42 Docs

Colors

TABLE OF CONTENTS

Colors are presented in an int format. It therefore requires some tricky things in order to obtain an int which can contain the ARGB values.

The color integer standard

We shift bits to use the TRGB format. To define a color, we initialize it as follows: `0xTTRRGGBB`, where each character represents the following:

Letter	Description
`T`	transparency
`R`	red component
`G`	green component
`B`	blue component

RGB colors can be initialized as above, a few examples would be:

Color	TRGB representation
red	`0x00FF0000`

Color	TRGB representation
green	`0x0000FF00`
blue	`0x000000FF`

Encoding and decoding colors

We can use two methods to encode and decode colors:

- BitShifting
- char/int conversion

BitShifting

Since each byte contains `2^8 = 256` values (1 byte = 8 bits), and RGB values range from 0 to 255, we can perfectly fit a integer (as an int is 4 bytes). In order to set the values programatically we use `bitshifting`. Let's create a function which does precisely that for us, shall we?

```
int	create_trgb(int t, int r, int g, int b)
{
	return (t << 24 | r << 16 | g << 8 | b);
}
```

Because ints are stored from right to left, we need to bitshift each value the according amount of bits backwards. We can also do the exact opposite and retrieve integer values from a encoded TRGB integer.

```
int	get_t(int trgb)
{
	return ((trgb >> 24) & 0xFF);
}

int	get_r(int trgb)
{
	return ((trgb >> 16) & 0xFF);
}
```

```
int	get_g(int trgb)
{
	return ((trgb >> 8) & 0xFF);
}

int	get_b(int trgb)
{
	return (trgb & 0xFF);
}
```

Char/int conversion

Since each byte contains `2^8 = 256` values (1 byte = 8 bits), and RGB values range from 0 to 255, we can perfectly fit a `unsigned char` for each TRGB parameters `{T, R, G, B}` (char is 1 byte) and fit a `int` for the TRGB value (int is 4 bytes). In order to set the values programatically we use type converting.

```
int	create_trgb(unsigned char t, unsigned char r, unsigned char g, unsigned char b)
{
	return (*(int *)(unsigned char [4]){b, g, r, t});
}

unsigned char	get_t(int trgb)
{
	return (((unsigned char *)&trgb)[3]);
}

unsigned char	get_r(int trgb)
{
	return (((unsigned char *)&trgb)[2]);
}

unsigned char	get_g(int trgb)
{
	return (((unsigned char *)&trgb)[1]);
}
```

```
unsigned char   get_b(int trgb)
{
      return (((unsigned char *)&trgb)[0]);
}
```

To understand the conversion you can refere to the table bellow, where `0x0FAE1` is the variable address of `int trgb`.

Address	char	int
0x0FAE1	unsigned char b	int trgb
0x0FAE2	unsigned char g	[allocated]
0x0FAE3	unsigned char r	[allocated]
0x0FAE4	unsigned char t	[allocated]

Test your skills!

Now that you understand the basics of how the colors can be initialized, get comfy and try creating the following color manipulation functions:

`add_shade` is a function that accepts a double (distance) and a int (color) as arguments, 0 will add no shading to the color, whilst 1 will make the color completely dark. 0.5 will dim it halfway, and .25 a quarter way. You get the point.

`get_opposite` is a function that accepts a int (color) as argument and that will invert the color accordingly.