Introduction

Lib3d is a library of code for the creation of 3d games for the SAM Coupé computer. It draws multi-colour Mode 4 wireframe objects with hidden face removal.

Lib3d is designed to take control of the machine; you supply routines for the important, game-specific tasks and Lib3d calls them as and when it needs to. In return for surrendering complete control, it supplies routines that allow you to plot and manipulate 3d objects.

Limitations

The following limitations stem from design decisions and will not be addressed by future versions of the library:

- the maximum size of any individual object is just less than 4 units × 4 units × 4 units.

The following are limitations in the current version of the code that may be rectified at a later date:

- the only means currently supported for orienting objects is Euler angles (see later for a fuller explanation of what this means).

Overview and Memory Map

Lib3d itself is an assembly source file. That file uses references that instruct the assembler to include the contents of other files at certain points. In order to write a program that uses Lib3d, you need to provide named routines in these files. Lib3d will call your routines as required.

Lib3d uses quite a lot of the SAM's memory for its own purposes. If you are targeting a 256 kb machine then your program can occupy up to approximately 50 kb of memory. In that circumstance, you do not need to worry about memory paging — Lib3d will handle that for you.

If the machine has more than 256 kb then how you use the rest is up to you.

As a result of the two-way communication process that occurs between a game and the library, the rest of this documentation is split into two main sections: the functions you need to provide for Lib3d and the functions that Lib3d provides for you. Appendices describe the various 3d data formats.

The rest of this section documents the Lib3d memory map.

The Lib3d kernel and global resource store

The Lib3d kernel sits in the low 32 kb of RAM. All 3d code and Lib3d's built-in paging mechanisms are contained in that 32 kb. It also contains the processor stack and the **global resource store**.

The global resource store is the area that you should use for storing all of your 3d models, as it is the only area that the 3d drawing code has access to. It is also where you should put any variables or routines that all of the rest of your code is expected to be able to access, in order to ensure compatibility with future versions of Lib3d.

Although the size of the global resource store will vary as code is added to or subtracted from Lib3d, it will hopefully remain relatively constant. Please see the table at the end of this documentation for its size in the current version of Lib3d.

Collectively the 32 kb containing the kernel and the global resource store is known as the kernel page.

The user page

At present, all user code that is not in the global resource store is stored in a separate 32 kb area known as the user page. In practice only a tiny bit less than 30 kb of this is free.

Whenever user code is entered, it will be paged into the upper 32 kb of memory, and the kernel page will be occupying the lower 32 kb of memory. Lib3d provides a couple of functions for direct pixel access to the screen, but it is envisioned that the majority of drawing will be achieved through the 3d routines contained in the Lib3d kernel.

Screens

Lib3d uses triple buffering. That means that it maintains a circular queue of three separate screen buffers. At any time, your game code will have one of them reserved for drawing to and the TV screen will be displaying one of the others.

A queue of three rather than two is kept because it is quite unlikely that your code will finish drawing a new screen in less than the time it takes a TV to display an old screen, and very unlikely that it will finish in an exact multiple of the time it takes the TV to display an old screen. So, if there were only two buffers then when the CPU finished drawing one screen it would need to wait until the TV had finished with the other before it could start drawing again. And that is time you don't want to throw away.

Because of the way the SAM lays out its memory, this means that Lib3d uses 96 kb of memory for the triple buffer. 24 kb of that isn't actually used for pixel graphics, but Lib3d uses that for other screen-tracking purposes.

Multiplication tables

Lib3d uses a 4 kb multiplication table. It stores 2 kb of this in the first 2 kb of RAM. The other 2 kb is stored in the final 2 kb of RAM. In order to ensure continuous operation, the 2 kb that goes at the top of RAM is duplicated across all pages that Lib3d expects to occupy the top 32 kb of the memory map. The first 2 kb is stored at the beginning of the kernel page.

Other pages

Lib3d also maintains a 64 kb reciprocal table for perspective projection. 32 kb is currently reserved for multiplication code that is intended to form part of a future optimisation of the library.

Functions You Must Supply

Unless specified otherwise, the values in all of the z80 registers are insignificant upon entry of any of your routines, and you may leave any values you like in the registers when you return control.

You must supply separate functions for updating your game state and for drawing a new frame.

Init

The **Init** function goes in init.z80s. It is called exactly once, when the game is first loaded. You can use it to initialise the state of your game, or to do more advanced tasks such as loading a high-score table from disk.

Update

The **Update** function should update the state of your game and goes in Update.z80s. So, supposing you are writing a Battlezone clone, it should parse the player input, update the player position, run the artificial intelligence for each of the enemy tanks and move them accordingly.

No drawing should be done in the Update function.

Lib3d provides you with enough information to ensure that your game runs at a constant speed from the player's point of view irrespective of the frame rate. So when there are too many objects on screen, your game should appear to get more jerky but not slower. Similarly, your game should run at the same speed whether on a normal SAM or one with an enhanced 12 or 20 Mhz processor — it'll just be a great deal smoother on those machines.

Lib3d assumes that you ideally want to do 50 updates a second. However, because of the costs of performing 3d calculations on the SAM, you will rarely achieve this. So, upon entry into your Update routine, the register pair DE will contain the number of updates that you should do now in order to make sure that 50 have been done in the previous second.

If your game is running at an even 50 frames per second, DE will always have the value 1.

If your game is running at an even 25 frames per second, DE will always have the value 2.

If your game is running at an even 33 and a third frames per second, DE will alternate between containing 1 and 2.

Normally your game will run at a variable frame rate, causing the values in DE to be similar from one update to the next but not to necessarily follow a well-defined pattern.

Draw

The **Draw** function goes in draw.z80s and should be used to draw a new frame.

The contents of previous frames that were drawn by Lib3d will have been automatically erased before this function is called. Other pixels near those touched by Lib3d may also have been cleared.

IRQ hook code

The contents of the file IRQ.z80s are included directly into Lib3d's end-of-display interrupt handling code. They should be short, and fast. This code will have access to the contents of the global resource store, but will not reliably have access to any other memory area.

This hook is primarily provided to allow the inclusion of background music.

Functions Supplied

SetCamera_Rot3

Takes a pointer to an Euler information block in IY, and sets up the display so that the camera is in the position described.

This function will overwrite the contents of all registers.

DrawModel_Rot3

Takes a pointer to an Euler format model information block in IY and draws it to the display.

This function will overwrite the contents of all registers.

BeginScreenAccess

User code should call this to gain pixel access to the screen, which will be paged so that the top left pixel is at address 0 in memory. While the user code has pixel access, it shouldn't call any other Lib3d functions or try to access data in the global resource store.

This function will overwrite the contents of BC and A.

EndScreenAccess

User code should call this function to end their pixel access.

This function will overwrite the contents of BC and A.

FIXMUL

Multiplies together the 8.8 fixed point number in BC and the 8.8 fixed point number in DE and stores the result to HL.

The original contents of BC, DE, HL and A are discarded by this operation.

FIXMULADD

Multiplies together the 8.8 fixed point number in BC and the 8.8 fixed point number in DE and adds the result to HL.

The original contents of BC, DE and A are discarded by this operation.

FIXMUL1010 / FIXMUL1010ADD

FIXMUL1010 and FIXMUL1010ADD are identical to FIXMUL and FIXMULADD except that the top 6 bits of each number must be just the sign extension of the low 10 bits.

These routines use the multiplication tables and are substantially faster than FIXMUL and FIXMULADD.

FIXMUL1610 / FIXMUL1610ADD

FIXMUL1610 and FIXMUL1610ADD are intermediate precision multipliers between FIXMUL and FIXMUL1010. The top 6 bits of DE are assumed to be a sign extension of the low 10 bits.

These routines take approximately ten sixteenths as long as FIXMUL to execute. They are substantially slower than FIXMUL1010.

FIXDIV

Divides the 8.8 fixed-point number in BC by the 8.8 fixed-point number in DE and stores the result to HL.

The original contents of BC, DE, HL and A are discarded by this operation.

LONGMUL1010

Multiplies together the 8.8 fixed point number in BC and the 8.8 fixed point number in DE and stores the result to A:HL. This routine is like FIXMUL, but does not truncate the result.

The original contents of BC, DE, HL and A are discarded by this operation.

DIV2010

Divides the 12.8 fixed point number in A:HL by the 8.8 fixed point number in BC and stores the result to HL.

The original contents of BC, DE, HL and A are discarded by this operation.

Lib3d State Values

CAMERA_XVEC_X

CAMERA_XVEC_Y

CAMERA_XVEC_Z

CAMERA_YVEC_X

CAMERA_YVEC_Y

CAMERA_YVEC_Z

CAMERA_ZVEC_X

CAMERA_ZVEC_Y

CAMERA_ZVEC_Z

Appendix A: Basic Data Types

Appendix B: 3d Model Format

Appendix C: Simple Example Program

Appendix D: Simple Example Game