Lecture 2 : Introduction to EDWIN

Later in the course, there will be a study of graphics packages in general, and the different criteria which can be used to compare them. The next two lectures consist of a study of the EDWIN package in detail, which will be the basis of the course practical. EDWIN is available on all the local machines, and has been exported to a number of other sites world wide. The EDWIN design aim was maximum functionality for minimum amount of code. EDWIN was designed in the days when the Department had a large number of small machines, each with different pieces of graphics hardware, and all had different software for them. Consequently EDWIN restricted itself to a 2-D coordinate system, with no picture structuring or menus, as these can be provided by user level programs, and there inclusion would have prevented EDWIN fitting on the smaller machines. EDWIN is device, machine and language independent, and provides a standard, simple interface for local programmers.

The structure of EDWIN is shown on the back page of these notes.

The remainder of these notes is a description of the basic procedures in detail.

1) Controlling EDWIN :

PROCEDURE InitialiseFor (dev : integer)

This is normally the first EDWIN procedure to be called. The value of DEV chooses which device driver is to be used to send graphics instructions to the terminal being used. Some typical values of DEV are -

4006 4010 4012 4014	: Tektronix storage tubes
5555	: Data-Type X5A terminals
550	: Perkin-Elmer 550 terminals
200	: Visual 200 terminals
7220	: HP A3 flat bed plotter (GP15)
300	: Printronix printers (LP15,LP41,LP78)
936	: Calcomp drum plotter (GP23)
7221	: HP A3 flat bed plotters (on Vax)
7475	: HP A3 plotter (on Vax ?)
7580	: HP A1 plotter (on Vax)
67	: 'Charles' terminals (on Vax)

When this procedure is called all the attributes are reset to the defaults, and the coordinate space which is mapped onto the screen is set to be 0 to 1023 by 0 to 1023. (The default 'window'.)

Any system specific action to place the terminal line into graphics mode is done at this time.

PROCEDURE TerminateEdwin

This must be the last EDWIN procedure to be called. It tidies up by sending any pending graphics output to the device, and ensures that the device is reset to its normal state, ready to be used again.

Examples of this are moving the paper that has been used past the pen on a plotter, so that the next job can't write over it by mistake, and reseting interactive terminals such as the Tektronix, Charles and X5A to be in text mode.

PROCEDURE NewFrame

In the case of an interactive terminal this procedure clears the screen, and for plotters this instructs the plotter that this is the start of a new plot. It should always be used before any graphics output is done, and may be used at any point during the program to start a new picture. Note that in the case of GP15 the ERCC only allow 1 picture (ie. 1 call to newframe) in a file, (any more pictures results in only the first one being plotted).

FUNCTION DefaultDevice : INTEGER

This function returns a number which is a 'best guess' at the type of terminal to use, depending on the current terminal line. On EMAS this uses the setting of the command TERMINAL TYPE, and on VAX/VMS the type is derived from the information provided by SHOW TERM. In the case of VAX network terminals, users must set their terminal type by one of the following commands, (otherwise the function will return the result 0).

The result of DEFAULT DEVICE can always be 'forced' by the statement

```
$ ASSIGN 4012 EDWIN_DEFAULT_DEVICE        { on VMS }
```

2) Basic drawing procedures :

All the following procedures can be used to produce pictures on the graphics terminal that was selected when 'InitialiseFor' was called. (Note that if 'InitialiseFor' was not called, the default is that a 'null' graphics device is selected, and consequently these procedures will have no effect). As a drawing proceeds, there is a point, called the current position, which is updated after each of the following procedures is called. The effect of the procedures is described using the point (CX, CY) which is the current position.

PROCEDURE MoveAbs (x, y : integer)

This procedure updates the current position, but no drawing occurs. ie. the new current position is the point (X, Y).

PROCEDURE MoveRel (dx, dy : integer)

This procedure updates the current position, in this case relative to the old position. ie. the new current position is the point (CX+DX, CY+DY).

PROCEDURE LineAbs (x, y : integer)

This procedure draws a line from the point (CX, CY) to the point (X, Y).

PROCEDURE LineRel (dx, dy : integer)

This procedure draws a line from the point (CX, CY) to the point (CX+DX, CY+DY) and the current position is updated.

PROCEDURE Character (ch.: char)

This procedure draws a character whose lower left corner is at the current position, and after the character has been drawn, the current position is updated to (CX+SX, CY) where SX is the size of the characters. Characters can either be generated by the device (the default state), or can be generated as a sequence of lines by EDWIN. In the first case these can sometimes only be a fixed size, but EDWIN generated characters can be any size and in a number of different fonts. Run EDWINDIR:CHARS on VAX on the 7221 plotters to get two plots of the GIMMS fonts.

3) Procedures for changing device attributes :

Analagous to the fact that a graphics package has a known current position at any time, which can affect the rest of the drawing, there are also a number of attributes associated with the package. If any of these are changed then they remain in effect until they are set to a new value, or the package is re-initialised. If an attempt is made to set an EDWIN attribute which is not available on the device or it is out of the range of valid parameters for the attribute then the default attribute is used.

PROCEDURE SetColour (colour : integer)

This procedure chooses the colour of pen which a plotter is to use, or the colour of lines in the case of a colour display. The normal range of colours is:

1 for Black (default)	5 for Purple
2 for Blue	6 for Orange
3 for Green	7 for Lime Green
4 for Red	8 for Brown

Some devices such as the HP plotters have all pens defined, on others fewer are avilable. The colour affects both the colour of lines and the colour of characters which are drawn after the colour is set.

PROCEDURE SetLineStyle (style : integer)

This procedure selects one of a number of line styles, which affect lines which are drawn after a style is set. (Characters are always drawn with solid lines). The range of values is:

0 for normal lines (default)	————————
1 for dotted lines	.....
2 for chain lines (not on HP's)	—— . ——— . —
3 for short dash lines	- - - - - - - - - - - -
4 for long dash lines	— — — — — —

PROCEDURE SetCharQuality (qual : integer)

This procedure decides whether character draw are to be generated by the device hardware, or by EDWIN. A value of 0, (the default), selects device characters, and 1 selects EDWIN generated characters.

The following limitations exist on device character sizes -

4002 4006 4010 4012 200 550 :	Only 1 size available
4014 X5A :	Only 4 sizes available
Charles :	Only 3 sizes available
Plotters :	Any size available

PROCEDURE SetCharFont (font : integer)

This allows users to select either one of a terminals hardware character sets in the case of plotters with about 6 character sets, (this occurs if character of quality 0 are drawn after a font has been selected). If character of quality one are being drawn then this selects either the default EDWIN software character set (fixed character pitch, with limited rotation) if the font is 0, or one of the quality character sets which are described in the EDWIN release notes.

PROCEDURE SetCharSize (size : integer)

All the characters are drawn in a rectangle, which by default is 12 units in X by 20 units in Y. This procedURE SetS the X dimension of the character box, and the Y direction is scaled in proportion. If the device can't draw characters of the size requested, then the nearest size available is used.

PROCEDURE SetCharRot (angle : integer)

Characters may be rotated. In the case of software characters which font 0, they can only be rotated in steps of 90 degrees, but for software character in other fonts, and hardware characters on most plotters this can be any angle. On most interactive displays hardware text cannot be rotated. The default is that text is not rotated, ie. angle = 0.

The structure of a typical Pascal program using EDWIN :

(Examples of Imp programs can be found in the EDWIN Users Guide).

```
PROGRAM name (INPUT, OUTPUT);
   { who why what when }
VAR
   { declare some variables }
   { declare the EDWIN procedures as EXTERN's }
   { any user written procedures }
BEGIN
   { set up some initial states };
   InitialiseFor (DEV);
   { solve the problem };
   { Tidy up.  (eg. give answer) };
   TerminateEdwin;
END.
```

An Example program using EDWIN :

This program is designed to run using the APM Pascal compiler. If the program is to be run on EMAS, the '%include' statement should be removed and the file ECSLIB.EDWIN_PAS should be edited into the program at that point in the file.

```
PROGRAM boxes (input, output);

  { Program to draw a set of boxes,   JGH   April 1982 }

CONST   L = 100;        { Length of box }
        W = 100;        { Width of box }
        Black = 1;
        Blue = 2;

VAR i, j : integer;

%INCLUDE 'edwin:specs.pas'

PROCEDURE box (x, y, l, w : integer);
BEGIN
   MoveAbs (x, y);
   SetColour (blue);
   LineRel (l, 0);   LineRel (0, w);
   LineRel (-l, 0);  LineRel (0, -w);
   SetColour (black);
END;

BEGIN
   InitialiseFor (default device);
   NewFrame;
   FOR i := 1 TO 4 DO BEGIN
      FOR j := 1 TO 4 DO BEGIN
         box (j * 200 - 100, i * 200, l, w);
      END;
   END;
   TerminateEdwin;
END.
```

Lecture 4 : EDWIN revisited

PROCEDURE StoreOn (stream : integer)
PROCEDURE StoreOff

If storing is enabled then a record of the graphics commands sent to the device is recorded in the file which is defined on the stream given as a parameter. This can then be redrawn at a later date by the command VIEWPDF which was mentioned before. The default is that storing is off. The EMAS command DEFINE stream,file must be given before storing is enabled. This procedure cannot be used from VMS Pascal at present.

PROCEDURE ViewOn (stream : integer)
PROCEDURE ViewOff

These procedures allow the drawing of the picture to be suppressed which allows a display file to be created by use of the 'store' procedures without the need to redraw the picture. The default is that the picture will be drawn on stream 0. In the case of output to files (eg. Printronix or Calcomp plotters), users must make a call to 'view_on' BEFORE they call 'initialise_for' otherwise the initialisation sequence will not be output!

There is a module of Procedures using EDWIN which are designed to draw geometric shapes. There is nothing special about these procedures, as any user may have written them, but they are available to help reduce the effort required to produce a graphics program. The procedures also allow the use of any device specific features, such as hardware shading or hardware circle drawing if this is available.

The procedures for drawing are -

PROCEDURE Rectangle (LowX, LowY, HighX, HighY : integer)

This draws a rectangle between the two corners specified (in World coordinates). If the device has a hardware fill primitive, then this procedure will generate a shaded rectangle, (see SetShadeMode below).

PROCEDURE Box (L, W : integer; VAR C, D : pointfm)

This draws a general box of length L and width W, with centre point C. The angle of the box can be specified by the direction vector D. This procedure is useful in VLSI applications, but probably not many others!

PROCEDURE Circle (Radius : integer)

This draws a circle centred on the current position, with the radius given as a parameter. In the case of the HP plotters, this uses the hardware circle generator. The default is that the circle will be drawn with 15 degree chords, although the user can specify more accurate circles using the procedure SetChordStep which is also described below.

PROCEDURE Arc (Ox, Oy, Radius, StartAng, EndAng : integer)

This procedure draw an arc (line) centred at OX, OY of radius RAD, between the start angle and end angle specified. The relative directions of the angles determines the direction of the arc. The tolerance of the arc can be chosen by SET CHORD STEP procedure.

PROCEDURE Sector (Ox, Oy, Radius, StartAng, EndAng : integer)

This procedure is similar to ARC, except the area traced by the arc, and the centre point is treated as a closed polygon, and will be shaded if shade mode is set to be solid.

PROCEDURE Polygon (NumPts : integer, PointArray : pointa)

This draws a polygon, the points being specified by the array of coordinates which is given as a parameter. The polygon is shaded if shading mode is set, and is clipped if it crosses the edge of the viewport.

PROCEDURE SetShadeMode (mode : integer)

If MODE is 0 then shapes are not shaded (only the outline is drawn). This is the default state. If MODE is 1 (and the terminal has a hardware shading facility) then the shapes are shaded.

PROCEDURE SetChordStep (step : integer)

This chooses the chord step which is used when drawing circles. The default size of step is 15 degrees, but users can specify a smaller step size to get more accurate circles.

The following example shows how the procedure POLYGON can be used. Note that the user must choose the maximum size of polygon which can be drawn when they write their program. It should also be noted that the program is only to show how POLYGON can be used, and for the example given it would be very much more efficient to use RECTANGLE!

```
program test (input, output);

type pointfm = record
                  x, y : integer;
               end;

     pointa = array [0..10] of pointfm;
     { User must choose max size at compile time }

var p : pointa;
    i : integer;

%include 'Edwin:Specs.pas'
%include 'Edwin:Shapes.pas'

begin
   InitialiseFor (DefaultDevice);
   Newframe;
   p[1].x := 250;   p[1].y := 250;
   p[2].x := 750;   p[2].y := 250;
   p[3].x := 750;   p[3].y := 750;
   p[4].x := 250;   p[4].y := 750;
   Polygon (4, p);
TerminateEdwin;
end.
```

The following procedures are provided in EDWIN for the user to control the window and viewport which controls the appearance of the picture.

PROCEDURE Window (lowx, highx, lowy, highy : integer)

This procedure sets up a window in the world space. The default value of 0 to 1023 is set when 'initialise_for' is called. By altering the window the picture can be enlarged or contracted on the screen.

PROCEDURE InquireWindow (VAR lowx, highx, lowy, highy : integer)

This procedure gives the values which were used at the last call of 'window'. (Probably not of much use for the CS2 practical).

PROCEDURE Viewport (lowx, highx, lowy, highy : integer)

This procedure sets a view port on the graphics terminal being used. The default is that the whole screen is used on an interactive terminal, and paper size of A4 is used on a plotter. If an attempt is made to set a size which is larger than the terminal can support, the largest size of viewport is used.

PROCEDURE InquireViewport (VAR lowx, highx, lowy, highy : integer)

This procedure gives the values of the current viewport settings.

PROCEDURE AspectRatioing (flag : integer)

If a square picture is required to be drawn on a rectangular viewport, the picture would be distorted. The default is that aspect ratioing is enabled, and the window is enlarged to make the picture have the correct aspect ratio. If distorted pictures are required it may be turned off. Flag = 0 is off, and Flag = 1 is on.

Another module which is available in the EDWIN library (again currently only on APMs and VAX) is a set of procedures to generate matrices which represent two dimensional transformations, and to maintain a stack of these.

The following procedures are available to create transformation records -

PROCEDURE UnityTransform (VAR T : Transfm)

This procedure creates a unit transformation :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

PROCEDURE TranslateTransform (X, Y : real; VAR T : Transfm)

This procedure create a translation transformation :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X & Y & 1 \end{pmatrix}$$

PROCEDURE MirrorXTransform (VAR T : Transfm)

This procedure create a transformation which is a mirroring of the X coordinates, ie. a mirroring of the picture in the Y axis.

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

PROCEDURE MirrorYTransform (VAR T : Transfm)

This procedure create a transformation which is a mirroring of the Y coordinates, ie. a mirroring of the picture in the X axis.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

PROCEDURE ScaleTransform (XS, YS : real; VAR T : Transfm)

This procedure creates a transformation which is a scale factor for the X and Y coordinates.

$$\begin{pmatrix} XS & 0 & 0 \\ 0 & YS & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

PROCEDURE RotDvTransform (A, B : real; VAR T : Transfm)

This procedure create a transformation which is a rotation specifed by the direction vector (A, B). Note that D = SQRT (A*A+B*B)

$$\begin{pmatrix} A/D & B/D & 0 \\ -B/D & A/D & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

PROCEDURE RotAngTransform (ANGL : real; VAR T : Transfm)

This procedure creates a transformation which is an anit-clockwise rotation of the angle specifed.

$$\begin{pmatrix} Cos\ A & Sin\ A & 0 \\ -Sin\ A & Cos\ A & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

PROCEDURE ComposeTransform (VAR TA, TB, TC : Transfm)

This procedure takes two transformations TA and TB and uses matrix multiplication to generate the composite matrix TC. Remember that the order of the parameters TA and TB is critical!

The following procedures can be used to manipulate the transformation stack. Note that this is a indepenent modules from the main part of EDWIN, and if users are using it they must transform any points explicitly before any EDWIN procedures are called to draw the resulting transformed picture.

PROCEDURE InitTransform

This procedure can be used to reset the stack to the 'empty stack'. This intialises the stack by adding a unit matrix as the first element of the stack. It is automatically called of the user does not call it explicitly.

PROCEDURE PushTransform (VAR T : Transfm)

This procedure takes the transformation which is specifed in the parameter, and composes this with the current transformation which is on the top of the transformation stack. This new resulting transformation is added as the top element of the transformation stack. An error is signalled if the stack overflows, (currently 100 elements).

PROCEDURE SetTransform (VAR T : Transfm)

This procedure is similar to PushTransform, except that the matrix specifed is added to the top of the stack without composing it with the current top transformation on the stack.

PROCEDURE PopTransform

This procedure deletes the current top transformation on the transformation stack, (which results in the second top transformation becoming the top one, etc.). An error is signalled if an attempt is made to pop the stack when it is empty.

PROCEDURE GetTransform (VAR T : Transfm)

This procedure can be used to take a copy of the current top of the stack. Its main use would be to call SetTransform at a later point during the run of the program.

The following procedures can be used to transform points, vectors and boxes with respect to the top of the transformation stack. Note that in the case of transforming a vector there is no need to worry about any resulting translations. The box (typically a bounding Box) is specifed by two points which represent the opposite corners.

PROCEDURE PointTransform (VAR P, NP : Pointfm)
PROCEDURE VectorTransform (VAR V, NV : Pointfm)
PROCEDURE BbTransform (VAR OPL, OPU, NPL, NPU : Pointfm)

The following example program shows how the transformation stack can be used to produce a picture consisting of one line drawn at a number of different orientations around one of its end points.

```
program txdemo (input, output);

Type

   %include 'edwin:typcs.pas'      { Defines 'transfm' and 'pointfm' }

var      i : integer;
        tx : transfm;
     p, np : pointfm;

   %include 'edwin:specs.pas'
   %include 'edwin:txstack.pas'

begin
   P.X := 250;   P.Y := 0;
   InitialiseFor (DefaultDevice);
   Newframe;

   I := 0;
   InitTransform;
   while I <= 360 do begin
      RotAngTransform (i, tx);
      PushTransform (tx);
      PointTransform (p, np);
      PopTransform;
      MoveAbs (500, 500);
      LineAbs (500+np.x, 500+np.y);
      I := I + 15;
   end;

   TerminateEdwin;
end.
```