AlVisu : a simple 3D viewer

What it is ?

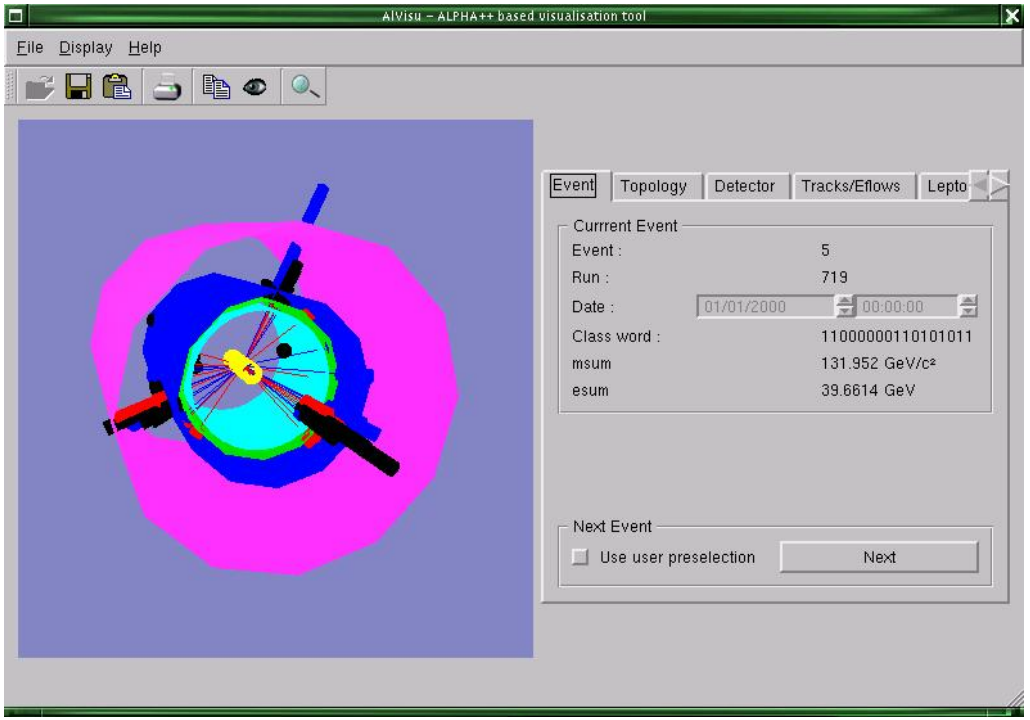
Simple 3D viewer

- vertex information
- simple detector
  (to guide the eyes)
- based on ALPHA++

What it is not ?

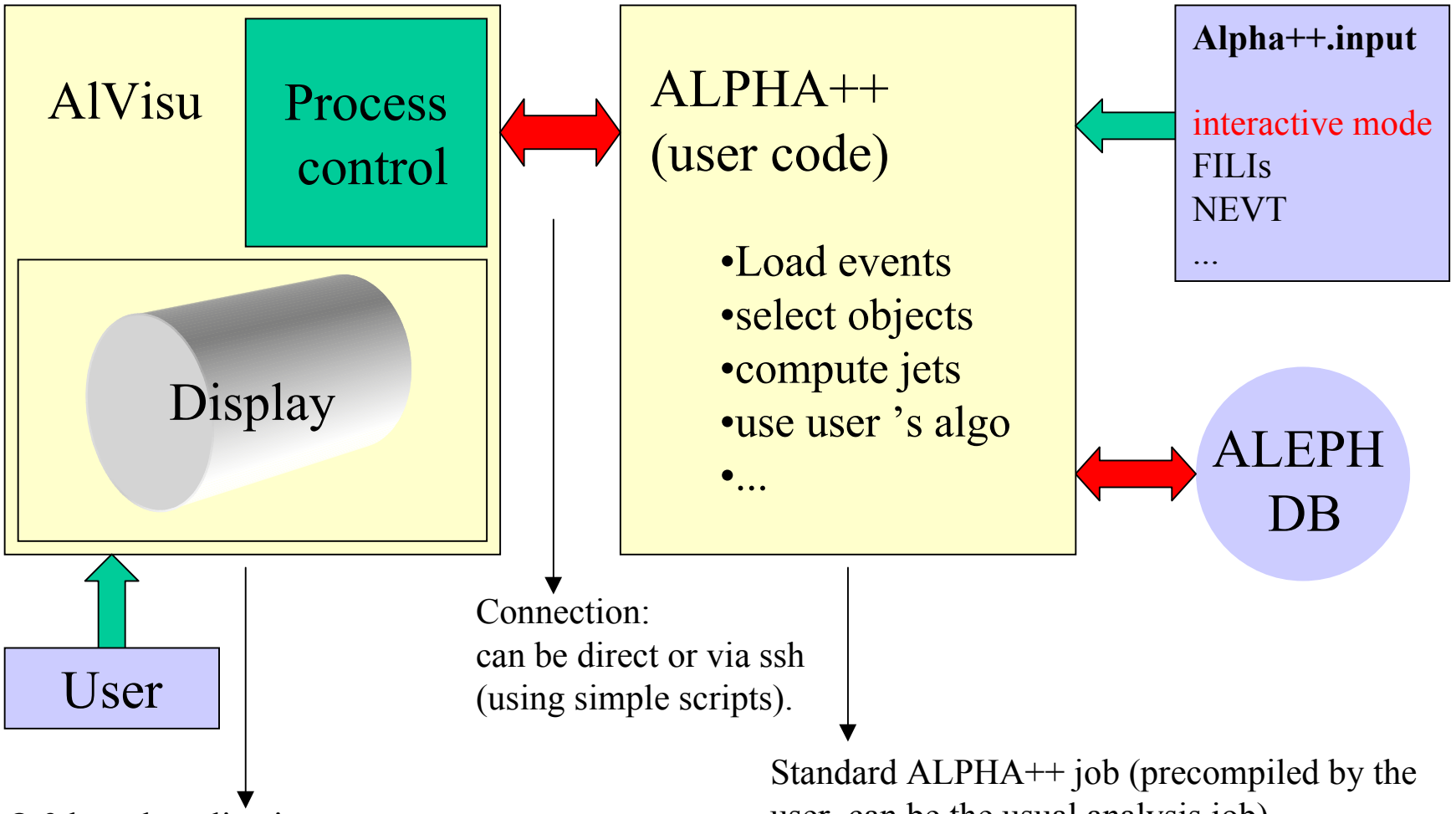
A Dali replacement

- no calo information
- no curvature
- ...



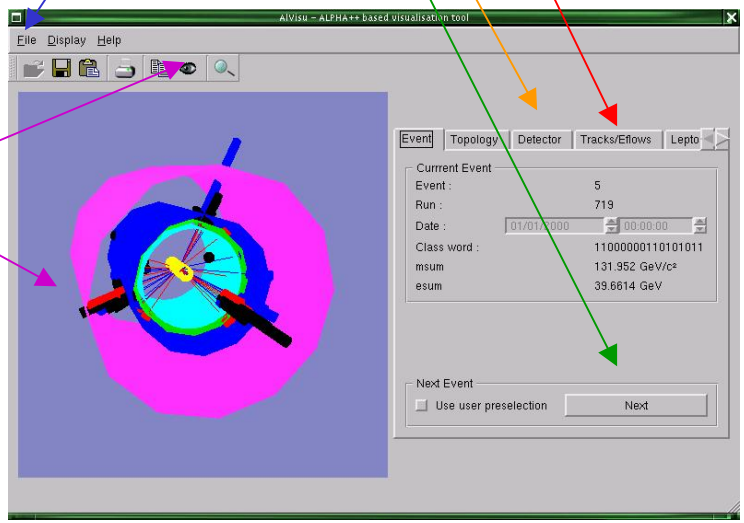
You can turn around an event, and better understand the signature (dynamical cuts), but you don't have access to all the details of the event (only point-like information)

How it works ?


AlVisu
Process control
Display
ALPHA++
(user code)
•Load events
•select objects
•compute jets
•use user 's algo
•...
Alpha++.input
interactive mode
FILIs
NEVT
...
ALEPH DB
User
Connection:
can be direct or via ssh
(using simple scripts).
•Qt3 based application
•compiled and executed on any computer
Standard ALPHA++ job (precompiled by the user, can be the usual analysis job).

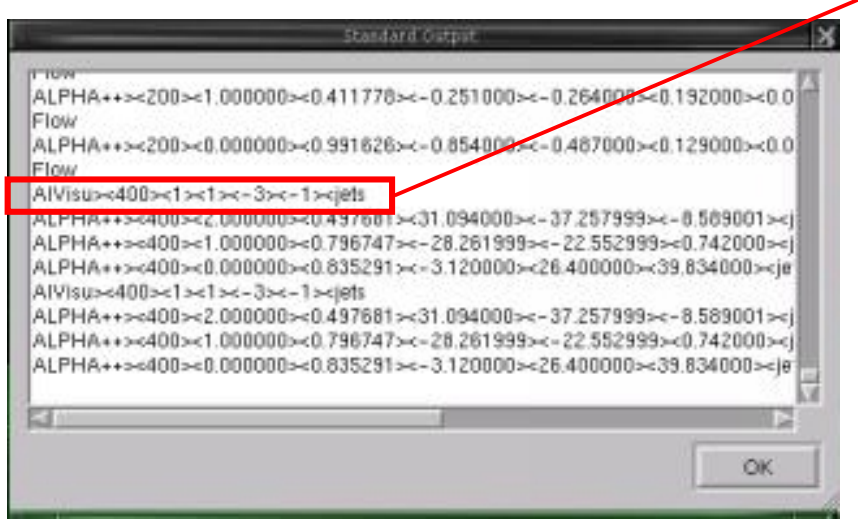
A typical AlVisu session

- AlVisu
  - Run AlVisu
- File, Open… select the ALPHA++ binary to you want to run
  - alpha++.input must be in the AlVisu directory, or you must use scripts (e.g. to run via ssh)
  - this will unlock the main interface
  - you can see the messages send from/to ALPHA++ in the standard output window
- Press 'Request next event' on the main window
  - wait… the tape has to be staged…
  - when the event is loaded, run/event are displayed on the main display
- Select the track tab and request the Eflows
  - they are now displayed on the scene
- Select the detector tab and display the ecal endcaps
- Choose your view angle
  - look at the event :-)
- File, Quit.
  - Done.



Listening the conversation...

The « standard output » window allows you to look at the messages send to/from ALPHA++



Typical message:

AlVisu><400><1><1><-3><-1><jets

sender

Request : 400 = jets

options

comment

To use a remote code via ssh

AlVisu is a very graphical user interface, not realy suited to remote X connections.

It 's interesting to be able to run AlVisu localy and the « slave » ALPHA++ code remotely (on lxplus)

How to do it ?

Set up a script localy:

```
#!/bin/sh
ssh login@lxplus.cern.ch '~/remotescript'
```

Set up a script remotely:

```
#!/bin/tcsh
cd ~/alpha++/temp/Applications
Linux/test1 -l0
```

Then simply select the local script from AlVisu to start.
Don't forget to type your password in the xterm where AlVisu was started.

What 's new in ALPHA++

3 classes were added in the driver part:

AlephInteractiveHandler

```
void  ConfigureProtocol (string myname, string yourname)
void  InitiateConnection (int &ier)
int   HandleEvent (AlphaBanks &bb)
void  CloseConnection ()
bool  KeepAllEvents ()
Static AlephInteractiveHandler * TheAlephInteractiveHandler ()
```

- is responsible for the communication with AlVisu via the standard output.
- is a singleton instanciated by the Execution manager
- is used in the newly created interactive loop

AlephAbstractInteractiveFunction

```
virtual string   Name ()=0
virtual int      Code ()=0
virtual void     Run (vector< float > &options, AlphaBanks &EventInfo)=0
virtual vector< pair< string,float > > OptionsList ()=0
void SendMessage (int code, vector< float > &options, string comment)
```

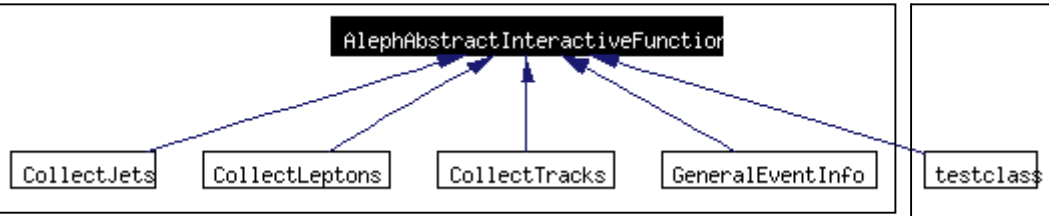
- Is an abstract function
- is the generic class for all interactive actions

AlephRegisteredAction <T>

- Used to register an interactive function.

What 's new in ALPHA++

Default situation:



Example code test1

In AlephExManagers,
the interactive functions are declared:

```
// initialisation of default interactive functions
AlephRegisteredAction<GeneralEventInfo> code100;
AlephRegisteredAction<CollectTracks> code200;
AlephRegisteredAction<CollectLeptons> code300;
AlephRegisteredAction<CollectJets> code400;
```

Modification of an existing class: UserEvent now returns a bool.

- True: good event
- False: bad event

This is used to display only good events (if requested) according to user's definition.

→ Version 3.4 not backward compatible...

Adding a new function

It 's very easy to add a new functionnality to AlVisu using the standard interface.

- Create a new class, deriving from AlephAbstractInteractiveFunction.
- Implement:
  - virtual string Name()
    returning the name of the function
  - virtual int Code()
    returning a number larger than 500
  - virtual vector<pair<string,float> > OptionsList()
    returning a vector of pair (option name, default value)
  - virtual void Run(vector<float>& options,AlphaBanks& EventInfo)
    the actual function (UserEvent-like)

- In UserInit(),
  instanciate AlephRegisteredAction, templeted with your new class.

Conventions

There are some conventions in the format of the returned objects.

Argument 1: index.

Decreasing number n→0.

When 0 is received, this is interpreted as the last object, and averything is drawn.

Argument 2: object type.

1	a track
2	an eflow
3	a lepton
4	a jet

Next arguments are function of the object type:

track:	px, py, pz, ch	
eflow:	E, px, py, pz, ch	
lepton:	E, px, py, pz, ch, flavour(1:e 2:μ 3:τ)	
jets:	cos(θ), px, py, pz	*cos(θ) gives the cone half-angle*

In your code, you can use the SendMessage function
to send well-formated messages (with additionnal code and message).

An example

```
class testclass:public AlephAbstractInteractiveFunction
{
      public:
            testclass(AlephInteractiveHandler*ptr):
                                    AlephAbstractInteractiveFunction(ptr) {}
            virtual string Name() {return "test";}
            virtual int Code() {return 555;}
            virtual void Run(vector<float>& options,AlphaBanks&
EventInfo)
                  {
                        // dummy routine: returns always the same track,
eflow and the same jet
                        vector<float> output;
                        output.push_back(2);
                        output.push_back(1); // a track
                        output.push_back(3); // px
                        output.push_back(0); // py
                        output.push_back(0); // pz
                        output.push_back(0); // ch
                        SendMessage(Code(),output,"Dummy test routine");
                        output.clear();
                        output.push_back(1);
                        output.push_back(2);  // an eflow
                        output.push_back(10); // E
                        output.push_back(0);  // px
                        output.push_back(10); // py
                        output.push_back(0);  // pz
                        output.push_back(1);  // ch
                        SendMessage(Code(),output,"Dummy test routine");
                        output.clear();
                        output.push_back(0);
                        output.push_back(4);   // a jet
                        output.push_back(0.5); // sin theta ~ radius
                        output.push_back(0);   // px
                        output.push_back(0);   // py
                        output.push_back(5);   // pz
                        SendMessage(Code(),output,"Dummy test routine");
                  }
            virtual vector<pair<string,float> > OptionsList()
                  {
                        vector<pair<string,float> > output ;
                        pair<string,float> mypair;
                        mypair.first = "option 1";
                        mypair.second = 10;
                        output.push_back(mypair);
                        mypair.first = "dummy 2";
                        mypair.second = 3.1415;
                        output.push_back(mypair);
                        return output;
                  }
};

void AlephExManager::UserInit()
{
                  AlephRegisteredAction<testclass> mytestclass;
}
```

Documentation

•Up to now, this presentation is the main source of informations.

•A chapter will be available in the ALPHA++ manual, describing the ALPHA++ part and the full protocol.

•There are some informations on the website:
| *http://cern.ch/aleph-proj-alphapp/doc/alvisu.html*

•The interface *should be* self-explanatory

If you use it and have problems/questions, please contact me.