Alpha Channel explained

Using Blender 2.76

To start with

Whatever color you see around you is made out of 3 and only 3 colors. Red – Green – Blue. Called, the ‘Basic Colors’. ( RGB for short ) Depending on how much of each color you mix with each other will give you a final certain color.

If I have three buckets of light, each containing an amount of the basic colors Red, Green and Blue, pour them together in a forth bucket and mix it all together it will become a certain color.

In this example the purple color is a mix of 70 % Red (of the buckets’ volume), 12% Green (of the buckets’ volume) and 85% Blue (of the buckets’ volume).

If we want to write this in a short way to give the purple color a number, referring to the basic RGB colors, we could do it like this :

Purple Color Number = 701285

So far so simple.
First things first: An image file

Basically, an image file is nothing more than a table with an X number of columns width, and an X number of rows height. It means that every cell of this table represents a pixel. If you scale an image to the extreme (In Photoshop or Gimp) you see the pixels appearing as squares. Every cell of an image table, or pixel, contains 1 number and 1 number only because a pixel can have only 1 color. The number of this color, or data, is written in the same way as the buckets example, not in % but in hexadecimal or ‘HEX’ for short.

If you give an image file to Blender it reads the number, or data, in every cell in the table, row by row, column by column and shows the corresponding color on the screen.
**Why and what is Hexadecimal?**

Hexadecimal is a convenient way to express binary numbers in computers. In which bytes are used and a basic color has a byte at its disposal.

In our decimal system we have ten numbers or digits. (deci means 10)

\[0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9\]

A basic color has a byte at its disposal and a byte is always defined as containing eight binary digits.

If I say in English a hundred, you know that I am talking of a number with 3 digits, XXX. If I say a thousand you know it is a number containing 4 digits, XXXX. The same goes for computers, a byte is defined as containing 8 digits.

**A byte = XXXXXXXX (8 times)**

The smallest number in our decimal system with 8 digits is 00000000 and the biggest number is 99999999. To calculate the number of different combinations possible we use the formula:

\[10^8\]

10 to the power 8. (\(10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10\), 8 times) 10 Because there are 10 different numbers and 8 because there are 8 digits.

So that makes: \(10^8 = 1.000.000.000\) different combinations because 00000000 also is a combination.

A computer uses the binary system and that system knows only two numbers, a ‘0’ and a ‘1’ The reason for that is that a computer works with switches, believe it or not, but it has a lot of them, all packed into a chip, and it turns them on or off extremely fast. A switch is on (1) or it is off (0).

The smallest number is 00000000 and the biggest number will be 11111111. (All switches are off, or all switches are on.) Calculated in the same way \(2^8\) (\(2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2\), 8 times. 2 Because there are only 2 numbers, and 8 because there are 8 digits) that makes 256 different combinations possible. As a result of that, the three basic colors (Red/Green/Blue) each can have 256 different values. From 0 to 255. As with the buckets, 0 = 0% and 255 = 100%

Let’s take an example, we fill our buckets again, but this time not with %, but with digital numbers.

<table>
<thead>
<tr>
<th>The value for red is</th>
<th>10101010</th>
<th>(combination 170 out of the 256 possibilities)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The value for green is</td>
<td>00011001</td>
<td>(combination 25 out of the 256 possibilities)</td>
</tr>
<tr>
<td>The value for blue is</td>
<td>11001101</td>
<td>(combination 205 out of the 256 possibilities)</td>
</tr>
</tbody>
</table>

As explained in the illustration above, a color can have only one number, so we place them all after each other, as with the buckets. That means a color mixed with these red, green and blue values will give this digital number:

\[\text{RGB number} = 101010100001100111001101\]

The computer knows exactly that the first 8 digits are the Red value, the middle 8 digits the Green value and the last 8 digits are the Blue value. But …

**How on earth are you ever going to write a number like this on a piece of paper to remember the kind of color you have used?**

We need a better way to write this down.
The system used is the ‘Hexadecimal’ system. HEXA means 6 and DECI 10. Therefore 16. The symbols will be:

\[0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - A - B - C - D - E - F\]  
(We need 16 symbols so we add letters.)

To replace an 8 digit number, XXXXXXXX containing of only two different digits, 1 and 0, with a 16 digit system, we need a number with only 2 digits XX. Because \(2^8 = 256\) and \(16^2\) also is 256.

Our binary color number 1010101000110011001101 converted to a HEX number will gives us \#AA19CD.

In this HEX combination are the first two digits the RED value, the middle two the GREEN and the last two the BLUE value.

Now I can take a piece of paper and write this number down.

*The # symbol is commonly used to indicate that what follows has to be interpreted as a number. FF can have different meanings, Fast Forward or 255. If you write #FF there is no doubt.*

You can find this number in Blender in every color wheel if you click on the HEX button. Let’s write our example HEX number \#AA19CD in the color wheel for the viewport color of our cube and we have a purple looking cube.

Note if you play with the color how this number changes. If you click on the textbox you don’t have to write the # symbol.

If you switch back to RGB, what is the default, Blender does not give the values of the basic colors from 0 to 255 but from 0 to 1 instead.

0 representing minimum (0) and 1 representing maximum (255). Why Blender does it this way is beyond the scoop of this document.

If you go to the color picker in Photoshop you will find these numbers back.
The minimum number in a hexadecimal system is #000000 (Black) and the maximum number is #FFFFFF (White). All F’s, because F is the biggest digit.

Remember? $0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - A - B - C - D - E - F$ (We need 16 symbols so we add letters.)

Here is our first ‘table’ illustration again. This should clear up why it is called ‘CHANNEL’

#AA19CD

Red Channel
Green Channel
Blue Channel
What is Transparency?

So far everything went fine and dandy, way back in time when computers still wore diapers. Steven Jobs could make his ‘Graphical Users Interface’ (GUI), no more command line wise guys stuff anymore. Icons could be shown on the screen and the user just had to click a button with the mouse. All made of bitmaps and images.

But the more computers were conquering the world, the more applications were written for it and of cause, software for photos, taken with digital cameras, found worldwide customers. Manipulating the pictures you have taken, an awesome idea.

But there was a small problem. What if I have an image of myself and I want to add something else in the picture ? I can lay it on top of my image but I don’t want the gray background of it, I only want the logo. You can’t just simply remove a pixel out of the image table. It means that the gray pixels have to be transparent, so the background shows through it.

In the system, so far explained, there is no way of doing that. A value is minimum #000000 and that is ‘Black’ or a value is maximum #FFFFFF and that is “White”. All the numbers in between, all give some kind of a color.

Now what?

We have to come up with a way to define that a pixel has no color data, it is transparent.

But there is more. What if I want my logo to have a shadow that fades out ? And a little shine effect too perhaps ? Whether or not it has a color is not enough, we have to give it a value. How much transparency a pixel has. Or, how much the background should peep through.

If you open our logo, with its shadow and glare effect, in the ‘UV/Image editor’ in Blender you see this.

Of course, Blender can not show trasparency on the screen, or you would have a hole in your monitor, so it solves it with a checker board like background.

As you see, at the borders it is 100% transparent. But the more it goes towards the logo it becomes more 100% opaque or 0% transparent.

The glare and shadow fades out towards the borders.

It means that we need something like minimum and maximum and all the values in between.
At last, the ALPHA CHANNEL

We already gave the colors, Red/Green/Blue this minimum and maximum value. We gave it a byte at their disposal with values variable from 0 (minimum) to 255 (maximum).

Remember the HEX number? #XXXXXX

So what stops us from giving the total color number one more byte? #XXXXXX + XX = #XXXXXXX

Again we can give extra values from 0 to 255 to every pixel in our bitmap table and assign it to something. How about the ‘Alpha Channel’

But ... ... wait ... ... if everything has to do with transparency, than why on earth do they call it the ‘Alpha Channel’ and not the ‘Transparency Channel’?

Well ... ...  

If you really want an answer to that question you’d better ask Mister Alvy Ray Smith, Ed Catmull, Tom Duff or Tom Porter for it. Because they have invented it and received the technical academy award on March 1996 from the Academy of Motion Picture Arts and Science for “pioneering inventions in digital image compositing”. They got an Oscar award for inventing the ‘Alpha channel’.

We came all the way to say in a few lines what the alpha channel actually is.

So, we have a value for the transparency of every pixel from 0 (100% transparent) to 255 (100% opaque or 0% transparency). But this value has no color, it is just a value how much the background should shine through. So how do we make it visible. Easy, make 0 Black and 255 White and all the values in between are shades of gray.

If you have an image that has an alpha channel and you want to see it, (represented as Black and White) open your image in Blenders ‘UV/Image editor’, click the ‘Alpha’ button and the alpha channel of our logo looks like this:

As you can see, the logo itself is fully white or 0% transparent, and the more it fades out to the border it turns to complete black or 100% transparent.
You have to be careful though, not every image file has the capability to contain an Alpha Channel and not all image files that do contain an alpha channel have the values from 0 to 255.

Very understandable, it has all to do with history.

At first, way back in time when computers still wore diapers, all there was, was RGB. The very first bitmap file was called BMP for Windows and DIB for Mac.

The next step towards the Alpha channel was, is there transparency, yes or no, no values in between. GIF was one of the first bitmap files that had an Alpha channel, but it was a simple yes or no, a 0 or a 1.

The next step was to give the alpha channel fully gradable values from 0 to 255 and one of those files is a PNG.

To find out if a file extension has the capability to contain an Alpha Channel is to go to the ‘Output’ menu in the ‘Render’ tab. There you can select differed file types to save your work in. If it has the capability to contain an Alpha channel, an option box appears with RGBA (Red/Green/Blue/Alpha). As you can see, a JPEG file does not have an Alpha Channel, there is no RGBA button available. (BW = Black/White)

If there is no transparency in your image, there is no need to save it as RGBA, click the RGB button instead. As you know, an RGBA pixel has this color data #XXXXXXXX. But an RGB pixel has only this color data : #XXXXXX, it has a byte less.

An image file does not only contain the table data for the pixels, it has a header too. In this header is written how the file has to be read, whether or not a program should take the alpha channel in account when it reads the file. Therefore, if Blender writes the file, it does not write that extra alpha XX byte when it saves (If you select the RGB button), and writes that information into the header of that file. (Listen up guys, there is no alpha information in this file). It means that the file size is smaller, it does not contain that extra alpha byte.

This image saved as RGBA PNG has a size of 389 kB.

Saved as RGB PNG it is only 249 kB

The difference of 140 kb doesn’t seems to be much. But if you have an animation with 1.000 frames, it is 140 MB.

If you load the final rendered image sequence of an animation of 1.000 frames into the ‘Video Sequence Editor’, to turn it into a video file, Blender is dealing with 140 MB useless data that only slows the performance down.

So, if there is no transparency in your work, you’d better just save it as RGB only.
Some examples.

Now that we know everything about the alpha channel, how can we use it?

Example 1

In this example I have used a simple plane to hold the logo. In the top left window you can see that the logo image has an alpha channel. (The checker board background.) We need two shaders, one for the logo, we use a diffuse shader for that, and one to make the background transparent, so that will be a transparent shader. We mix them together and use the alpha channel output of the image as the mix factor.

How does this work?

A mix shader mixes two inputs depending on a factor. If the factor is set to 1, whatever goes into the bottom input comes 100% out. If the factor is set to 0, whatever goes into the top input comes 100% out. If the factor is set to 0.5, both inputs come out 50%.

So the more the factor value goes to 1, more of the bottom and lesser of the top comes out and of course, the more the factor value goes to 0, the more of the top comes out and lesser from the bottom.
Remember the HEX number for our purple color \( \# \text{AA19CD} \) ?

Because our purple color is fully opaque, (0 % transparent) the number for the alpha channel will be maximum or \( \# \text{FF} \) (white). So the total RGBA number will be: \( \# \text{AA19CDFF} \)

An image texture node separates the RGB channels from the alpha channel and gives it their corresponding outputs. As told in the beginning of this document, Blender reads an image file pixel by pixel (row by row, column by column). That means that pixel by pixel goes through the mix node.

In HEX the value for the alpha channel goes from minimum \( \#00 \) to maximum \( \#\text{FF} \)

In our decimal system it is from 0 to 255.

But Blender converts it from 0 to 1 (0% - 100 %)

Because the alpha value in our purple example is maximum \( \#\text{FF} \), the mix factor in the mix shader node will be 1 and whatever goes in the bottom input comes 100 % out.

The lesser the value of the alpha byte, or the more it goes to 0 (the more black), the more transparency will be mixed.
Example 2

Now, this is a tricky one.

What if we have an image with no alpha channel but a black background?

What we can do is to use the black color to determine the mix factor of the mix shader node.

First of all we have to determine the black color. We use a color mix node for that. It mixes color 1 with color 2. Make the mix factor 1. It works the same as a mix shader node. If you make this FAC 1, 100% of color 2 comes out regardless the color of color 1.

As you know, an alpha channel has no color, it is a value. The value for black is #00, it is absolutely 0. Therefore, every number higher than 0 should NOT be transparent. We use a math node and select the option ‘Greater Than’. The greater than option tests if input 1 is greater than input 2, returns 1 for true, 0 for false. So we have two possible values for the output of this math node, 1 or 0. That result goes into the factor input of our mix node. Because we have no alpha channel we use the same RGB output of the image texture node for the math comparator.

By tweaking a bit with color 2 we can adjust the level of blackness until we are happy with the result. If you set the viewport to render you can watch your tweaking in real time.
Unfortunately, whatever is black will become transparent.

There are a few different ways to make black transparent, depending on the kind of image you have, this is one way you can do it. If the subject of the image has no absolute black in it, this is a cool way.
Example 3

An alpha channel is in fact a Black and White image map inside an image map. So, what stops us from making our own B/W image and use it as an alpha channel?

That is exactly what I have done for the illustration at the very beginning of this document.

Of course I made the total illustration with different files, different render layers for the shadow and joined it all together in Photoshop and added some text.

Let us take only this part.

In Photoshop I made a black to white gradient image.

In the compositor I imported it with an ‘Image Node’

First I have to cheat the actual alpha channel.

I have rendered with the ‘Transparent’ option selected in the render tab.

Mix the gradient B/W image with the render layer, using a color mix node, and use the alpha output of the render layer as the mix factor.

If you want to see the alpha output of the render layer, you plug it in the image input of the composite node. Whatever is the transparent background will be black and whatever is the character will be white.

In the same way, if I plug the output of the mix node into the image input of the composite node I can see the result.

And the result is the fake alpha channel.
Plug the render layer into the image input and the fake alpha channel into the alpha input of the composite node than is this the result.
THANK YOU MY LITTLE SISTER. NOW I CAN WALK AGAIN.

GOT IT!