



ROOT Graphics: Recent Developments and Future plans

Olivier Couet (CERN)

Valeriy Onuchin (IHEP)

Timur Pocheptsov JINR (supported by INTAS grant)



- Introduction
- 2D Graphics latest developments
 - TPolarGraph
 - TPie
 - Exclusion Plots
 - Spectrum Painter
 - TAsImage/TImage/TImageDump
- Next 2D Graphics Developments
- 3D Graphics latest developments
 - GL in Pad: 2D and 3D mixed
 - TH2, TF2, TH3, TF3 representations
 - Parametric Functions
 - 3D Interactions
- Next 3D Graphics Developments
- Conclusion

The **ROOT Graphics work package** is in charge of:

1. The low and high level 2D graphics (*O.Couet*).
2. The graphics output (PostScript, PDF, gif, jpeg etc ...) (*O.Couet, V.Onuchin*).
3. The image processing (TASImage/TImage) (*V.Onuchin*).
4. The 3D graphics based on OpenGL: GL-in-Pad and GLViewer (*T.Pocheptsov, M.Tadel*).

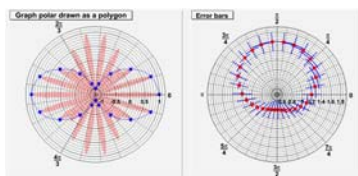
This talk presents the main developments done in these various domains since the last ROOT workshop (in particular in 2D and 3D graphics).

A special emphasis will be made on the 3D graphics developments made by *T.Pocheptsov*.

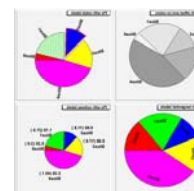
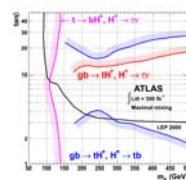
Finally, it will indicate the future developments foreseen (mainly in 3D graphics using OpenGL).

Many developments, improvements, and bug fixes have been done in the 2D graphics part of ROOT since the last workshop. The next slides present the most significant 2D graphics' **developments**:

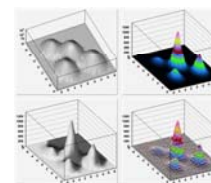
TPolarGraph



Exclusion Plots

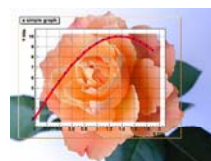


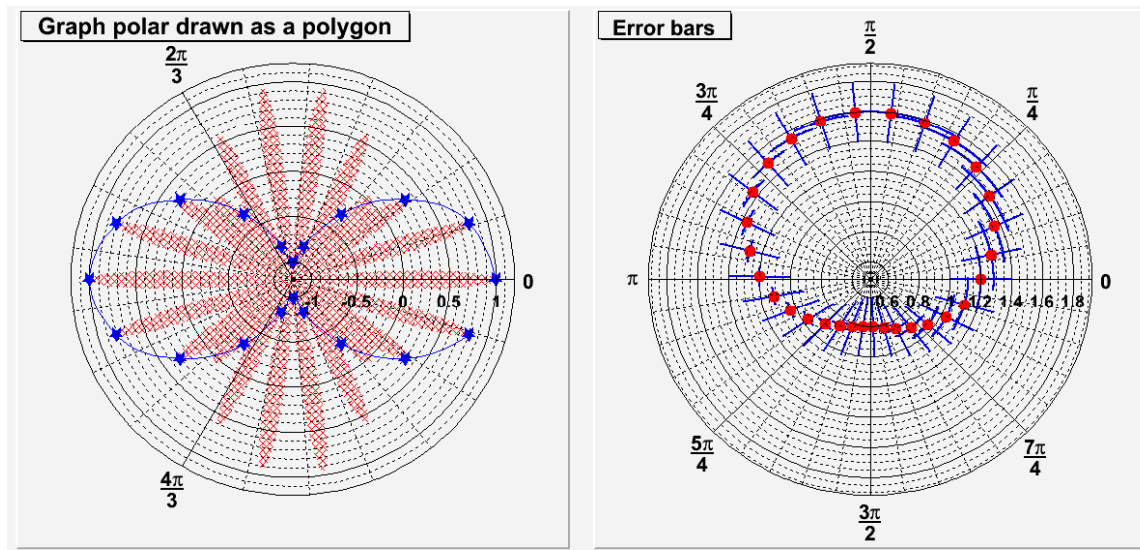
TPie



Spectrum Painter

TASImage/TImage/TImageDump





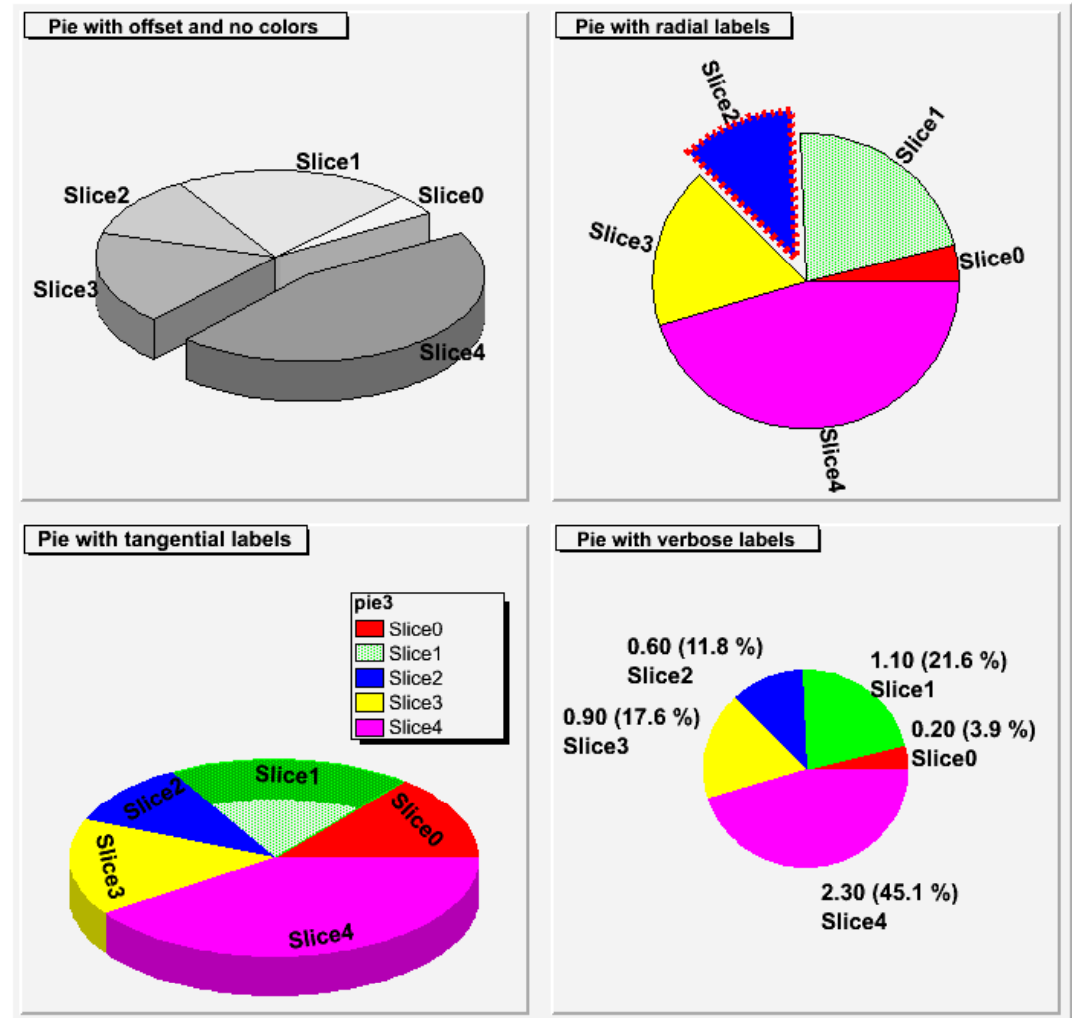
The initial version of this class was developed by *Sebastian Boser* and extended later by *Mathieu Demaret*.

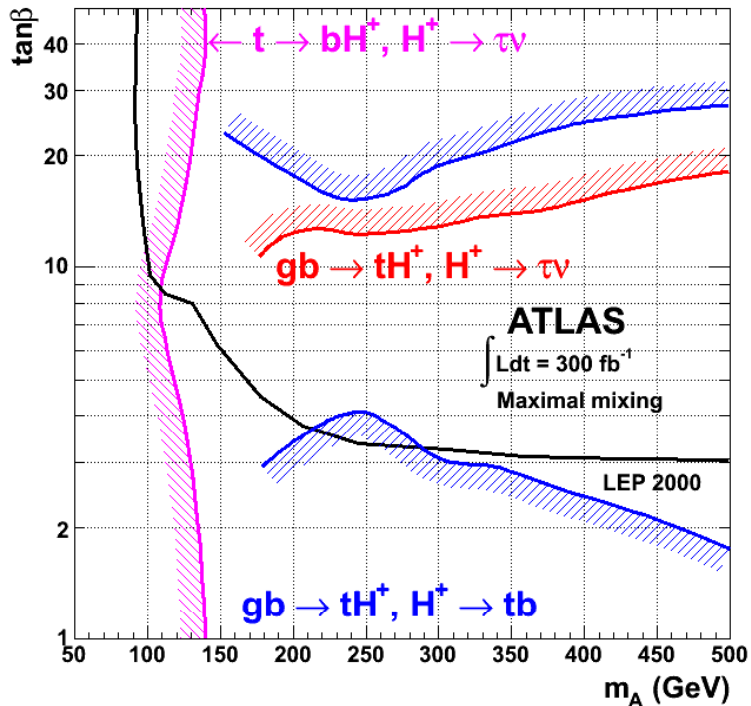
It creates a **polar graph** (including error bars).

- **TGraphPolar** is a **TGraphErrors** represented in **polar coordinates**.
- It uses the class **TGraphPolargram** to draw the **polar axis**.
- Several settings are available: labeling in PI, label orientation etc...
- Needs more improvements and developments (Editor).

This class was developed with *Guido Volpi*.

- It allows to define and draw **pie charts**.
- Various options to draw a pie chart (flat, 3D effect, label format etc ...)
- Flexible and intuitive way to manipulate the drawing interactively.
- Can also be used to draw TH1 histograms: a **TPie** can be **constructed from a TH1**.





Graphs showing an **exclusion zone** are very frequent. Before this new facility there was no easy way to draw them (several graphs were needed for one plot).

A `TGraph` drawn with the options `"C"` or `"L"` can have an exclusion zone.

It is activated when the absolute value of the `TGraph` line width (set thanks to `SetLineWidth`) is greater than 99.

In that case the line width number is interpreted as:

$$100*ff+ll = ffl1$$

- `ll` represents the **normal line width**.
- `ff` is the **filled area width**.
- The line width sign defines on which side of the graph the hatches are drawn.
- The fill area attributes are used to draw the hatched zone.



The “Spectrum Painter” is a **surface and contours drawing package**. It has been developed by *Miroslav Morhac* several years ago in C and was recently transformed in a ROOT class.

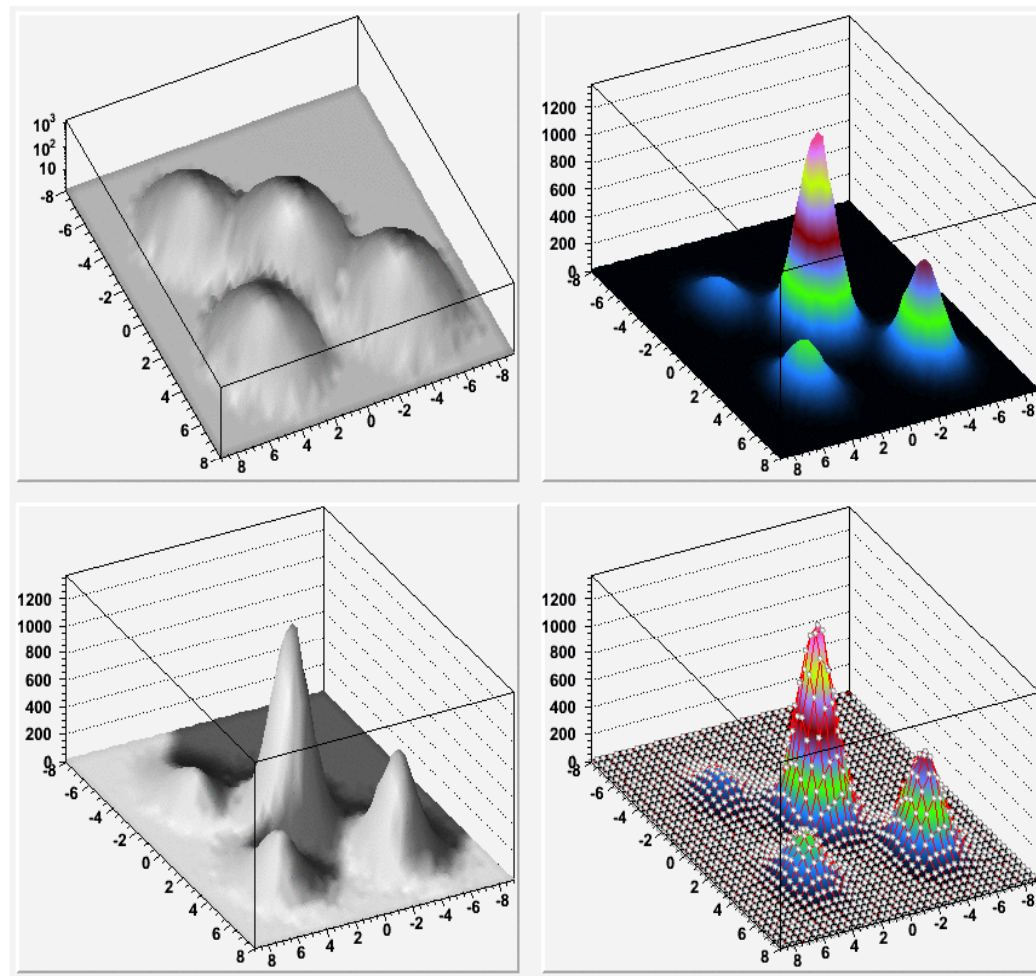
It is called by `THistPainter::Paint` when drawing a TH2 with the "SPEC" option.

It offers many different drawing options which can be **combined together**.

It is a **complement** of the existing SURF, CONT and LEGO options.

A complete description of its capabilities is given in the help of:

```
TSpectrum2Paint::PaintSpectrum()
```

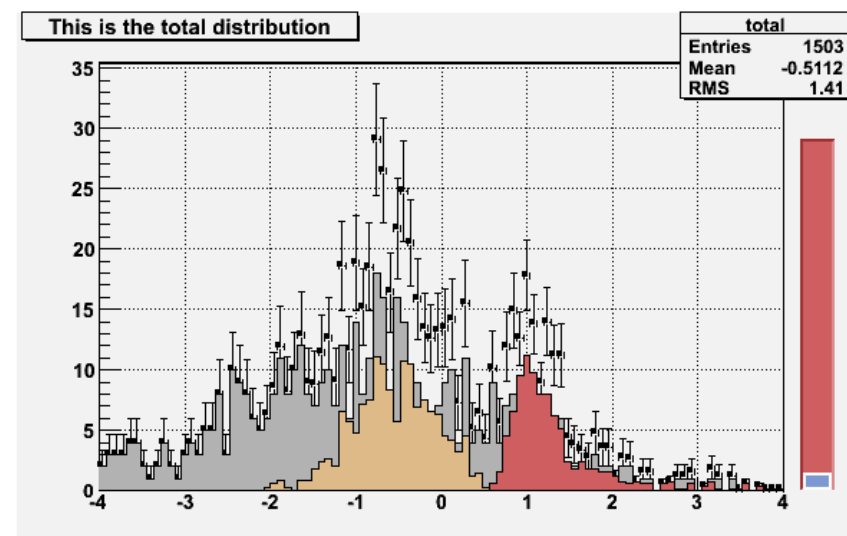




- Animated gif files can be generated directly in ROOT. No need for an external tool like **gifsicle**. (`$ROOTSYS/tutorials/image/hsumanim.C`)



- The tutorial (`$ROOTSYS/tutorials/image/trans_graph.C`) demonstrates, among others things, how to manipulate images in order to make them transparent.



- Picture output of **any formats** (gif, png, jpeg, etc...). Can be generated in **batch mode**. Previously it was only possible with PostScript or PDF formats.

The list of **developments** needed in 2D graphics is quite long here are some examples:

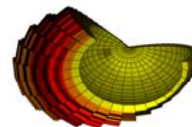
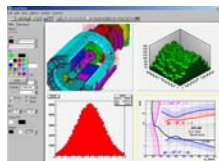
- Complete TPDF and TSVG to match exactly the TPostScript capabilities.
- Revisit the way TGraph is painted. No need to use an histogram.
- Reverse X and Y axis.
- Add missing symbols in TLatex.
- Extend the list of available text fonts.
- Extend the list of markers' types. User definable markers.
- Improve the contour plots (contour labels, etc...).
- Improve lego and surface plots (improve SAME option, etc ...).
- Revisit the 1D histogram's horizontal painting.
- Implement a way to draw many histograms in one go by dividing automatically the pad with the needed number of zones.
- Plotting Histograms' overflows.
- Paint 3D triangles.
- TGaxis rewrite.
- Etc, etc, etc ...

The 3D Graphics GL developments have been done in:

1. The possibility to draw graphics produced by GL in a 2D pad (**GL-in-Pad**).
2. Data sets representations using GL.
3. The **GL-Viewer** (this will be presented by *M. Tadel* in the next talk.)

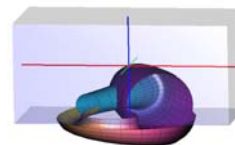
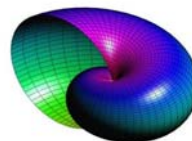
The next slides will present the following **developments**:

GL in Pad: 2D and 3D mixed

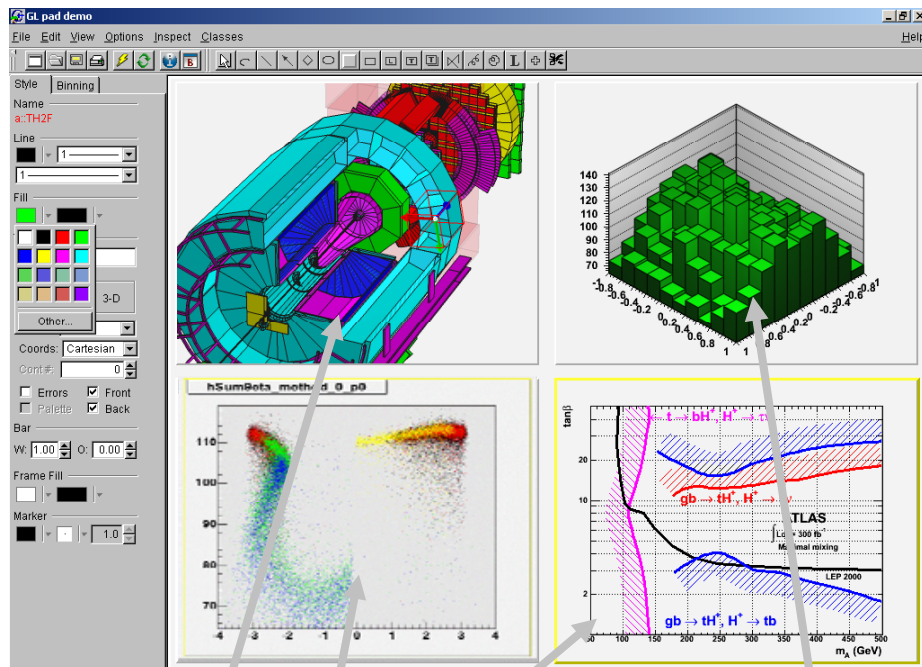


TH2, TF2, TH3, TF3 representations

Parametric Functions



3D Interactions



A TCanvas can contain pads with GL graphics rendering. It is enough to declare the canvas the following way:

```
TCanvas c("glc","c");
```

or to do:

```
gStyle->SetCanvasPreferGL(kTRUE);
```

After this all the new canvas will “prefer GL”. It means that:

- **Detectors geometries** will be painted in the pad using GL,
- It is possible to paint **3D graphics representations** like Lego, Surface, etc ... using GL.
- **Other pads** are painted using basic 2D.

PostScript files generated from such “mixed” canvases are a **combination** of **gl2ps** output (for the GL parts) and **TPostScript** output (for the 2D part).



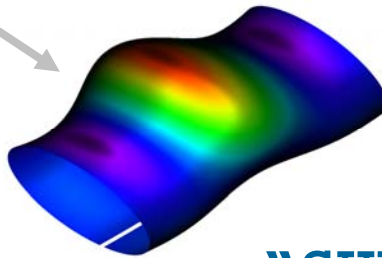
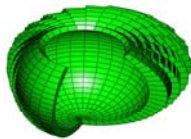
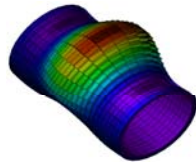
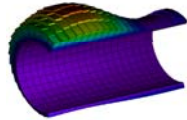
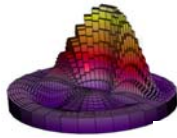
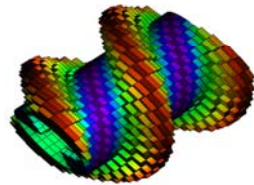
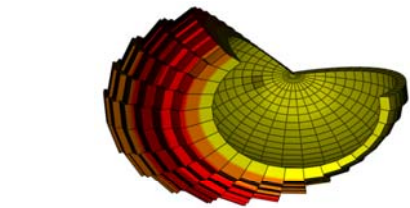
TH2, TF2, TH3, TF3 representations



It was already possible to display TH2, TH3, TF2, TF3 using combination of options like "SURF", "LEGO", "POL", "CYL", etc ...

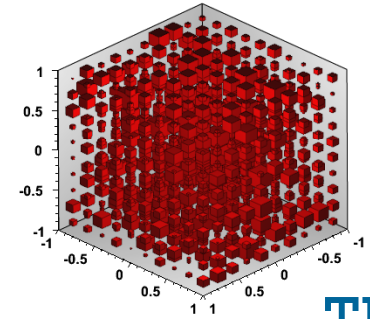
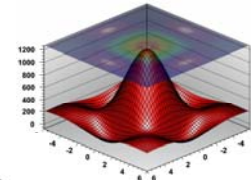
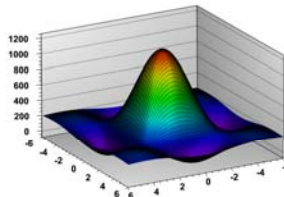
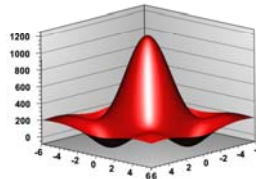
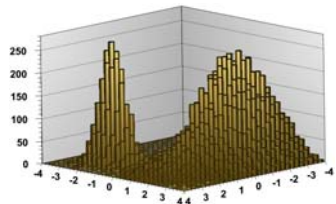
The same options are available in "GL-mode". It is enough to add the option "gl" in the list of option and to draw in a canvas which "prefer gl"
Example:

```
h2->Draw("gl surf1 cyl");
```

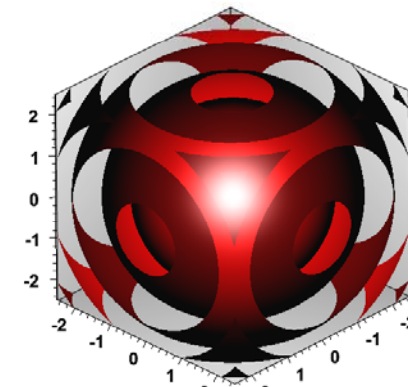
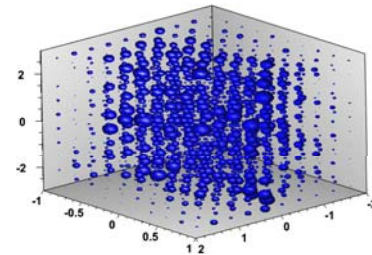


"LEGO"

"SURF"



TH3



TF3



Parametric Functions

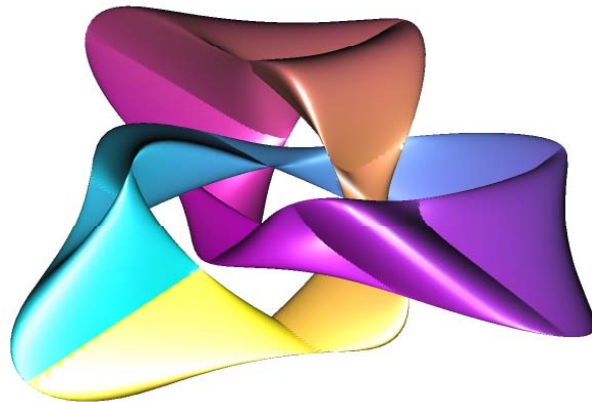
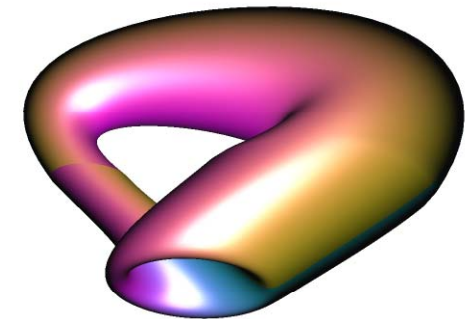
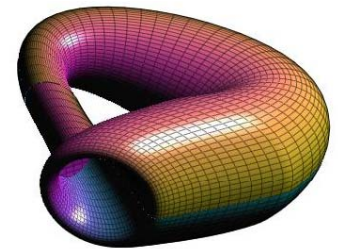
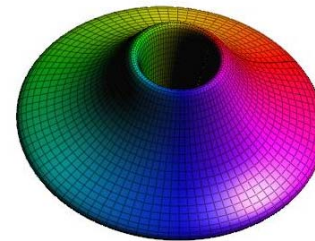
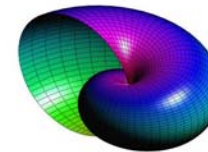
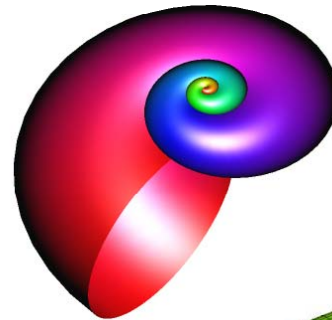
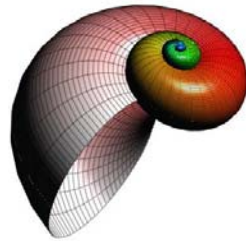
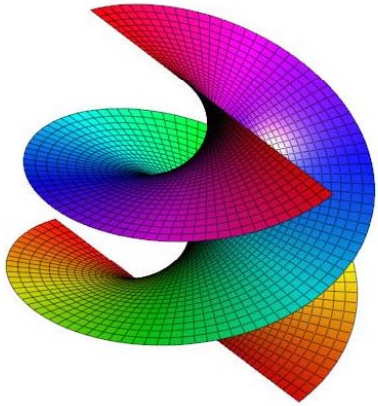


A parametric surface is defined by three functions:

$$S(u,v) : \{x(u,v), y(u,v), z(u,v)\}.$$

Example, a conchoid:

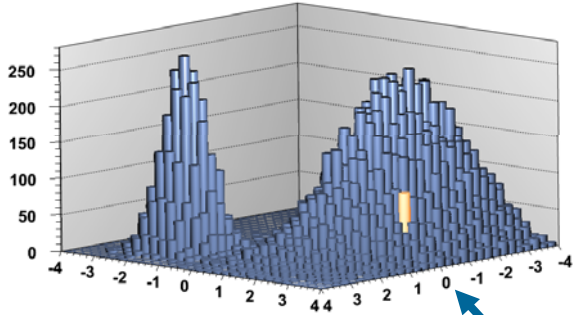
```
TGLParametricEquation p1("Conchoid",
    "1.2^u*(1+cos(v))*cos(u)",
    "1.2^u*(1+cos(v))*sin(u)",
    "1.2^u*sin(v)-1.5*1.2^u",
    0., 6*TMath::Pi(), 0., TMath::TwoPi());
```



Parametric surfaces can be drawn only using a GL-pad

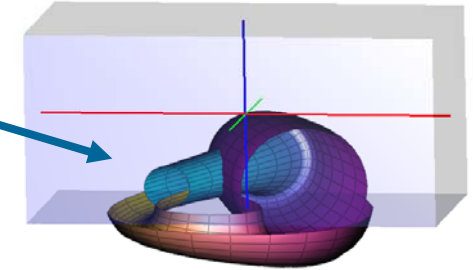


3D Interactions

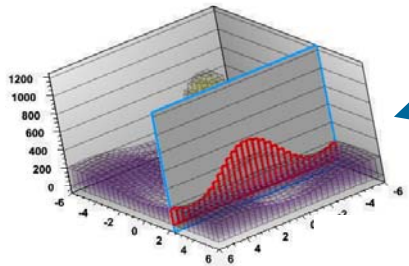


Bins are highlighted when the cursor runs over.

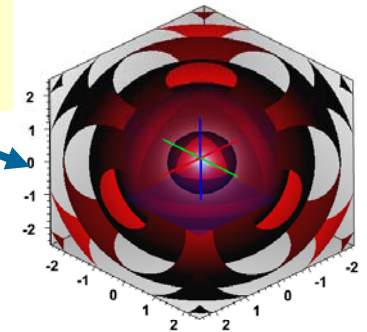
Press "s" to change color scheme
Press "w" to switch on/off wireframe



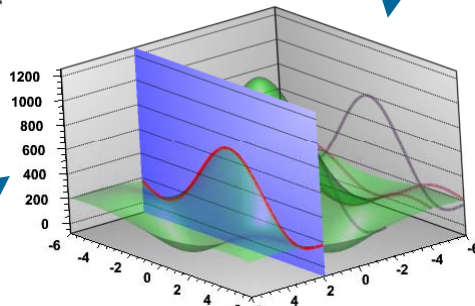
Box cut: press "c" to enable/disable.
Move by click on the axis.



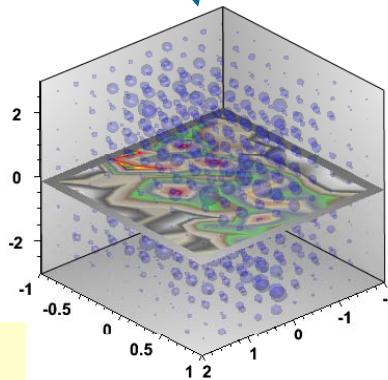
left shift + left mouse button pressed :
move back plane.



Double-click on the left mouse
button: return to initial view



Project on back plane: press "p" .



On TH3 the 2D contour is drawn
in real time on the cutting plane.



Next 3D Developments (1)



The **GL-in-Pad** mechanism involves some **memory copies** (GL-Buffer to pixmap, pixmap to screen) and therefore, is much **slower** than the direct GL rendering done in the **GL viewer**.

It has been now demonstrated that it's **possible to mix** standard **2D graphics** and **3D GL graphics** in the **same pad**. In order to benefit fully from the GL graphics accelerators, the next plans are to render the 2D graphics in a the GL-Viewer (ie: a true graphics accelerated context).

This is on going work, involving a lot of prototyping. (see *M. Tadel* talk).



Next 3D Developments (2)



One important missing basic primitive in the GL-Viewer is the **Text**:

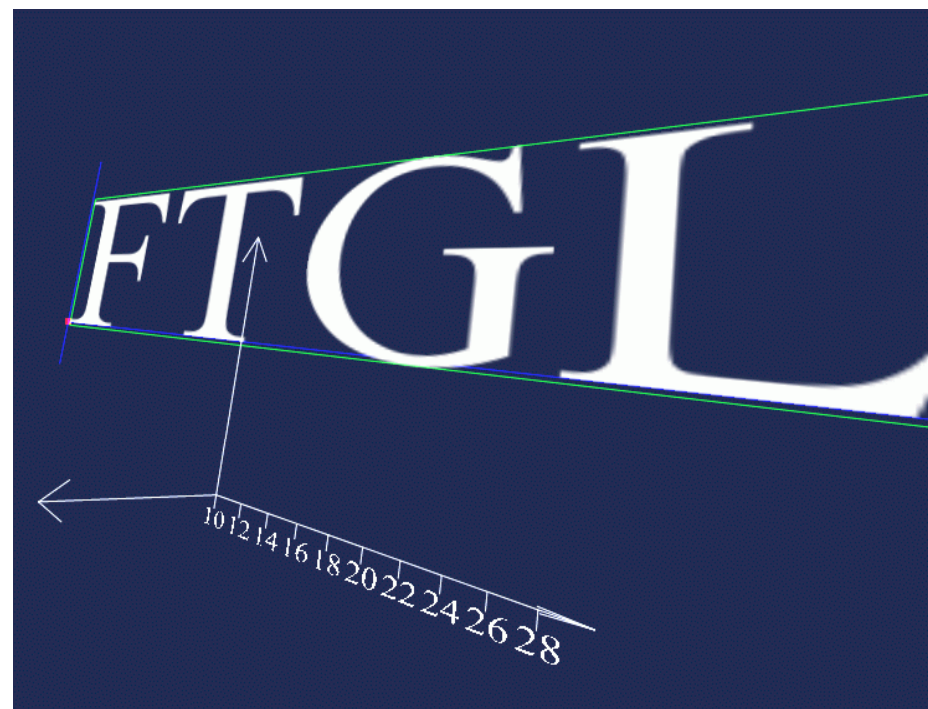
A new class `TLatex3D`, inheriting from `TLatex`, is needed to draw text in GL scenes. After some investigations we found that the best way is surely to base it on the FTGL package (see next slide).

One other urgent class is the true 3D axis. Right now the axis drawing on the GL-plot in the pad is done using the 2D class `TGaxis`. The result is not good. A new class `TGaxis3D` is needed to draw axis in 3D space. It will have some common parts with `TGaxis`. It will use `TLatex3D` to draw the axis labels.

FTGL is a library for rendering **TrueType fonts in OpenGL**. It is based on the newest FreeType2 API. **FTGL** supports all the following rendering types:

- Outline.
- Polygonal.
- Texture mapped.
- Bitmap.
- Anti-aliased pixmap.

We have made a small prototype of 3D axis. We were pleased to see that the drawing speed of FTGL text is very fast (real time when the image is rotated).



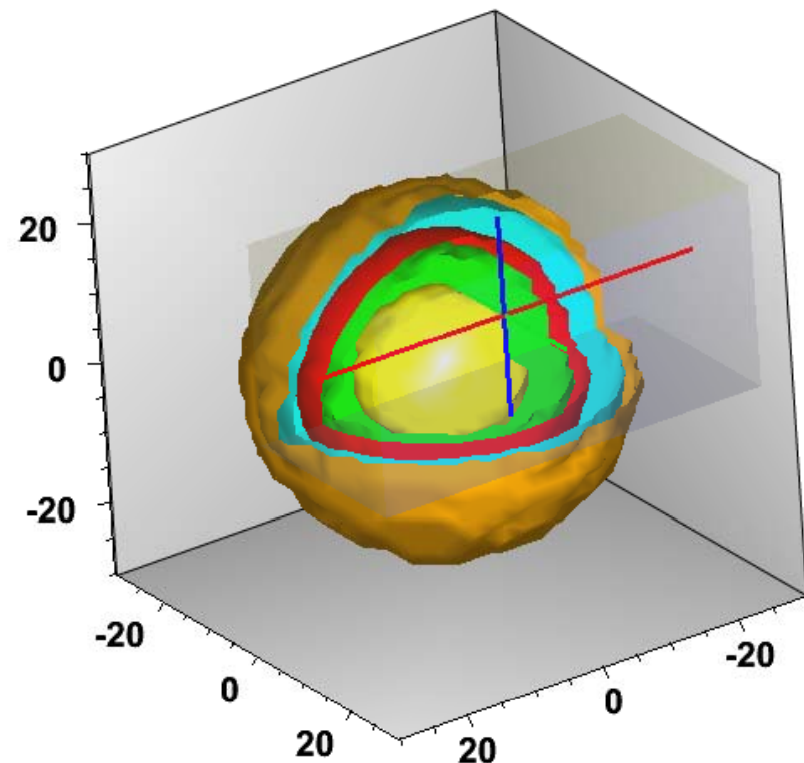
See <http://www.opengl.org/resources/features/fontsurvey/#ftgl>

Possibility to paint TH3 using iso-surfaces (iso-3D contours). This is the very last development done by *T.Pocheptsov*. The final implementation will be able to draw “**3D contour plots**”.

Like for the 2D contour plots, it will be possible to define the **iso-surfaces** either **automatically** (some equidistant surfaces between vmin and vmax) or with user **define values**.

```
void iso()
{
  gStyle->SetCanvasPreferGL(true);
  TH3C * volume = new TH3C("V","V",
                          20,-20,20, 20,-20,20, 20,-20,20);
  TCanvas * canvas = new TCanvas("C","C",600,600);
  float x,y,z;

  for(x=-20;x<20;x+=1)
    for(y=-20;y<20;y+=1)
      for(z=-20;z<20;z+=1)
        if((x*x+y*y+z*z)<200)
          volume->Fill(x,y,z,1);
  canvas->Draw();
  volume->Draw("gliso");
}
```





Several new developments have been achieved in 2D and 3D graphics since the last ROOT workshop.

The main future developments will be in the integration of the 2D and 3D graphics in the GL-Viewer ie: a 3D graphics accelerated context.

We can even hope that, in the long term, X11 and win32 graphics will be fully replaced by accelerated GL.