第三章：实验统计分析工具ROOT

授课人：董燎原
中国科学院高能物理研究所

https://root.cern.ch/courses

Histograms 直方图

Outline

- 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

What is Histogram

- In statistics, a graphical representation, showing a visual impression of the distribution of data.

- An estimate of the probability distribution of a variable

- Each rectangle erected over discrete intervals (bins)

- Area of a rectangle equals to the frequency of the observations in the interval (bin).

- Height of a rectangle also equals to the frequency density of the interval, i.e., the frequency divided by the width of the interval.

- The total area of the histogram is equal to the number of data.

- A histogram may also be normalized displaying relative frequencies, with the total area equaling 1.

Example:

Table of Ages (binned)

Age	Number
20-22	1
22-24	5
24-26	14
26-28	14
28-30	10
30-32	5
32-34	5
34-36	1
36-38	1
38-40	1

Entries 57
Mean 27.85
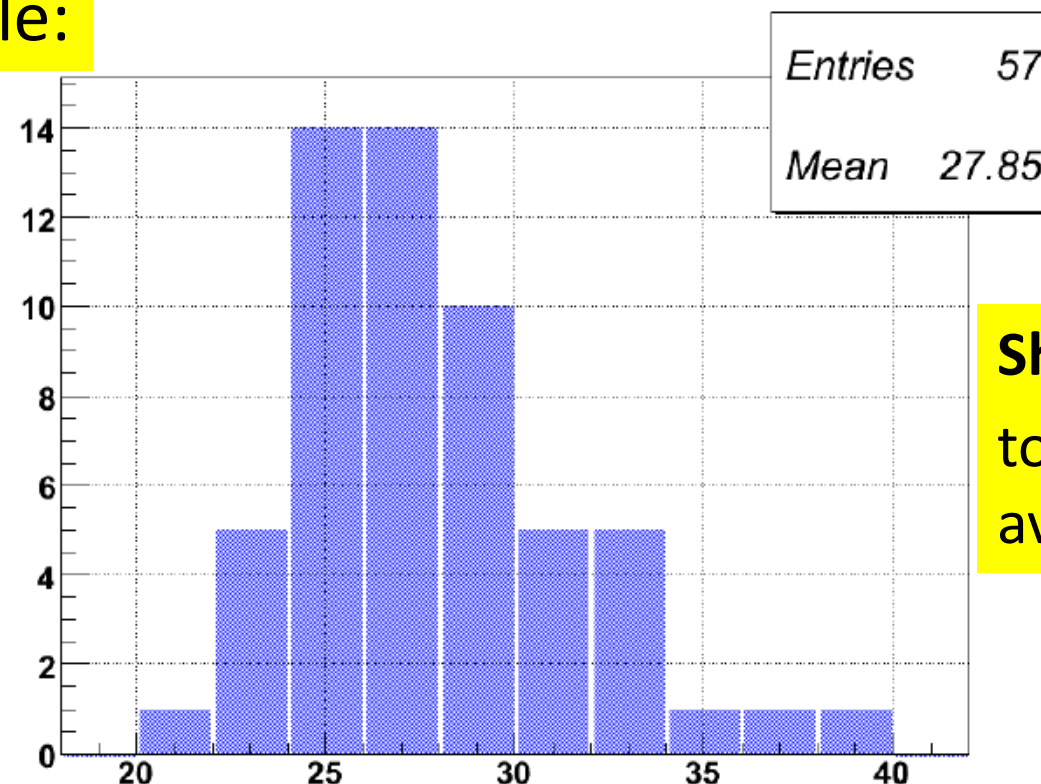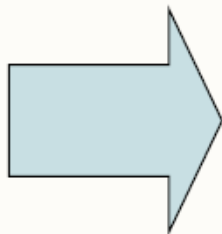
Shows distribution of ages:

total number: 57 participants;

average: 27 years 10 months 6 days

4

- Presenting data to determine which causes dominate

- Understanding the distribution of occurrences of different problems, causes, consequences, etc.

  - What is the most common system response?

  - What distribution (center, variation and shape) does the data have?

  - Does the data look symmetric or is it skewed to the left or right?
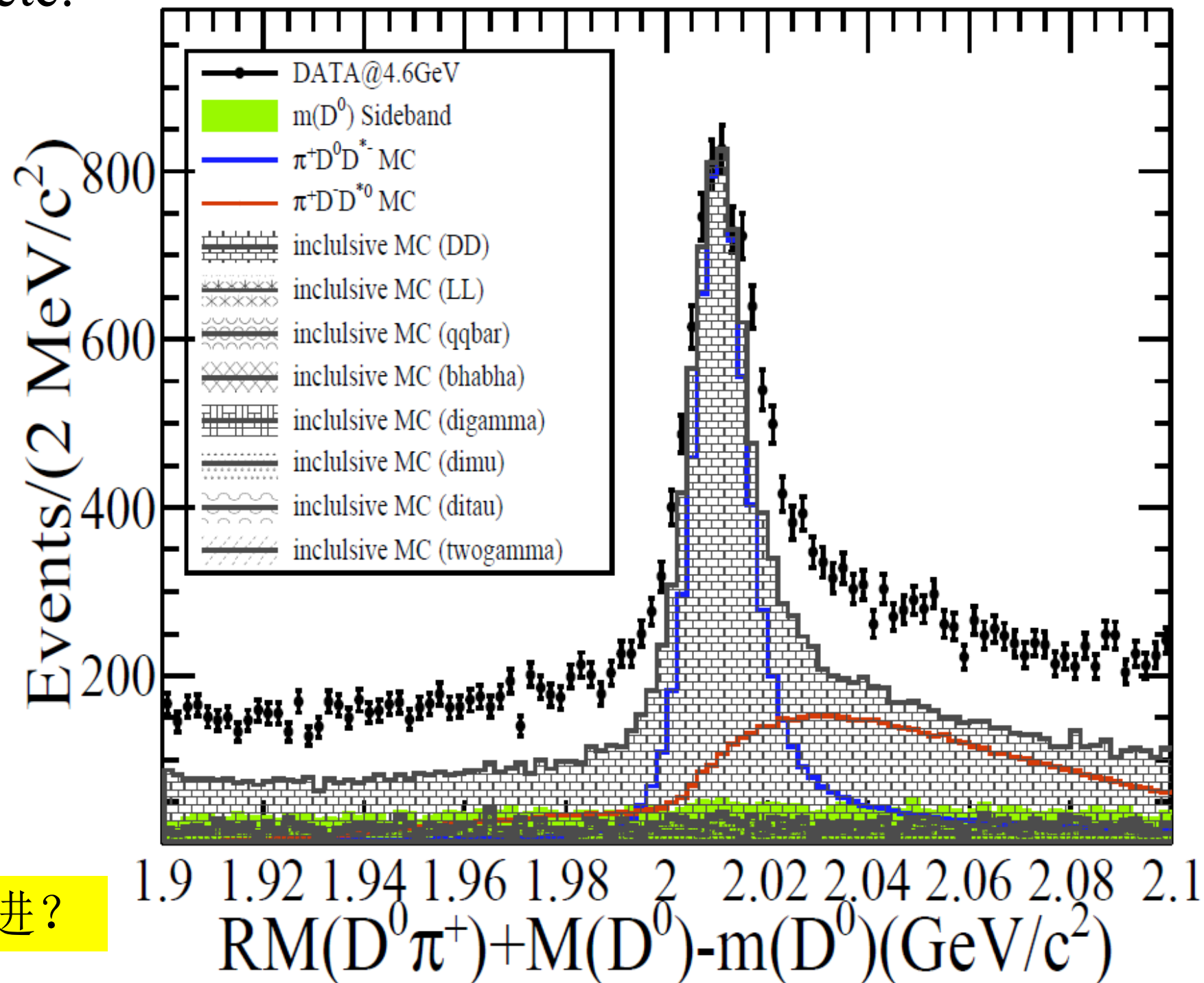
右图有什么地方需要改进？

Histogram Classes

ROOT supports the following histogram types:
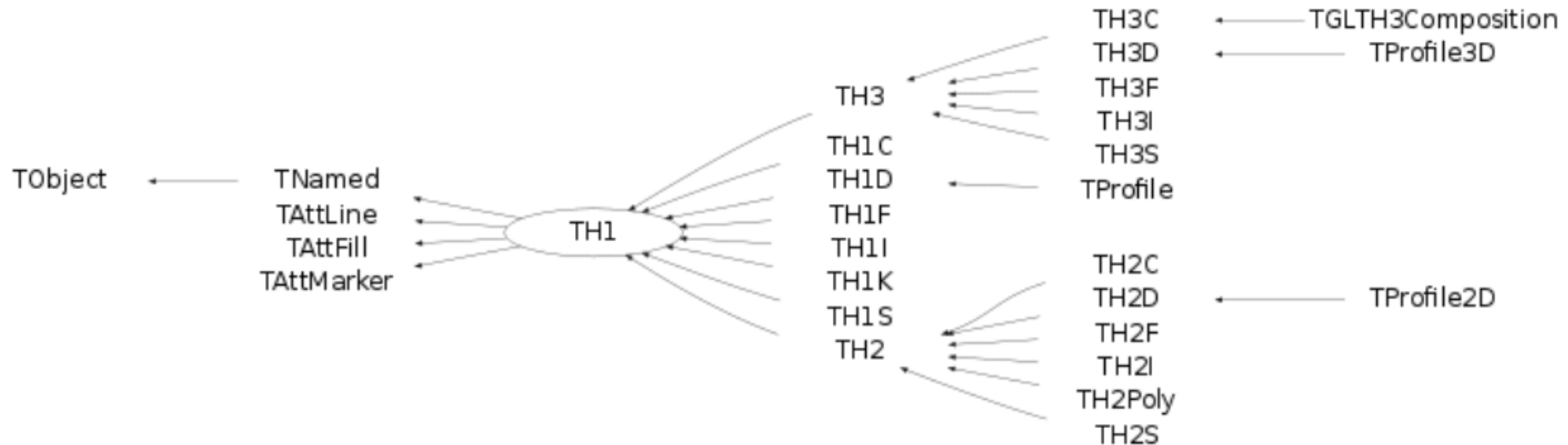
1-D histograms:
   TH1C: one byte (255) per channel.
   TH1S: one short (65535) per channel.
   TH1I: one integer (2147483647) per channel.
   TH1F: one float (7 digits) per channel.
   TH1D: one double (14 digits) per channel.

2-D histograms: …
3-D histograms: …
Profile histograms
   TProfile: one dimensional profiles
   TProfile2D: two dimensional profiles
   …

- 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.

Creating Histograms

TH1 constructor: the name of the histogram (name), the title (title), the number of bins (nbinsx), the x minimum (xlow), x maximum (xup) and array of low-edges for each bin (xbins).

```cpp
TH1F();
TH1F(const TVectorF& v);
TH1F(const TH1F& h1f);
TH1F(const char* name, const char* title, Int_t nbinsx,
const Float_t* xbins);
TH1F(const char* name, const char* title, Int_t nbinsx,
const Double_t* xbins);
TH1F(const char* name, const char* title, Int_t nbinsx,
Double_t xlow, Double_t xup);
virtual ~TH1F();
```

For equidistant bins:

```cpp
TH1F * h1 = new TH1F("h1","h1 title",100,0,4.4);
TH2F * h2 = new TH2F("h2","h2 title",40,0,4,30,-3,3);
```

For non-equidistant bins:

```cpp
TH1D * h1 = new TH1D("h1","Example histogram",nbins,xbins);
```

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

Filling Histograms

- A histogram is typically filled with statements like:

```
h1->Fill(x);
h1->Fill(x,w); //with weight
h2->Fill(x,y);
h2->Fill(x,y,w);
h3->Fill(x,y,z);
h3->Fill(x,y,z,w);
```

```
root[] TH1F h1("h1","Histo from a Gaussian",100,-3,3);
root[] h1.FillRandom("gaus",5000);
root[] h1.Draw();
```
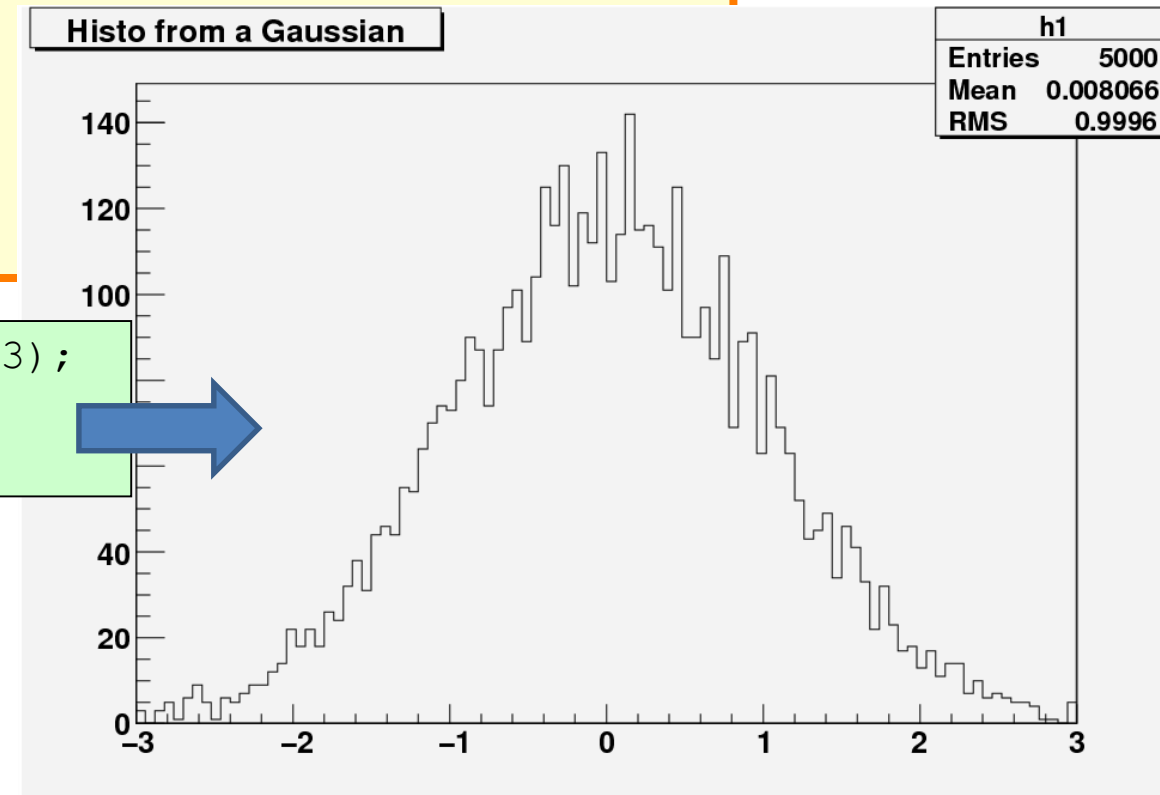

- Increment a bin number by calling
    TH1::AddBinContent()
replace the existing content via
    TH1::SetBinContent()
access the bin content of a given bin:
    TH1::GetBinContent()

```
Double_t binContent = h->GetBinContent(bin);
```

Drawing Histograms

- call the `Draw` method of a histogram (**TH1::Draw**) to draw the histogram

- it creates a THistPainter object and saves a pointer to painter as a data member of the histogram

- By default, the **TH1::Draw** clears the pad before drawing the new image of the histogram. You can use the "SAME" option to leave the previous display in tact and superimpose the new histogram.

- The same histogram can be drawn with different graphics options

- Most options can be concatenated without spaces or commas

- The options are not case sensitive

- You can also set the default drawing option with **TH1::SetOption**. To see the current option use **TH1**::GetOption.

Try examples in tutorials: *hist/draw2dopt.C*

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**

Giving Titles to the X, Y and Z Axis

Because the axis title is an attribute of the axis,

you have to get the axis first and then call **TAxis**::SetTitle.

```
h->GetXaxis()->SetTitle("X axis title");
h->GetYaxis()->SetTitle("Y axis title");
```

The histogram title and the axis titles can be any **TLatex** string.

```
h->GetXaxis()->SetTitle("E_{T}");
```

specify the histogram title and the axis titles at creation time given
in the "title" parameter. They must be separated by "**;**":

```
TH1F* h=new TH1F("h","Histogram title;X Axis;Y Axis;Z Axis",100,0,1);

TH1F* h=new TH1F("h","Histogram title;;Y Axis",100,0,1);

TH1F* h=new TH1F("h",";;Y Axis",100,0,1);

h->SetTitle("Histogram title;Another X title Axis");
```

Setting Axis Attributes

Use TH1::GetXaxis() to get the pointer to the TAxis object to set the Axis attributes

See Methods in
  Taxis class

```
TAxis*     TH1::GetXaxis() const
TAxis*     TH1::GetYaxis() const
TAxis*     TH1::GetZaxis() const
```

```
h->GetXaxis()->SetLabelColor(color)
              SetLabelFont(font)
              SetLabelOffset(offset)
              SetLabelSize(size)
              SetTickLength(length)
              SetTitleOffset(offset)
              SetTitleSize(size)
              SetTitleColor(color)
              SetTitleFont(font)
```

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 $\sqrt{N}$, 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 scaling (normalization) <mark>直方图的归一化</mark>
  - useful for plotting histograms in the same pad
  （归一化到同一积分可直观比较两种分布之间的差别）
  - useful for seeing histogram as an estimate of a probability density function (PDF) **（可直观估计几率密度函数）**

Method 1:
```
Double_t scale = norm/h->Integral();
h->Scale(scale); //归一化到norm（=1.0)
```
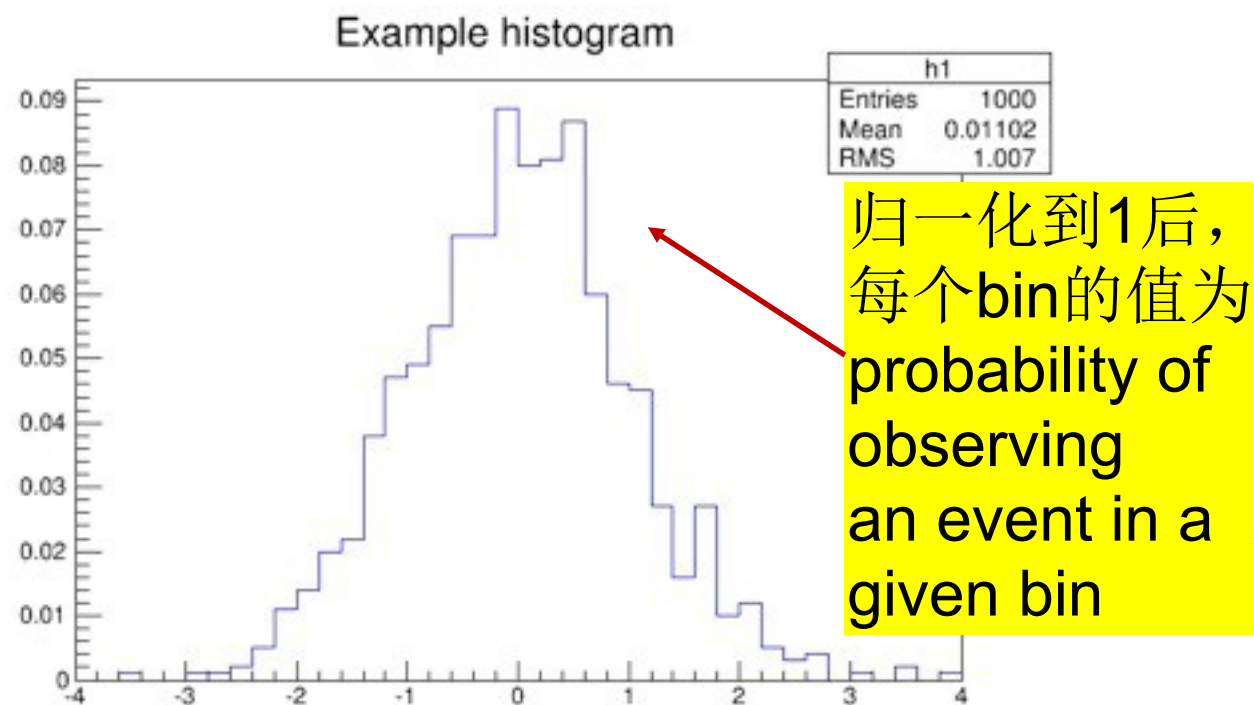
Method 2:
```
h->DrawNormalized("option_t", norm);
```


归一化到1后，每个bin的值为 probability of observing an event in a given bin
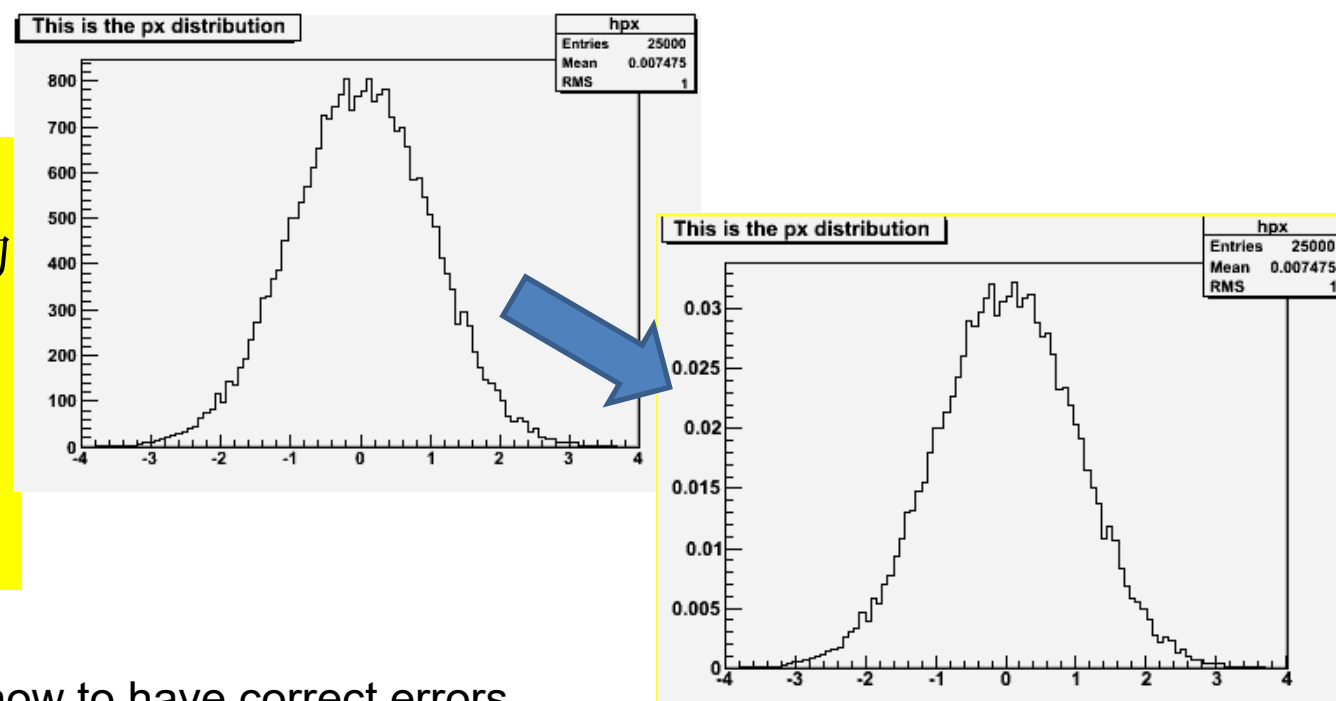
N.B. : After scaling the error will not be correct. We will see later how to have correct errors
归一化后，BinContent变了，BinError也变化了，需要确保误差运算正确无误。

- Add Histograms: 直方图的相加和相减
  - merge two histograms which have same axis:

```
TH1::SetDefaultSumw2();

TH1D *h3 = new TH1D("h3","h1+h2",nbin,xmin,xmax);

h3->Add(h1,h2,a,b);  //h3 = a*h1+b*h2  (一般a=b=1)
```

    相加：常用于实验数据的叠加。

  - can also be used to subtract histograms

```
h3->Add(h1,h2,a,-b);  //h3 = a*h1-b*h2  (一般a=b=1)
```

    相减：常用于实验中本底的扣除。

注意：
1. 两个直方图的区间大小和区间数必须相同，才能操作。
2. 在归一化和加减乘除中，要正确得到统计误差，需要调用： TH1::SetDefaultSumw2();
   或对每个直方图（如his）调用 his->Sumw2();

静态函数**SetDefaultSumw2**中的注释：
// static function. When this static function is called with sumw2=kTRUE,
// all new histograms will automatically activate the storage of the sum of squares of errors,
// ie TH1::Sumw2 is automatically called.

- Divide Histograms: <mark>直方图相除和相乘</mark>

```
TH1D *h3 = new TH1D("h3","h1/h2",nbin,xmin,xmax);
h3->Sumw2();
h3->Divide(h1,h2,a,b);
H3->Multiply(h1,h2,a,b);
```

相除：常用于实验中的效率的估计。
相乘：常用于实验中的对分布的修正（如效率修正等）。

– if h1 is a subset of h2, the bin content of h3 is binomially distributed ➠use option "B" to get the correct bin errors (h1和h2不独立：如h1包含于h2)

```
h3->Divide(h1,h2,a,b,"B");
```

二项分布误差：
$$\sigma = \sqrt{\dfrac{\dfrac{n_1}{n_2}(1 - \dfrac{n_1}{n_2})}{n_2}}$$

Miscellaneous Operations

TH1::`Smooth()` - smoothes the bin contents of a 1D histogram.

TH1::`Integral(Option_t *opt)`- returns the integral of bin contents in a given bin range. If the option "width" is specified, the integral is the sum of the bin contents multiplied by the bin width in x.

TH1::`GetMean(int axis)` - returns the mean value along axis.

TH1::`GetRMS(int axis)` - returns the Root Mean Square along axis.

TH1::`GetEntries()` - returns the number of entries.

TH1::`Add(const TH1* h1, Double_t c1 = 1)`

TH1::`Add(TF1* h1, Double_t c1 = 1, Option_t* option = "")`

TH1::`Add(const TH1* h1, const TH1* h2, Double_t c1 = 1, Double_t c2 = 1)` - add histograms weighted by `c1 and c2`.

Superimposing Histograms with Different Scales

```
TH1F *h1 = new TH1F("h1","my histogram",100,-3,3);
... // （填高斯分布）
h1->Draw();
TH1F *hint1 = new TH1F("hint1","my histogram",100,-3,3);
... // （填积分值）
//scale hint1 to the pad coordinates
Float_t rightmax = 1.1*hint1->GetMaximum();
Float_t scale = gPad->GetUymax()/rightmax;
hint1->SetLineColor(kRed);
hint1->Scale(scale);
hint1->Draw("same");
... // （draw an axis on the right side 看下面例子的代码在右边画一个轴）
```

example:
$ROOTSYS/tutorials/hist/twoscales.C
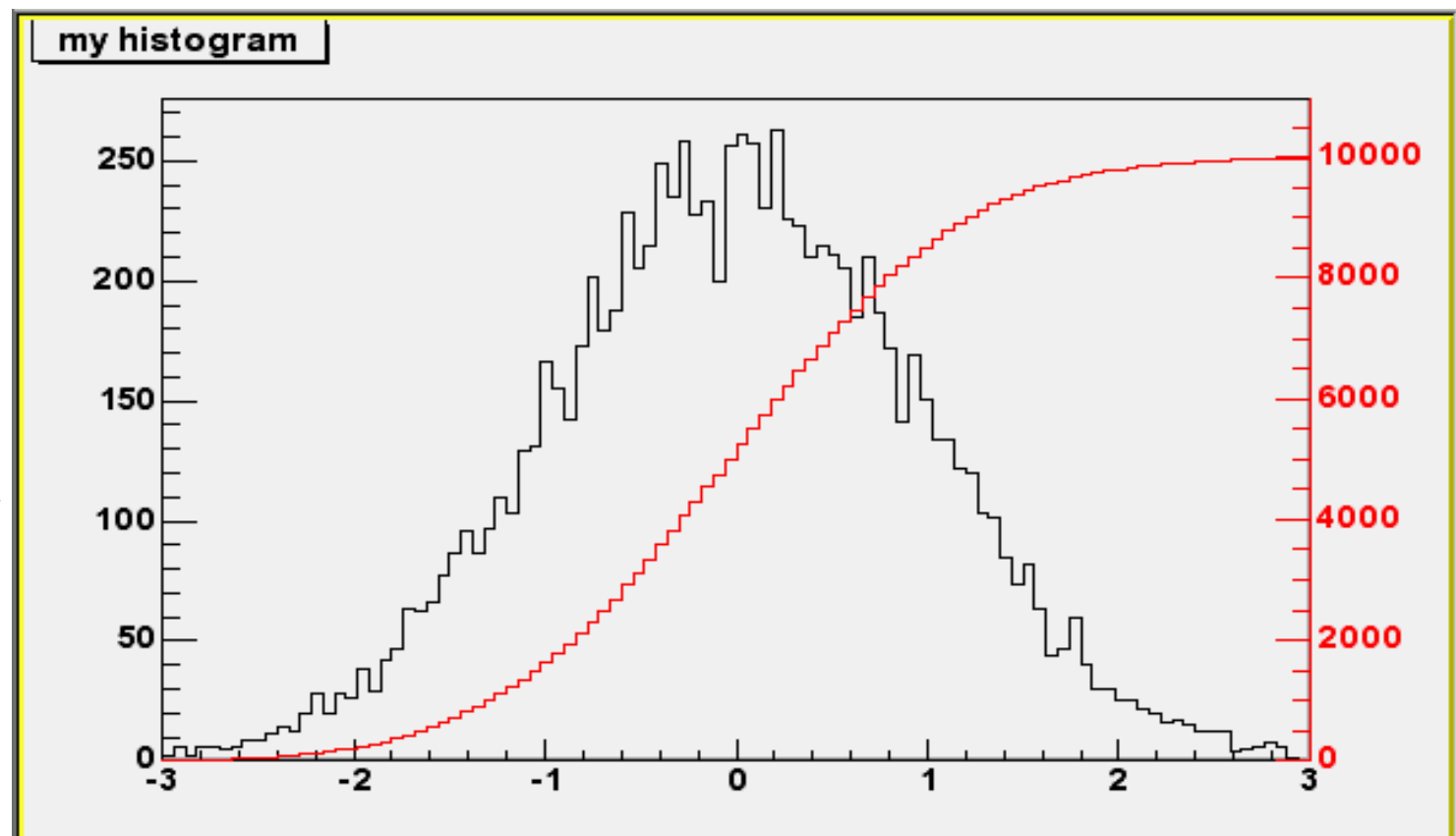
Histogram Stacks

- `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"
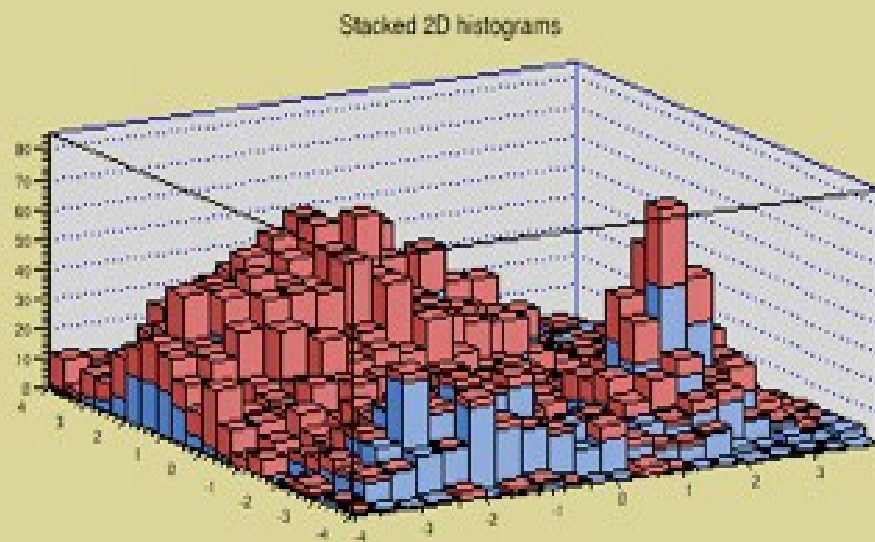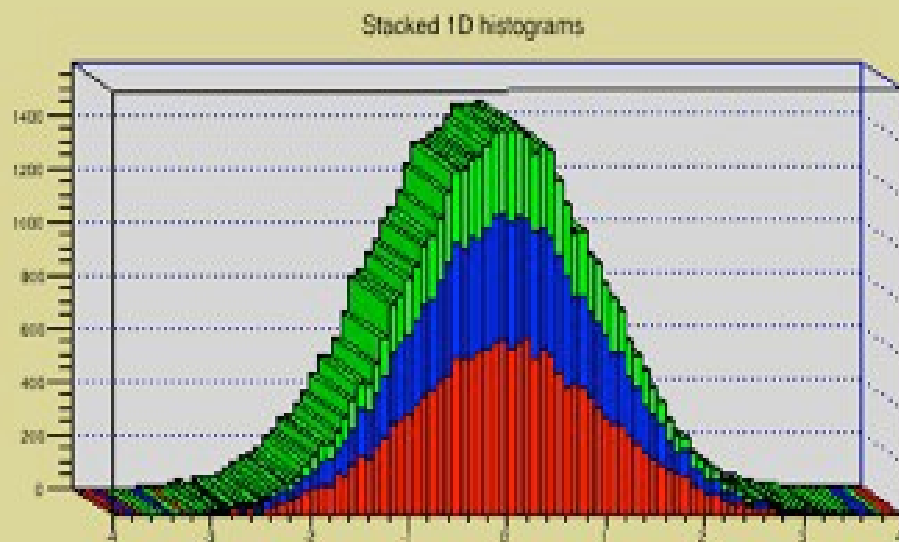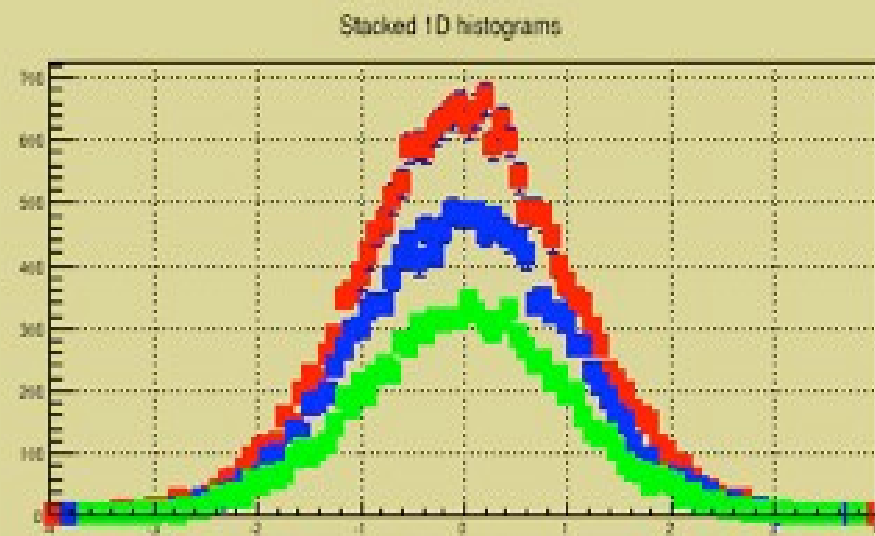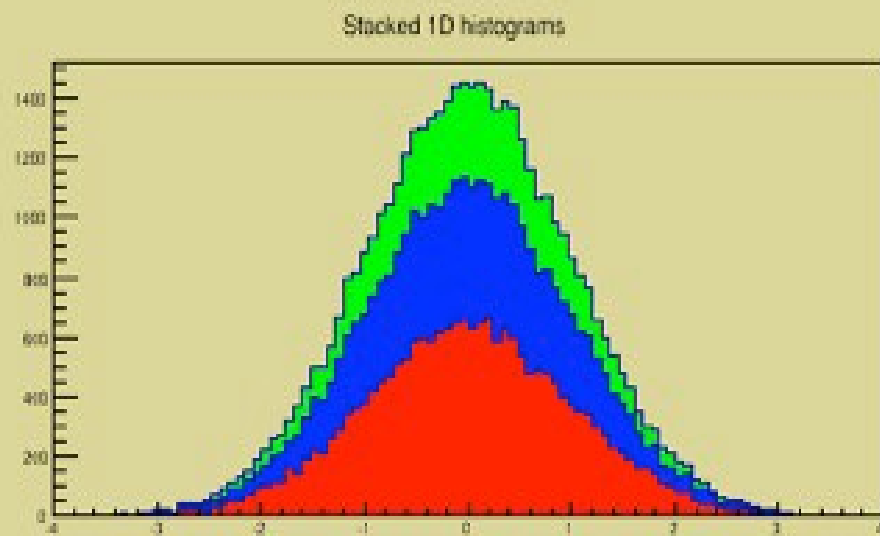
Histogram Stacks

A **THStack** is a collection of **TH1** (or derived) objects. Use **THStack**::Add(**TH1** *h) to add a histogram to the stack.

```
TCanvas c1("c1","stacked hists",10,10,700,900);
THStack hs("hs","test stacked histograms");
hs.Add(h1); \\red
hs.Add(h2); \\blue
hs.Add(h3); \\green
c1.Divide (1,2);
c1.cd(1);
hs.Draw();
c1.cd(2);
hs->Draw("nostack");
```
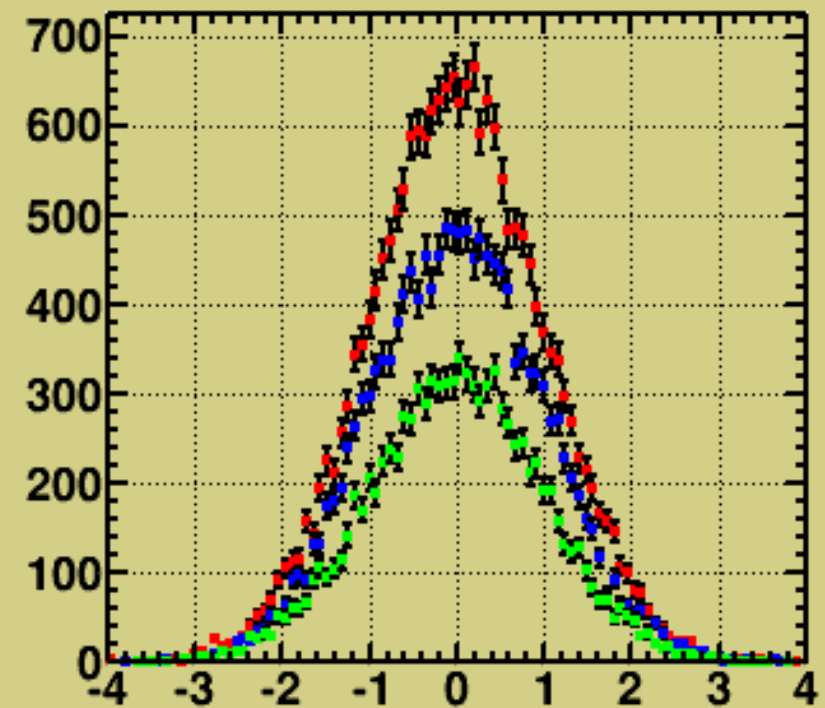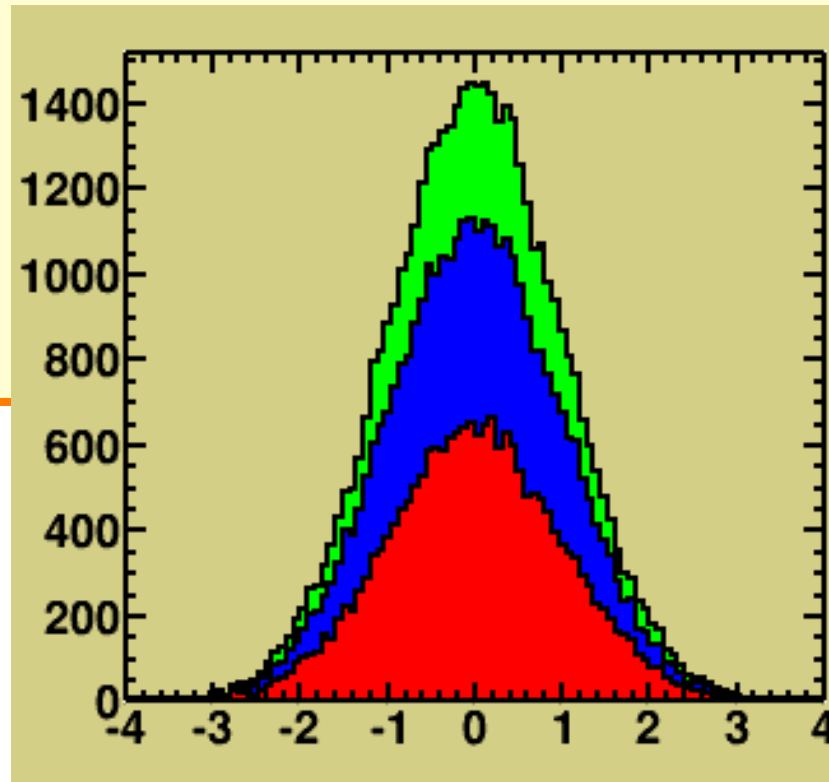
常用于实验数据中不同类型本底的叠加。

example: $ROOTSYS/tutorials/hist/hstack.C

Histogram Stacks：example

一个显示本底成分的例子：

```
Hist_data->Draw("E1");
Hist_others->SetFillColor(kBlue);
Hist_opencharm->SetFillColor(kGreen);
Hist_qqbar->SetFillColor(kRed);
Hist_DsSTDs_DsDs->SetFillColor(7);
THStack *hs = new THStack("hs","Stacked");
hs->Add(Hist_others);
hs->Add(Hist_qqbar);
hs->Add(Hist_opencharm);
hs->Add(Hist_DsSTDs_DsDs);
hs->Draw("same");
Hist_data->Draw("sameE1");
```

为何还要再画一次数据的图？
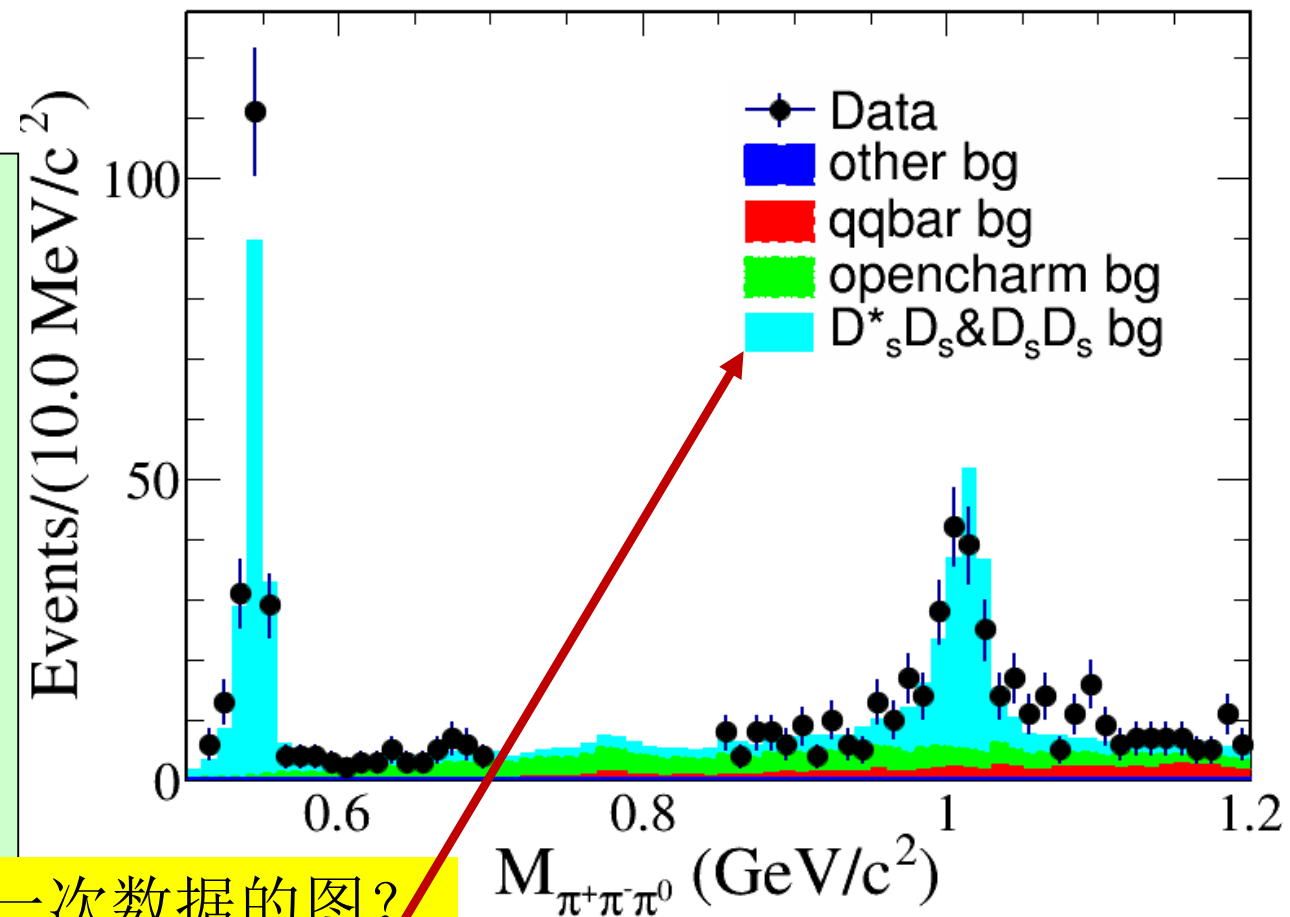
```
leg = new TLegend(0.55,0.60,0.85,0.9);
   leg->AddEntry(Hist_data,"Data","lep");
   leg->AddEntry(Hist_others,"other bg","f");
   leg->AddEntry(Hist_qqbar,"qqbar bg","f");
   leg->AddEntry(Hist_opencharm,"opencharm bg","f");
   leg->AddEntry(Hist_DsSTDs_DsDs,"D*_{s}D_{s}&D_{s}D_{s} bg","f");
   leg->SetTextSize(0.06);
   leg->SetLineColor(10);
   leg->SetFillColor(10);
   leg->Draw();
```

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`)
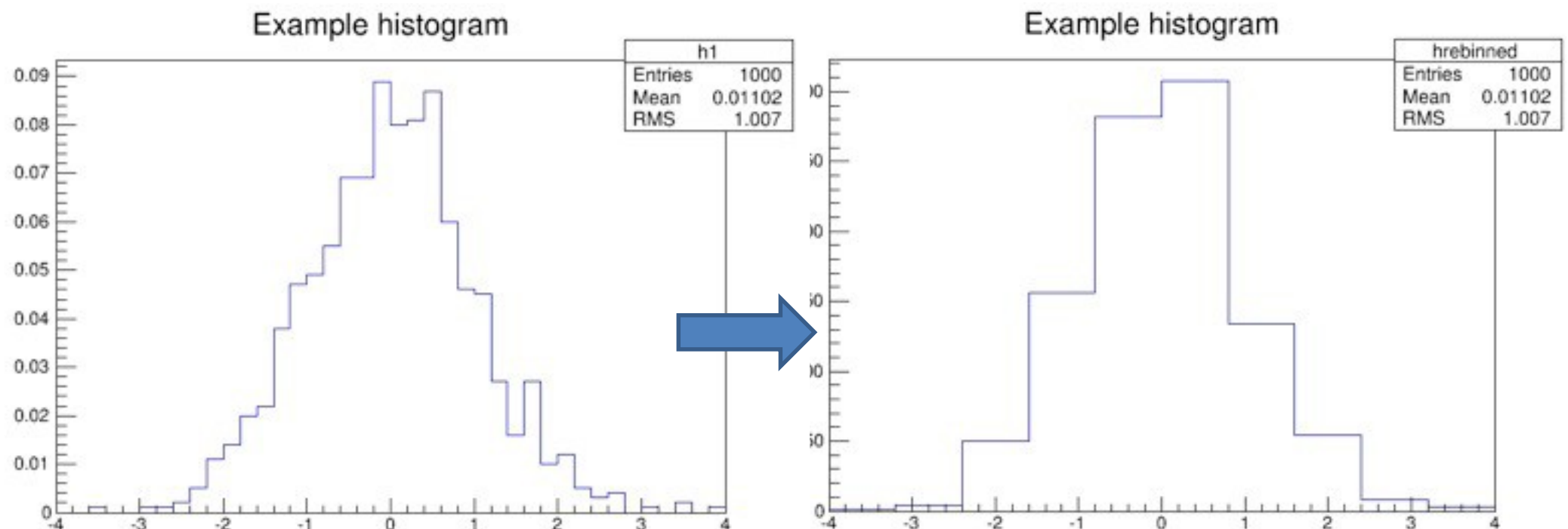
Statistics Display

By default, a histogram drawing includes the statistics box.
Use **TH1**::SetStats(kFALSE) to eliminate the statistics box.
gStyle->SetOptStat(mode) allow you to set displayed information.

The parameter mode has up to nine digits that can be set OFF (0) or ON as follows:
mode = ksiourmen (default = 000001111 or nemr)

```
n= 1  the name of histogram ---------------------n
e= 1  the number of entries ---------------------e
m= 1  the mean value ----------------------------m
m= 2  the mean and mean error values-------------M
r= 1  the root mean square (RMS)-----------------r
r= 2  the RMS and RMS error----------------------R
u= 1  the number of underflows ------------------u
o= 1  the number of overflows -------------------o
i= 1  the integral of bins ----------------------i
s= 1  the skewness ------------------------------s
s= 2  the skewness and the skewness error--------S
k= 1  the kurtosis------------------------------k
k= 2  the kurtosis and the kurtosis error--------K
```

Statistical Issues in Histogram

- ## Mean
- ## Minimum / Maximum
- ## Standard Deviation (RMS)
- ## Bin width / Bin number

This is the px distribution	hpx	
	Entries	25000
	Mean	0.007475 ± 0.006325
	RMS	1 ± 0.004472
	Underflow	3
	Overflow	1
	Integral	2.5e+04
	Skewness	-0.03791 ± 0.01549
	Kurtosis	-0.0248 ± 0.03099
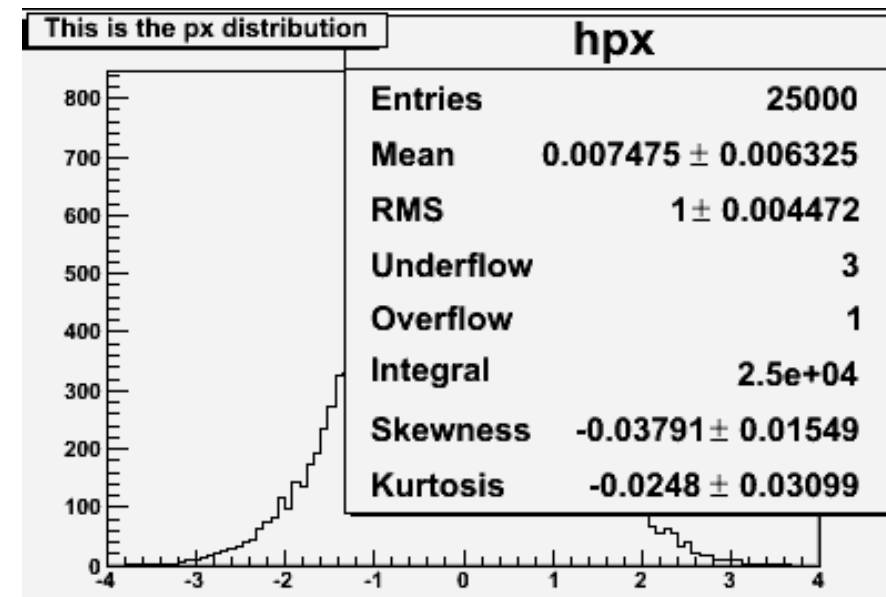
- 偏度 $$skewness = \frac{\sum_{i=1}^{N}(Y_i - \bar{Y})^3}{(N-1)s^3}$$

- 峰度： $$kurtosis = \frac{\sum_{i=1}^{N}(Y_i - \bar{Y})^4}{(N-1)s^4} - 3$$

正态分布

Negative Skew
(large tail to the left)

Positive Skew
(large tail to the right)

positive and negative
kurtosis
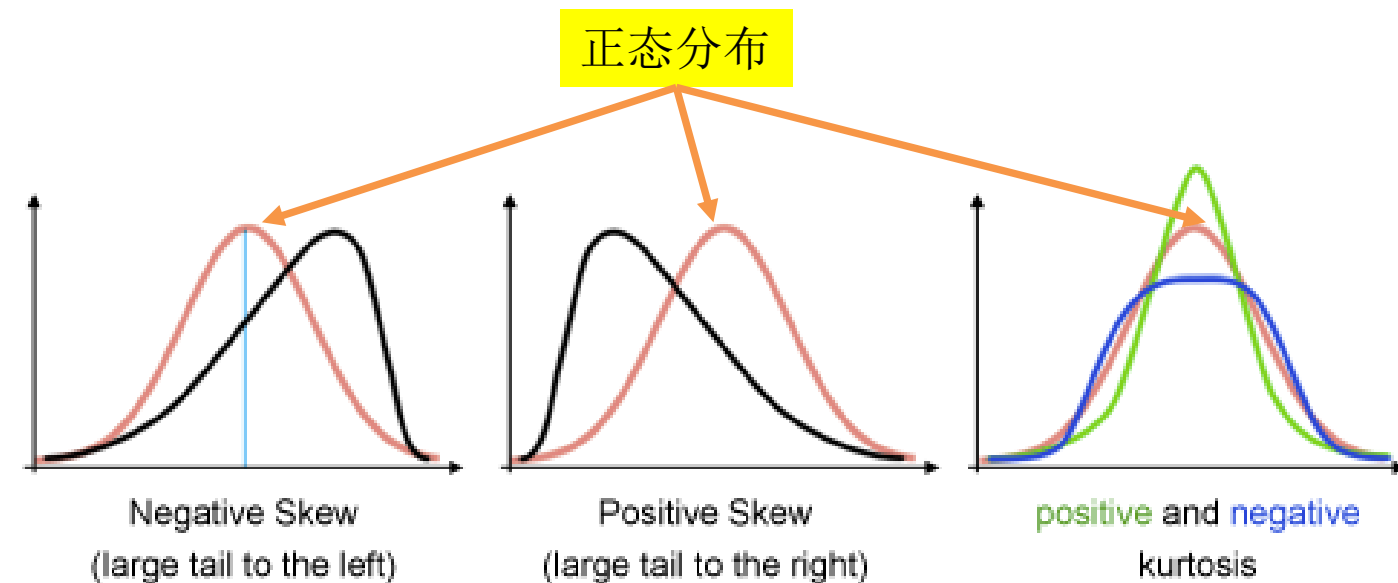
The binning issue: no "best" number of bins, and different bin sizes can reveal different features of the data. It requires some judgment, and perhaps some experimentation, based on the analyst's experience.

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)
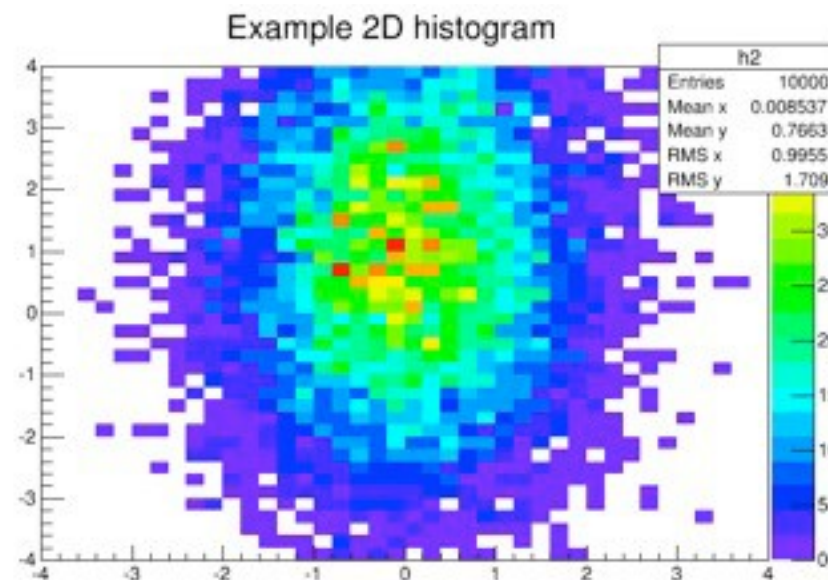
- ## Contour plots:

  ```
  h2->Draw("CONTZ");
  ```
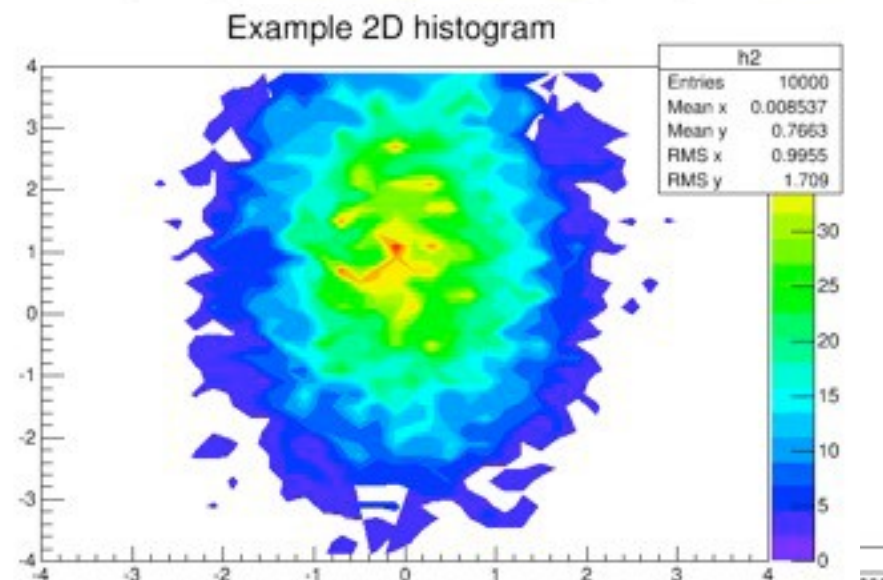
- ## Lego plots:

  ```
  h2->Draw("LEGO");
  ```
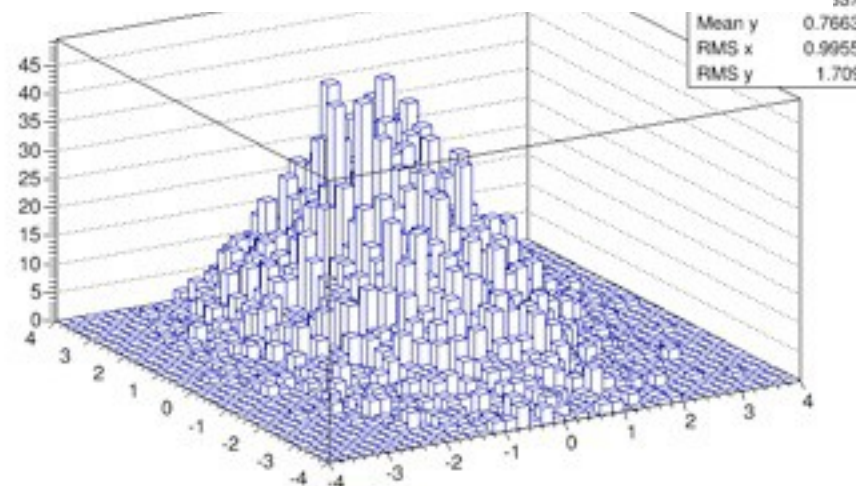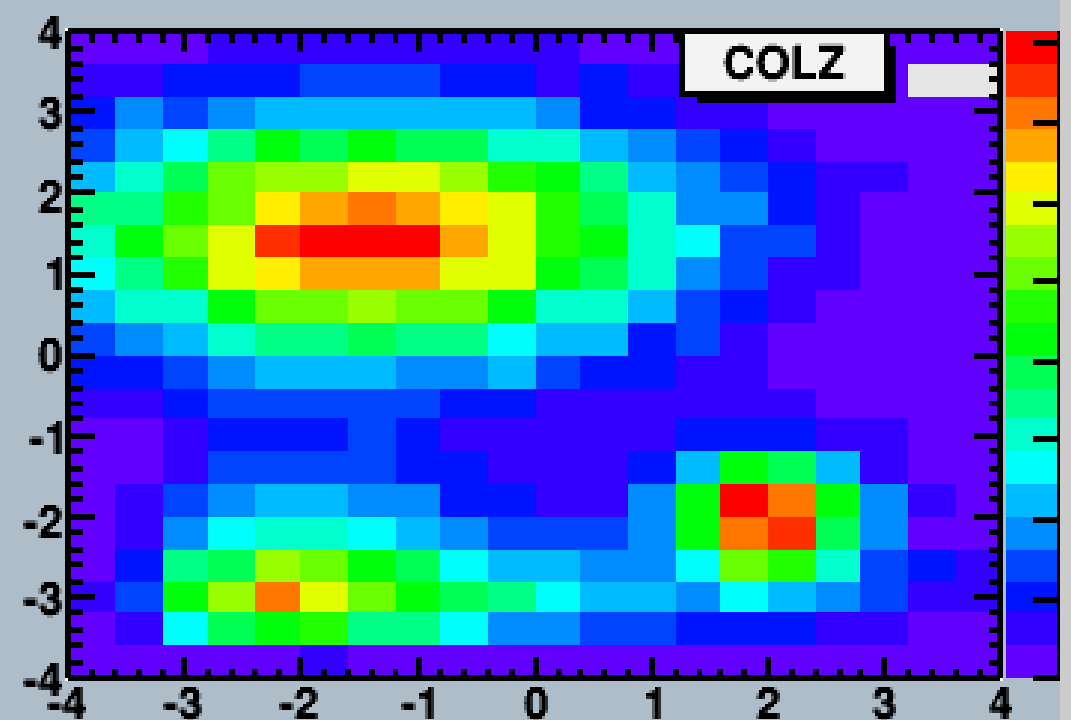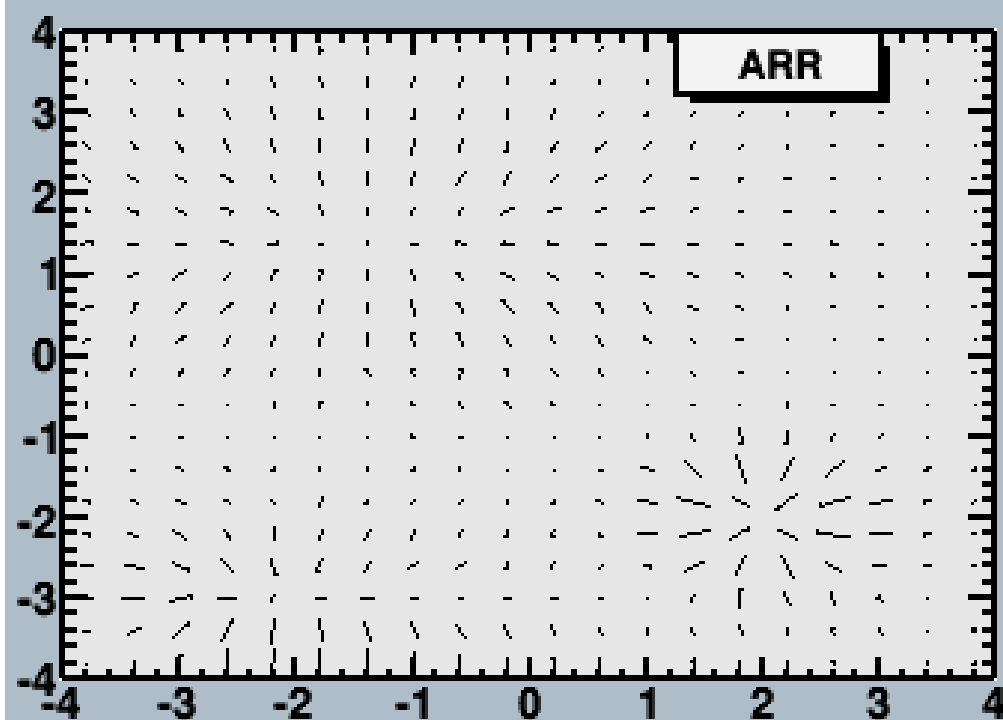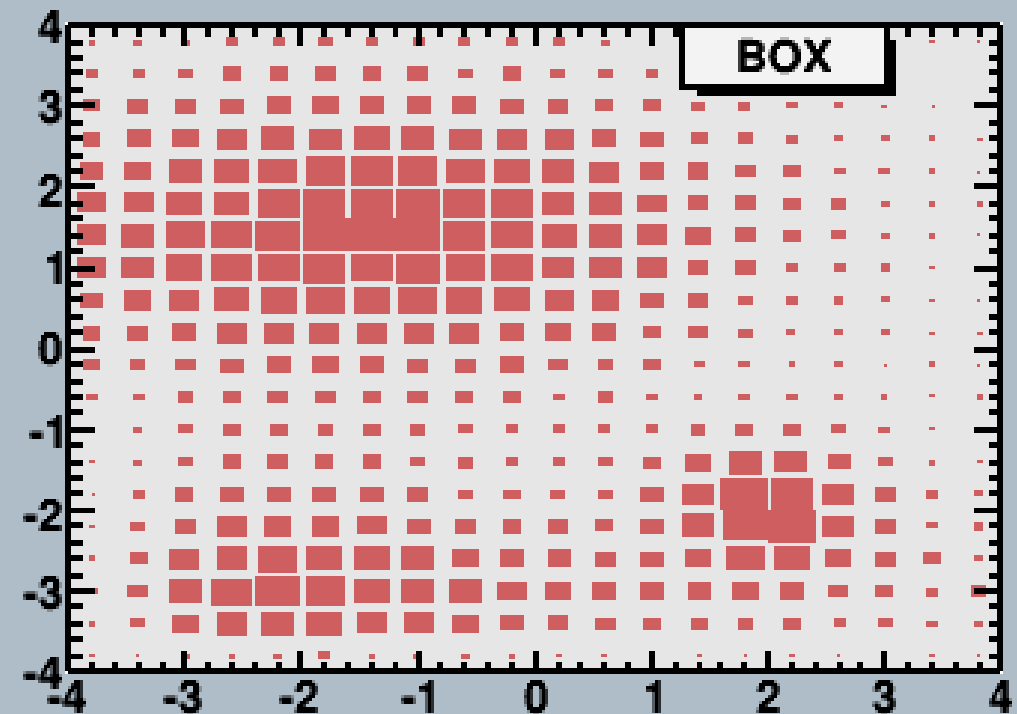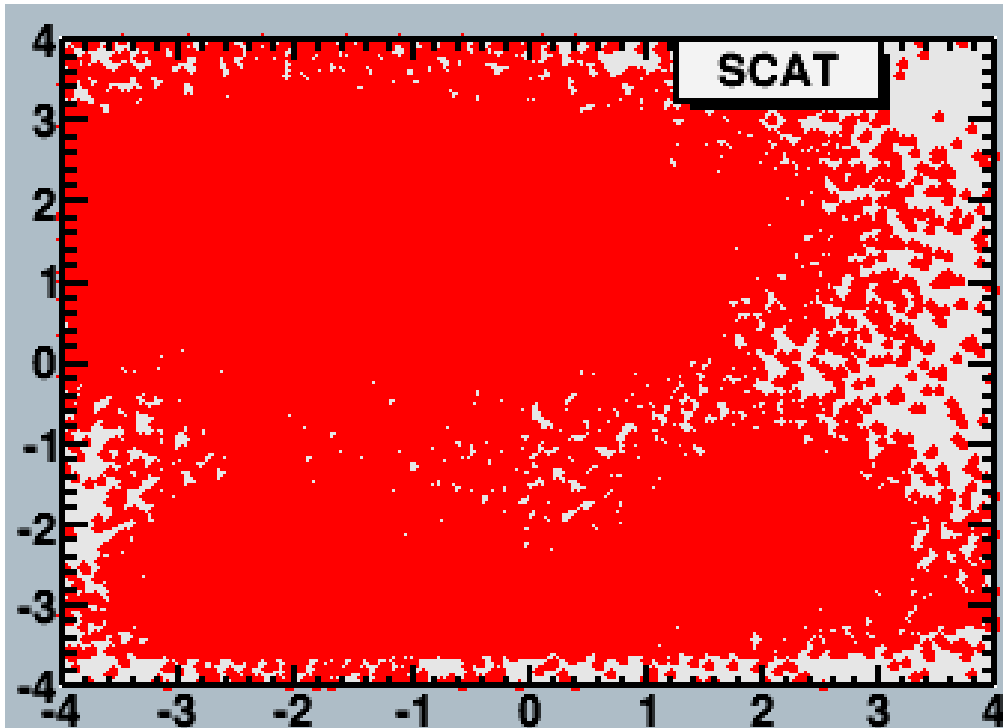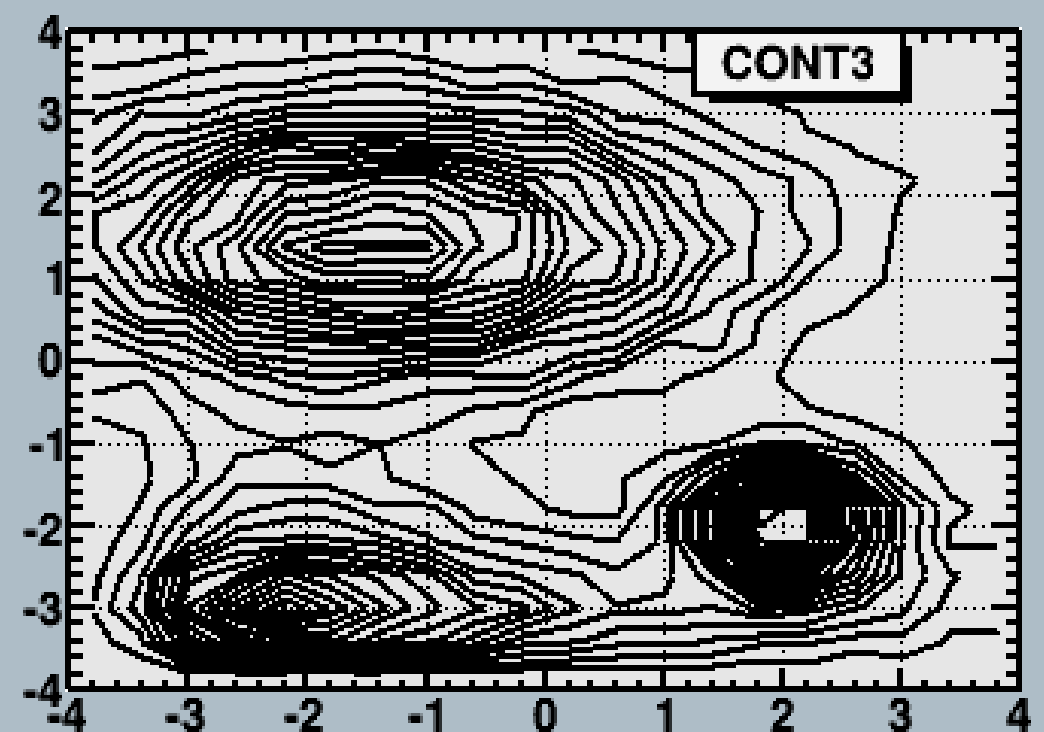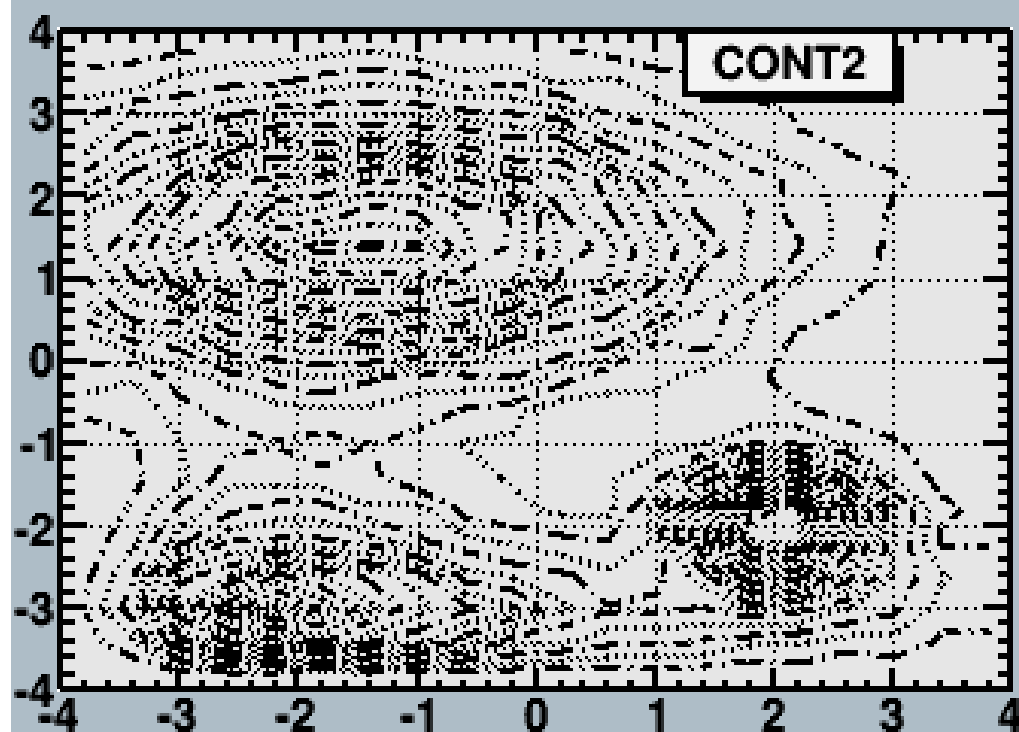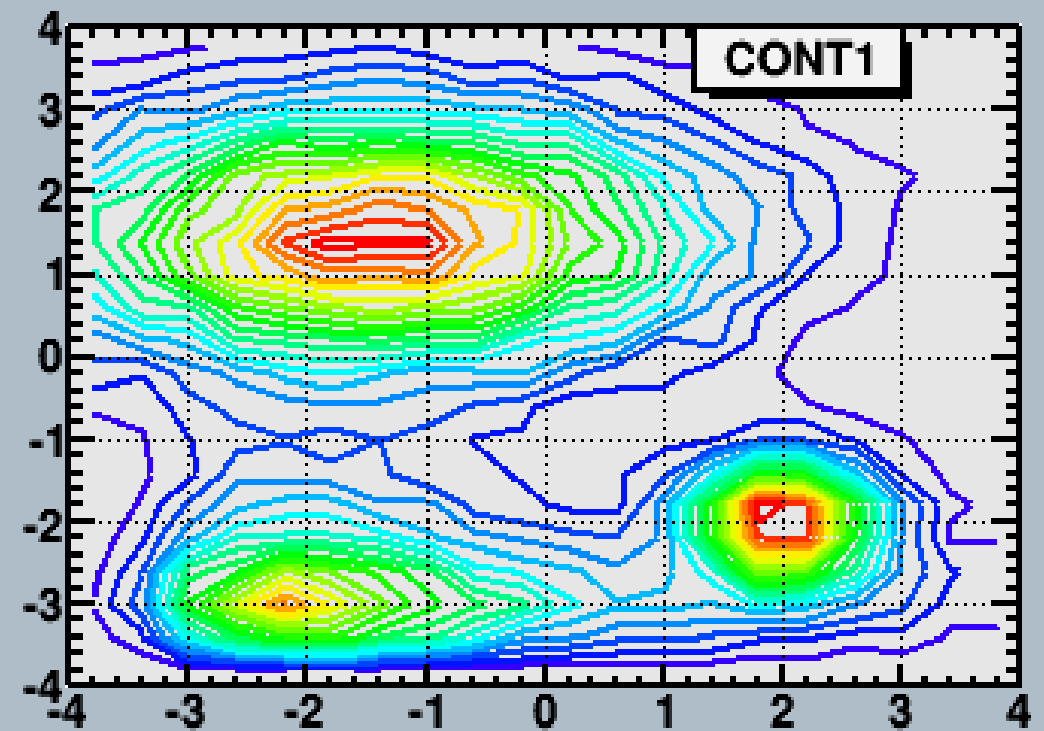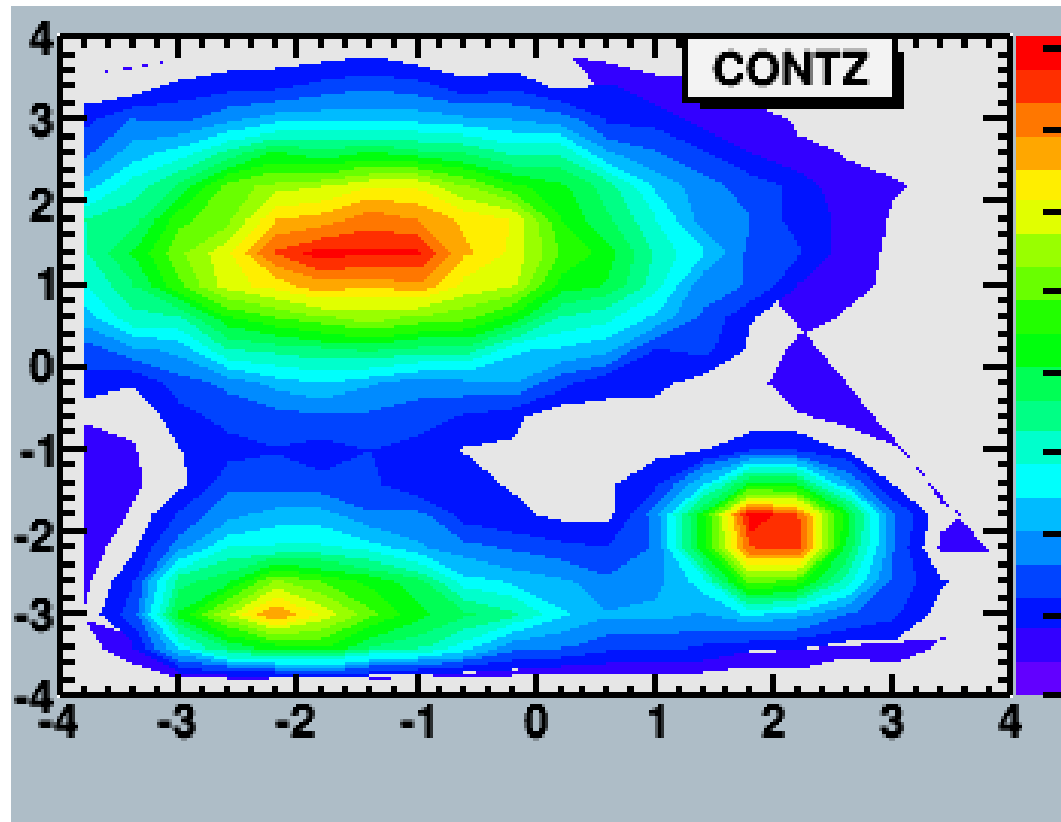
root *hist/draw2dopt.C*
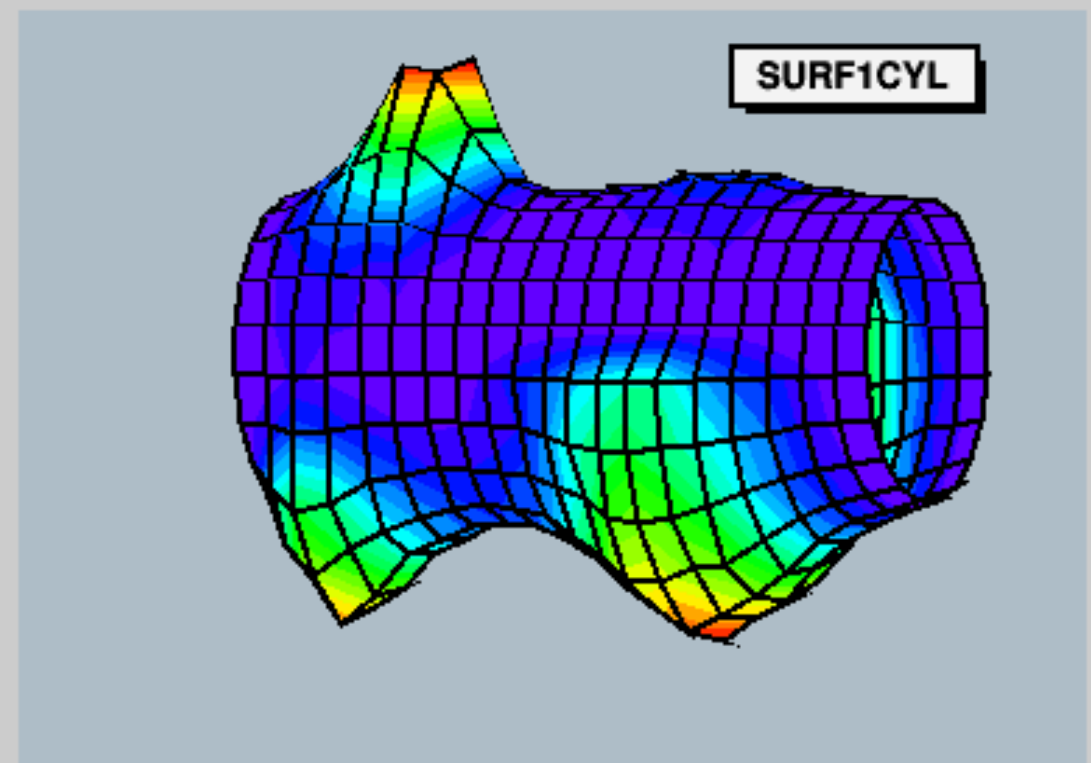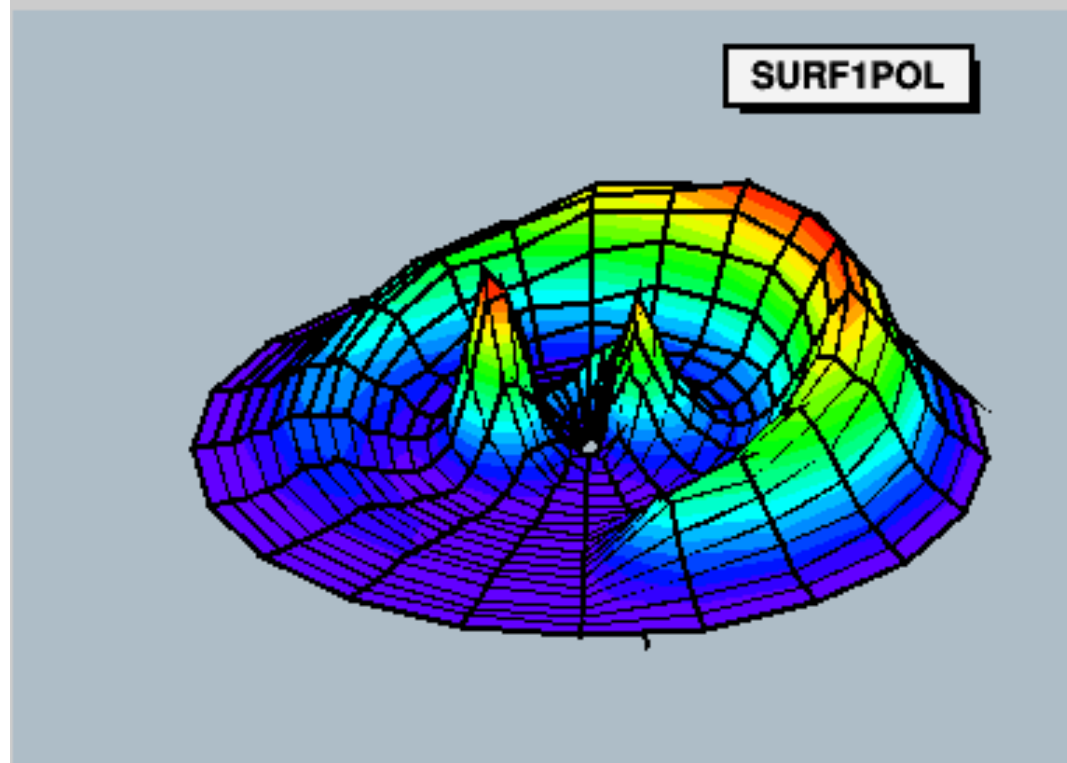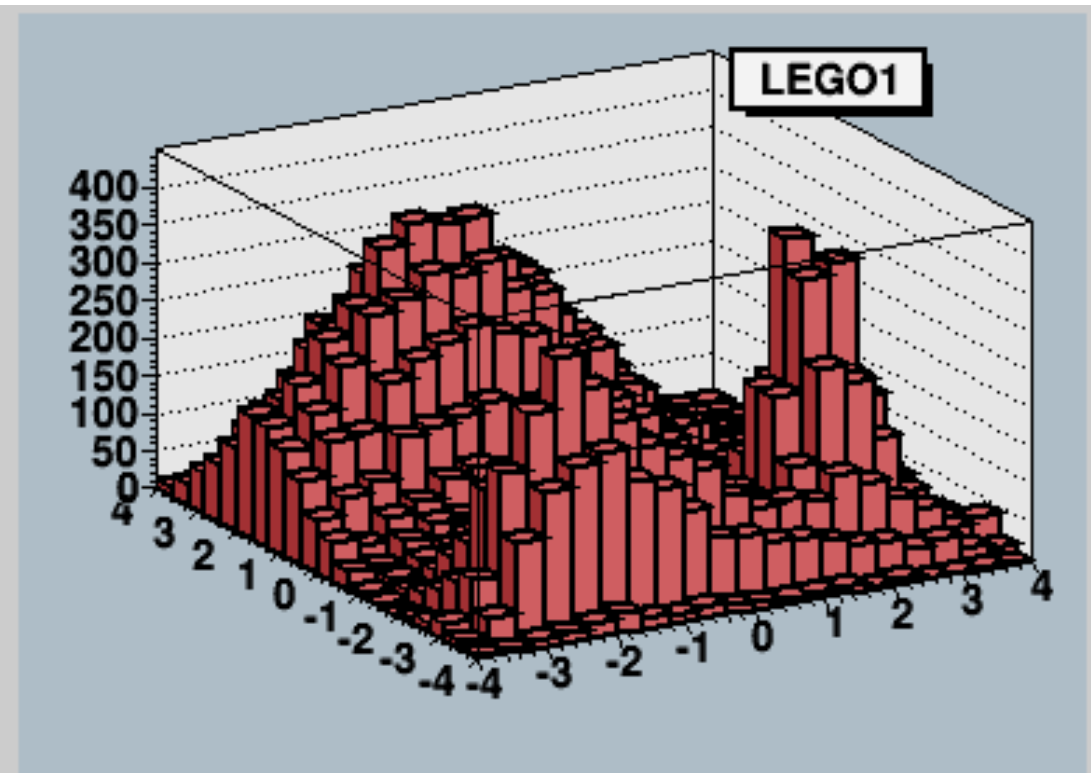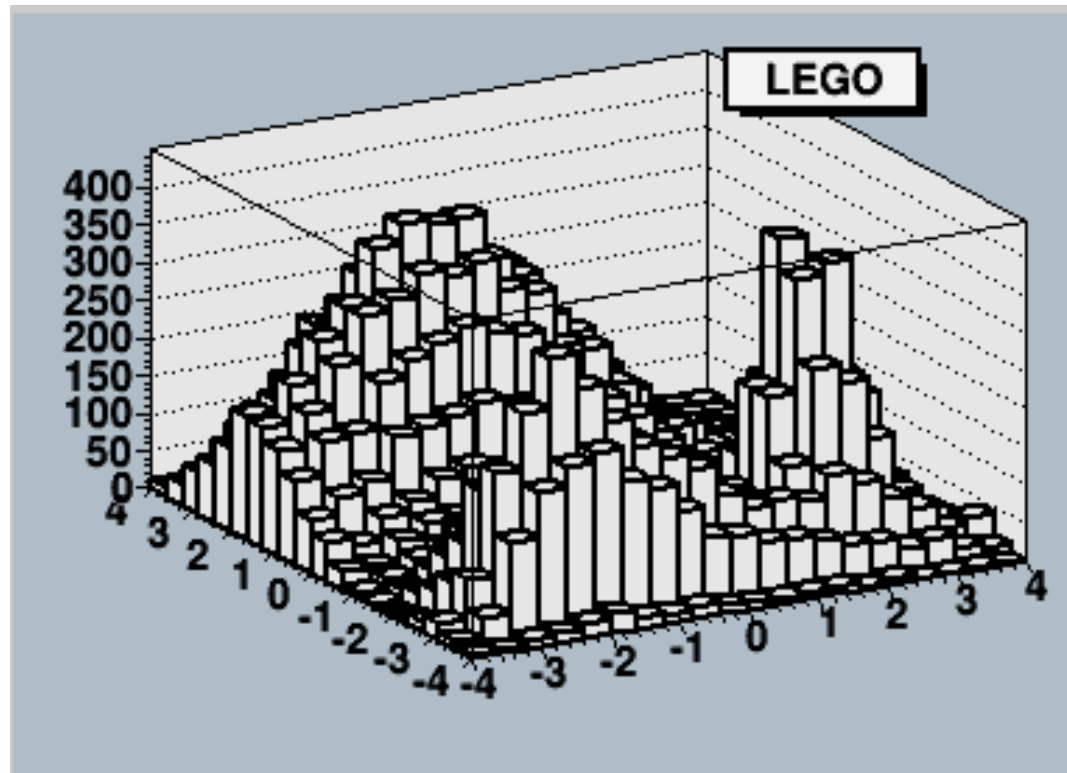
root *hist/draw2dopt.C*

root *hist/draw2dopt.C*

root *hist/draw2dopt.C*
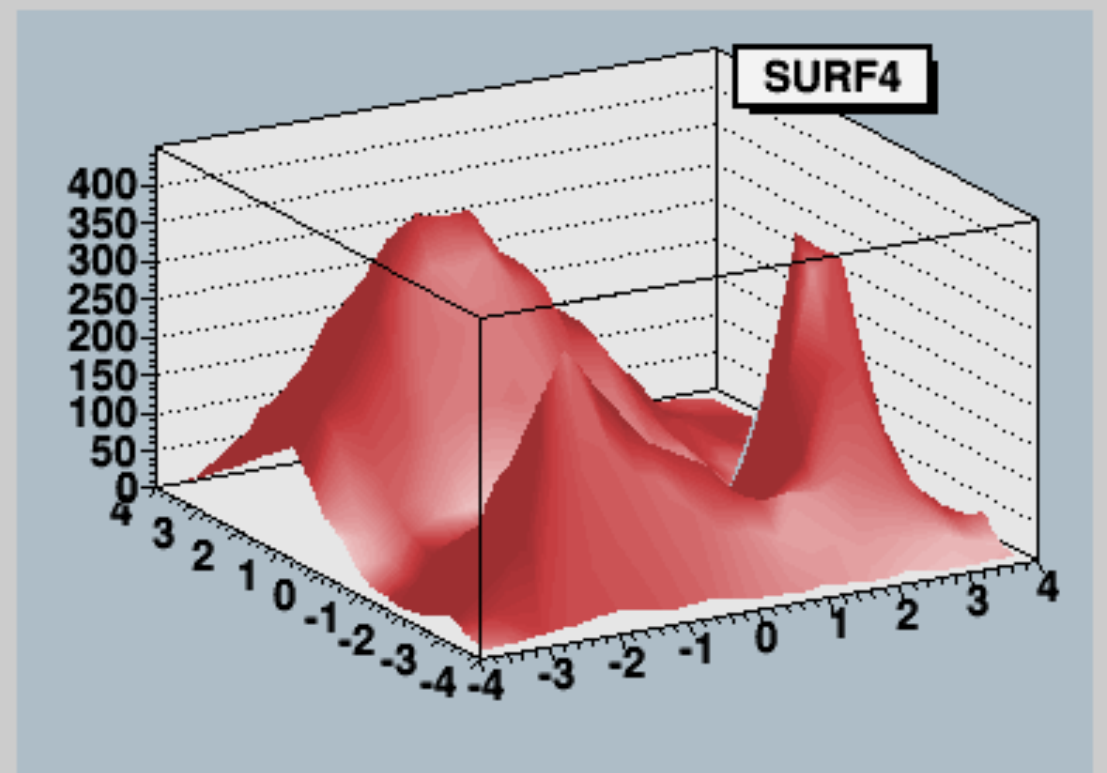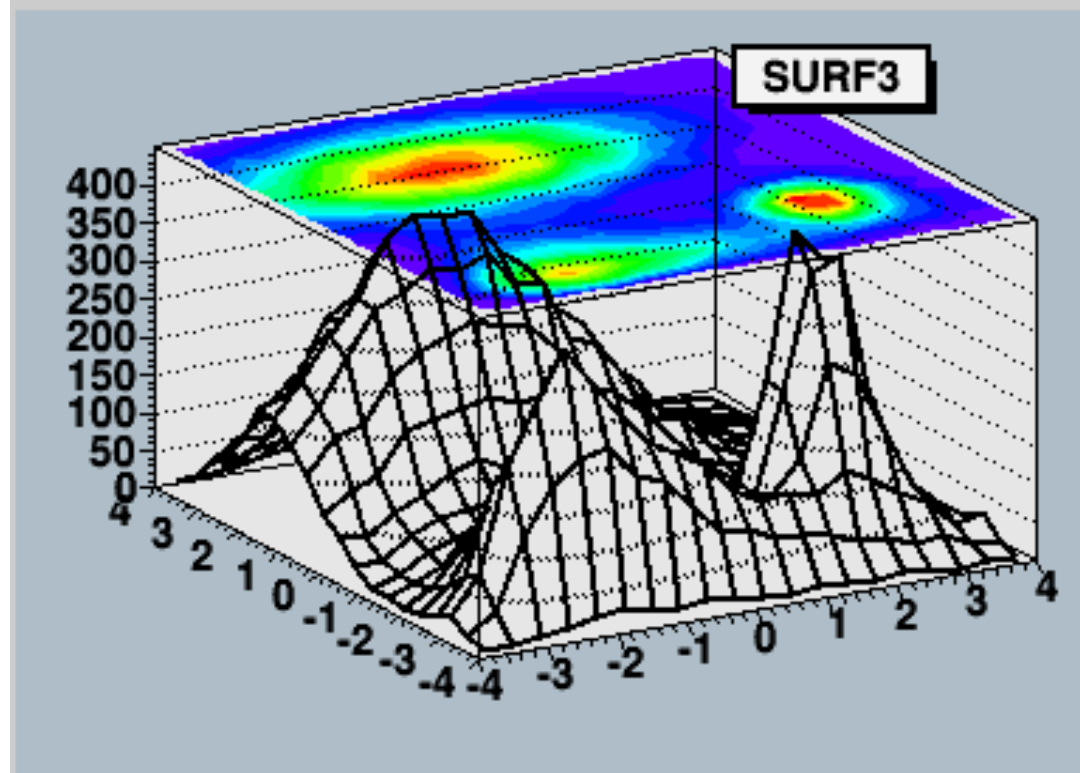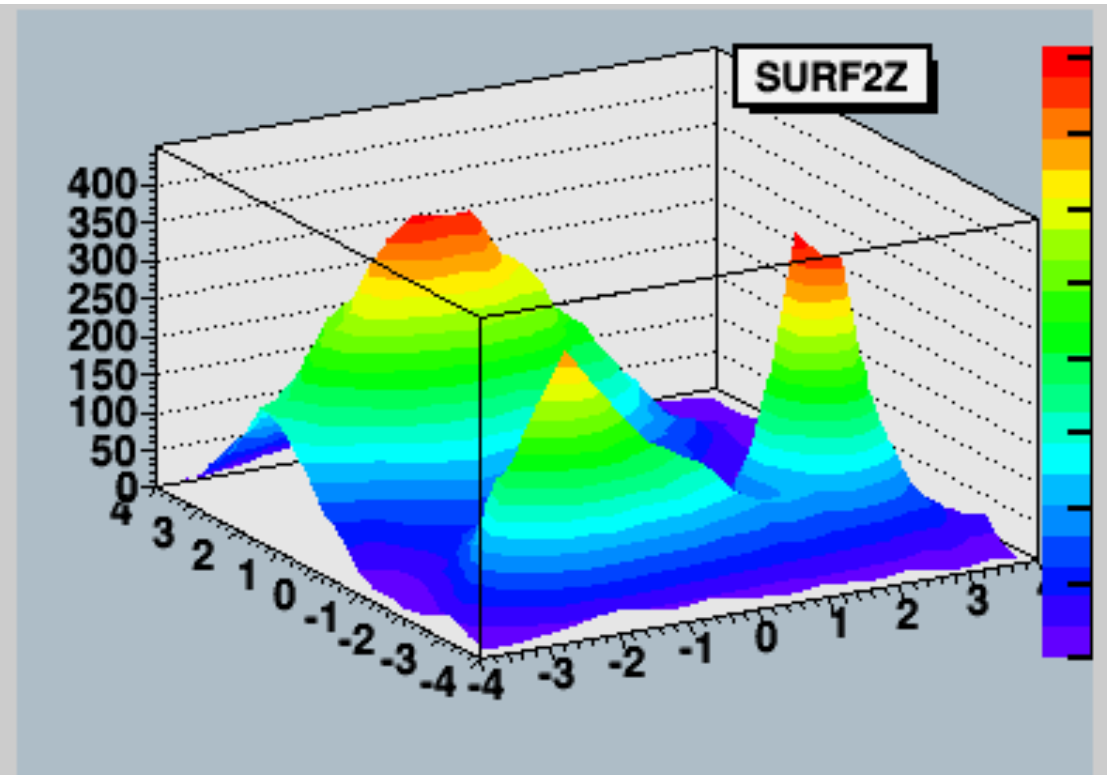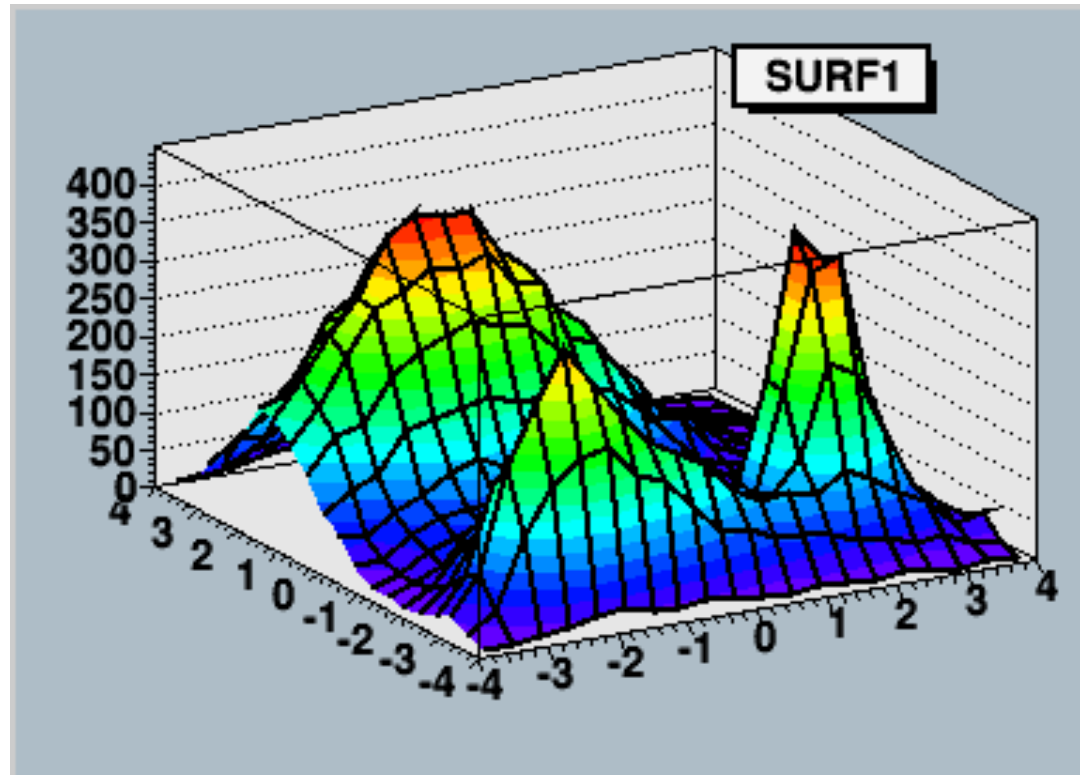
Other Histogram Types

- `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

Boxes

Boxes	
Entries	280739
Mean x	102.3
Mean y	2.011
RMS x	46.6
RMS y	0.8906

example:
`$ROOTSYS/tutorials/hist/th2polyBoxes.C`
`$ROOTSYS/tutorials/hist/sparsehist.C`
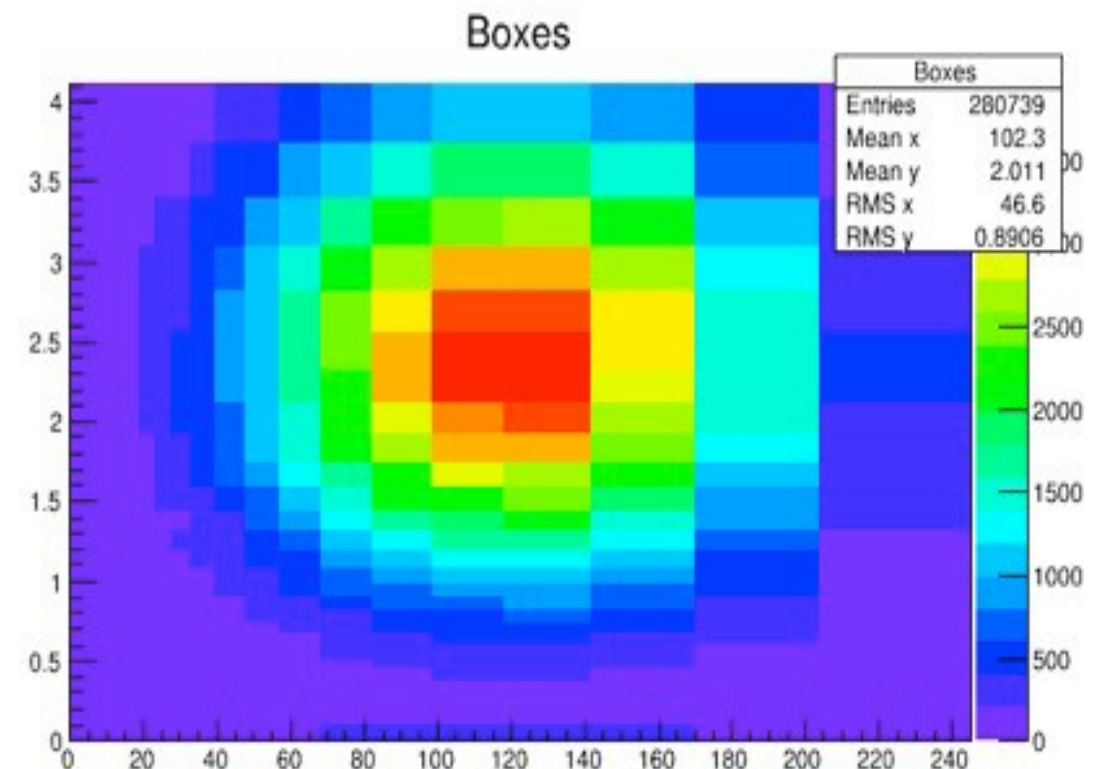
More on Graphs

- ## 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
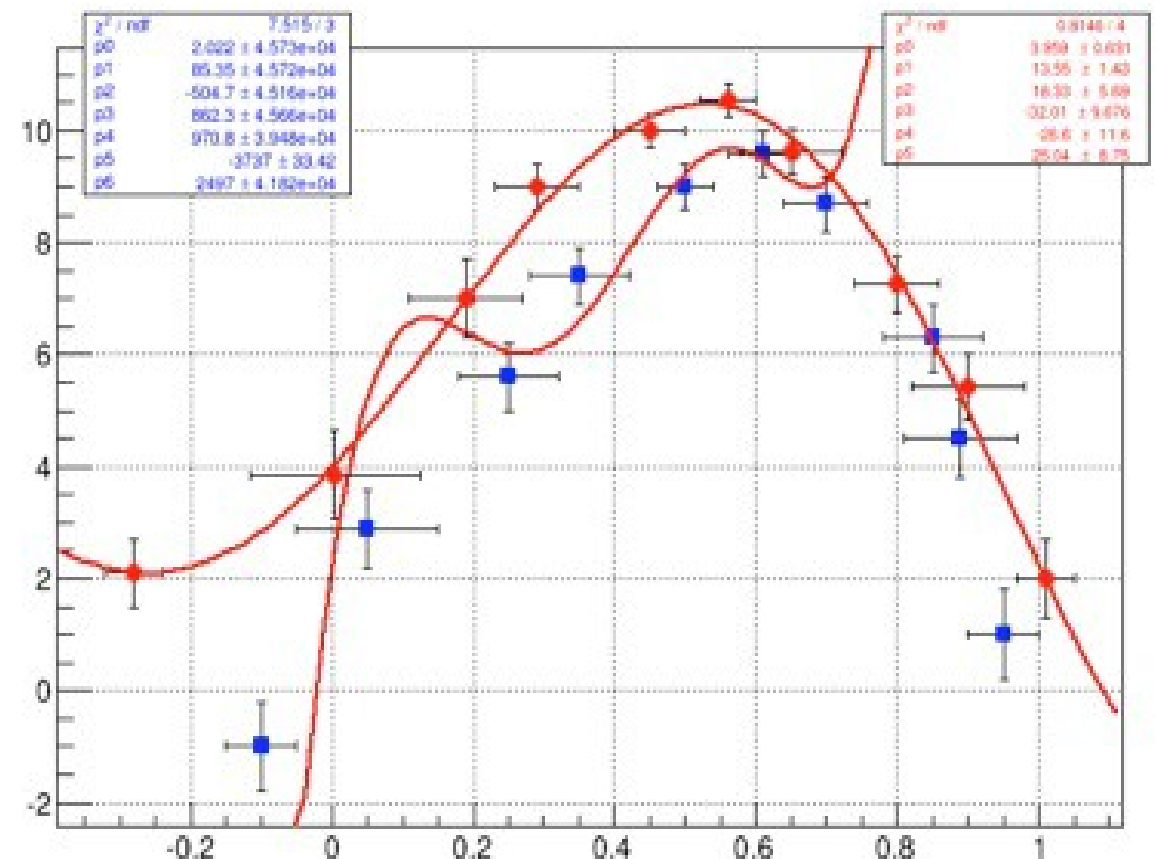
2D Function on the Graph2DErrors points
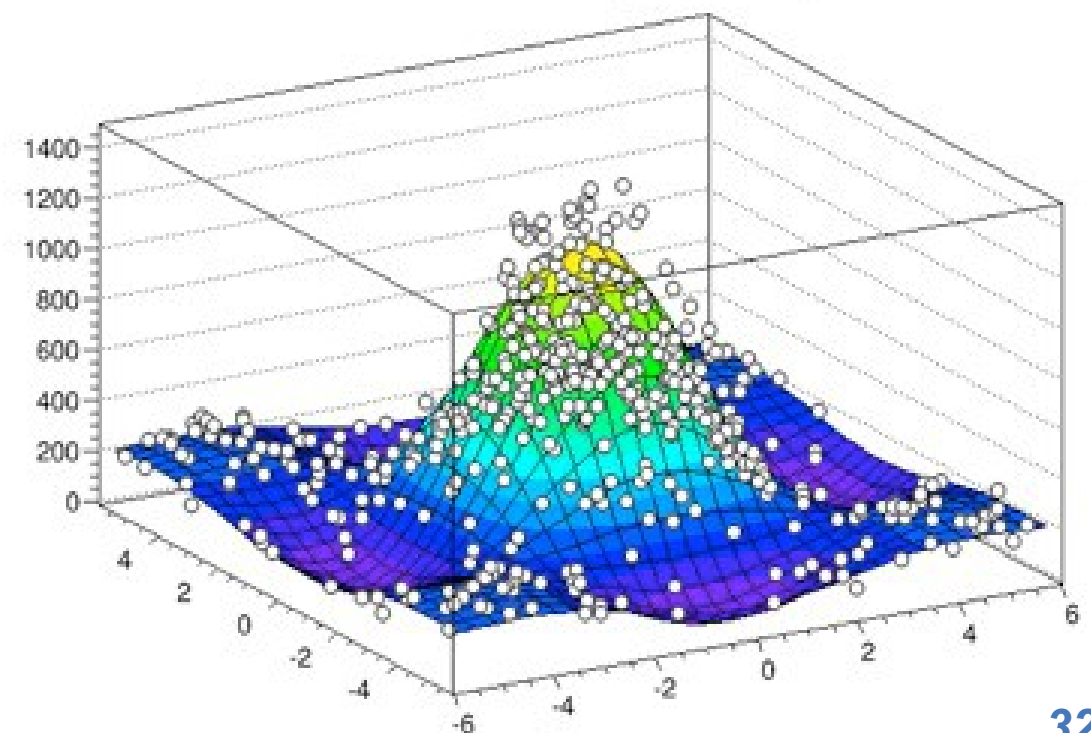
Profile Histograms

- 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",40,-4.0,4.0);
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();
```
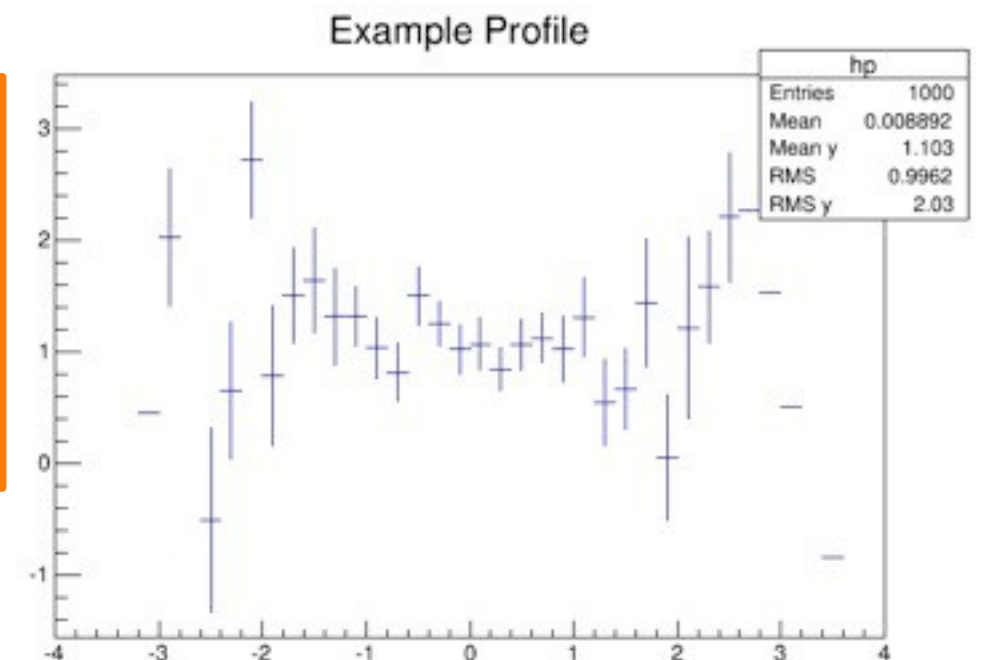
  – various options for displaying errors
  – default uses error in the sample mean (RMS/$\sqrt{N}$)

Projections

- One can make:
  - a 1-D projection of a 2-D histogram or profile. See **TH2**::ProfileX, **TH2**::ProfileY, **TProfile**::ProjectionX, **TProfile2D**::ProjectionXY, **TH2**::ProjectionX, **TH2**::ProjectionY.
  - a 1-D, 2-D or profile out of a 3-D histogram see **TH3**::ProjectionZ, **TH3**::Project3D.
- These projections can be fit via: **TH2**::FitSlicesX, **TH2**::FitSlicesY, **TH3**::FitSlicesZ.

hist/ DynamicSlice.C

TProfile2D

```
Profile2D(){
 gStyle->SetPalette(1);
 TCanvas *c1 = new TCanvas("c1","Profile histogram example",200,10,700,500);
 hprof2d = new TProfile2D("hprof2d","TProfile2D;px;py;pz",40,-4,4,40,-4,4,0,20);
 Float_t px, py, pz;
 for ( Int_t i=0; i<25000; i++) {
    gRandom->Rannor(px,py);
    pz = px*px + py*py;
    hprof2d->Fill(px,py,pz,1);
 }
 hprof2d->Draw("SURF3ZE");
}
```

Return 2 numbers distributed following a gaussian with mean=0 and sigma=1
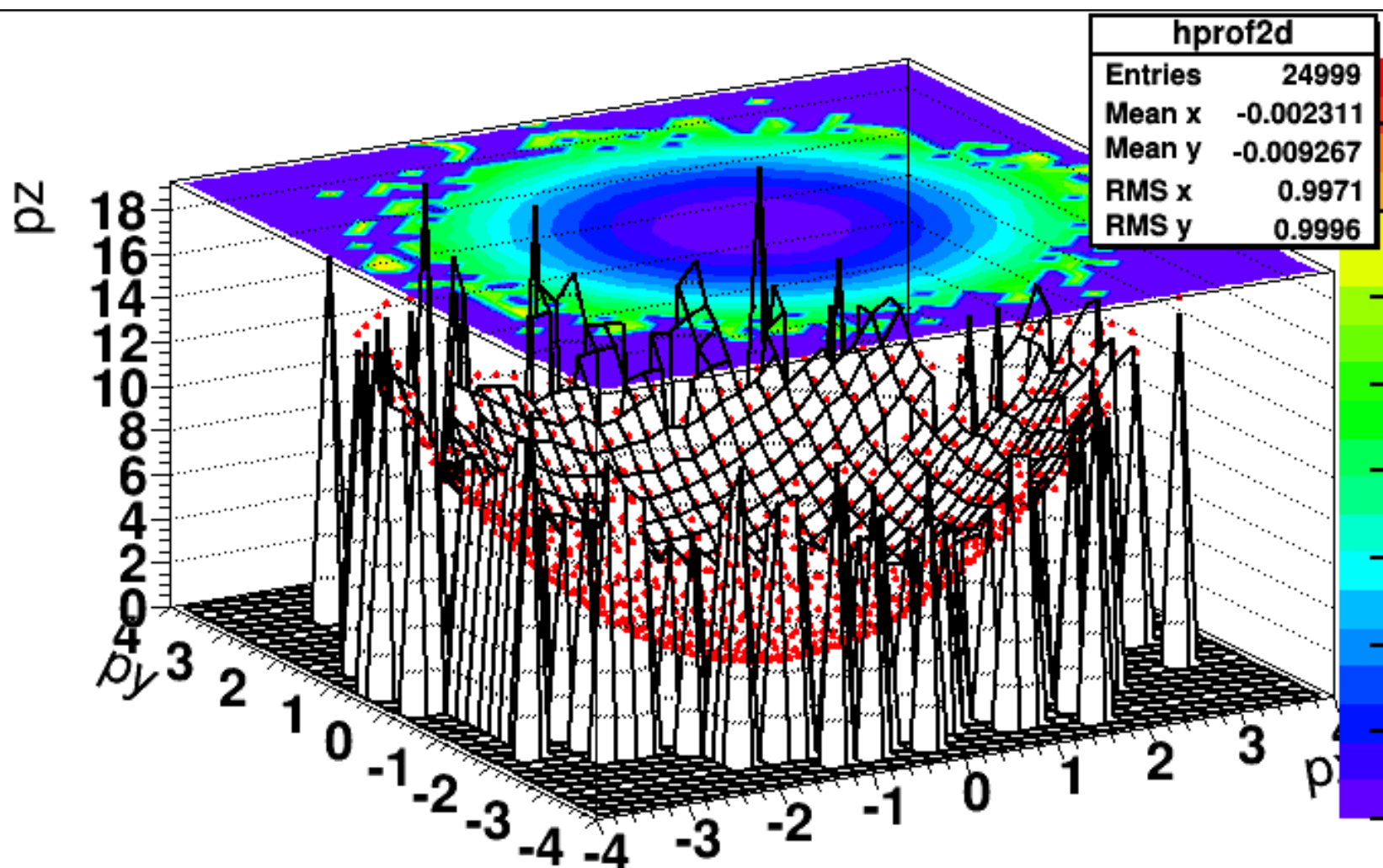
Weighted Histograms

- 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:

      ```
      h2->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)

Histograms in Files

The following statements create a ROOT file and store a histogram on the file

```
TFile f("histos.root","new");
TH1F h1("hgaus","histo from a gaussian",100,-3,3);
h1.FillRandom("gaus",10000);
h1->Write();
```

To read this histogram in another ROOT session, do:

```
TFile f("histos.root");
TH1F *h = (TH1F*)f.Get("hgaus");
```

One can save all histograms in memory to the file by:

```
file->Write();
```

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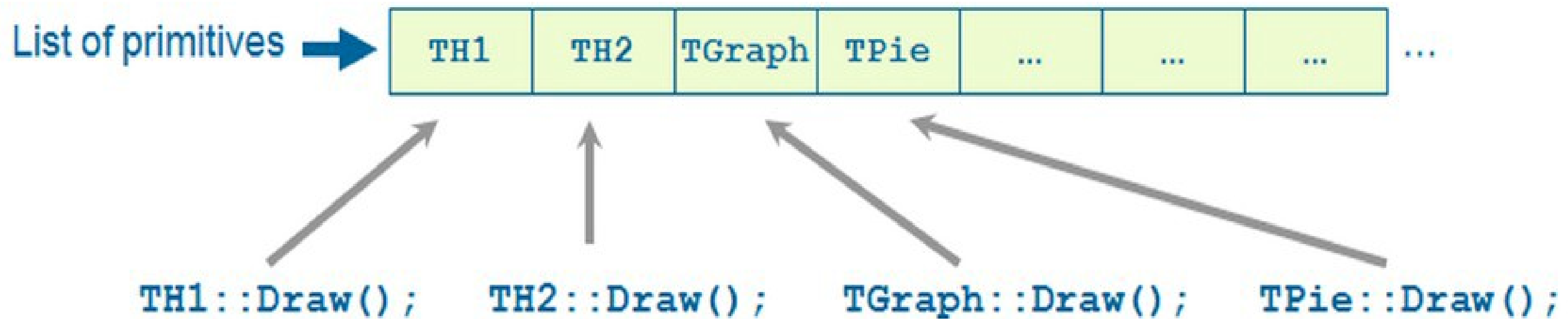
Adding an element into a Graphics Pad is done by the *Draw()* method of each classes.

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.

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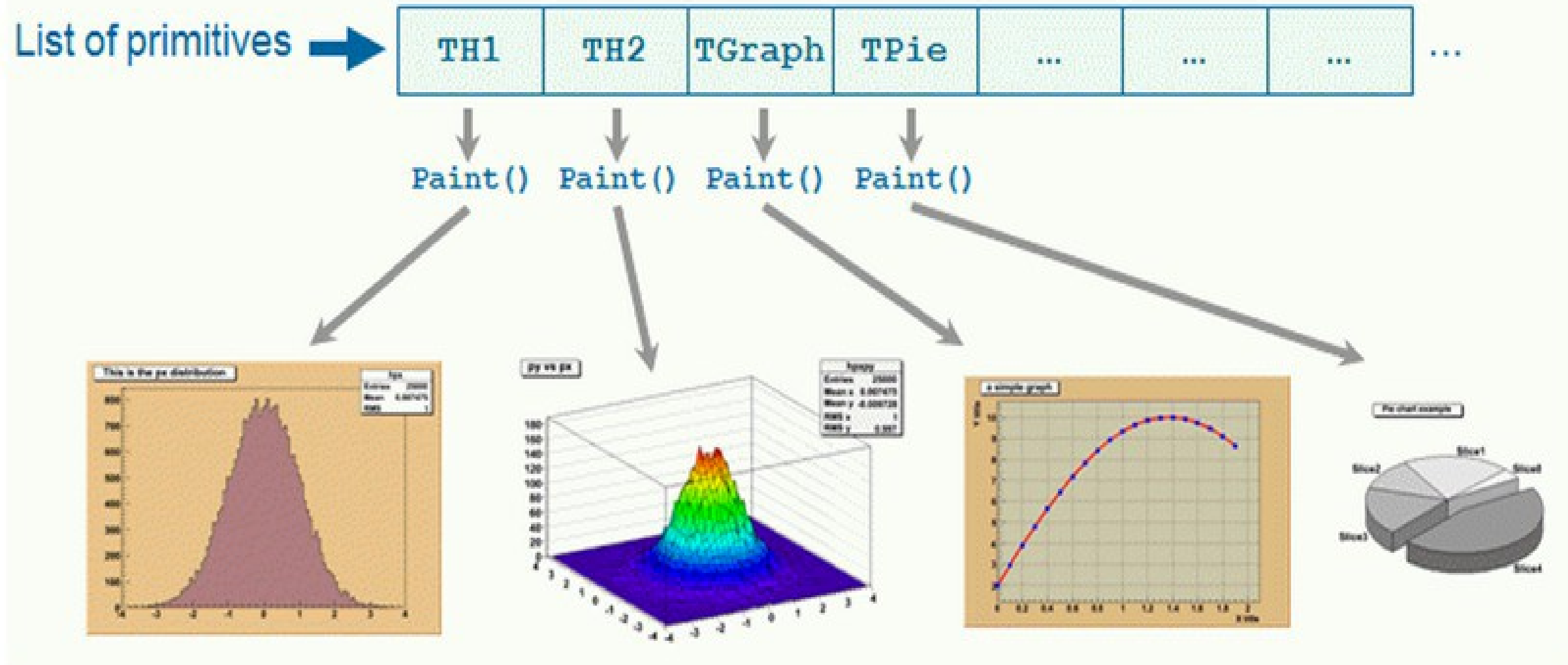
The Graphics Pad's (re)painting does not need to be done explicitly by the ROOT user. It is done automatically at the end of a macro execution or when a Graphics Pad has been modified.

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 can 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();
```
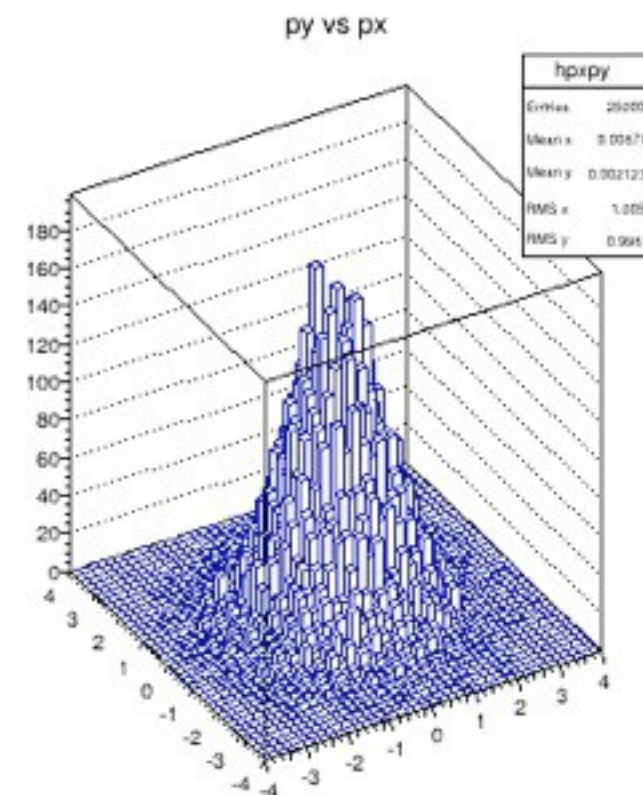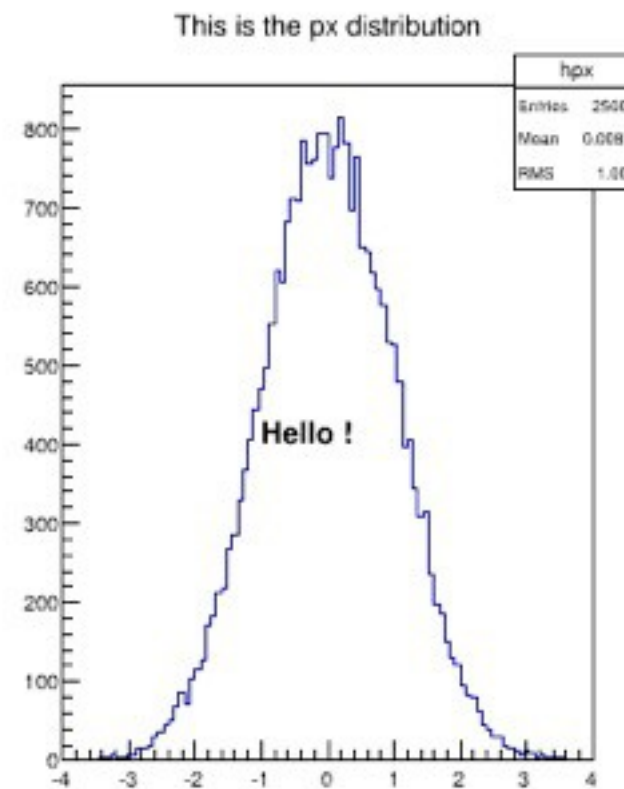
Visualization Techniques in ROOT

How ROOT visualize

2, 3, 4 and N data variables

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.
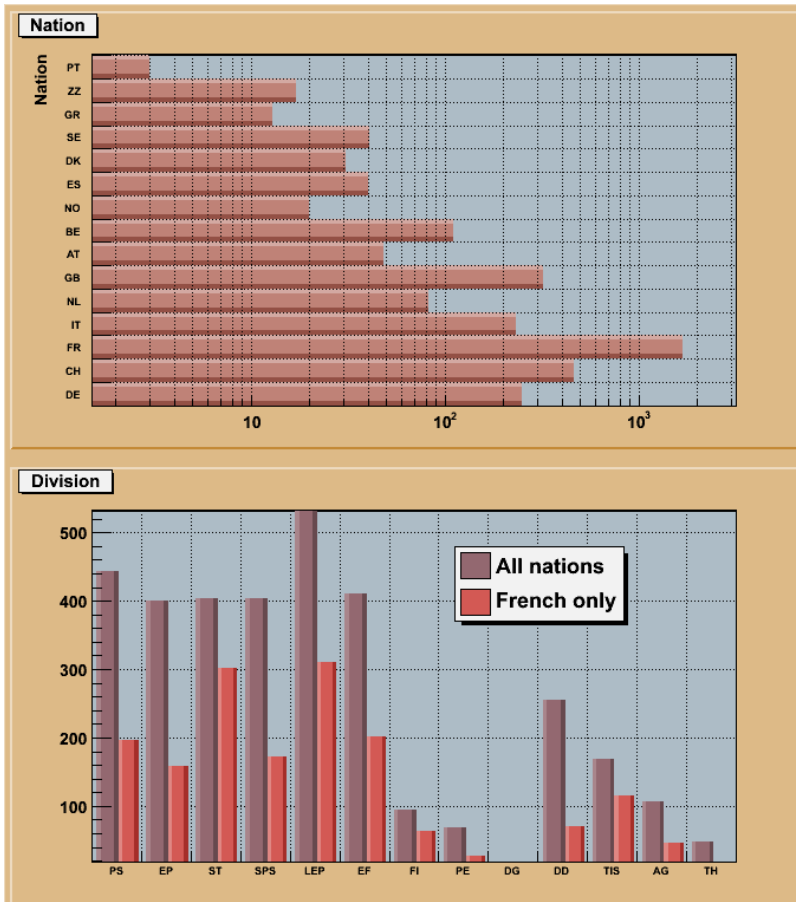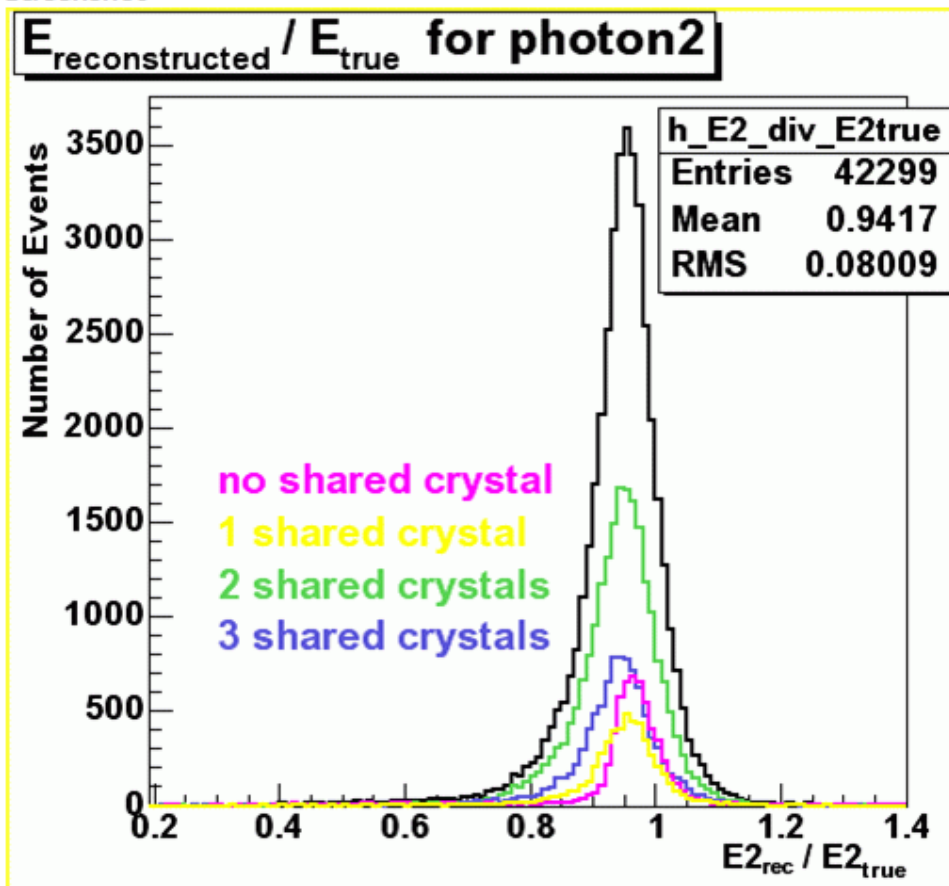
Bar charts and lines are a common way to represent 1D histograms.
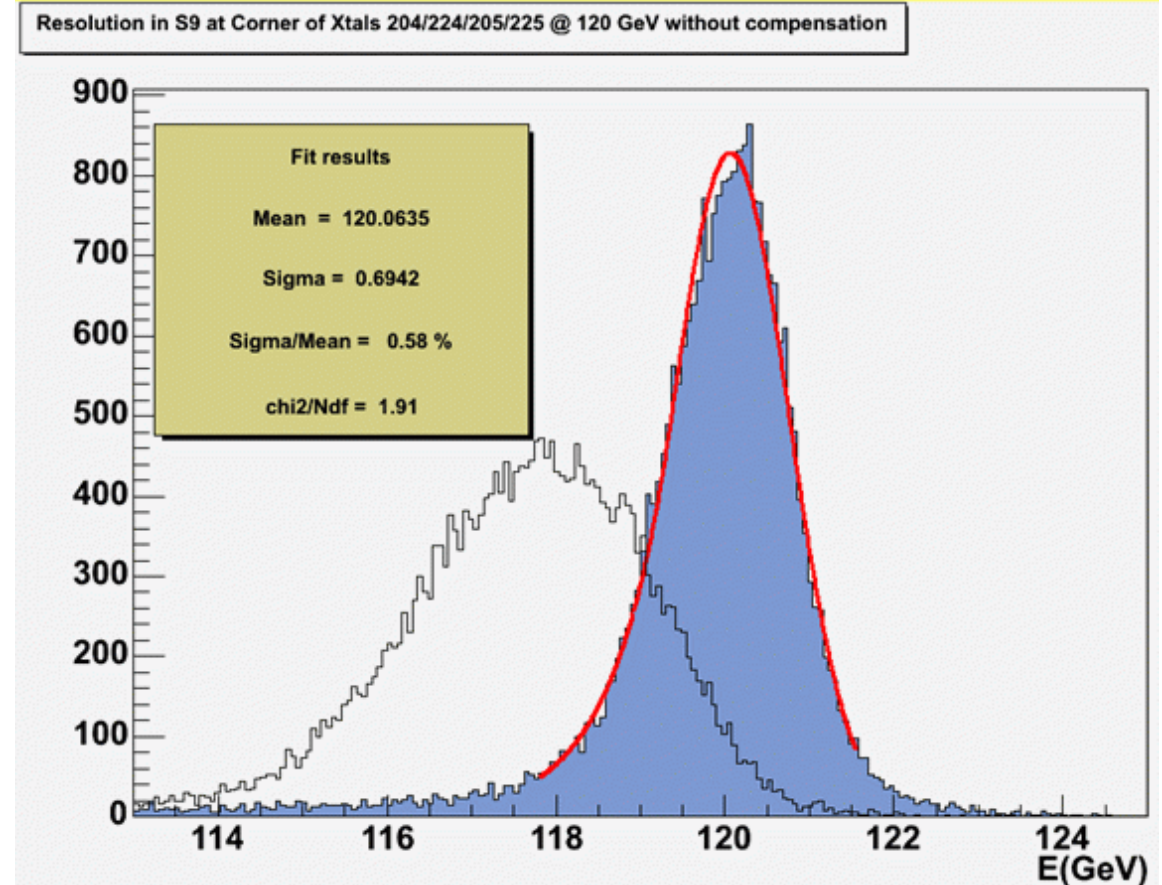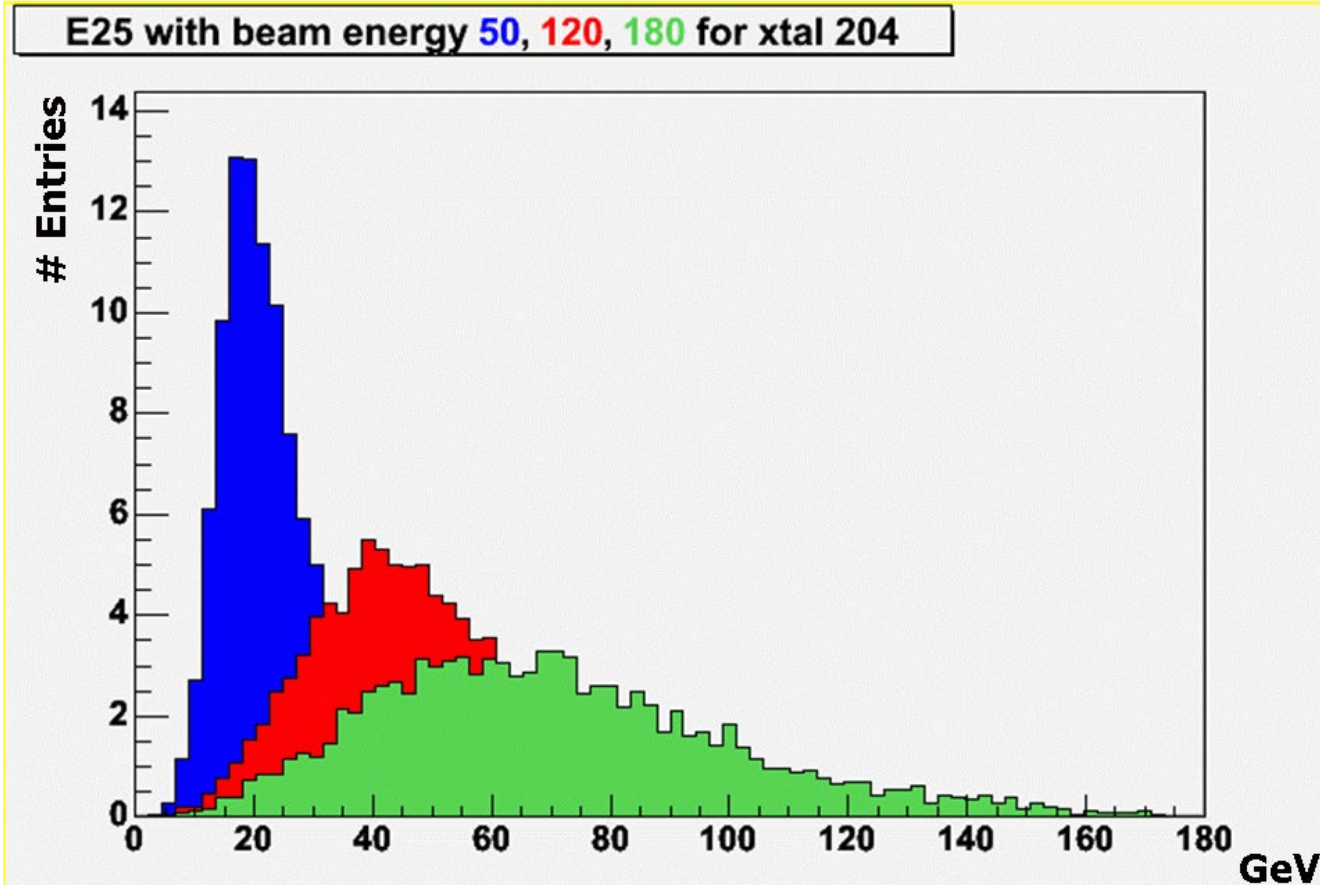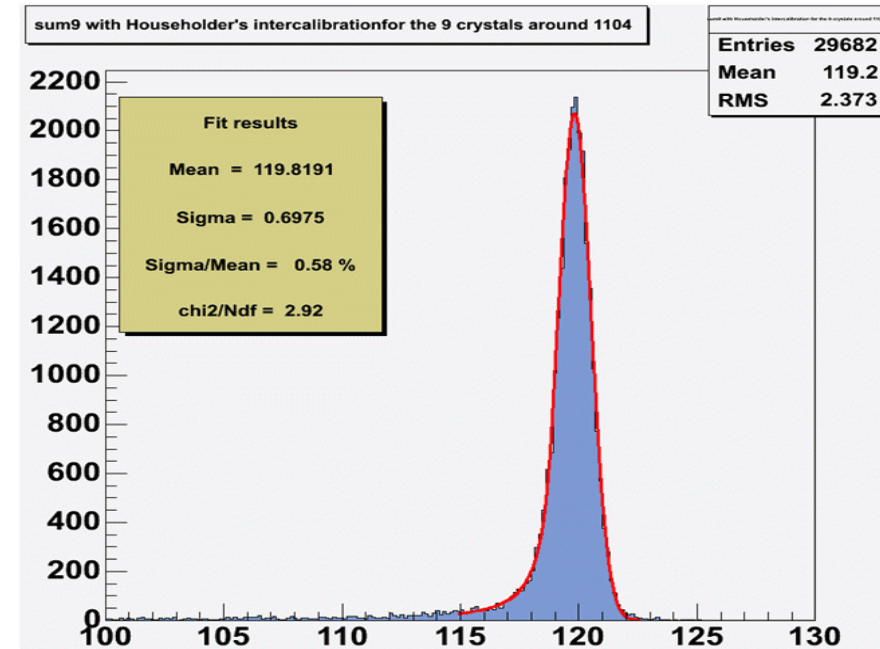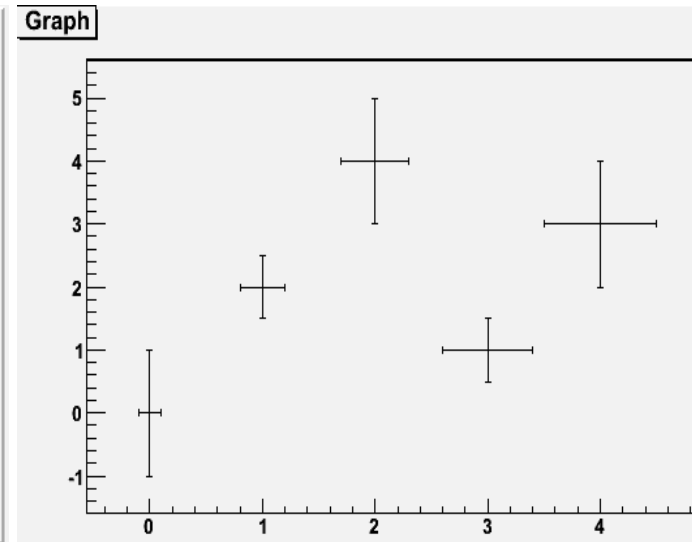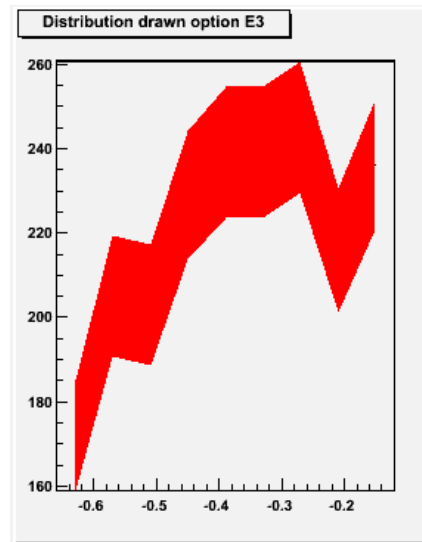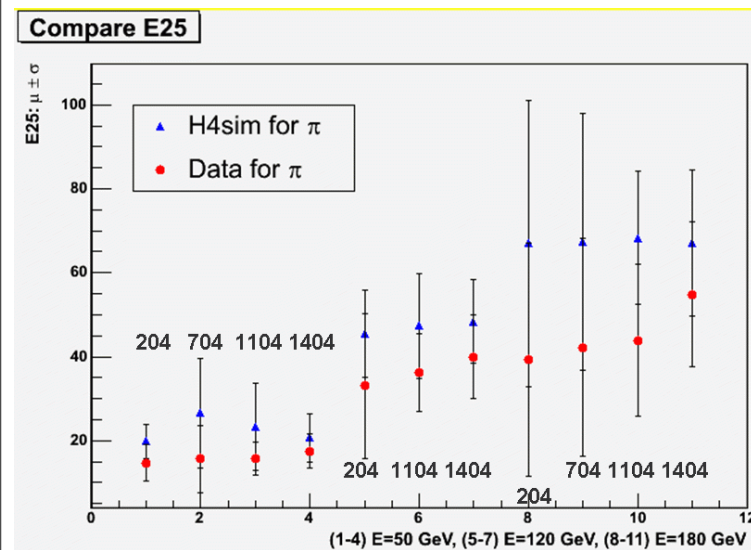
Errors can be represented as bars, band, rectangles. They can be symmetric, asymmetric or bent. 1D histograms and graphs can be drawn that way
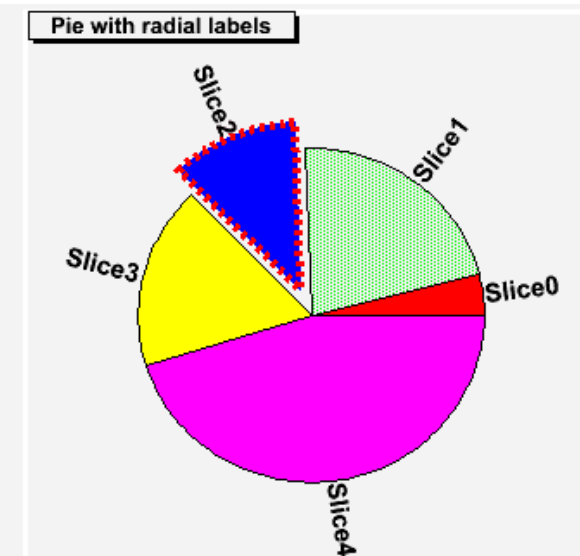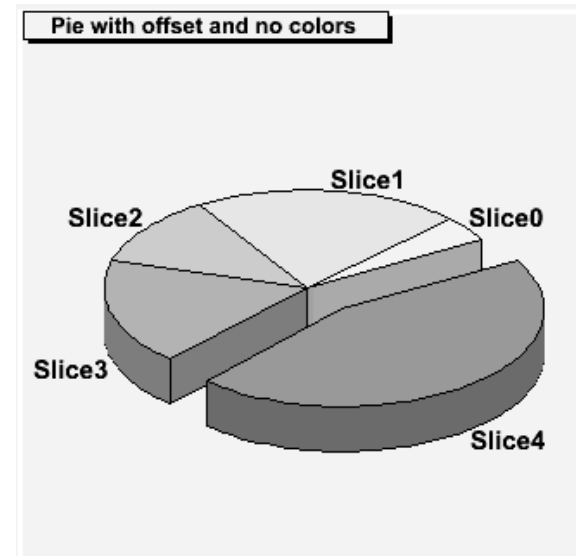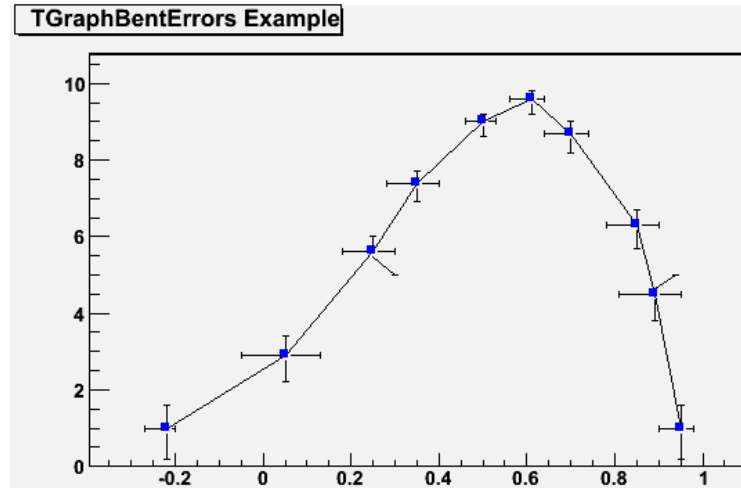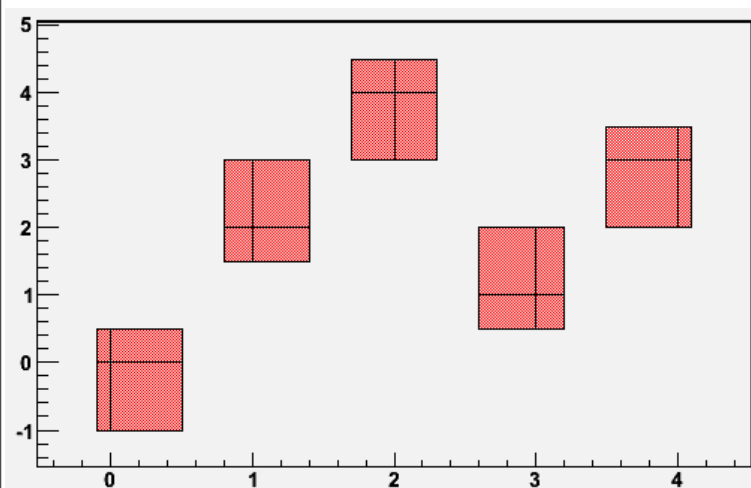
Pie charts can be used to visualize 1D histograms. They also can be created from a simple mono dimensional vector.
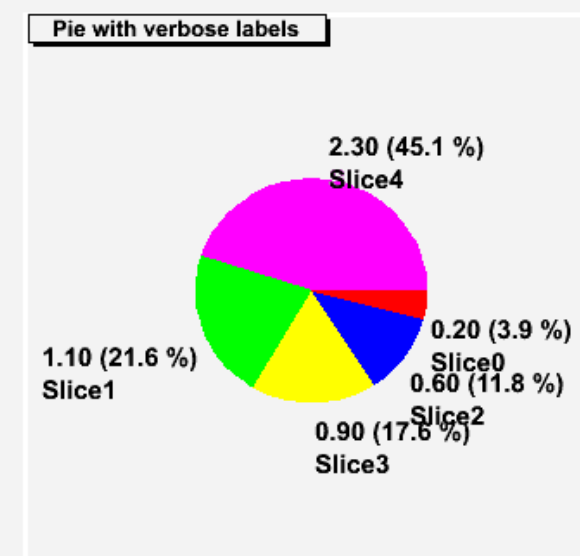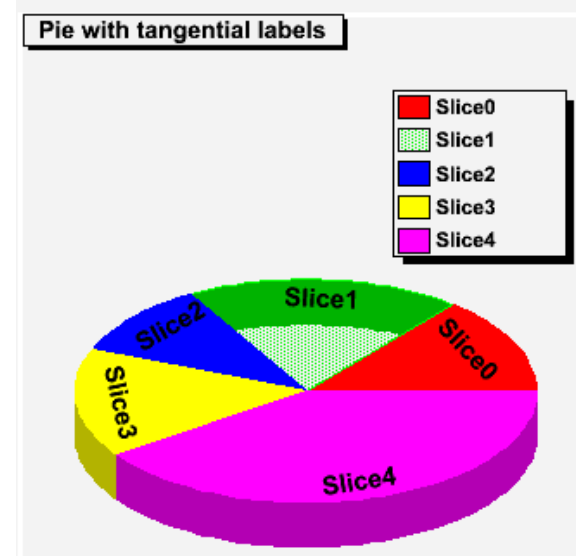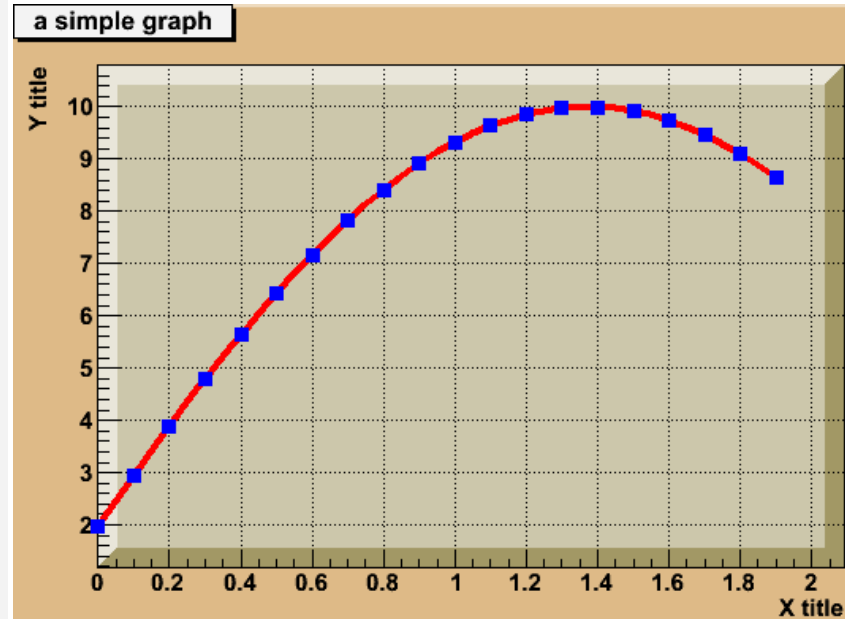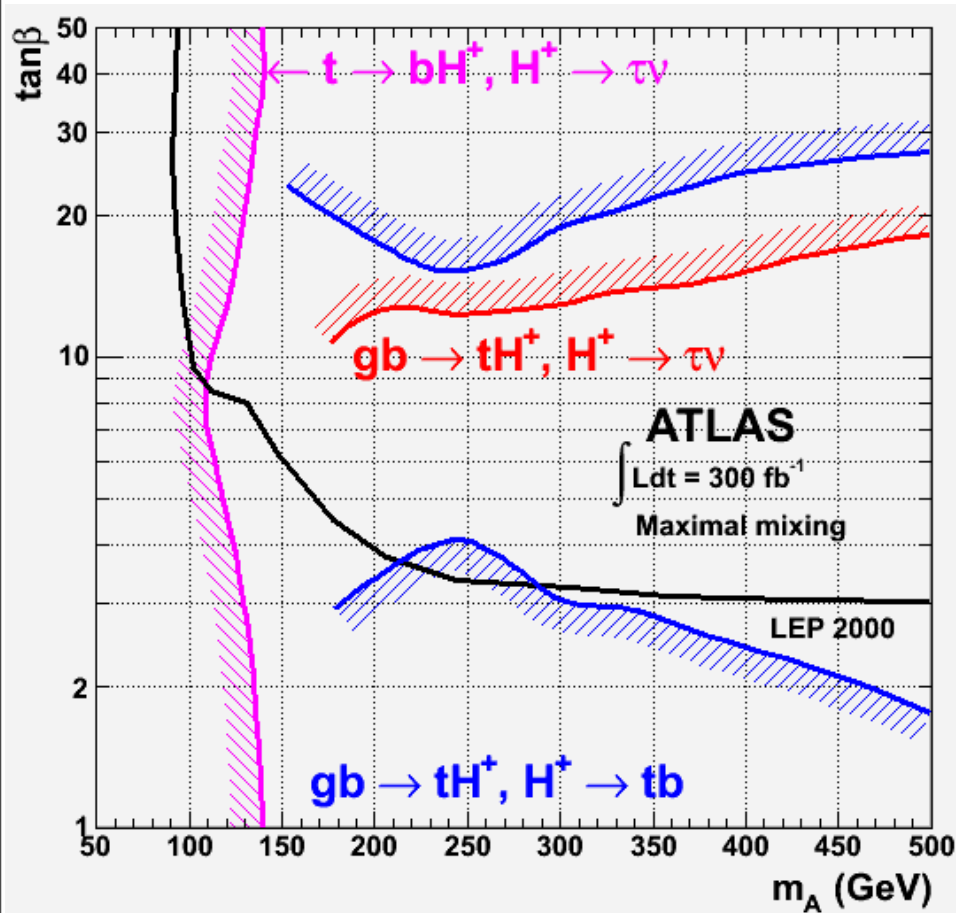
Graphs can be drawn as simple lines, like functions. They can also visualize exclusion zones or be plotted in polar coordinates.

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.
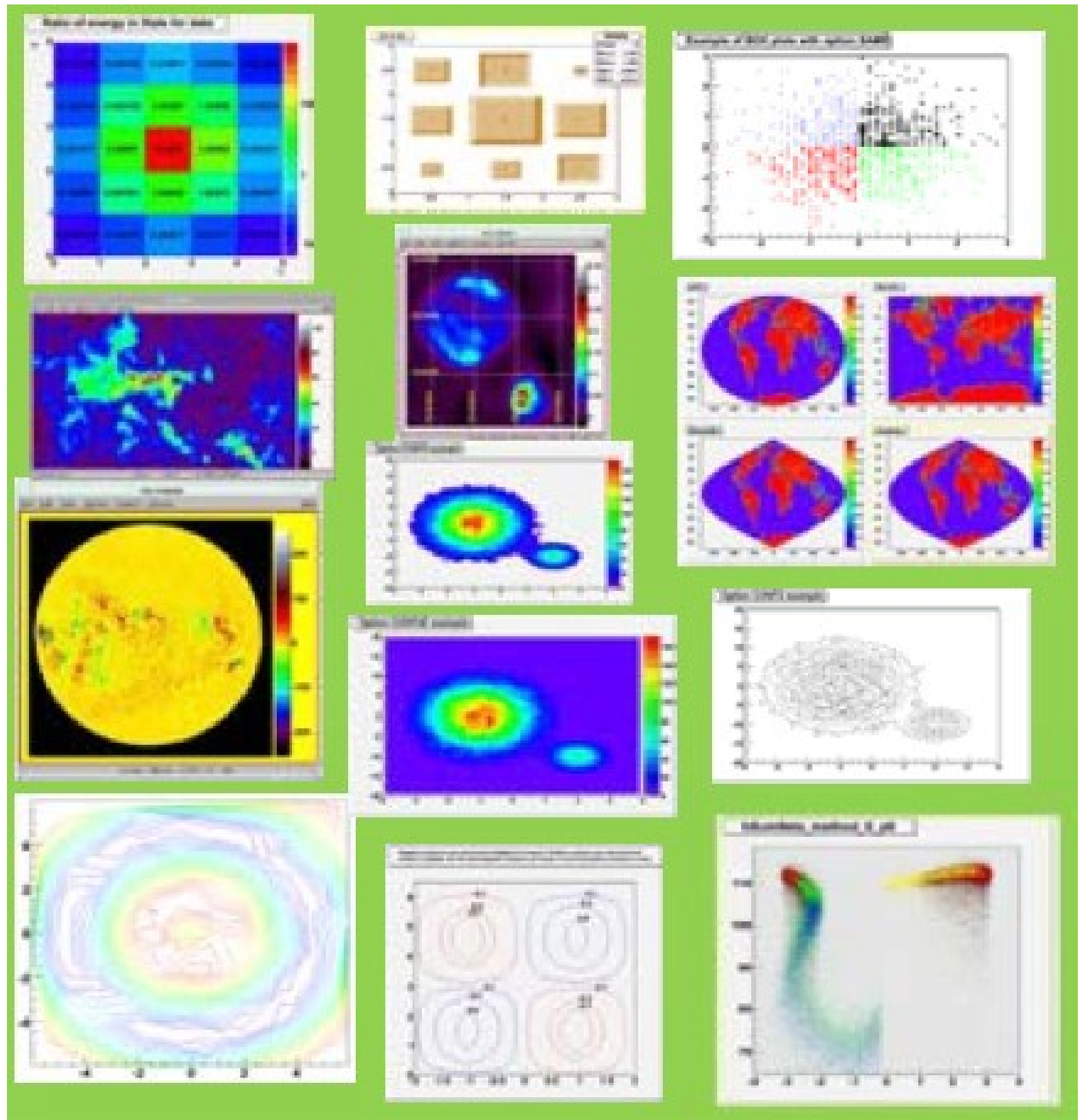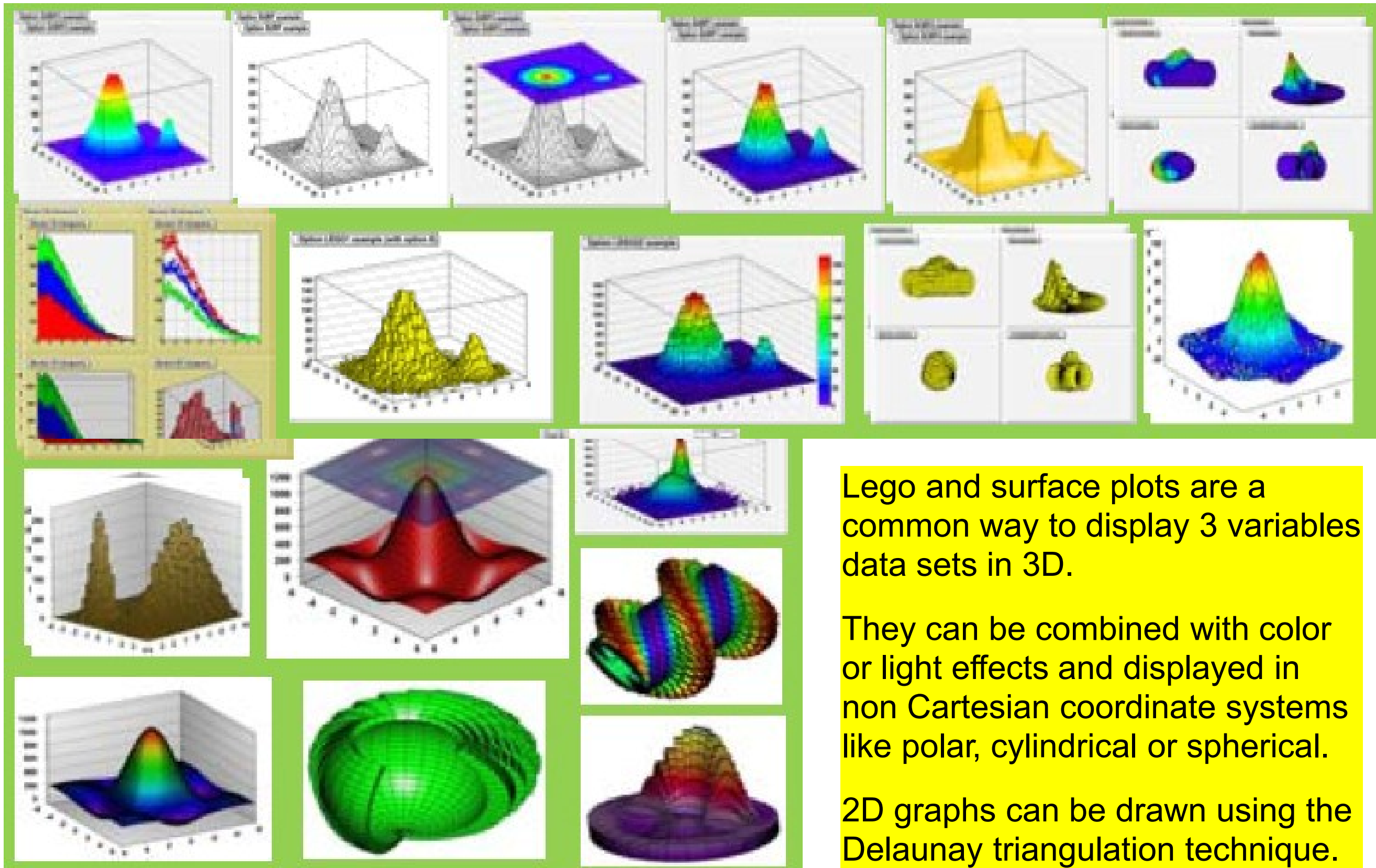
Lego and surface plots are a common way to display 3 variables data sets in 3D.

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.
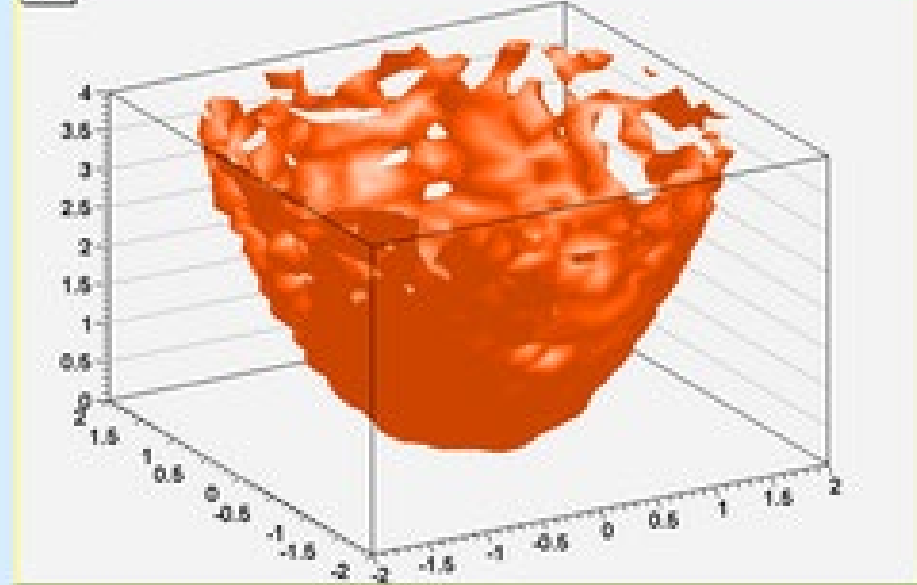
4 Variables Techniques (1/2)

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.
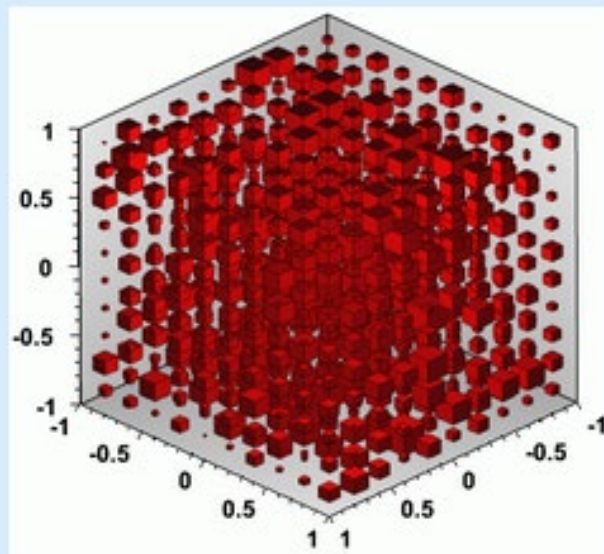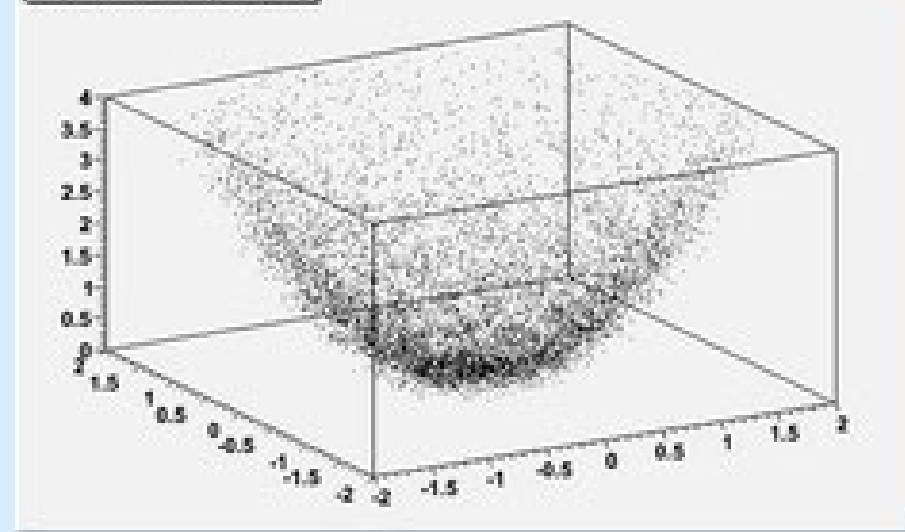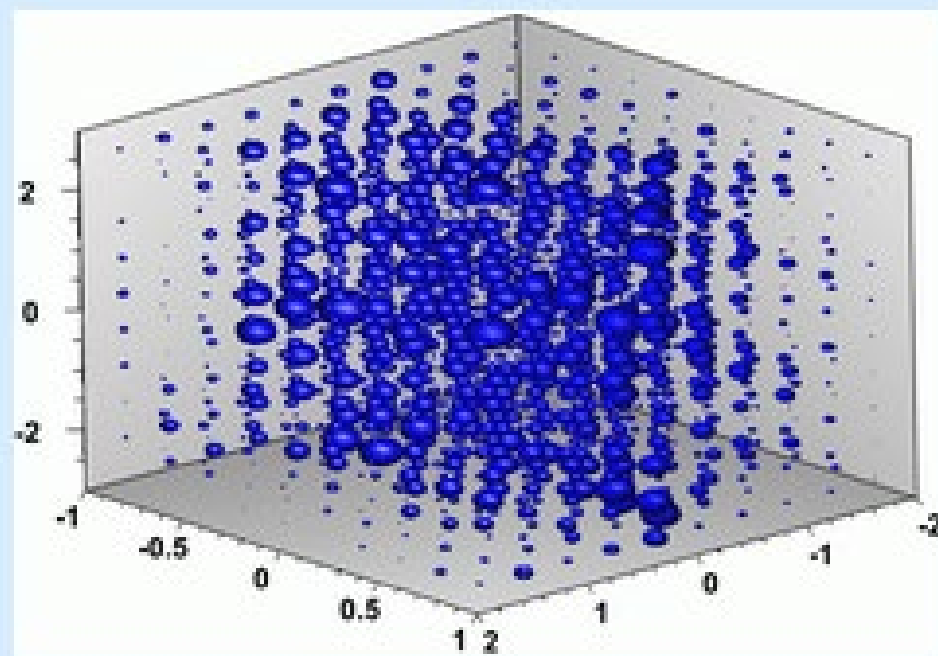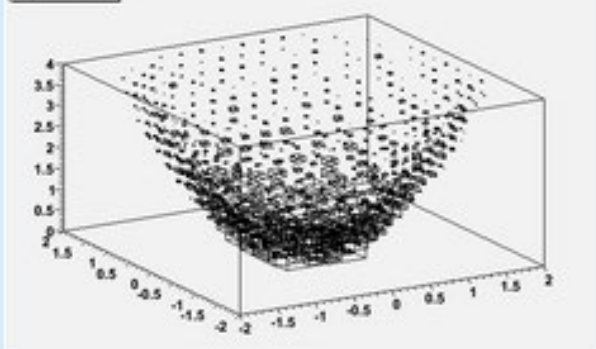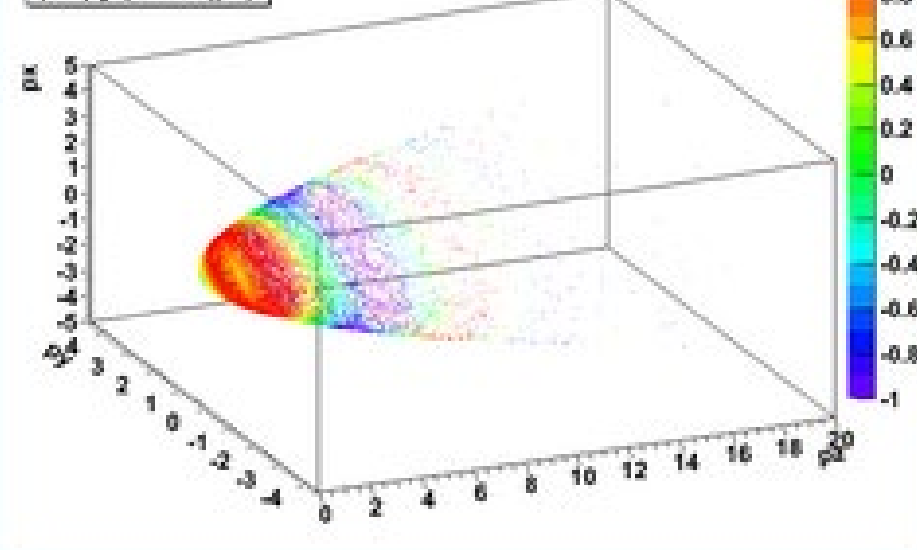
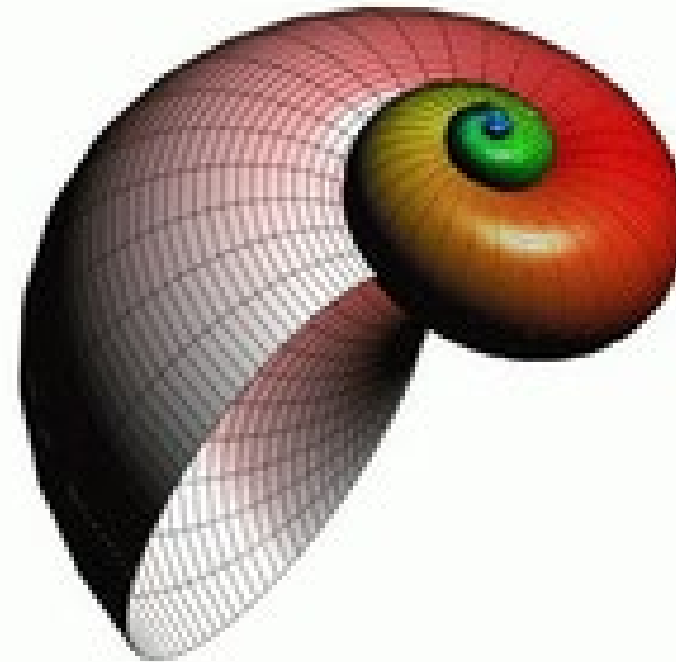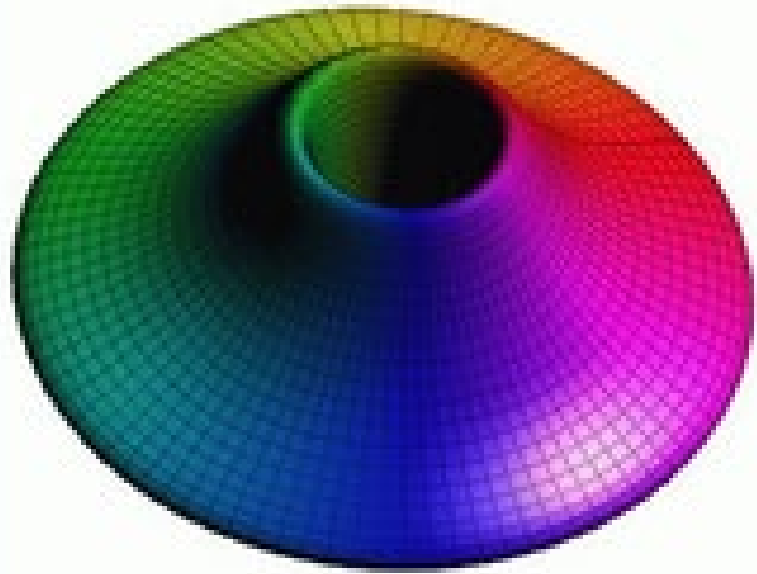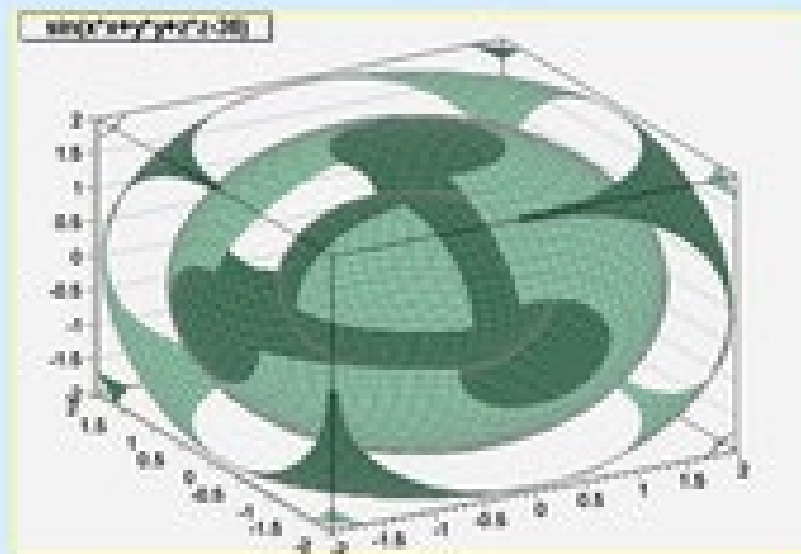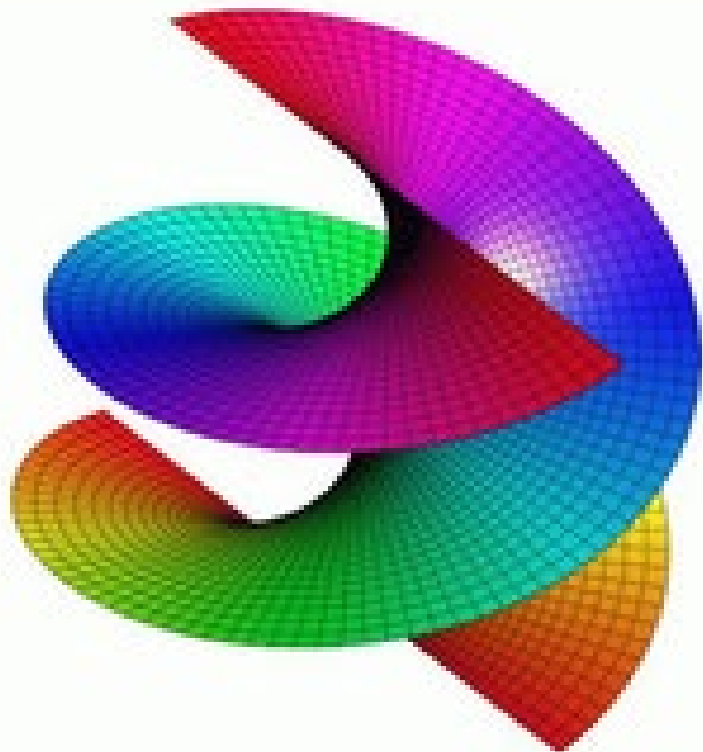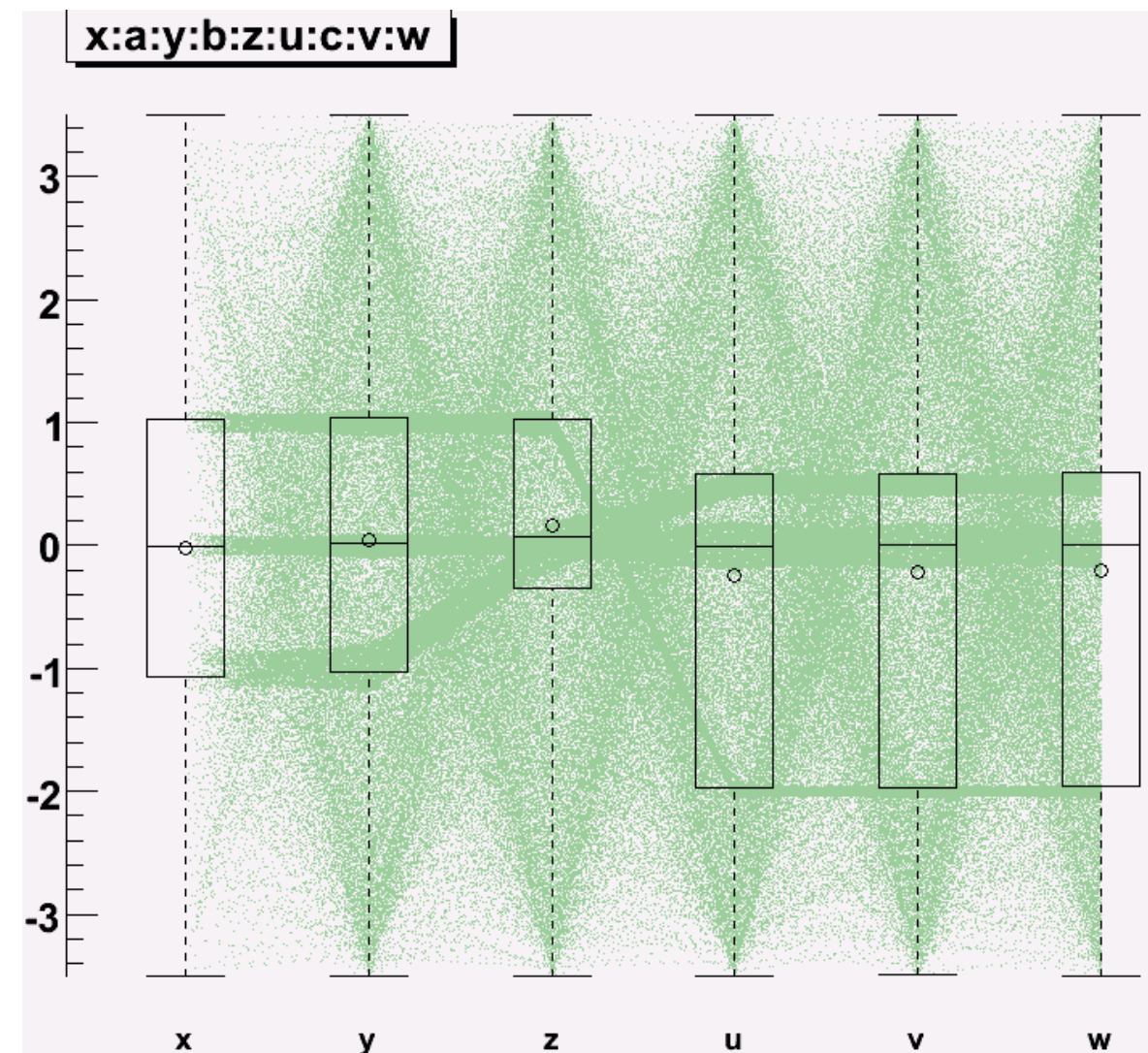Functions like t= f(x,y,z) and 3D histograms are 4 variables objects.
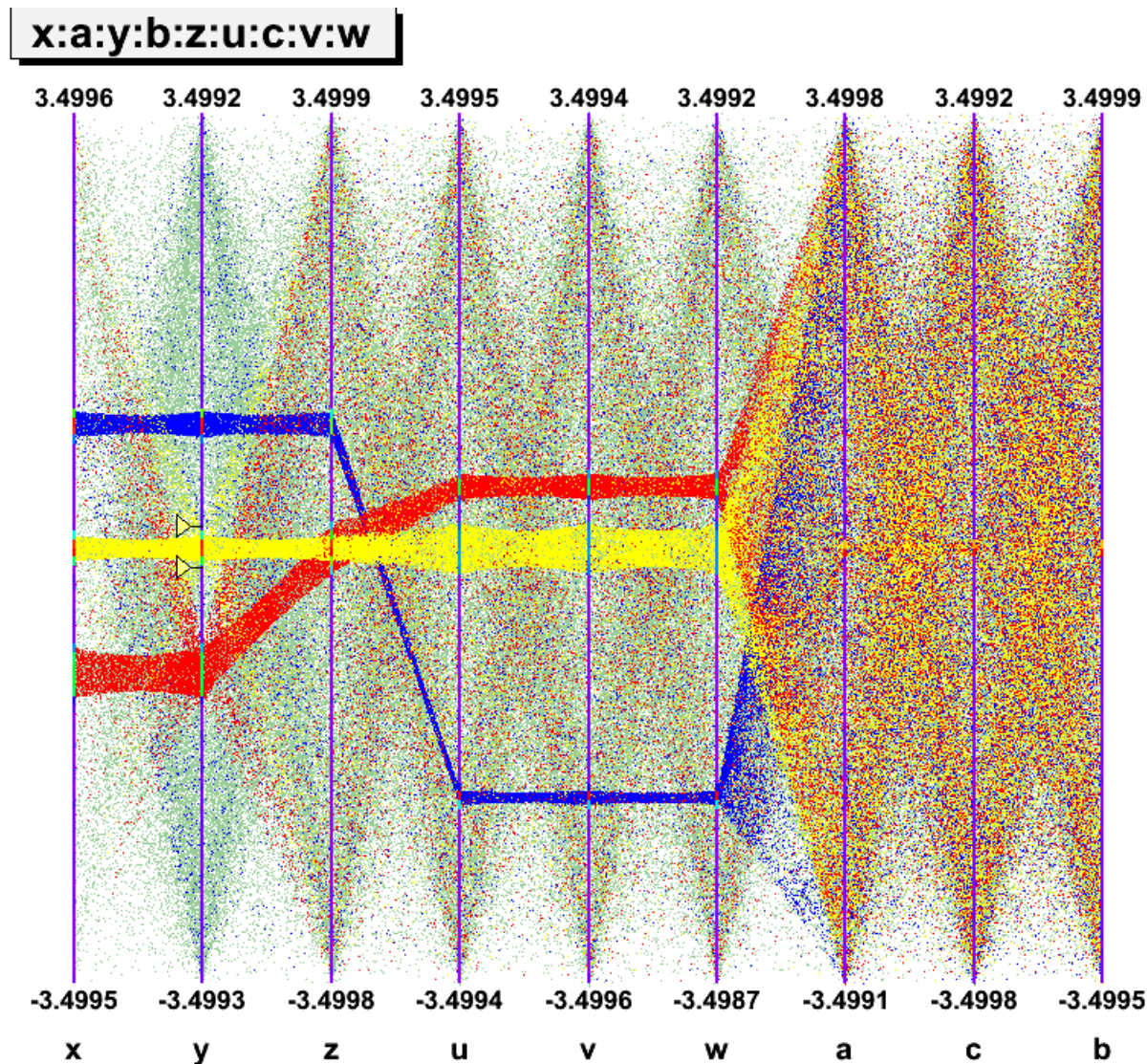
ROOT can render using OpenGL.

It allows to enhance the plots' quality and the interactivity.

Cutting planes, projection and zoom allow to better understand the data set or function.

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.
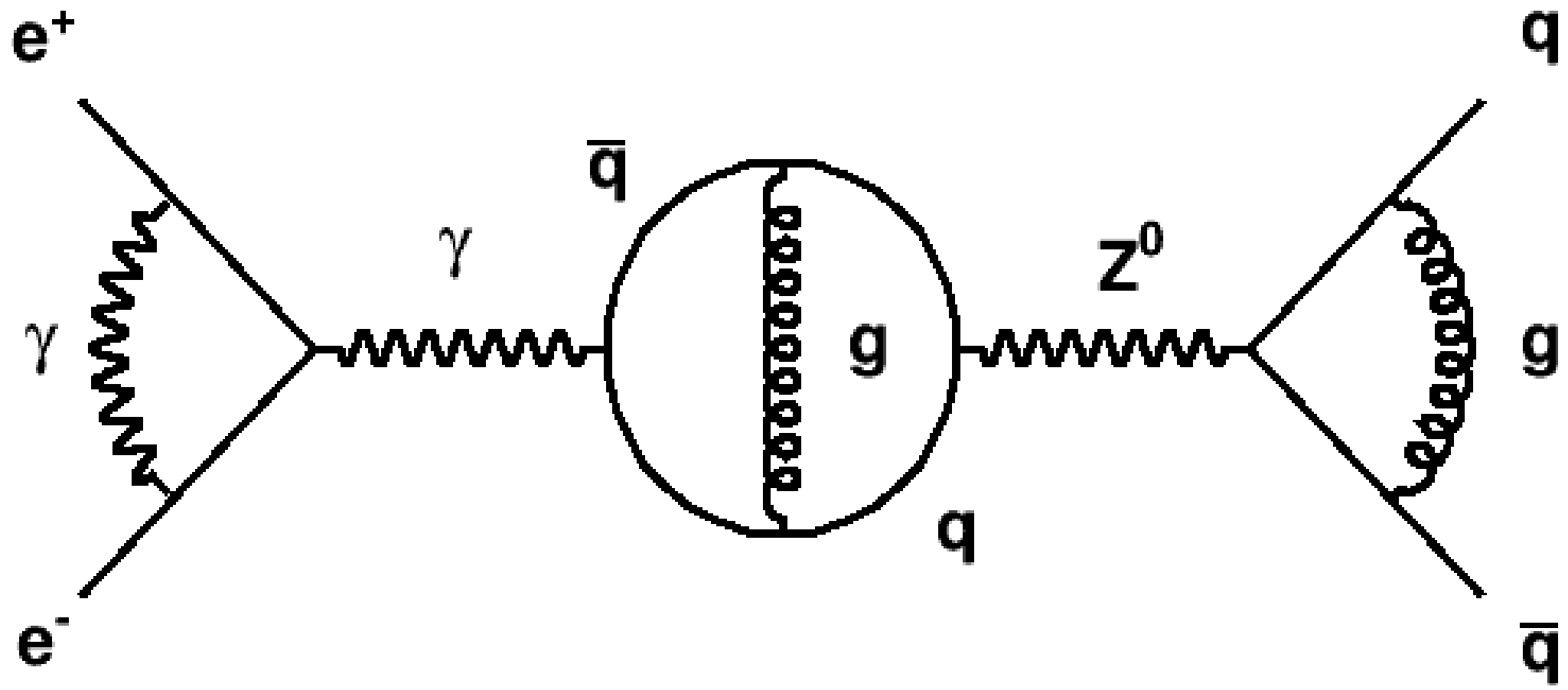
$ROOTSYS/tutorials/graphics/feynman.C

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

Dig $ROOTSYS/tutorials/hist

- Run the example macros

- Try to understand the codes of the example macros:

    references: *http://root.cern.ch/drupal/content/reference-guide*

    class header descriptions

- "*grep*" those macros to find what you need