# TColor Class Reference



The color creation and management class.



## Introduction

Colors are defined by their red, green and blue components, simply called the RGB components. The colors are also known by the hue, light and saturation components also known as the HLS components. When a new color is created the components of both color systems are computed.

At initialization time, a table of colors is generated. An existing color can be retrieved by its index:

```
TColor *color = gROOT->GetColor(10);
```

Then it can be manipulated. For example its RGB components can be modified:

```
color->SetRGB(0.1, 0.2, 0.3);
```

A new color can be created the following way:

```
Int_t ci = 1756; // color index
TColor *color = new TColor(ci, 0.1, 0.2, 0.3);
```

**Since**

**6.07/07:** **TColor::GetFreeColorIndex()** allows to make sure the new color is created with an unused color index:

```
Int_t ci = TColor::GetFreeColorIndex();
TColor *color = new TColor(ci, 0.1, 0.2, 0.3);
```

Two sets of colors are initialized;

- The basic colors: colors with index from 0 to 50.
- The color wheel: colors with indices from 300 to 1000.

## Basic colors

The following image displays the 50 basic colors.

```
{
   TCanvas *c = new TCanvas("c","Fill Area colors",0,0,500,200);
   c->DrawColorTable();
   return c;
}
```


## The color wheel

The wheel contains the recommended 216 colors to be used in web applications.

The colors in the color wheel are created by **`TColor::CreateColorWheel`**.

Using this color set for your text, background or graphics will give your application a consistent appearance across different platforms and browsers.

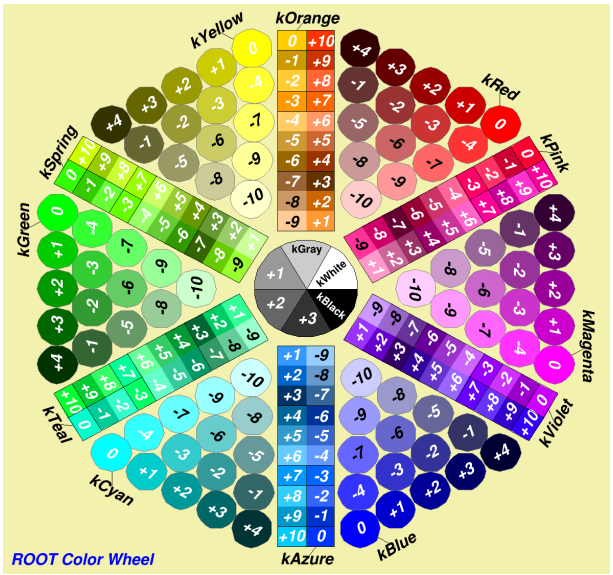
Colors are grouped by hue, the aspect most important in human perception. Touching color chips have the same hue, but with different brightness and vividness.

Colors of slightly different hues clash. If you intend to display colors of the same hue together, you should pick them from the same group.

Each color chip is identified by a mnemonic (e.g. kYellow) and a number. The keywords, kRed, kBlue, kYellow, kPink, etc are defined in the header file **Rtypes.h** that is included in all **ROOT** other header files. It is better to use these keywords in user code instead of hardcoded color numbers, e.g.:

```
myObject.SetFillColor(kRed);
myObject.SetFillColor(kYellow-10);
myLine.SetLineColor(kMagenta+2);
```

```
{
   TColorWheel *w = new TColorWheel();
   cw = new TCanvas("cw","cw",0,0,400,400);
   w->SetCanvas(cw);
   w->Draw();
}
```




The complete list of predefined color names is the following:

```
kWhite  = 0,   kBlack  = 1,   kGray    = 920,  kRed    = 632,  kGreen  = 416,
kBlue   = 600, kYellow = 400, kMagenta = 616,  kCyan   = 432,  kOrange = 800,
kSpring = 820, kTeal   = 840, kAzure   =  860, kViolet = 880,  kPink   = 900
```

Note the special role of color `kWhite` (color number 0). It is the default background color also. For instance in a PDF or PS files (as paper is usually white) it is simply not painted. To have a white color behaving like the other color the simplest is to define an other white color not attached to the color index 0:

```
Int_t ci = TColor::GetFreeColorIndex();
```

```
TColor *color = new TColor(ci, 1., 1., 1.);
```

## Bright and dark colors

The dark and bright color are used to give 3-D effects when drawing various boxes (see **TWbox**, **TPave**, **TPaveText**, **TPaveLabel**, etc).

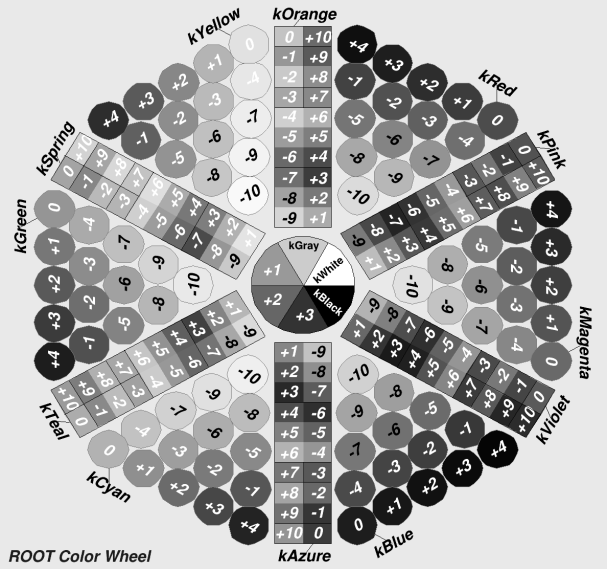
- The dark colors have an index = color_index+100
- The bright colors have an index = color_index+150
- Two static functions return the bright and dark color number corresponding to a color index. If the bright or dark color does not exist, they are created:

```
Int_t dark   = TColor::GetColorDark(color_index);
Int_t bright = TColor::GetColorBright(color_index);
```

## Grayscale view of of canvas with colors

One can toggle between a grayscale preview and the regular colored mode using **`TCanvas::SetGrayscale()`**. Note that in grayscale mode, access via RGB will return grayscale values according to ITU standards (and close to b&w printer gray-scales), while access via HLS returns de-saturated gray-scales. The image below shows the **ROOT** color wheel in grayscale mode.

```
{
   TColorWheel *w = new TColorWheel();
   cw = new TCanvas("cw","cw",0,0,400,400);
   cw->GetCanvas()->SetGrayscale();
   w->SetCanvas(cw);
   w->Draw();
}
```



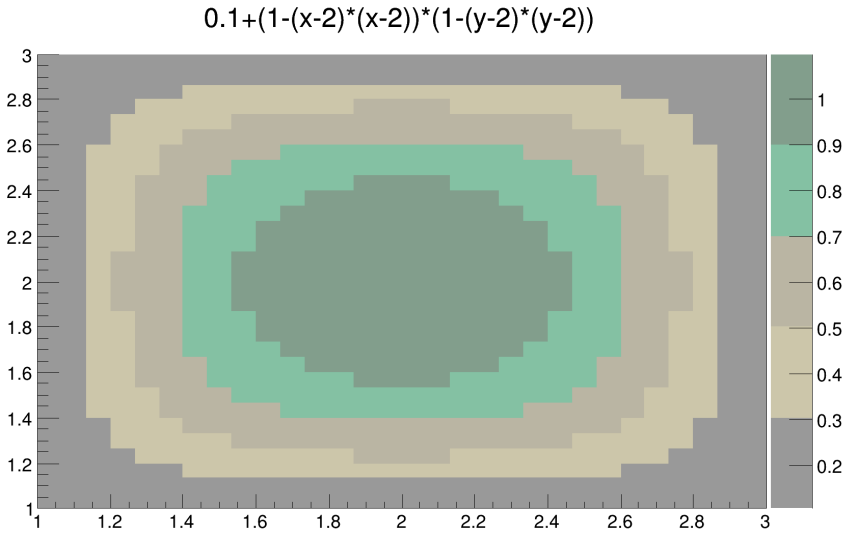

## Color palettes

It is often very useful to represent a variable with a color map. The concept of "color palette" allows to do that. One color palette is active at any time. This "current palette" is set using:

```
gStyle->SetPalette(...);
```

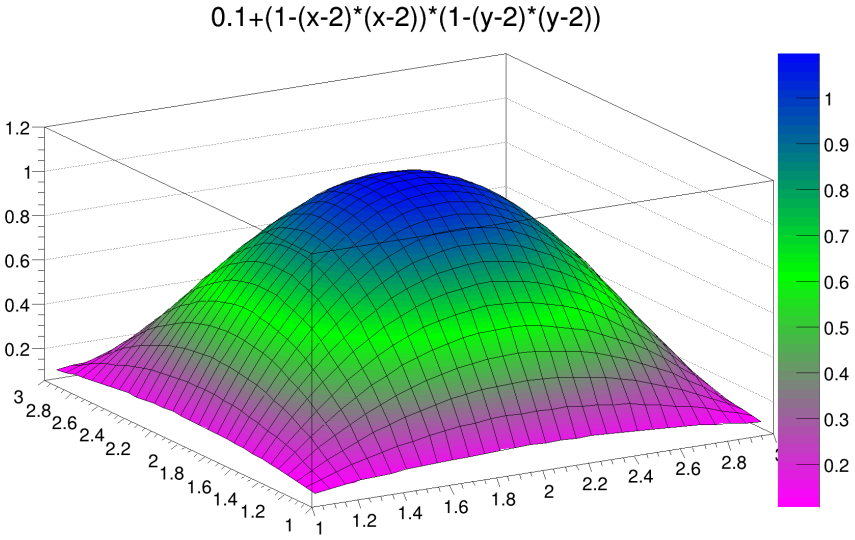
This function has two parameters: the number of colors in the palette and an array of containing the indices of colors in the palette. The following small example demonstrates how to define and use the color palette:

```
{
   TCanvas *c1  = new TCanvas("c1","c1",0,0,600,400);
   TF2 *f1 = new TF2("f1","0.1+(1-(x-2)*(x-2))*(1-(y-2)*(y-2))",1,3,1,3);
   Int_t palette[5];
   palette[0] = 15;
   palette[1] = 20;
   palette[2] = 23;
   palette[3] = 30;
   palette[4] = 32;
   gStyle->SetPalette(5,palette);
   f1->Draw("colz");
   return c1;
}
```




To define more a complex palette with a continuous gradient of color, one should use the static function **`TColor::CreateGradientColorTable()`**. The following example demonstrates how to proceed:

```
{
   TCanvas *c2  = new TCanvas("c2","c2",0,0,600,400);
   TF2 *f2 = new TF2("f2","0.1+(1-(x-2)*(x-2))*(1-(y-2)*(y-2))",1,3,1,3);
   const Int_t Number = 3;
   Double_t Red[Number]    = { 1.00, 0.00, 0.00};
   Double_t Green[Number]  = { 0.00, 1.00, 0.00};
   Double_t Blue[Number]   = { 1.00, 0.00, 1.00};
   Double_t Length[Number] = { 0.00, 0.50, 1.00 };
   Int_t nb=50;
   TColor::CreateGradientColorTable(Number,Length,Red,Green,Blue,nb);
   f2->SetContour(nb);
   f2->SetLineWidth(1);
   f2->SetLineColor(kBlack);
   f2->Draw("surf1z");
   return c2;
}
```




The function **`TColor::CreateGradientColorTable()`** automatically calls `gStyle->`**`SetPalette()`**, so there is not need to add one.

After a call to **`TColor::CreateGradientColorTable()`** it is sometimes useful to store the newly create palette for further use. In particular, it is recommended to do if one wants to switch between several user define palettes. To store a palette in an array it is enough to do:

```
Int_t MyPalette[100];
Double_t Red[]    = {0., 0.0, 1.0, 1.0, 1.0};
Double_t Green[]  = {0., 0.0, 0.0, 1.0, 1.0};
Double_t Blue[]   = {0., 1.0, 0.0, 0.0, 1.0};
```

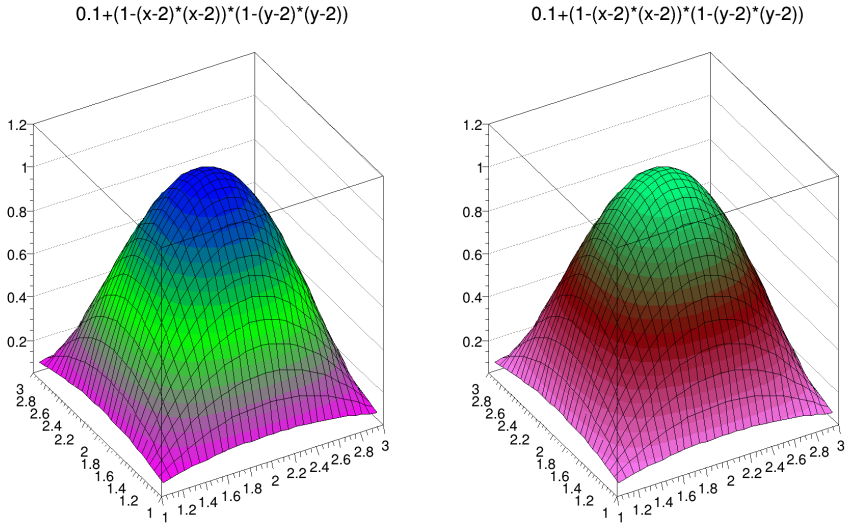
```
Double_t Length[] = {0., .25, .50, .75, 1.0};
Int_t FI = TColor::CreateGradientColorTable(5, Length, Red, Green, Blue, 100);
for (int i=0;i<100;i++) MyPalette[i] = FI+i;
```

Later on to reuse the palette `MyPalette` it will be enough to do

```
gStyle->SetPalette(100, MyPalette);
```

As only one palette is active, one need to use **TExec** to be able to display plots using different palettes on the same pad. The tutorial **multipalette.C** illustrates this feature.

```
#include "TStyle.h"
#include "TColor.h"
#include "TF2.h"
#include "TExec.h"
#include "TCanvas.h"

void Pal1()
{
   static Int_t  colors[50];
   static Bool_t initialized = kFALSE;

   Double_t Red[3]    = { 1.00, 0.00, 0.00};
   Double_t Green[3]  = { 0.00, 1.00, 0.00};
   Double_t Blue[3]   = { 1.00, 0.00, 1.00};
   Double_t Length[3] = { 0.00, 0.50, 1.00 };

   if(!initialized){
      Int_t FI = TColor::CreateGradientColorTable(3,Length,Red,Green,Blue,50);
      for (int i=0; i<50; i++) colors[i] = FI+i;
      initialized = kTRUE;
      return;
   }
   gStyle->SetPalette(50,colors);
}

void Pal2()
{
   static Int_t  colors[50];
   static Bool_t initialized = kFALSE;

   Double_t Red[3]    = { 1.00, 0.50, 0.00};
   Double_t Green[3]  = { 0.50, 0.00, 1.00};
   Double_t Blue[3]   = { 1.00, 0.00, 0.50};
   Double_t Length[3] = { 0.00, 0.50, 1.00 };

   if(!initialized){
      Int_t FI = TColor::CreateGradientColorTable(3,Length,Red,Green,Blue,50);
      for (int i=0; i<50; i++) colors[i] = FI+i;
      initialized = kTRUE;
      return;
   }
   gStyle->SetPalette(50,colors);
}

void multipalette() {
   TCanvas *c3  = new TCanvas("c3","C3",0,0,600,400);
   c3->Divide(2,1);
   TF2 *f3 = new TF2("f3","0.1+(1-(x-2)*(x-2))*(1-(y-2)*(y-2))",1,3,1,3);
   f3->SetLineWidth(1);
   f3->SetLineColor(kBlack);

   c3->cd(1);
   f3->Draw("surf1");
   TExec *ex1 = new TExec("ex1","Pal1();");
   ex1->Draw();
   f3->Draw("surf1 same");

   c3->cd(2);
   f3->Draw("surf1");
   TExec *ex2 = new TExec("ex2","Pal2();");
   ex2->Draw();
   f3->Draw("surf1 same");
}
```




**Since**

**6.26:** The function **`TColor::CreateColorTableFromFile("filename.txt")`** allows you to create a color palette based on an input ASCII file. In contrast to **`TColor::CreateGradientColorTable()`**, here the length (spacing) is constant and can not be tuned. There is no gradient being interpolated between adjacent colors. The palette will contain the exact colors stored in the file, that comprises one line per color in the format "r g b" as floats.

## High quality predefined palettes

**Since**

**6.04:** 63 high quality palettes are predefined with 255 colors each.

These palettes can be accessed "by name" with `gStyle->SetPalette(num)`. `num` can be taken within the following enum:

```
kDeepSea=51,            kGreyScale=52,     kDarkBodyRadiator=53,
kBlueYellow= 54,        kRainBow=55,       kInvertedDarkBodyRadiator=56,
kBird=57,               kCubehelix=58,     kGreenRedViolet=59,
kBlueRedYellow=60,      kOcean=61,         kColorPrintableOnGrey=62,
kAlpine=63,             kAquamarine=64,    kArmy=65,
kAtlantic=66,           kAurora=67,        kAvocado=68,
kBeach=69,              kBlackBody=70,     kBlueGreenYellow=71,
kBrownCyan=72,          kCMYK=73,          kCandy=74,
kCherry=75,             kCoffee=76,        kDarkRainBow=77,
kDarkTerrain=78,        kFall=79,          kFruitPunch=80,
kFuchsia=81,            kGreyYellow=82,    kGreenBrownTerrain=83,
kGreenPink=84,          kIsland=85,        kLake=86,
kLightTemperature=87,  kLightTerrain=88,  kMint=89,
kNeon=90,               kPastel=91,        kPearl=92,
kPigeon=93,             kPlum=94,          kRedBlue=95,
kRose=96,               kRust=97,          kSandyTerrain=98,
kSienna=99,             kSolar=100,        kSouthWest=101,
kStarryNight=102,       kSunset=103,       kTemperatureMap=104,
kThermometer=105,       kValentine=106,    kVisibleSpectrum=107,
kWaterMelon=108,        kCool=109,         kCopper=110,
kGistEarth=111,         kViridis=112,      kCividis=113
```

As explained in Crameri, F., Shephard, G.E. & Heron, P.J. The misuse of colour in science communication. Nat Commun 11, 5444 (2020) some color maps can visually distord data, specially for people with colour-vision deficiencies.

For instance one can immediately see the disadvantages of the Rainbow color map, which is misleading for colour-blinded people in a 2D plot (not so much in a 3D surfaces).

The `kCMYK` palette, is also not great because it's dark, then lighter, then half-dark again. Some others, like `kAquamarine`, have almost no contrast therefore it would be almost impossible (for a color blind person) to see something with a such palette.

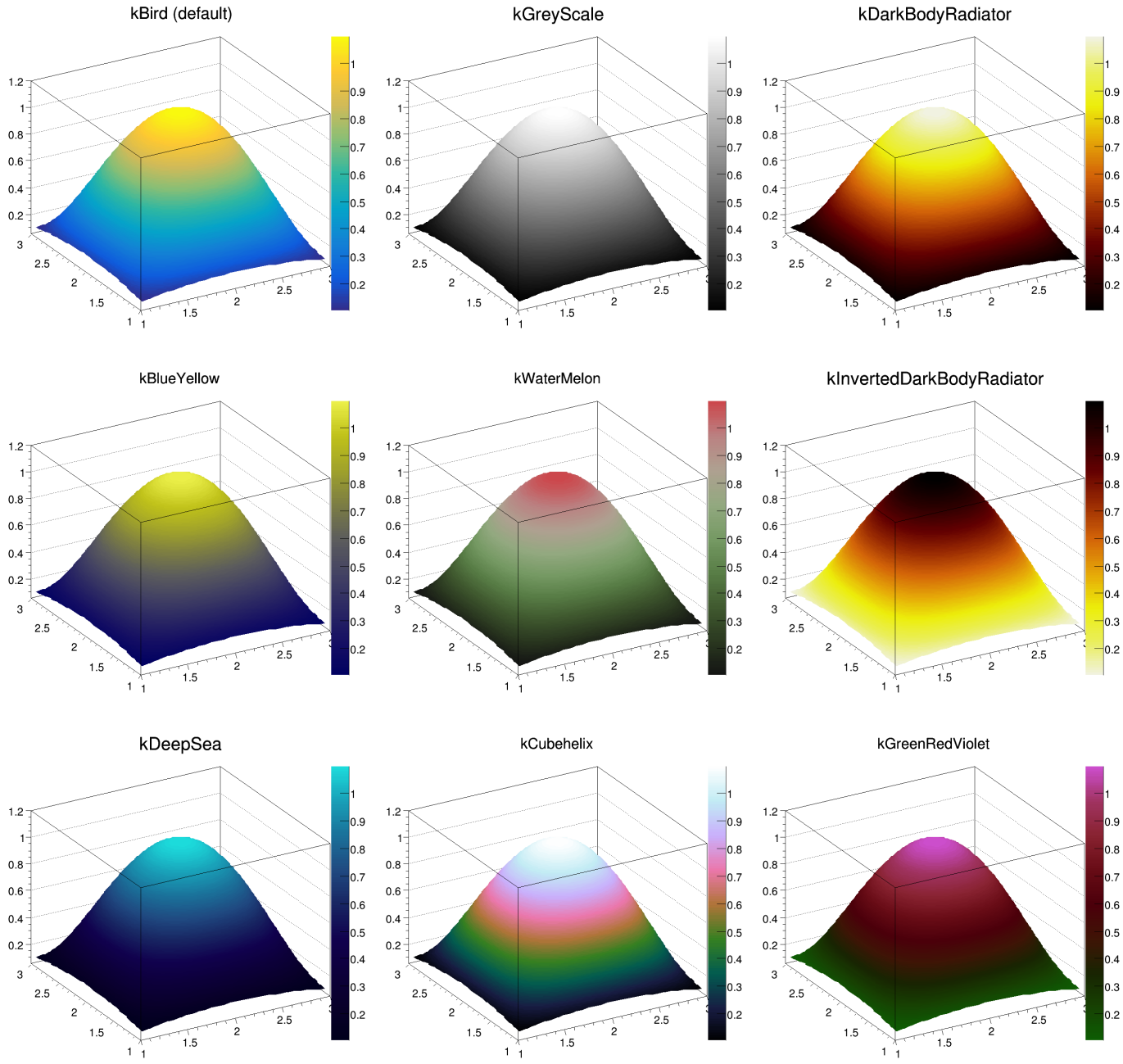
Therefore the palettes are classified in two categories: those which are Colour Vision Deficiency friendly and those which are not.

An easy way to classify the palettes is to turn them into grayscale using **`TCanvas::SetGrayscale()`**. The grayscale version of a palette should be as proportional as possible, and monotonously increasing or decreasing.

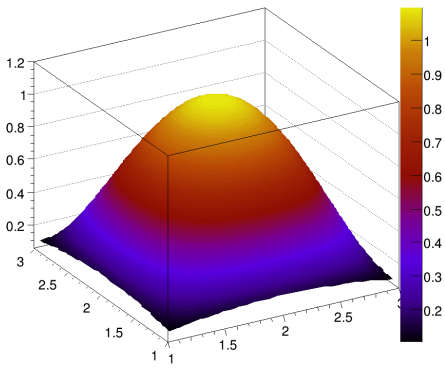
Unless it is symmetrical, then it is fine to have white in the borders and black in the centre (for example an axis that goes between -40 degrees and +40 degrees, the 0 has a meaning in the **perceptualcolormap.C** example).

A full set of colour-vision deficiency friendly and perceptually uniform colour maps can be downloaded and used with **ROOT** (since 6.26) via: `gStyle->SetPalette("filename.txt")` or `TColor::CreateColorTableFromFile("filename.txt")`. Remember to increase the number of contours for a smoother result, e.g.: `gStyle->SetNumberContours(99)` if you are drawing with "surf1z" or `gStyle->SetNumberContours(256)` if with "colz".

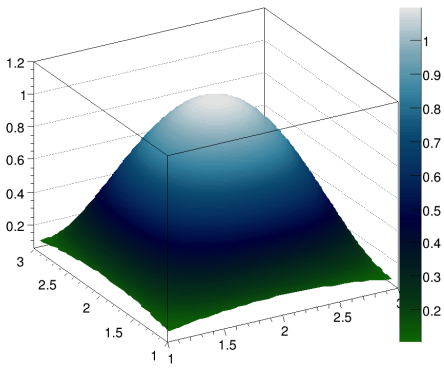
### Colour Vision Deficiency (CVD) friendly palettes



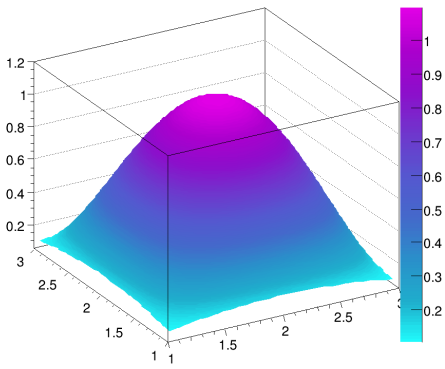
kBlueRedYellow



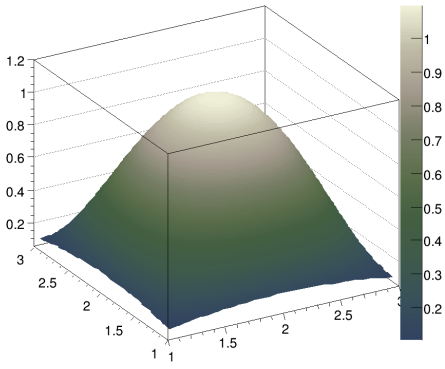
kOcean



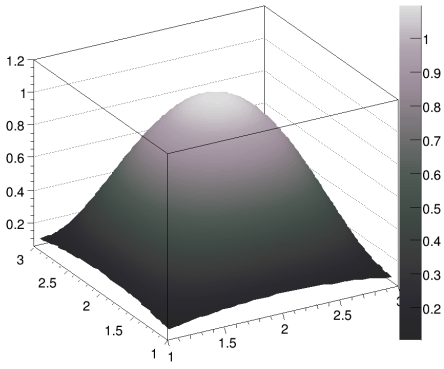
kCool



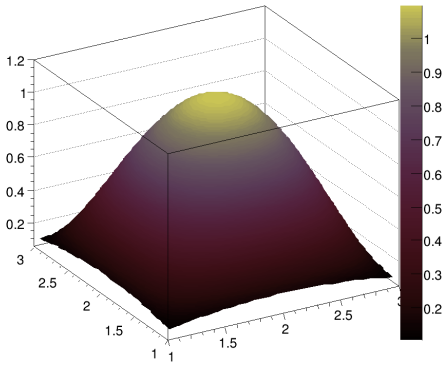
kAlpine



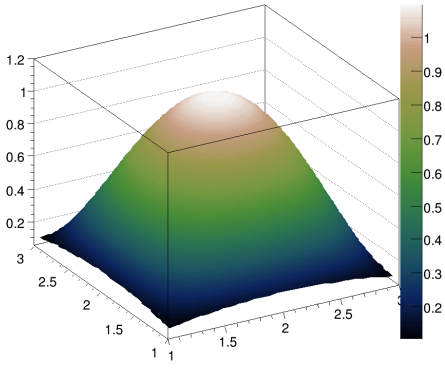
kPigeon



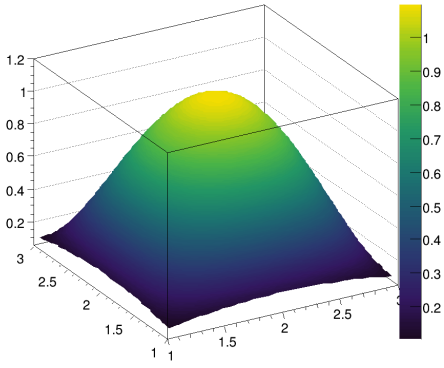
kPlum



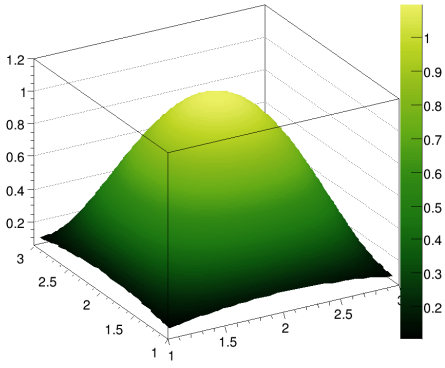
kGistEarth



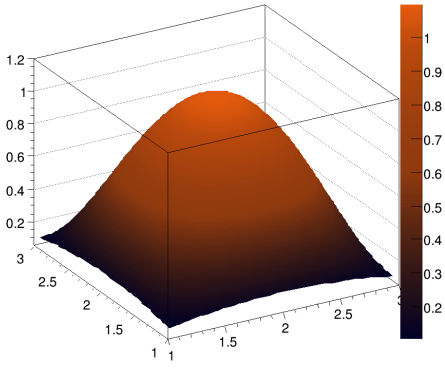
kViridis



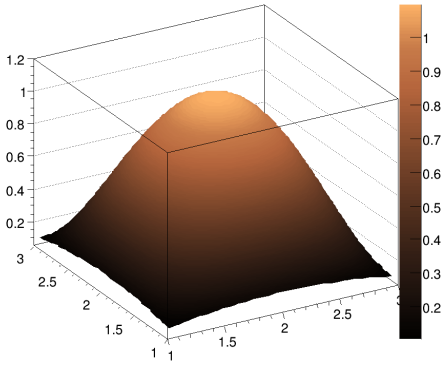
kAvocado



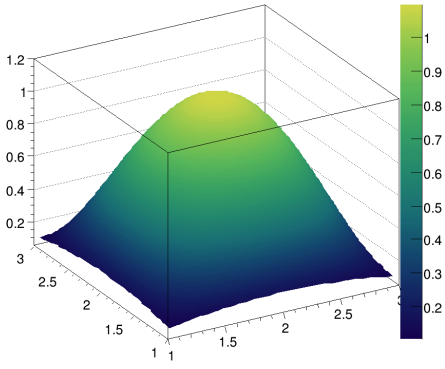
kRust



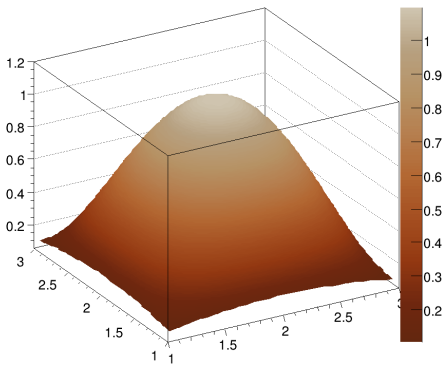
kCopper



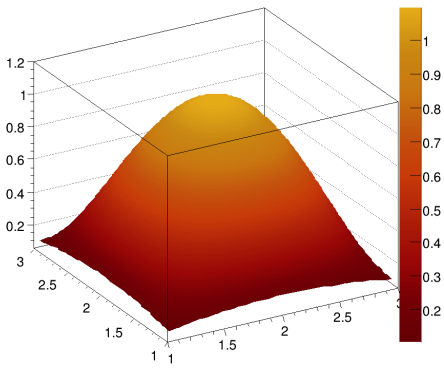
kBlueGreenYellow



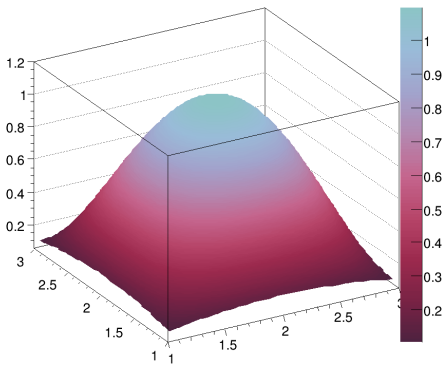
kSienna



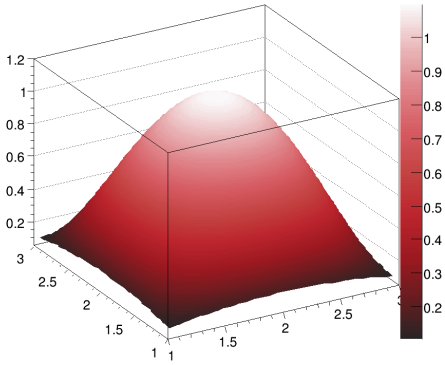
kSolar



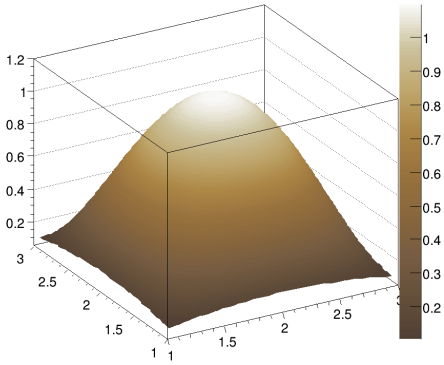
kCandy



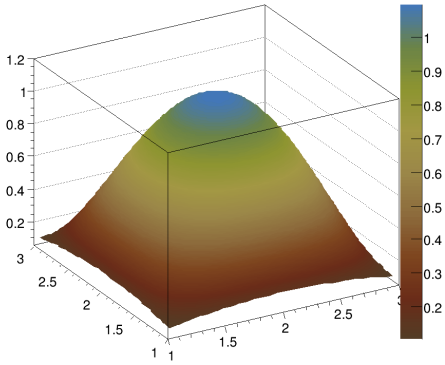
kCherry



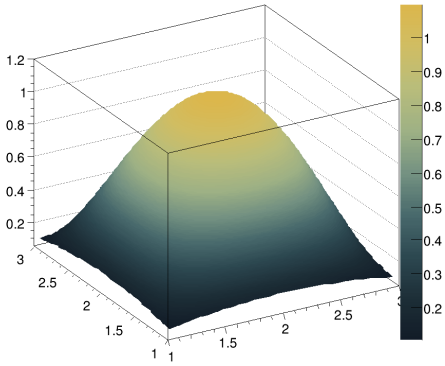
kCoffee



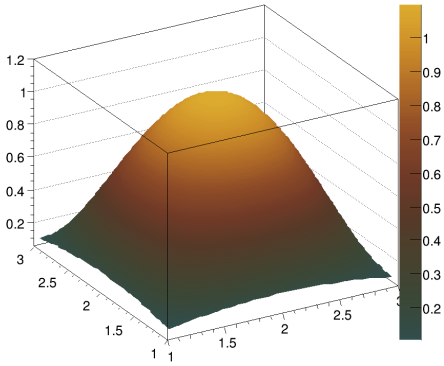
kSouthWest



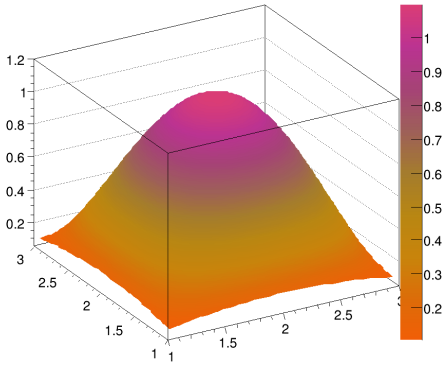
kStarryNight



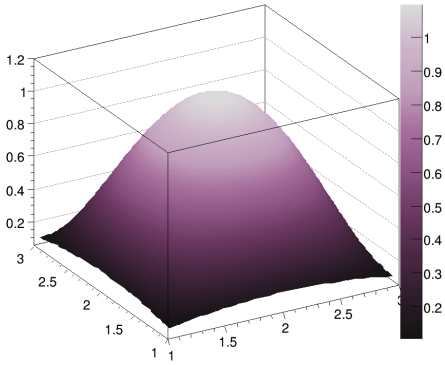
kFall



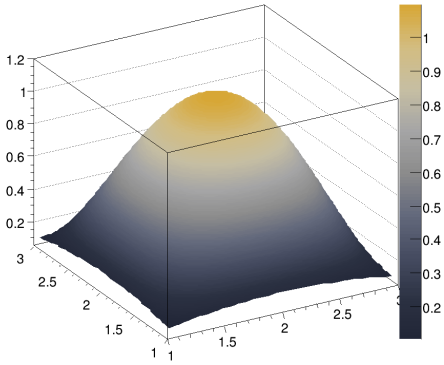
kFruitPunch



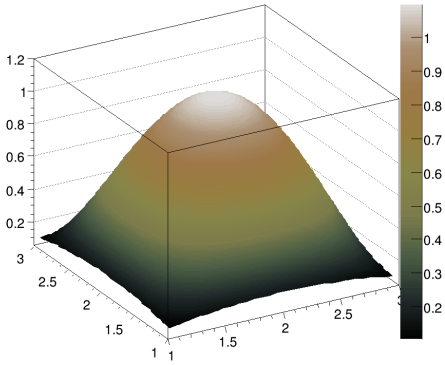
kFuchsia

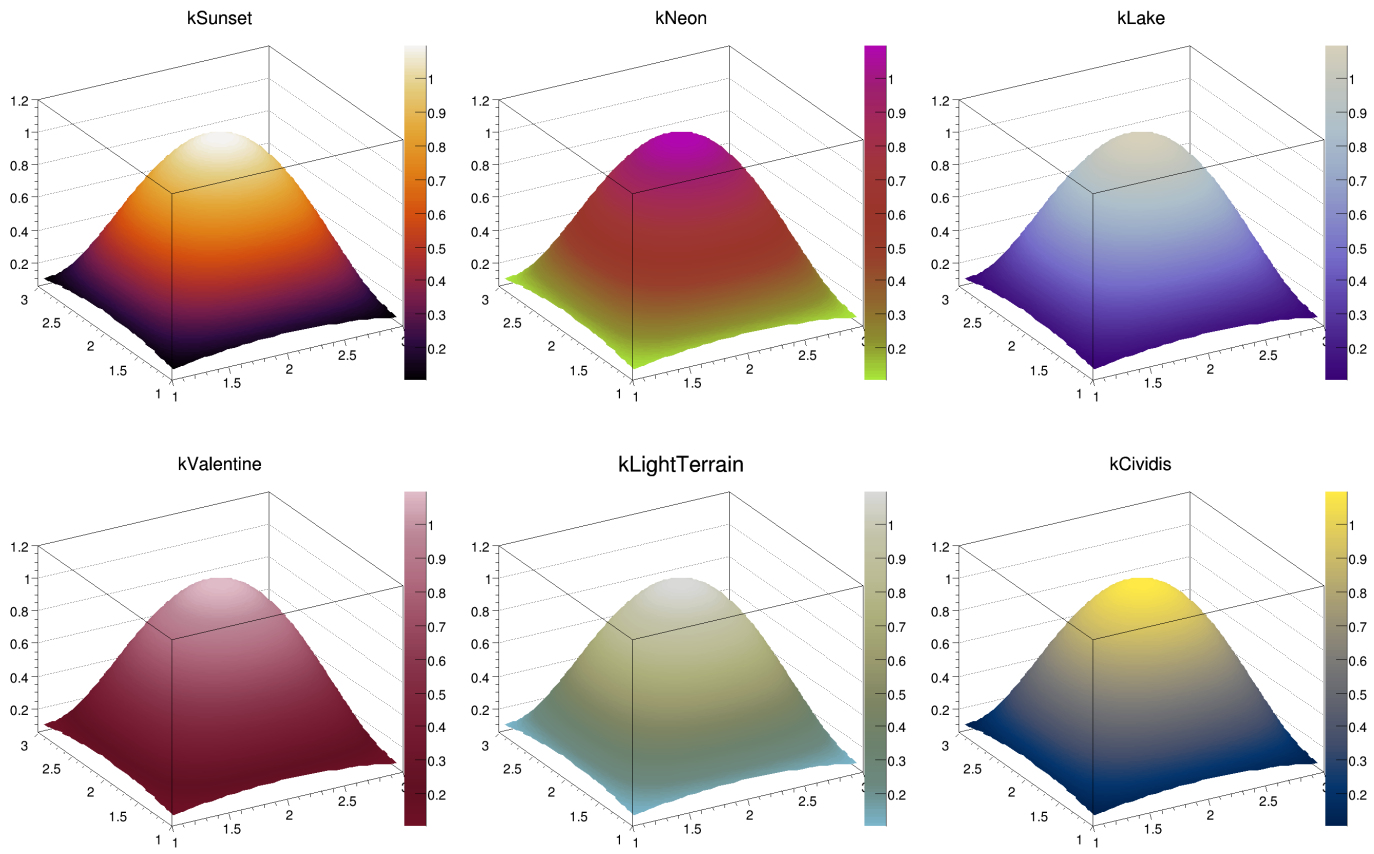


kGreyYellow

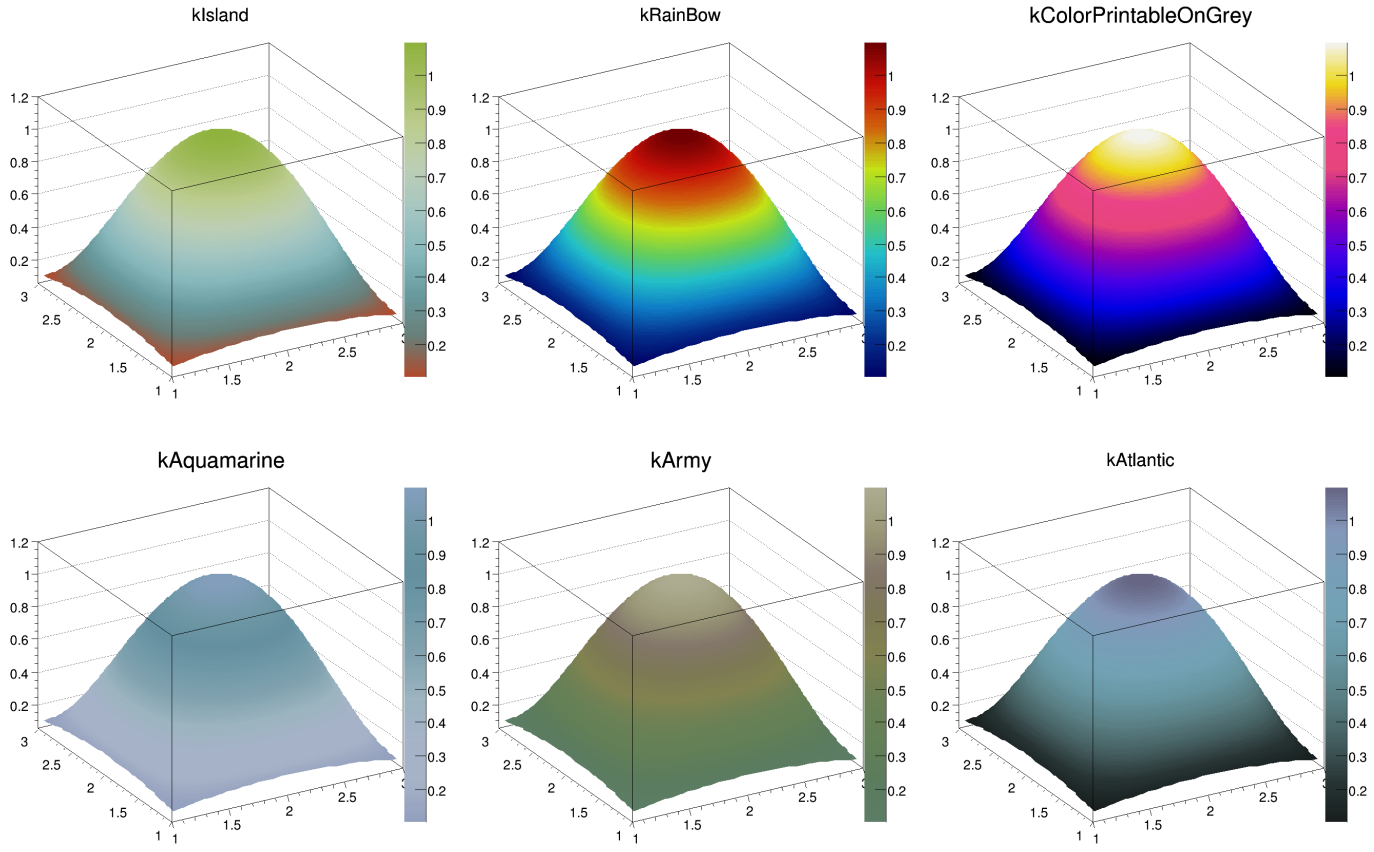


kGreenBrownTerrain

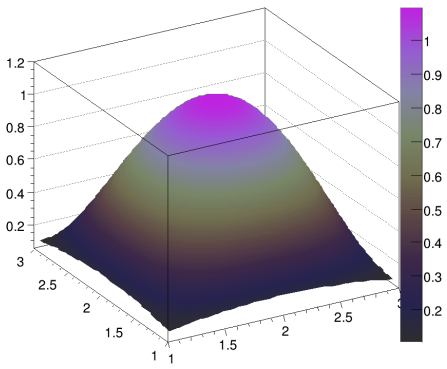




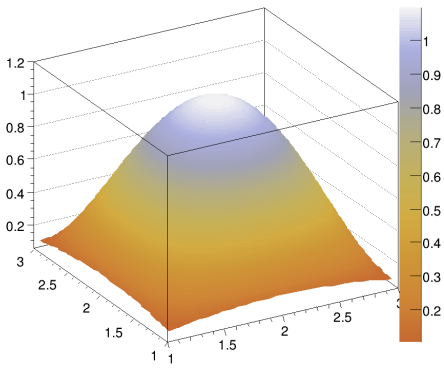

**Non Colour Vision Deficiency (CVD) friendly palettes**



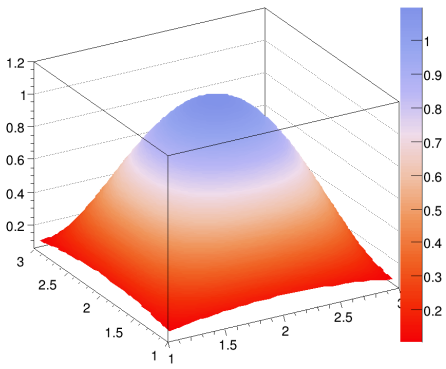
kAurora



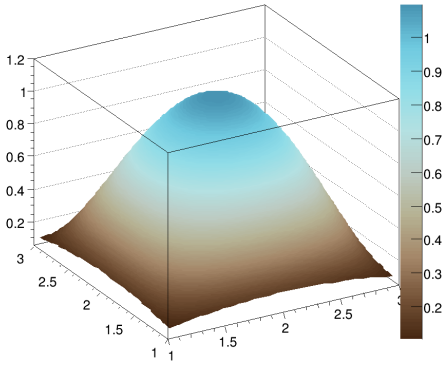
kBeach



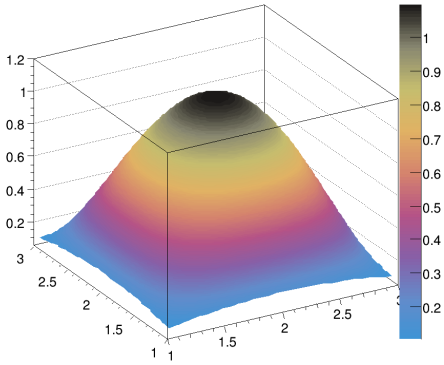
kBlackBody



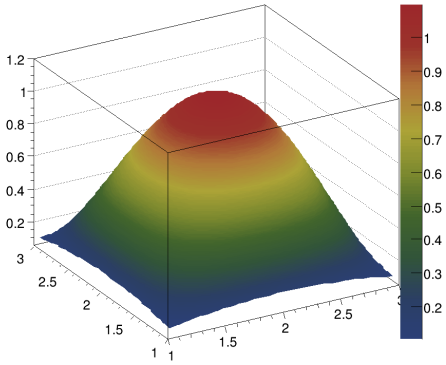
kBrownCyan



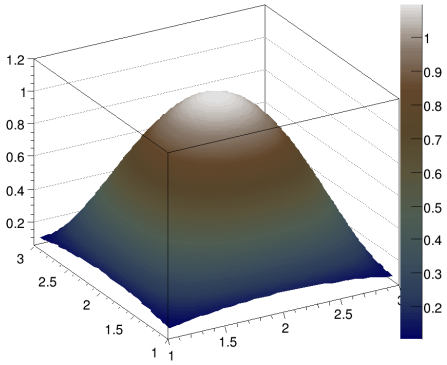
kCMYK



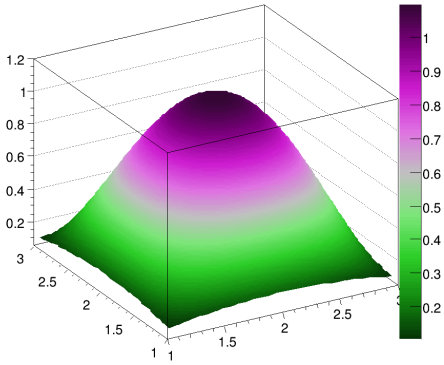
kDarkRainBow



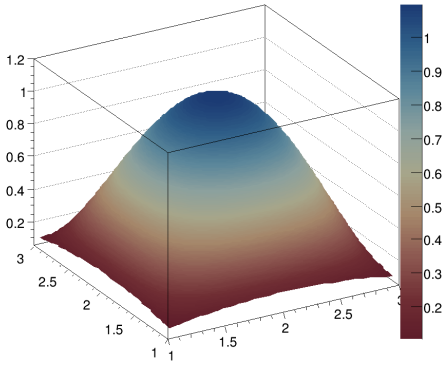
kDarkTerrain



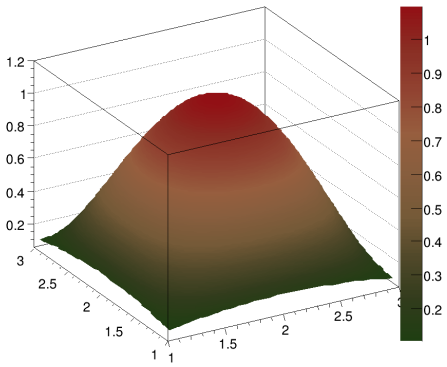
kGreenPink



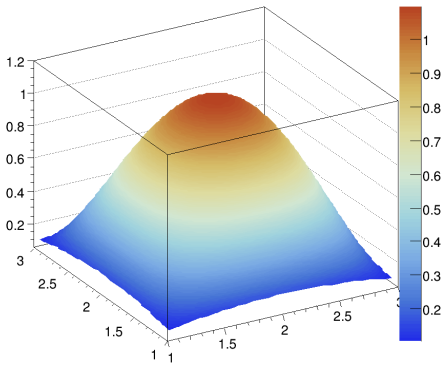
kRedBlue



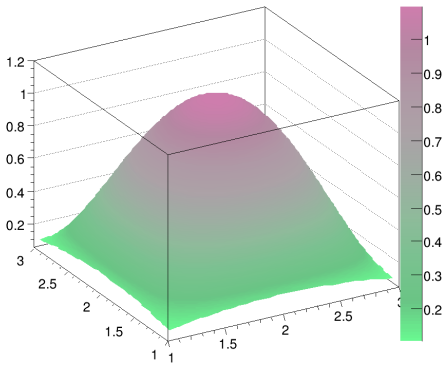
kRose

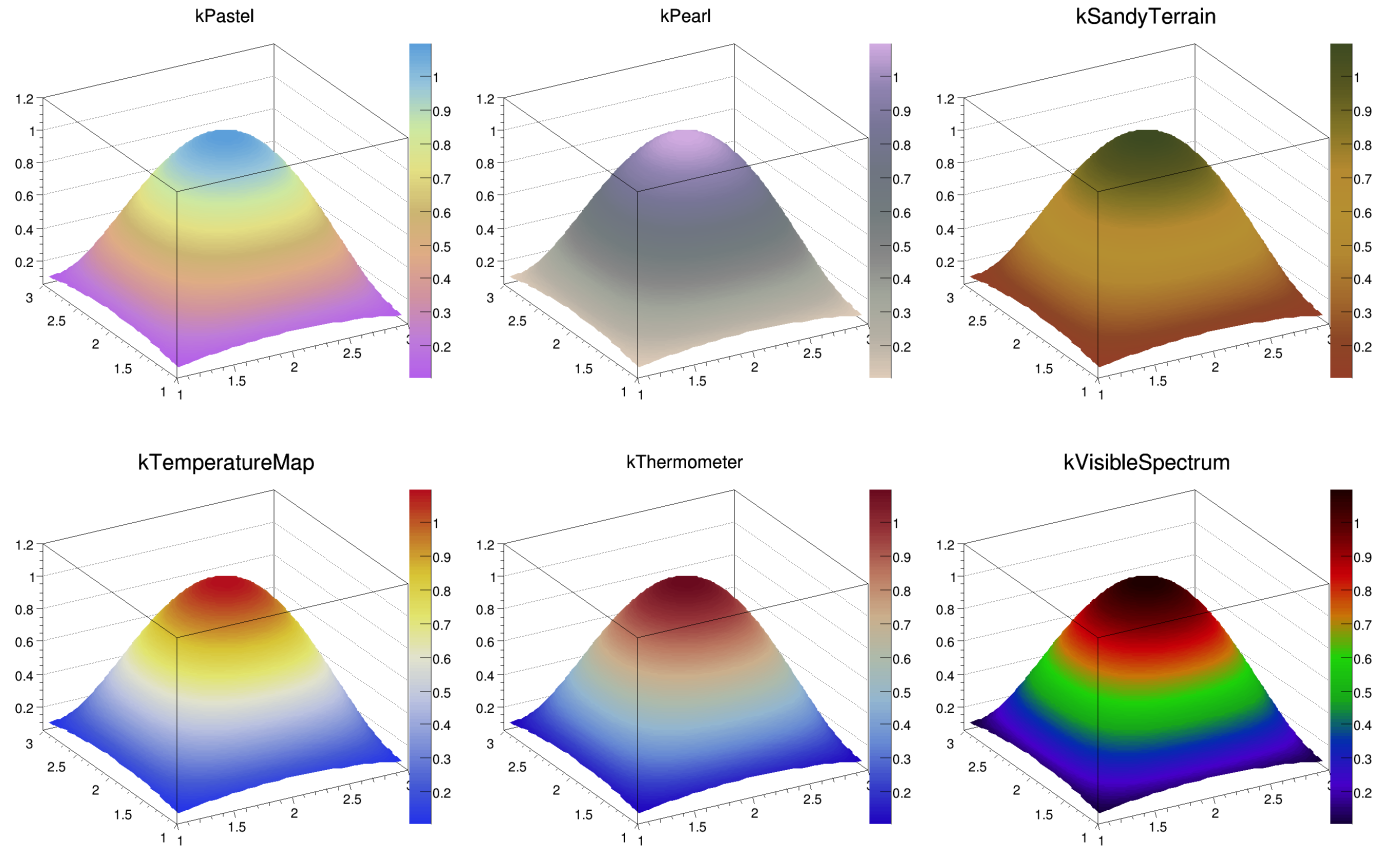


kLightTemperature



kMint

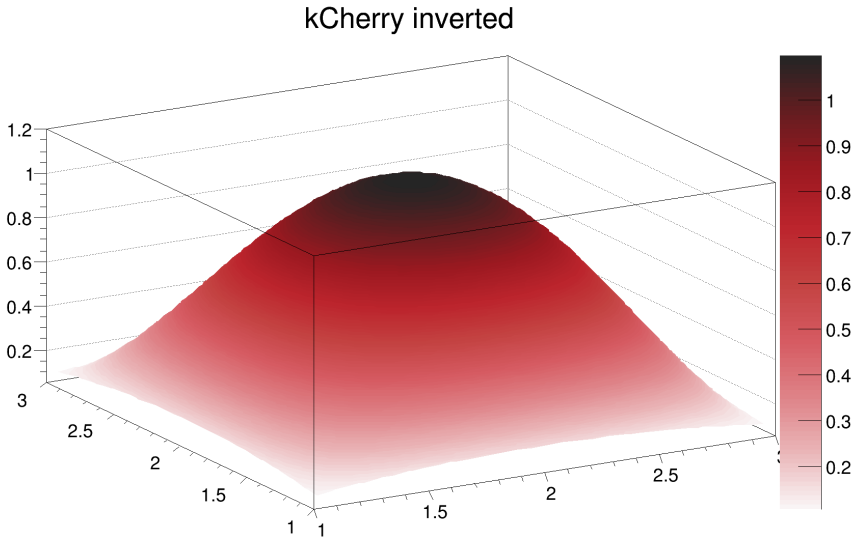





## Palette inversion

Once a palette is defined, it is possible to invert the color order thanks to the method **TColor::InvertPalette**. The top of the palette becomes the bottom and vice versa.

```
{
   auto c  = new TCanvas("c","c",0,0,600,400);
   TF2 *f2 = new TF2("f2","0.1+(1-(x-2)*(x-2))*(1-(y-2)*(y-2))",0.999,3.002,0.999,3.002);
   f2->SetContour(99); gStyle->SetPalette(kCherry);
   TColor::InvertPalette();
   f2->Draw("surf2Z"); f2->SetTitle("kCherry inverted");
}
```




## Color transparency

To make a graphics object transparent it is enough to set its color to a transparent one. The color transparency is defined via its alpha component. The alpha value varies from `0.` (fully transparent) to `1.` (fully opaque). To set the alpha value of an existing color it is

enough to do:

```
TColor *col26 = gROOT->GetColor(26);
col26->SetAlpha(0.01);
```

A new color can be created transparent the following way:

```
Int_t ci = 1756;
TColor *color = new TColor(ci, 0.1, 0.2, 0.3, "", 0.5); // alpha = 0.5
```

An example of transparency usage with parallel coordinates can be found in parallelcoordtrans.C.

To ease the creation of a transparent color the static method **`GetColorTransparent(Int_t color, Float_t a)`** is provided. In the following example the `trans_red` color index point to a red color 30% transparent. The alpha value of the color index kRed is not modified.

```
Int_t trans_red = GetColorTransparent(kRed, 0.3);
```

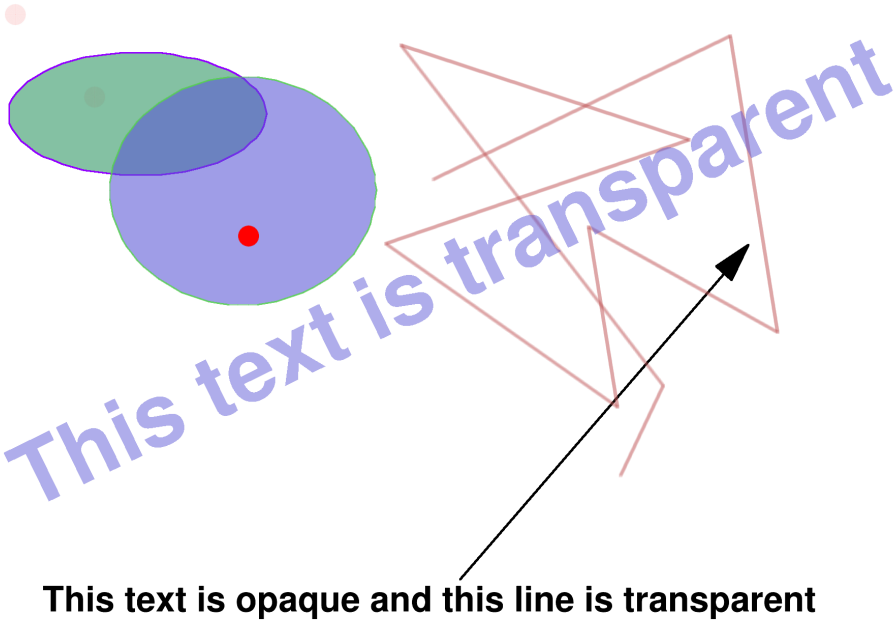
This function is also used in the methods `SetFillColorAlpha()`, `SetLineColorAlpha()`, `SetMarkerColorAlpha()` and `SetTextColorAlpha()`. In the following example the fill color of the histogram `histo` is set to blue with a transparency of 35%. The color `kBlue` itself remains fully opaque.

```
histo->SetFillColorAlpha(kBlue, 0.35);
```

The transparency is available on all platforms when the flag `OpenGL.CanvasPreferGL` is set to 1 in `$ROOTSYS/etc/system.rootrc`, or on Mac with the Cocoa backend. On the file output it is visible with PDF, PNG, Gif, JPEG, SVG, TeX ... but not PostScript. The following macro gives an example of transparency usage:

```
void transparency()
{
   auto c1 = new TCanvas("c1", "c1",224,330,700,527);
   c1->Range(-0.125,-0.125,1.125,1.125);

   auto tex = new TLatex(0.06303724,0.0194223,"This text is opaque and this line is transparent");
   tex->SetLineWidth(2);
   tex->Draw();

   auto arrow = new TArrow(0.5555158,0.07171314,0.8939828,0.6195219,0.05,"|>");
   arrow->SetLineWidth(4);
   arrow->SetAngle(30);
   arrow->Draw();

   // Draw a transparent graph.
   Double_t x[10] = {
   0.5232808, 0.8724928, 0.9280086, 0.7059456, 0.7399714,
   0.4659742, 0.8241404, 0.4838825, 0.7936963, 0.743553};
   Double_t y[10] = {
   0.7290837, 0.9631474, 0.4775896, 0.6494024, 0.3555777,
   0.622012, 0.7938247, 0.9482072, 0.3904382, 0.2410359};
   auto graph = new TGraph(10,x,y);
   graph->SetLineColorAlpha(46, 0.1);
   graph->SetLineWidth(7);
   graph->Draw("l");

   // Draw an ellipse with opaque colors.
   auto ellipse = new TEllipse(0.1740688,0.8352632,0.1518625,0.1010526,0,360,0);
   ellipse->SetFillColor(30);
   ellipse->SetLineColor(51);
   ellipse->SetLineWidth(3);
   ellipse->Draw();

   // Draw an ellipse with transparent colors, above the previous one.
   ellipse = new TEllipse(0.2985315,0.7092105,0.1566977,0.1868421,0,360,0);
   ellipse->SetFillColorAlpha(9, 0.571);
   ellipse->SetLineColorAlpha(8, 0.464);
   ellipse->SetLineWidth(3);
   ellipse->Draw();

   // Draw a transparent blue text.
   tex = new TLatex(0.04871059,0.1837649,"This text is transparent");
   tex->SetTextColorAlpha(9, 0.476);
   tex->SetTextSize(0.125);
   tex->SetTextAngle(26.0);
   tex->Draw();

   // Draw two transparent markers
   auto marker = new TMarker(0.03080229,0.998008,20);
   marker->SetMarkerColorAlpha(2, .3);
   marker->SetMarkerStyle(20);
```

```
   marker->SetMarkerSize(1.7);
   marker->Draw();
   marker = new TMarker(0.1239255,0.8635458,20);
   marker->SetMarkerColorAlpha(2, .2);
   marker->SetMarkerStyle(20);
   marker->SetMarkerSize(1.7);
   marker->Draw();

   // Draw an opaque marker
   marker = new TMarker(0.3047994,0.6344622,20);
   marker->SetMarkerColor(2);
   marker->SetMarkerStyle(20);
   marker->SetMarkerSize(1.7);
   marker->Draw();

}
```




Definition at line 19 of file TColor.h.

## Public Member Functions

| | |
|---|---|
| | **TColor** ()<br>Default constructor. |
| | **TColor** (const **TColor** &color)<br>Color copy constructor. |
| | **TColor** (**Float_t** r, **Float_t** g, **Float_t** b, **Float_t** a=1)<br>Fast **TColor** constructor. |
| | **TColor** (**Int_t** color, **Float_t** r, **Float_t** g, **Float_t** b, const char *name="", **Float_t** a=1)<br>Normal color constructor. |
| virtual | **~TColor** ()<br>Color destructor. |
| const char * | **AsHexString** () const<br>Return color as hexadecimal string. |
| void | **Copy** (**TObject** &color) const override<br>Copy this color to obj. |
| **Float_t** | **GetAlpha** () const |
| **Float_t** | **GetBlue** () const |
| virtual **Float_t** | **GetGrayscale** () const |

| | |
|---|---|
| Float_t | GetGreen () const |
| virtual void | GetHLS (Float_t &h, Float_t &l, Float_t &s) const |
| Float_t | GetHue () const |
| Float_t | GetLight () const |
| Int_t | GetNumber () const |
| ULong_t | GetPixel () const<br>Return pixel value corresponding to this color. |
| Float_t | GetRed () const |
| virtual void | GetRGB (Float_t &r, Float_t &g, Float_t &b) const |
| Float_t | GetSaturation () const |
| TClass * | IsA () const override |
| void | ls (Option_t *option="") const override<br>List this color with its attributes. |
| TColor & | operator= (const TColor &color) |
| void | Print (Option_t *option="") const override<br>Dump this color with its attributes. |
| virtual void | SetAlpha (Float_t a) |
| virtual void | SetRGB (Float_t r, Float_t g, Float_t b)<br>Initialize this color and its associated colors. |
| void | Streamer (TBuffer &) override<br>Stream an object of class TObject. |
| void | StreamerNVirtual (TBuffer &ClassDef_StreamerNVirtual_b) |

Public Member Functions inherited from TNamed

Public Member Functions inherited from TObject

## Static Public Member Functions

| | |
|---|---|
| static TClass * | Class () |
| static const char * | Class_Name () |
| static constexpr Version_t | Class_Version () |
| static void | CreateColorsCircle (Int_t offset, const char *name, UChar_t *rgb)<br>Create the "circle" colors in the color wheel. |
| static void | CreateColorsGray ()<br>Create the Gray scale colors in the Color Wheel. |
| static void | CreateColorsRectangle (Int_t offset, const char *name, UChar_t *rgb)<br>Create the "rectangular" colors in the color wheel. |
| static Int_t | CreateColorTableFromFile (TString fileName, Float_t alpha=1.)<br>Static function creating a color palette based on an input text file. |
| static void | CreateColorWheel ()<br>Static function steering the creation of all colors in the color wheel. |
| static Int_t | CreateGradientColorTable (UInt_t Number, Double_t *Stops, Double_t *Red, Double_t *Green, Double_t *Blue, UInt_t NColors, Float_t alpha=1., Bool_t setPalette=kTRUE)<br>Static function creating a color table with several connected linear gradients. |
| static const char * | DeclFileName () |
| static Bool_t | DefinedColors ()<br>Static function returning kTRUE if some new colors have been defined after initialisation or since the last call to this method. |
| static Int_t | GetColor (const char *hexcolor)<br>Static method returning color number for color specified by hex color string of form: "#rrggbb", where rr, gg and bb are in hex between [0,FF], e.g. |
| static Int_t | GetColor (Float_t r, Float_t g, Float_t b) |

Static method returning color number for color specified by r, g and b.

static Int_t **GetColor** (Int_t r, Int_t g, Int_t b)
Static method returning color number for color specified by r, g and b.

static Int_t **GetColor** (ULong_t pixel)
Static method returning color number for color specified by system dependent pixel value.

static Int_t **GetColorBright** (Int_t color)
Static function: Returns the bright color number corresponding to n If the **TColor** object does not exist, it is created.

static Int_t **GetColorDark** (Int_t color)
Static function: Returns the dark color number corresponding to n If the **TColor** object does not exist, it is created.

static Int_t **GetColorPalette** (Int_t i)
Static function returning the color number i in current palette.

static Int_t **GetColorTransparent** (Int_t color, Float_t a)
Static function: Returns the transparent color number corresponding to n.

static Int_t **GetFreeColorIndex** ()
Static function: Returns a free color index which can be used to define a user custom color.

static Int_t **GetNumberOfColors** ()
Static function returning number of colors in the color palette.

static const TArrayI & **GetPalette** ()
Static function returning the current active palette.

static void **HLS2RGB** (Float_t h, Float_t l, Float_t s, Float_t &r, Float_t &g, Float_t &b)
Static method to compute RGB from HLS.

static void **HLS2RGB** (Int_t h, Int_t l, Int_t s, Int_t &r, Int_t &g, Int_t &b)
Static method to compute RGB from HLS.

static void **HLStoRGB** (Float_t h, Float_t l, Float_t s, Float_t &r, Float_t &g, Float_t &b)

static void **HSV2RGB** (Float_t h, Float_t s, Float_t v, Float_t &r, Float_t &g, Float_t &b)
Static method to compute RGB from HSV:

static void **InitializeColors** ()
Initialize colors used by the **TCanvas** based graphics (via **TColor** objects).

static void **InvertPalette** ()
Invert the current color palette.

static Bool_t **IsGrayscale** ()
Return whether all colors return grayscale values.

static ULong_t **Number2Pixel** (Int_t ci)
Static method that given a color index number, returns the corresponding pixel value.

static void **Pixel2RGB** (ULong_t pixel, Float_t &r, Float_t &g, Float_t &b)
Convert machine dependent pixel value (obtained via RGB2Pixel or via **Number2Pixel()** or via **TColor::GetPixel()**) to r,g,b triplet.

static void **Pixel2RGB** (ULong_t pixel, Int_t &r, Int_t &g, Int_t &b)
Convert machine dependent pixel value (obtained via RGB2Pixel or via **Number2Pixel()** or via **TColor::GetPixel()**) to r,g,b triplet.

static const char * **PixelAsHexString** (ULong_t pixel)
Convert machine dependent pixel value (obtained via RGB2Pixel or via **Number2Pixel()** or via **TColor::GetPixel()**) to a hexadecimal string.

static void **RGB2HLS** (Float_t r, Float_t g, Float_t b, Float_t &h, Float_t &l, Float_t &s)
Static method to compute HLS from RGB.

static void **RGB2HLS** (Int_t r, Int_t g, Int_t b, Int_t &h, Int_t &l, Int_t &s)
Static method to compute HLS from RGB.

static void **RGB2HSV** (Float_t r, Float_t g, Float_t b, Float_t &h, Float_t &s, Float_t &v)
Static method to compute HSV from RGB.

| | |
|---|---|
| static ULong_t | **RGB2Pixel** (**Float_t r**, **Float_t g**, **Float_t b**)<br>Convert r,g,b to graphics system dependent pixel value. |
| static ULong_t | **RGB2Pixel** (**Int_t r**, **Int_t g**, **Int_t b**)<br>Convert r,g,b to graphics system dependent pixel value. |
| static void | **RGBtoHLS** (**Float_t r**, **Float_t g**, **Float_t b**, **Float_t** &**h**, **Float_t** &**l**, **Float_t** &s) |
| static Bool_t | **SaveColor** (std::ostream &out, **Int_t** ci)<br>Save a color with index > 228 as a C++ statement(s) on output stream out. |
| static void | **SetColorThreshold** (**Float_t** t)<br>This method specifies the color threshold used by GetColor to retrieve a color. |
| static void | **SetGrayscale** (**Bool_t** set=**kTRUE**)<br>Set whether all colors should return grayscale values. |
| static void | **SetPalette** (**Int_t** ncolors, **Int_t** ***colors**, **Float_t** alpha=1.)<br>Static function. |

▸ Static Public Member Functions inherited from **TNamed**

▸ Static Public Member Functions inherited from **TObject**

## Protected Attributes

**Int_t** **fNumber**
Color number identifier.

▸ Protected Attributes inherited from **TNamed**

## Private Member Functions

void **Allocate** ()
Make this color known to the graphics system.

## Static Private Member Functions

static **Float_t** **HLStoRGB1** (**Float_t** rn1, **Float_t** rn2, **Float_t** huei)
Static method. Auxiliary to **HLS2RGB()**.

## Private Attributes

**Float_t** **fAlpha**
Alpha (transparency)

**Float_t** **fBlue**
Fraction of Blue.

**Float_t** **fGreen**
Fraction of Green.

**Float_t** **fHue**
Hue.

**Float_t** **fLight**
Light.

**Float_t** **fRed**
Fraction of Red.

**Float_t** **fSaturation**
Saturation.

## Additional Inherited Members

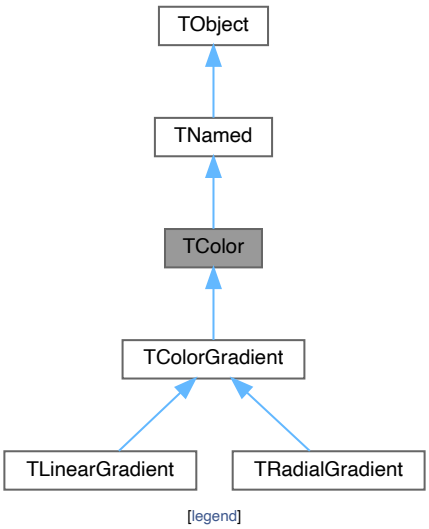
▸ Public Types inherited from **TObject**

▸ Protected Types inherited from **TObject**

▸ Protected Member Functions inherited from **TObject**

`#include <TColor.h>`

Inheritance diagram for TColor:




## Constructor & Destructor Documentation

### ◆ TColor() [1/4]

TColor::TColor ( )

Default constructor.

Definition at line **1039** of file **TColor.cxx**.

### ◆ TColor() [2/4]

```
TColor::TColor ( Int_t         color,
                 Float_t       r,
                 Float_t       g,
                 Float_t       b,
                 const char *  name = "",
                 Float_t       a = 1
               )
```

Normal color constructor.

Initialize a color structure. Compute the RGB and HLS color components.

Definition at line **1050** of file **TColor.cxx**.

### ◆ TColor() [3/4]

TColor::TColor ( **Float_t** r,
**Float_t** g,
**Float_t** b,
**Float_t** a = 1
)

Fast **TColor** constructor.

It creates a color with an index just above the current highest one. It does not name the color. This is useful to create palettes.

Definition at line **1095** of file **TColor.cxx**.

### ◆ TColor() [4/4]

TColor::TColor ( const **TColor** & color )

Color copy constructor.

Definition at line **1126** of file **TColor.cxx**.

### ◆ ~TColor()

TColor::~TColor ( ) virtual

Color destructor.

Definition at line **1114** of file **TColor.cxx**.

## Member Function Documentation

### ◆ Allocate()

void TColor::Allocate ( ) private

Make this color known to the graphics system.

Definition at line **1806** of file **TColor.cxx**.

### ◆ AsHexString()

const char * TColor::AsHexString ( ) const

Return color as hexadecimal string.

This string can be directly passed to, for example, **TGClient::GetColorByName()**. String will be reused so copy immediately if needed.

Definition at line **1269** of file **TColor.cxx**.

## Class()

static **TClass** * TColor::Class ( ) static

**Returns**

**TClass** describing this class

## Class_Name()

static const char * TColor::Class_Name ( ) static

**Returns**

Name of this class

## Class_Version()

static constexpr **Version_t** TColor::Class_Version ( ) inline static constexpr

**Returns**

Version of this class

Definition at line **105** of file **TColor.h**.

## Copy()

void TColor::Copy ( **TObject** & color ) const override virtual

Copy this color to obj.

Reimplemented from **TObject**.

Definition at line **1290** of file **TColor.cxx**.

## CreateColorsCircle()

void TColor::CreateColorsCircle ( **Int_t** offset,
const char * name,
**UChar_t** * rgb
) static

Create the "circle" colors in the color wheel.

Definition at line **1322** of file **TColor.cxx**.

## CreateColorsGray()

void TColor::CreateColorsGray ( ) static

Create the Gray scale colors in the Color Wheel.

Definition at line 1306 of file TColor.cxx.

## CreateColorsRectangle()

void TColor::CreateColorsRectangle ( Int_t offset,
const char * name,
UChar_t * rgb
) static

Create the "rectangular" colors in the color wheel.

Definition at line 1342 of file TColor.cxx.

## CreateColorTableFromFile()

Int_t TColor::CreateColorTableFromFile ( TString fileName,
Float_t alpha = 1.
) static

Static function creating a color palette based on an input text file.

Every color in the file will take the same amount of space in the palette.

**See also**

https://doi.org/10.1038/s41467-020-19160-7

**Note**

This function is designed to load into ROOT the colour-vision deficiency friendly and perceptually uniform colour maps specially designed in https://doi.org/10.5281/zenodo.4491293, namely the .txt files stored in the subfolders of ScientificColourMaps7.zip, e.g. batlow/batlow.txt

**Parameters**

**fileName** Name of a .txt file (ASCII) containing three floats per line, separated by spaces, namely the r g b fractions of the color, each value being in the range [0,1].

**alpha** the global transparency for all colors within this palette

**Returns**

a positive value on success and -1 on error.

**Author**

Fernando Hueso-González

Definition at line 2269 of file TColor.cxx.

## CreateColorWheel()

void TColor::CreateColorWheel ( ) static

Static function steering the creation of all colors in the color wheel.

Definition at line 1362 of file TColor.cxx.

## ◆ CreateGradientColorTable()

```
Int_t TColor::CreateGradientColorTable ( UInt_t     Number,
                                         Double_t * Stops,
                                         Double_t * Red,
                                         Double_t * Green,
                                         Double_t * Blue,
                                         UInt_t     NColors,
                                         Float_t    alpha = 1.,
                                         Bool_t     setPalette = kTRUE
                                       ) static
```

Static function creating a color table with several connected linear gradients.

- Number: The number of end point colors that will form the gradients. Must be at least 2.
- Stops: Where in the whole table the end point colors should lie. Each entry must be on [0, 1], each entry must be greater than the previous entry.
- Red, Green, Blue: The end point color values. Each entry must be on [0, 1]
- NColors: Total number of colors in the table. Must be at least 1.
- alpha: the opacity factor, between 0 and 1. Default is no transparency (1).
- setPalette: activate the newly created palette (true by default). If false, the caller is in charge of calling TColor::SetPalette using the return value of the function (first palette color index) and reconstructing the Int_t palette[NColors+1] array.

Returns a positive value (the index of the first color of the palette) on success and -1 on error.

The table is constructed by tracing lines between the given points in RGB space. Each color value may have a value between 0 and 1. The difference between consecutive "Stops" values gives the fraction of space in the whole table that should be used for the interval between the corresponding color values.

Normally the first element of Stops should be 0 and the last should be 1. If this is not true, fewer than NColors will be used in proportion with the total interval between the first and last elements of Stops.

This definition is similar to the povray-definition of gradient color tables.

For instance:

```
UInt_t Number = 3;
Double_t Red[3]   = { 0.0, 1.0, 1.0 };
Double_t Green[3] = { 0.0, 0.0, 1.0 };
Double_t Blue[3]  = { 1.0, 0.0, 1.0 };
Double_t Stops[3] = { 0.0, 0.4, 1.0 };
```

This defines a table in which there are three color end points: RGB = {0, 0, 1}, {1, 0, 0}, and {1, 1, 1} = blue, red, white The first 40% of the table is used to go linearly from blue to red. The remaining 60% of the table is used to go linearly from red to white.

If you define a very short interval such that less than one color fits in it, no colors at all will be allocated. If this occurs for all intervals, ROOT will revert to the default palette.

Original code by Andreas Zoglauer (zog@mpe.mpg.de)

Definition at line 2375 of file TColor.cxx.

## ◆ DeclFileName()

static const char * TColor::DeclFileName ( ) inline static

**Returns**

Name of the file containing the class declaration

Definition at line **105** of file **TColor.h**.

## ◆ DefinedColors()

**Bool_t** TColor::DefinedColors ( ) static

Static function returning kTRUE if some new colors have been defined after initialisation or since the last call to this method.

This allows to avoid the colors and palette streaming in **TCanvas::Streamer** if not needed.

Definition at line **1480** of file **TColor.cxx**.

## ◆ GetAlpha()

**Float_t** TColor::GetAlpha ( ) const inline

Definition at line **64** of file **TColor.h**.

## ◆ GetBlue()

**Float_t** TColor::GetBlue ( ) const inline

Definition at line **60** of file **TColor.h**.

## ◆ GetColor() [1/4]

**Int_t** TColor::GetColor ( const char * hexcolor ) static

Static method returning color number for color specified by hex color string of form: "#rrggbb", where rr, gg and bb are in hex between [0,FF], e.g.

"#c0c0c0".

The color retrieval is done using a threshold defined by SetColorThreshold.

If specified color does not exist it will be created with as name "#rrggbb" with rr, gg and bb in hex between [0,FF].

Definition at line **1823** of file **TColor.cxx**.

## ◆ GetColor() [2/4]

```
Int_t TColor::GetColor ( Float_t r,
                         Float_t g,
                         Float_t b
                       )
```
static

Static method returning color number for color specified by r, g and b.

The r,g,b should be in the range [0,1].

The color retrieval is done using a threshold defined by SetColorThreshold.

If specified color does not exist it will be created with as name "#rrggbb" with rr, gg and bb in hex between [0,FF].

Definition at line **1844** of file **TColor.cxx**.

## ◆ GetColor() [3/4]

```
Int_t TColor::GetColor ( Int_t r,
                         Int_t g,
                         Int_t b
                       )
```
static

Static method returning color number for color specified by r, g and b.

The r,g,b should be in the range [0,255]. If the specified color does not exist it will be created with as name "#rrggbb" with rr, gg and bb in hex between [0,FF].

The color retrieval is done using a threshold defined by SetColorThreshold.

Definition at line **1910** of file **TColor.cxx**.

## ◆ GetColor() [4/4]

```
Int_t TColor::GetColor ( ULong_t pixel )
```
static

Static method returning color number for color specified by system dependent pixel value.

Pixel values can be obtained, e.g., from the GUI color picker.

The color retrieval is done using a threshold defined by SetColorThreshold.

Definition at line **1862** of file **TColor.cxx**.

## ◆ GetColorBright()

```
Int_t TColor::GetColorBright ( Int_t n )
```
static

Static function: Returns the bright color number corresponding to n If the **TColor** object does not exist, it is created.

The convention is that the bright color nb = n+150

Definition at line **1970** of file **TColor.cxx**.

## ◆ GetColorDark()

**Int_t** TColor::GetColorDark ( **Int_t** n ) static

Static function: Returns the dark color number corresponding to n If the **TColor** object does not exist, it is created.

The convention is that the dark color nd = n+100

Definition at line **2002** of file **TColor.cxx**.

## ◆ GetColorPalette()

**Int_t** TColor::GetColorPalette ( **Int_t** i ) static

Static function returning the color number i in current palette.

Definition at line **1450** of file **TColor.cxx**.

## ◆ GetColorTransparent()

**Int_t** TColor::GetColorTransparent ( **Int_t** n,
**Float_t** a
) static

Static function: Returns the transparent color number corresponding to n.

The transparency level is given by the alpha value a. If a color with the same RGBa values already exists it is returned.

Definition at line **2034** of file **TColor.cxx**.

## ◆ GetFreeColorIndex()

**Int_t** TColor::GetFreeColorIndex ( ) static

Static function: Returns a free color index which can be used to define a user custom color.

```
Int_t ci = TColor::GetFreeColorIndex();
TColor *color = new TColor(ci, 0.1, 0.2, 0.3);
```

Definition at line **2071** of file **TColor.cxx**.

## ◆ GetGrayscale()

virtual **Float_t** TColor::GetGrayscale ( ) const inline virtual

Definition at line **65** of file **TColor.h**.

## ◆ GetGreen()

**Float_t** TColor::GetGreen ( ) const inline

Definition at line **59** of file **TColor.h**.

## ◆ GetHLS()

virtual void TColor::GetHLS ( **Float_t** & h,
**Float_t** & l,
**Float_t** & s
) const inline virtual

Definition at line **54** of file **TColor.h**.

## ◆ GetHue()

**Float_t** TColor::GetHue ( ) const inline

Definition at line **61** of file **TColor.h**.

## ◆ GetLight()

**Float_t** TColor::GetLight ( ) const inline

Definition at line **62** of file **TColor.h**.

## ◆ GetNumber()

**Int_t** TColor::GetNumber ( ) const inline

Definition at line **56** of file **TColor.h**.

## ◆ GetNumberOfColors()

**Int_t** TColor::GetNumberOfColors ( ) static

Static function returning number of colors in the color palette.

Definition at line **1470** of file **TColor.cxx**.

## ◆ GetPalette()

const **TArrayI** & TColor::GetPalette ( ) static

Static function returning the current active palette.

Definition at line **1462** of file **TColor.cxx**.

## ◆ GetPixel()

**ULong_t** TColor::GetPixel ( ) const

Return pixel value corresponding to this color.

This pixel value can be used in the GUI classes. This call does not work in batch mode since it needs to communicate with the graphics system.

Definition at line **1494** of file **TColor.cxx**.

## ◆ GetRed()

**Float_t** TColor::GetRed ( ) const inline

Definition at line **58** of file **TColor.h**.

## ◆ GetRGB()

virtual void TColor::GetRGB ( **Float_t** & r,
**Float_t** & g,
**Float_t** & b
) const inline virtual

Definition at line **52** of file **TColor.h**.

## ◆ GetSaturation()

**Float_t** TColor::GetSaturation ( ) const inline

Definition at line **63** of file **TColor.h**.

## ◆ HLS2RGB() [1/2]

void TColor::HLS2RGB ( **Float_t** hue,
**Float_t** light,
**Float_t** satur,
**Float_t** & r,
**Float_t** & g,
**Float_t** & b
) static

Static method to compute RGB from HLS.

The l and s are between [0,1] and h is between [0,360]. The returned r,g,b triplet is between [0,1].

Definition at line **1511** of file **TColor.cxx**.

## ◆ HLS2RGB() [2/2]

```
void TColor::HLS2RGB ( Int_t   h,
                       Int_t   l,
                       Int_t   s,
                       Int_t & r,
                       Int_t & g,
                       Int_t & b
                     )                                                    static
```

Static method to compute RGB from HLS.

The h,l,s are between [0,255]. The returned r,g,b triplet is between [0,255].

Definition at line **1551** of file **TColor.cxx**.

## ◆ HLStoRGB()

```
static void TColor::HLStoRGB ( Float_t   h,
                               Float_t   l,
                               Float_t   s,
                               Float_t & r,
                               Float_t & g,
                               Float_t & b
                             )                                            inline static
```

Definition at line **74** of file **TColor.h**.

## ◆ HLStoRGB1()

```
Float_t TColor::HLStoRGB1 ( Float_t rn1,
                            Float_t rn2,
                            Float_t huei
                          )                                               static private
```

Static method. Auxiliary to **HLS2RGB()**.

Definition at line **1536** of file **TColor.cxx**.

## ◆ HSV2RGB()

void TColor::HSV2RGB ( **Float_t** hue,
**Float_t** satur,
**Float_t** value,
**Float_t &** r,
**Float_t &** g,
**Float_t &** b
) static

Static method to compute RGB from HSV:

- The hue value runs from 0 to 360.
- The saturation is the degree of strength or purity and is from 0 to 1. Purity is how much white is added to the color, so S=1 makes the purest color (no white).
- Brightness value also ranges from 0 to 1, where 0 is the black.

The returned r,g,b triplet is between [0,1].

Definition at line **1577** of file **TColor.cxx**.

## ◆ InitializeColors()

void TColor::InitializeColors ( ) static

Initialize colors used by the **TCanvas** based graphics (via **TColor** objects).

This method should be called before the ApplicationImp is created (which initializes the GUI colors).

Definition at line **1143** of file **TColor.cxx**.

## ◆ InvertPalette()

void TColor::InvertPalette ( ) static

Invert the current color palette.

The top of the palette becomes the bottom and vice versa.

Definition at line **3285** of file **TColor.cxx**.

## ◆ IsA()

**TClass** * TColor::IsA ( ) const inline override virtual

**Returns**
**TClass** describing current object

Reimplemented from **TObject**.

Reimplemented in **TColorGradient**, **TLinearGradient**, and **TRadialGradient**.

Definition at line **105** of file **TColor.h**.

## ◆ IsGrayscale()

**Bool_t** TColor::IsGrayscale ( ) static

Return whether all colors return grayscale values.

Definition at line **2229** of file **TColor.cxx**.

## ls()

void TColor::ls ( **Option_t** * option = "" ) const override virtual

List this color with its attributes.

Reimplemented from **TObject**.

Definition at line **1633** of file **TColor.cxx**.

## Number2Pixel()

**ULong_t** TColor::Number2Pixel ( **Int_t** ci ) static

Static method that given a color index number, returns the corresponding pixel value.

This pixel value can be used in the GUI classes. This call does not work in batch mode since it needs to communicate with the graphics system.

Definition at line **2082** of file **TColor.cxx**.

## operator=()

**TColor** & TColor::operator= ( const **TColor** & color )

Definition at line **1131** of file **TColor.cxx**.

## Pixel2RGB() [1/2]

void TColor::Pixel2RGB ( **ULong_t** pixel,
**Float_t** & r,
**Float_t** & g,
**Float_t** & b
) static

Convert machine dependent pixel value (obtained via RGB2Pixel or via **Number2Pixel()** or via **TColor::GetPixel()**) to r,g,b triplet.

The r,g,b triplet will be [0,1].

Definition at line **2143** of file **TColor.cxx**.

## Pixel2RGB() [2/2]

void TColor::Pixel2RGB ( **ULong_t** pixel,
**Int_t &** r,
**Int_t &** g,
**Int_t &** b
) static

Convert machine dependent pixel value (obtained via RGB2Pixel or via **Number2Pixel()** or via **TColor::GetPixel()**) to r,g,b triplet.

The r,g,b triplet will be [0,255].

Definition at line **2158** of file **TColor.cxx**.

## PixelAsHexString()

const char * TColor::PixelAsHexString ( **ULong_t** pixel ) static

Convert machine dependent pixel value (obtained via RGB2Pixel or via **Number2Pixel()** or via **TColor::GetPixel()**) to a hexadecimal string.

This string can be directly passed to, for example, **TGClient::GetColorByName()**. String will be reused so copy immediately if needed.

Definition at line **2175** of file **TColor.cxx**.

## Print()

void TColor::Print ( **Option_t** * option = **""** ) const override virtual

Dump this color with its attributes.

Reimplemented from **TObject**.

Definition at line **1642** of file **TColor.cxx**.

## RGB2HLS() [1/2]

void TColor::RGB2HLS ( **Float_t** rr,
**Float_t** gg,
**Float_t** bb,
**Float_t &** hue,
**Float_t &** light,
**Float_t &** satur
) static

Static method to compute HLS from RGB.

The r,g,b triplet is between [0,1], hue is between [0,360], light and satur are [0,1].

Definition at line **1651** of file **TColor.cxx**.

## RGB2HLS() [2/2]

```
void TColor::RGB2HLS ( Int_t   r,
                       Int_t   g,
                       Int_t   b,
                       Int_t & h,
                       Int_t & l,
                       Int_t & s
                     )                                                    static
```

Static method to compute HLS from RGB.

The r,g,b triplet is between [0,255], hue, light and satur are between [0,255].

Definition at line **1741** of file **TColor.cxx**.

## ◆ RGB2HSV()

```
void TColor::RGB2HSV ( Float_t   r,
                       Float_t   g,
                       Float_t   b,
                       Float_t & hue,
                       Float_t & satur,
                       Float_t & value
                     )                                                    static
```

Static method to compute HSV from RGB.

- The input values:
  - r,g,b triplet is between [0,1].
- The returned values:
  - The hue value runs from 0 to 360.
  - The saturation is the degree of strength or purity and is from 0 to 1. Purity is how much white is added to the color, so S=1 makes the purest color (no white).
  - Brightness value also ranges from 0 to 1, where 0 is the black.

Definition at line **1706** of file **TColor.cxx**.

## ◆ RGB2Pixel() [1/2]

```
ULong_t TColor::RGB2Pixel ( Float_t r,
                            Float_t g,
                            Float_t b
                          )                                               static
```

Convert r,g,b to graphics system dependent pixel value.

The r,g,b triplet must be [0,1].

Definition at line **2098** of file **TColor.cxx**.

## ◆ RGB2Pixel() [2/2]

```
ULong_t TColor::RGB2Pixel ( Int_t r,
                            Int_t g,
                            Int_t b
                          )                                   static
```

Convert r,g,b to graphics system dependent pixel value.

The r,g,b triplet must be [0,255].

Definition at line **2120** of file **TColor.cxx**.

## ◆ RGBtoHLS()

```
static void TColor::RGBtoHLS ( Float_t   r,
                               Float_t   g,
                               Float_t   b,
                               Float_t & h,
                               Float_t & l,
                               Float_t & s
                             )                                inline static
```

Definition at line **79** of file **TColor.h**.

## ◆ SaveColor()

```
Bool_t TColor::SaveColor ( std::ostream &  out,
                           Int_t           ci
                         )                                    static
```

Save a color with index > 228 as a C++ statement(s) on output stream out.

Return kFALSE if color not saved in the output stream

Definition at line **2188** of file **TColor.cxx**.

## ◆ SetAlpha()

```
virtual void TColor::SetAlpha ( Float_t  a )                 inline virtual
```

Definition at line **68** of file **TColor.h**.

## ◆ SetColorThreshold()

void TColor::SetColorThreshold ( **Float_t** t ) static

This method specifies the color threshold used by GetColor to retrieve a color.

**Parameters**

[in] **t** Color threshold. By default is equal to 1./31. or 1./255. depending on the number of available color planes.

When GetColor is called, it scans the defined colors and compare them to the requested color. If the Red Green and Blue values passed to GetColor are Rr Gr Br and Rd Gd Bd the values of a defined color. These two colors are considered equal if (abs(Rr-Rd) < t & abs(Br-Bd) < t & abs(Br-Bd) < t). If this test passes, the color defined by Rd Gd Bd is returned by GetColor.

To make sure GetColor will return a color having exactly the requested R G B values it is enough to specify a nul :

```
TColor::SetColorThreshold(0.);
```

To reset the color threshold to its default value it is enough to do:

```
TColor::SetColorThreshold(-1.);
```

Definition at line **1895** of file **TColor.cxx**.

## SetGrayscale()

void TColor::SetGrayscale ( **Bool_t** set = **kTRUE** ) static

Set whether all colors should return grayscale values.

Definition at line **2237** of file **TColor.cxx**.

## SetPalette()

void TColor::SetPalette ( **Int_t** ncolors,
**Int_t** * colors,
**Float_t** alpha = 1.
) static

Static function.

The color palette is used by the histogram classes (see **TH1::Draw** options). For example **TH1::Draw**("col") draws a 2-D histogram with cells represented by a box filled with a color CI function of the cell content. if the cell content is N, the color CI used will be the color number in colors[N],etc. If the maximum cell content is > ncolors, all cell contents are scaled to ncolors.

`if ncolors <= 0` a default palette (see below) of 50 colors is defined. The colors defined in this palette are OK for coloring pads, labels.

```
index 0->9   : grey colors from light to dark grey
index 10->19 : "brown" colors
index 20->29 : "blueish" colors
index 30->39 : "redish" colors
index 40->49 : basic colors
```

`if ncolors == 1 && colors == 0`, a Rainbow Color map is created with 50 colors. It is kept for backward compatibility. Better palettes like kBird are recommended.

High quality predefined palettes with 255 colors are available when `colors == 0`. The following value of `ncolors` give access to:

```
if ncolors = 51 and colors=0, a Deep Sea palette is used.
if ncolors = 52 and colors=0, a Grey Scale palette is used.
if ncolors = 53 and colors=0, a Dark Body Radiator palette is used.
if ncolors = 54 and colors=0, a Two-Color Hue palette is used.(dark blue through neutral gray to bright
      yellow)
if ncolors = 55 and colors=0, a Rain Bow palette is used.
```

```
if ncolors = 56 and colors=0, an Inverted Dark Body Radiator palette is used.
if ncolors = 57 and colors=0, a monotonically increasing L value palette is used.
if ncolors = 58 and colors=0, a Cubehelix palette is used
                              (Cf. Dave Green's "cubehelix" colour scheme at http://www.mrao.cam.ac.uk
if ncolors = 59 and colors=0, a Green Red Violet palette is used.
if ncolors = 60 and colors=0, a Blue Red Yellow palette is used.
if ncolors = 61 and colors=0, an Ocean palette is used.
if ncolors = 62 and colors=0, a Color Printable On Grey palette is used.
if ncolors = 63 and colors=0, an Alpine palette is used.
if ncolors = 64 and colors=0, an Aquamarine palette is used.
if ncolors = 65 and colors=0, an Army palette is used.
if ncolors = 66 and colors=0, an Atlantic palette is used.
if ncolors = 67 and colors=0, an Aurora palette is used.
if ncolors = 68 and colors=0, an Avocado palette is used.
if ncolors = 69 and colors=0, a Beach palette is used.
if ncolors = 70 and colors=0, a Black Body palette is used.
if ncolors = 71 and colors=0, a Blue Green Yellow palette is used.
if ncolors = 72 and colors=0, a Brown Cyan palette is used.
if ncolors = 73 and colors=0, a CMYK palette is used.
if ncolors = 74 and colors=0, a Candy palette is used.
if ncolors = 75 and colors=0, a Cherry palette is used.
if ncolors = 76 and colors=0, a Coffee palette is used.
if ncolors = 77 and colors=0, a Dark Rain Bow palette is used.
if ncolors = 78 and colors=0, a Dark Terrain palette is used.
if ncolors = 79 and colors=0, a Fall palette is used.
if ncolors = 80 and colors=0, a Fruit Punch palette is used.
if ncolors = 81 and colors=0, a Fuchsia palette is used.
if ncolors = 82 and colors=0, a Grey Yellow palette is used.
if ncolors = 83 and colors=0, a Green Brown Terrain palette is used.
if ncolors = 84 and colors=0, a Green Pink palette is used.
if ncolors = 85 and colors=0, an Island palette is used.
if ncolors = 86 and colors=0, a Lake palette is used.
if ncolors = 87 and colors=0, a Light Temperature palette is used.
if ncolors = 88 and colors=0, a Light Terrain palette is used.
if ncolors = 89 and colors=0, a Mint palette is used.
if ncolors = 90 and colors=0, a Neon palette is used.
if ncolors = 91 and colors=0, a Pastel palette is used.
if ncolors = 92 and colors=0, a Pearl palette is used.
if ncolors = 93 and colors=0, a Pigeon palette is used.
if ncolors = 94 and colors=0, a Plum palette is used.
if ncolors = 95 and colors=0, a Red Blue palette is used.
if ncolors = 96 and colors=0, a Rose palette is used.
if ncolors = 97 and colors=0, a Rust palette is used.
if ncolors = 98 and colors=0, a Sandy Terrain palette is used.
if ncolors = 99 and colors=0, a Sienna palette is used.
if ncolors = 100 and colors=0, a Solar palette is used.
if ncolors = 101 and colors=0, a South West palette is used.
if ncolors = 102 and colors=0, a Starry Night palette is used.
if ncolors = 103 and colors=0, a Sunset palette is used.
if ncolors = 104 and colors=0, a Temperature Map palette is used.
if ncolors = 105 and colors=0, a Thermometer palette is used.
if ncolors = 106 and colors=0, a Valentine palette is used.
if ncolors = 107 and colors=0, a Visible Spectrum palette is used.
if ncolors = 108 and colors=0, a Water Melon palette is used.
if ncolors = 109 and colors=0, a Cool palette is used.
if ncolors = 110 and colors=0, a Copper palette is used.
if ncolors = 111 and colors=0, a Gist Earth palette is used.
if ncolors = 112 and colors=0, a Viridis palette is used.
if ncolors = 113 and colors=0, a Cividis palette is used.
```

These palettes can also be accessed by names:

```
kDeepSea=51,           kGreyScale=52,    kDarkBodyRadiator=53,
kBlueYellow= 54,       kRainBow=55,      kInvertedDarkBodyRadiator=56,
kBird=57,              kCubehelix=58,    kGreenRedViolet=59,
kBlueRedYellow=60,     kOcean=61,        kColorPrintableOnGrey=62,
kAlpine=63,            kAquamarine=64,   kArmy=65,
kAtlantic=66,          kAurora=67,       kAvocado=68,
kBeach=69,             kBlackBody=70,    kBlueGreenYellow=71,
kBrownCyan=72,         kCMYK=73,         kCandy=74,
kCherry=75,            kCoffee=76,       kDarkRainBow=77,
kDarkTerrain=78,       kFall=79,         kFruitPunch=80,
kFuchsia=81,           kGreyYellow=82,   kGreenBrownTerrain=83,
kGreenPink=84,         kIsland=85,       kLake=86,
kLightTemperature=87, kLightTerrain=88, kMint=89,
kNeon=90,              kPastel=91,       kPearl=92,
kPigeon=93,            kPlum=94,         kRedBlue=95,
kRose=96,              kRust=97,         kSandyTerrain=98,
kSienna=99,            kSolar=100,       kSouthWest=101,
kStarryNight=102,      kSunset=103,      kTemperatureMap=104,
kThermometer=105,      kValentine=106,   kVisibleSpectrum=107,
kWaterMelon=108,       kCool=109,        kCopper=110,
kGistEarth=111         kViridis=112,     kCividis=113
```

For example:

```
gStyle->SetPalette(kBird);
```

Set the current palette as "Bird" (number 57).

The color numbers specified in the palette can be viewed by selecting the item "colors" in the "VIEW" menu of the canvas toolbar. The color parameters can be changed via **TColor::SetRGB**.

Note that when drawing a 2D histogram h2 with the option "COL" or "COLZ" or with any "CONT" options using the color map, the number of colors used is defined by the number of contours n specified with: `h2->SetContour(n)`

Definition at line **2566** of file **TColor.cxx**.

### ◆ SetRGB()

void TColor::SetRGB ( **Float_t** r,
**Float_t** g,
**Float_t** b
) virtual

Initialize this color and its associated colors.

Definition at line **1759** of file **TColor.cxx**.

### ◆ Streamer()

void TColor::Streamer ( **TBuffer** & R__b ) override virtual

Stream an object of class **TObject**.

Reimplemented from **TObject**.

Reimplemented in **TColorGradient**, **TLinearGradient**, and **TRadialGradient**.

### ◆ StreamerNVirtual()

void TColor::StreamerNVirtual ( **TBuffer** & ClassDef_StreamerNVirtual_b ) inline

Definition at line **105** of file **TColor.h**.

## Member Data Documentation

### ◆ fAlpha

**Float_t** TColor::fAlpha private

Alpha (transparency)

Definition at line **29** of file **TColor.h**.

### ◆ fBlue

**Float_t** TColor::fBlue private

Fraction of Blue.

Definition at line **25** of file **TColor.h**.

## ◆ fGreen

**Float_t** TColor::fGreen private

Fraction of Green.

Definition at line **24** of file **TColor.h**.

## ◆ fHue

**Float_t** TColor::fHue private

Hue.

Definition at line **26** of file **TColor.h**.

## ◆ fLight

**Float_t** TColor::fLight private

Light.

Definition at line **27** of file **TColor.h**.

## ◆ fNumber

**Int_t** TColor::fNumber protected

Color number identifier.

Definition at line **21** of file **TColor.h**.

## ◆ fRed

**Float_t** TColor::fRed private

Fraction of Red.

Definition at line **23** of file **TColor.h**.

## ◆ fSaturation

**Float_t** TColor::fSaturation private

Saturation.

Definition at line **28** of file **TColor.h**.

- core/base/inc/**TColor.h**
- core/base/src/**TColor.cxx**