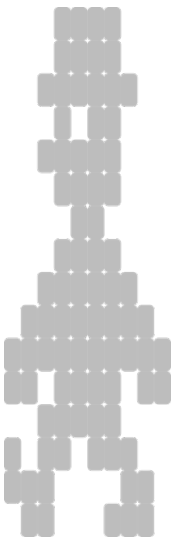
Define the height in millimetres of each SVG Objects that are created.

**Default: 20.0mm per pixel.**

### 8.5.1 Usage

Request the Pixel Width set to 40.0 mm per pixel.

```
./PNG2ObjV3.py -svg -svgopen -svgph 40.0  JSWStrideSingle.png
```



## 8.6 -svgrpx or --svg_radius_percent_x

Set the Radius Percentage on the X-Axis per pixel. Note some SVG Editors don't correctly parse this, Affinity Designer does. Functionality will change to make explicit the rounding in mm.

The closer to 100% results in a more circular shape.

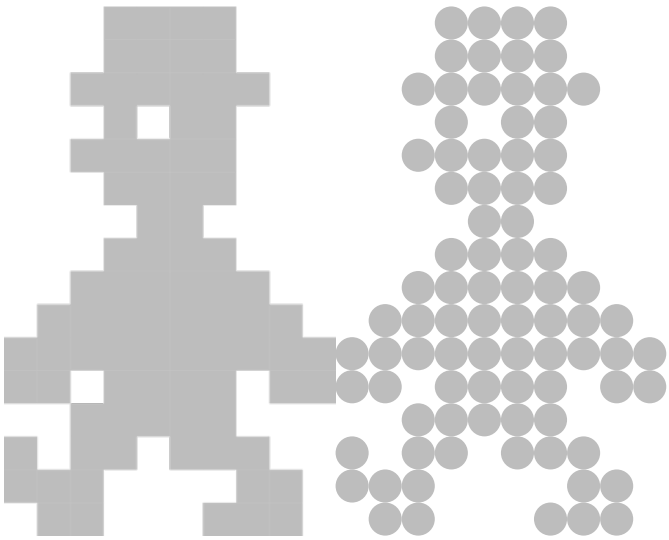
In testing, I found different art programs interpret and render rounded rectangles elements differently. Affinity Designer appears to ignore the **RY** Parameter, therefore **RX = RY**. Web Browser and Illustrator handle this differently to. Although the command and output data set the RX and RY Values, interpretation of the specs vary by program.

See:
https://www.w3.org/TR/SVG11/shapes.html#RectElementRXAttribute

Default: 25%

### 8.6.1 Usage

```
./PNG2ObjV3.py -svg -svgopen -svgrpx 0 JSWStrideSingle.png
./PNG2ObjV3.py -svg -svgopen -svgrpx 100 JSWStrideSingle.png
```



## 8.7 -svgrpy or --svg_radius_percent_y

Set the Radius Percentage on the Y-Axis per pixel. Note some SVG Editors don't correctly parse this, Affinity Designer does. The closer to 100% results in a more circular shape.

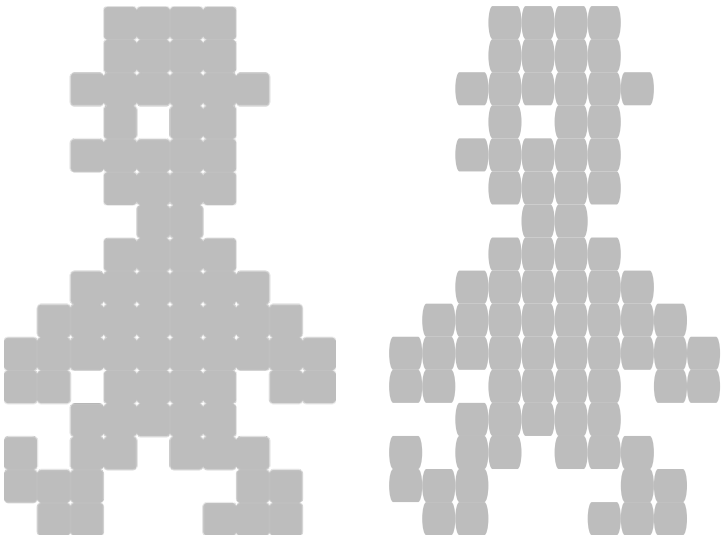
In testing, I found different art programs interpret and render rounded rectangles elements differently. Affinity Designer appears to ignore the **RY** Parameter, therefore **RX = RY**. Web Browser and Illustrator handle this differently to. Although the command and output data set the RX and RY Values, interpretation of the specs vary by program.

See:
https://www.w3.org/TR/SVG11/shapes.html#RectElementRXAttribute

Default: 25%

### 8.7.1 Usage

```
./PNG2ObjV3.py -svg -svgopen -svgrpy 0 JSWStrideSingle.png
./PNG2ObjV3.py -svg -svgopen -svgrpy 100 JSWStrideSingle.png
```



**RY = 0** **RY = 100.0**

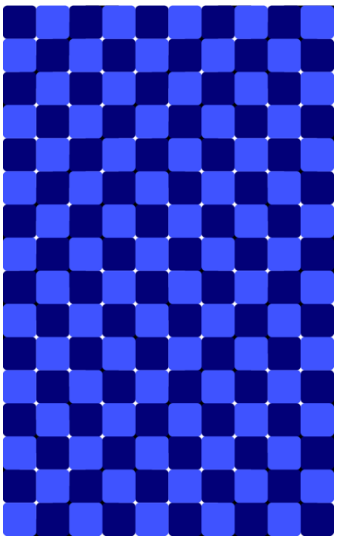
## 8.8 -illusion or --illusion

Creates a basic optical illusion using positive and negative space between the rounded rectangles to give the viewer the illusion the image wobbles.

On its own, it's useful for providing a blank grid. The grid size will take all defaults and the size of the PNG image in pixels.

### 8.8.1 Usage

```
./PNG2ObjV3.py -svg -illusion JSWStrideSingle.png
```



## 8.9 -f400 or --frame400

This option is used for creating an SVG file suitable for The Range 400mm Frame.

The internal usable dimensions at the time of writing are 397mm, with a 405mm border to fit inside the frame.

https://www.therange.co.uk/home-furnishings/picture-frames-and-wall-art/picture-frames/deep-box-frames/grey-wood-effect-box-frame/?position=7&s=320436#320436

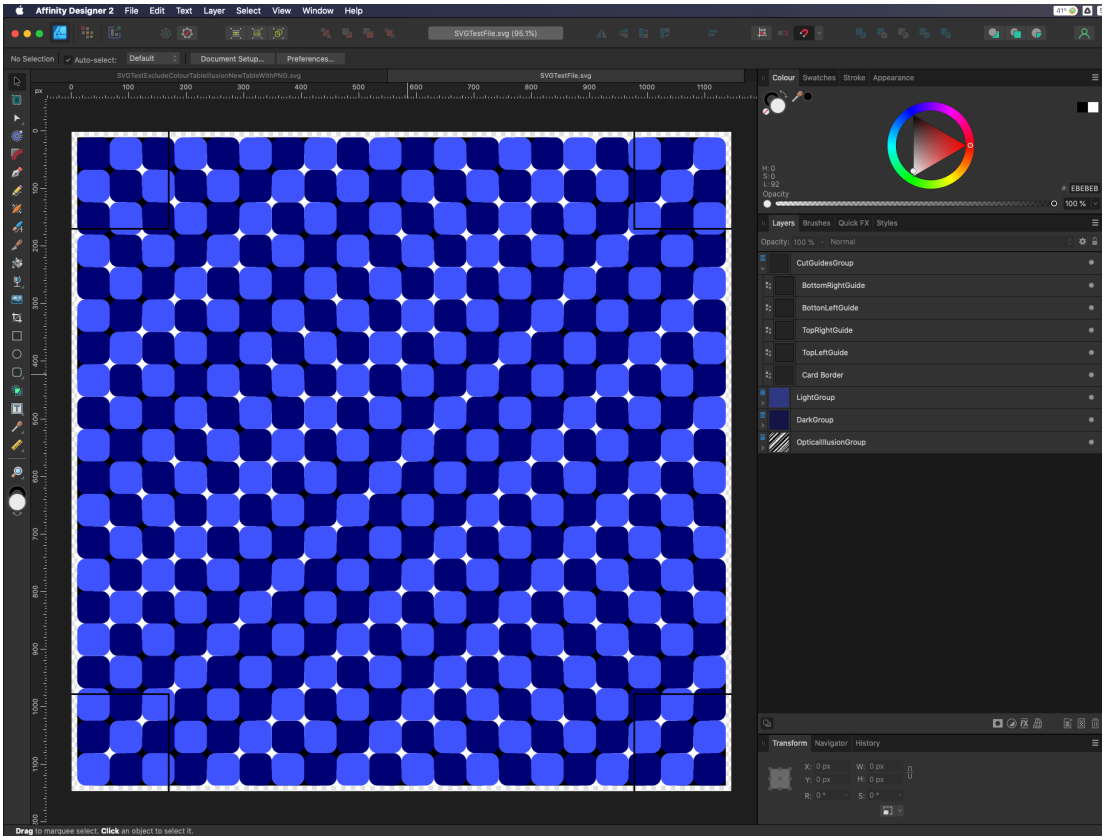
The program will add cut guides suitable for use with a MAPED straight cutter, which has a 60mm offset from the bottom edge of the ruler to the actual blade.

70mm is needed if using the 45 degree bevel cutter.

The purpose of this is to mark out on Mountboard or Paper where to align the ruler to create a more precise cut.

Typically, you would use this option with the **-outline** parameter to create a design suitable for your printer, plotter or Cutting Machine.

https://www.amazon.co.uk/Maped-UGM172500-60cm-Mount-Cutter/dp/B003GDX636




### 8.9.1 Usage

For markup, you will most likely use **-outline** in combination.

```
./PNG2ObjV3.py -svg -svgopen -f400 JSWStrideSingle.png
./PNG2ObjV3.py -svg -svgopen -f400 -outline JSWStrideSingle.png
```

## 8.10 -outline or --outlineOnly

This parameter takes on two differing forms. On its own it simply produces an outline of the PNG Image in the form of circles where each pixel is detected.

### 8.10.1 Usage

```
./PNG2ObjV3.py -svg -outline JSWStrideSingle.png
./PNG2ObjV3.py -svg -outline JSWStride.png
```

Results in the two images below.

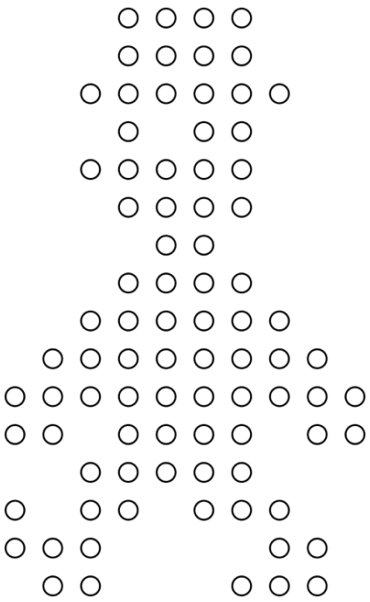


**Image with Transparency**

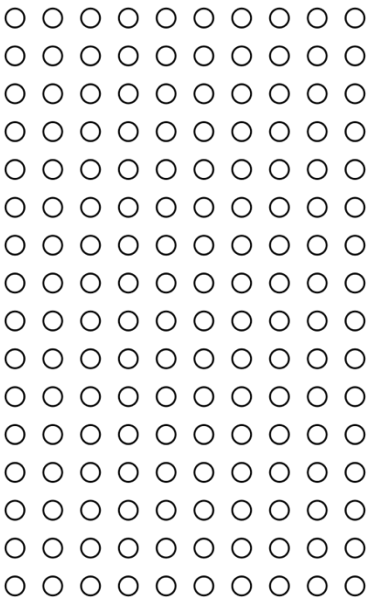


**Image without Transparency**

Where a transparency pixel isn't defined (ALPHA Channel < ALPHACUT OFF) all valid pixels will be represented with a circle.

**JSWStrideSingle.png** is single colour with transparency, **JSWStride.png** has no transparency utilising the colour black for a background.

If you the transparency colour is known to you, add a colour exclusion list using the **-EL** or Process Colour List **-PC** then all pixels will be used, resulting in something like this.

Another combination to achieve the same result would be:

### 8.10.2 Usage

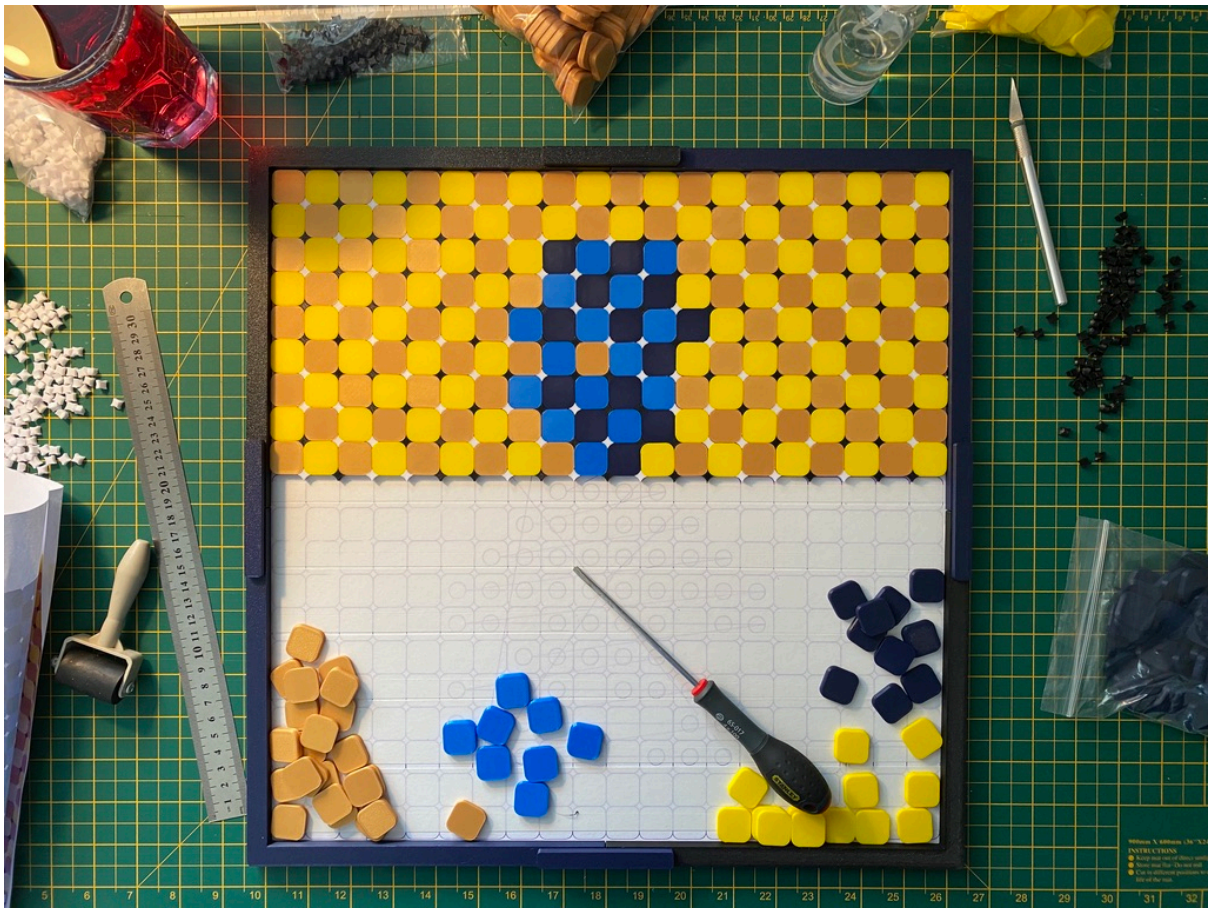
The version requests the program to ignore all pixels that have colour attribute **#00000** or Black.

```
./PNG2ObjV3.py -svg -el #000000 -outline JSWStride.png
```

The version requests the program to process only that have colour attribute **#bdbdbd** or Grey.

```
./PNG2ObjV3.py -svg -pc #bdbdbd -outline JSWStride.png
```

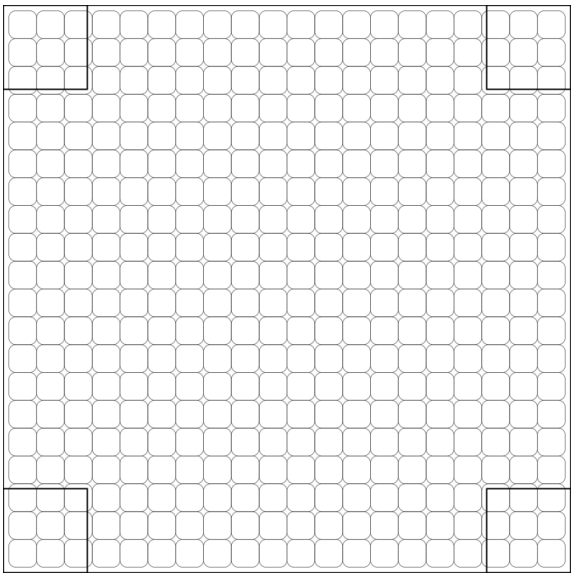
The power of the outline parameter comes in to play when creating 3D Printed Art as locaters/placeholders for pieces to be added to canvas/mountboard/paper etc.



### 8.10.3 Combined with F400 Parameter

As discussed above, outline and f400 are the most useful parameter combination, creating a file with cut guides and location markers to aid constructing 3D Printed or cut objects on the grid.

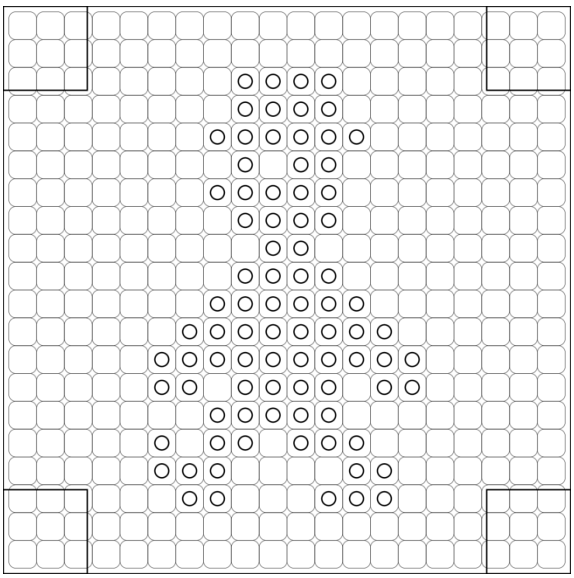
```
./PNG2ObjV3.py -svg -f400 -outline JSWStrideSingle.png
```

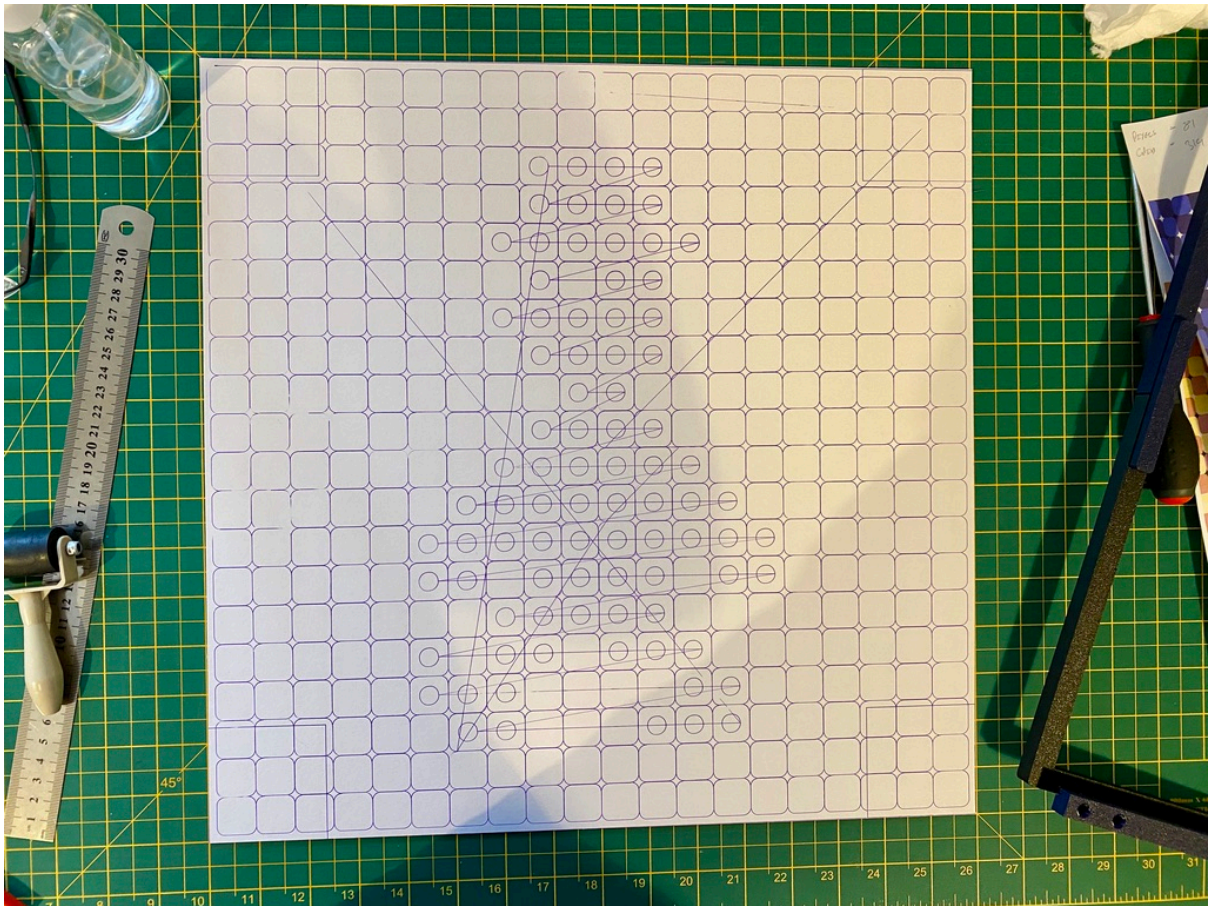


Finally, you will need markers for the PNG Image you want to place on the grid.

### 8.10.4 Usage

```
./PNG2ObjV3.py -svg -f400 -outline -svgaddpng JSWStrideSingle.png
./PNG2ObjV3.py -svg -f400 -el #000000 -outline -svgaddpng JSWStride.png
```





The joys of firmware issues with the Plotter I'm using… yep the pen doesn't lift when moving position/location.  Thankfully this design is covered with the 3D Printed parts.

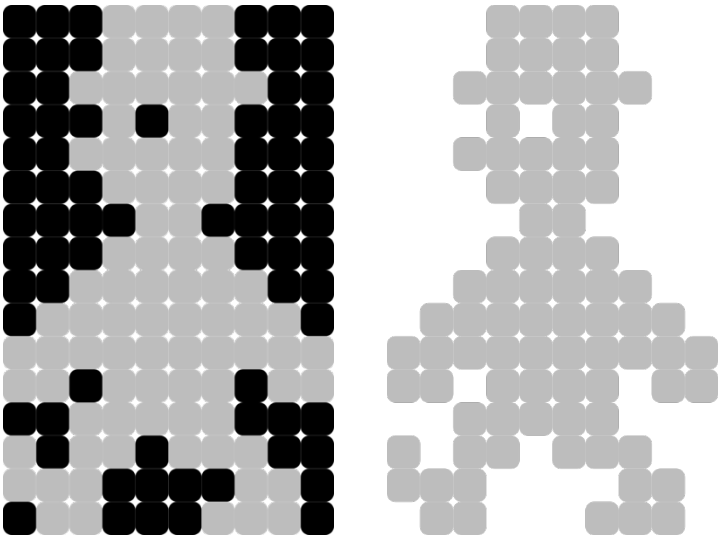
## 8.11 -svgaddpng or --svgaddpng

This parameter is only necessary when creating the Optical Illusion File or Frame Outlines ready for markup and cutting/creating.

### 8.11.1 Usage

```
./PNG2ObjV3.py -svg -svgaddpng JSWStride.png
./PNG2ObjV3.py -svg -svgaddpng JSWStrideSingle.png
```

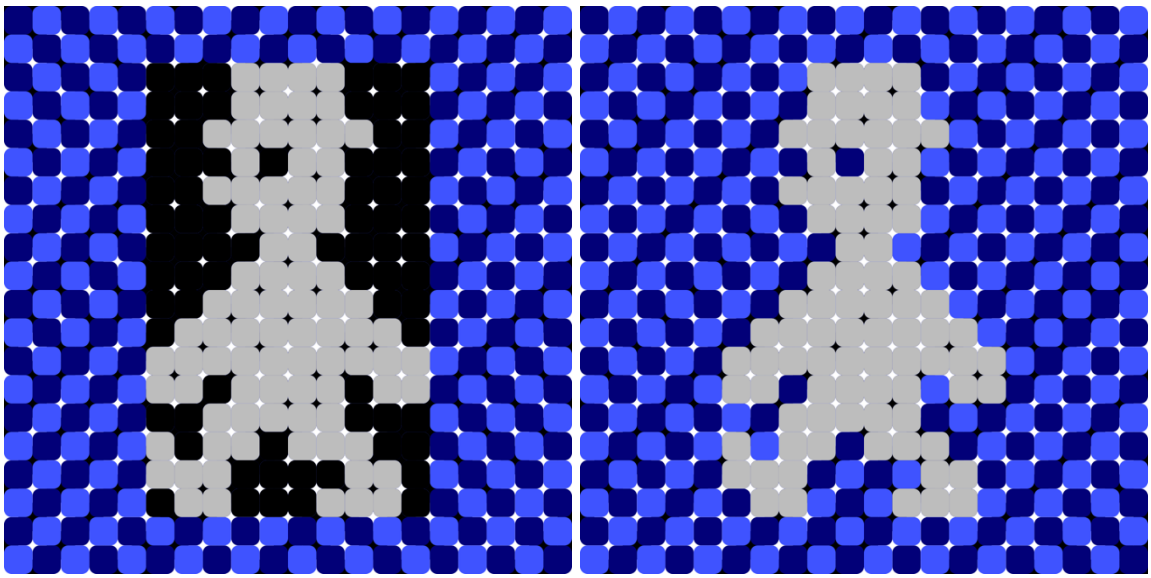
Results in creating the default object.



### 8.11.2 Combined with -illusion or -f400

If you are creating artwork, you will need to add this option to either the -illusion parameter or -f400 parameter to include the PNG Image in the design.

```
./PNG2ObjV3.py -svg -svgopen -illusion -svgaddpng JSWStride.png
./PNG2ObjV3.py -svg -svgopen -illusion -svgaddpng JSWStrideSingle.png
./PNG2ObjV3.py -svg -svgopen -illusion -svgaddpng -el #000000 JSWStride.png
./PNG2ObjV3.py -svg -svgopen -illusion -svgaddpng -pc #bdbdbd JSWStride.png
```



Remember, if your image doesn't include transparency, you'll need to either exclude the colours not required or ensure the parts of the PNG image are transparent you don't want included in the final design.

## 8.12 -svgopen

This parameter will attempt to open the final created file with your systems default SVG File handler. I'm currently using Affinity Designer and Adobe Illustrator. Designer is the default application. Though it could be your web browser or free programs like Inkscape and GIMP

### 8.12.1 Usage

```
./PNG2ObjV3.py -svg -svgopen -illusion -svgaddpng -svgopen JSWStrideSingle.png
```

## 8.13 -ict or --illusioncolourtable

Pre-set colour combination tables are defined in the program, though you'll want to change these on the fly without modifying code. To achieve this use the -ICT command to send your preferred colour codes when creating your design.

Four colours in the Hexadecimal (HTML Colour Codes) must be defined each starting with a # and consist of six hexadecimal digits without space.

i.e. #000000 = Black, or #ff0000 = Red.

If you're unfamiliar, take a look here: https://en.wikipedia.org/wiki/Web_colors

The colour codes are ordered as follows

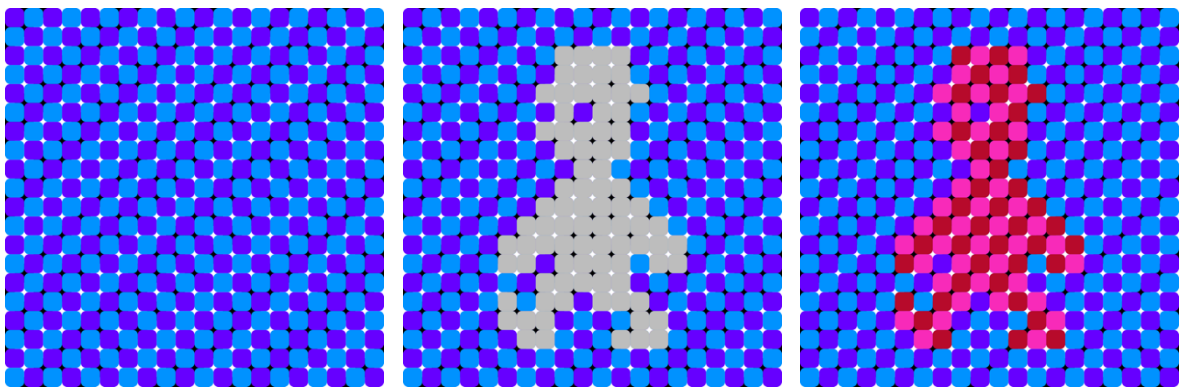
**Grid Light Colour, Grid Dark Colour, Image Light Colour, Image Dark Colour**

The first two relate to the actual grid created for the optical illusion, the second two colours relate to the colour substitution for each pixel that's placed on the grid.

Add the **-UGC** parameter to use Grid Colours instead of the PNG Image Colours.

### 8.13.1 Usage

```
./PNG2ObjV3.py -svg -ict #0092FF #6600FF #F82AB9 #B70A2B -illusion JSWStrideSingle.png
./PNG2ObjV3.py -svg -ict #0092FF #6600FF #F82AB9 #B70A2B -svgaddpng -illusion JSWStrideSingle.png
./PNG2ObjV3.py -svg -ict #0092FF #6600FF #F82AB9 #B70A2B -svgaddpng -illusion -ugc
JSWStrideSingle.png
```



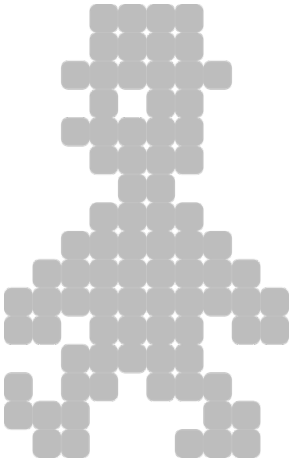
## 8.14 -urc or --userealcolours

This option is the default and uses the actual pixel colour information found in the PNG Image.

It's rather redundant and will be removed in a later version.

### 8.14.1 Usage

```
./PNG2ObjV3.py -svg -urc JSWStrideSingle.png
```

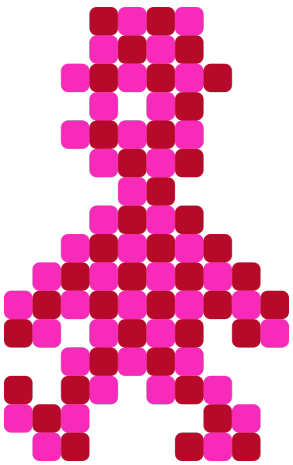


## 8.15 -ugc or --usegridcolours

This option will substitute the original PNG Colours with alternating grid, light and dark colours either defined in default, the colour set requested or your custom colours provide in the -ICT Parameter.

### 8.15.1 Usage

```
./PNG2ObjV3.py -svg -ugc JSWStrideSingle.png
```



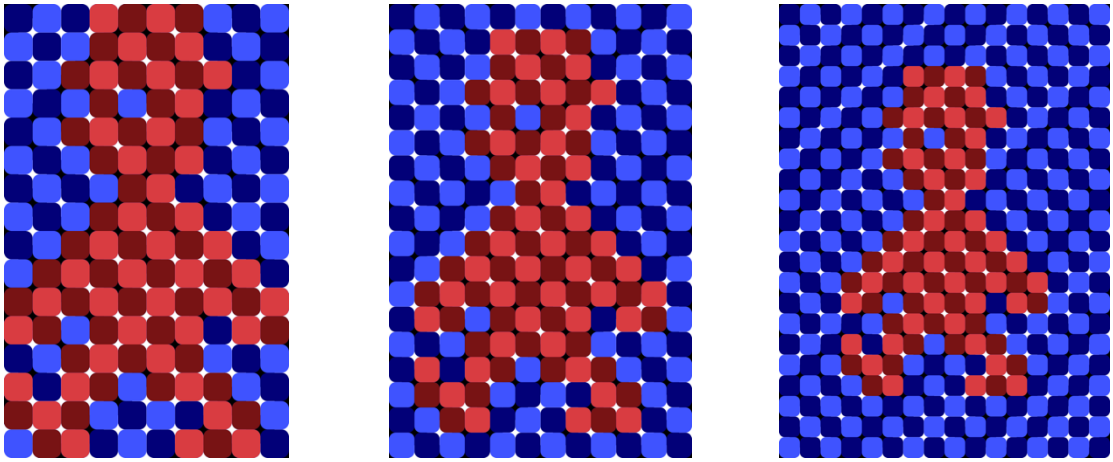
## 8.16 -mb or --minimumborder

This parameter is only applicable with the **-illusion** parameter set.

This will add x number of pixels as a minimum border around the PNG Image to add to the grid.

The default is 0 (No minimum border).

Examples below show Minimum Border Width of Zero, One and Three



### 8.16.1 Usage

```
./PNG2ObjV3.py -svg -illusion -svgaddpng -ugc -mb 0 JSWStrideSingle.png
./PNG2ObjV3.py -svg -illusion -svgaddpng -ugc -mb 1 JSWStrideSingle.png
./PNG2ObjV3.py -svg -illusion -svgaddpng -ugc -mb 3 JSWStrideSingle.png
```

Default = 0

## 8.17 -mgw or --minimumgridwidth

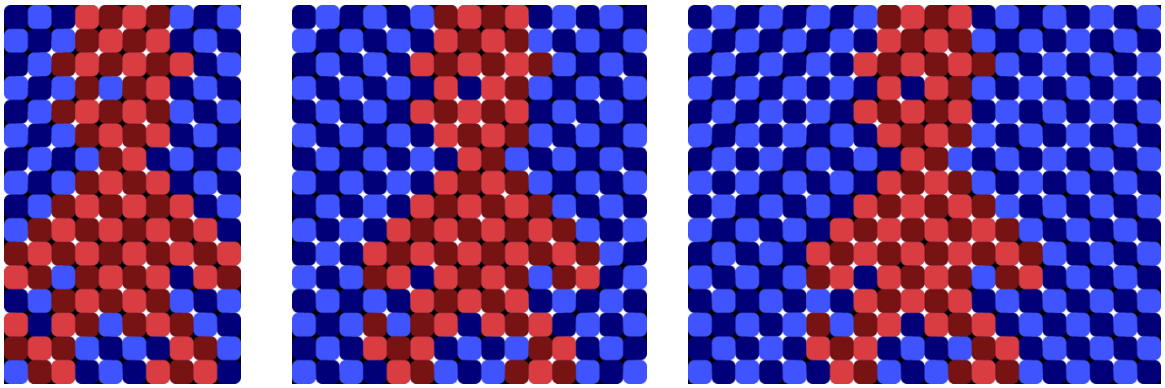
Here we define the **minimum** Illusion Grid Block width. For example, if the PNG Image information is less than the value defined, then the minimum width will apply. If the PNG Image is bigger, then PNG Image width will be used.

Default = 4

### 8.17.1 Usage

```
./PNG2ObjV3.py -svg -illusion -svgopen -svgaddpng -ugc -mgw 8 JSWStrideSingle.png
./PNG2ObjV3.py -svg -illusion -svgopen -svgaddpng -ugc -mgw 15 JSWStrideSingle.png
./PNG2ObjV3.py -svg -illusion -svgopen -svgaddpng -ugc -mgw 20 JSWStrideSingle.png
```

Using the example above results in the following images.



Note the PNG Image will always be rendered centred (Block Aligned) to the grid.

## 8.18 -mgh or --minimumgridheight

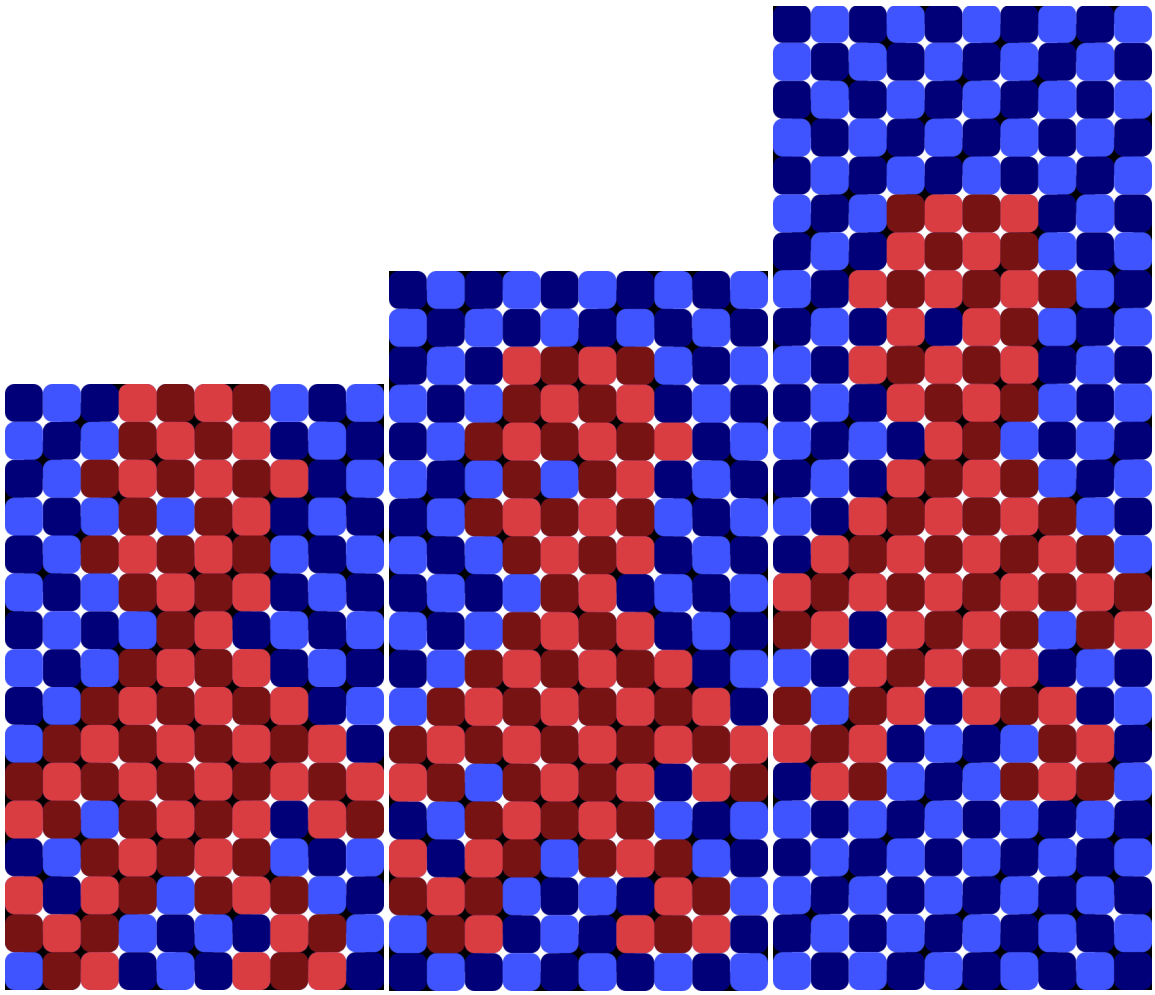
Here we define the **minimum** Illusion Grid Block Height. For example, if the PNG Image information is less than the value defined, then the minimum Height will apply. If the PNG Image is bigger, then PNG Image width will be used.

Default = 4

### 8.18.1 Usage

```
./PNG2ObjV3.py -svg -illusion -svgopen -svgaddpng -ugc -mgh 8 JSWStrideSingle.png
./PNG2ObjV3.py -svg -illusion -svgopen -svgaddpng -ugc -mgh 19 JSWStrideSingle.png
./PNG2ObjV3.py -svg -illusion -svgopen -svgaddpng -ugc -mgh 26 JSWStrideSingle.png
```

Using the example above results in the following images.



Note the PNG Image will always be rendered centred (Block Aligned) to the grid.

## 8.19 -ilc or --illusioncircle

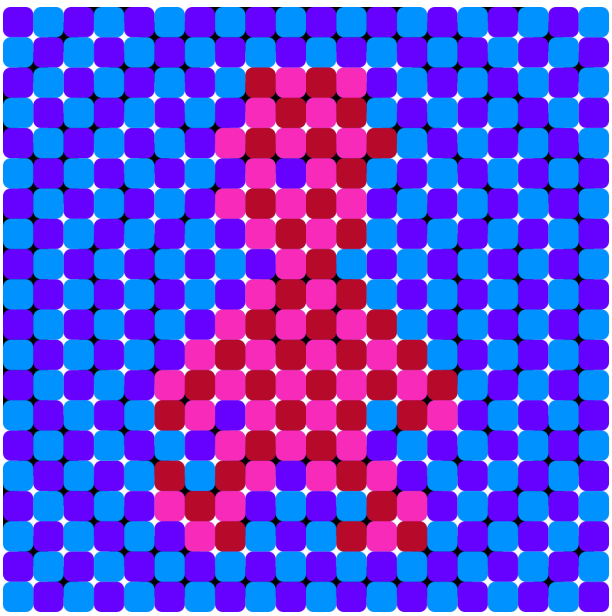
This feature attempts to set the Positive and Negative Space between pixels in a circular style.

It can't be a perfect circle since the space between the blocks would result in both Black and White colours on the circular boundary.

The program will create an Array of Data representing the points where negative/positive space resides and using Bresenham's Algorithm, creates circular points within the array, creating the SVG data appropriate.

```
   **  *******  **
  *  ***********  *
 *  ***        ***  *
*  **   *****   **
* **  **     **  **
 **  *  *****  *  **
 ** *  *     *  * **
**  * *       * *  *
** * *         * * *
** * *   ***   * * *
** * *   * *   * * *
** * *   ***   * * *
** * *         * * *
**  * *       * *  *
 ** *  *     *  * **
 **  *  *****  *  **
* **  **     **  **
*  **   *****   **
 *  ***        ***  *
  *  ***********  *
```

Which creates a different style Optical Illusion shown below.



### 8.19.1 Usage

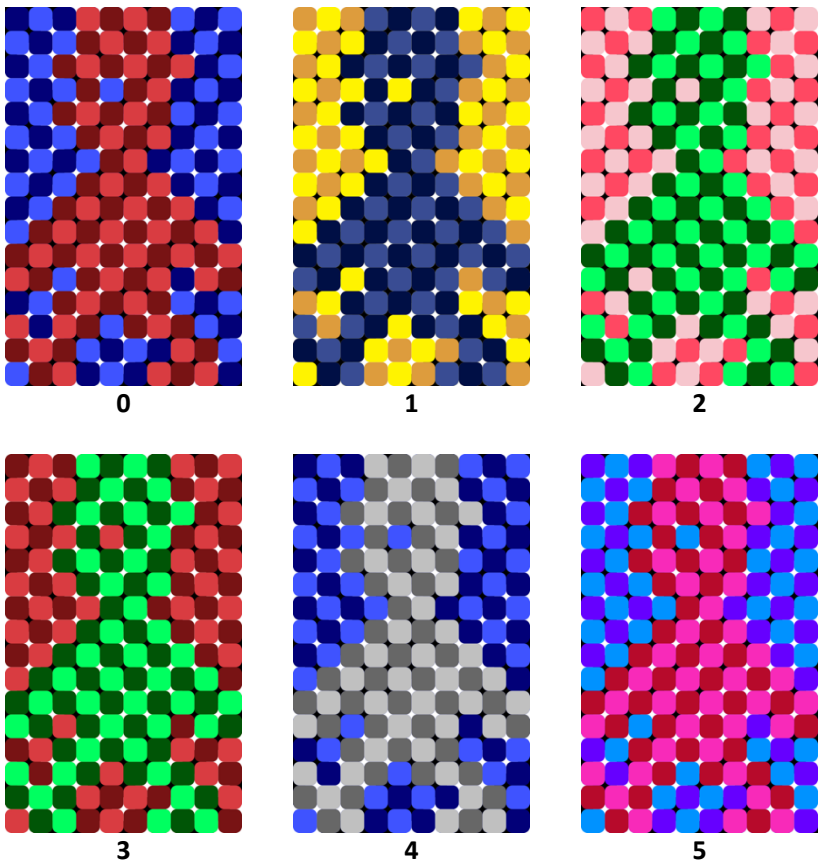
```
./PNG2ObjV3.py -svg -illusion -svgopen -svgaddpng -ugc -mgh 20 -mgw 20 -ilc  JSWStrideSingle.png
```

## 8.20 -cset or --colourset

A few present colour pairings are hardcoded into the program for simple use if you don't wish to define your own colour set using the -ICT parameter. Add the set number you're requesting, an integer between 0 and 5 currently.

If you request a higher number than the sets coded, the program will simply MOD the request with the maximum number of sets available.

**CSET** Examples shown below.




### 8.20.1 Usage

```
./PNG2ObjV3.py -svg -illusion -svgaddpng -ugc -CSET 5 JSWStrideSingle.png
```

## 8.21 -stmax or --maxstreak

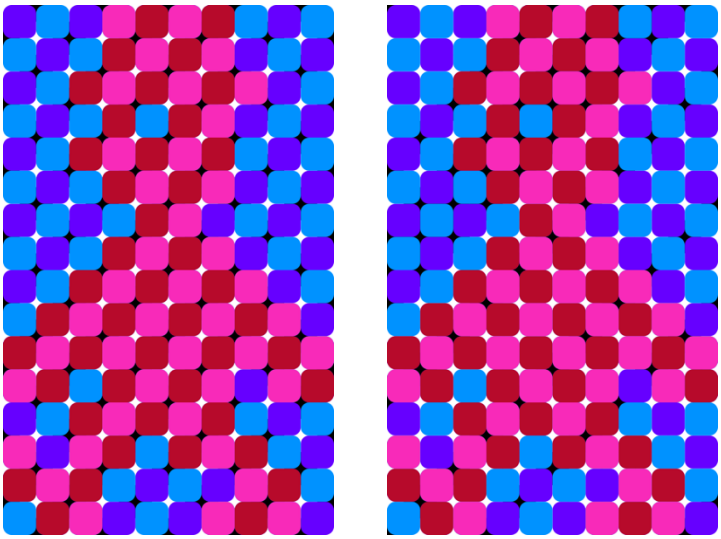
Streak Max parameter is useful for defining the maximum number of runs of the same positive/negative space between grid elements allowed.

**Default 3**

The program randomly chooses light or dark elements when creating the optical illusion, however due to the nature of pseudo randomness, the illusion could break with too many runs of black/white infill pixels.

The STMAX parameter helps by forcing a break in the pattern as a maximum and switching to the alternate colour.

For example, setting STMAX to the value 1 results in forcing a colour switch every pixel as seen in the example below. Setting STMAX to a higher value creates a differing result.



**STMAX 1** **STMAX 5**

### 8.21.1 Usage

```
./PNG2ObjV3.py -svg -illusion -svgaddpng -ugc -cset 5 -stmax 1 JSWStrideSingle.png
./PNG2ObjV3.py -svg -illusion -svgaddpng -ugc -cset 5 -stmax 5 JSWStrideSingle.png
```