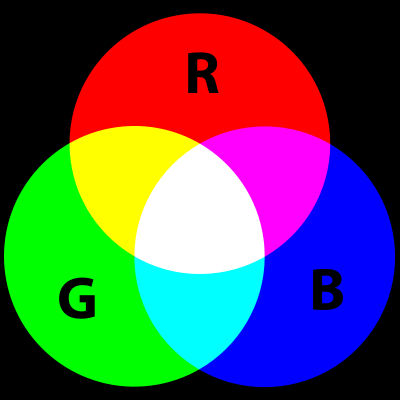
**BASIC MODEL COLOR AND MAT IN OPENCV**

1. **The basic model color of image:**
   1. **RGB color image:**

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

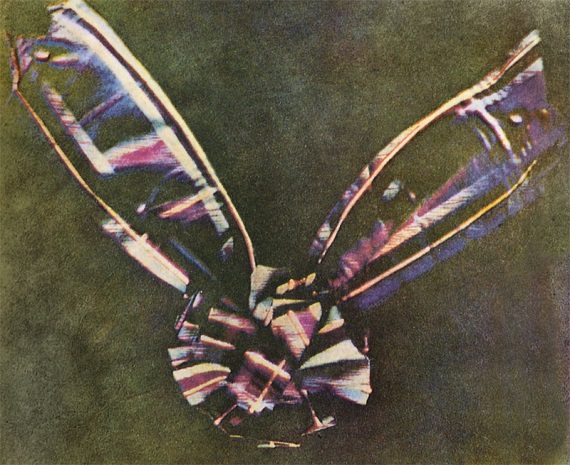


Nowaday, the RGB color model used very popular in television, computer screen, mobile devices.

* + 1. **History of RGB color model:**

The RGB color model is developed by **Thomas Young** and **Hermann Helmholtz** in the early to mid nineteenth century, and on **James Clerk Maxwell'**s color triangle that elaborated that theory (circa 1860).

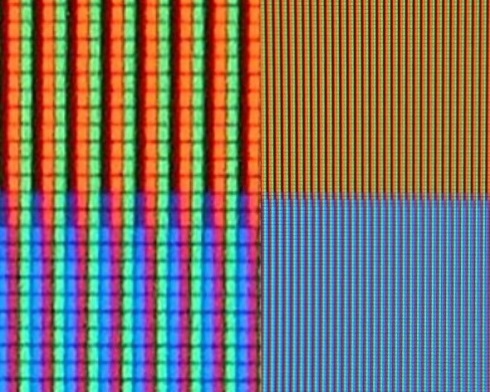
The first experiments’s Maxwell about RGB color model were made in 1861. He involved the process of combining three color-filtered separate takes. To reproduce the color photograph, three matching projections over a screen in a dark room were necessary.



Before the development of practical electronic TV, there were patents on mechanically scanned color systems as early as 1889 in Russia. The color TV pioneer John Logie Baird demonstrated the world's first RGB color transmission in 1928, and also the world's first color broadcast in 1938, in London. In his experiments, scanning and display were done mechanically by spinning colorized wheels. The Columbia Broadcasting System (CBS) began an experimental RGB field-sequential color system in 1940. Images were scanned electrically, but the system still used a moving part: the transparent RGB color wheel rotating at above 1,200 rpm in synchronism with the vertical scan. The camera and the cathode-ray tube (CRT) were both monochromatic. Color was provided by color wheels in the camera and the receiver. More recently, color wheels have been used in field-sequential projection TV receivers based on the Texas Instruments monochrome DLP imager.The modern RGB shadow mask technology for color CRT displays was patented by Werner Flechsig in Germany in 1938.

Early personal computers of the late 1970s and early 1980s, such as those from Apple, Atari and Commodore, did not use RGB as their main method to manage colors, but rather composite video. IBM introduced a 16-color scheme (four bits—one bit each for red, green, blue, and intensity) with the Color Graphics Adapter (CGA) for its first IBM PC (1981), later improved with the Enhanced Graphics Adapter (EGA) in 1984. The first manufacturer of a truecolor graphic card for PCs (the TARGA) was Truevision in 1987, but it was not until the arrival of the Video Graphics Array (VGA) in 1987 that RGB became popular, mainly due to the analog signals in the connection between the adapter and the monitor which allowed a very wide range of RGB colors. Actually, it had to wait a few more years because the original VGA cards were palette-driven just like EGA, although with more freedom than VGA, but because the VGA connectors were analogue, later variants of VGA (made by various manufacturers under the informal name Super VGA) eventually added truecolor. In 1992, magazines heavily advertised truecolor Super VGA hardware.

* + 1. **RGB devices:**

****

RGB sub-pixels in an LCD TV

RGB color model can show image with a color in nature so, RGB color model used very popular in television, computer screen, mobile device.



* 1. **Grayscale**

Grayscale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest.

Grayscale images are often the result of measuring the intensity of light at each pixel in a single band of the electromagnetic spectrum (e.g. infrared, visible light, ultraviolet, etc.), and in such cases they are monochromatic proper when only a given frequency is captured. But also they can be synthesized from a full color image; see the section about converting to grayscale.



Nowaday, in computing, we have 2 type of grayscale is 8 bits of each pixel (255 levels) and 16 bits of each pixel (65,535 levels).

* 1. **Binary color model:**

A binary image is a digital image that has only two possible values for each pixel. Typically the two colors used for a binary image are black and white though any two colors can be used.

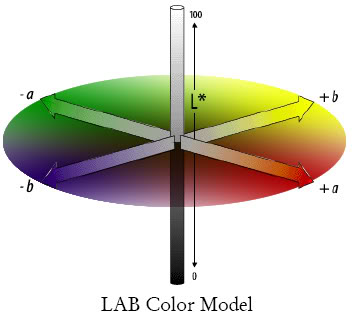
So, what is difference between grayscale and binary image?

Black and White image contains only two levels. Gray image represent by black and white shades or combination of levels for e.g 8 bit gray image means total 28 - 1 = 255 levels form black to white.



* 1. **LAB color model:**

The 'Lab' colour space is one of the least used image formats as no print production, picture library, or website would ever consider using it. And yet, ironically, everyone does. 'Lab' covers all potential colours, every colour that the human eye is able to see. As such it is an absolute and therefore is not colour managed as there are no degrees of 'Lab' for a colour profile to describe. Because of this Photoshop uses it as its native colour space. So every time an image is converted from RGB to CMYK, or back again, it is passed through 'Lab' in the process.

****

In LAB mode, the sample is represented by a combination of three channels:

* L (lightness - Luminance): Channel L is the vertical axis, shows the brightness of color, with a value of 0 (black) to 100 (White). This channel is solely contains information about brightness, color does not contain real value.
* Channel "a": Contains the value of color from Green (-) to the Red (+).
* Channel "b": Contains the value of color from Blue (-) to Yellow (+).

1. **Mat in OpenCV:**

Mat in OpenCV is a class with an n-dimensional dense numerical single-channel or multi-channel array.

|  |  |
| --- | --- |
| **Parameters:** | * **ndims** – Array dimensionality. * **rows** – Number of rows in a 2D array. * **cols** – Number of columns in a 2D array. * **roi** – Region of interest. * **size** – 2D array size: Size(cols, rows) . In the Size() constructor, the number of rows and the number of columns go in the reverse order. * **sizes** – Array of integers specifying an n-dimensional array shape. * **type** – Array type. Use CV\_8UC1, ..., CV\_64FC4 to create 1-4 channel matrices, or CV\_8UC(n), ..., CV\_64FC(n) to create multi-channel (up to CV\_CN\_MAX channels) matrices. * **s** – An optional value to initialize each matrix element with. To set all the matrix elements to the particular value after the construction, use the assignment operator Mat::operator=(const Scalar& value) . * **data** – Pointer to the user data. Matrix constructors that take data and step parameters do not allocate matrix data. Instead, they just initialize the matrix header that points to the specified data, which means that no data is copied. This operation is very efficient and can be used to process external data using OpenCV functions. The external data is not automatically deallocated, so you should take care of it. * **step** – Number of bytes each matrix row occupies. The value should include the padding bytes at the end of each row, if any. If the parameter is missing (set to AUTO\_STEP ), no padding is assumed and the actual step is calculated as cols\*elemSize(). * **steps** – Array of ndims-1 steps in case of a multi-dimensional array (the last step is always set to the element size). If not specified, the matrix is assumed to be continuous. * **m** – Array that (as a whole or partly) is assigned to the constructed matrix. No data is copied by these constructors. Instead, the header pointing to m data or its sub-array is constructed and associated with it. The reference counter, if any, is incremented. So, when you modify the matrix formed using such a constructor, you also modify the corresponding elements of m . If you want to have an independent copy of the sub-array, use Mat::clone() . * **img** – Pointer to the old-style IplImage image structure. By default, the data is shared between the original image and the new matrix. But when copyData is set, the full copy of the image data is created. * **vec** – STL vector whose elements form the matrix. The matrix has a single column and the number of rows equal to the number of vector elements. Type of the matrix matches the type of vector elements. The constructor can handle arbitrary types, for which there is a properly declared [DataType](http://docs.opencv.org/modules/core/doc/basic_structures.html#DataType) . This means that the vector elements must be primitive numbers or uni-type numerical tuples of numbers. Mixed-type structures are not supported. The corresponding constructor is explicit. Since STL vectors are not automatically converted to Mat instances, you should write Mat(vec) explicitly. Unless you copy the data into the matrix ( copyData=true ), no new elements will be added to the vector because it can potentially yield vector data reallocation, and, thus, the matrix data pointer will be invalid. * **copyData** – Flag to specify whether the underlying data of the STL vector or the old-style CvMat or IplImage should be copied to (true) or shared with (false) the newly constructed matrix. When the data is copied, the allocated buffer is managed using Mat reference counting mechanism. While the data is shared, the reference counter is NULL, and you should not deallocate the data until the matrix is not destructed. * **rowRange** – Range of the m rows to take. As usual, the range start is inclusive and the range end is exclusive. Use Range::all() to take all the rows. * **colRange** – Range of the m columns to take. Use Range::all() to take all the columns. * **ranges** – Array of selected ranges of m along each dimensionality. |

Parameters “**type**” decided on the type of data stored by image.4

A Mapping of Type to Numbers in OpenCV

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | C1 | C2 | C3 | C4 |
| CV\_8U |  | 0 | 8 | 16 | 24 |
| CV\_8S |  | 1 | 9 | 17 | 25 |
| CV\_16U |  | 2 | 10 | 18 | 26 |
| CV\_16S |  | 3 | 11 | 19 | 27 |
| CV\_32S |  | 4 | 12 | 20 | 28 |
| CV\_32F |  | 5 | 13 | 21 | 29 |
| CV\_64F |  | 6 | 14 | 22 | 30 |

The naming sheme for the types is CV\_<bit-depth>{U|S|F}C<number\_of\_channels>.

EX:

* CV\_8UC4 translates to: four channels of unsigned char
* CV\_16S translates to: 1 channel of signed 2-byte integer.