# Matrox Simple Interface for DOS

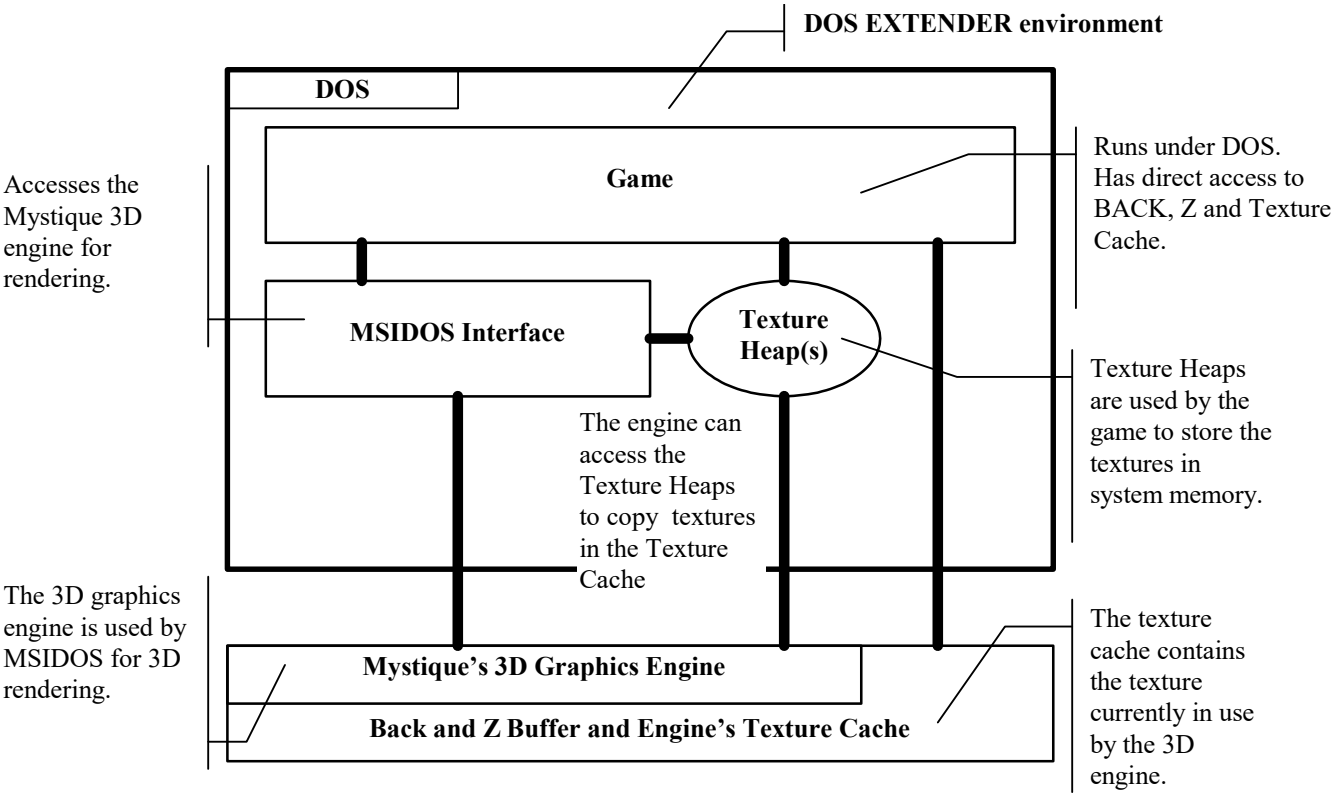
# Specification


## Version 1.5

### Last Updated: July 11, 1997

# INDEX

# The overall architecture




The game has access to a set of functions to initialize, clean-up, mark the begin and end of a frame, render textured triangles, draw lines, create and manage textures.

The Matrox Simple Interface for DOS accesses the Mystique 3D engine directly. The game is provided with direct frame buffer accesses to the BACK, Z and Texture Cache memory to perform dumb frame buffer rendering and texture management. A powerful texture management architecture is provided, giving to the game all the flexibility to manage directly the engine's texture cache, to create texture heaps from which the engine updates the texture cache and even to have texture stored anywhere in system memory.

# The operating environment

The game must be compiled with Watcom C/C++ 10.0 or more together with the Rational DOS Extender that is provided with that compiler. The library provided is compiled with Watcom C/C++ 10.6 (Register Passing option). The msidos.h file that specifies the API calls must be also be included in the compilation. A CASEEXACT option is also required when linking the final .exe.

Supported and tested graphic modes:

| Screen Resolution | Other Capabilities | Used Video Memory |
|---|---|---|
| 512x384x16bits | Z / No Z | ~1.2 M / ~0.8 M |
| 640x400x16bits | Z / No Z | ~1.5 M / ~1.0 M |
| 640x480x16bits | Z / No Z | ~ 1.8 M / ~1.2 M |

# MSIDOS API function list

## *Initialize and Exit functions*

```
T_msiInfo* msiInit  (    LONG Width,
                         LONG Height,
                         LONG Planes,
                         BOOL ZBuffer,
                         BOOL Debug   )
```

This entry point will initialize the Matrox Simple Interface library and the Mystique 3D engine to the specified full screen resolution and pixel depth. The allocation of a Z buffer is optional for games that pre-sort the triangles. A BACK buffer is always allocated as the rendering always occurs in the back buffer. Optional debug information may be written to the file msidebug.log in the current directory.

Returns a pointer to a T_msiInfo structure that gives some useful information for the dumb frame buffer renderer (pointers to the back and z buffers and texture cache). The pointers of T_msiInfo are valid for direct accesses only between the calls to msiSetParameters(NULL) and msiSetParameters(-1 or a valid pointer). Initially the pointers are set to NULL. The MSIDOS library will update the T_msiInfo structure every time the msiSetParameters(NULL) function is called and reset the pointers to NULL every time the msiSetParameters(-1 or a valid pointer) function is called (**do not cache these pointers)**.

Returns a NULL pointer if an error occurred. Possible causes are: Specified screen mode not supported by the API, not enough memory on display adapter. Check the msidebug.log file for more information about the problems.

**Returns** TRUE if successful.

```
BOOL msiExit()
```

This entry point is used to return to normal DOS text mode.

**Returns** TRUE if successful.

## *Start and End Frame functions*

```
BOOL msiStartFrame  (    BOOL ClearRGB,
                         float r,
                         float g,
                         float b,
                         BOOL ClearZ,
                         float z        )
```

This entry point is used to mark the beginning of a frame. Optional color and depth may be specified to clear the color and Z buffers. If ClearRGB is set to FALSE, no clear is performed on the color buffer. If ClearZ is set to FALSE, no clear is performed on the Z buffer.

**Note:** Dithering is NOT performed on RGB clear. This function will return an error if MSIDOS is in direct access mode; msiSetParameters (-1 or a valid pointer) must be called before calling this entry point in such a situation.

**Returns** TRUE if successful.

```
BOOL  msiEndFrame  (    BOOL Dump,
                        LONG Frame,
                        BOOL Wait       )
```

This entry point is used to mark the end of a frame. The BACK buffer is swapped/flipped to the FRONT buffer. Optionally the frame can be written to a file named frameXXX.bmp in the current pixel format (XXX is replaced with the value of the parameter **Frame**).

When this entry point is called, the graphics engine may not be ready to swap/flip the buffers because it is still processing the current frame. If **Wait** is set to TRUE, the game blocks and waits until the processing of the current frame is completed. If **Wait** is set to FALSE and the graphics engine is still processing the current frame, MSIDOS will postpone the completion of this command to the next call to msiStartFrame or msiSetParameters(NULL) or msiEndFrame(*, *, TRUE).

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters (-1 or a valid pointer) must be called before to calling this entry point in such a situation.

**Returns** TRUE if successful.

## *Memory related functions*

```
LPBYTE msiAllocTextureHeap    (    LONG Pages     )
```

Allocate a texture heap of **Pages** 4K pages. A heap can contain any number of textures and is managed by the game. A heap is specified to MSIDOS by its base address.

**Note:** The heap size is limited to 1008 pages. (almost 4 MB of memory).

**Returns** Valid texture heap if successful
**Returns** NULL if an error occurred.

```
BOOL msiFreeTextureHeap    (    LPBYTE TextureHeap  )
```

Free a previously allocated texture heap. The texture heap content is lost after this operation. The parameter pTextureHeap is the value returned by msiAllocTextureHeap.

**Returns** TRUE if successful.

```
BOOL msiIsMemoryBusy  (  T_msiMemoryStatus *pMemStatus  )
```

**Note: This is used with functions which have T_msiMemoryStatus as a parameter.**

**Returns** TRUE if memory area has not been processed.

## *Rendering functions*

```
BOOL msiRenderTriangle (       T_msiVertex* pV0,
                               T_msiVertex* pV1,
                               T_msiVertex* pV2,
                               BYTE Opacity        )
```

This entry point is used to render a triangle according to the actual rendering parameters. This function supports gouraud and textured 3D triangles with many texturing options like decal, modulate, transparency, etc ... This function receives 3 pointers to the triangle vertices. All the information required by the actual rendering parameters must be set to appropriate values.

Each vertex contains:

| | |
|---|---|
| **x, y** | Viewport coordinates. (0,0) is the top-left of the viewport |
| **z** | Depth coordinate. This value range from 0.0 to 1.0 |
| **invW** | Inverse homogeneous clip coordinate used for perspective correction. Must be > 0 and < 32768. This value should be set to 1.0 for linear interpolation of the texture coordinates. |
| **r, g, b** | Surface color. These values range from 0.0 to 255.0 |
| **mr, mg, mb** | Texture modulation factor. These values range from 0.0 to 255.0 |
| **u, v** | Texture coordinates. These values should be in the range 0.0 to 1.0 when the clamp_u, clamp_v texturing parameters are set to TRUE. In other cases these values are not restricted to this range and the texture coordinates wrap around based on the fractional part. The restriction on u,v is then -32768 < u/w, v/w < 32768. |

The triangle passed to this function must be clipped to the viewport as no 3D clipping will be performed by this function. An opacity factor for the triangle, ranging from 0% to 100%, may be specified on a triangle basis. This factor is used to perform screen door transparency while rendering the triangle. The value 0 means fully transparent, while the value 100 means fully opaque.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters(-1 or a valid pointer) must be called before to calling entry point in such a situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL msiRenderZPlane    (    T_msiZPlane*   pZPlane,
                             BYTE           Opacity    )
```

This entry point is used to render a plane of constant Z according to the actual rendering parameters. This function supports flat and textured rendering with many texturing options like decal, modulate, transparency, etc. ... This function receives the Z plane top-left coordinates along with the screen and texture extents. All the information required by the actual rendering parameters must be set to appropriate values.

A Z plane specification contains:

| | |
|---|---|
| **x, y** | Top-Left viewport coordinates; (0,0) is the top-left of the viewport |
| **xWidth, yHeight** | Viewport extents of the Z plane; x + xWidth and y + yHeight must give valid viewport coordinates. |
| **z** | Depth coordinate. This value range from 0.0 to 1.0 |
| **r, g, b** | Surface color. These values range from 0.0 to 255.0 |
| **mr, mg, mb** | Texture modulation factor. These values range from 0.0 to 255.0 |
| **u, v** | Top-Left texture coordinates. **As opposed to the msiRenderTriangle,** these values should be in the range [0 to texture width-1] and [0 to texture height-1]; the clamp_u, clamp_v texturing parameters are implicitly considered as being set to TRUE. |
| **uWidth, vHeight** | Texture extents of the Z plane; u + uWidth and v + vHeight must give valid texture coordinates. |

The Z plane passed to this function must be clipped to the viewport as no 3D clipping will be performed by this function. An opacity factor for the Z plane, ranging from 0% to 100%, may be specified. This factor is used to perform screen door transparency while rendering the Z plane. The value 0 means fully transparent, while the value 100 means fully opaque.

## This entry point will be highly optimized for this very specific case:

- The texture is in the current frame buffer format (T_msiInfo.msiColor)

  - xWidth **==** uWidth **and** yHeight **==** vHeight

- T_msiParameters.msiTexture.Enable = TRUE
- T_msiParameters.msiTexture.Modulate = FALSE
- T_msiParameters.msiTexture.Decal = FALSE
- T_msiParameters.msiTexture.Clamp_u = TRUE
- T_msiParameters.msiTexture.Clamp_v = TRUE
- T_msiParameters.msiDepth.Enable = FALSE
- T_msiParameters.msiColor.Dither = FALSE

In this very specific case msiRenderZPlane is equivalent to a simple bitblit with color keying ...

**Note:** The **x, y, u, v, xWidth, yHeight, uWidth and vHeight** values are INTEGER as opposed to the msiRenderTriangle entry point where only floating point values are used.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters(-1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL msiBlitRect (     LPBYTE              Pheap,
                       LPBYTE              Pmemory,
                       T_msiMemoryStatus       *pMemStatus,
                       WORD                SourcePitch,
                       WORD                SourcePlanes,
                       WORD                BlitWidth,
                       WORD                BlitHeight,
                       DWORD               DestinationOff,
                       DWORD               KeyingColor,
                       DWORD               KeyingMask      )
```

**:: IMPORTANT :: IMPORTANT :: IMPORTANT ::**

**The blit function only works in 16bits, only supports a destination of pitch 640, and only draws to the drawbuffer at the current time.**

This entry point is used to blit a rectangle to any plane using the parameters specified in the function. This function supports transparent blits and blits to the Z, Draw, Display and Offscreen buffers. This function receives a pointer to a heap and a pointer to memory inside the heap. It ***will blit from heap memory only***. You should not reuse this memory until you know that the blit was done. KeyingColor and KeyingMask determine if there is any transparency, if you want no transparency use KeyingColor=0xFFFF and KeyingMask=0x0000. The function returns TRUE if the command is accepted, FALSE otherwise.

The parameters are explained here:

**Pheap** Pointer to a valid texture heap.

**Pmemory** Pointer to a valid area inside the texture heap pointed by Pheap.

**PMemStatus** Pointer to a T_msiMemoryStatus structure that will be filled by msiBlitRect. This is then used internally by msiIsMemoryBusy to check if the blit has been executed.

**SourcePitch** Pitch of the source rectangle in bytes. **Must be DWORD aligned**.

**SourcePlanes** Number of planes of source bitmap 4/8/15/16/32

**BlitWidth** Width of source rectangle

**BlitHeight** Height of source rectangle

**DesitnationOff** Offset in **bytes** in the destination window. The **2 MSB**'s decide the destination window in this manner:

**00 Draw Window**
**01 Display Window**
**10 Offscreen Window**
**11 Z Window**

**KeyingColor** Transparency color key

**KeyingMask** Transparency color mask

**Note:** Source rectangle pixels have to be in the format of the destination window's pixels

**Note:** For the Offscreen buffer, everything is stored linearly.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before calling this entry point.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL   msiDrawSingleLine (     DWORD color,
                               DWORD XYStart,
                               DWORD XYEnd,
                               DWORD LineStyle  )
```

This entry point is used to draw a single line.

XYStart and XYEnd can be easily programmed by using the following define:

**#define PT( x , y )  ( ( (y) & 0xFFFF ) << 16 ) | ( (x)  & 0xFFFF ) )**

The parameters are explained here:

**color** Color to use for the lines. This must be formatted in the current pixel format.

**XYStart** The x,y value of the first point; using the **#define pt(x,y)** given above.

**XYEnd** The x,y value of the second point; using the **#define pt(x,y)** given above.

**LineStyle** Contains a 32 bit pattern to use while drawing the lines.
A '0' means no write, a '1' means write.

A linestyle of 0x00000000 is a solid linestyle as fully transparent lines are not very useful.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before calling this entry point.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

## *msiSetParameters General State change function*

```
BOOL msiSetParameters    ( T_msiParameters* pParameters )
```

This entry point is used to set the rendering parameters for texturing, z and color processing.

**Note:** Before doing any dumb frame buffer rendering, call this entry point with a NULL pointer. This way the MSI library can setup the device for direct frame buffer accesses. To quit the direct access mode, this entry point must be called with -1 or a valid pointer.

**Note:** When requesting the direct access mode, the game blocks waiting for the graphics engine to finish pending rendering. ***This may slow down the game if not used properly***.

**Note:** This function **must** be called inside a Start/End Frame pair when doing anything else than dumb frame buffer rendering.

### TEXTURE PARAMETERS

**Enable** Enable texturing. If this value is set to TRUE then the invW, u and v values of the vertex must be valid, otherwise they are ignored.

**Width, Height** Dimensions of the texture. These values must be power of 2 values. The Width also specifies the pixel pitch of the texture image.

**Planes** Specifies the bit per texel. For PseudoColor textures this value can be 4 or 8; for TrueColor textures it can be 15 or 16. PseudoColor textures contain an index to a true color palette in 5:6:5 RGB format.TrueColor textures of 15 bit contains colors in the 1:5:5:5 ARGB format and textures of 16 bits contain colors in the 5:6:5 RGB format.

**pMem** This value is a pointer to the texture image formatted according to the previous Width, Height and Planes values. The texture image can be located in a texture heap or in normal system memory. If the texture is already in the texture cache, this parameter must be set to NULL and CacheOffset is used to reference the texture.

**pHeap** This value is a pointer to a previously allocated texture heap where the texture referenced by pMem can be found. The value NULL means that the texture is not allocated in a texture heap.

**CacheOffset** This value is an offset in the texture cache memory where the texture can be found or where the texture referenced by pMem must be copied.
**The 5 LSB's must be 0**

**Clamp_u**
**Clamp_v** Indicates that the u and v texel coordinates must be clamped to the texture dimensions. Otherwise they wrap around on the texture boundaries.

**Modulate** Indicates that the texel color must be modulated with the surface modulation factors. The mr, mg and mb values of the vertex must be valid.

**Decal** Indicates that the texel color must be blended with the surface color. The blending is implemented as a transparency test. For PseudoColor textures the color keying gives the blending info. For TrueColor textures in the 1555 ARGB format, the alpha keying is used. For TrueColor textures in the 565 RGB format, the color keying is used. The r, g and b values of the vertex must be valid.

**Note**: If both Decal and Modulation are set to TRUE, then the rendering is performed in two passes. (**This is a slow process)**

**Transparency** Indicates that texture transparency is to be performed. Texture transparency is controlled by the color keying. Pixels covered by transparent texture areas are not written to the frame buffer.

**KeyingColor** Specifies the color key and the associated mask. The
**KeyingColorMask** texture is defined as transparent if the following equation is TRUE

texel & KeyingColorMask == KeyingColor

**Note:** The color key is specified in the texture format (4, 8 15 or 16 bit planes).

**KeyingAlpha** Specifies the alpha key and the associated mask. The
**KeyingAlphaMask** texture is defined as transparent if the following equation is TRUE

texel bit 15 & KeyingAlphaMask == KeyingAlpha

**Note:** The alpha key is 1 bit and is used for 15 bit TrueColor textures only.

## TEXTURE LUT PARAMETERS

**pMem** For PseudoColor textures, this value references the lookup table for texel indices. The lookup table contains 16 bit colors in the 5:6:5 RGB format. For 4 bit PseudoColor textures, this pointer references an array of 16 colors; for 8 bit PseudoColor textures, the array is of 256 colors. The LUT can be located in a texture heap or in normal system memory. If the LUT is in the texture cache, this parameter must be set to NULL and CacheOffset is used to reference the LUT.

**pHeap** This value is a pointer to a previously allocated texture heap where the LUT referenced by pMem can be found. The value NULL means that the LUT is not allocated in a texture heap.

**CacheOffset** This value is an offset in the texture cache memory where the LUT can be found or where the LUT referenced by pMem must be copied.
**The 5 LSB's must be 0**

## DEPTH PARAMETERS

**Enable** Enable the depth processing (compare and update)

**Compare** Define the z compare to use. msiCMP_*

**Protect** Specifies that Z buffer update must not be performed.

## COLOR PARAMETERS

**Dither** Indicate that dithering must be performed.

**Protect** Specifies that color buffer update must not be performed.

**Note: msiSetParameters has been optimized so that if you reload a CLUT with the same CacheOffset, it will not reload the CLUT. Furthermore, if you alternate between the same CLUT4 and CLUT8, only the first 16 entries of the CLUT will be reloaded.**

**Returns** TRUE if successful.

## *Texture State change functions*

```
BOOL msiSetTextureCLUT4(        DWORD Offset,
                                DWORD Index    )
```

This entry point is used to change the CLUT4 index and/or load the CLUT4 in this index. You can call this function to change a CLUT4 without calling msiSetParameters.

The Mystique Graphics Engine can have up to 16 cached CLUT4 entries.

Passing a value higher than 0x80000000 as the Offset will tell the Graphics Engine that it must now use the cached CLUT at index: Index.

The parameters are explained here:

**Offset** Offset inside the Mystique's texture memory to the CLUT4 to load.
**The 5 LSB's must be 0**

**Index** 0-15 indicate which cached entry is being used in the 16 CLUT4 spaces.

Examples:

msiSetCLUT4( 512, 0 ) will load the CLUT4 at offset 512 in the Mystique's memory into CLUT4 index 0. And set the used CLUT4 to index 0.

msiSetCLUT4( 1024, 1 )will load the CLUT4 at offset 1024 in the Mystique's memory into CLUT4 index 1. And set the used CLUT4 to index 1.

msiSetCLUT4( 0xFFFFFFFF, 0 )will set the used CLUT4 to be the one at index 0

msiSetCLUT4( 0xFFFFFFFF, 1 )will set the used CLUT4 to be the one at index 1

**Note: msiSetParameters has been optimized so that if you reload a CLUT with the same CacheOffset, it will not reload the CLUT. msiSetTextureCLUT4 is not optimized in this way. However when you load a CLUT to position 0, this new CLUT4's offset will be cached for msiSetParameters.**

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

BOOL **msiSetTextureCLUT8** ( DWORD Offset )

This entry point is used to change the CLUT8. You can call this function to change a CLUT8 without calling msiSetParameters.

The Mystique Graphics Engine can have 1 cached CLUT8 entry.

The parameters are explained here:

**Offset** Offset inside the Mystique's texture memory to the CLUT8 to load.
**The 5 LSB's must be 0**

**Note: msiSetParameters has been optimized so that if you reload a CLUT with the same CacheOffset, it will not reload the CLUT. MsiSetTextureCLUT8 is not optimized in this way. However when you load a CLUT8 in this way, we keep this offset for the next msiSetParameters.**

**Note: If you are currently set to CLUT4 (with msiSetParameters) and do a CLUT8 load with this function, only the first 16 entries of the CLUT8 will be loaded to index 0 of the Cached CLUT4 entries.**

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

BOOL **msiSetTextureTransparency** ( DWORD MaskKey )

This entry point is used to change the Transparency mask and key without calling msiSetParameters. This can be used to set transparency between triangles if you have selected transparency=True in a previous call to msiSetParameters.

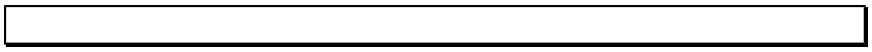
The parameters are explained here:

**MaskKey** Contains the mask and key of the color being used for transparency.
MaskKey = (Key & 0x0000FFFF) | (Mask & 0x0000FFFF)<<16
a MaskKey of 0x0000FFFF is always non transparent.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.



```
BOOL msiSetTextureOffset ( DWORD Offset )
```

This entry point is used to change the Offset to the current used texture without calling msiSetParameters. This can be used to change the used texture between triangles. Make sure that these different triangles are of the same type (same width, height, color planes...)

The parameters are explained here:

**Offset** Offset inside the Mystique's texture memory to the texture to use.
**The 5 LSB's must be 0**

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL msiSetTexturePlanes( LONG Planes )
```

This entry point is used to change the number of planes of the current used texture without calling msiSetParameters. This can be used to adjust the number of planes when changing the texture between triangles using msiSetTexOffset.

The Parameters are explained here:

**Planes** The number of planes in bits per pixel used by the current texture pointed to inside the Mystique's texture memory.
**The valid numbers of planes are 4, 8, 15, and 16 bits per pixel**

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL msiSetTextureSize( LONG Width, LONG Height )
```

This entry point is used to change the texture width and height parameters of the current used texture without calling msiSetParameters. This can be used to adjust the texture size and width when changing the texture between triangles using msiSetTexOffset.

The Parameters are explained here:

**Width** The Width of the texture in pixels.
**Height** The Height of the texture in pixels.

**Note:** The Parameters MUST be valid powers of 2 and must lie in the range 8 to 1024.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL msiSetTextureWrap( BOOL ClampU, BOOL ClampV )
```

This Entry point is used to set up the wrapping attributes without calling msiSetParameters.

The Parameters are explained here:

**ClampU** Indicates that the u and v texel coordinates must be clamped to the texture dimensions.
**ClampV** Otherwise they wrap around on the texture boundaries.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL msiSetTextureBlend( ULONG msiTexBlend )
```

This entry point is used to set the Decal and Modulate Parameters without calling msiSetParameters. Valid parameters are DECAL, MODULATE, and DECAL | MODULATE.

The Parameters are explained here:

**DECAL** Sets Decal Blending.. (See TEXTURE PARAMETERS above.)
**MODULATE** Sets Modulate Blending.. (See TEXTURE PARAMETERS above.)

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

BOOL **msiSetTextureEnable** ( BOOL Enable )

This entry point is used to select between Textured Triangles and Gouraud Shaded Triangles.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

## *General State change functions*

```
BOOL msiSetDepthCompare( DWORD Cmp )
```

This entry point is used to change how Z-Buffer comparison ( Depth Processing ) without calling msiSetParameters.

The Parameters are explained here:

**Cmp** The new Depth Compare parameter to be used for depth processing.

**The valid compare parameters are:**

**msiCMP_NEVER**
**msiCMP_LESS**
**msiCMP_EQUAL**
**msiCMP_LEQUAL**
**msiCMP_GREATER**
**msiCMP_NOTEQUAL**
**msiCMP_GEQUAL**
**msiCMP_ALWAYS**

**Note: If depth protection is on and the user calls this routine with a valid depth parameter, depth protection is automatically switched off and the depth processing set to the depth parameter.**

**Note: An invalid parameter will turn depth processing off.**
**e.g. msiSetTextureDepth(0).**

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

```
BOOL msiSetDitherEnable( BOOL Dither )
```

This entry point is used to turn color dithering on/off without calling msiSetParameters. Sending TRUE will turn color dithering on, and FALSE will turn it off.

**Note:** This function will return an error if MSIDOS is in direct access mode; msiSetParameters ( -1 or a valid pointer) must be called before to call this entry point in such situation.

**Note:** This function will return an error if it is not called between msiStartFrame and msiEndFrame.

**Returns** TRUE if successful.

## *Special functions*

```
BOOL msiSplashScreen( void )
```

This entry point will display a Matrox Splash Screen.

Game Developers should display this in their games.  It should be used right after a call to msiInit or at the very end of a game; right before a call to msiExit.

**Note:** This function will overwrite the Texture Cache.  Hence, don't load textures into the Texture Cache before calling this.

**Note:** This function will return an error if MSIDOS is in direct access mode.

**Note:** This function will return an error if it is called inside an msiStartFrame - msiEndFrame pair.

**Returns** TRUE if successful.

# Textures and LUT management

MSIDOS supports a very flexible texture management scheme. The game has full control over texture management and uses the many MSIDOS mechanisms to handle texture trashing, allocation, copy, etc … MSIDOS allows the game to store the texture anywhere: system memory, MSIDOS texture heaps or engine texture cache. There are performance issues related to the mechanism used to update the engine texture cache.

**Note:** The texture cache, texture heaps and the texture image are linear memory spaces.

The 3D graphics engine can only use textures present in the offscreen texture cache to perform texture mapped rendering. The game has a direct access to the texture cache to load textures. The game manages the texture cache sub-allocation.

As offscreen memory is a limited resource, there may not be enough texture cache memory to contain all the textures required for a frame. In this situation MSIDOS offers two mechanisms to update the texture cache with a new texture, trashing one or more textures in the texture cache.

MSIDOS implements texture heaps. A texture heap resides in system memory and is known by the 3D graphics engine so it can, in an autonomous way, access it to update the texture cache. The game uses the texture heap memory as any other system memory to store textures. A game can allocate many texture heaps as required and store many textures in a single heap.

If for some reasons it is not possible to have a texture in the texture cache or in a texture heap, MSIDOS allows textures to be located anywhere in system memory. There is a performance hit to have a texture outside a texture heap or the texture cache, as the 3D graphics engine cannot access directly the texture in such situation.

The following table shows how to use the possible mechanisms based on the msiTexture parameters specified in the T_msiParameters structure.

| pHeap | pMem | CacheOffset | Action |
|---|---|---|---|
| NULL | NULL | within texture cache | setup for actual cached texture |
| NULL | within system memory | within texture cache | copy from system memory to cache |
| VALID | within texture heap | within texture cache | copy from texture heap to cache |

a) If the texture is already in the texture cache, the game calls msiSetParameters with CacheOffset pointing to the texture in the texture cache. This is the fastest situation as no texture copy is required. In this situation the value of pHeap and pMem must be NULL.

b) If the texture is in a texture heap, the game calls msiSetParameters with pMem pointing to the texture in the texture heap, pHeap indicating the heap to use and CacheOffset indicating where to copy the texture in the texture cache.

c) If the texture is not in the texture cache nor in a texture heap, the game calls msiSetParameters with pMem pointing to the texture in system memory, pHeap must be NULL and CacheOffset indicating where to copy the texture in the texture cache.

Pseudo-color textures are associated with a LUT. As for the textures, the LUT can be located in different memory spaces.

The following table shows how to use the possible mechanisms based on the msiLUT parameters specified in the T_msiParameters structure.

| pHeap | pMem | CacheOffset | Action |
|---|---|---|---|
| NULL | NULL | within texture cache | setup for actual cached LUT |
| NULL | within system memory | within texture cache | copy from system memory to cache |
| VALID | within texture heap | within texture cache | copy from texture heap to cache |

# System considerations

a) In MSIDOS, certain pointers require the _far keyword. This is sometimes confusing to programmers since the API is compiled in a flat memory model. Please see the demo code as an example.

b) When running a MSIDOS game under a Win95 DOS Session, it is up to the game developer to provide a .pif or desktop shortcut that will not allow any switching of the game to the Win95 desktop (ex: ALT-TAB, CTRL-ESC,etc.) Switching to the desktop and back to the game will corrupt it.

c) A CASEEXACT option is also required when linking the final .exe.