pygame.transform

pygame module to transform surfaces

- pygame.transform.flip— flip vertically and horizontally
- pygame.transform.scale — resize to new resolution
- pygame.transform.rotate — rotate an image
- pygame.transform.rotozoom — filtered scale and rotation
- pygame.transform.scale2x — specialized image doubler
- pygame.transform.smoothscale — scale a surface to an arbitrary size smoothly
- pygame.transform.get_smoothscale_backend — return smoothscale filter version in use: 'GENERIC', 'MMX', or 'SSE'
- pygame.transform.set_smoothscale_backend — set smoothscale filter version to one of: 'GENERIC', 'MMX', or 'SSE'
- pygame.transform.chop — gets a copy of an image with an interior area removed
- pygame.transform.laplacian — find edges in a surface
- pygame.transform.average_surfaces — find the average surface from many surfaces.
- pygame.transform.average_color — finds the average color of a surface
- pygame.transform.threshold — finds which, and how many pixels in a surface are within a threshold of a color.

A Surface transform is an operation that moves or resizes the pixels. All these functions take a Surface to operate on and return a new Surface with the results.

Some of the transforms are considered destructive. These means every time they are performed they lose pixel data. Common examples of this are resizing and rotating. For this reason, it is better to re-transform the original surface than to keep transforming an image multiple times. (For example, suppose you are animating a bouncing spring which expands and contracts. If you applied the size changes incrementally to the previous images, you would lose detail. Instead, always begin with the original image and scale to the desired size.)

pygame.transform.flip()

flip vertically and horizontally

flip(Surface, xbool, ybool) -> Surface

This can flip a Surface either vertically, horizontally, or both. Flipping a Surface is non-destructive and returns a new Surface with the same dimensions.

pygame.transform.scale()

resize to new resolution

scale(Surface, (width, height), DestSurface = None) -> Surface

Resizes the Surface to a new resolution. This is a fast scale operation that does not sample the results.

An optional destination surface can be used, rather than have it create a new one. This is quicker if you want to repeatedly scale something. However the destination must be the same size as the (width, height) passed in. Also the destination surface must be the same format.

pygame.transform.rotate()

rotate an image

rotate(Surface, angle) -> Surface

Unfiltered counterclockwise rotation. The angle argument represents degrees and can be any floating point value. Negative angle amounts will rotate clockwise.

Unless rotating by 90 degree increments, the image will be padded larger to hold the new size. If the image has pixel alphas, the padded area will be transparent. Otherwise pygame will pick a color that matches the Surface colorkey or the topleft pixel value.

pygame.transform.rotozoom()
filtered scale and rotation
rotozoom(Surface, angle, scale) -> Surface
This is a combined scale and rotation transform. The resulting Surface will be a filtered 32-bit Surface. The scale argument is a floating point value that will be multiplied by the current resolution. The angle argument is a floating point value that represents the counterclockwise degrees to rotate. A negative rotation angle will rotate clockwise.

pygame.transform.scale2x()
specialized image doubler
scale2x(Surface, DestSurface = None) -> Surface
This will return a new image that is double the size of the original. It uses the AdvanceMAME Scale2X algorithm which does a 'jaggie-less' scale of bitmap graphics.

This really only has an effect on simple images with solid colors. On photographic and antialiased images it will look like a regular unfiltered scale.

An optional destination surface can be used, rather than have it create a new one. This is quicker if you want to repeatedly scale something. However the destination must be twice the size of the source surface passed in. Also the destination surface must be the same format.

pygame.transform.smoothscale()
scale a surface to an arbitrary size smoothly
smoothscale(Surface, (width, height), DestSurface = None) -> Surface
Uses one of two different algorithms for scaling each dimension of the input surface as required. For shrinkage, the output pixels are area averages of the colors they cover. For expansion, a bilinear filter is used. For the x86-64 and i686 architectures, optimized MMX routines are included and will run much faster than other machine types. The size is a 2 number sequence for (width, height). This function only works for 24-bit or 32-bit surfaces. An exception will be thrown if the input surface bit depth is less than 24.

pygame.transform.get_smoothscale_backend()
return smoothscale filter version in use: 'GENERIC', 'MMX', or 'SSE'
get_smoothscale_backend() -> String
Shows whether or not smoothscale is using MMX or SSE acceleration. If no acceleration is available then "GENERIC" is returned. For a x86 processor the level of acceleration to use is determined at runtime.

This function is provided for pygame testing and debugging.

pygame.transform.set_smoothscale_backend()
set smoothscale filter version to one of: 'GENERIC', 'MMX', or 'SSE'
set_smoothscale_backend(type) -> None
Sets smoothscale acceleration. Takes a string argument. A value of 'GENERIC' turns off acceleration. 'MMX' uses MMX instructions only. 'SSE' allows SSE extensions as well. A value error is raised if type is not recognized or not supported by the current processor.

This function is provided for pygame testing and debugging. If smoothscale causes an invalid instruction error then it is a pygame/SDL bug that should be reported. Use this function as a temporary fix only.

pygame.transform.chop()
gets a copy of an image with an interior area removed
chop(Surface, rect) -> Surface
Extracts a portion of an image. All vertical and horizontal pixels surrounding the given rectangle area are removed. The corner areas (diagonal to the rect) are then brought together. (The original image is not altered by this operation.)

NOTE: If you want a “crop” that returns the part of an image within a rect, you can blit with a rect to a new surface or copy a subsurface.

pygame.transform.laplacian()
find edges in a surface
laplacian(Surface, DestSurface = None) -> Surface
Finds the edges in a surface using the laplacian algorithm.

pygame.transform.average_surfaces()
find the average surface from many surfaces.
average_surfaces(Surfaces, DestSurface = None, palette_colors = 1) -> Surface
Takes a sequence of surfaces and returns a surface with average colors from each of the surfaces.

palette_colors - if true we average the colors in palette, otherwise we average the pixel values. This is useful if the surface is actually greyscale colors, and not palette colors.

Note, this function currently does not handle palette using surfaces correctly.

pygame.transform.average_color()
finds the average color of a surface
average_color(Surface, Rect = None) -> Color
Finds the average color of a Surface or a region of a surface specified by a Rect, and returns it as a Color.

pygame.transform.threshold()
finds which, and how many pixels in a surface are within a threshold of a color.
threshold(DestSurface, Surface, color, threshold = (0,0,0,0), diff_color = (0,0,0,0), change_return = 1, Surface = None, inverse = False) -> num_threshold_pixels
Finds which, and how many pixels in a surface are within a threshold of a color.

It can set the destination surface where all of the pixels not within the threshold are changed to diff_color. If inverse is optionally set to True, the pixels that are within the threshold are instead changed to diff_color.

If the optional second surface is given, it is used to threshold against rather than the specified color. That is, it will find each pixel in the first Surface that is within the threshold of the pixel at the same coordinates of the second Surface.

If change_return is set to 0, it can be used to just count the number of pixels within the threshold if you set

If change_return is set to 1, the pixels set in DestSurface will be those from the color.

If change_return is set to 2, the pixels set in DestSurface will be those from the first Surface.

You can use a threshold of (r,g,b,a) where the r,g,b can have different thresholds. So you could use an r threshold of 40 and a blue threshold of 2 if you like.