Matrox Simple Interface

Usage Guide

Version 1.1

Last Updated :April 3, 1997

INDEX

Introduction

MSI (Matrox Simple Interface) is a simplified API for accessing the 3D features of some of our graphics cards. This API allows the game developer to use our hardware accelerated features such as Gouraud Shaded and Textured triangles, hardware Z-buffer, special case Zplanes, etc. At the same time, the hardware will be taking advantage of BUS-MASTERING which provides higher frame rates. This guide deals with optimizing your usage of the MSI API, and related issues, so that you may derive the maximum benefits from BUS-MASTERING.

Bus-Mastering

Bus mastering allows the desynchronizing of the graphics card and the computer. This lets the program using the card continue executing, while the card is rendering graphics. This means that even if the graphics card takes 1 second to draw something, the program can still continue to do its own work during that time. However, to be able to use this feature to its peak, several constraints must be respected.

Bus-Mastering constraints

The constraints for BUS-MASTERING are:

- Minimize or totally eliminate the number of ***Direct Frame Buffer*** accesses. Place any ***Direct Frame Buffer*** access at the start of a frame, before any rendering occurs.
- Use **msiEndFrame** with the WAIT parameter set to **FALSE.**
- Implement Texture loading by using BUS-MASTERING (Heaps).
- Minimize Heap allocation.

Other MSI constraints

Please keep these other MSI constraints in mind:

- **CacheOffset**'s 5 LSB's must always be 0.
- **msiSetParameters** calls that set values to other than NULL or -1 must always be done inside a **msiStartFrame/msiEndFrame** pair.
- When entering Direct Access Mode (by calling **msiSetParameters** with NULL), you must always exit (by calling **msiSetParameters** with 1 or valid pointer), before doing a **msiStartFrame** or **msiEndFrame**.

Best Usage

To maximize BUS-MASTERING, the proper use of MSI is:

GAME STARTS
GAME PRELOADS TEXTURES TO TEXTURE CACHE

GAME PRE-CALCULATES THE FRAME
msiStartFrame
    DIRECT FRAME BUFFER ACCESS (TRY TO ELIMINATE)
    RENDERING OF PRE-CALCULATED FRAME
        - **msiRenderTriangle**
        - **msiSetParameters**
        - **msiRenderZPlane**
msiEndFrame (WAIT=FALSE)

When the API is programmed in this way, you maximize the time that the engine is actually doing BUS-MASTERING. Entering Direct Frame buffer access or calling **msiEndFrame** with WAIT=TRUE, stops the BUS-MASTERING and takes away the great performance gains. Doing Texture loads from system memory instead of from a Heap, will also have a negative effect on BUS-MASTERING.

Calling the **msiEndFrame** with WAIT=FALSE means that the flip of the view/draw buffers is done at the next **msiStartFrame.**

Bus-Mastered Texture loads

Again, it is important to use BUS-MASTERED Texture loads (Heaps) if you want to maximize the effects of BUS-MASTERING. However, there is a very important point to understand about Heap management:

Whenever you use **msiAllocTextureHeap**, we allocate 1 page more than what you ask for (needed for internal use). This means that if you allocate 100 heaps of 4 pages each, you are actually using 100 heaps of 5 pages each. It would be better to allocate large heaps, in which you internally manage your textures. Remember that this is fixed memory and therefore the amount that is allocated decreases the amount of memory available to the system.