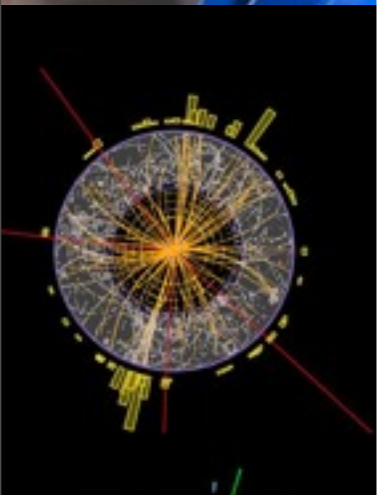
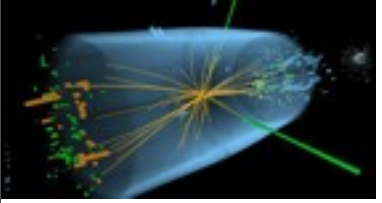
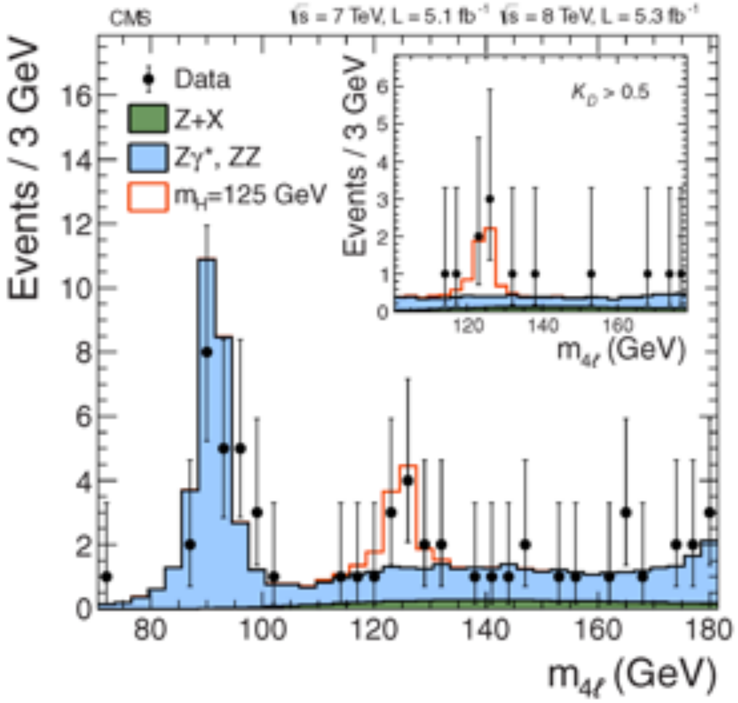




Working with Histograms

ROOT Training at IRMM

26th February 2013

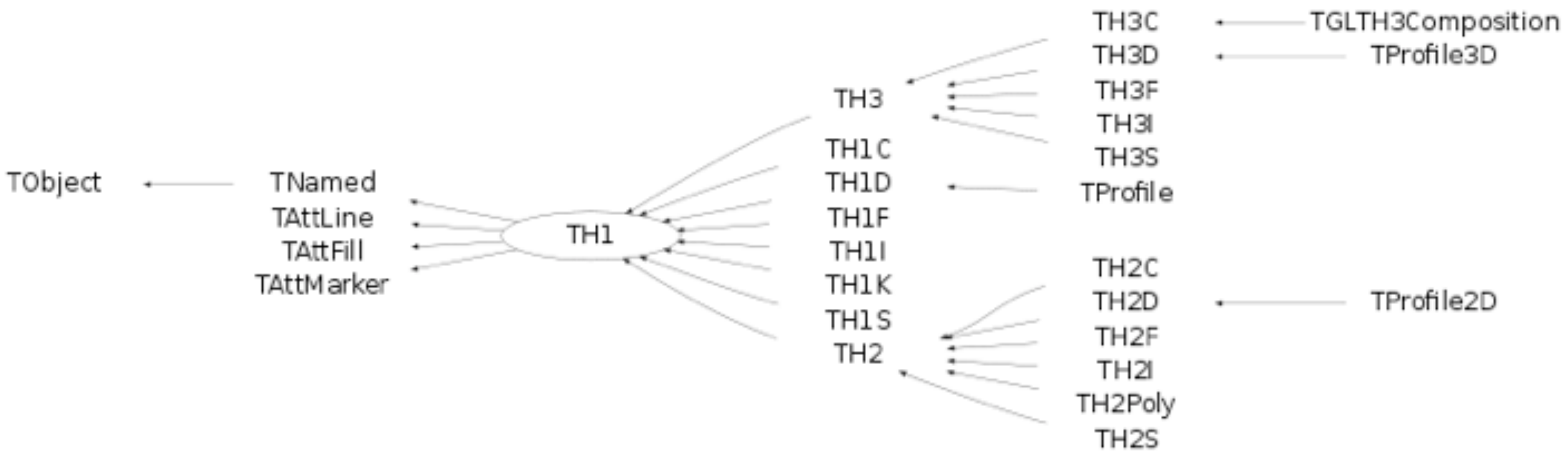


Dr Lorenzo Moneta
CERN PH-SFT
CH-1211 Geneva 23
sftweb.cern.ch
root.cern.ch

Working with Histograms

- We have seen:
  - what is a histogram;
  - how to create a one-dimensional histogram;
  - how to draw the histogram
- We will see and work on with the exercises
  - how to extract information from an histogram
  - how to manipulate histograms
  - histograms in multi-dimension
  - what is a profile histogram
  - weighted histograms
  - sparse histograms
- More detailed on the ROOT Graphics Pad and the Canvas
- Visualization techniques in ROOT

Histogram Classes



- TH1 is the base class for all Histogram classes.
- TH1,TH2,TH3 are generic. Specialized should be used for constructing the objects.
- Most used classes are TH1(2,3)F and TH1(2,3)D.
- For non-equidistant bins use a separate constructor:

```
TH1D * h1 = new TH1D(“h1”,”Example histogram”,nbins,xbins);
```

  - `xbins:` array of (`nbins+1`) values with the bin-edges

Axis and Bins

- The histogram class has an axis class which contains the bins

```
TAxis * axis = h1->GetXaxis();
```

  - one can query the number of bins, lower/upper bin edge from the axis :

```
axis->GetNbins();
axis->GetBinLowEdge(bin_number);
axis->GetBinCenter(bin_number);
axis->GetBinUpEdge(bin_number);
```

  - for the corresponding bin number given x value use

```
bin_number = axis->FindBin(x_value);
```

  - one can set the axis range (e.g. for zooming the histogram)

```
axis->SetRange(firstbin,lastbin);
```

  - N.B. : axis bin number starts from 1
    - bin number 0 is the **Underflowbin**
    - bin number `nbin+1` is the **Overflowbin**

Getting Data From an Histogram

- To extract content and error from bin number `ibin` of the histogram `h1`:

```
root [1] h1->GetBinContent(ibin)
(const Double_t)1.10000000000000000e+01
root [2] h1->GetBinError(ibin)
(const Double_t)3.31662479035539981e+00
```

- By default histograms have a bin error equal to √N, where N is the bin content.
  - It is assumed that the observed bin content follows a Poisson distribution.
- One can set a different error for each bin:

```
root [1] h1->SetBinError(ibin,error)
```

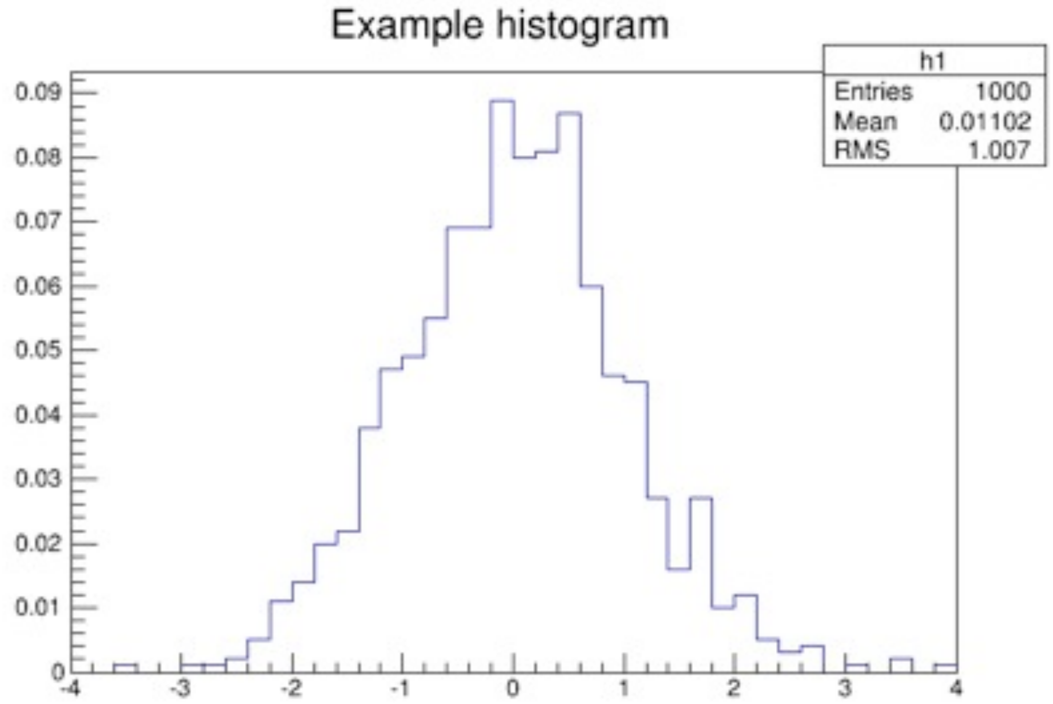
- N.B: when setting the error in each bins, the histogram will store them in an internal array and will change its default style when drawn

Histogram Operations

- Histogram scaling (normalization)
  - useful for plotting histogram in the same pad
  - useful for seeing histogram as an estimate of a probability density function (PDF)

```
scale = 1.0/h->Integral();
h1->Scale(scale);
```

probability of observing an event in a given bin



N.B. : After scaling the error will not be correct. We will see later how to have correct errors

Histogram Operations (2)

- Add Histograms:
  - merge two histograms which have same axis:

```
TH1D *h3 = new TH1D(“h3”,”h1+h2”,nbin,xmin,xmax);
h3->Add(h1,h2);
```

  - can also be used to subtract histograms

```
h3->Add(h1,h2,1.,-1.);
```

- Divide Histograms:

```
TH1D *h3 = new TH1D(“h3”,”h1/h2”,nbin,xmin,xmax);
h3->Divide(h1,h2);
```

  - if h1 is a subset of h2, the bin content of h3 is binomially distributed ➠use option “B” to get the correct bin errors

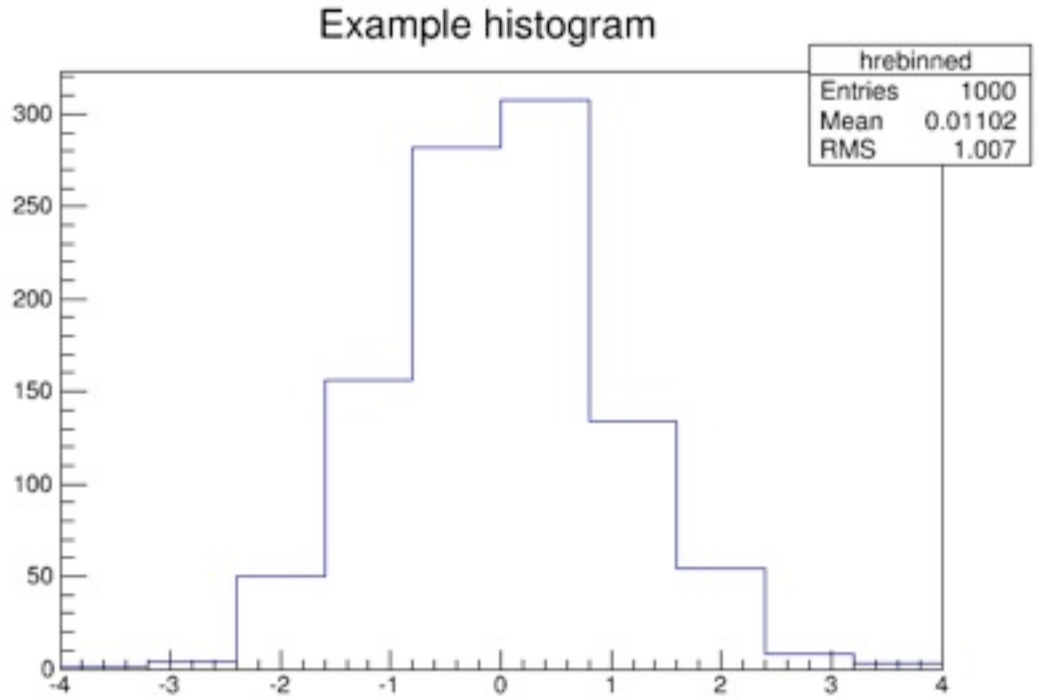
```
 h3->Divide(h1,h2,1.,1.,”B”);
```

Histogram Re-Binning

- Rebin: merge bins together.
  - Possible to merge adjacent bins in a given group,
    - e.g. an histogram with 100 bins can be re-binned in a histogram with 25 bins .
  - Possible to merge bins using new bin edges provided by the user, i.e making a new a variable bin histograms.
    - the new bin edges must corresponds to existing bin edges. It is not possible to split bins.
  - Histogram errors and statistics are re-computed according to the new binning.

```
TH1 * hrb = h1->Rebin(4,”hrebinned”);
```

Rebin original histogram (40 bins) in a new one with 10 bins grouping 4 bins together (`ngroup =4`)



Time for Exercises!

Put in practice the concepts to which you were just exposed: read the instructions here

https://twiki.cern.ch/twiki/bin/view/Main/RootIRMMTutorial2013HistogramsExercises

and solve exercises 1 and 2

2D Histograms

- Frequency distribution of (X,Y) observations.
  - Construct 2D histogram specifying the number of bins in X axis, the minimum and maximum of axis range and the same for the Y axis

```
TH2D * h2 = new TH2D(“h2”,”Example 2D Hist”,40,-4.,4.,40,-4.,4);
```

  - fill 2D histogram with 10000 x,y normal data

```
for (int i = 0; i<10000; ++i) {
  double x = gRandom->Gaus(0,1);
  double y = gRandom->Gaus(1,2);
  h2->Fill(x,y);
}
```

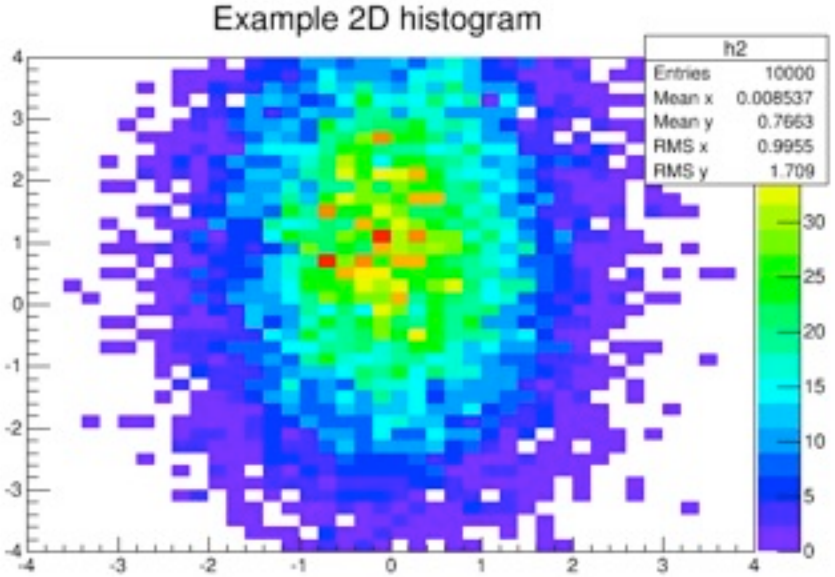
  - use `h2->Draw()` for drawing, several options are available
    - Color, Contour, lego, surface and box plots

Drawing 2D histograms

- Color plots:

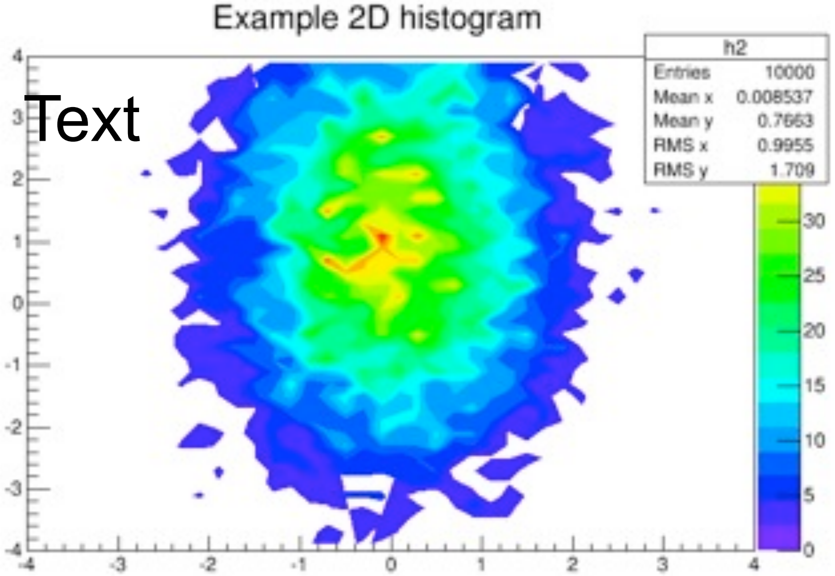
```
h2->Draw(“COLZ”);
```

“Z” means drawing the color palette for the bin content (i.e. the Z axis)



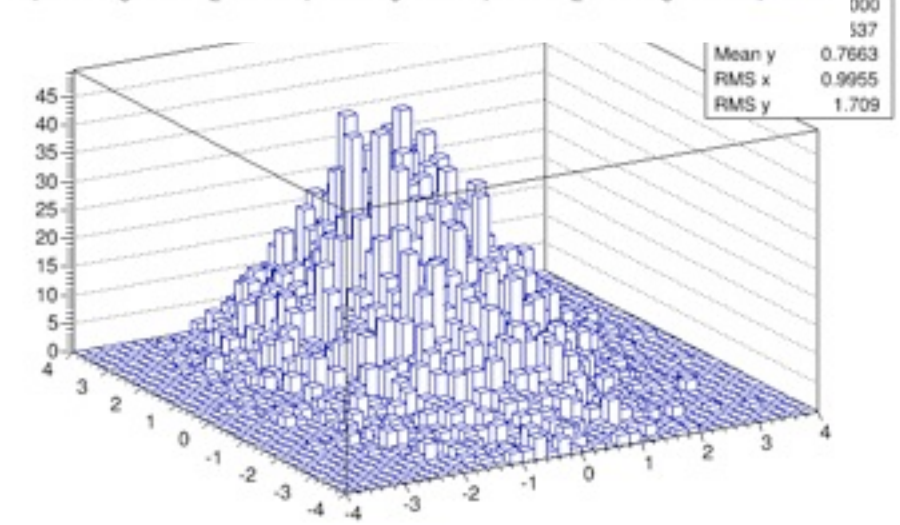
- Contour plots:

```
h2->Draw(“CONTZ”);
```



- Lego plots:

```
h2->Draw(“LEGO”);
```



Profile Histograms

- A 2D Histogram can be projected into a 1D histogram

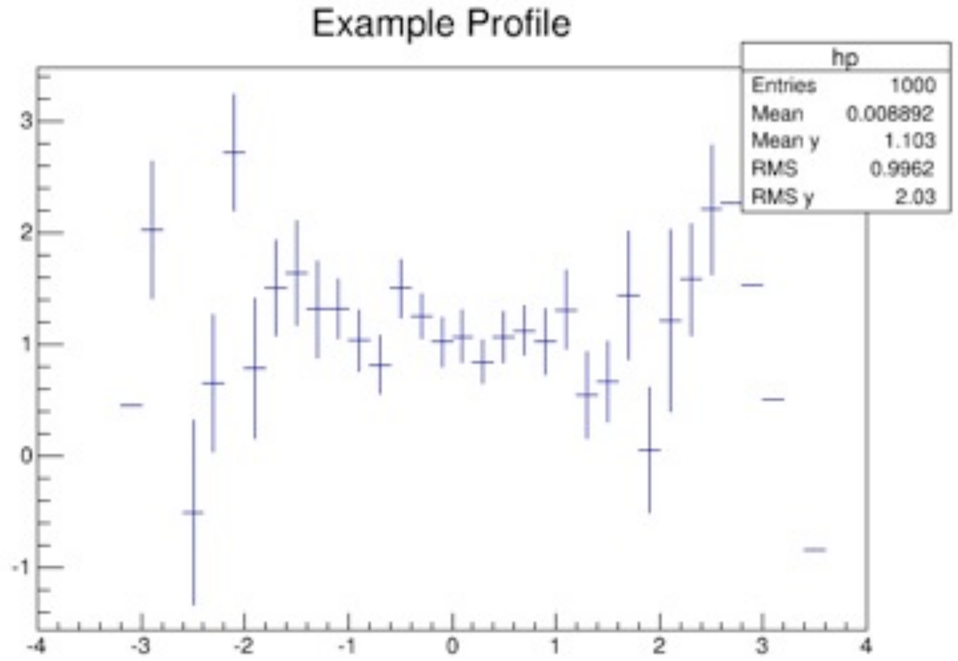
```
TH1 * hX = h2->ProjectionX();
```

  - when projecting on the X axis the bins with the same x value are all summed together.

- A Profile histogram is a different way of projecting 2D data.
  - for each bin in x the sample mean of the y values is plotted

```
TProfile * hp = new
TProfile(“hp”,”hp”,nbin,xmin,xmax);
for (int i = 0; i<10000; ++i) {
   double x = gRandom->Gaus(0,1);
   double y = gRandom->Gaus(1,2);
   hp->Fill(x,y);
}

hp->Draw();
```



  - various options for displaying errors
  - default uses error in the sample mean (RMS/√N)

Time for Exercises!

Put in practice the concepts to which you were just exposed: read the instructions here

https://twiki.cern.ch/twiki/bin/view/Main/RootIRMMTutorial2013HistogramsExercises

and solve exercise 3

Weighted Histograms

- Histogram can be filled with a “weight”
  - observations do not contribute equally, but some of them contribute more or less than others;
  - the weight expresses how much an observation contributes.
    - Filling a 1D histogram with observation x and weight w:

```
h1->Fill(x,w);
```

    - Filling a 2D histogram with observation x,y and weight w:

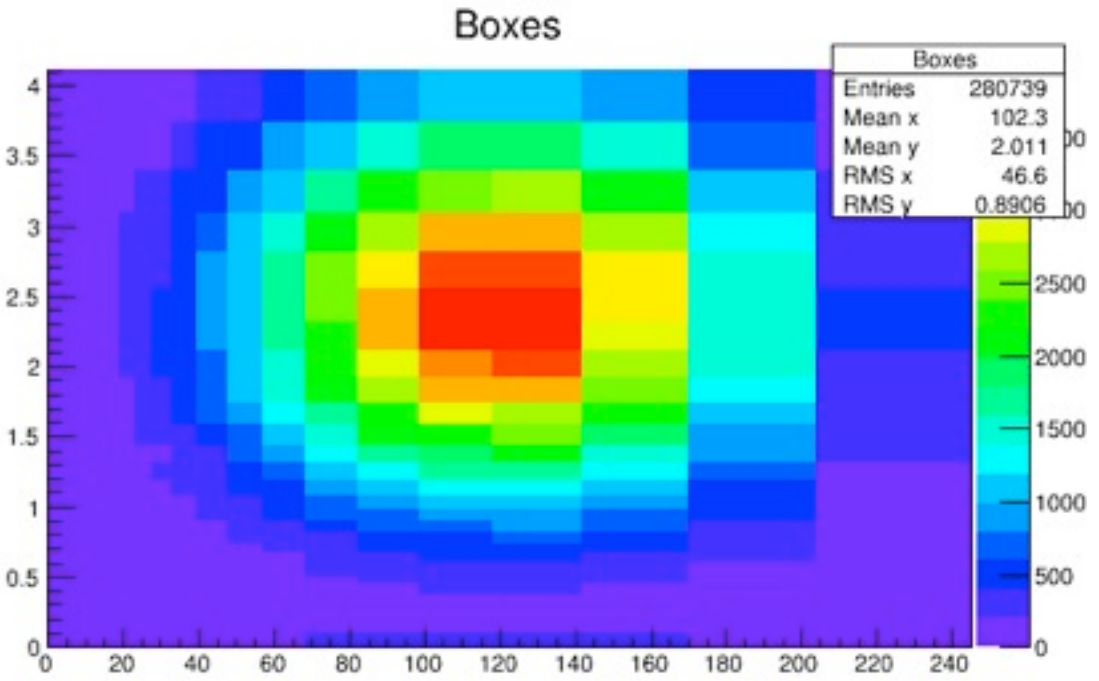
```
h1->Fill(x,y,w);
```

  - A weighted histogram will have and display as:
    - bin content = sum of all the weights accumulated in the bin;
    - bin error = $\sqrt{W2}$: W2 = sum of the weight square in the bin.

(in ROOT versions <= 5.34, if one has called TH1::Sumw2() before filling the histogram)

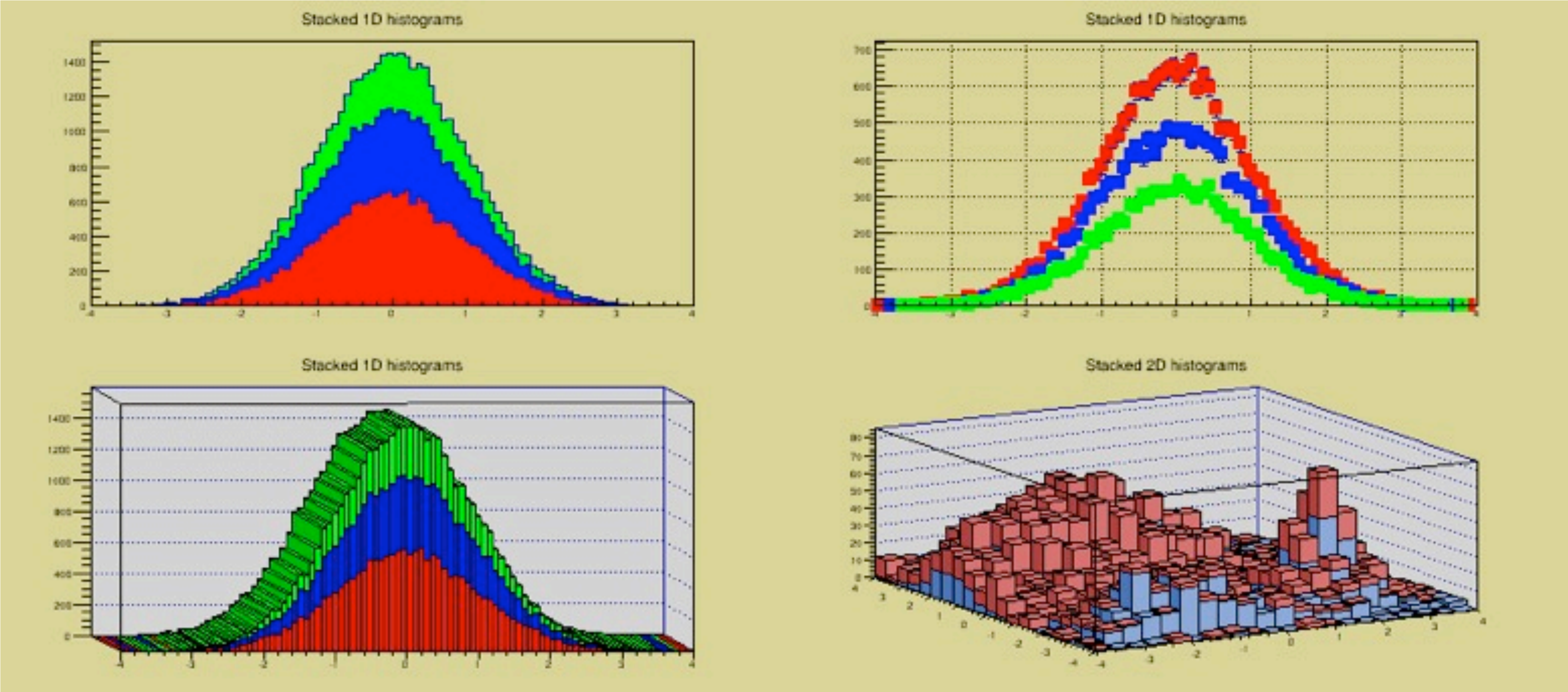
Other Histogram Types

- `THnSparse`
  - multi-dimensional sparse histogram useful to save memory only when a small fractions of the bins are filled
  - often more effective than using a TH3
- `THn`
  - non-sparse multi-dimensional histogram (NDim > 3)
- `TH2Poly`
  - 2D histogram with polygons shapes



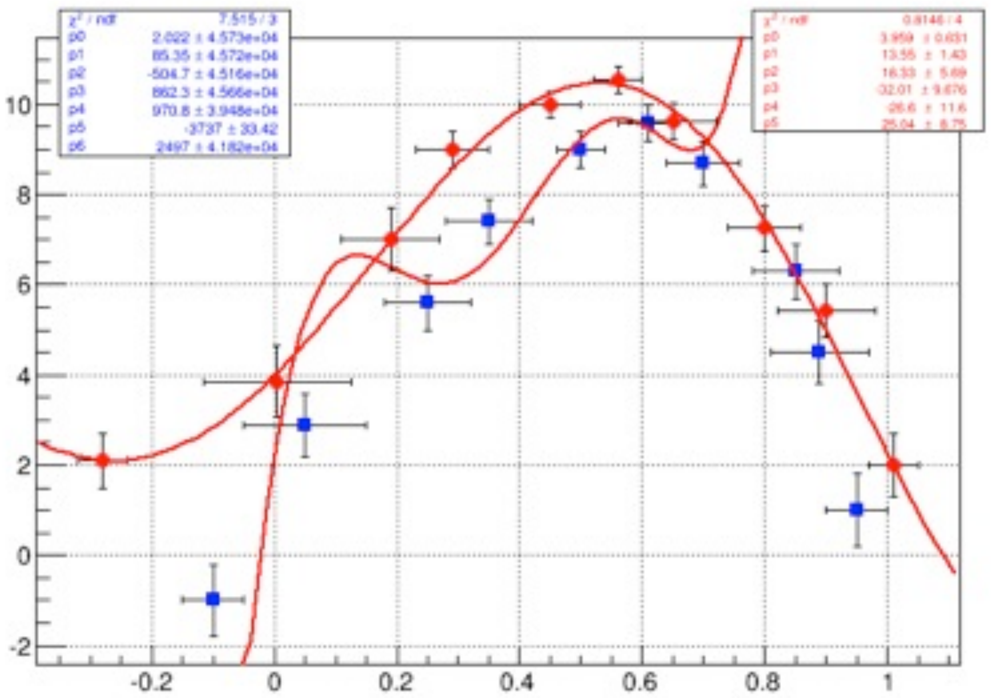
Collection of Histograms

- `THStack`
  - Collection of 1D or 2D histograms.
  - Allow to draw several histograms in one go, on top of each other.
  - The frame is computed automatically.
  - Often better than doing several plots with option “SAME”



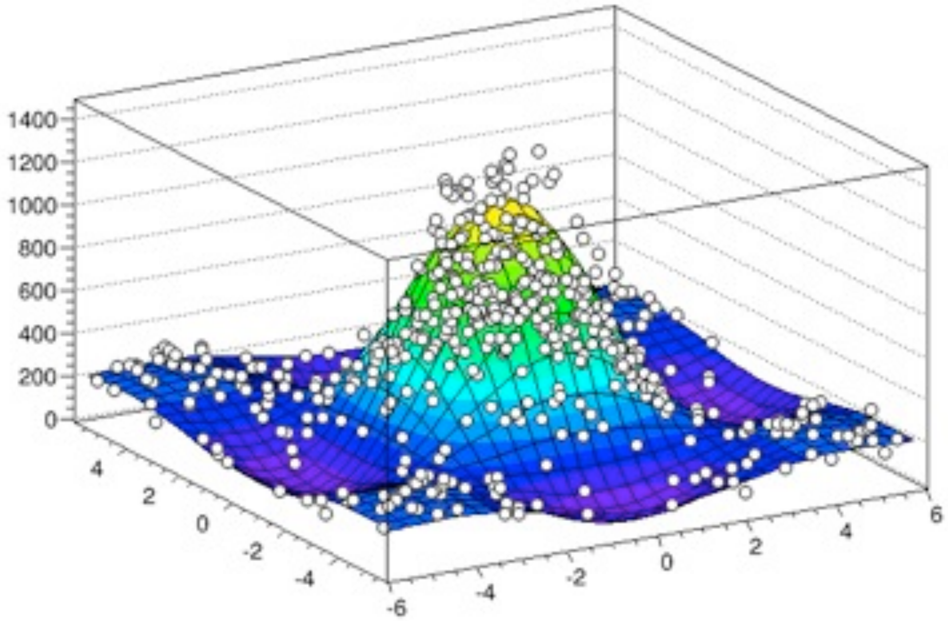
More on Graphs

- `TMultiGraph`
  - Graphs collection
  - Equivalent to THStack but for graphs
  - Allow to Draw several graphs in one go



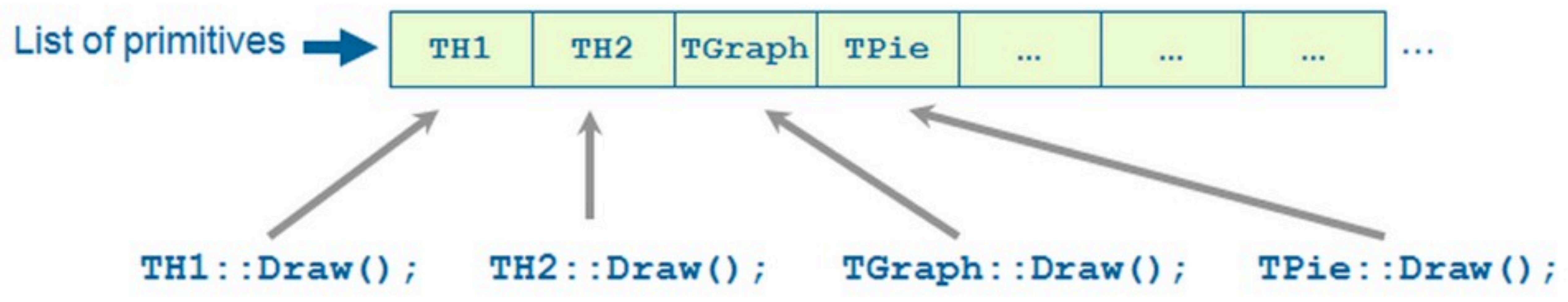
- 2-Dimensional Graphs
  - `TGraph2D`, `TGraph2DErrors`
    - classes for storing and drawing (X,Y,Z) data and (for `TGraph2DErrors`) with error in both X,Y and Z
      - X,Y are independent variables and Z the dependent variable

2D Function on the Graph2DErrors points



The Graphics Pad (1/2)

The ROOT graphics is build around the *Graphics Pad* concept (class `TPad`). A Graphics *Pad* is a linked list of primitives of any type (graphs, histograms, shapes, tracks, etc.). It is a kind of *display list* as shown on the following picture:



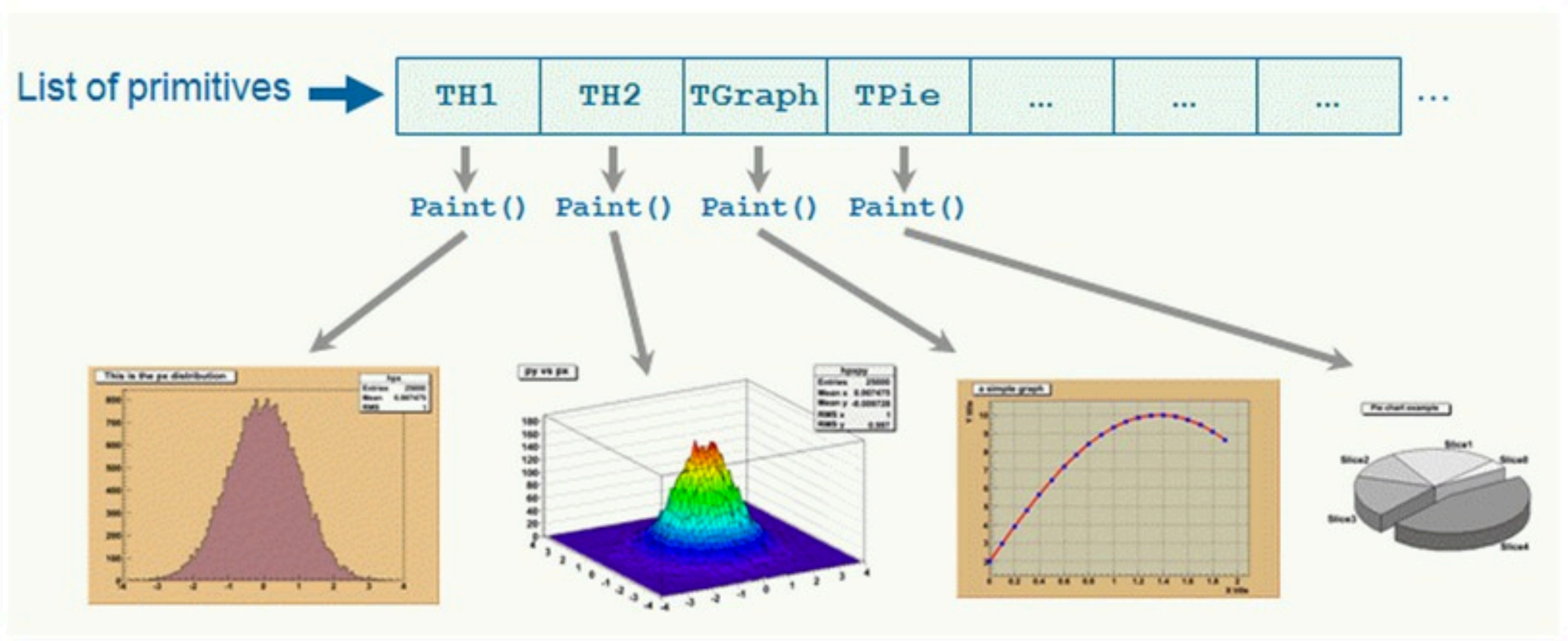
Adding an element into a Graphics Pad is done by the *Draw()* method of each classes.

On the previous picture the *Draw()* method has been called on: a 1D histogram, a 2D histogram, a graph and finally a pie chart.

All these objects are now stored in the Graphics Pad's display list.

The Graphics Pad (2/2)

A Graphics Pad is painted by calling sequentially the *Paint()* method of each object in the list of primitives, as shown on the following picture:



The Graphics Pad's (re)painting does not need to be done explicitly by the ROOT user. It is done automatically at the end of a macro execution or when a Graphics Pad has been modified.

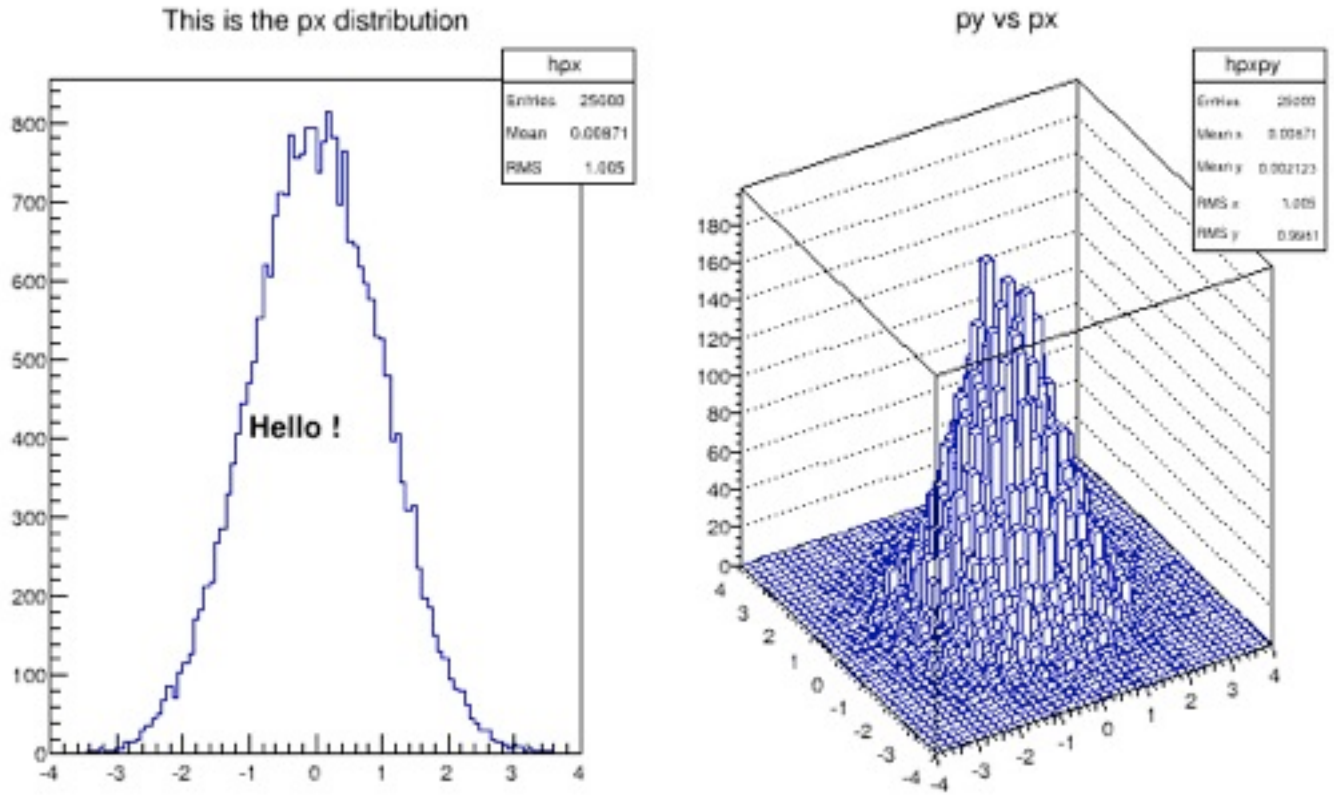
In some cases a pad need to be painted during a macro execution. To force the pad painting `gPad->Update()`should be performed.

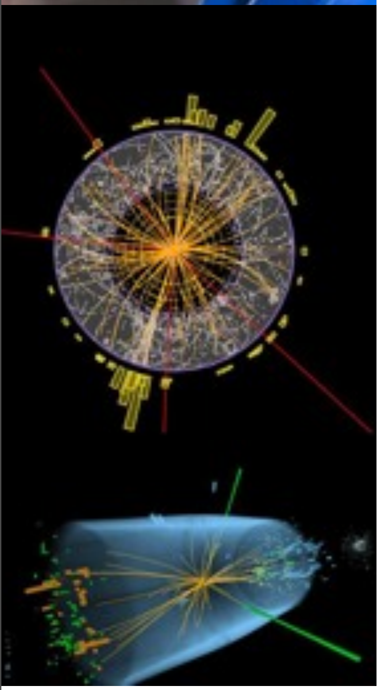
The Graphics Canvas

A canvas is the graphics window in which the ROOT graphics will be displayed. It can be created using the class `TCanvas`. One ca create as many canvases as needed during a ROOT session.

A `TCanvas` usually contains at least one `TPad`. Most of the time it contains several, each of them having its own coordinate system. A simple way to quickly make several pads in a canvas is to use the method `Divide()` like in the following example:

```
TCanvas *c = new TCanvas("c","my canvas",
600,400);
c->Divide(2,1);
c->cd(1);
hpx->Draw();
c->cd(2);
hpxpy->Draw("lego");
c->cd(1);
TText *T = new TText(-1.,400.,"Hello !");
T->Draw();
```





Visualization Techniques in ROOT

How ROOT visualize

2, 3, 4 and N data variables

Dr Lorenzo Moneta
CERN PH-SFT
CH-1211 Geneva 23
sftweb.cern.ch
root.cern.ch

Visualization Techniques

The ROOT framework provides many techniques to visualize multi-variable data sets from 2 until N variables.

- 2 variables visualization techniques are used to display Trees, Ntuples, 1D histograms, functions y=f(x), graphs .

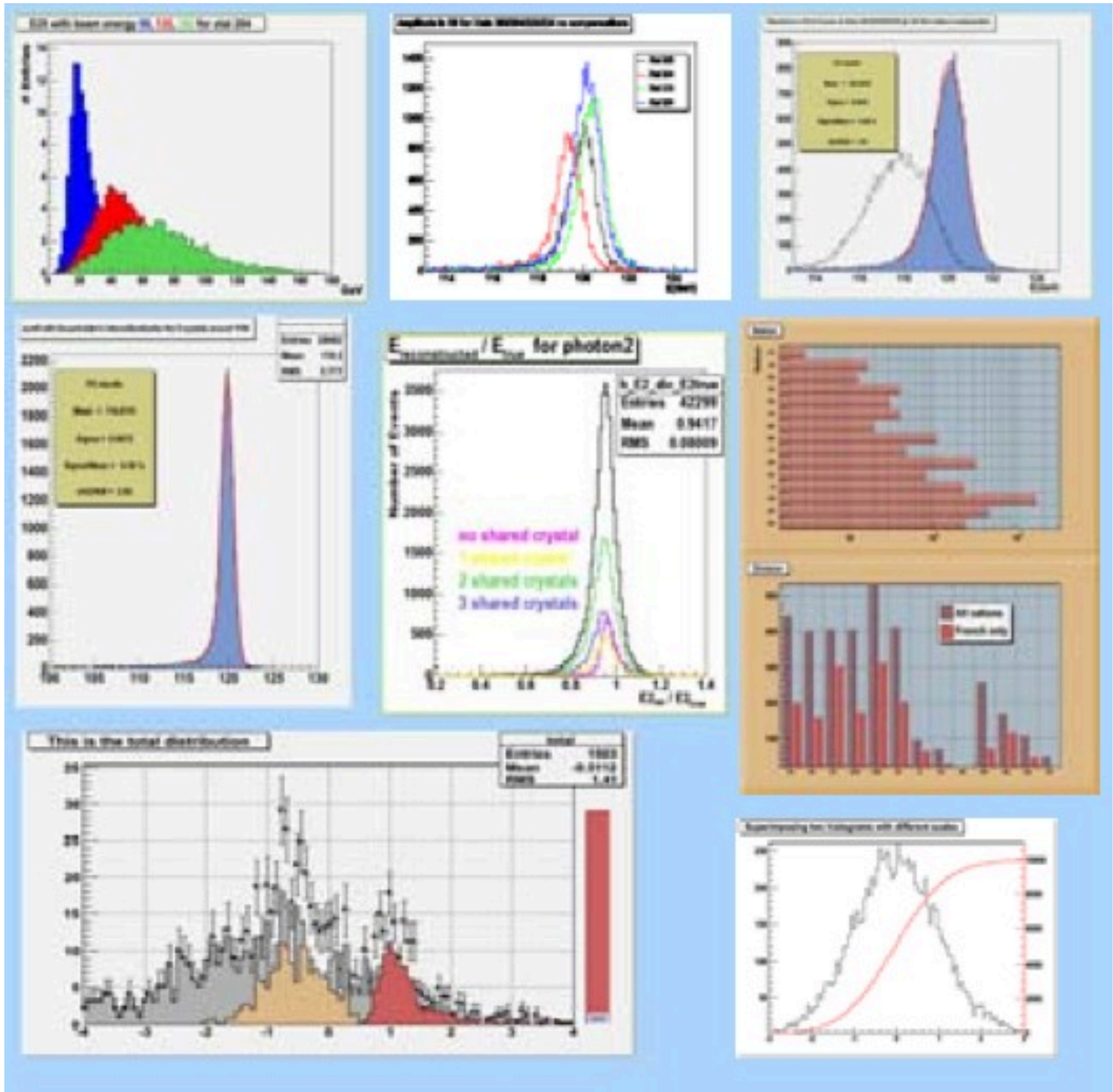
- 3 variables visualization techniques are used to display Trees, Ntuples, 2D histograms, 2D Graphs, 2D functions ...

- 4 variables visualization techniques are used to display Trees, Ntuples, 3D histograms, 3D functions ...

- N variables visualization techniques are used to display Trees and Ntuples ...

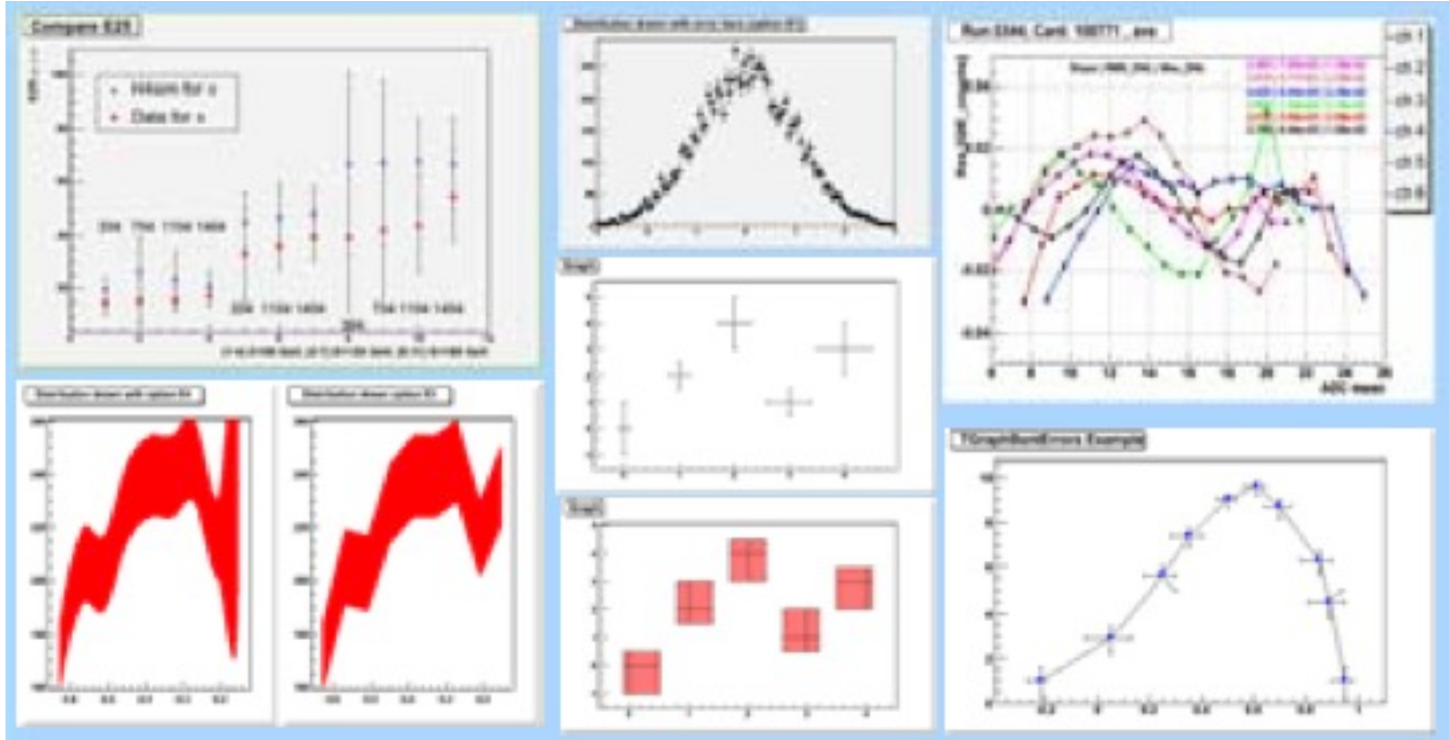
The next slides present them all, highlighting the best use one can do of each of them.

2 Variables Techniques (1/3)



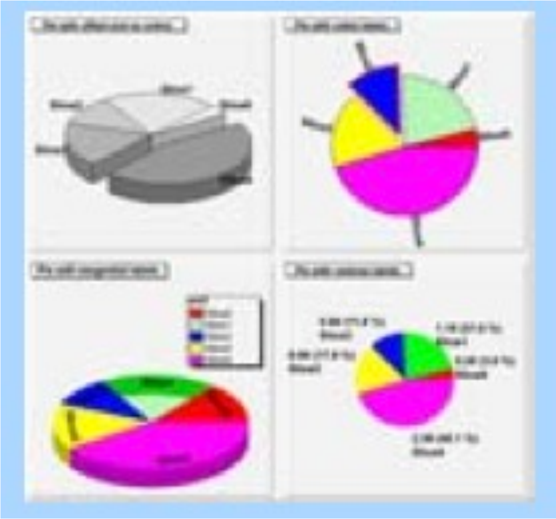
Bar charts and lines are a common way to represent 1D histograms.

2 Variables Techniques (2/3)

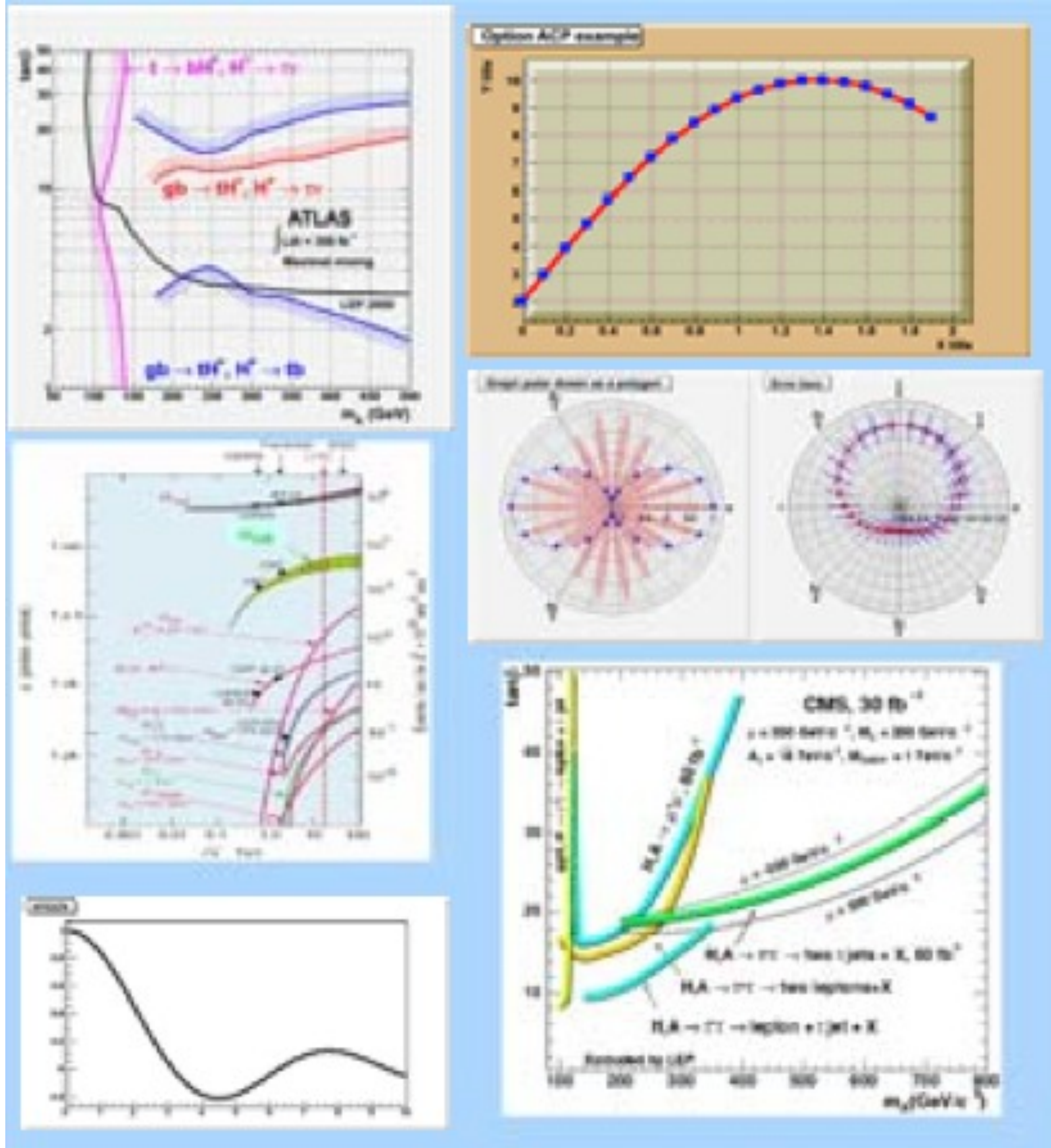


Errors can be represented as bars, band, rectangles. They can be symmetric, asymmetric or bent. 1D histograms and graphs can be drawn that way

Pie charts can be used to visualize 1D histograms. They also can be created from a simple mono dimensional vector.



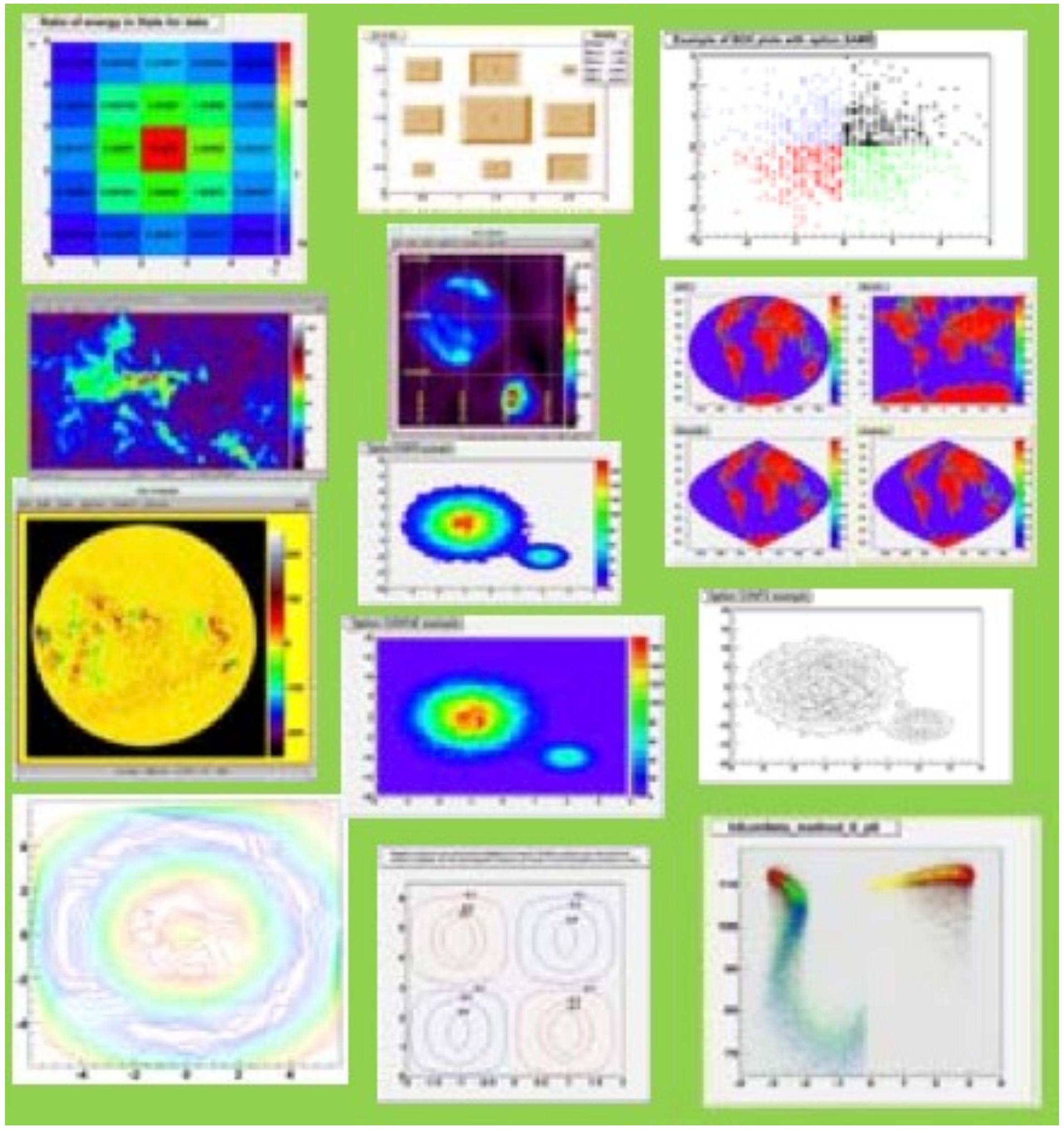
2 Variables Techniques (3/3)



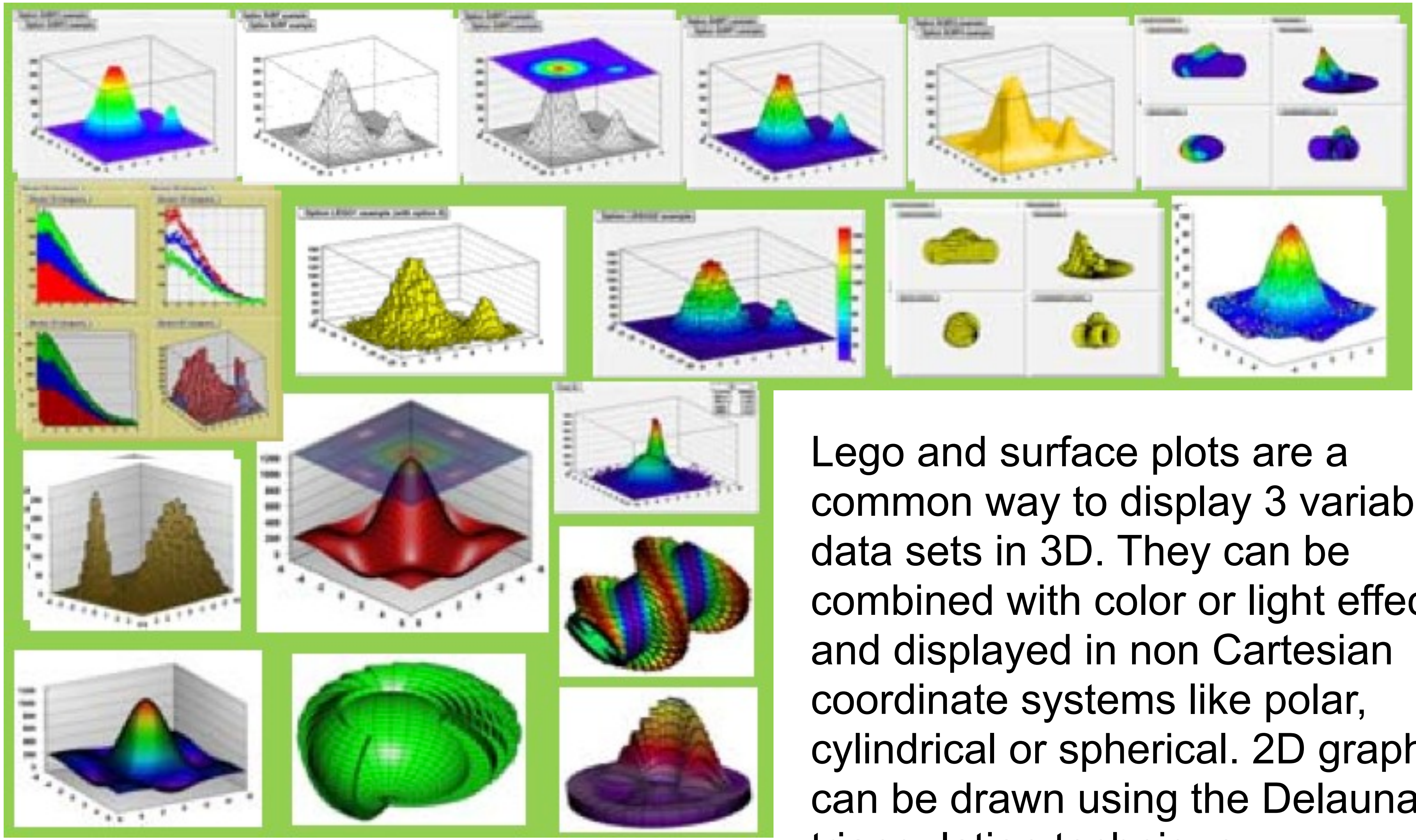
Graphs can be drawn as simple lines, like functions. They can also visualize exclusion zones or be plotted in polar coordinates.

3 Variables Techniques (1/2)

Several techniques are available to visualize 3 variables data sets in 2D. Two variables are mapped on the X and Y axis and the 3rd one on some graphical attributes like the color or the size of a box, a density of points (scatter plot) or simply by writing the value of the bin content. The 3rd variable can also be represented using contour plots. Some special projections (like Aitoff) are available to display such contours.



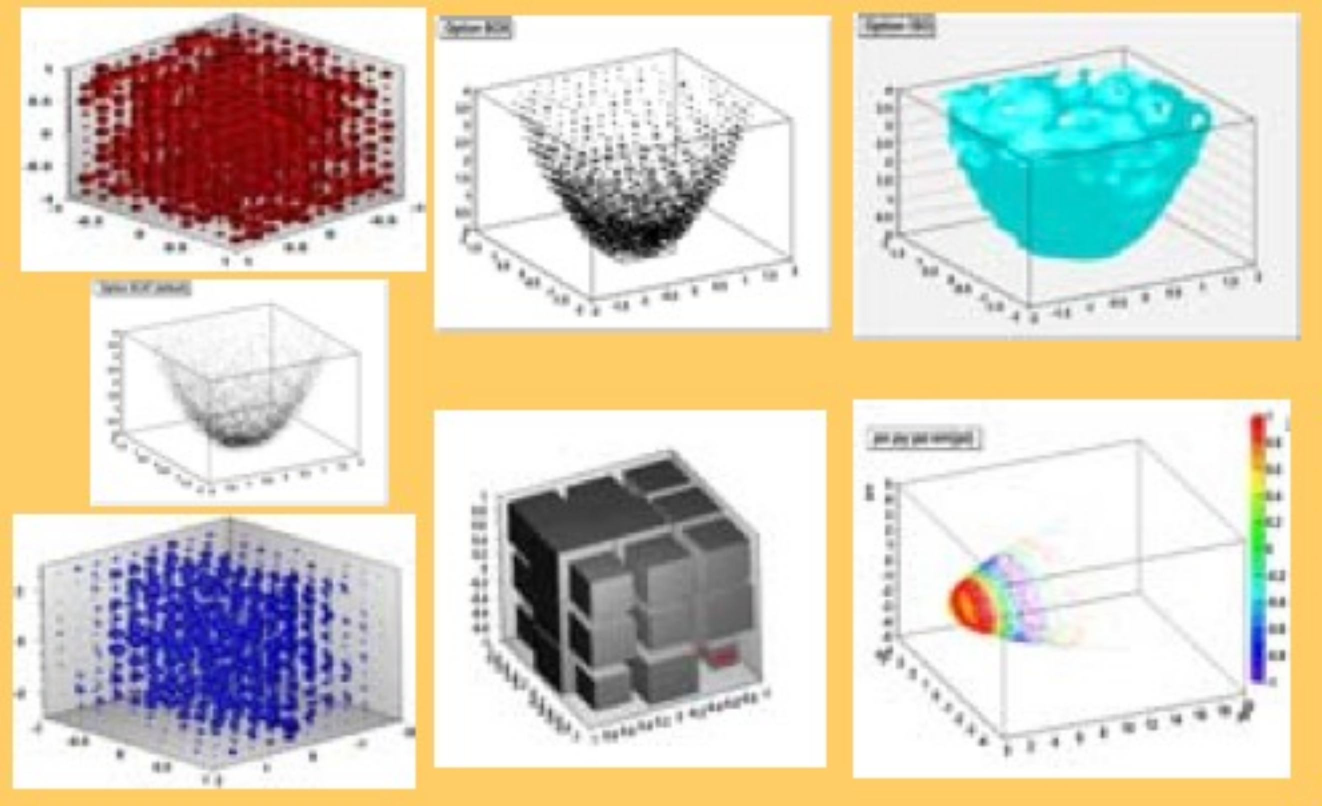
3 Variables Techniques (2/2)



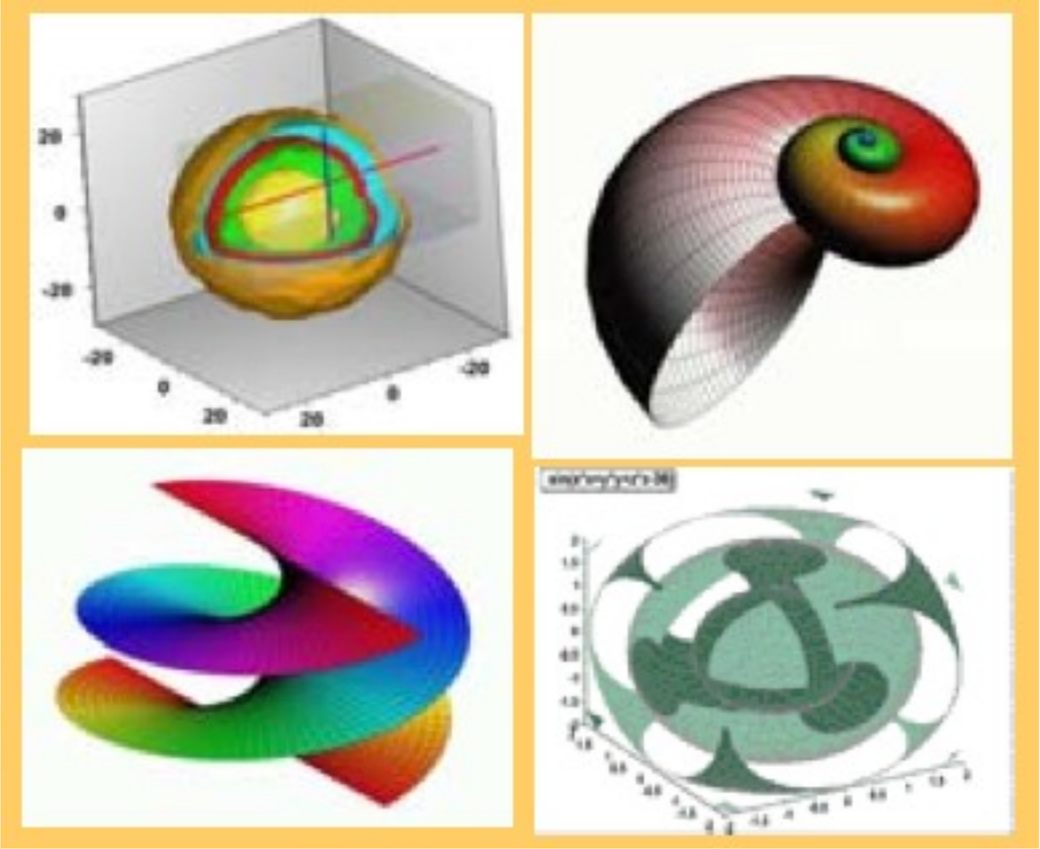
Lego and surface plots are a common way to display 3 variables data sets in 3D. They can be combined with color or light effects and displayed in non Cartesian coordinate systems like polar, cylindrical or spherical. 2D graphs can be drawn using the Delaunay triangulation technique.

4 Variables Techniques (1/2)

The 4 variables data set representations are extrapolations of the 3 variables ones. Rectangles become boxes or spheres, contour plots become iso-surfaces. The scatter plots (density plots) are drawn in boxes instead of rectangles. The 4th variable can also be mapped on colors. The use of OpenGL allows to enhance the plots' quality and the interactivity.

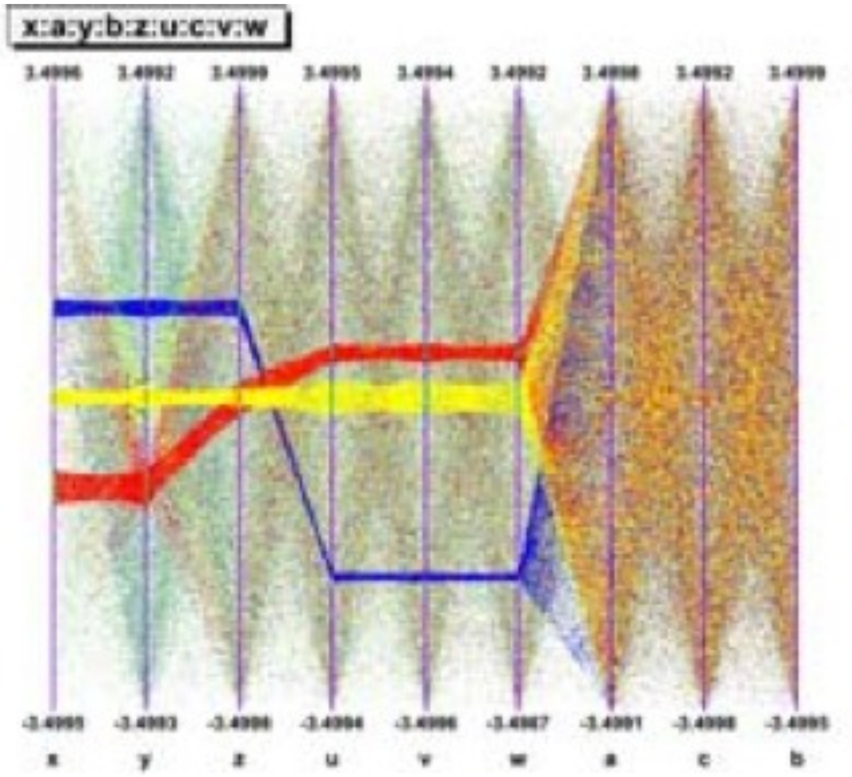


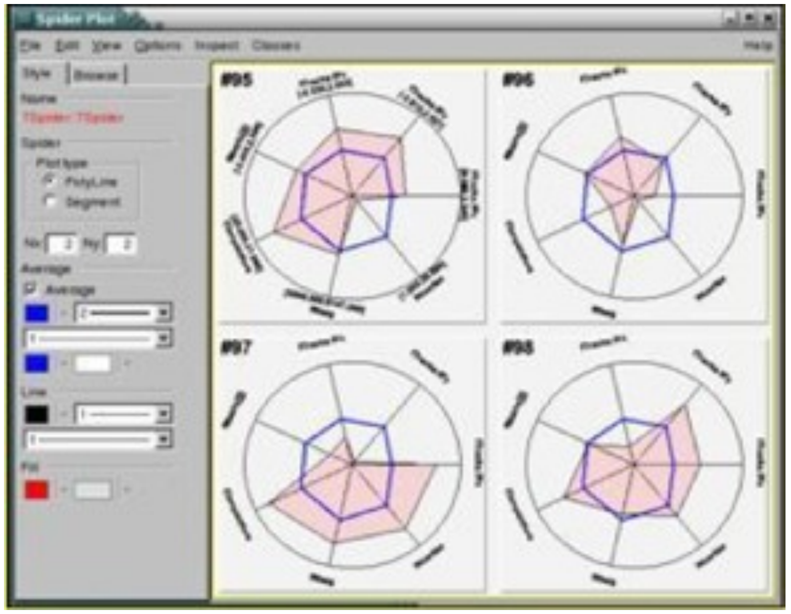
4 Variables Techniques (2/2)



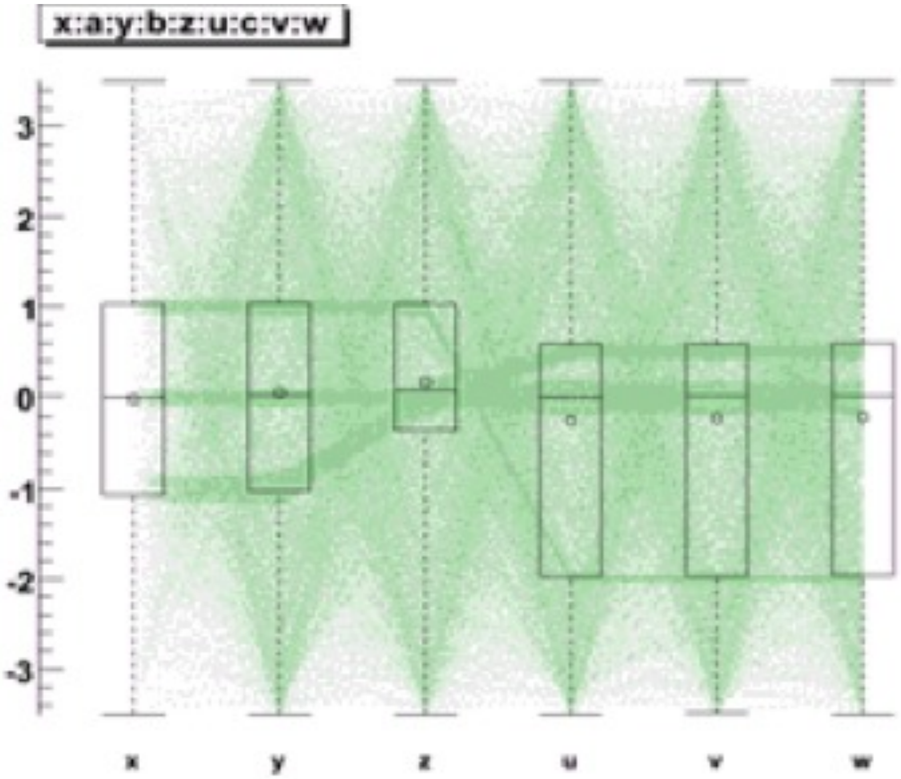
Functions like t= f(x,y,z) and 3D histograms are 4 variables objects. ROOT can render using OpenGL. It allows to enhance the plots' quality and the interactivity. Cutting planes, projection and zoom allow to better understand the data set or function.

N Variables Techniques





Above 4 variables more specific visualization techniques are required; ROOT provides three of them. The parallel coordinates (above) the candle plots (right) which can be combined with the parallel coordinates. And the spider plot (top right). These three techniques, and in particular the parallel coordinates, require a high level of interactivity to be fully efficient.



Summary

- We have learned more about ROOT Histograms
  - how to access their information
  - how to perform operations on histograms
- Multi-dimensional histograms and projections
  - what is a profile histogram
- Remember:
  - all graphics options for plotting all histogram types are documented in the `THistPainter` class:
    - http://root.cern.ch/root/html/THistPainter.html

- Next we will look on
  - what is fitting
  - how to fit histograms (and graphs) in ROOT