ROOT Graphics

Olivier Couet (CERN PH-SFT)

"ROOT Turns 20" Users' Workshop 15-18 September 20015

Abstract

Graphics News: News on Palettes, Transparency, Interactive editing, LaTex Dump...

ROOT graphics had many developments since the last workshop. We will summarise them, emphasising the most recent and noticeable ones and give an overview on the ongoing and planned projects. font align

News on Palettes

A default color map should be chosen carefully. Most users use it.

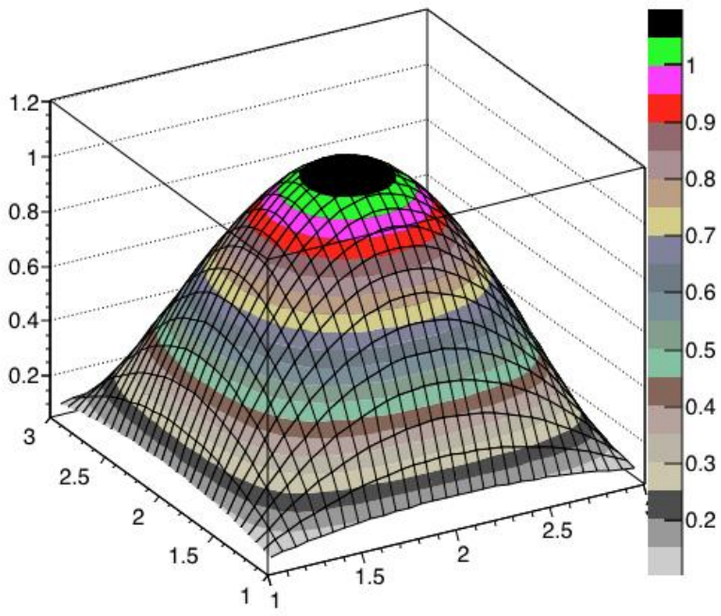
It should …

1. be visually appealing,
2. order colors in the same intuitive way for all people,
3. not be sensitive to vision deficiencies,
4. have a maximal perceptual resolution,
5. not hide structures in the data,
6. not introduce gradients not related to data,

News on Palettes

The ROOT's default palette history…

The first default palette was:

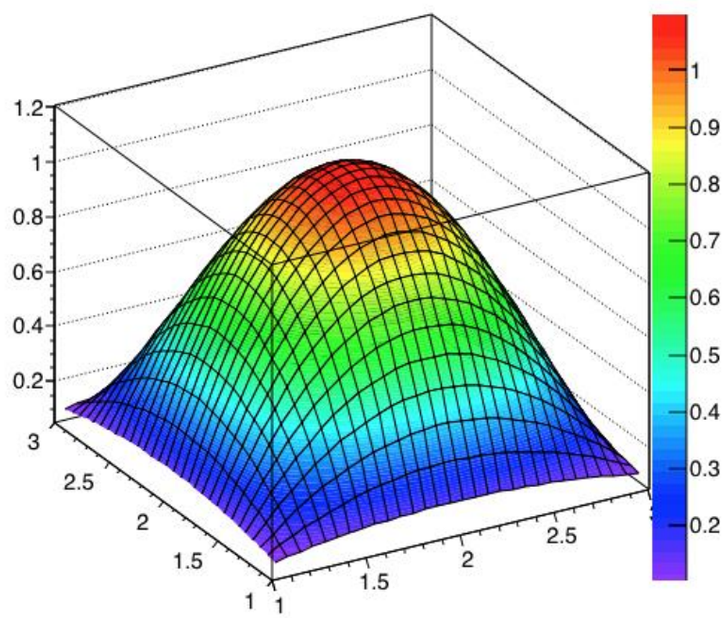


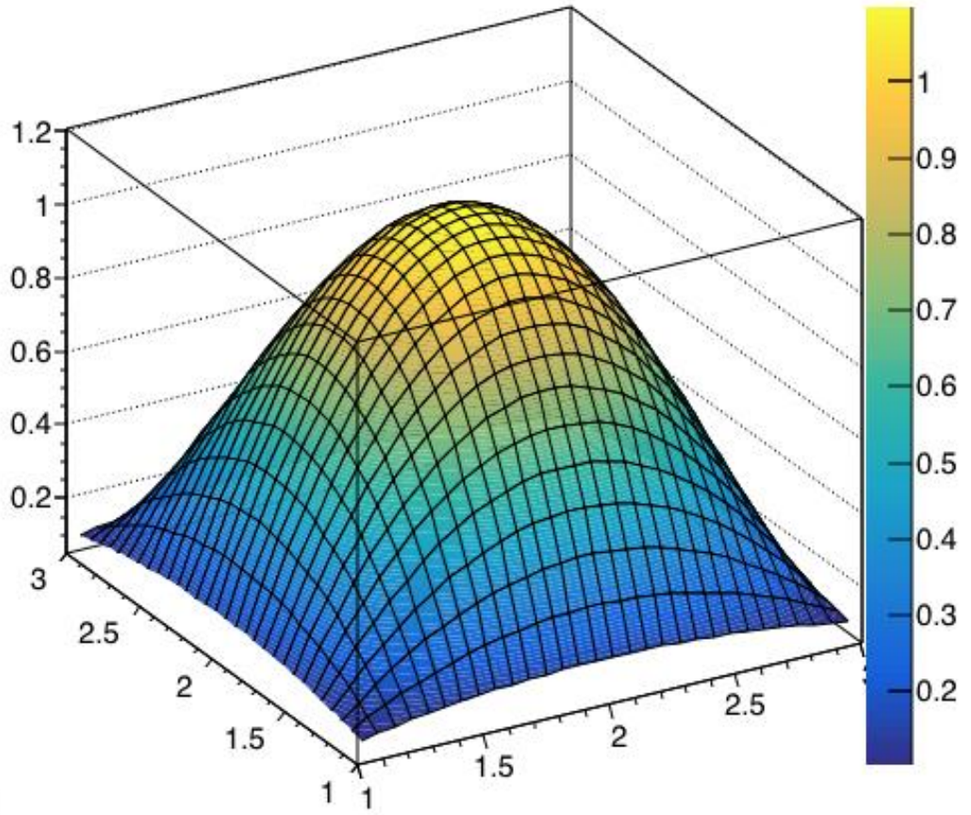
None of the 6 points mentioned in the previous slide were covered.

Then it was changed to Rainbow:

Only points 1 & 2 were covered (visually appealing & intuitive)

[Because of the many issues with the rainbow color map](), the default palette has been changed to:





The 6 points mentioned in the previous slide are covered.

News on Palettes

- The palettes were accessed by number with `gStyle->SetPalette(num)`.
- To set the default palette do: `gStyle->SetPalette()`
- We have now **62 palettes.** Thanks to the following enum they now be **accessed by name**:

```
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
```

default palette

News on Palettes

The new palettes are defined with 255 colors. Many names and colors' definitions have been taken from http://www.rcnp.osaka-u.ac.jp/~noji/colormap.

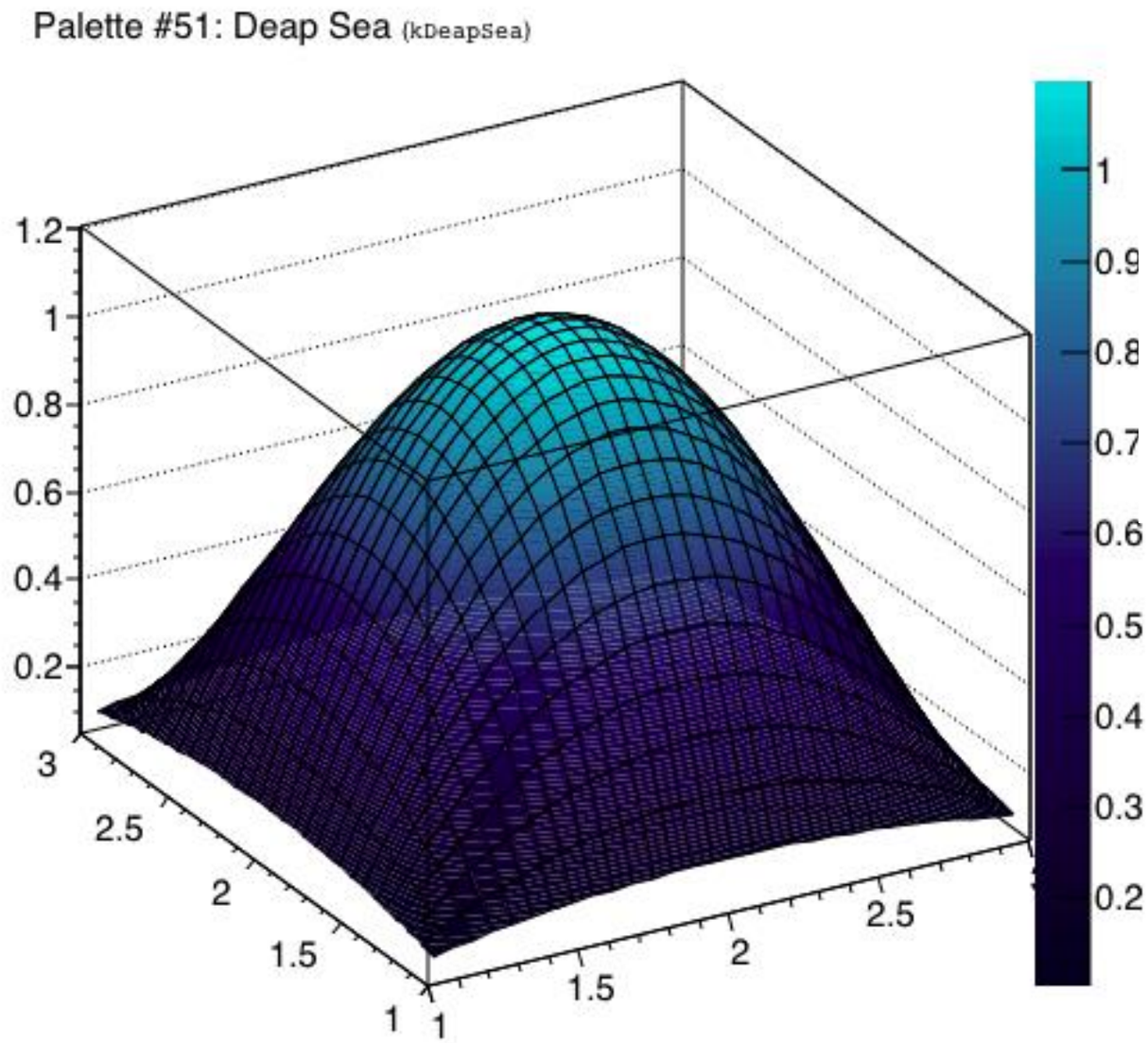
They are defined thanks to a common piece of code:

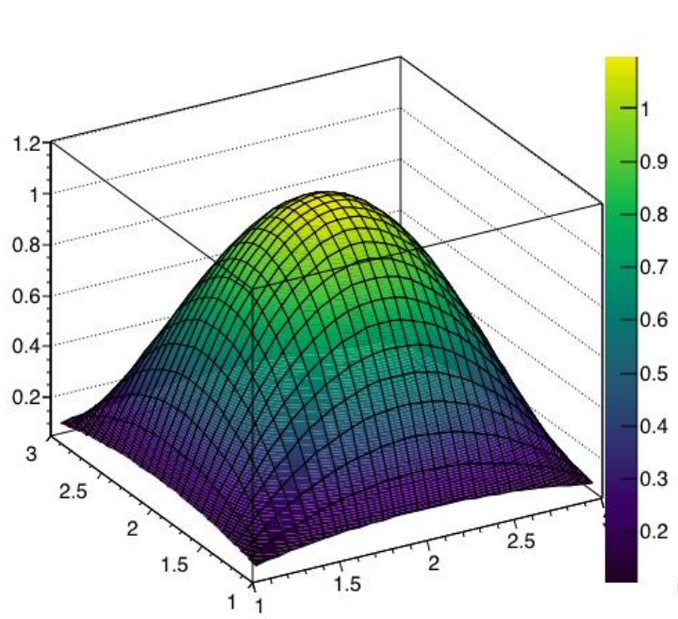
```
Double_t stops[9] = { 0.0000, 0.1250, 0.2500, 0.3750, 0.5000, 0.6250, 0.7500, 0.8750, 1.0000};

Double_t red[9]   = { … };
Double_t green[9] = { … };
Double_t blue[9]  = { … };

TColor::CreateGradientColorTable(9, stops, red, green, blue, 255, alpha);
```

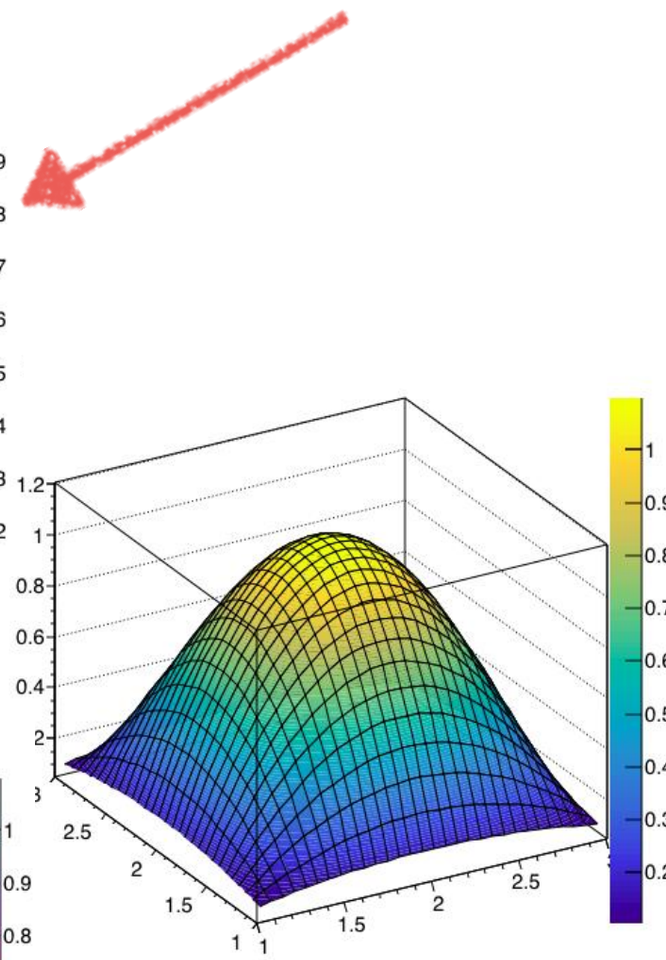
News on Palettes

The following animation goes through all the palettes:

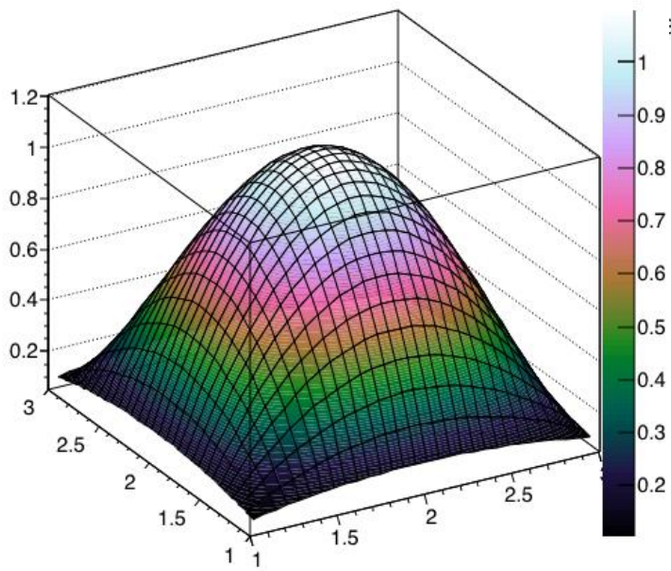
http://betterfigures.org/2015/07/10/a-welcome-development-for-matplotlib/







kBird: new default



https://www.mrao.cam.ac.uk/~dag/CUBEHELIX/

Transparency

Is now an attribute like any other:

The static method `GetColorTransparent(Int_t color, Float_t a)` makes a transparent version of an existing color..

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);
```

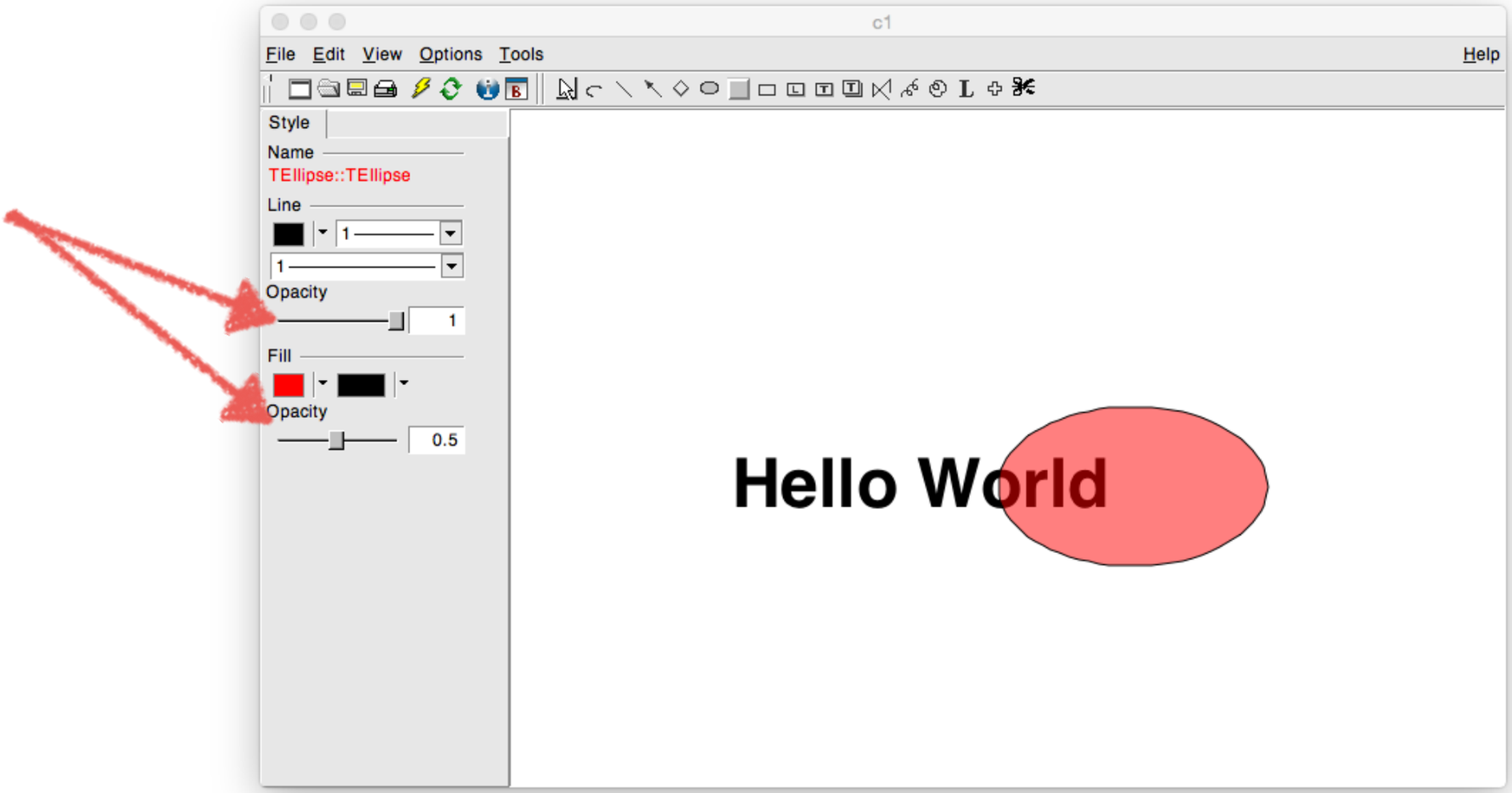
This function is also used in the methods `SetFillColorAlpha()`, `SetLineColorAlpha()`, `SetMarkerColorAlpha()` and `SetTextColorAlpha()`.

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);
```

Transparency

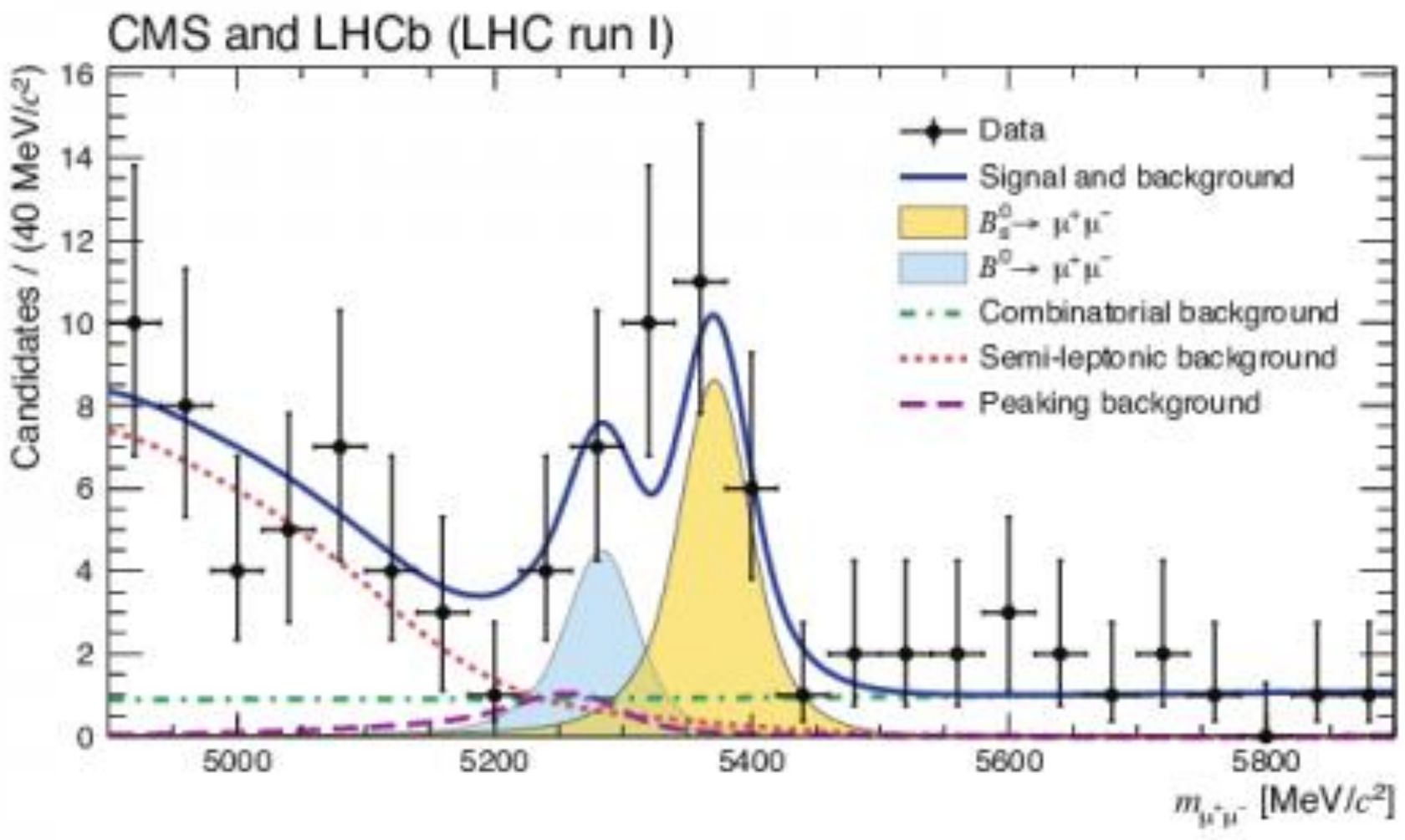
Can be change form the Graphics editor:



Transparency is available on all platforms running OpenGL, or on Mac with the Cocoa backend.
On the file output it is visible with PDF, PNG, Gif, JPEG, SVG, TeX ... but not PostScript.

Transparency

Since it was introduced in ROOT, transparency has been used to produce high quality plots for publications:



Interactive editing

Graphics objects can now be moved “opaque" on a canvas. In the past only the bonding box lines were shown when an object was moved or resized.

This feature is turned **"on"** by default thanks to the following options in `system.rootrc`:

```
Canvas.MoveOpaque:            true
Canvas.ResizeOpaque:          true
```

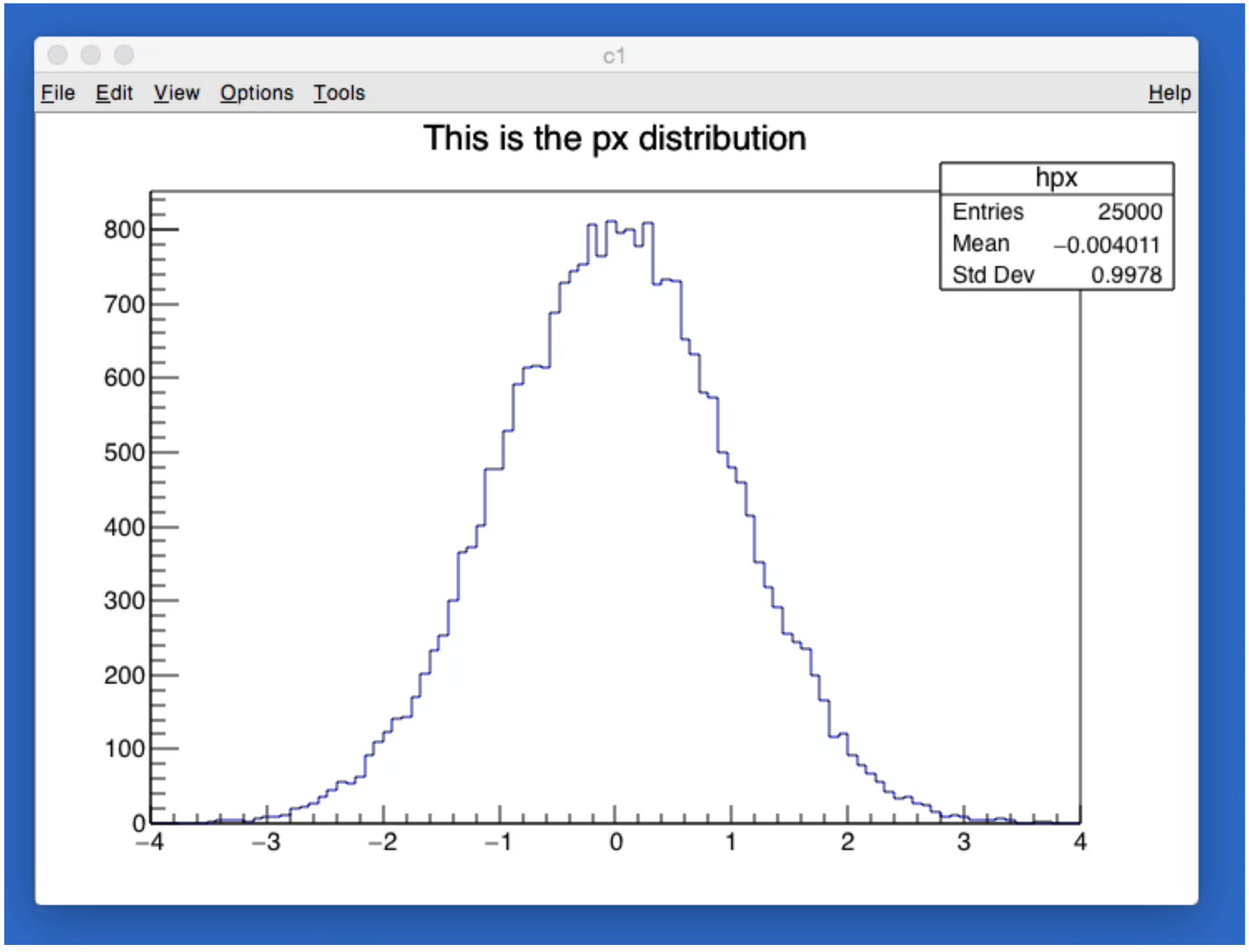
Guide lines allows to interactively align objects and make sure they have the same size .

This feature is turned **"on"** by default thanks to the following option in `system.rootrc`:

```
Canvas.ShowGuideLines:        true
```

Interactive editing

“Move opaque” and "Guide lines" make interactive canvas’ editing much easier…



Saving Canvas as TeX

Being able to generate TeX graphics can be useful for several reasons:

- To have an **easy way to modify the image,** in particular the labels and titles (ASCII file).
- To have the **same font** in all labels, legends, plot titles etc**. as in the text body of a document**, and more generally the **same look and feel**.
- Render **Math formulae using TeX**.

- The TeX text engine is powerful and can render any complex math formulae.
- But more tricky is the graphics rendering: lines, polygons, markers etc ...

One possibility is to render them using PDF or PostScript and render the text using TeX.
But that’s **not very practical** as **two files are needed** to render one picture.

A better way is to use **a dedicated environment like PGF/TikZ.**

Saving Canvas as TeX

*" **PGF** (A Portable Graphic Format for TeX) is a **macro package for creating graphics**. It is platform- and format-independent and **works together with the most important TeX backend** drivers, including pdftex and dvips. It **comes with a user-friendly syntax layer called TikZ**. "*
(http://pgf.sourceforge.net/)

To generate a such file in ROOT it is enough to do:

```
gStyle->SetPaperSize(10.,10.);
hpx->Draw();
gPad->Print("hpx.tex");
```

Saving Canvas as TeX

Then, the generated file (hpx.tex) can be included in a LaTeX document (simple.tex) in the following way:

```
\documentclass{article}

\usepackage{tikz}
\usetikzlibrary{patterns}
\usetikzlibrary{plotmarks}

\title{A simple LaTeX example}
\author{O.Couet}
\begin{document}
\maketitle
The following image as been generated using the TTeXDump class.
To include it in a LaTeX document it is enough to specify the following
three directives at the top of the LaTex document:
\begin{verbatim}
\usepackage{tikz}
\usetikzlibrary{patterns}
\usetikzlibrary{plotmarks}
\end{verbatim}
Then to include the picture ({\tt hpx.tex} in this case) in a LateX
 document it is done the usual way:
\begin{verbatim}
\scalebox{0.3}{\input{hpx.tex}}
\end{verbatim}
\par
\begin{figure}[htbp]
\begin{center}

\scalebox{0.5}{\input{hpx.tex}}

\caption{Image ({\tt hpx.tex}) generated thanks to {\tt TTeXDump}}
\end{center}
\end{figure}
\end{document}
```

A simple LaTeX example

O.Couet

The following image (`hpx.tex`) as been generated using the `TTeXDump` class. To include it in a LaTeX document it is enough to specify the following three directives at the top of the LaTex document ...

```
\usepackage{tikz}
\usetikzlibrary{patterns}
\usetikzlibrary{plotmarks}
```

... and to include the picture in a LateX document with:

```
\scalebox{0.3}{\input{hpx.tex}}
```

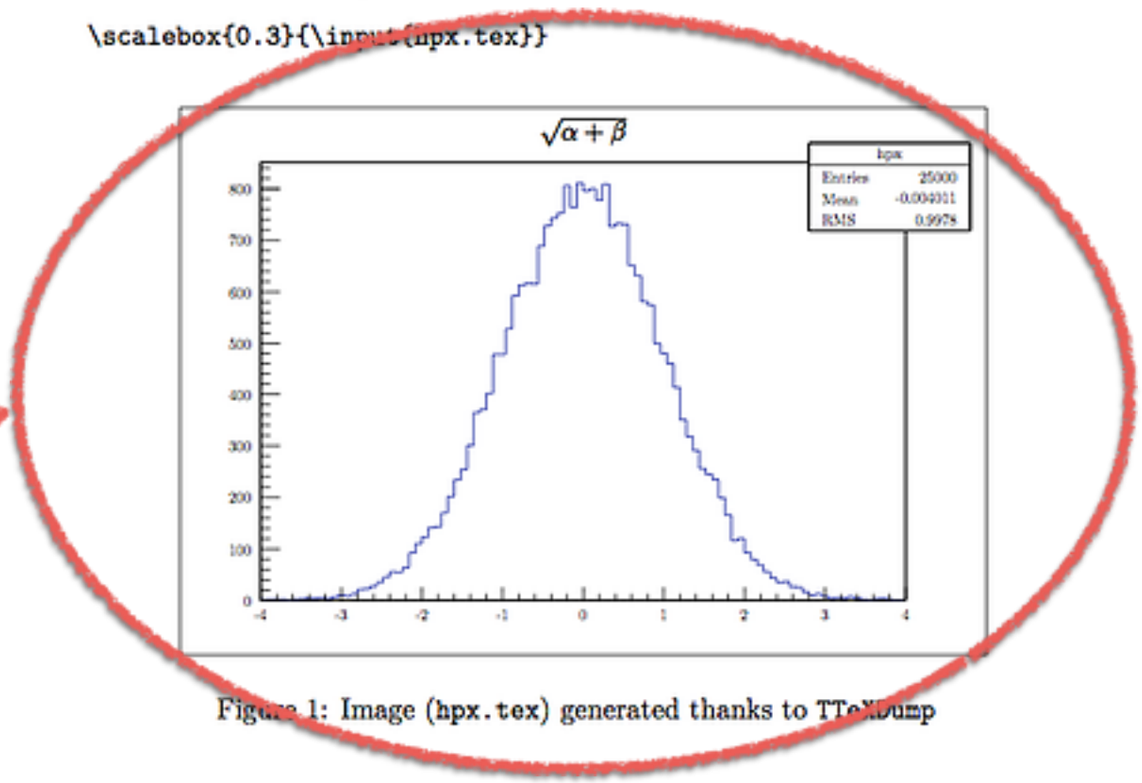


Figure 1: Image (`hpx.tex`) generated thanks to TTeXDump

Saving Canvas as TeX

We received several a good feedbacks about this new feature:

Submitted by andalenavals (not verified) on Fri, 24/04/2015 - 22:05.
It is awesome!…

Submitted by Anonymous (not verified) on Thu, 26/02/2015 - 09:12.
This is indeed a great feature.

Submitted by Anonymous (not verified) on Wed, 09/04/2014 - 16:44.
Thanks for implementing this option! It makes life so much easier!

Submitted by M. Pitzer (not verified) on Tue, 18/02/2014 - 09:47.
I'm very happy that this feature is now implemented in root 5.34!

Submitted by Daniel (not verified) on Tue, 22/10/2013 - 07:27.
Many thanks for implementing this feature! I have been waiting for this for a long time. Awesome!

Submitted by Alexander Voigt (not verified) on Mon, 30/09/2013 - 20:52.
Many thanks for implementing this tex engine! Personally, I was always a bit unsatisfied about ROOT not being able to produce nice tex code for graphics. As far as I know under all the plotting tools only Gnuplot had a comparable tex engine so far.

Submitted by Marcelo (not verified) on Sun, 25/08/2013 - 14:39.
Thanks for this new feature. Some time ago I migrated all my generation of graphs to PGFPlots, which uses pgf internally, for that I had to output my data in tables and afterwards process it with tex. Maybe now I can use the tex file generated by root directly. That would be very nice.

Submitted by DRD (not verified) on Sun, 18/08/2013 - 23:41.
I've been a big fan of PGF/TikZ for quite some time now -- it's very nice to see this available as part of ROOT graphics, thanks!

Submitted by Luca Baldini (not verified) on Sun, 18/08/2013 - 14:28.
thanks for making this happen!

OpenGL Backend

ROOT has now 4 main graphics backends for screen output:

1) **X11**, available on all platforms (except Windows) but does not support transparency.

2) **Win32 / GDK**, on Windows, does not support transparency.

3) **Cocoa/Quartz**, support transparency but is available only on MacOSX.

4) **OpenGL** support transparency and is available everywhere.

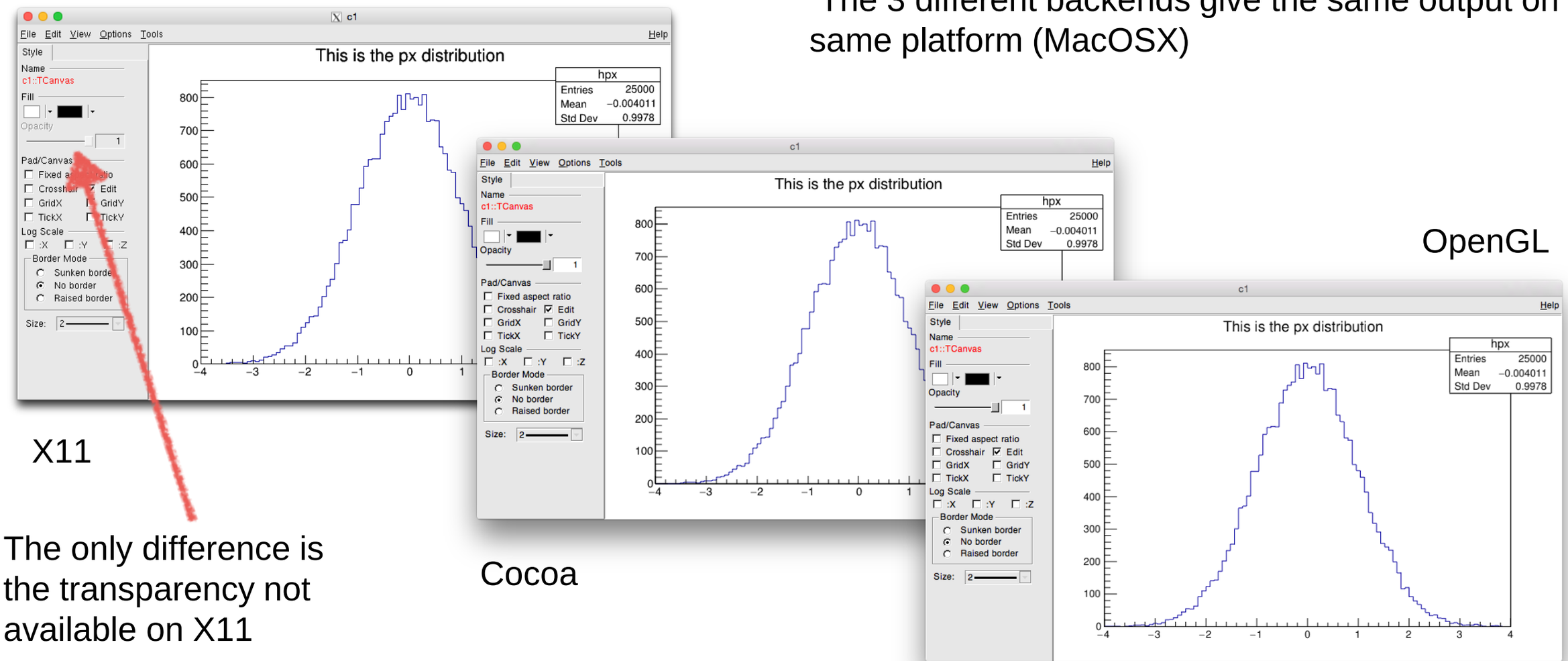
OpenGL Backend

If it exists, **OpenGL** is linked at build time with ROOT.

Then it can be turn **"on"** as being the default backend by setting the following option in `system.rootrc`

```
OpenGL.CanvasPreferGL:                    1
```

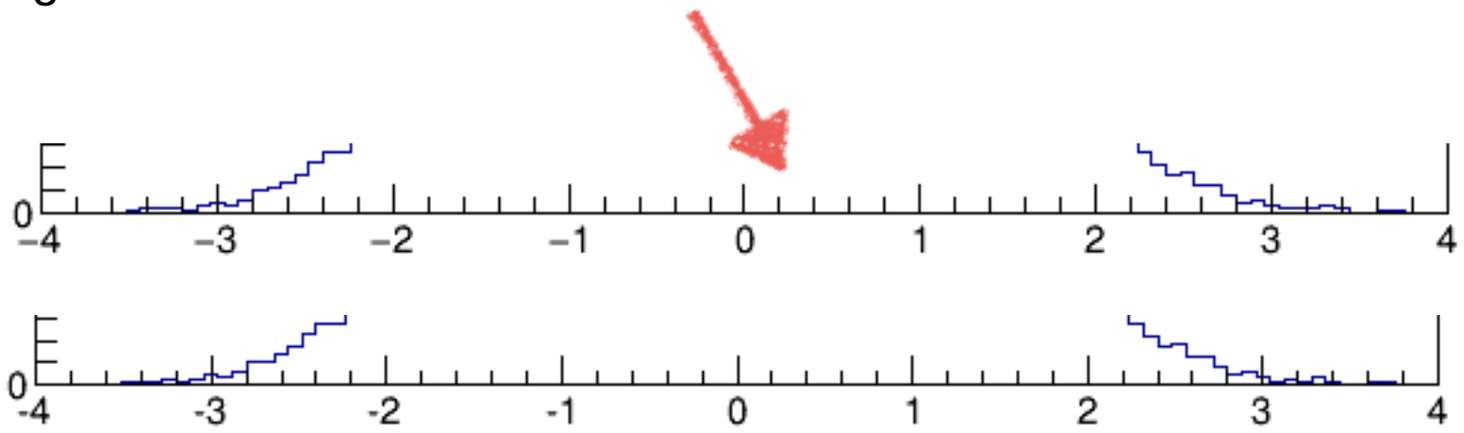
The 3 different backends give the same output on the same platform (MacOSX)



Improvements

Here is a partial list of the improvements done in various graphics area.

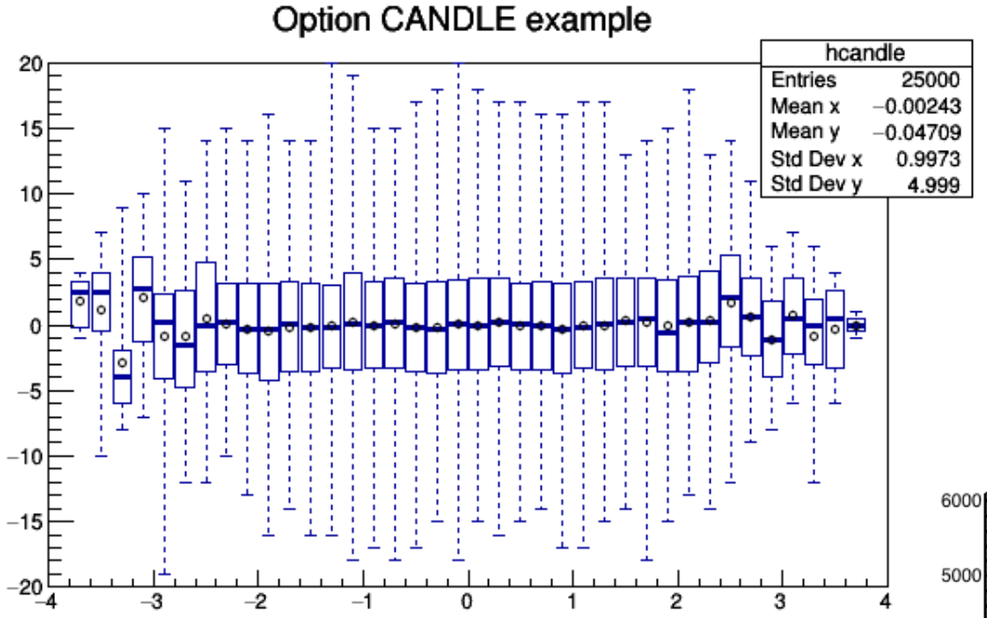
- Better line width matching with screen in PDF/PS output .
- Transparency implemented in TeX output.
- Implement missing math symbols in SVG output.
- Make TMathText work with FTGL (OpenGL text).
- Line width = 0. Is now a valid value. It is useful to hide lines, frame, etc… during interactive editing.
- Implement option `pads` for `TMultigraph`, equivalent to the one in `THStack`. It allows to draw all the `TGraphs` in separated pads.
- Implement typographically correct minus sign for axis labels and stats.



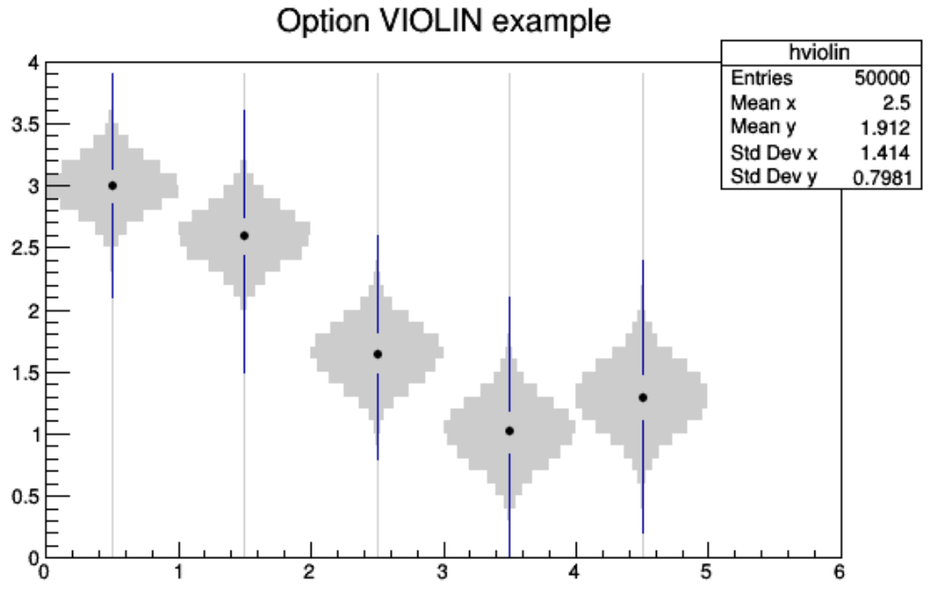
New drawing options …

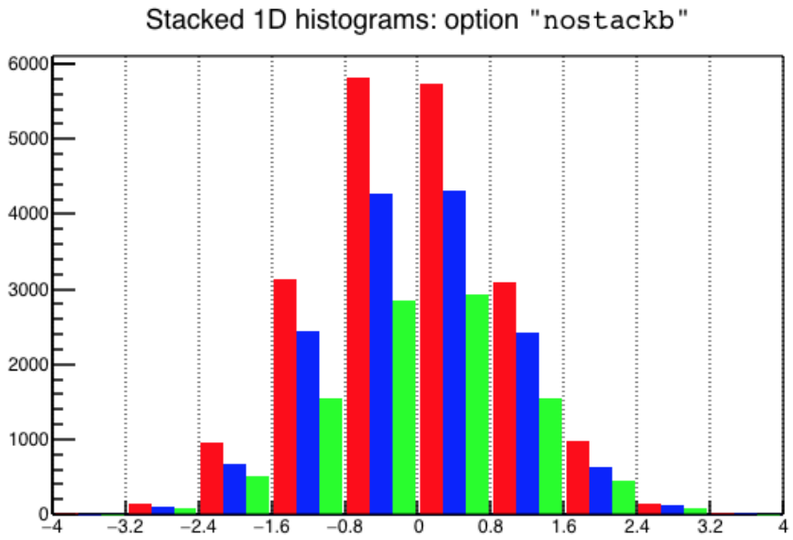
ROOT users constantly request new visualisation techniques. They sometimes are implement in close collaboration with users. Some improvements are also required for existing visualisation techniques.

- New drawing option “candle” and "violin" for **2D histograms**.
- A THStack drawing option to draw the histograms next to each other as bar charts



"The candle plot is a standardised way of displaying the distribution of data based on the five number summary: minimum, first quartile, median, third quartile, and maximum."

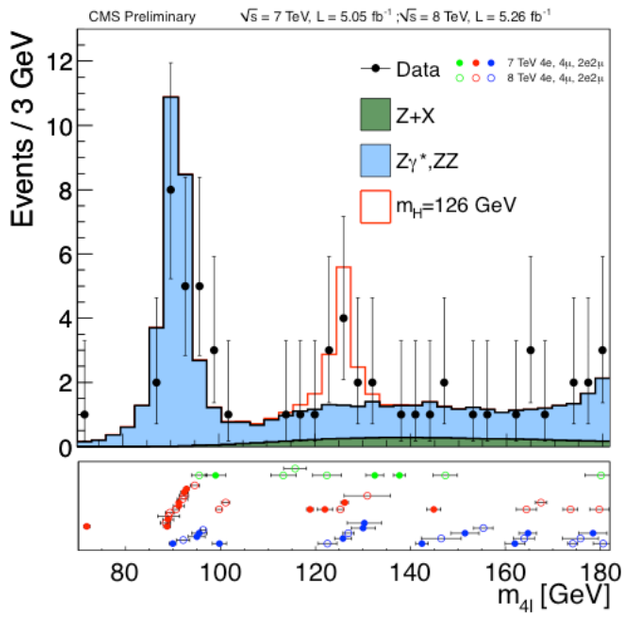




A violin plot is a candle plot that also encodes the pdf information at each point. Quartiles and mean are also represented at each point, with a marker and two lines.

Future work

- The first message we got from the survey consolidate the existing graphics tools.
- Nice plots ready for publication must be simpler to produce, by providing:
  - Ratio plots
  - "Undo" in interactive editing
  - Fixed fonts size as default
  - Optimised style for the legend



- Enhance the plotting options for horizontal histogram plots.
- Symlog scale