1996-10-31 Bo Nordin (bosse&cb.uu.se)

Lecture notes, computer graphics (F03) (Foley 2, some from Foley 7)

Software overview

We have seen an application (AVS) and talked about hardware, today we will see what the person who designs/writes an application needs (in general terms). Later specific standards such as GKS/PHIGS. May also serve as a repetition or introduction to C/C++.

Input/output *primitives* (used to create a graphical *scene*).

API (application programmer's interface) - standards later

today: what do we (the programmer of *applications*) need.
overview:

Application <-> functional <-> procedural <-> device level <-> hardware.

AVS, IMP GKS, PHIGS CGI (VDI, soft->hard)

(CGM - computer graphics metafile (earlier VDM) several images in one file
CGI - computer graphics interface (command interpreter (dev-ind), drivers (dev-.dep)))

I will describe functions that exist in most packages. Syntax may differ, (language-dependent, for instance) (I use C with some C++ extensions). But mainy procedural, not object-oriented.

Output - commands that write graphics (will show 2D, 3D simple extensions)
Input - interaction
Pixel manipulation (raster-only)

Raster

+: area fill, patten fill, allows manipulation of detail (pixel), images!

SRGP - device-independent (but screen/window coordinates). Screen coordinate system.
output refs: (QuickDraw, XLib) ------ input refs (PHIGS)

Warning: the book says a more complete reference manual with the packages.
...and the files in the packages says the book is the complete reference work

Raster - grid points - limitations (lines only between grid points). Cartesian ^ > (but X...)
resolution
We'll assume that primitives are mapped to the whole screen (or a window - virtual screen).

Data types needed

int,
point (defPoint) typedef struct { int x,y; } point;
rect (def_Retangle) typedef struct { point bottomLeft, point topRight; } rect;

Output primitives:

Lines (everything may be created from lines)

```
line( int x1, int y1, int x2, int y2): ex line(100,100, 200, 300);
line( point x1, point x2);  /* point is a struct */
polyLine( int count, int *x, int *y); /* several connected line segments */
polyLine( int count, point *p);
```

(** segment(int count, int *segment); don't mention, what's this?? **)
point(int x, int y);
polyPoint(int count, point *p)
Versions for ints (with Coord added to the name), point
ex: histogram 256 values;

```
#define SIZE 4  /* so scaling of either axis !! */
/* also discuss discretization problem: (what's zero and what's 1)

void drawHistogram( int *vec, int size) {  /* assume right-handed Cartesian *./
  point axes[] = { {10, 100}, (10, 10}, {300, 10} };
  point *hist = (point *) malloc (sizeof(point) * (size  * 3 + 1));
  int i, j;
  hist[0] = axes[1];
  for (i=0,j=1; i < size; i++,j++) {
     hist[j].y = vec[i];
     hist[j++].x = i * SIZE;
     hist[j].y = vec[i];
     hist[j++].x = (i  +1) * SIZE:
     hist[j].y = 0;
     hist[j++].x = (i+1) * SIZE;
  }
 polyLine( 3, axes );
 polyLine( size*3 + 1, hist);
}
```

Markers:
marker(int x, int y);
marker(point *p):
polyMarker(int count, int *x, int *y);
polyMarker(int count, point *p);

Example: scatterogram

Polygons (may also use polyline, but closes automatically):
polygon(int count, point *p);
rectangle(point bottomLeft, point topRight); /* why special ?? */
rectangle(rect *r);

Circles, ellipses:
ellipseArc(rectangle *extent, float startAngle, float stopAngle); /* bounding box,
counter-clockwise "rectangular" degrees */

Filled area:
filledPolygon(...) (odd-crossing rule, note: do on primitives, not frame buffer)
filledEllipseArc(...) /* aka pie slice */
filledRectangle(...)

Text;
text(point *origin, char *text); /* origin, aka anchor point baseline */

GDP (generalized drawing primitive)

Attributes (**aspects**)

Most primitives may be displayed in various shapes and colours:

Modal (global variables that retain their value until explicitly changed - primitives are drawn with attributes in effect when drawn, no effect on previously drawn

(otherwise lots of parameters)

(one could have a list of attributes stored with each primitive, *bundled*)

Defaults exist (but explicit settings are better, save, restore)

(Poly)Lines, ellipses, polygons, rectangles:

```
        lineStyle( int ); /* constants: CONTINUOUS, DASHED, DOTTED, etc. */
                          /*write-enable mask   00001110000111 */
int     setLineStyle( bitmap *bm); /* create your own */
        lineWidth( int ); /* often pixels */
```

Marker:

```
        markerSize(int size);     /* often length of the size of the extent */
        markerStyle( int mstyle); /* MARKER_CIRCLE, MARKER_SQUARE, etc. */
```

General colour attribute (indexed cf. bundled)

Depends heavily on the hardware (at least 0/1)

```
      inquireDepth(void);
      colour( int ix);  /* through a colour table , 2**d, d is depth  dynamic changing */
                        /* colour names may exist: COLOR_BLACK, COLOUR_WHITE */
                        /* note that
        loadColorTable(int ix,  int r, int g, int b);  /* may load several?? */
```

Areas can be outline/filled

```
        fillStyle( int ) /* SOLID - current value of the colour index
                    BITMAP_OPAQUE - regular non-solid, small bitmap, tiled (rep.) stippled
                    BITMAP_TRANSPARENT - regular non-solid write-enable mask
                    PIXMAP -opaque (any depth)
                    */
        setFillBitmapPattern( int patternIndex );
int     defineBitmapPattern( bitmap *map); /* depth 1 */
        backgroundColor( int ); /* not screen background, used in BITMAP_OPAQUE */
        setFillPixmapPattern( int patternIndex );  /* also called cellArray */
```

For ellipses:

```
       arcMode( int );    /* ArcChord, ArcPieSlice */

        inquireAttributes( attributes *a); /* access to fields, allows selective restoration */
        setAttributes( attributes *a);
```

Text attributes: font (Times, Helvetica), size (), appearance (roman, *italic*, **bold**).

And combinations....

simple: fixed-size bitmap

in addition intercharacter spacing, spacing between lines, angle of the line.

usage Annotations, menus

```
       inquireTextExtent( char *text, int *width, int *height, int *descent);
```

Interaction/input primitives

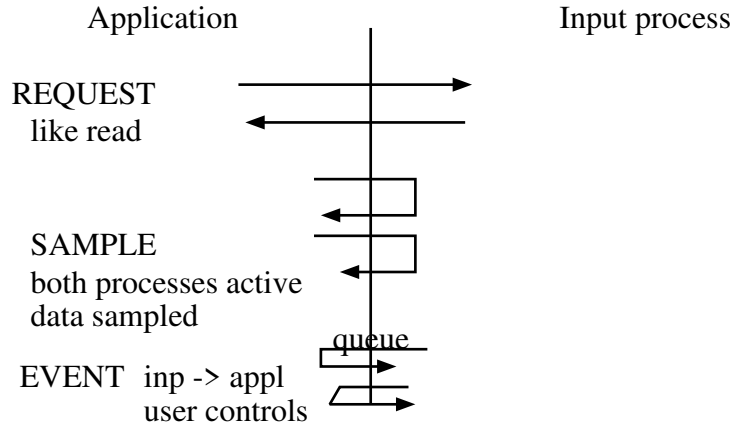
HCI, General guidelines:
simple/consistent
not too many choices
show options clearly -- grey out unavailable options
appropriate feedback -- highlight current choice (inverse video, framed, shaded) *echo*
different cursor
graceful error recovery -- undo/redo

Logical input devices:

locator (pos. in wc)	physical: mouse
stroke (sequence of positions)	mouse
choice (alternative, menu)	mouse/menu
valuator (float)	slider/control dial, scrollbar
pick (identifies an object)	mouse
string (a character string)	keyboard

echo (show characters, move a cursor)
prompt (appearance of cross-hairs, change of cursor, appearance of a text window)

Input devices can be used in one of three modes:



Typical workstations have one or several devices of each type
ex 3 locators (one per mouse button)

Request: input process, waiting for the user to act/reply:

```
typedef struct {
 point position;
 enum { UP, DOWN } button[NUMBUTTONS];
}

void requestPos( point *pos ) {
  locatorMeasure p;
  request( LOCATOR, p, 1000);  /* waits until the user presses a button 1000 - timeout*/
  pos = p.position;
}
```

Devices can be activated/deactivated:
setinputMode(int device, int mode); /* SAMPLE/EVENT/NONE */

Sampling (polling) vs. event-driven processing (in both cases - the application pgm and the input process are both active simultaneously).
sampling: the application pgm is in control, input in bgr always providing current data
event: the input process is in control: delivers data & puts it in a queue

in sampling the programmer reads values (often at maximum speed). very CPU intensive

setInputMode(LOCATOR, SAMPLE);

```
void drawOutline() {
  locatorMeasure p;
  point pos;
  int i;
  do  sample( LOCATOR, p ) while ( p.button[0] == UP );  /* NOTE! kills the machine */
  pos = p.position;
  do {
     sample( LOCATOR, &p );
     line( pos, p.position);
     pos = p.position;
  } while (p.button[0] == DOWN);
}
```

/* problems: lines become longer if the operator moves the mouse faster */

instead several devices are enabled for input and then waits for *interrupts.*
event-driven interaction is a synchronous simulation of interrupt-driven interaction
(this means that the package monitors the devices and stores info about events in a queue and the application is an event-driven loop:

device waitEvent(int timeout);

change previous example:

```
setInputMode( LOCATOR, EVENT );
device dev;
dev = waitEvent( 10000 );
if (dev == LOCATOR ) {
   setInputMode( LOCATOR, SAMPLE );
   drawOutline();  /* only the sample mode */.
   setInputMode( LOCATOR, EVENT );
}
```

keyboardProcessingMode(EDIT / RAW); /* RAW: char-by-char WYSIWYG */

Trigger by pressing a key or a mouse button.

getValue(LOCATOR, locatorMeasure *measure);/* called when waitEvent returns LOCATOR */

the most important feature of event mode: waiting for several events *simultaneously*:

```
typedef struct { int type; union ; } event;
event ev;
 int quit = 0;
  do {
   getevent( &ev );
   switch (ev->type) {
    1:.....break;  /*   KEYBOARD  (if in raw mode) if 'q' quit = TRUE;
    2:.....break;  /* LOCATOR: getValue( LOCATOR, p);DrawCircle( p.position ); */
   }
 }
 while (!quit);
```

event-loop is strictly *sequential*, user <-> computer. mouse-ahead (events should *echo*!!)
future: *parallel* conversation.
think time dominates (dream time??) also in fast-paced games no CPU time if true

input devices may also have attributes:
locatorEchoType() /* NONE/CURSOR/RUBBER_LINE/etc./ */
buttonMask(int); /* define active buttons */

input devices maybe controlled:
setLocatorMeasure(point):

pick correlation. - the process of finding higher-level structures. (menus, etc.)

Raster Graphics

Complex menus: create them in memory and then copy them to the screen (frame buffer) this in order to make them appear/disappear quickly.
In raster graphics this is simple: create them in off-screen bitmaps/pixmaps. (cf. buffering) called *canvases* Moving blocks of pixels back and forth is fast (given a copyPixel operation). Each canvas has its own attributes. In this simple example we assume that the package always uses the bit depth of the hardware.

```
 int createCanvas( int w, int h); /* returns an identifier: addressed 0-(w-1) & 0-(h-1)
     useCanvas( int canvasID );  /* NOT displayed!! */
     deleteCanvas( int canvasID );
     rectangle inquireCanvasExtent( int canvasID );
```

Example: pulldownmenu (Foley 51) (if time do it in detail using funcs above:)

Clipping rectangles: protection and window->viewport transformation:

```
 clipRectangle( rectangle clip);
 rectangle inqClipRectangle();
```

copyPixel (called *bitBlt* - bit block transfer, or pixBlt)
first in microcode on ALTO early 1970's
Example: pulldownmenu (Foley 51) (if time do it in detail using funcs above:)

0. create menu body canvas
1. when button down highlight the menu header
2. calc menu extent & save what's shown now
3. copy menu body from canvas (change ClipRectangle to the whole screen!)
4. wait for button up (highlight!), get locator measure
5. restore
6. calculate choice from pos.

```
int performPullldownInteraction( int menuID ) { /* menu is a canvas */
  int w, h , saved, choice;
  point pos;
  rectangle r;
  point despos = {0, 0};
  active = inquireActiveCanvas();
  inquireCanvasSize( menuID, &w, &h );
  r  = getMenuExtent( menuID );
  saved = createCanvas( w, h );
  useCanvas( saved );
  copyPixel( SCREEN_CANVAS, r,  destpos ):
  useCanvas( SCREEN_CANVAS );
  copyPixel( menuID, inquireCanvasExtent(menuID), r.bottomLeft );
   /* wait for interaction and calculate pos */
  copyPixel( saved, inquireCanvasExtent( saved), r.bottomLeft ):
  deleteCanvas( saved );
  useCanvas( active );
  return choice;
}
```

Write Mode / RasterOp

D = S op D (op is one of *replace, or, xor, and*). op is called RasterOp
RasterOps are done on the pixel values, which are *indices*, not the actual colours.
xor is used by window managers for moving windows, etc. AVS bounding box !!

Rubber-rectangle example:

```
MoveWindow ( point from, point w)  /* called when the button is pressed */
 rectangle r;
 locatorMeasure p;
  point pos;
  int i;
  writeMode ( XOR );
  pos = p.position;
  rectangle ( r = defRectangle( pos.x, pos.y, pos.x+w.width, posy+w.height) );
  do {
     sample( LOCATOR, &p );
      if ((p.position.x != pos.x) || (p.position.y != pos.y)) {
        rectangle ( r );
        rectangle( r = defRectangle() );
       }
     pos = p.position;
  } while (p.button[0] == DOWN);
  rectangle ( r ); /* remove the last */
}
```

Other issues

No Object-level operations (cf. AVS!)
No application ("world") coordinate systems
Storage of primitives for respecification/refresh or transformations or hardcopy
Immediate mode vs. *retained* mode.

structures - reminiscent of programming languages. (aka segments)
post, unpost, delete
transformations.

Subroutine package vs. *interpreted languages* such as PostScript.