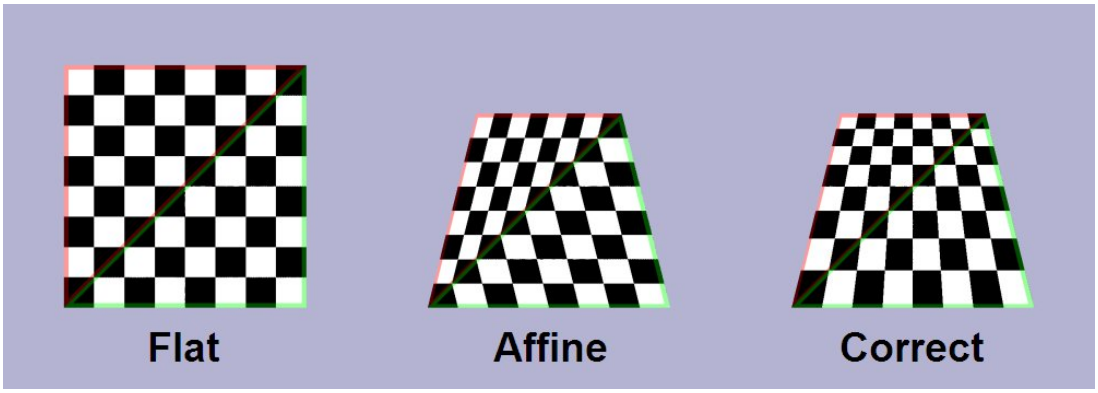
Perspective-Correct Texture Mapping

Our simple model of texture mapping works like this: projection, including the perspective divide, sets up the x, y coordinates of every triangle vertex for the rasterizer. The rasterizer linearly interpolates texture coordinates s, t across the triangle, as it does with z or color information. Then for each fragment, we use the interpolated s, t values to look up the texture.

This would lead to unrealistic images like the one in the middle in this picture from Wikipedia:



The image on the right is correct; the rectangles appear to get smaller as they get farther away, in x as well as y. Most of the shrinkage in the middle image goes into one triangle, while the other stays mainly the same size. Even if you draw a quad instead of a triangle, your image is not correct; the heights of the texture squares would not get smaller as they get farther away, even though the widths would.

To achieve this "perspective correct" texture mapping, we need to figure out the texture coordinates in a more sophisticated way. What we'd really like to do is first interpolate texture coordinates, and then do the perspective divide, but this would not work well with the rasterizer - how would it figure out what the parameter value at a fragment center ought to be? So what we'll do instead is, for a fragment with parameter α figured out using the usual divide-first-then-interpolate scheme, calculate what would have been the parameter β corresponding to that exact point if we had interpolated using the perspective-correct interpolate-first-then-divide method.

To get a formula for β, consider the rasterizer interpolating values along a row of pixel centers. A fragment $f = (f_x, f_y)$ in the row has a fixed f_y (the y-value of the row of fragments), and f_x is interpolated as usual between some x_0/w_0 and x_1/w_1, using a parameter α. The starting and ending values are ratios because they were produced by the perspective divide.

$$f_x = (1 - \alpha)x_0/w_0 + \alpha x_1/w_1$$

Now consider what would have happened if we could interpolate first and divide afterwards, this time using a parameter β:

$$f_x = \frac{(1-\beta)x_0 + \beta x_1}{(1-\beta)w_0 + \beta w_1}$$

We set these two things equal, and then, for a given α, solve for β. We get:

$$\beta = \frac{\alpha w_0}{(1-\alpha)w_1 + \alpha w_0}$$

Notice, not surprisingly, that x_0 and x_1 drop out. We could linearly interpolate anything - including s and t - across the triangle, and the value given by β will be correct for the fragment whose position is given by α.

Modern graphics hardware does this for you without even being asked!