SONY COMPUTER ENTERTAINMENT EUROPE

GUIDELINES FOR PLAYSTATION PAL TITLES

DESIGNING PLAYSTATION GAMES FOR NTSC AND PAL TERRITORIES

Version 2.1

19 June, 1996

DOCUMENT HISTORY

Version 1.0	mid-1995 (Allan J. Murphy)	Initially called "**Guidelines for PlayStation PAL Conversions".**
Version 2.0	June-1996 (Various)	Renamed and updated to reflect the new "Global Technical Requirements"
Version 2.1	1-July-1996 (Paul Holman)	Updated following internal review.

INDEX

GUIDELINES FOR PLAYSTATION PAL TITLES

DESIGNING PLAYSTATION GAMES FOR NTSC AND PAL TERRITORIES

INTRODUCTION

This document attempts to describe some of the issues you will need to face if you are designing a PlayStation product to be published in the European territory in addition to those of Japan and America.

The most significant difference between the PlayStation consumer machines in these three territories is that they have to deal with different display types - the US and Japan use the NTSC standard for TV displays, and in Europe the most common display type is PAL.

As such, there are some implications which you may not have considered, especially in the way that the PlayStation deals with various facets of game production. Some of these may be obvious, some may not.

DIFFERENCES BETWEEN PAL AND NTSC

The main differences between PAL and NTSC displays are:

Frame Rate

NTSC displays run at **60 Hertz**, i.e. they refresh the screen 60 times per second. PAL screens run at **50 Hertz**, i.e. they refresh the screen 50 times per second.

A standard game programming technique is to use an interrupt called the vertical blank interrupt to drive game logic. This interrupt occurs when the electron gun inside the TV starts its return from the bottom of the picture to the top of the picture to start refreshing the display again. Obviously, if your game uses this interrupt to trigger game logic, you have to consider the implications of this interrupt happening 50 times a second, not 60 times a second.

Number of Raster lines on Screen

An NTSC display is **525** raster lines high, whereas the PAL screen is **625** raster lines high.

The direct implication for PlayStation game products is that your NTSC title probably does not draw a tall enough display to fill a PAL TV screen. For example, a common resolution to run an NTSC PlayStation game is 320 pixels wide by 240 pixels high. On an NTSC TV, 240 pixels high is enough to fill the screen. On a PAL TV, 240 high is not tall enough to fill the screen; 256 pixels high is about right. This means that you may have to make some changes to the way your game draws its display, and find some mechanism for filling the extra space. There are various ways to do this.

OVERVIEW OF TECHNICAL ISSUES

There are five areas of concern in designing for converting a game to PAL. These issues are due both to the difference in the display technologies, and also to the PlayStation the libraries used to drive it.

Libraries and Hardware

First, you need to know how to develop a PAL version of your product - choosing the hardware from SCEE, using the PlayStation libraries for PAL operation.

Rendered Sequences

What do you need to do to create full screen rendered sequences for PAL mode ?

Music and Sound Effects

What are the implications of different frame rates for your game's music and sound effects ?

Graphics and VRAM Allocation

What are the different ways of making your display full screen for both PAL & NTSC ? How do you re-arrange your game code and data to function effectively with different screen heights ?

Game Mechanics and Motion Capture

How best to design your game's mechanics to make the game work well in both PAL and NTSC and mode ? What are the implications of the display change for a typical game ?

Has your Motion Capture data been processed to reflect both PAL and NTSC frame rates ?

PAL HARDWARE SUPPORT

Development Systems

PlayStation development systems (such as the DTL-H2000 and the DTL-H2500) are PAL switchable.

- The DTL-H2000's main ISA board has a jumper clearly labelled NTSC/PAL. To switch the hardware to PAL, set this jumper to PAL.
- The DTL-H2550 is software switchable from the DESICONS monitor.

When placed in a PC, and connected to a suitable TV monitor in this mode, you will see a black and white "test card" output on your monitor when using the standard AV cables.

Once you have set the development system to PAL mode your development system will behave exactly like a PAL consumer PlayStation both in terms of display output (e.g. 50 Hertz display, 625 lines) and also in terms of the machine internals (vertical blank interrupt 50 times a second, and so on).

You will need to make a software switch inside your code to ensure that your programs produce colour PAL mode correctly.

Debug Stations

Different Debug Stations are available for each territory, and are used for checking code running on as near to a final machine as possible (i.e. only having 2Mb of main memory. And with an OSD similar to that of the consumer machines). To fully test PAL titles, you should obtain a DTL-H1002[1] PAL Debug Station.

PAL SOFTWARE LIBRARY SUPPORT

There areas in which PAL specific software issues need to be considered:

- the graphics library (to make the software switch from NTSC to PAL)
- the sound libraries (these are tied to the vertical blank rate, and so a small change must be made to make sequenced sound work in PAL mode).
- rendered sequences (which should be created and played at the correct frame rate)
- when laying out VRAM (display) memory

[1] Will be renamed as DTL-H1002 during the latter part of '96 (to meet "CE" electrical guidelines).

RENDERED SEQUENCES

This section describes the changes you will need to make to any rendered sequences in your games. This section assumes your rendered sequence is using the PlayStation MDEC hardware to decompress the sequence data.

The issues are:

- Changes in Frame Rate
- Changes in screen size (for full screen)

Frame Rate (fps)					
NTSC	30	15	10	7.5	
PAL	25	16.66	12.5	10	6.25

Sectors per Frame					
NTSC	5	10	15	20	
PAL	6	9	12	15	24

FRAME RATE CHANGE

Reference the table (above) for the range of frame rates (in frames per second) available for your MDEC renders. These frame rates are directly determined by the number of CD sectors you can read between the vertical blanks (see the second table).

For example, in NTSC mode, at 30 fps, you can read 5 CD sectors during the period between 2 vertical blanks; at 15 fps, you can read 10 CD sectors because you have the time in between 4 vertical blanks.

Although NTSC and PAL machines have CD drives that run at the same speed, (i.e. the CD data rate is constant), in PAL mode, you have more time between vertical blanks, so you can read correspondingly more data. Thus with PAL, Because you have more time, at 25 fps you can read 6 CD sectors, and at 12.5 fps, you can read 12 sectors.

The advantage is that the quality of your render can be better under PAL than NTSC. The disadvantages are that you may have to convert or re-render an existing NTSC rendered sequence to ensure that the render doesn't appear too slow in PAL mode if you use the same set of frames as in NTSC mode. This if you have stored 1 frame per MDEC frame for an NTSC sequence, you may have to re-render the whole sequence at 25 / 12.5 / 6.75 frames per second.

One solution is store your render on an analogue media (tape or whatever) to allow re-sampling at the right frame rate (for example, if you have digitised some video rather than a rendered sequence). If you have a considerable amount of render or streaming (e.g. in a streaming based game or branching video game) you might like to see exactly how bad it looks in PAL mode before you consider spending the time/money on re-rendering or re-digitising.

INTERLEAVING XA AUDIO

The other consideration from the above problems with frame rate changes is how do you interleave XA audio ? The interleaving ratio is different, because the number of sectors you are using for each frame’s image is probably different. The simple answer is the MovConv tool, which will interleave XA audio sectors in a PAL stream properly. If you don’t use MovConv, then you will have to determine whether you need to make changes to whatever tools you are using to allow the interleaving to work properly for PAL.

MUSIC AND SOUND EFFECTS

This section describes the minimal changes which need to be made to sound data and code for the change from NTSC to PAL.

XA AUDIO

The problems of XA audio are described above for the case of interleaving XA with streamed video data. If you are using raw XA only as a sound source, you have no problems, because the CD rate is fixed and does not change between the two display modes.

DA AUDIO

The situation for using DA audio (Red Book CD audio) is very simple - you don't need to change anything, as the CD rate is fixed.

SPU AUDIO & SEQ SEQUENCES

The situation regarding SPU audio is a little more complex. VAG (SPU ADPCM data, or raw samples) will play back at the same speed regardless of the display type, as this type of playback is dependent on the SPU internal hardware clock.

However, if you are playing back a sequence (SEQ, which is based on standard MIDI and created using the SMF2SEQ tool) designed for NTSC the individual samples will play at the right rate, but they will be triggered wrongly by LIBSPU and LIBSND. This is because the libraries are tied to the vertical blank rate of the machine in software.

Fortunately, there is a simple workaround for this, which should only cause problems in extremely rare cases:- alter the tempo of the sequence to compensate for the change in frame rate.

This can be done manually (by altering the speed of your original sequence), or alternatively using a switch in the software libraries so that sequences are played back at the right rate automatically. So for example, a sequence triggering a particular sample twice a second (every 30 vertical blanks in NTSC mode, every 25 vertical blanks in PAL mode) which was converted for NTSC, should have either its tempo speeded up by a factor of 30/25, or should run one fifth faster.

In libraries version 3.1 and onwards, it is possible to set the whole system to PAL mode by calling:

```
long SetVideoMode(long mode);  /* Part of   libetc.lib in  libs 3.1  */
/* Modes are: MODE_PAL, MODE_NTSC. */
```

This sets up **libgpu** and **libpsu**/**libsn** to PAL or NTSC mode, as long as you call it before you initialise any of the other libraries. This saves you both setting the *pad0* field (as above) and manually altering the tempo for the sound libraries. The support for making the change in tempo is included in libraries version 3.1 and onwards. The tick mode of the sound libraries can be changed as follows:

```
SsSetTickMode(SS_TICKVSYNC);   /* Ticks according to mode of    libs, set as above */
SsSetTickMode(SS_TICK60);      /* Ticks at NTSC speed anyway,    eg 60Hz */
```

```
                     /* Video mode as set above is relevant */
SsSetTickMode(SS_TICK50);      /* Ticks at PAL speed anyway,    eg 50Hz */
                     /* Video mode as set above is relevant */
```

GRAPHICS & VRAM ALLOCATION

This section describes the changes you will need to make to your product in order to produce an acceptable graphics display in PAL mode, along with details of how to build a taller display to fill the PAL screen.

ALTERNATE GRAPHICS MODES

First, the standard resolution used for NTSC titles is 240 pixels. This is not tall enough to fill the whole PAL display. There are three main approaches to this problem.

Letter-boxing

This approach is the easiest, and also the least attractive, acceptable. In this case, you take the NTSC display which your game already generates, and simply centre it on the PAL display; this results in no changes to your graphics display code (at least in terms of creating extra material to fill the gaps) but also means there is a significant black border at the top and bottom of your display (which doesn't look very attractive, and is fairly obvious).

Since 1st June 1996, SCEE has expected that no titles should be Letter-boxed..

Letter-boxing with border filling

The next alternative is to letterbox the NTSC display (as above) and fill in the borders with additional graphics.

There is a significant problem with this approach - the concept of filling borders is actually patented by Atari Corporation; this technique was used to fill the whole display in an early Atari console. As such SCE cannot endorse this approach, nor can it provide source code to do this as part of the software libraries for PlayStation development.

Using a full screen PAL display

The remaining technique is to use full screen PAL (e.g. at least 256 lines rather than 240 lines) high drawing and display area. A 256 high display will fill the PAL screen properly without significant borders.

An existing NTSC title (especially if its 3D) may require a significant amount of reworking to use a 256 high display, and we strongly recommend that plans for PAL support are made during initial game design.

In addition, simply "stretching" your existing NTSC display may adversely effect the aspect ratio.

WRITING CODE THAT SUPPORTS BOTH 240 NTSC AND 256 HIGH PAL DISPLAYS

You must make the software switch for PAL mode displays. This is performed using *SetVideoMode*(). With this set, when you call *PutDispEnv*(), *libgpu* will set the display to the mode you have requested. See the code fragment below for an example.

```
#ifdef PAL_256          /* Do we want 256 (PAL) high ?    */
  #define  FRAME_X     320     /* Assume 320 pixels wide   */
  #define  FRAME_Y     256     /* Frame is 256 high.       */
  #define SCREEN_X     0       /* Offset display by 0 in X    */
  #define SCREEN_Y     24      /* Offset display by 24 in Y   */
#else            /* Otherwise assume 240 NSTC high    */
  #define  FRAME_X     320     /* Assume 320 wide display.    */
  #define  FRAME_Y     240     /* Make frame 240 pixels.   */
  #define SCREEN_X     0       /* Offset X by 0.      */
  #define SCREEN_Y     28      /* Offset Y by 28 pixels.     */
#endif
/* Note: display offsets may or may not centre the display on a      */
/* particular TV. SCEE will provide offsets which give an        */
/* average centring for various displays after some research.    */

typedef  struct {
   DRAWENV     draw;       /* drawing environment */
   DISPENV      disp;       /* display environment */
} DB;
DB db[2];  /* Standard drawing and display environment set up.      */

void main()
{
# ifdef PAL_256
  SetVideoMode(MODE_PAL);   /* This is for   libs 3.1 and onwards. */
# endif

  /* Set up drawing and display environments as per usual.       */
  SetDefDrawEnv(& db[0].draw, 0,   0,        FRAME_X, FRAME_Y);
  SetDefDispEnv(& db[0].disp, 0,   FRAME_Y, FRAME_X, FRAME_Y);
  SetDefDrawEnv(& db[1].draw, 0,   FRAME_Y, FRAME_X, FRAME_Y);
  SetDefDispEnv(& db[1].disp, 0,   0,        FRAME_X, FRAME_Y);

  /* Apply offsets to display to centre the display       */

  db[1].disp.screen.x =  db[0].disp.screen.x = SCREEN_X;
  db[1].disp.screen.y =  db[0].disp.screen.y = SCREEN_Y;

/* Set up the width and height of the display on screen.     0 is default */
  db[1].disp.screen.h =  db[0].disp.screen.h = FRAME_Y;
  db[1].disp.screen.w =  db[0].disp.screen.w = 0;

# ifdef libs30
/* We recommend that you use libraries 3.1 or above, which provided support
   for SetVideoMode. If you are using libraries 3.0 or below, a different
   technique is required: Set the pad0 field to tell GPU to go to PAL mode.
*/
    db[1].disp.pad0 =  db[0].disp.pad0 = 1;
# endif
}
```

VRAM IMPACT & SUGGESTED OPTIMISATIONS

The above ideas have varying effects on the use of VRAM. The sections below describe the VRAM impact.

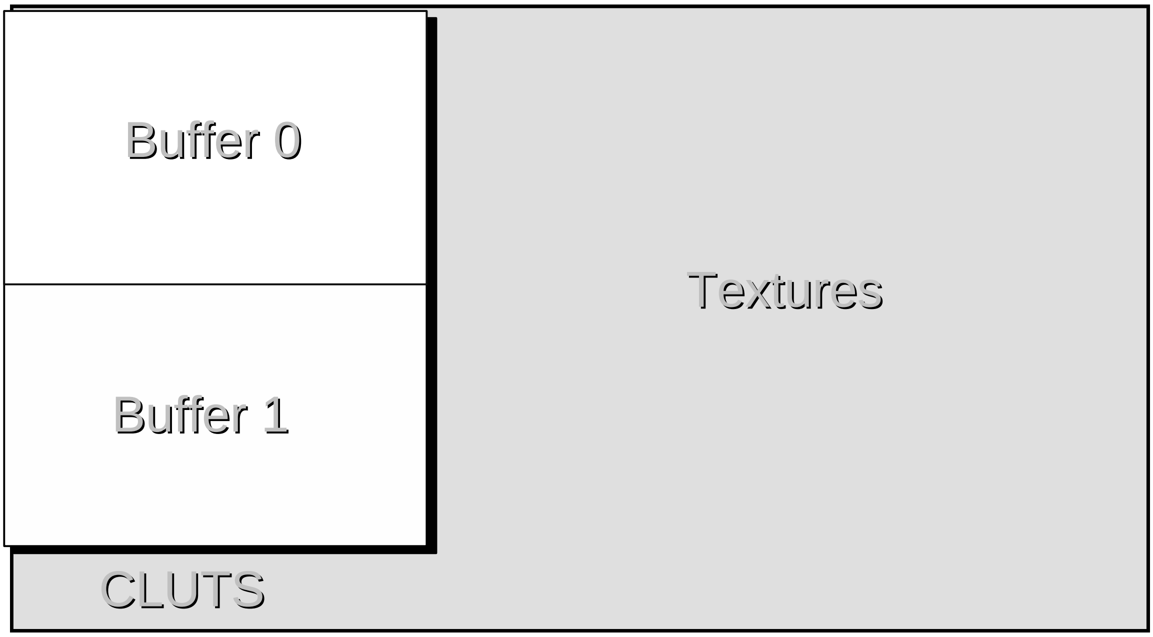

Figure 1 Typical layout of VRAM for NTSC which will preclude full screen PAL

Letter-boxing

This approach has no impact on VRAM arrangement; your NTSC title remains exactly the same in terms of display and VRAM, so no alterations are needed.

Using a 256 high display

This is the hardest of all; in the typical example of a 320x240 NTSC title, the display and drawing areas in VRAM are kept on the left hand side of the VRAM area (Figure 1), with a 320x32 area under the two areas kept for CLUT usage (or perhaps some sprite or texture map data). So in order to fit a 320x256 display and drawing area, either a 320x32 area must be found elsewhere in VRAM, or the contents of VRAM re-arranged to allow for this extra space elsewhere (Figure 2).

Obviously, this will most likely have a knock-on effect in many products which swap textures in and out of VRAM or do tricks with moving textures or the display / drawing areas around, and may require a revision of the title's VRAM handling mechanisms. The easiest case is a product where VRAM is not really that full, and CLUTs or small textures can be relocated from the area below the drawing and display areas.

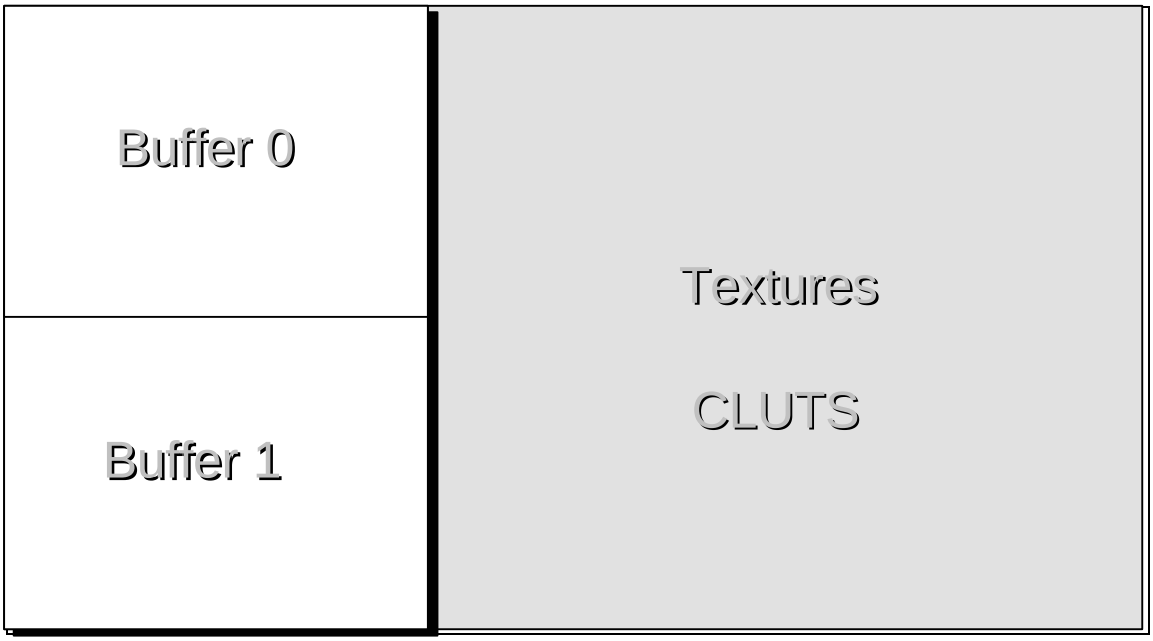

Figure 2 Layout of VRAM for full screen PAL

Obviously, your title may organise VRAM completely differently than that described above (there are some PlayStation titles with triple buffers!), but the fact remains that a 256 high display will require you to find extra VRAM space from somewhere. If you are lucky then the re-organisation will be simple; but the worst case may be very serious and might require a lot of re-coding and graphic re-arrangement. Some possibilities are dropping textures from objects in the game (for the case of a 3D title) or using a smaller texture map stretched over a larger area (for 2D or 3D, but impacts performance and graphic quality).

OVERSCAN AND SAFE AREAS

You should consider the overscan area on TVs, and its impact on graphic displays (for both NTSC and PAL).

Overscan is where the electron gun from the TV draws more of a raster line than can actually be seen on the TV screen; this is perceived as some of the display going missing from the right, left, top or bottom of the display. For game design this is important as player status information which is inside a possible overscan area may not be displayed on some TVs.

In general, placing important game information, lettering or even important game play elements hard against any edge of the display is a bad idea. Unfortunately, overscan is different for each different type of TV, so general guidelines have to be given rather than hard figures. You may find that on a particular TV you can see all of the display, others may show only one edge of the display.

For NTSC, vertical resolutions such as 240 and 480 have full overscan.

For PAL, vertical resolutions of 256 and 512 have no overscan (288 would be required).

You should ensure that no important information (scores, text) is positioned near display edges (around 8 or 16 pixels away from the edges depending on resolution).

GAME MECHANICS

In relation to designing titles for both NTSC and PAL, this issue is the most difficult to provide guidelines on as the game mechanics are entirely generated by you, the developer.

It is almost certain that the game mechanics and event generation in your game is tied to the vertical blank interrupt, however. For example, in a fighting game, the fact the user hit the 'punch' button is recorded at vertical blank 'n'; the game then knows that for the next 30 vertical blanks it is drawing the character punching on screen, and at vertical blank 'n+30' it must check to see if the punch hit the opponent.

If this is the way the game works in NTSC, then all the timings must be altered for PAL; the animation of the player punching must run for only 25 vertical blanks, and the check to see if the opponent hit must occur at 'n+25'. However, the 30 frames of animation for the NTSC punch cannot easily be converted to 25 frames of animation for a PAL punch; but they will look wrong in PAL mode. Similar types of timing problems can be found for any game genre, more or less.

Converting game mechanics to run at the correct rate for PAL is definitely a non-trivial task, but essential to prevent your game appearing to run at a rate one sixth slower than its NTSC equivalent..

If the mechanics are not designed or converted correctly, it will appear as if there is something wrong with your PAL game conversion, which is not good either for your image or your sales.

MOTION CAPTURE DATA

Many titles use motion capturing techniques to correctly convey the movement of "live" objects (to capture the subtleties in the movement of an athlete or animal).

The process of transforming the captured information, into data to be used within a game will normally make the data dependent on the frame rate.

You may need to ensure that two sets of data (for the PAL and NTSC variants of your game) are collected during this initial capture/transformation process, or to sample at rates such as 300 Hz (which can be subdivided down to either PAL or NTSC rates).

SUMMARY

Plan your development, considering both PAL and NTSC formats during the initial design

If possible - localise / customise for each display type, to take advantage of the benefits each form provides; e.g. PAL can calculate / draw more in one frame. NTSC has a faster refresh rate.

Technically, you should consider a number of key issues:.

- SPU sequenced sound should run at the correct speed.

  There is no reason why this should not be the case.

- Rendered sequences should run at the correct speed for PAL.

  This may be quite difficult to achieve, unless the product has been designed for both NTSC and PLA systems, but the product will look inferior without this effort;.

- The display should be full screen PAL (at least 256/512 pixels)

  You are strongly encouraged to make this change, because the title will look inferior to the NTSC version otherwise, which results in problems with selling the game and also the PlayStation hardware.

- Text and essential display items should not be near the edges of the display.

  This requirement just needs some testing on various TVs, and judicious placement of status information and important game-play items in the display.

- Some attempt should be made at converting the game mechanics.

  This may be the hardest change of all. However you must remember that a game that runs 1/6th slower for the European market may not sell as well as its correctly designed competitors.