We have examined what would be required to convert Habitat to use an all bitmap display, eliminating the mid-screen mode switching that may or may not be the cause of problems on certain early-rev Commodore 64s. These are the alterations we foresee being required, together with their impacts on the program.

To begin with, it is helpful to review the memory map of the present Habitat display area.

```
Address Use
------- ---
 0x4000 Bitmap #1 "start" / Sprites
 0x4100 Sprites
 0x4200 Sprites
 0x4300 Sprites
 0x4400 Text page start and Color nybbles
 0x4500 Word balloon area
 0x4600 Color nybbles
 0x4700 Color nybbles
 0x4800 Color nybbles / Sprite pointers
 0x4900 Start of displayed bitmap #1
 0x4a00 Habitat graphic image window
    .
    .
 0x5fff

 0x6000 Bitmap #2 "start" / Character set
 0x6100 Character set
 0x6200 Character set
 0x6300 Character set
 0x6400 Tables
 0x6500 Tables
 0x6600 Tables
 0x6700 Tables
 0x6800 Tables
 0x6900 Start of displayed bitmap #2
 0x6a00 Habitat graphic image window
    .
    .
 0x7fff
```

Although each display bitmap "begins" at the start of its respective bank of memory, the data that actually goes to the screen does not begin until later, since we have a mixed mode screen. The space that is not used for the display bitmap is devoted to other purposes. (When you experience a "meltdown", what you are in fact seeing is this other data rendered as a bitmap.) In a fully bitmapped version, the memory map turns into something like this:

```
 0x4000 Bitmap #1 start -- Word balloon area
 0x4100 Word balloon display area
    .
    .
 0x4a00 Habitat graphic image window
    .
    .
 0x5fff

 0x6000 Bitmap #2 "start" / Sprites
 0x6100 Sprites
```

```
 0x6200 Sprites
 0x6300 Sprites
 0x6400 Text page start and Color nybbles
 0x6500 Color nybbles
 0x6600 Color nybbles
 0x6700 Color nybbles
 0x6800 Color nybbles / Sprite pointers
 0x6900 Start of displayed bitmap #2
 0x6a00 Habitat graphic image window
    .
    .
 0x7fff
```

Since the word balloon display area does not need to be double-buffered as the graphic image display does, we can continue using the corresponding space in bitmap #2 for the color and sprite information (which need to be in one of the display banks anyhow). However, we need to find a new home for the character set definition and the tables that were previously stored there. This will require about 2K of space. The only place that this can come from is the heap.

Additionally, we will require text display routines to copy characters from the character set definition tables into the display area. Our estimate is that the various assorted routines needed for this purpose will require approximately .5K at a minimum. This too must come off the heap.

Since the bitmap display mode that we are using for the graphic area of the screen is not well suited to the display of text, a choice will have to be made about the resolution and format of the text area. There are two alternatives. The first is to redefine the character set to use a 4x8 matrix (as opposed to the 8x8 matrix presently used by the Commodore 64's hardware text mode). This will result in a significant degradation in readability but will otherwise preserve the text display as it currently exists. The alternative approach is to go to a higher resolution. This costs us the use of color in the text window to distinguish the various players' remarks from one another. A different means of indicating the speaker will have to be designed and implemented. An additional possible complication is that this approach will still require mode switching interrupts on the screen, though they will be switching between different bitmap modes rather than between bitmap mode and text mode. It is presently unknown if such resolution-mode switching will be subject to the same problems as we are now experiencing.

The performance impact of rendering bitmapped characters will be significant. The computation associated with the display of a single character will increase from (typically) a single store-accumulator instruction to a subroutine call. The subroutine will then perform the character set lookup and the multiple loads and stores required to copy the bitmap that defines the character. The resulting increase in computational overhead associated with single character display will thus increase by a factor of 100 or so. Character display operations that result in scrolling will consume another factor of 10 or so, due to the increased volume of data that must be moved and to the increased complexity of the move operation required. The overall impact will be a significant reduction in display frame rate whenever alterations to the text display portion of the screen are required.

The text entry line at the bottom of the screen will also be subject to these delays, with the added complication that this line scrolls horizontally. The present implementation manages this scrolling operation by the simple expedient of refreshing the entire line every jiffy that there is a change in the input buffer. This clearly will not work in the all bitmap case.

Special-case code will have to be created to make the choice of whether to scroll the line and if so in what direction. Furthermore, the scrolling itself will have to be custom implemented. The net result will be at least a factor of 100 increase in the delay associated with typing a character.

A further complication arises from the text display required by the book/paper interface. In this interface we presently have an entirely text mode screen. For the all bitmap system, there two alternatives available to us.

The first alternative is to render the book/paper display in bitmap as we render everything else. This will require a separate 640 byte buffer to hold the text itself, in addition to being extremely slow. The 640 bytes will, of course, have to come off the heap.

The second alternative is to continue to use an all text mode display for the book/paper interface. This entails either that we set aside a separate area of memory for the text display buffer (1K or so) or that we dynamically copy the character set definition into the bitmap area each time we switch to text mode, so that the display buffer and the character set will be in the same bank of memory (the Commodore 64 requires this). Additionally, we lose the present ability to display word balloons while in the book/paper interface, the word balloons now being set up to be rendered in bitmap mode (the alternative of including a parallel set of text mode word balloon routines seems somewhat extravagant).

In summary, the impacts of going to an all bitmap display for Habitat are as follows:

1) Loss of a minumum of 2.5K bytes and likelier more from the available heap space
2) Factor of 100 to 1000 degradation in text display performance (likely driving the frame rate to 1 or worse in some cases)
3) An undetermined but significant product delay (my initial guess is somewhere between 2 weeks and 2 months) while we install and debug the various changes required.