ROOT Graphics: Status and Future Plans

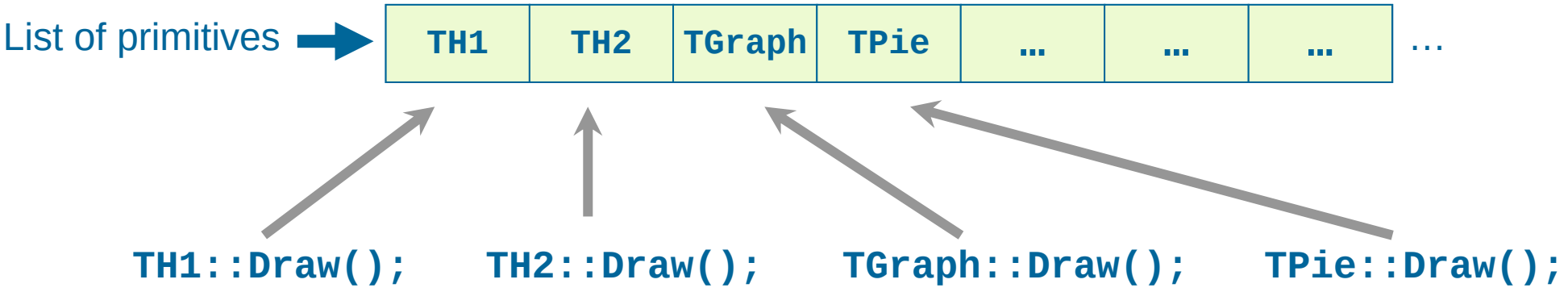
Olivier Couet (CERN)

Overview

- Basic Elements of ROOT Graphics.
- 2D Graphics Latest Developments.
- 3D Graphics Latest Developments.
- Multiple Variable Data Set Visualization.
  - Spider (Radar) Plots.
  - Parallel Coordinates Plots.
  - Box (Candle) Plots.
- Conclusion.

Basic Elements (1/5)

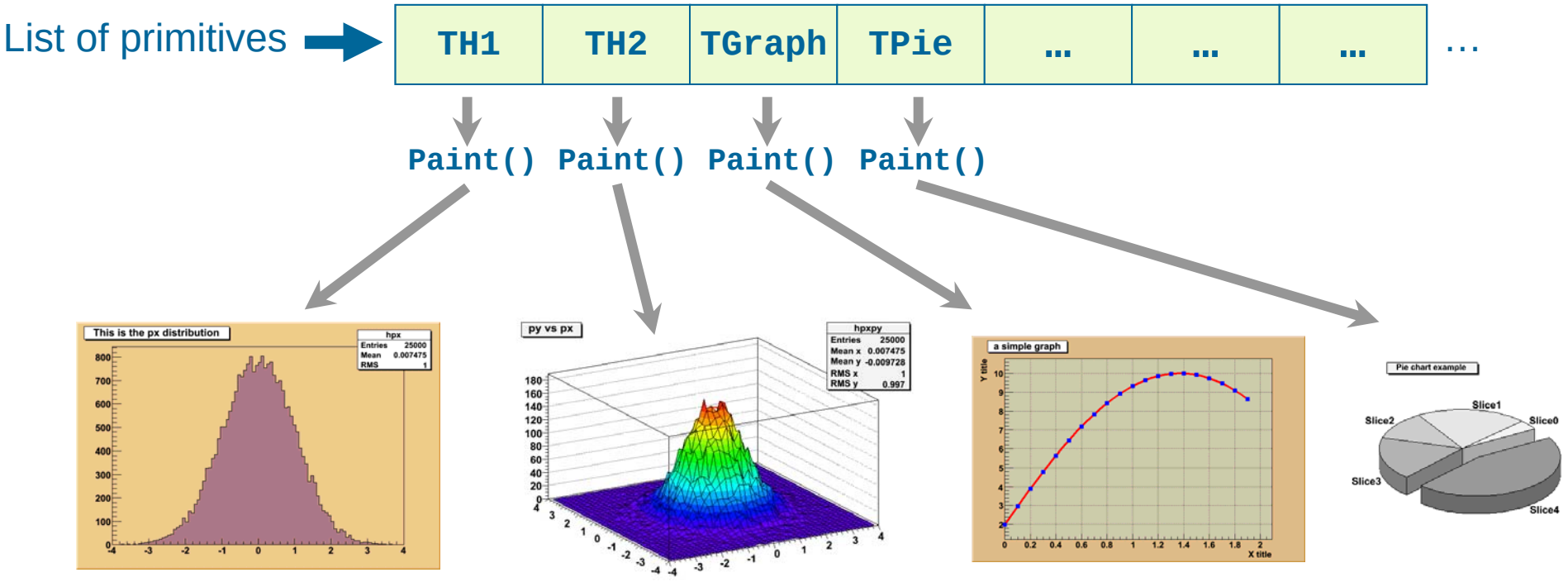
- The ROOT graphics is build around the pad concept (class **`TPad`**).
- A pad is a linked **list of primitives** of any type (graphs, histograms, shapes, tracks, etc.). It is a kind of “display list”.



- Adding an element into a pad is done by the **`Draw()`** method of each classes.

Basic Elements (2/5)

- A pad is painted by calling sequentially the **`Paint()`** method of each object in the **list of primitives.**



Basic Elements (3/5)

- The **list of primitives** is also used to pick objects and to interact with them.

Is `DistancetoPrimitive(px,py)` small enough ?

Is `DistancetoPrimitive(px,py)` small enough ? Yes ! …

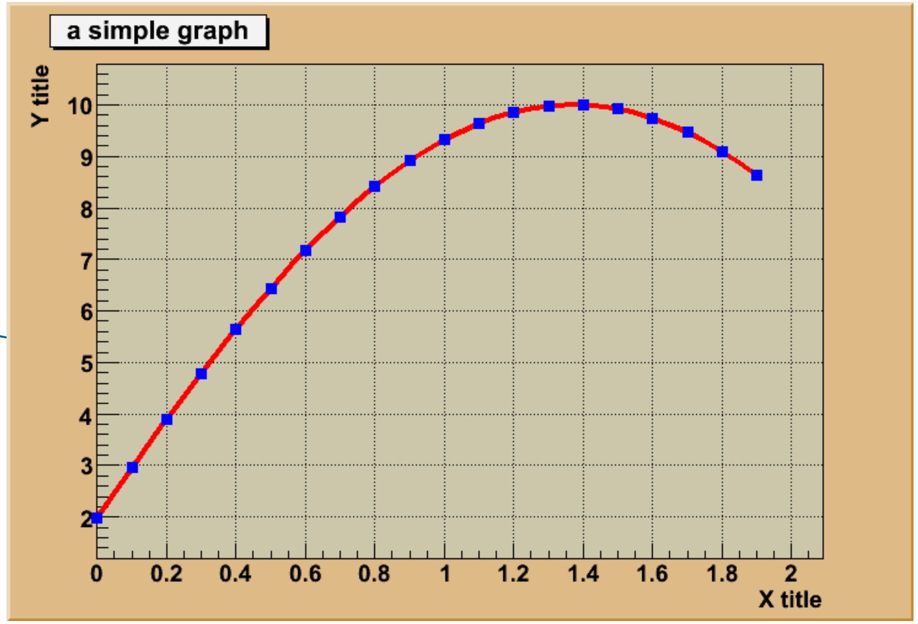
List of primitives → **TH1** | **TH2** | **TGraph** | **TPie** | **…** | **…** | **…** …

1. Picking:

For each object in the **list of primitives** the method **`DistancetoPrimitive(px,py)`** is called with the (px,py) cursor position (pixel). If the distance is small enough the object is considered as picked.

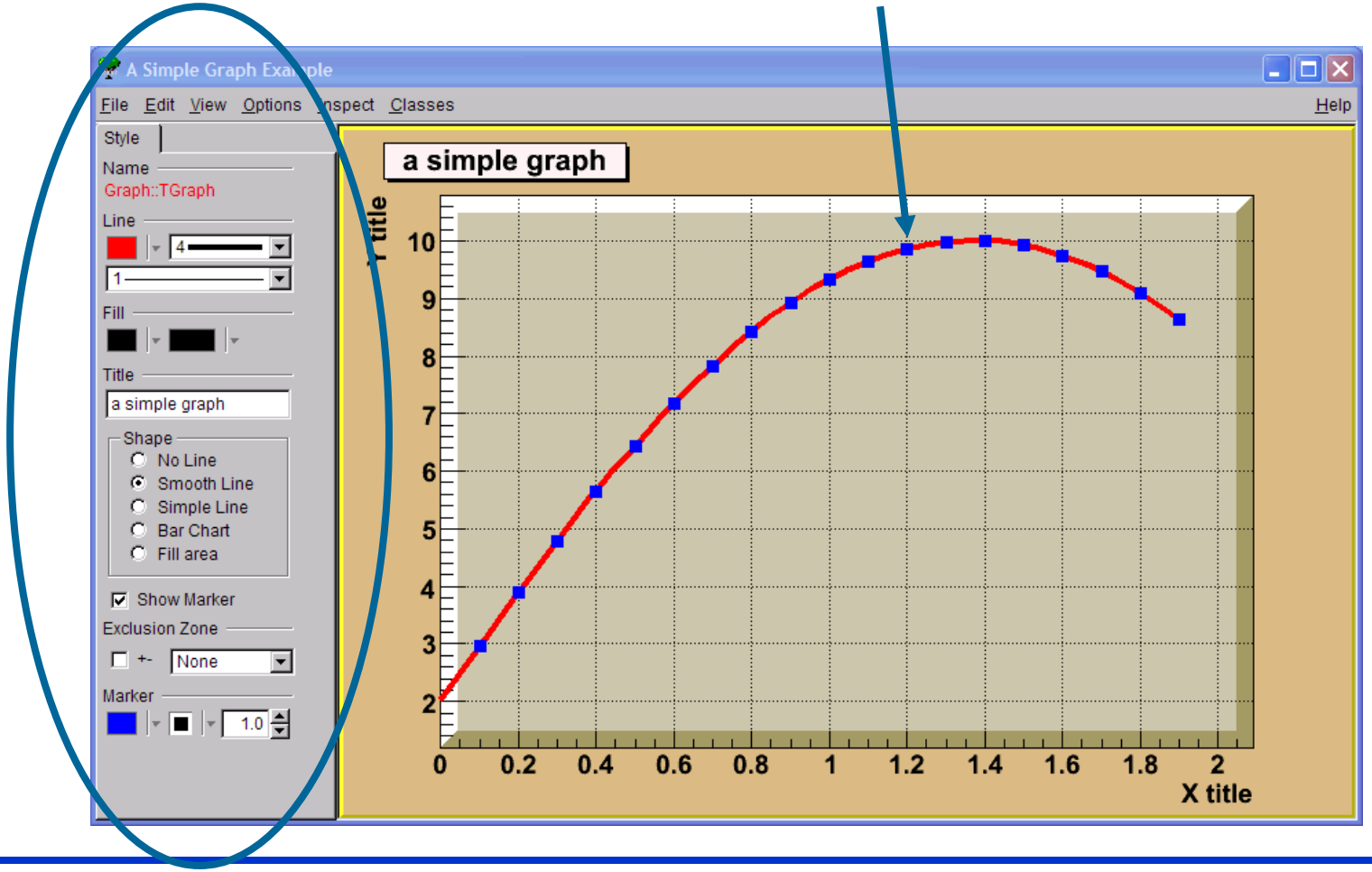
2. Interaction:

Once an object has been picked. It is possible to act on it thanks to the method **`ExecuteEvent(px,py)`** automatically called when the object is picked.



Basic Elements (4/5)

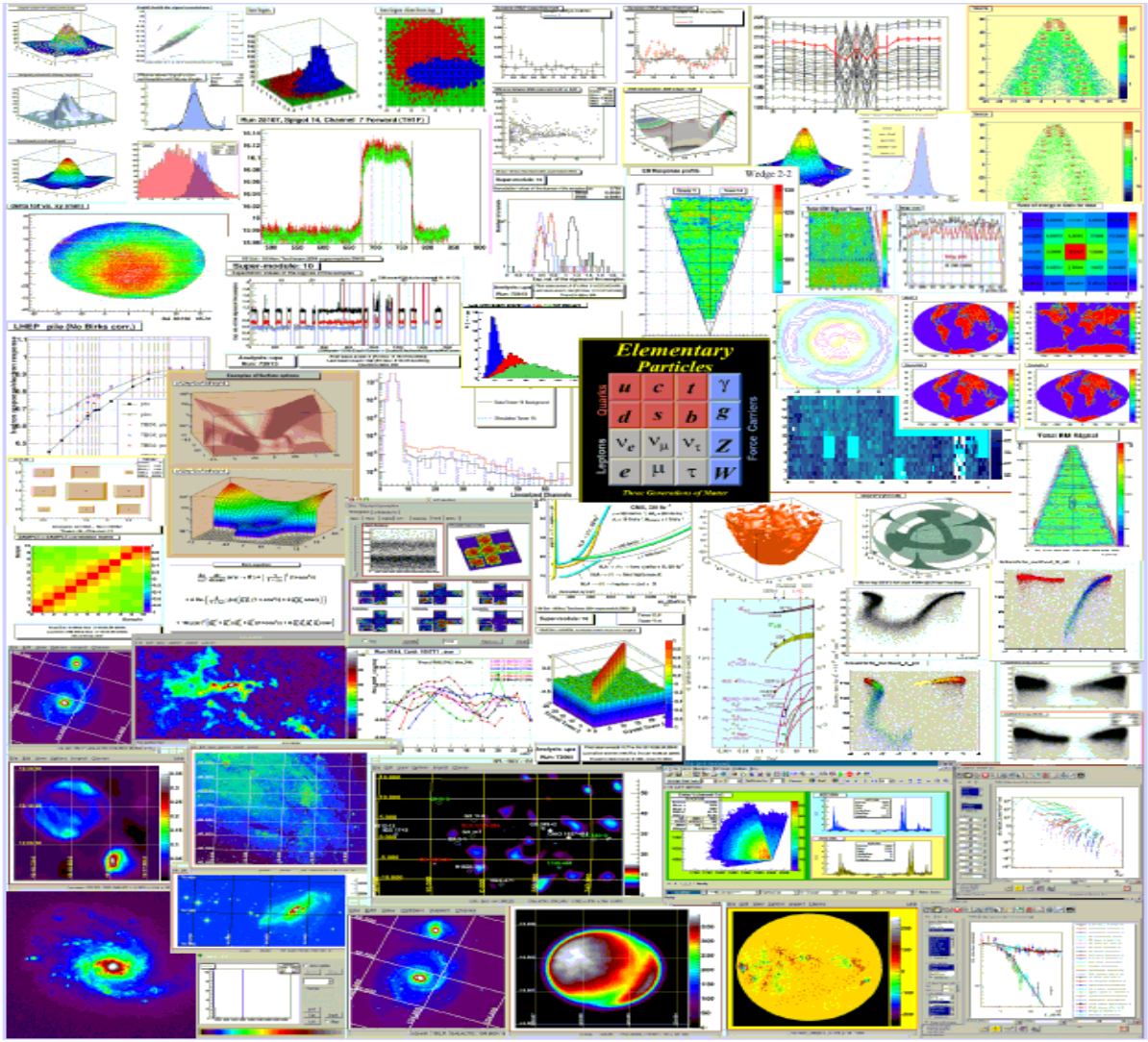
- For each element in the **list of primitives**, it is possible to access a dedicated **editor** allowing to perform specialized actions on the **selected object**.



Basic Elements (5/5)

Based on this mechanism it is possible to produce all kinds of plots and to interact with them.

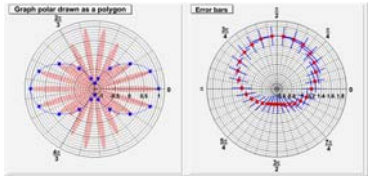
The new developments we will present later make a great use of these features.



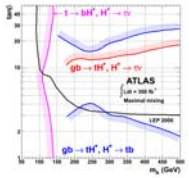
2D Graphics Latest Developments

The next slides present the latest significant 2D graphics' **developments** (last 18 months):

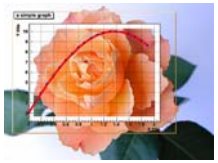
Polar Graphs



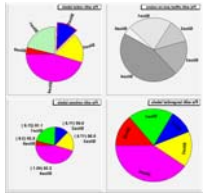
Exclusion Plots



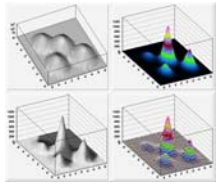
TASImage/TImage/TImageDump



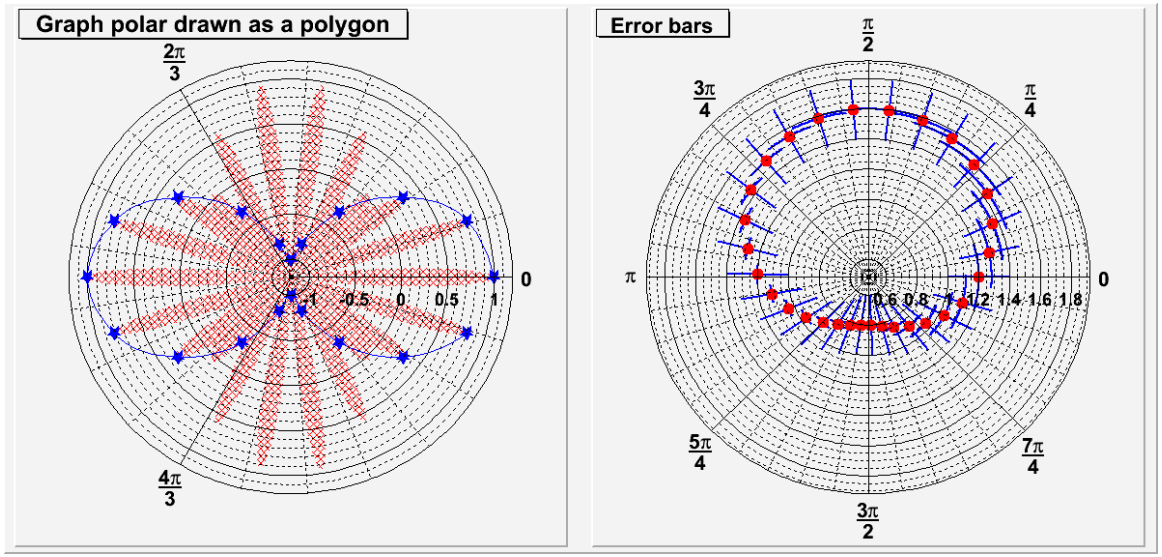
Pie Charts



Spectrum Painter



TPolarGraph



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

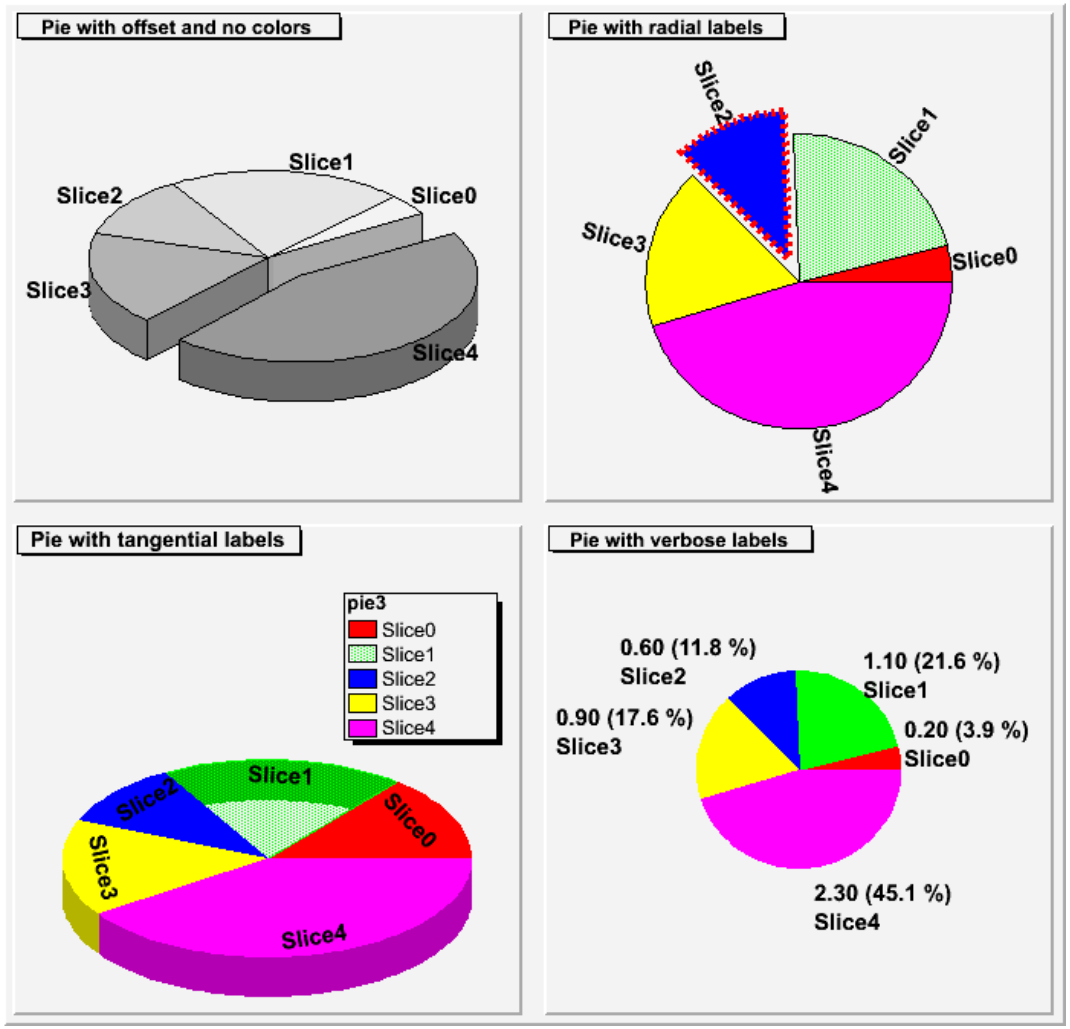
It creates a **polar graph** (including error bars).

- **`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...
- Still needs some developments (Editor).

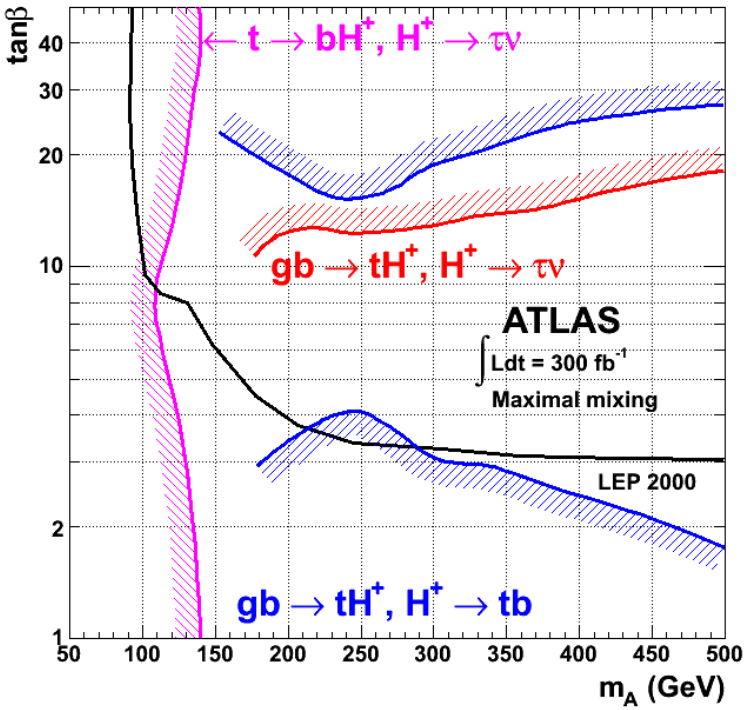
TPie

This class was developed in collaboration with *G.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`**.



Exclusion Plots



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 using **`SetLineWidth`**) is greater than 99.

In that case the line width number is interpreted as:

`100*ff+ll = ffll`

- **`ll`** represents the **normal line width (`ll>0`)**.
- **`ff`** is the **filled area width (`ff>0`)**.
- 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.

Spectrum Painter

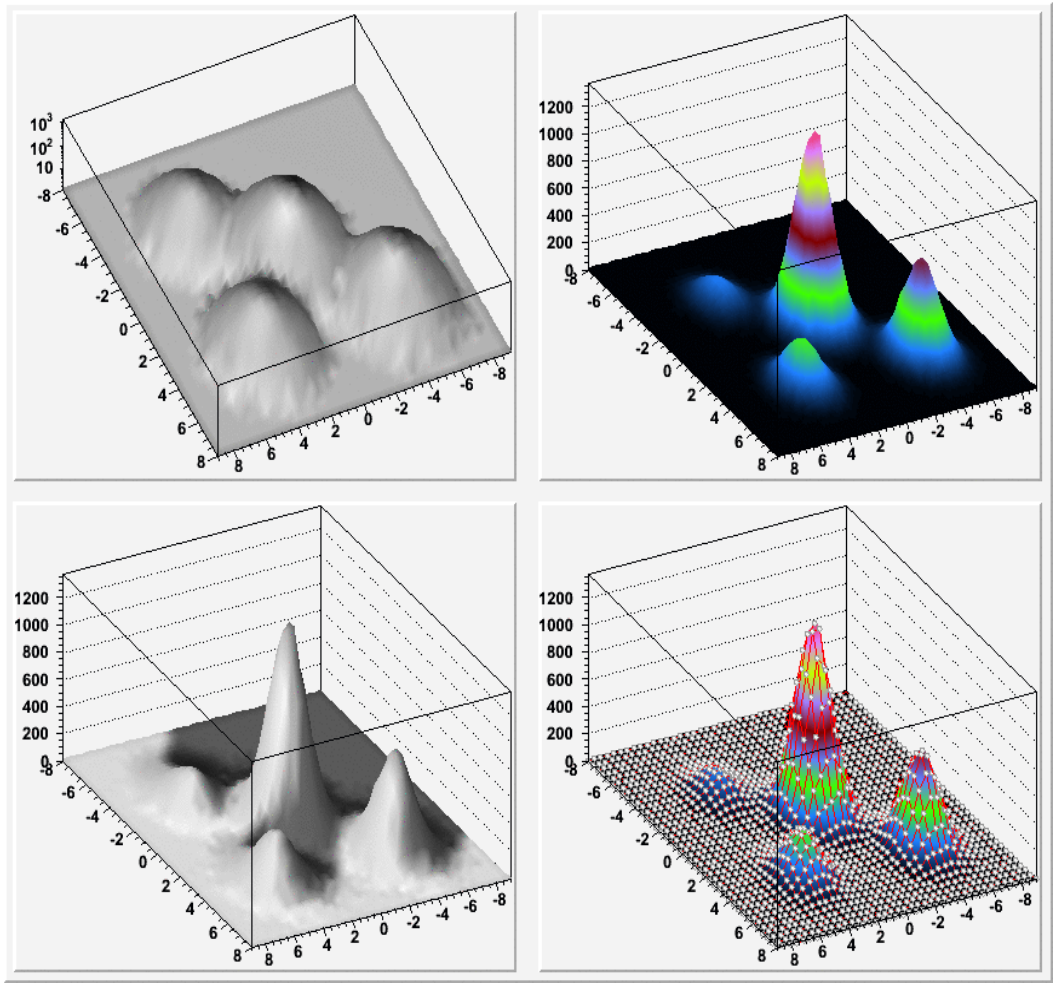
The “Spectrum Painter” is a **surface and contour drawing package**. It has been developed by *Miroslav Morhac* several years ago in C and has been transformed in a **ROOT** class.

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.

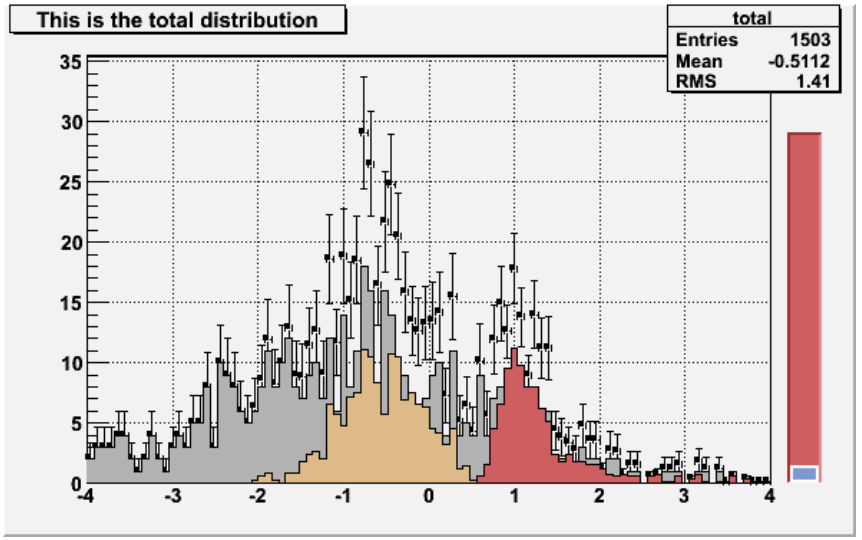
A complete description of its capabilities is given in the help of: **`TSpectrum2Paint::PaintSpectrum()`**

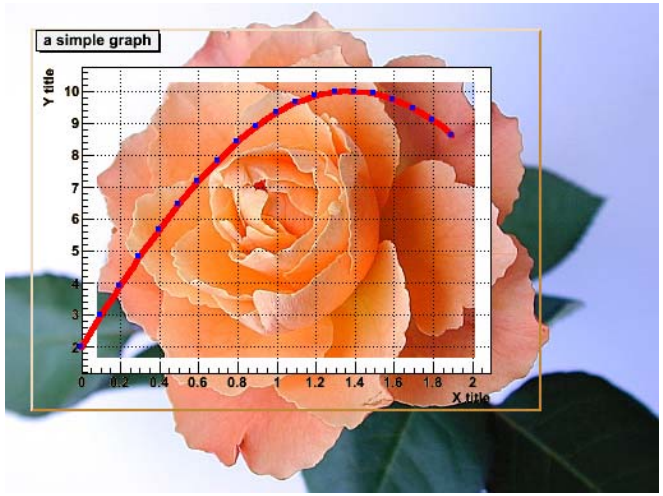


TASImage/TImage/TImageDump

Developments done by *V.Onuchin.*

• 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 then 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.

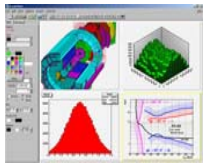
3D Graphics Latest Developments

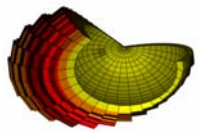
The 3D Graphics GL developments have been done to have:

1. The possibility to draw graphics produced by GL in a 2D pad (**GL-in-Pad**).
2. Data set representations using GL.
3. The **GL-Viewer** (see *M.Tadel* talk.)

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

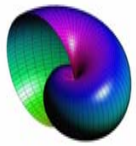
GL in Pad: 2D and 3D mixed

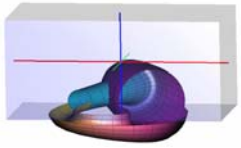




TH2, TF2, TH3, TF3 representations

Parametric Functions

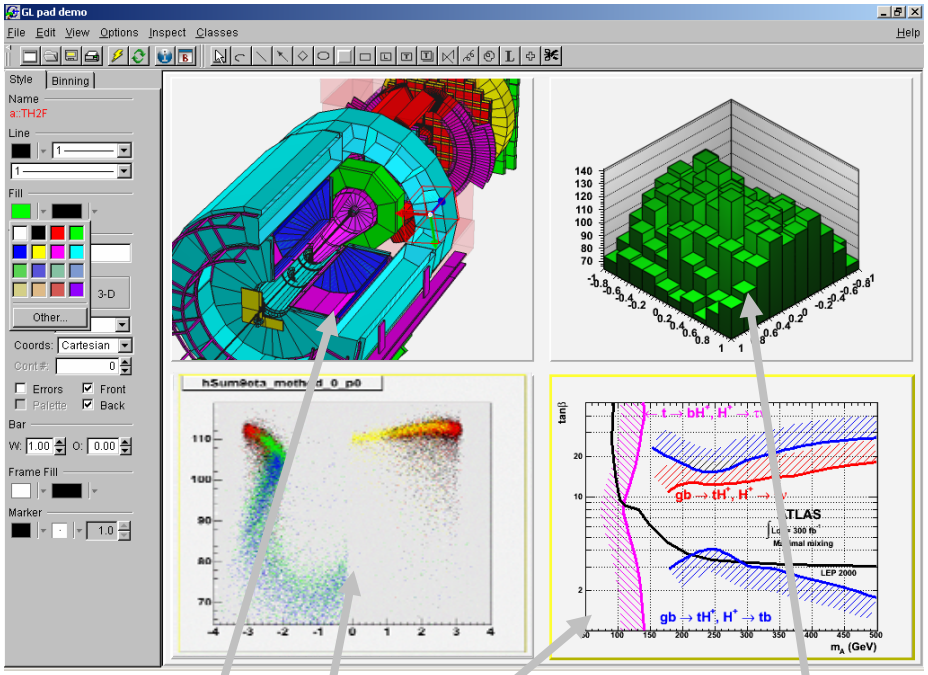




3D Interactions

All the **GL-in-Pad** developments are done by *T.Pocheptsov.*

GL in Pad: 2D and 3D Mixed



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

```
TCanvas c("glc","GL pad demo");
```

or to do:

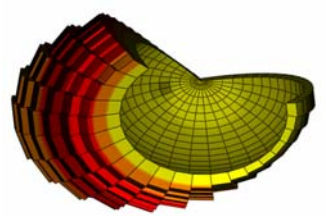
```
gStyle->SetCanvasPreferGL(kTRUE);
```

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).

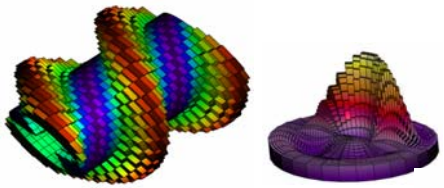
TH2, TF2, TH3, TF3 Representations

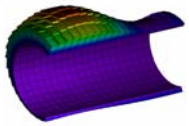


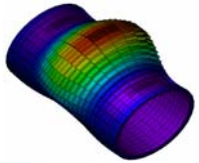
It was already possible to display **TH2**, **TH3**, **TF2**, **TF3** using combination of options like **"SURF","LEGO","POL","CYL"**, etc …

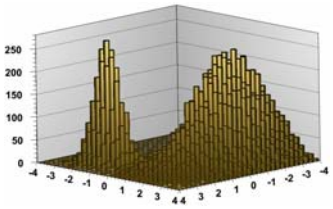
The same options are available in "GL-mode".
It is enough to add the option **"gl"** in the list of options, and to draw in a canvas which "prefer gl"
Example:

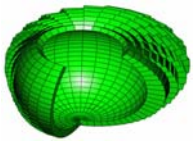
```
h2->Draw("gl surf1 cyl");
```



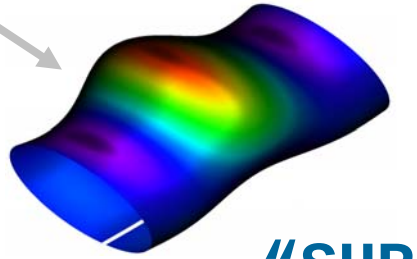




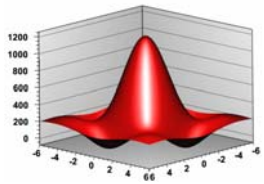


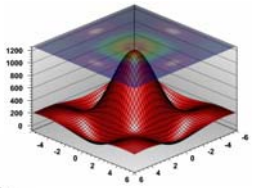


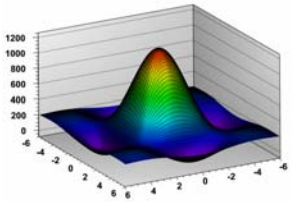
"LEGO"

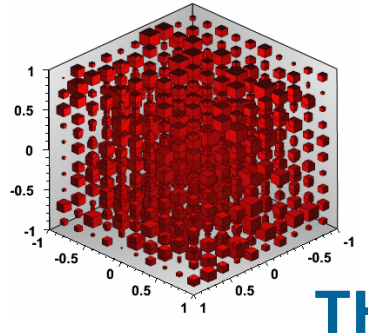


"SURF"

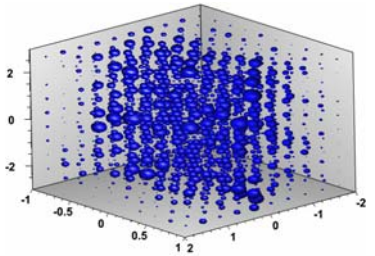


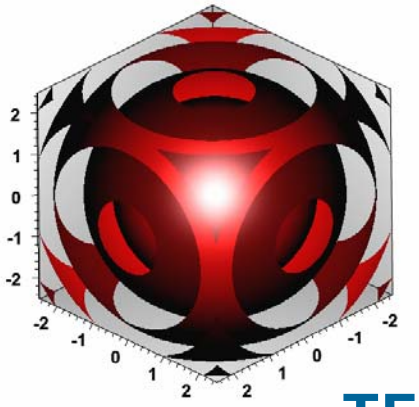






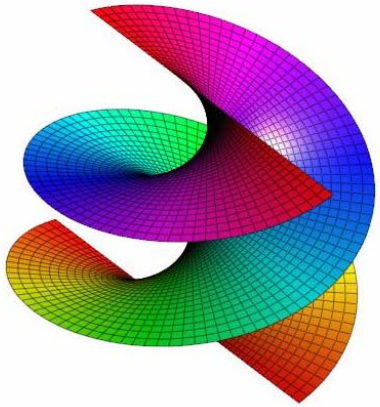
TH3





TF3

Parametric Functions

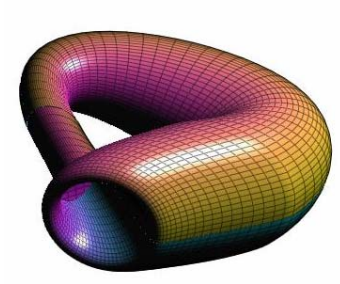


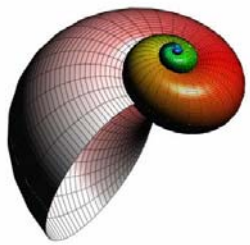
A parametric surface is defined by three functions:

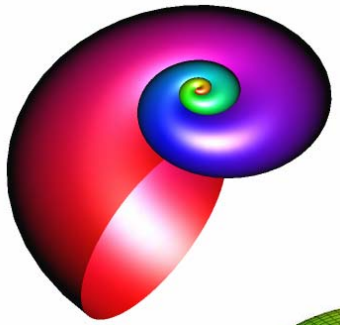
```
S(u,v) : {x(u,v), y(u,v), z(u,v)}.
```

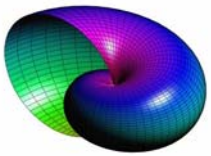
Example, a conchoid:

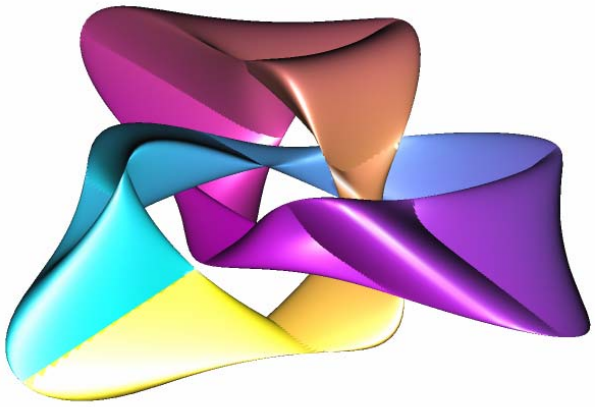
```
TGLParametricEquation p1("Conchoid",
                         "1.2^u*(1+cos(v))*cos(u)",
                         "1.2^u*(1+cos(v))*sin(u)",
                         "1.2^u*sin(v)-1.5*1.2^u",
                         0., 6*TMath::Pi(), 0., TMath::TwoPi());
```

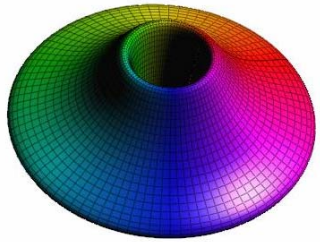


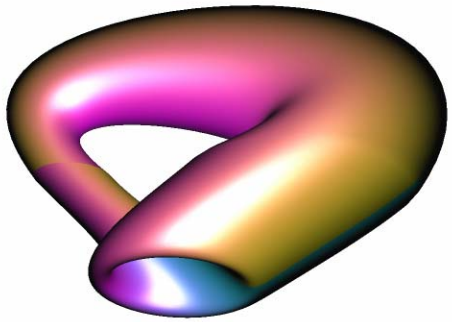












Parametric surfaces can be drawn using a GL-pad

Iso-Surfaces

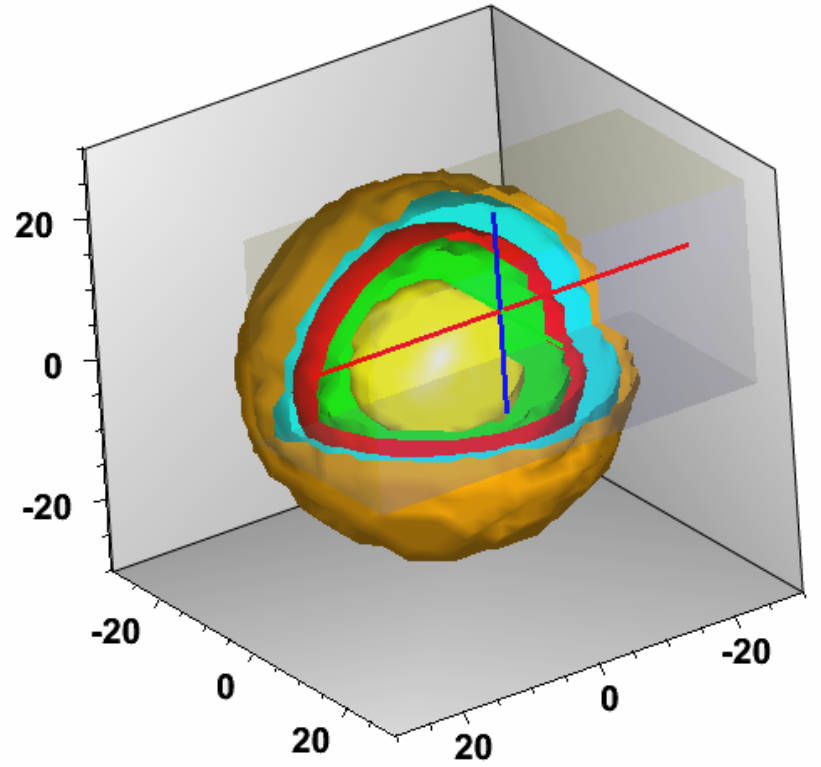
It is possible to paint a **TH3** histogram using iso-surfaces (iso-3D contours).

It can be seen as a **"3D contour plots".**

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

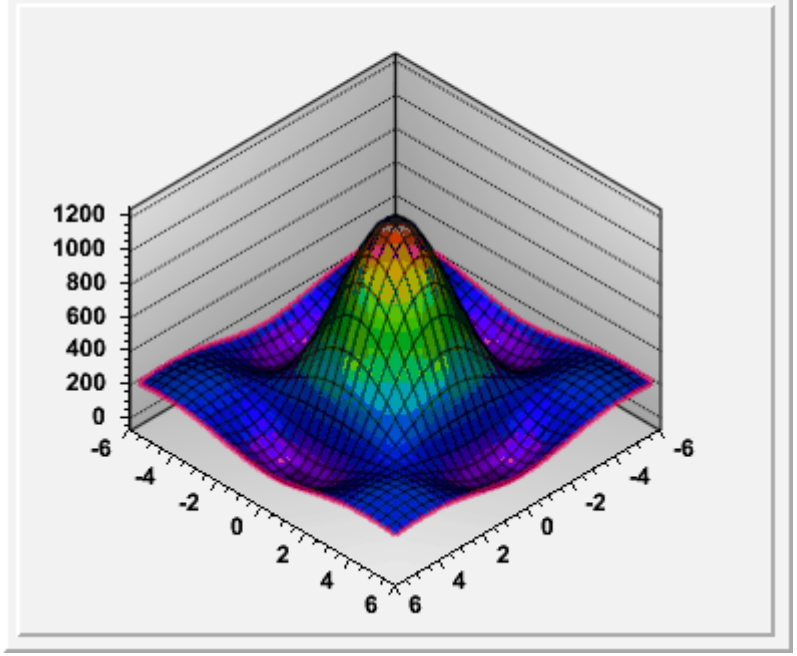
```
void iso()
{
   gStyle->SetCanvasPreferGL(true);
   TH3C * volume = new TH3C("V","V",
                   20,-20,20, 20,-20,20, 20,-20,20);
   TCanvas * canvas = new TCanvas("C","C",600,600);
   float x,y,z;

   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");
}
```

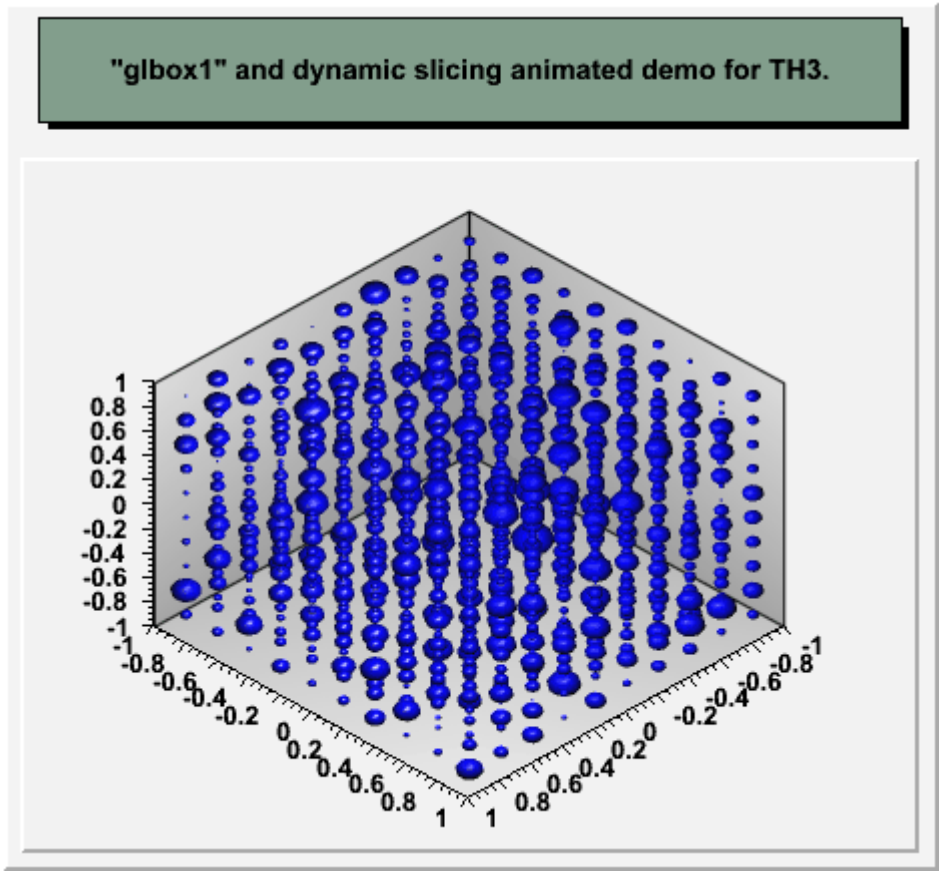


3D Interactions

"glsurf1" and dynamic slicing animated demo for TF2.

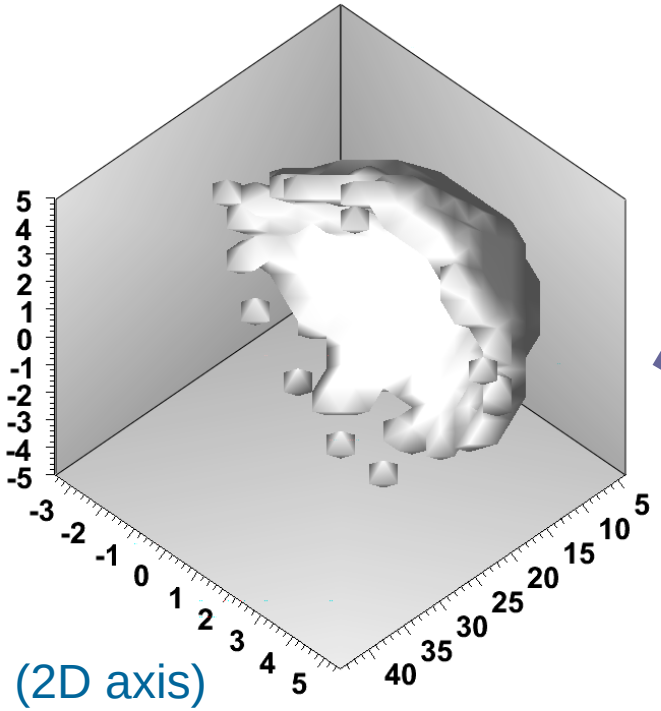


"glbox1" and dynamic slicing animated demo for TH3.

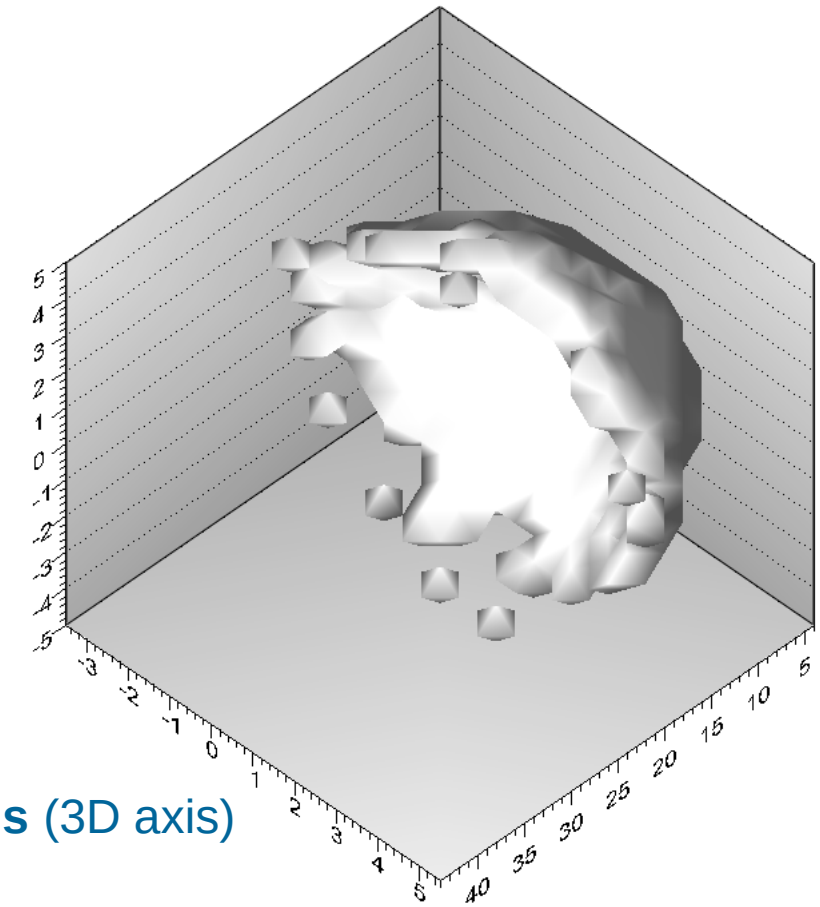


TGLText / TGLAxis

- **`TGLAxis`** uses **`TGLText`** which is based on the domain public library **FTGL**.
- **FTGL** allows to render ***freetype*** text in 3D using OpenGL.



`TGaxis` (2D axis)



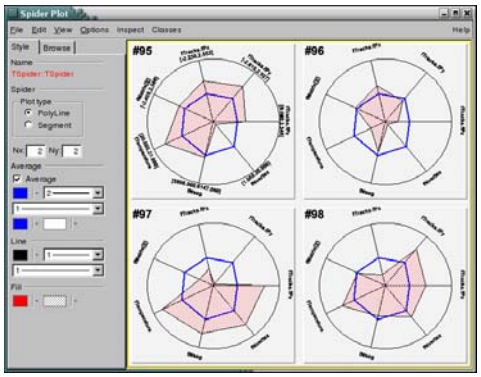
`TGLAxis` (3D axis)

Multiple Variables Visualization

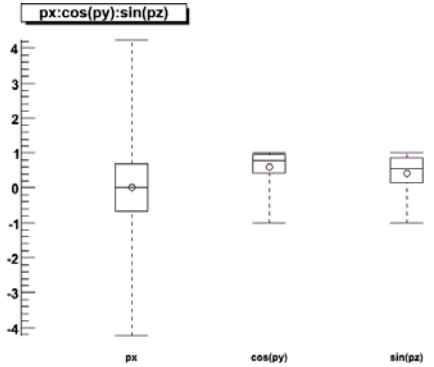
Because there was no way to visualize more than 4 variables of a **ROOT** tree, three new visualization techniques have implement last summer.

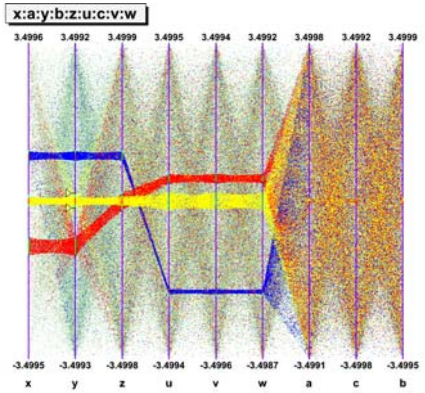
This work has been done with *B.Dalla Piazza.*

Scanning with Spider plot.



Box plots.



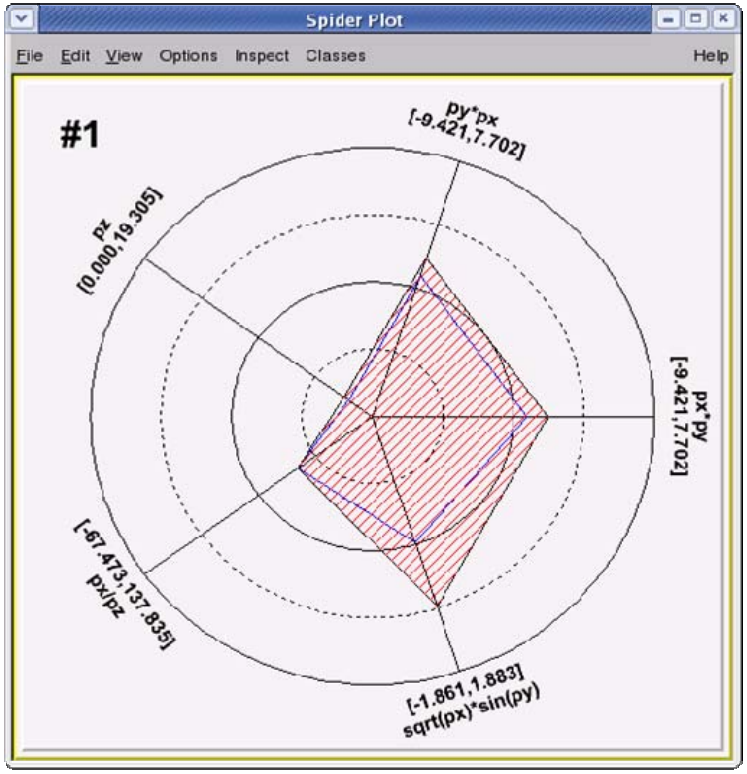


Parallel Coordinates plots.

Spider Plots (1/2)

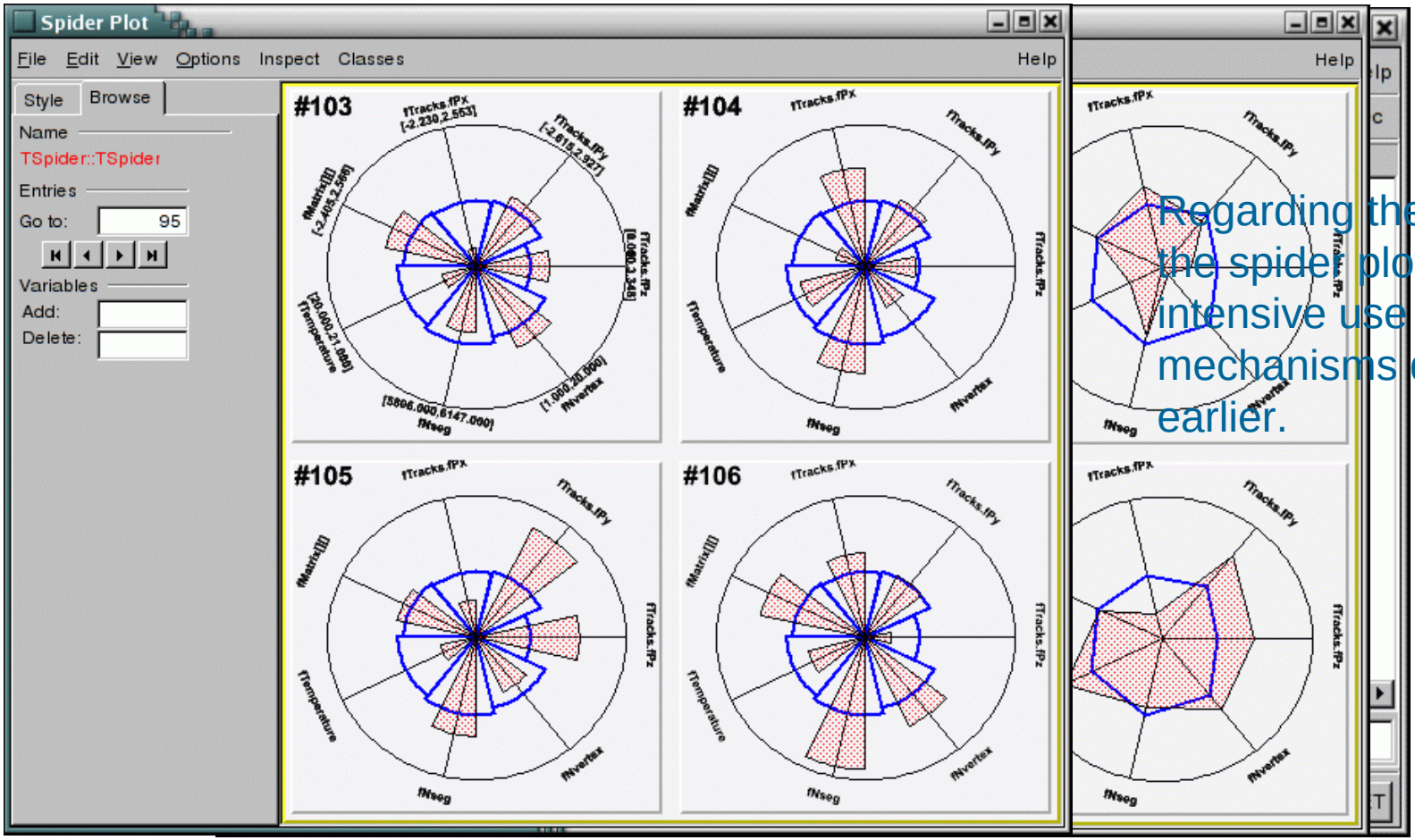
- **Spider plots** (sometimes called a "**web-plots**" or "**radar plots**") are used to **compare series of data points (events).** They use the human ability to **spot asymmetry**.

• Variables are represented on **individual axes** displayed **along a circle**.
• For each variable the **minimum value sits on the circle's center**, and the **maximum on the circle's radius**.
• Spider plots are not suitable for an accurate reading from the graph since, by it's nature, it can be difficult to read out very detailed values, but **gives quickly a global view of an event in order to compare it with the others**.



Spider (Radar) Plots (2/2)

- The tab Style allows to define the Spider Plot attributes.
- The tab Browse allows to navigate in the Tree.
  1) Select the Tree Viewer.
  2) Select the Spider Plot Viewer.



Regarding the interactivity, the spider plot make intensive use of the Pad mechanisms described earlier.

Parallel Coordinates Plots (1/13)

- The multidimensional system of **Parallel Coordinates Plots** (||-Coord) is a common way of **studying** and **visualizing multivariate data sets**. They were proposed by in A.Inselberg in 1981 as a new way to represent multi-dimensional information.

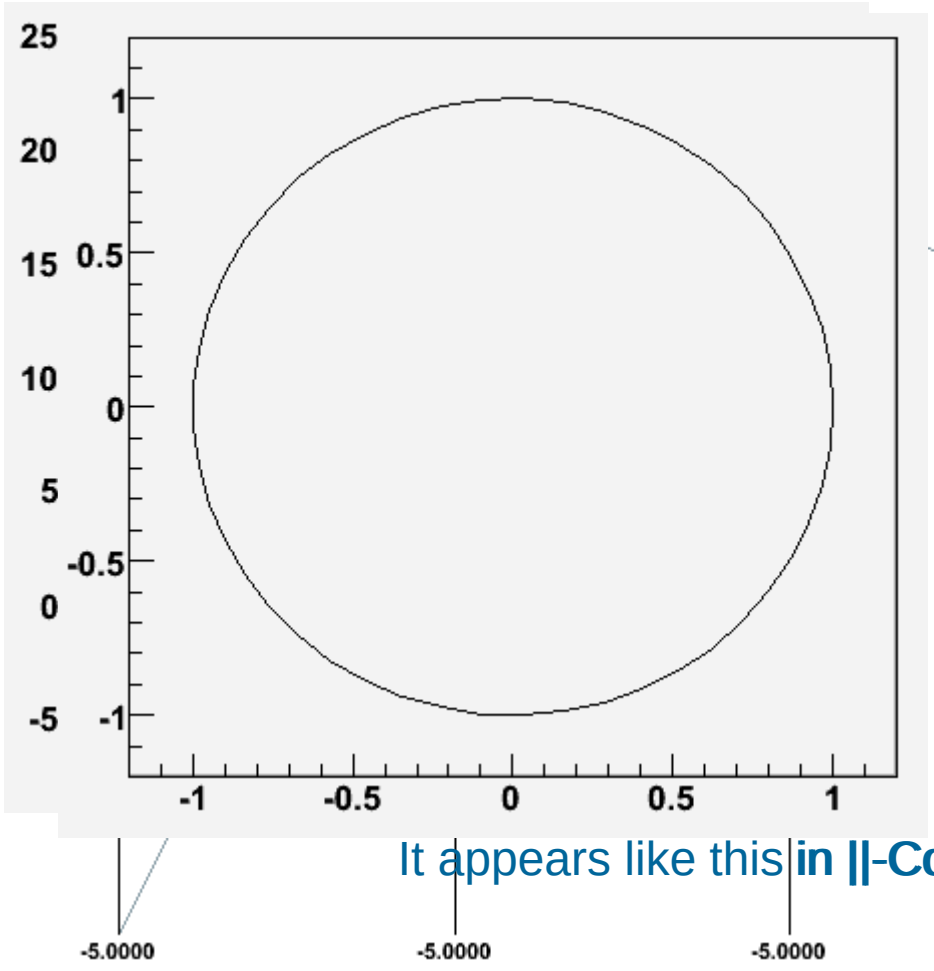
- In traditional Cartesian coordinates, axes are mutually perpendicular. In Parallel coordinates, **all axes are parallel** which allows to represent data in **much more than 3 dimensions.**

- **A point in n-dimensional space** is represented as a **polyline** with vertices on the parallel axes. The position of the vertex on the *i-th* axis corresponds to the *i-th* coordinate of the point.

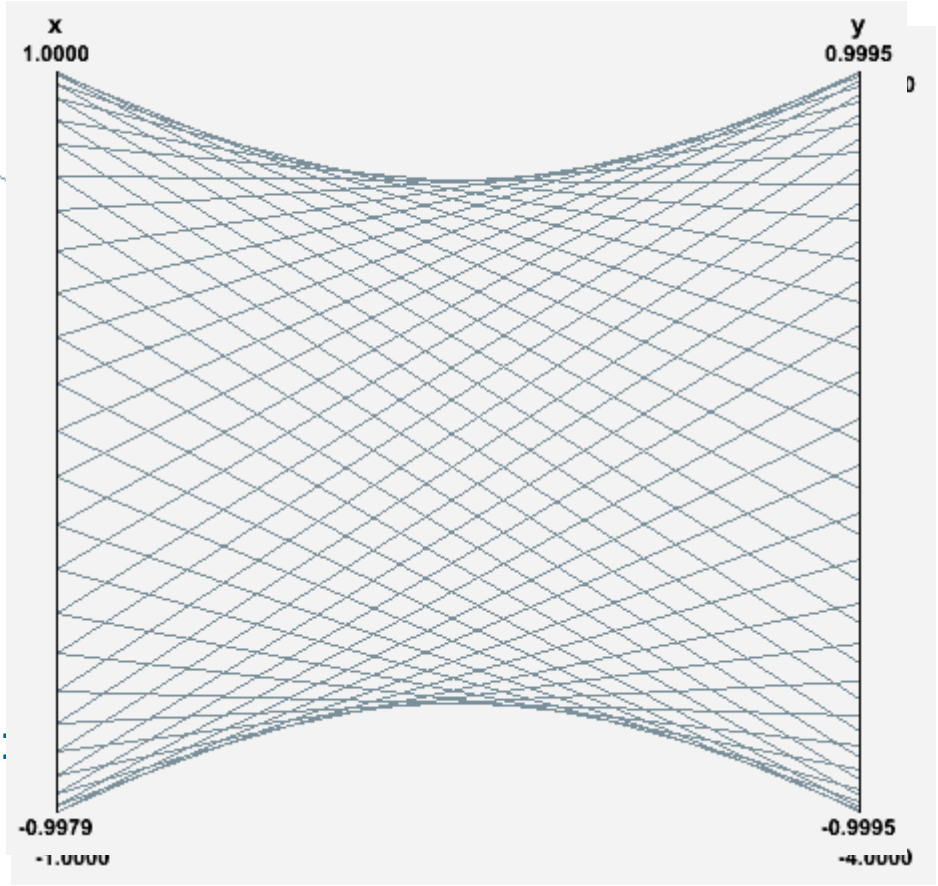
- To show a set of points in ||-Coord, a set of parallel lines is drawn, typically vertical and equally spaced.

Parallel Coordinates Plots (2/13)

- The ||-Coord representation of the point **(5,3,4,2,0,1)** is:



It appears like this **in ||-Coord**:



Parallel Coordinates Plots (3/13)

- ||-Coord plots are a widely used technique to **display** and **explore multi-dimensional data.**

- **It is good at:** spotting **irregular events**, see the **data trend**, finding **correlations** and **clusters**.

- **Its main weakness**: the **cluttering** of the output. But there are techniques to bypass it.

This summer, we have implemented ||-Coord plots in ROOT as a new plotting option "**PARA**" in the **`TTree::Draw()`** method.

Parallel Coordinates Plots (4/13)

This “pseudo C++” code produces the data set we’ll use to show the ||-Coord usage.

```
void parallel_example() {
   TNtuple *nt = new TNtuple("nt","Demo ntuple","x:y:z:u:v:w:a:b:c");
   for (Int_t i=0; i<3000; i++) {
      nt->Fill(   rnd,    rnd,    rnd,    rnd,    rnd,    rnd, rnd, rnd, rnd );
      nt->Fill(   s1x,    s1y,    s1z,    s2x,    s2y,    s2z, rnd, rnd, rnd );
      nt->Fill(   rnd,    rnd,    rnd,    rnd,    rnd,    rnd, rnd, s3y, rnd );
      nt->Fill( s2x-1, s2y-1,    s2z, s1x+.5, s1y+.5, s1z+.5, rnd, rnd, rnd );
      nt->Fill(   rnd,    rnd,    rnd,    rnd,    rnd,    rnd, rnd, rnd, rnd );
      nt->Fill( s1x+1, s1y+1, s1z+1,  s3x-2,  s3y-2,  s3z-2, rnd, rnd, rnd );
      nt->Fill(   rnd,    rnd,    rnd,    rnd,    rnd,    rnd, s3x, rnd, s3z );
      nt->Fill(   rnd,    rnd,    rnd,    rnd,    rnd,    rnd, rnd, rnd, rnd );
   }
}
```

9 variables: **x, y, z, u, v, w, a, b, c**.

3000*8 = 24000 events.

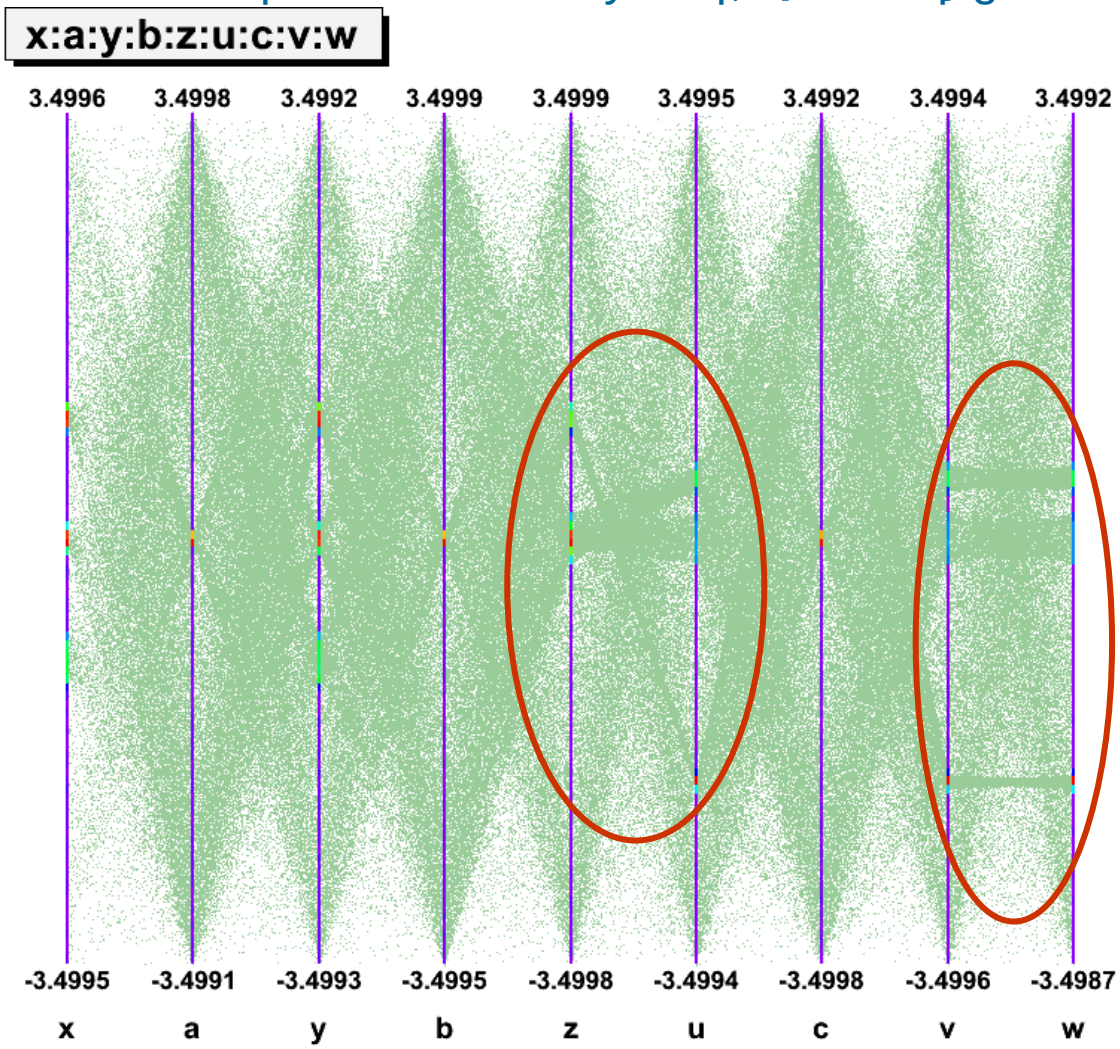
Three sets of random points distributed on spheres: **s1, s2, s3**

Random values (noise): **rnd**

The variables **a,b,c** are almost completely random. **a** and **c** are correlated via the 1st and 3rd coordinates of the 3rd “sphere”.

Parallel Coordinates Plots (5/13)

The usual command is : ntuple->Draw("x:a:y:b:z:u:c:v:w","","PARA") ; It gives:



The lines are plotted with a color palette.

The thickness can be changed.

The list of parameters can be adjusted in order to get the best representation.

The space between the dots is a parameter which **can be adjusted** in order to get the best result.

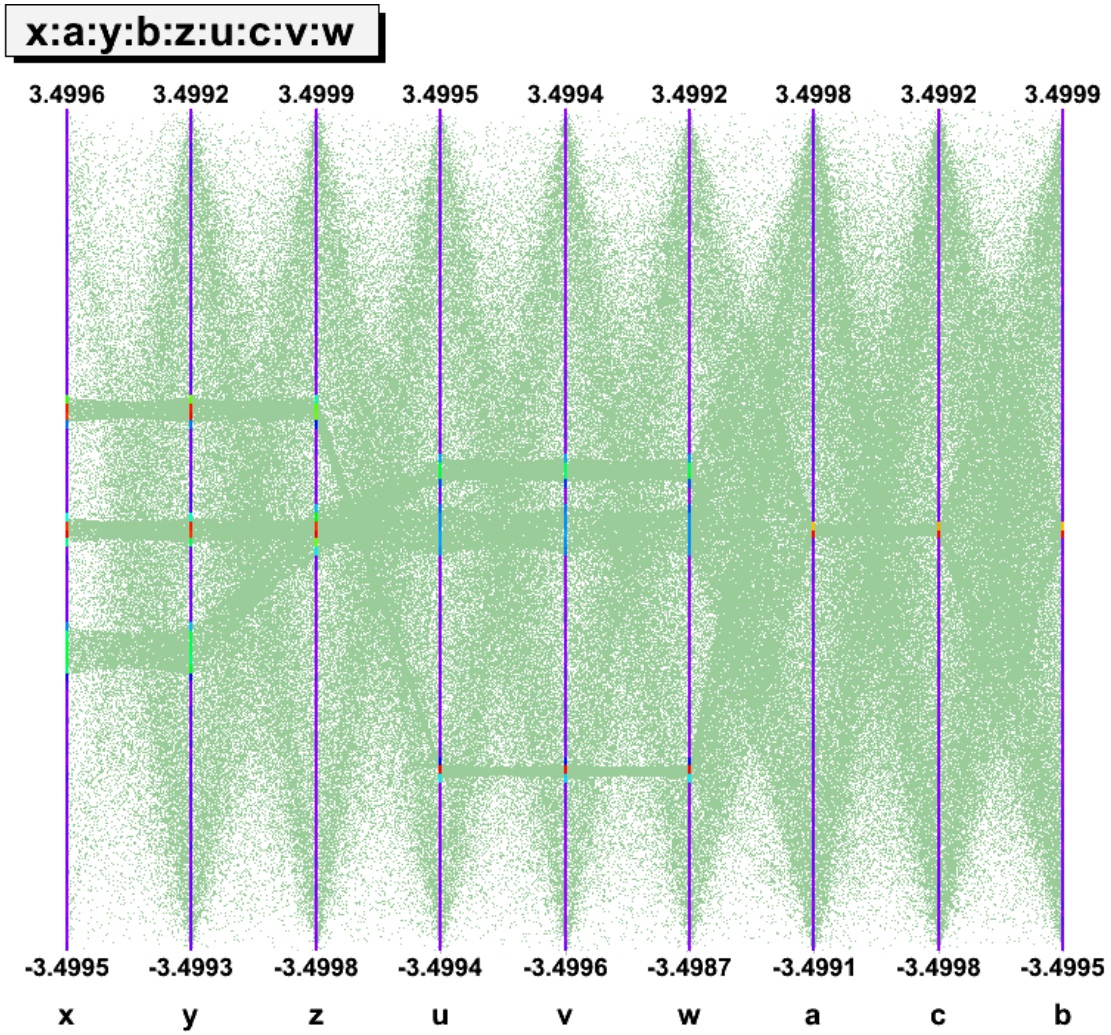
The clusters (in this case the “spheres”) now appear clearly !

But still the clusters are not visible…

The image cluttering is very high !

Parallel Coordinates Plots (6/13)

The order in which the axis are displayed is very important to show clusters:



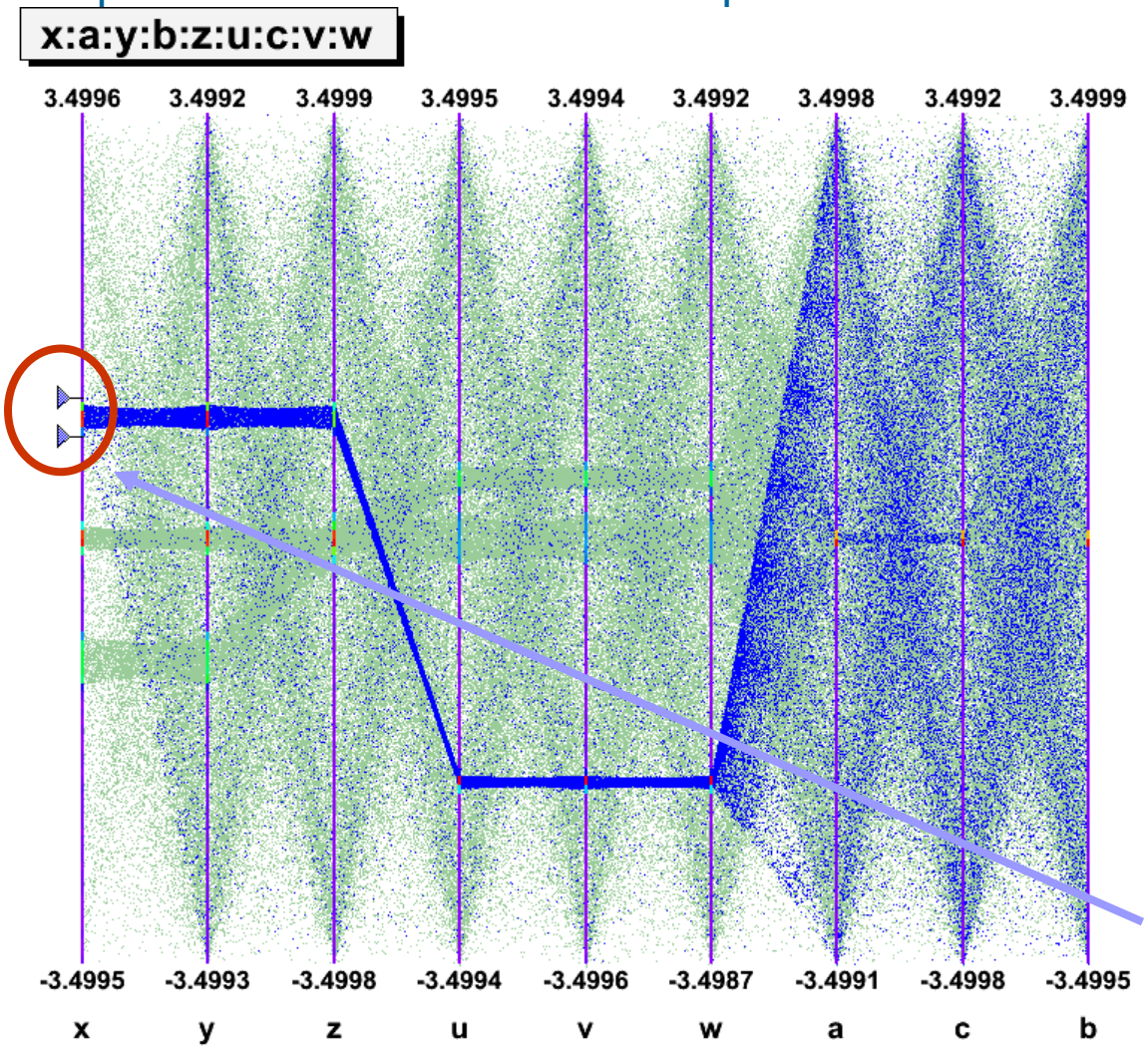
The axis can be **moved interactively**.

All the **clusters** we have introduced in the data set are now clearly visible.

The **correlation** between u, v, w after x, y, z is clear also.

Parallel Coordinates Plots (7/13)

To pursue further the data set exploration one can use selections.



A **selection** is a set of **ranges** combined together.

Within a selection, ranges along the same axis are combined with **OR**, and ranges on different axis with **AND**.

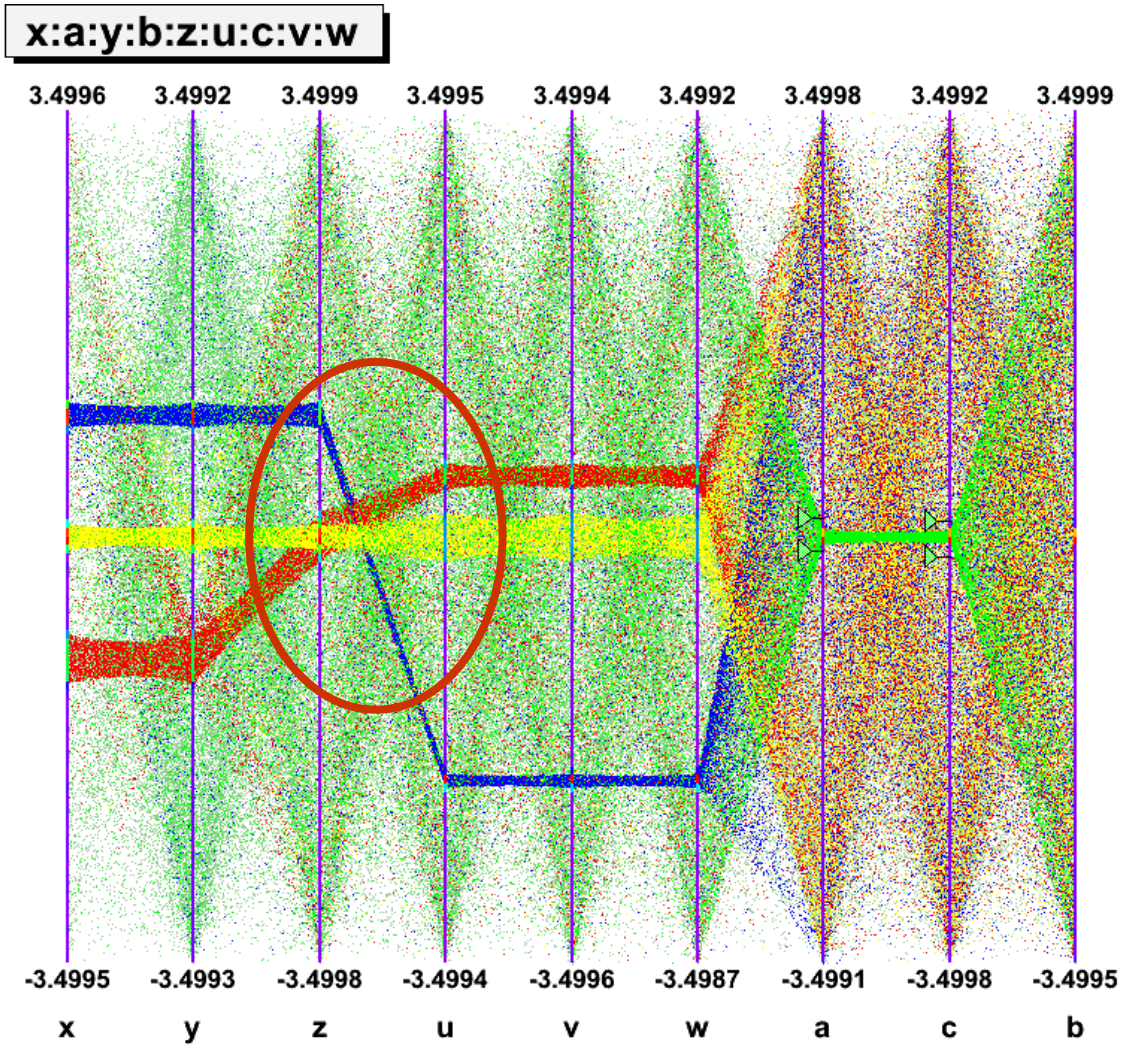
A selection is displayed on top of the complete data set using its own color.

Only the events fulfilling the selection criteria (ranges) are displayed.

Ranges are defined interactively using **cursors**.

Parallel Coordinates Plots (8/13)

Several selections can be defined:

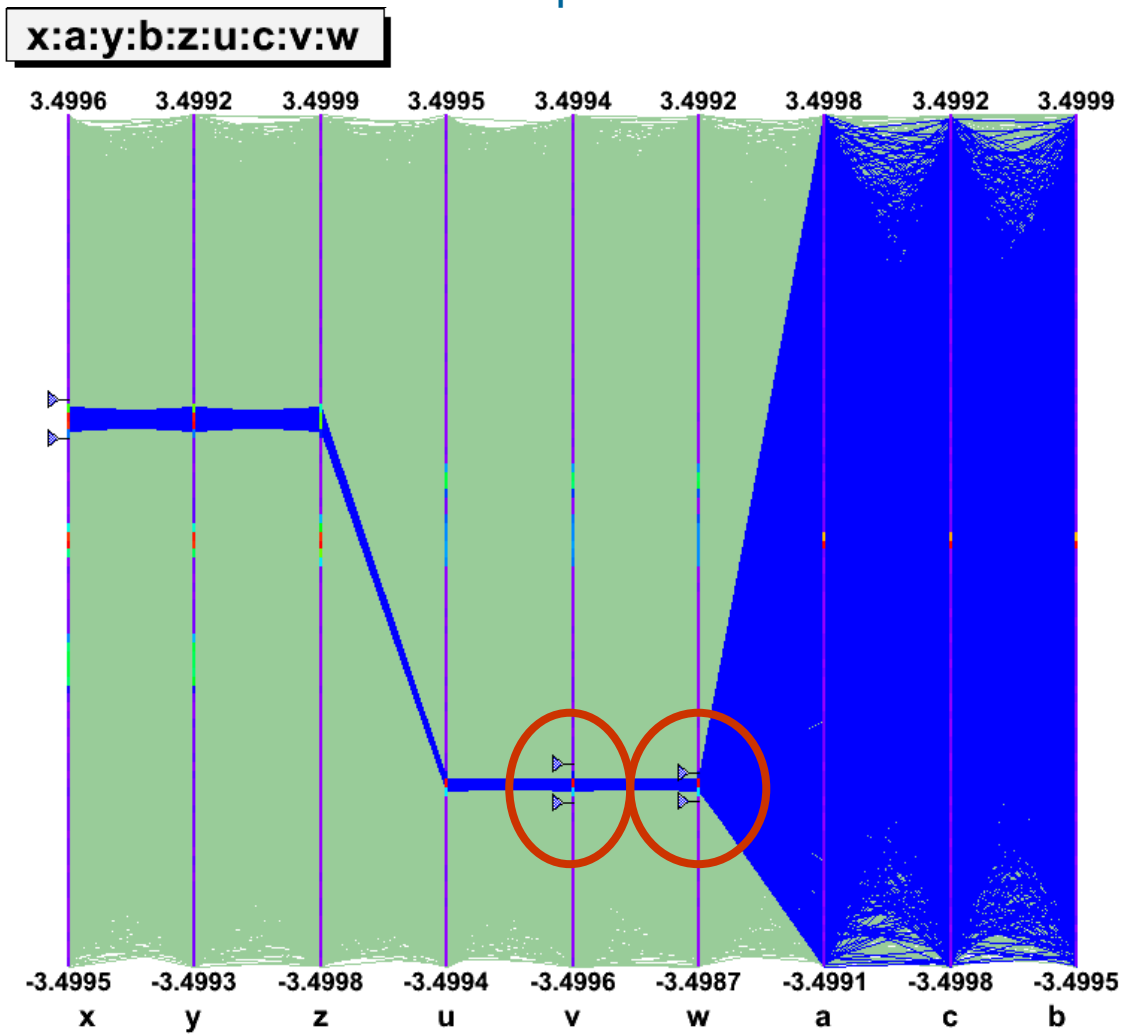


Each selection has its own color.

Thanks to the multiple selections this zone with crossing clusters is now understandable.

Parallel Coordinates Plots (9/13)

Selections allow to make precise events choices.



This single selection displayed with an appropriate dots-spacing shows clearly a cluster.

Displayed with solid lines the cluttering shows up again.

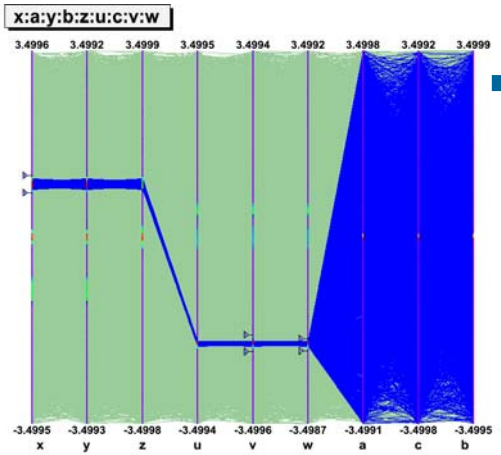
Adding a range clears the picture.

A third range allows to show one single event outside the cluster. It would have been hard to see it with dots-spacing.

A final adjustment selects precisely the cluster on 6 variables.

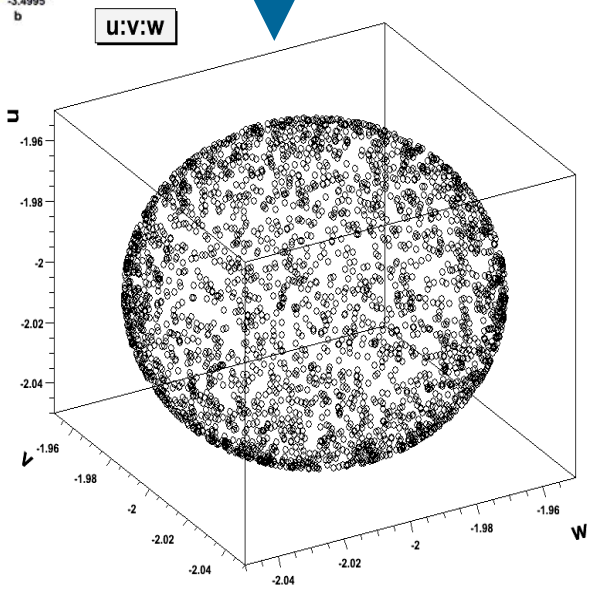
Parallel Coordinates Plots (10/13)

Selections can be saved as **`TEntryList`** and applied to the original tree .

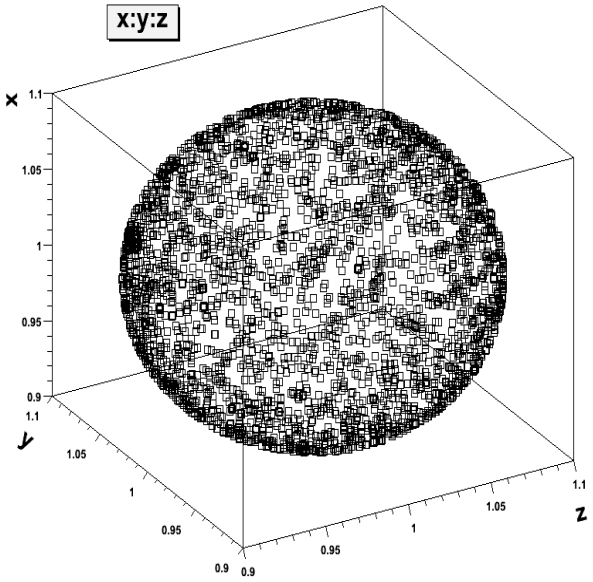


Apply the selection to the tree via a **`TEntryList`** .

`nt->Draw("u:v:w")`



`nt->Draw("x:y:z")`

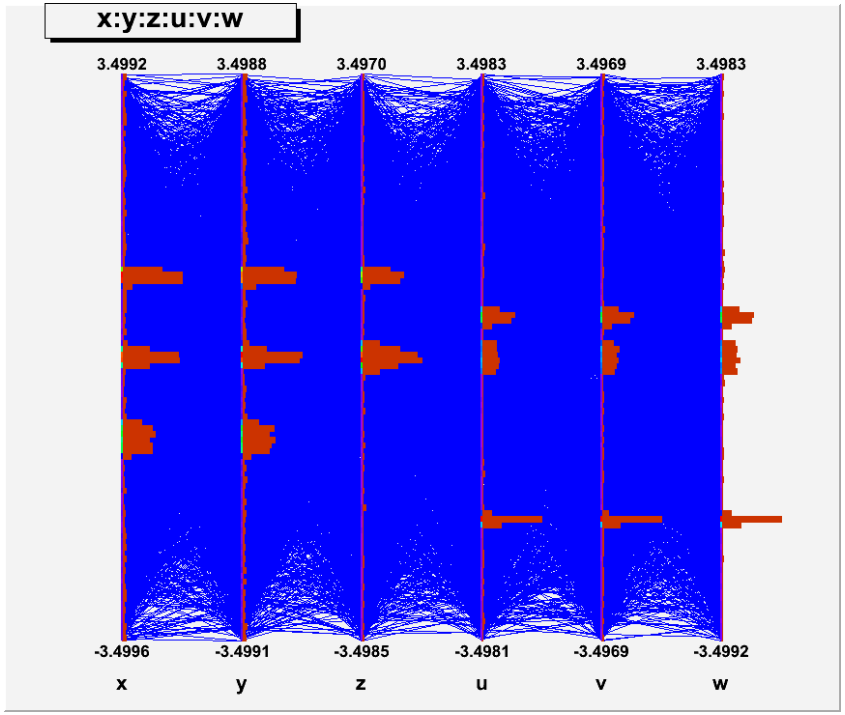


Parallel Coordinates Plots (11/13)

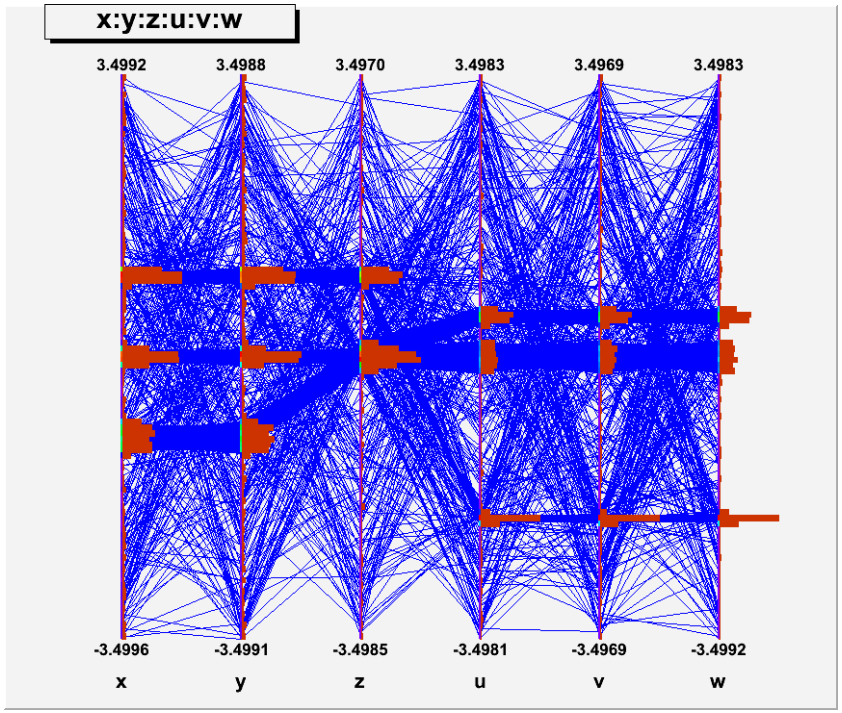
An other technique has been implemented in order to show clusters when the picture is cluttered. **A weight is assigned to each event**.

The weight value is **computed as**: $$weight = \sum_{i=1}^{n} b_i$$

Where b_i is the content of bin crossed by the event on the i^{th} axis. n is the number of axis. The events having the **bigger weight** are those **belonging to clusters**. Then, it is possible to paint only the events having a weight above a given level and the clusters appear.

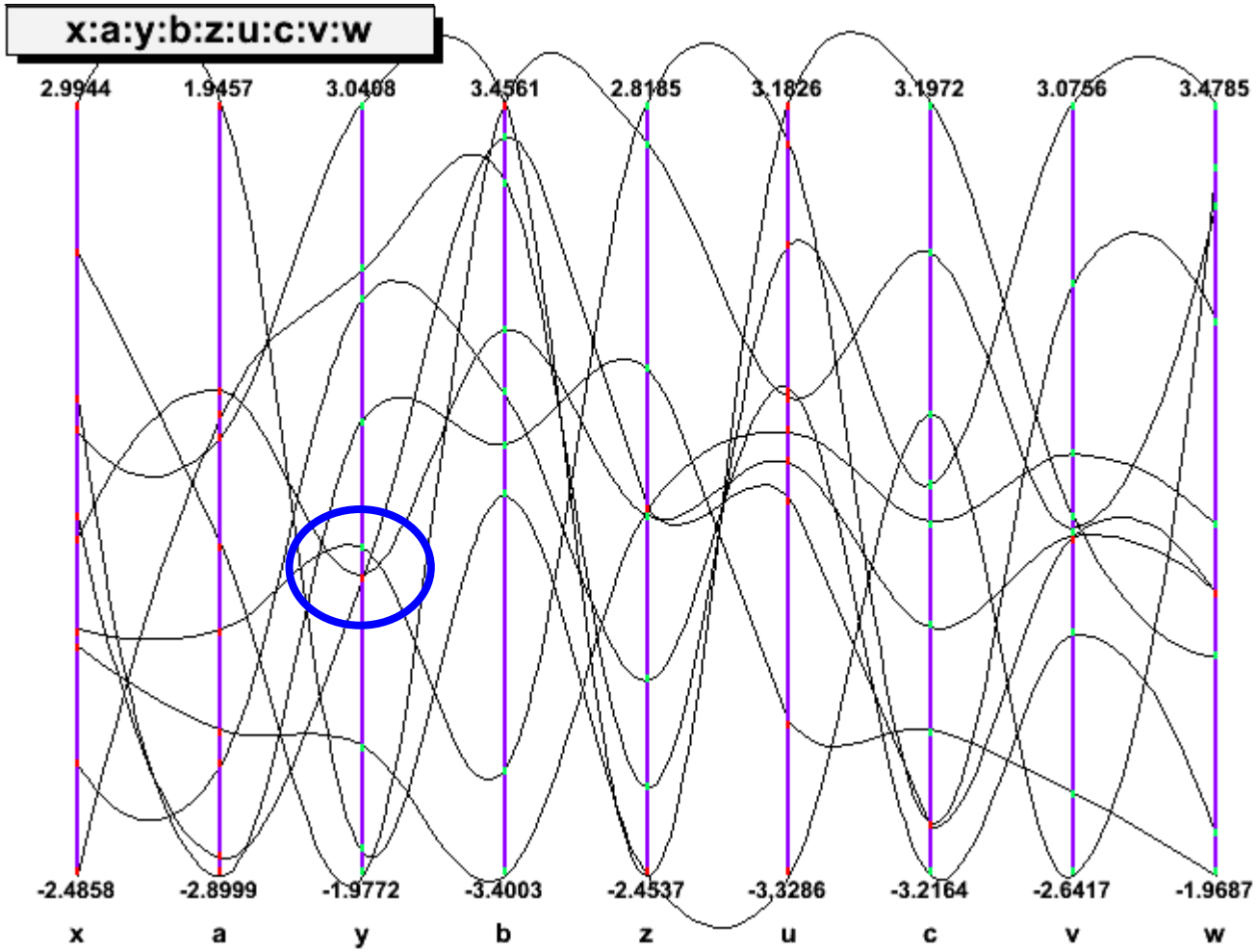


weight > 50



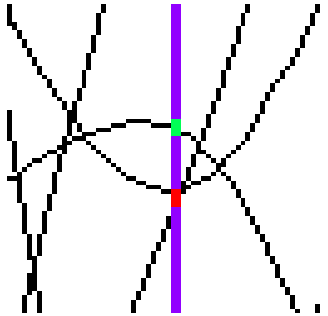
Parallel Coordinates Plots (12/13)

- With few events, displaying as smooth curves helps to differentiate events:



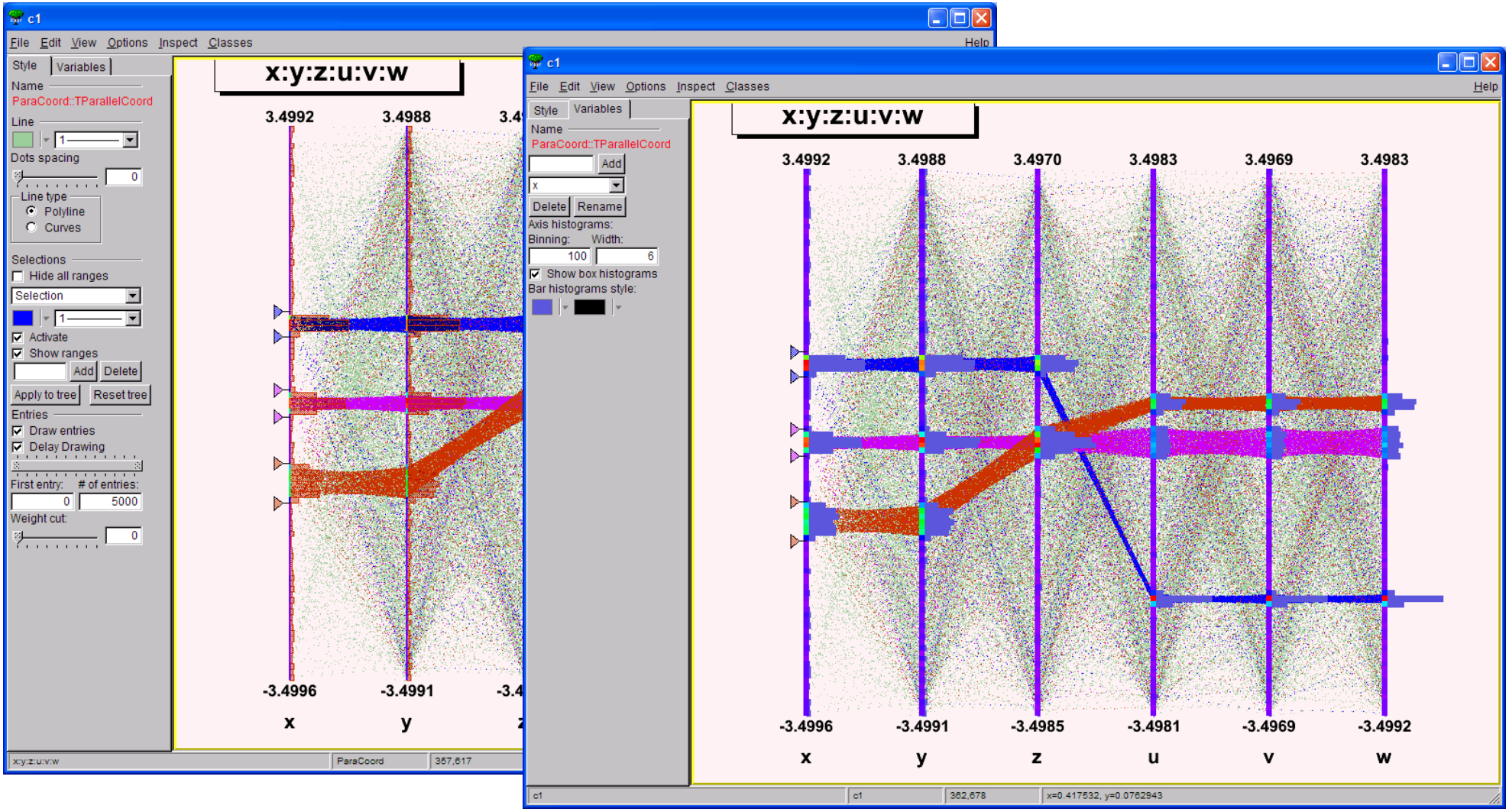
It is difficult to decide which lines are going together.

Displaying with smooth curves makes it clear.



Parallel Coordinates Plots (13/13)

The Graphical User Interface is very important to manipulate the ||-Coord plots:



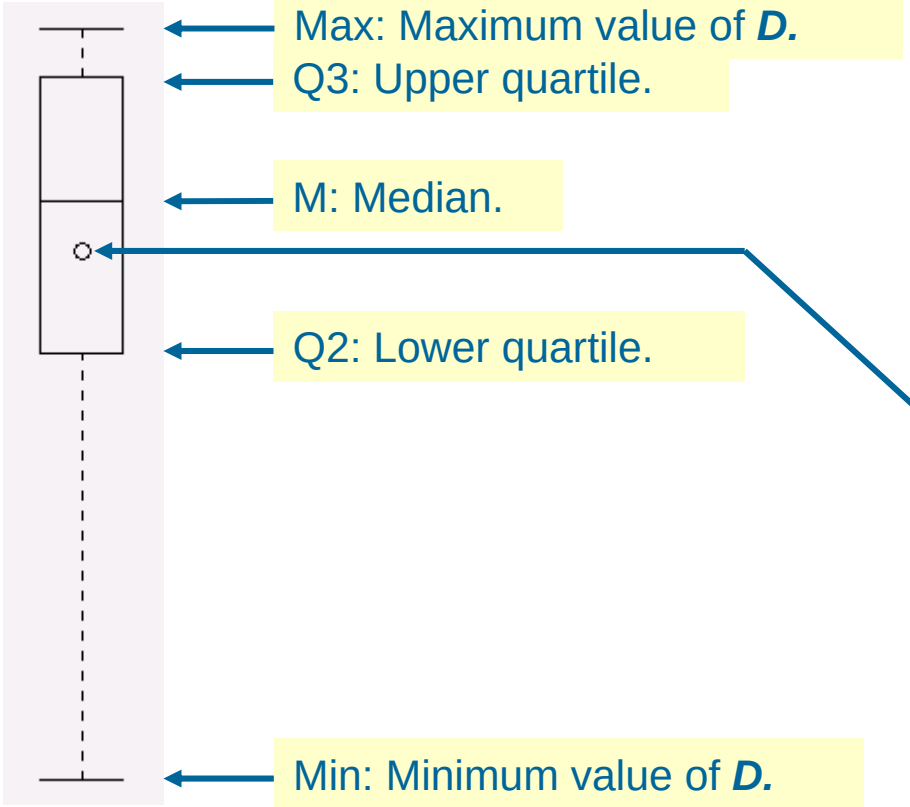
Box Plots (1/3)

- A **Box Plot** (also known as a “box-and whisker” plot or “candle stick” plot) is a convenient way to describe graphically a data distribution (***D***) **with only the five numbers**. It was invented in 1977 by John Tukey.

- The five numbers are:
  1. The minimum value of the distribution ***D*** (Min).
  2. The lower quartile (Q1): 25% of the data points in ***D*** are less than Q1.
  3. The median (M): 50% of the data points in ***D*** are less than M.
  4. The upper quartile (Q3): 75% of the data points in ***D*** are less than Q3.
  5. The maximum value of the distribution ***D*** (Max).

Box Plots (2/3)

For a distribution ***D***, the values Min, Q1, M, Q3, Max are represented the following way:



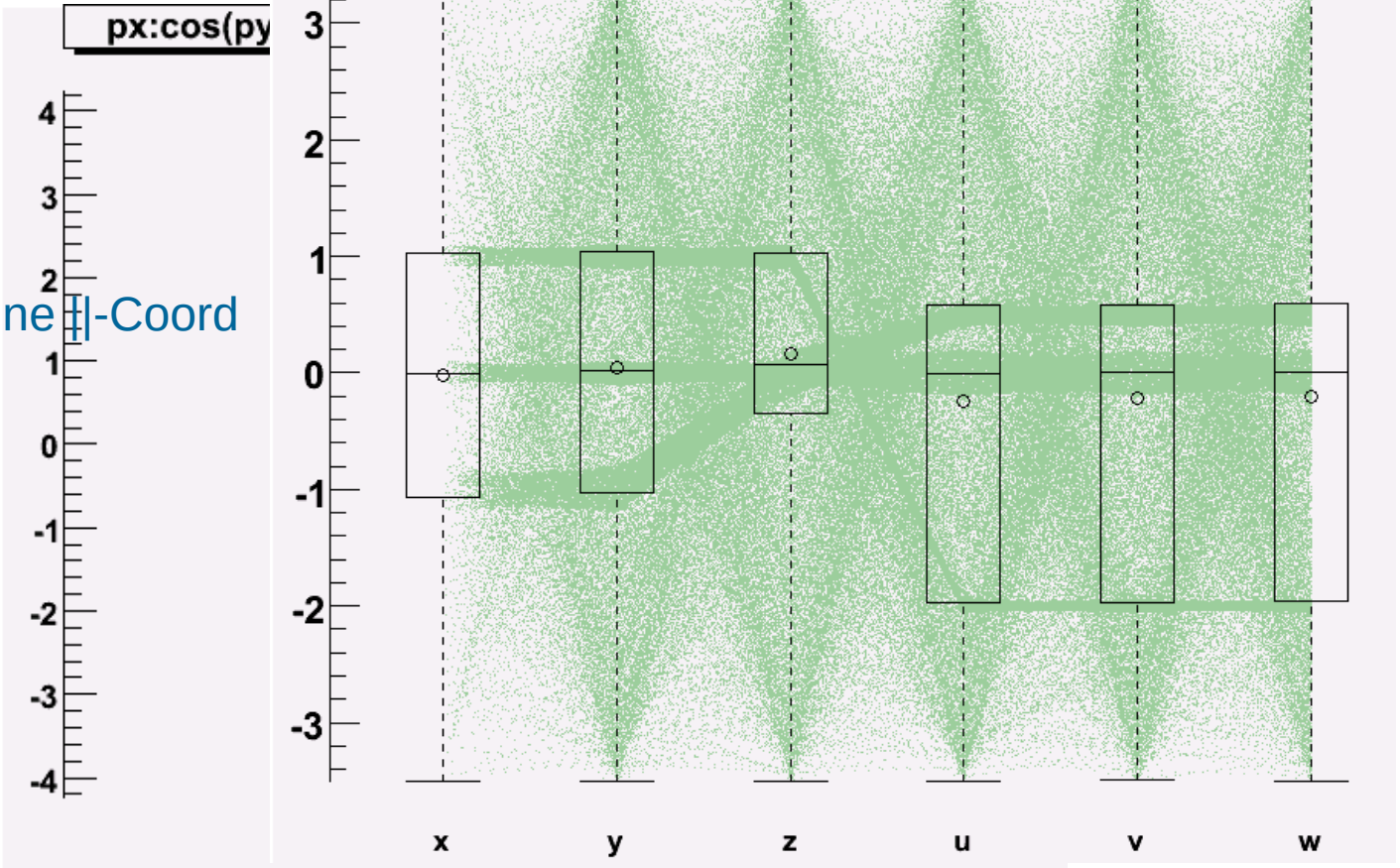
The **box** represents the middle 50% of the distribution ***D***. That is the **"body"** of the data set.

Very often the **mean value** of the distribution ***D*** is also represented.

Box Plots (3/3)

In **ROOT** Box Plots (Candle Plots) can be produced from a **`TTree`** using the **`"candle"`** option in **`TTree::Draw():`**

Example: **`tree->Draw("px:cos(`**



Its is possible to combine ||-Coord and Candle-Plots:

Conclusion

Important new developments have been made in 2D and 3D graphics during the past 18 months. More are needed:

- **2D Graphics**: The list of developments planed is quite long.
- **OpenGL 3D graphics**: The future developments will be in the integration of the 2D and 3D graphics in the GL context to benefit 3D graphics acceleration.
- **Multiple variable data set Visualization**:
  - ||-Coord:
    - To show the cluster and fight the cluttering several techniques are available. Like the transparency and shading. We should explore this tracks in the future. Right now we have implemented simple techniques which show good result already.
    - Find clusters automatically.
    - Sort axis automatically.
  - Investigate other techniques than ||-Coord.