BEAM_LINER –- AN OBJECT ORIENTED BEAM LINE MODELING C++ CODE

I.P. Yudin[#+], A.V. Trofimov, JINR, Dubna 141980, Moscow region, RUSSIA

Abstract

An integrated expert system has been developed to solve the charge particle beam optics problems [1,2]. The optics elements are a drift space, a bending magnet, a quadrupole, a sextupole, an octupole, a solenoid, an accelerating tube and some elements introduced by the users. BEAM_LINER is coded by the C++ language and the UNIX operation system with a graphical user-oriented interface on the PC Pentium II.

1. Introduction.

The user-oriented interface is created allowing to solve the beam matching problem, to compute of betatron functions, of phase advances and to investigate other problems. The beam motion is described on of the envelopes (of β-functions) and the frequencies of betatron oscilations (what is by the ideology of the methods of the snooping abroad of phase multitude). The motion of the single particles of the beam (trajectories) is described on the simple language of the transfer matrixes. But it is possible by the numerical integration, also.

2. The description of member functions and the class structure.

BEAM_LINER is written by means of object-oriented design [3] in X Window system without the using of ready libraries of control elements (such as Motif, Athena and etc.). That is why we created some minimal set of classes which are necessary for the building of user-oriented interface (some buttons, the edit boxes, the message boxes, etc). Moreover, we created the base class (Xelement) for the representation of elements, of which the system of beam optics itself is to be constructed. Below we will describe four from the set of elements: those are a drift of space, a quadrupole, a solenoid, and dipole. For them we created four different classes - successors of Xelement. Fact is that all classes of control elements are successors of one abstract base class and inherit its member functions - that allows to curtail greatly the measures of the main loop which receives and handles the events. The using of virtual functions lets us not to re-write the same fragments of code for the elements of different types.

----

* Work supported in part by Russian Federal Program ‘Integration’, contract No. K0085.

+ E-mail: yudin@cv.jinr.ru

In header-files obj1.h, obj2.h are described:
1) Abstract class Xobject
2) Class Xbut, successor of Xobject
3) Class Xbut1, successor of Xbut
4) Class Xfield, successor of Xobject
5) Abstract class Xelement.
6) Class Xdrift, successor of Xelement.
7) Class Xquadro, successor of Xelement.
8) Class Xdipol, successor of Xelement.
9) Class Xsolen, successor of Xelement.

2a. Classes of control elements:

```
class Xobject
 { public: Display *dis;
   Drawable win; /*main window's descriptor*/
   GC *prGC;/*points at the structure of graphic
   context*/
   unsigned int x; unsigned int y; unsigned int dx;
   unsigned int dy; unsigned short deep; unsigned short
   cr; unsigned  short col; char title [20]; char text [20];
   int f; unsigned short tp; virtual void show (); virtual
   void hide (); virtual void set_p();
   virtual void click1 (); virtual void click2 ();
   virtual void key_pressed (char, int); Xobject (Display,
   Drawable, GC, unsigned int, unsigned int, unsigned
   int, unsigned int);};

class XBut: public Xobject
 {public: unsigned short deep; unsigned short cr;
  void hide (); virtual void click1 (); void click2 ();
  void key_pressed (char, int); void show (); void
  show_pr ();
  XBut (Display, Drawable, GC, unsigned int, unsigned
  int, unsigned short, unsigned short, unsigned short,
  char, unsigned short, unsigned short);};

class XBut1: public Xbut
 {public: void click1 (); XBut1 (Display, Drawable,
   GC, unsigned int, unsigned int, unsigned short,
   unsigned short, unsigned short, char, unsigned short,
   unsigned short);};

class Xfield: public Xobject
 {public: unsigned short p; void get_symbol (char, int);
   void set_p (); void show (); void click1 ();
   void click2 (); void hide (); void key_pressed (char,
   int); Xfield (unsigned int, unsigned int, unsigned
   short, unsigned short, char, Display, Drawable, GC,
   unsigned short);};
```

Here dis points on the main display, win is the descriptor of window which owns an object, prGC points at the structure of the graphic context of window, x, y, dx, dy are coordinates of upper left corner and the measures of an object, accordingly. All these parameters are handed over when the constructor is being called. Fields deep and cr matter only for Xbut and Xbut1 classes, they define the geometry of buttons (for example, deep defines the thickness of buttons), as well as field f is, which indicates if the button is pressed or not. Col value defines the color of an object, line title contains inscription on button, or over the edit box in case of Xfield, and text line is appointed to store a text, inputed by user. At the window's creation it is being defined by masks, that only Expose, MotionNotify, KeyPress, and ButtonPress events will be handled in main loop. Event Expose, forcing window to redraw itself, comes, for example, with hitting one of buttons or with the text appearance or changing in an edit box; certainly, all control elements will be re-drawn in updated state. When one of three last events is being intercepted, program looks through the array which contains pointers at all existing control objects (the pointers are of Xobject * type). Using of virtual functions lets us to use this single scheme of objects' reaction on incoming events, though different classes may react differently.

Virtual functions click1 (), click2 (), key_pressed (char, int), inherited by all classes – successors of Xobject - define, how an object should respond on clicking with the left or right mouse key, or on events from keyboard, accordingly. Show() and hide() functions are responsible for showing/hiding some object when window is being updated. Class Xfield ignores Buttonpress events, the text box becomes active in the very moment the mouse's cursor comes in its borders. Besides, class Xfield has special member function get_symbol (...) for the input and processing of information, coming to edit box. Only this class really treats this event (classes Xbut and Xbut1 just ignore its).

2b. Classes Xelement, Xdrift, Xquadro, Xdipol, Xsolen:

```
class Xelement
 {public: Display *dis; Drawable win; Drawable win2;
   GC *prGC; unsigned short obj_type; unsigned int
 num; double lenght; unsigned int col; char name  [20];
   void show (unsigned int); void OpenWin2 (unsigned
   int, unsigned int); void respond (unsigned int,
   unsigned int); Xelement (double, unsigned int,
   Display, Drawable, GC, unsigned short); virtual
   double get_e1 (); virtual double get_e2 (); virtual void
   set_e1 (double); virtual void set_e2 (double); virtual
   double get_B (); virtual void set_B (double);};

class Xdrift: public Xelement
 {public: Xdrift (double, double, unsigned int, Display,
   Drawable, GC, unsigned short); void set_e1 (double);
   void set_e2 (double); void set_B (double); double
   get_e1 (); double get_e2 ();
   double get_B ();};

 class Xquadro: public Xelement
  {public: Xdrift (double, double, unsigned int, Display,
    Drawable, GC, unsigned short);
    void set_e1 (double); void set_e2 (double); void
    set_B (double); double get_e1 (); double get_e2 ();
    double get_B (); private: double B;};

class Xdipol: public Xelement
 {public: Xdrift (double, double, unsigned int, Display,
   Drawable, GC, unsignedshort);
   void get_e1 (double); void set_e2 (double); void
   set_B (double);  double get_e1 (); double get_e2 ();
   double get_B (); private: double B; double e1; double
e2;};
```

Class Xelement and it's successors demand particular description. dis, win and prGC values have the same specification, as in class Xobject. Descriptor win2 is appointed to access to control panel, which opens with the hitting of the right key of mouse. Panel is necessary to set values of the parameters of elements. After all values had been set, function respond (...) is called, which in its turn calls function OpenWin2 (...), assigning recommendations to the windows manager and creating window for panel, and then launches the main loop to intercept and handle the events. Window contains several edit boxes and a button, which has to be hit to stop the loop, and then the characteristic values of elements admit meanings stated in corresponding edit boxes.

All constructors and the member functions of classes are described in header files obj2.h, obj3.h, and obj_draw.cpp

3. The interface description.

The whole code of program is split into separately compilable modules. Module main.cpp contains function main, which fills objects array, creates the main program's window and launches the main loop. Main loop acts in the following way: in the moment when next event has been received, an array of objects is being looked through. If the mouse's cursor were located in the frontiers of an object (lets call such an object "active"), then it's corresponding response-function and then function ResponseTable, residing in file response.cpp, are being called. After they finished, the window is to be updated. If the right key of mouse (rEvent. type==ButtonPress, rEvent. xbutton. button==Button2) were pressed while cursor were located over the special position, averted for an element then, in case, if this position were not empty, a respond(...) element's member function is being called. If it was left key (rEvent. type==ButtonPress, rEvent. xbutton. button==Button1), while cursor where over one of the positions, then an element defined by significance obj_flag (this significance changes by hitting one of the buttons - Drift, Quadro, Solenoid, Dipol) is being placed in the position. All values, characterizing element, are assigned to equal zero by the default and may be changed later in the function respond(...), or an element maybe destroyed. By the hitting of button SAVE created configuration will be stored in file with the name, given in edit box filename. Previously saved configuration maybe downloaded later again by the hitting of LOAD button (all corresponding functions are described in file load_save.cpp). File is just a sequence of records, which contain the type of element and it's values. While loading, they are being read one after one, and elements are being placed in corresponding positions. Finally, Quit button brings to exit from program without saving of current configuration.

Buttons Drift, Quadro, Solenoid and Dipol serve to select the elements, button NEW destroyes the configuration and cancels all settings. Algorithm, calculating the trajectory itself, starts working with the hitting of Go button.

rec_col.cpp module also contains receive_colors(...) function, which defines colors. One of the first actions, performed by function main after standard procedures - the setting connection with graphic server, the definition of the number of main screen, and so on - is receive_colors()'s call; as a result, global variables which contain color values are defined. Uconv.cpp contains several accessorial functions, usefull for the conversion of type char in double, and etc.

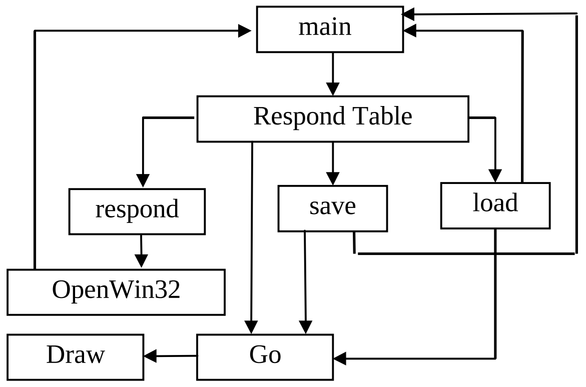

Fig.1. The base modules of the program.

Drawing itself is performed by function Draw which uses, in its turn, special graphic library PlPlot [4].
All global variable, constants and the prototypes of functions are described in header files init.h, init2.h.

More discriptions are avaible in [5].

4. References

[1] Andreev V. V., Yudin I. P. Third-Order Optics of the Real Solenoid Lens. In Proc. International Conference on HEACC'92. Humburg, Germany, July 15-18, 1992.
[2] Brown K.L., et al. TRANSPORT. A Computer Program for Designed Charged Particle Beam Transport System. SLAC-91, Rev. 2 UC-28 (1/A). May 1991.
[3] Grady Booch. Object Oriented Design. With Application. The Behjamin/Cummings Publishing Company, Inc. New York, 1991.
[4] PlPlot User Guide.
ftp dino.ph.utexas.com
[5] I.P.Yudin, A.V. Trofimov BEAM_LINER User Guide, JINR, Dubna, March 3, 1999.

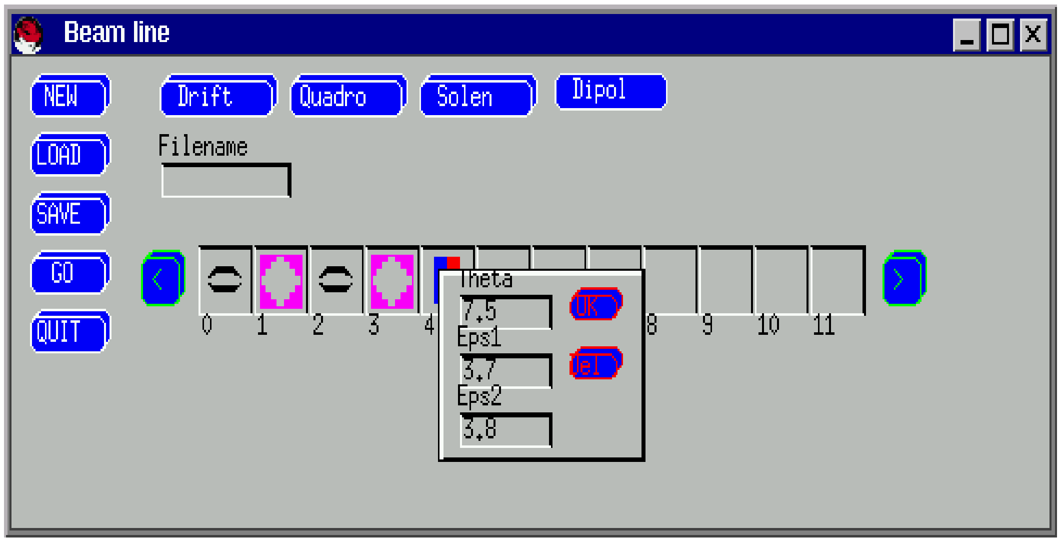

Fig.2. Example of control menu.