Here is a quick lesson in bit wise operators for you to better understand the Render3D class. It is basically just easier ways of performing certain operations after understanding binary.

<< and >> are bit shifts. For example: 1 in binary is obviously just 1. But if we do 1 << 3, the binary number is shifted by 3 powers of 2 making it 1000 and being the number 8. You can do this with any number such as 3 << 4 would be 110000. Pretty simple in that aspect. And >> shifts it backwards, such as 64 >> 4 would go from 64 (which is 100000) to 10 in binary. This is good for shifting colors in my game. 255 is the brightest a color can get and not shifting it by default is a blue color, shifting it by 8 (255 << 8) is green, and shifting it by 16 makes it a red color. 255 = FF in hexadecimal as you know, and in programming you can also specify it as FFx0 if you want to state integers in hexadecimal form instead of normal integer form. The x0 says that it is a hexadecimal number.

| is the OR operator. It basically merges the two binary numbers together. For example:

0011010 (26)

| 0101001 (41)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0111011 (59)

Now in programming I use this operator to merge colors together and also to merge Fonts together for the main menu, such as making the font bold and italic at the same time. So for example 255 << 16 | 255 << 8 |255 is the same as merging FF0000, 00FF00, and 0000FF together into one integer, which basically means that each pixel shows all of the colors it’s made up from completely. Like for an example a purple pixel would show both its red and blue components. Now lets say I do 255 | 255 | 255 then a purple pixel would only show its blue components so a purple pixel would appear blue. This is how I make the environmental suit turn everything into a green shade and when you drown in tears how the screen turns all blue.

Basically its like in an if statement when you use || to compare two Boolean expressions, except for its with binary. 0 is false and 1 is true. So for each number you compare 0 || 1 or 0 || 0 etc… So it takes all the binary digits that are TRUE for both numbers, and merges them together.

& is the AND bitwise operator. This is kind of like the opposite of OR. It basically takes only the things in common with both binary numbers and makes a new binary number from that. Its basically like 0 && 0 or 1 && 1, or 1 && 0 etc if you were doing a normal if statement:

0011010 (26)

& 0101001 (41)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0001000 (8)

As seen here only the integers that match up completely are turned on for a number. This is how the game renders the texture files on screen. It figures out where the pixel should go on the wall (because of how the texture is stretched or not depending on how your looking at it) and then determines which pixel in the image file correlates with the pixel that it is requesting. So & does this by only rendering the pixels that match on the binary level.