PHILIPS

PHILIPS INTERACTIVE MEDIA of America

Technical Note #77

Implementing a Compass Cursor in CD-I

Kevin Hunt and Doug Yoon

June 1, 1992

This note lays out a method for implementing a compass cursor (frequently used in traditional arcade games) in CD-I. The method described here relies heavily on the Balboa System Cursor/Hotspot Manager. Thus, familiarity with the Balboa System is required for successful implementation of the compass cursor described here.

Implementing a Compass Cursor in CD-I

Kevin Hunt and Doug Yoon

Why Use a Compass Cursor?

Mice and roller ball controllers are perfectly suited for moving a cursor around an area of the screen and for making arbitrary selections and moving objects, but they are inadequate as directional controllers. Joysticks, on the other hand, are designed as directional controllers—perfect for directing Pac-men (and women) around the screen, flying aircraft, or getting Mario past one more bad guy. However, joysticks perform abysmally as position indicators. The hand remote controller is a hybrid—it is a joystick that imitates a mouse: it converts directional information into positioning information (and really serves neither use well).

For games where directional control is more important than position control—for instance in a Nintendo style title—it is very helpful to have the thumbstick controller act as it should. Enter the compass cursor! It provides a method for converting cursor position information into direction information. The term "compass" applies to the way the directional information is returned—as one of eight points on a compass.

In concept, a compass cursor interface is easy to implement: put the cursor somewhere and poll its position until it's moved; derive the directional information from its new position; and—depending on the platform—either reposition the cursor back to "somewhere;" or save the new position as the starting point, and repeat the entire process.

With the thumbstick controller, the above process works relatively well, but it breaks down when a mouse or roller ball is used. For example, using a thumbstick, the player can make Zelda walk in a constant direction because the cursor is constantly moving (if the thumbstick is pointed in the same direction), causing a continual re-evaluation of the direction. A mouse must be constantly moved to give the same effect (making it a bit tiring to play the game). Joysticks also auto-center—the player lets go, the stick springs back to center, and Zelda stops.

Processor load is yet another problem with this approach (especially with respect to CD-I). The cursor, once moved, must be repositioned using **pt_coord()**, or Balboa's **vs(ve)_cp_position()**. If this is done every time the cursor is moved, it causes considerable load on the processor; if it isn't, accuracy and "crispness" are lost.

Based on suggestions by Drew Topel here at PIMA, and Lee Barnes at Viridis, I've designed a compass cursor interface that solves most of the problems and incurs as little load as possible. This interface installs a 3 x 3 hotspot grid and sets the cursor position in the middle square. The cursor is then constrained (by **ve_cp_rect()**) to the 3x3 grid. When the cursor moves from the center into one of the other squares, the Balboa System determines the square entered. This is now the compass direction. The cursor is not repositioned until it becomes idle, and no new processing occurs until the cursor moves into a different square.

Some of the problems cited above still exist when you use a mouse or roller ball with a title, but these can be dealt with via decisions and/or compromises made during the design phase of a title.

How Does the Compass Cursor Work?

Instead of finding the absolute coordinate position for the cursor, the compass cursor (also referred to here as CCURSOR) returns information about the direction the pointing device is moving. The CCURSOR detects user movement of the pointer in eight directions—north, south, east, west, northwest, southwest, southeast, and northeast. See Figure 1 below.

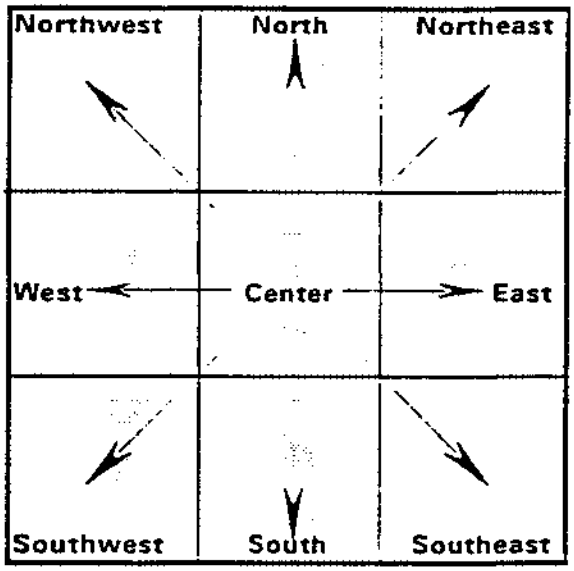

Figure 1: The Compass Cursor Grid

The compass cursor described here relies solely on the Balboa System Cursor/HotSpot Manager. Therefore, familiarity with the Balboa System, its structure, and its conventions is required for successful implementation of this cursor.

Centering and Sensitivity

There are two major concerns in implementing a compass cursor. The first is the ability to "center" the cursor in a timely manner, and the second is to set its sensitivity. Centering the cursor is crucial to the performance of a compass cursor, because the user should be able to "stick" infinitely in any direction. This centering, when done through a normal UCM call, is a time-consuming process that yields a sluggish compass cursor

and considerable processor overhead. Judicious use of Balboa library functions has solved this problem.

The second problem, sensitivity, is managed by setting a specific number of pixels that the cursor must travel before the direction of the CCURSOR is determined.

The CCURSOR Structure

The CCURSOR is managed through one structure and one function call. The structure definition follows:

```
typedef struct {
        VIDENV    *videnv;
        short org_x,
                  org_y,
                  *x_dir,
                  *y_dir,
                  *compass_dir,
                  sensitivity,
                  cur_idle;
        CALLBACK  user_thread,
                  button_func,
                  sleep_func,
                  wake_func;
        int       sleep_ticks;
        HOTSPOT   compass_tree[COMPASS_TREE_SIZE];
} COMPASS_CURSOR_CONTROL;
```

The following are definitions for elements of the CCURSOR structure:

videnv	A pointer to the video environment obtained from the Balboa call **ve_open ()** or generated through **vs_open**.
org_x, org_y	These are UCM coordinates that refer to the top-left corner of the compass cursor grid. Setting x or y to 0 (zero) gives you the default for settings for **org_x** (10) or **org_y** (50). (See Figure 2 below.)

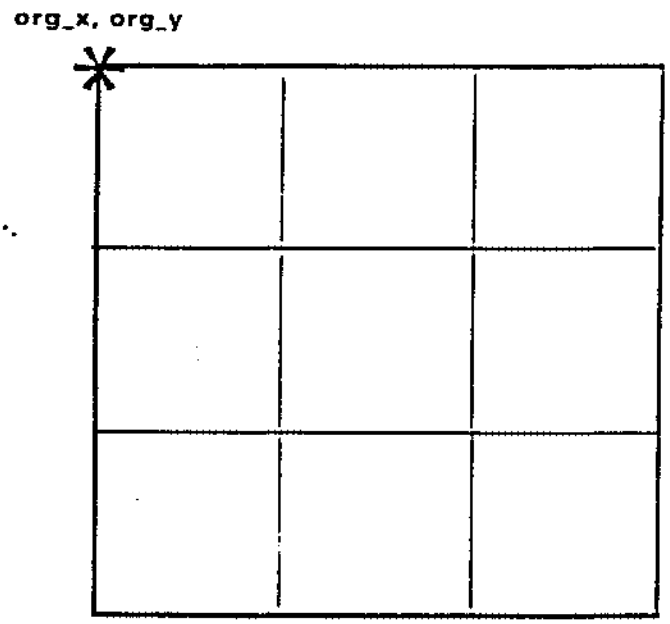

Figure 2: Compass Cursor Grid with Origin Shown

x_dir, y_dir

These are pointers to short integers. They should be static globals for use by the **user_thread** function. The two variables are defined as follow:

Values for x:

-1	left
0	center
1	right

Values for y:

-1	top
0	center
1	bottom

compass_dir

This is pointer similar to that for the **x_dir** and **y_dir** fields. This field points to a short integer into which a value from 1 through 9 is loaded. The values

	correspond to the locations designated on the compass grid. (See Figure 2 above.)
sensitivity	This is the size of each quadrant in pixels. It represents the number of pixels the cursor must travel before the direction of the cursor is determined in UCM coordinates.
cur_idle	For internal use only.
user_thread	This is a pointer to a function that is called upon installation of the CCURSOR hotspot grid. This is the software engineer's main line loop and, thus, should be self-dispatching.
button_func	This is a pointer to a function that is called when either button #1 or button #2 is pressed on the controller. This function is called only when the CCURSOR is centered. (This is easily modified to allow button presses while the pointing device is not centered.)
sleep_func	This is a pointer to a function that is called after **sleep_ticks** number of ticks goes by (100 milliseconds per tick).*
wake_func	This is a pointer to a function that is called when the cursor process wakes up.*
sleep_ticks	This is the amount of time to wait before calling the **sleep_func**. This timeout is expressed in 100 milliseconds.
compass_tree	For internal use only.

```
install_compass_cursor  (int  context,
COMPASS_CURSOR_CONTROL  *ccursor);
```

---

* See the *Balboa Run-Time Environment Reference Guide*, Volume II, pp.30-16 through 30-17, vs_cp_idle_function() for more information.

install_compass_cursor	This function installs and activates the compass cursor and dispatches the Callback **user_thread**. The first parameter is the context, which is a Balboa Callback convention, and the second parameter is the pointer to the data structure.

More Information

For the source code to this implementation of the compass cursor, please call or e-mail Sandeep Bhatia at Philips Interactive Media of America—phone: (310) 444-6555 and e-mail address: **uunet!aimla!singh**.