S3D Pascal Bindings

by Kostas Michalopoulos
http://runtimeterror.com/
badsectoracula@gmail.com

ABSTRACT

This document describes the S3 Virge **S3D** API bindings for Free Pascal and Delphi. It includes some brief documentation for the API, though the official documentation isn't really that much better :-P.

1. About

S3DTK is a unit for Free Pascal and Delphi that provides bindings to the **S3D** API, used by the S3 Virge family of GPUs. The unit has been tested with Free Pascal 2.2.4 (the last version of Free Pascal to support Windows 95 out of the box) and Delphi 2 (though for Delphi i only tested that the unit compiles - to actually use the unit you need DirectDraw support and i do not know of some bindings that work with it - there are DirectX bindings for Delphi 4 and later however, so at least the unit should be useful for those).

I wrote this unit mainly in the interest of retrocoding - my main PC during the late 90s and early 2000s had an S3 Virge with a Pentium MMX and at the time i wasn't even sure if the card actually supported 3D rendering. Turns out it did, but it took a while to figure that out.

2. Quickstart

The official documentation is very sparse, but as a quickstart here is what you need to do:

- Create a fullscreen window to render into
- Initialize the library with *S3DTK_InitLib*.
- Create a renderer with *S3DTK_CreateRenderer*.
- Create an IDirectDraw instance - DirectDraw is mainly used to manage video memory.
- Setup exclusive and fullscreen cooperative level and set the video mode to 640x480x16bpp or whatever.
- Obtain the base of the framebuffer (see below) to use for calculating surface offsets.
- Create a primary surface with a secondary buffer attached to it to act as a swap chain.

- Create two *TS3DTK_SURFACE* variables to hold info about the surfaces. S3D actually needs very little info about a surface: its size, pixel/texel format and the offset in video memory from the framebuffer. This bit is the trickiest one.
- Optionally create a 16bit surface to hold a z buffer - again you need to do this via DirectDraw and *TS3DTK_SURFACE*.

And that is about it. Beyond that it is mainly setting state and drawing triangles. In terms of memory management, S3D relies on DirectDraw to do the video memory allocation/etc so in general the process is allocating a DirectDraw surface and filling a *TS3DTK_SURFACE* variable with info about it. The S3D API itself does not actually know about DirectDraw, it is only used indirectly to obtain the memory offsets and to perform surface flipping. In theory it might be possible to allocate a big surface and chop it to smaller pieces with multiple *TS3DTK_SURFACE*s.

The only tricky bit is calculating the video memory offset. This is only shown in the example that comes with the S3D SDK, but what you need to do is to call the ***Lock*** method with the ***DDLOCK_SURFACEMEMORYPTR*** and ***DDLOCK_WAIT*** flags (the wait flag isn't strictly necessary but the lock can actually fail if the GPU was using the surface and i had that happen to me a few times, so it doesn't hurt). Then cast the lpSurface to a ***ULONG*** and call *S3DTK_LinearToPhysical* with the upper 20 bits (so that the lower 12 bits remain as-is) and then subtract from the framebuffer's base. A utility function can be something like:

```
function LinearToPhysical(Linear: ULONG): ULONG;
begin
  Result:=S3DTK_LinearToPhysical(Linear and $FFFFF000) +
                                 (Linear and $FFF);
end;
```

and used like (***Surface*** is a ***IDirectDrawSurface***, ***ddsd*** is a ***TDDSurfaceDesc*** variable and ***s3ds*** is a *TS3DTK_SURFACE* variable):

```
ddsd.dwSize:=SizeOf(ddsd);
if Failed(Surface.Lock(nil, ddsd, DDLOCK_SURFACEMEMORYPTR or
                                  DDLOCK_WAIT, 0)) then
  Halt; // ...or a more graceful way to handle the error :-P
s3ds.sfOffset:=LinearToPhysical(ULONG(ddsd.lpSurface)) -
               FramebufferPhysical;
Surface.Unlock(nil);
```

The ***FramebufferPhysical*** variable is a global you obtain after the library and renderer have been created (***S3D*** is a pointer to *TS3DTK_FUNCTION_LIST* and ***FramebufferLinear*** is a ***PByte*** - though it can be any pointer type really):

```
S3Dˆ.S3DTK_GetState(S3D,
                    S3DTK_VIDEOMEMORYADDRESS,
                    ULONG(@LinearAddress));
FramebufferPhysical:=LinearToPhysical(ULONG(LinearAddress));
```

See *S3DTK_GetState* for information about the states you can query.

3. Reference

Below are the types exposed by the S3DTK unit. Note that all types use a *T* prefix as is common in Pascal naming conventions and they also have a pointer equivalent with a *P* prefix (e.g. ***PS3DTK_LIB_INIT*** for *TS3DTK_LIB_INIT*):

- *TS3DTKVALUE* - The floating point type used by S3D.
- *TS3DTK_LIB_INIT* - Library initialization parameters.
- *TS3DTK_RENDERER_INITSTRUCT* - Renderer creation parameters.
- *TS3DTK_SURFACE* - Describes an S3D surface used for framebuffer, zbuffer and textures.
- *TS3DTK_RECTAREA* - A rectangular area. This is really an alias for Windows' RECT type.
- *TS3DTK_VERTEX_LIT* - A vertex with color information without texture coordinates.
- *TS3DTK_VERTEX_TEX* - A vertex with color information without texture coordinates.
- *TS3DTK_FUNCTION_LIST* - S3D renderer function pointers.
- *PS3DTK_LIB_INIT* - Pointer to TS3DTK_LIB_INIT.
- *PS3DTK_RENDERER_INITSTRUCT* - Pointer to TS3DTK_RENDERER_INITSTRUCT.
- *PS3DTK_SURFACE* - Pointer to TS3DTK_SURFACE.
- *PS3DTK_RECTAREA* - Pointer to TS3DTK_RECTAREA.
- *PS3DTK_VERTEX_LIT* - Pointer to TS3DTK_VERTEX_LIT.
- *PS3DTK_VERTEX_TEX* - Pointer to TS3DTK_VERTEX_TEX.
- *PS3DTK_FUNCTION_LIST* - Pointer to TS3DTK_FUNCTION_LIST.

Also the exposed functions:

- *S3DTK_InitLib* - Initialize the library.
- *S3DTK_ExitLib* - Shut down the library.
- *S3DTK_CreateRenderer* - Create a renderer instance and provide functions for accessing it.
- *S3DTK_DestroyRenderer* - Create a renderer instance and provide functions for accessing it.
- *S3DTK_PhysicalToLinear* - Map a physical address and size to a linear address.
- *S3DTK_LinearToPhysical* - Find the physical address the given linear address is mapped.
- *S3DTK_EnterCritical* - Serialize access to S3D among threads.
- *S3DTK_ReleaseCritical* - Release the previously obtain critical section.

3.1. Types Reference

Note that all types use a *T* prefix as is common in Pascal naming conventions and they also have a pointer equivalent with a *P* prefix (e.g. ***PS3DTK_LIB_INIT*** for *TS3DTK_LIB_INIT*):

- *TS3DTKVALUE* - The floating point type used by S3D.
- *TS3DTK_LIB_INIT* - Library initialization parameters.
- *TS3DTK_RENDERER_INITSTRUCT* - Renderer creation parameters.
- *TS3DTK_SURFACE* - Describes an S3D surface used for framebuffer, zbuffer and textures.
- *TS3DTK_RECTAREA* - A rectangular area. This is really an alias for Windows' RECT type.
- *TS3DTK_VERTEX_LIT* - A vertex with color information without texture coordinates.
- *TS3DTK_VERTEX_TEX* - A vertex with color information without texture coordinates.
- *TS3DTK_FUNCTION_LIST* - S3D renderer function pointers.
- *PS3DTK_LIB_INIT* - Pointer to TS3DTK_LIB_INIT.
- *PS3DTK_RENDERER_INITSTRUCT* - Pointer to TS3DTK_RENDERER_INITSTRUCT.
- *PS3DTK_SURFACE* - Pointer to TS3DTK_SURFACE.
- *PS3DTK_RECTAREA* - Pointer to TS3DTK_RECTAREA.
- *PS3DTK_VERTEX_LIT* - Pointer to TS3DTK_VERTEX_LIT.
- *PS3DTK_VERTEX_TEX* - Pointer to TS3DTK_VERTEX_TEX.
- *PS3DTK_FUNCTION_LIST* - Pointer to TS3DTK_FUNCTION_LIST.

3.1.1. TS3DTKVALUE

The floating point type used by S3D.

Declaration:

```
TS3DTKVALUE = Single;
```

Unit: S3DTK

3.1.2. TS3DTK_LIB_INIT

Library initialization parameters.

Declaration:

```
TS3DTK_LIB_INIT = record
  libFlags: ULONG;
  libVideoBufferLinAddr: ULONG;
  libMMIOSpaceLinAddr: ULONG;
end;
```

Fields:

- *libFlags* - Flags for initialization, must be ***S3DTK_INITPIO***. For making a windowed application this should also be OR'd with ***S3DTK_INIT2D_SERIALIZATION_ON***.

- *libVideoBufferLinAddr* - Reserved and must be 0.
- *libMMIOSpaceLinAddr* - Reserved and must be 0.

Description:

Using this type is not really necessary for fullscreen applications as the default assumption when calling *S3DTK_InitLib* with a nil parameter is to use the ***S3DTK_INITPIO*** flag.

Unit: S3DTK

See Also:

S3DTK_InitLib

3.1.3. TS3DTK_RENDERER_INITSTRUCT

Renderer creation parameters.

Declaration:

```
TS3DTK_RENDERER_INITSTRUCT = record
  initFlags: ULONG;
  initUserID: ULONG;
  initAppID: ULONG;
end;
```

Fields:

- *initFlags* - Flags for renderer initialization. Can be a logical OR of any of the flags mentioned below.
- *initUserID* - Reserved and must be 0.
- *initAppID* - Reserved and must be 0.

Description:

The flags field must be one of the following:

- ***S3DTK_FORMAT_FLOAT*** - Vertices use a floating point format.
- ***S3DTK_FORMAT_UVRANGE*** - Ensure the UV coordinates in texture mapping are within a range of 128 units. Setting this flag can make things a bit slower.
- ***S3DTK_FORMAT_XYRANGE*** - Ensure the XY coordinates when rendering are always within the rendering viewport. Setting this flag can make things a bit slower too.

Using this type is not really necessary as *S3DTK_CreateRenderer* will assume all of the above flags when a nil pointer is passed for initialization.

Unit: S3DTK

See Also:

S3DTK_CreateRenderer

3.1.4. TS3DTK_SURFACE

Describes an S3D surface used for framebuffer, zbuffer and textures.

Declaration:

```
TS3DTK_SURFACE = record
  sfOffset: ULONG;
  sfWidth: ULONG;
  sfHeight: ULONG;
  sfFormat: ULONG;
  reserved: array [0..4] of ULONG;
end;
```

Fields:

- *sfOffset* - Where in memory the surface is, depends on where the surface resides.
- *sfWidth* - Surface width.
- *sfHeight* - Surface height.
- *sfFormat* - Pixel or texel format, can be one of the values below.
- *reserved* - Reserved, should be set to 0s.

Description:

For surfaces containing drawable 2D image data (e.g. framebuffer, sprites, etc), the format can be one of the following values:

- ***S3DTK_VIDEORGB8*** - 8bit per pixel where each pixel is a palette index.
- ***S3DTK_VIDEORGB15*** - 16bit per pixel with a 0RRRRRGGGGGBBBBB arrangement (5 bits per component).
- ***S3DTK_VIDEORGB24*** - 24bit per pixel with a RRRRRRRRGGGGGGGGBBBBBBBB arrangement.

For textures the format can be one of the following values:

- ***S3DTK_TEXARGB8888*** - 32bit per texel with a AAAAAAAARRRRRRRRGGGGGGGGBBBBBBBB arrangement.
- ***S3DTK_TEXARGB4444*** - 16bit per texel with a AAAARRRRGGGGBBBB arrangement.
- ***S3DTK_TEXARGB1555*** - 16bit per texel with a ARRRRRGGGGGBBBBB arrangement.
- ***S3DTK_TEXPALETTIZED8*** - 8bit per texel where each texel is a palette index.

For zbuffers the only acceptable value is ***S3DTK_Z16*** for 16bit per pixel.

Surface location:

Surfaces can reside in either video or system memory. When a surface is in system memory the sfOffset field is the linear memory address (a 32bit pointer casted to ***ULONG***) and their format must be OR'd with ***S3DTK_SYSTEM***. For surfaces in video memory the field is actually the offset from the framebuffer which can be calculated as explained in the *Quickstart* section. The format can be OR'd with ***S3DTK_VIDEO*** but this is just for documentation purposes as the value of that constant is 0.

Unit: S3DTK

See Also:

S3DTK_LinearToPhysical

3.1.5. TS3DTK_RECTAREA

A rectangular area. This is really an alias for Windows' ***RECT*** type.

Declaration:

```
TS3DTK_RECTAREA = Windows.TRECT;
```

Fields:

- *Left* - Leftmost pixel.
- *Top* - Topmost pixel.
- *Right* - Rightmost pixel.
- *Bottom* - Bottommost pixel.

Description:

Note that often the Right and Bottom values are *not* inclusive!

Unit: S3DTK

3.1.6. TS3DTK_VERTEX_LIT

A vertex with color information without texture coordinates.

Declaration:

```
TS3DTK_VERTEX_LIT = record
  X: TS3DTKVALUE;
  Y: TS3DTKVALUE;
  Z: TS3DTKVALUE;
  W: TS3DTKVALUE;
  B: BYTE;
```

```
  G: BYTE;
  R: BYTE;
  A: BYTE;
end;
```

Fields:

- *X* - X coordinate in screen space (pixels). Remember that *TS3DTKVALUE* is a floating point type - vertices can lie “between” pixels. This must be a value between 0 to 4095, inclusive.
- *Y* - Y coordinate in screen space. Similar to X. Also must be between 0 to 4095.
- *Z* - Z value to use for z buffering. Only 16bit zbuffers are supported so this must be a value between 0 to 65535.
- *W* - Ignored.
- *B* - Blue component for the color.
- *G* - Green component for the color.
- *R* - Red component for the color.
- *A* - Alpha component for the color, used for alpha blending and fog density.

Description:

This record is used to draw triangles without any texture applied to them (ie. gouraud shading). The W field is ignored and only exists so that the same vertex data can be used for both textured and untextured rendering.

Unit: S3DTK

See Also:

TS3DTK_VERTEX_TEX

3.1.7. TS3DTK_VERTEX_TEX

A vertex with color information without texture coordinates.

Declaration:

```
TS3DTK_VERTEX_TEX = record
  X: TS3DTKVALUE;
  Y: TS3DTKVALUE;
  Z: TS3DTKVALUE;
  W: TS3DTKVALUE;
  B: BYTE;
  G: BYTE;
  R: BYTE;
  A: BYTE;
  D: TS3DTKVALUE;
  U: TS3DTKVALUE;
```

```
  V: TS3DTKVALUE;
end;
```

Fields:

- *X* - X coordinate in screen space (pixels). Remember that *TS3DTKVALUE* is a floating point type - vertices can lie “between” pixels. This must be a value between 0 to 4095, inclusive.
- *Y* - Y coordinate in screen space. Similar to X. Also must be between 0 to 4095.
- *Z* - Z value to use for z buffering. Only 16bit zbuffers are supported so this must be a value between 0 to 65535.
- *W* - Actually Z value in clip space, used for perspective texture mapping.
- *B* - Blue component for the color.
- *G* - Green component for the color.
- *R* - Red component for the color.
- *A* - Alpha component for the color, used for alpha blending and fog density.
- *D* - Level of detail for mipmapped textures - this can be filled by the library.
- *U* - U texture coordinate in texels (this is *not* normalized, the middle of a 32x32 texture is at 16 not at 0.5). Must be between 0 and 2047, inclusive.
- *V* - V texture coordinate in texels. Similar to U.

Description:

This record is used to draw triangles with texture mapping applied to them.

Unit: S3DTK

See Also:

TS3DTK_VERTEX_LIT

3.1.8. TS3DTK_FUNCTION_LIST

S3D renderer function pointers.

Declaration:

```
TS3DTK_FUNCTION_LIST = record
  S3DTK_SetState: function(
    pFuncStruct: Pointer;
    State: ULONG;
    Value: ULONG): ULONG; cdecl;
  S3DTK_GetState: function(
    pFuncStruct: Pointer;
    State: ULONG;
    Value: ULONG): ULONG; cdecl;
  S3DTK_TriangleSet: function(
    pFuncStruct: Pointer;
```

```
    pVertexSet: PULONG;
    NumVertexes: ULONG;
    SetType: ULONG): ULONG; cdecl;
  S3DTK_TriangleSetEx: function(
    pFuncStruct: Pointer;
    pVertexSet: PULONG;
    NumVertexes: ULONG;
    SetType: ULONG;
    pSetState: PULONG;
    NumStates: ULONG): ULONG; cdecl;
  S3DTK_BitBlt: function(
    pFuncStruct: Pointer;
    pDestSurface: TS3DTK_LPSURFACE;
    pDestRect: TS3DTK_LPRECTAREA;
    pSrcSurface: TS3DTK_LPSURFACE;
    pSrcRect: TS3DTK_LPRECTAREA): ULONG; cdecl;
  S3DTK_BitBltTransparent: function(
    pFuncStruct: Pointer;
    pDestSurface: TS3DTK_LPSURFACE;
    pDestRect: TS3DTK_LPRECTAREA;
    pSrcSurface: TS3DTK_LPSURFACE;
    pSrcRect: TS3DTK_LPRECTAREA;
    TransparentColor: ULONG): ULONG; cdecl;
  S3DTK_RectFill: function(
    pFuncStruct: Pointer;
    pDestSurface: TS3DTK_LPSURFACE;
    pDestRect: TS3DTK_LPRECTAREA;
    FillPattern: ULONG): ULONG; cdecl;
  S3DTK_GetLastError: function(
    pFuncStruct: Pointer): Integer; cdecl;
end;
```

Methods:

- *S3DTK_SetState* - Set a S3D state.
- *S3DTK_GetState* - Get a S3D state.
- *S3DTK_TriangleSet* - Draw triangles.
- *S3DTK_TriangleSetEx* - Set state and draw triangles.
- *S3DTK_BitBlt* - Copy a rectangular area from one surface to another.
- *S3DTK_BitBltTransparent* - Copy a rectangular area from one surface to another with color keying.
- *S3DTK_RectFill* - Fill a rectangular area in video memory with the given value.
- *S3DTK_GetLastError* - Provides details for a failed call.

Description:

This record contains the function pointers returned by *S3DTK_CreateRenderer* that can be used to make calls to the renderer, like altering its state, drawing triangles, etc.

Unit: S3DTK

See Also:

S3DTK_CreateRenderer

3.1.8.1. S3DTK_SetState

Set a S3D state.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_SetState: function(
  pFuncStruct: Pointer;
  State: ULONG;
  Value: ULONG): ULONG; cdecl;
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.
- *State* - One of the state constants described below.
- *Value* - The new state. The actual value depends on the state constant.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Description:

Set a new S3D state. The actual state will be applied once new triangles are rendered.

States:

The valid values for the ***State*** parameter and the potential values the ***Value*** parameter can have are the following:

- ***S3DTK_ALPHABLENDING*** - Controls alpha blending. The value can be one of ***S3DTK_ALPHAOFF***, which disables alpha blending, ***S3DTK_ALPHATEXTURE***, which enables alpha blending and takes alpha from the texture or ***S3DTK_ALPHASOURCE***, which enables alpha blending and takes alpha from the value stored in the vertices. Note that when fog is

enabled this cannot be ***S3DTK_ALPHASOURCE*** since the alpha value in the vertices is used to control fog density.

- ***S3DTK_CLIPPING_AREA*** - Value is a pointer to a *TS3DTK_RECTAREA* that contains the clipping rectangle inside the rendering surface.
- ***S3DTK_D_LEVEL_SUPPLIED*** - If this is set to ***S3DTK_ON*** the library will use the D values for mipmapping provided by the vertex data, otherwise it will be calculated automatically (default is off).
- ***S3DTK_DRAWSURFACE*** - Value is a pointer to a *TS3DTK_SURFACE* that will be used for all further rendering operations.
- ***S3DTK_FOGCOLOR*** - Controls fog. If the value is ***S3DTK_FOGOFF*** (the default), the fog will be turned off. Otherwise the value controls the fog color which is calculated by mixing the color to be written with the fog color using the alpha value stored in the vertex data (i.e. final_pixel = (1.0 - alpha) * fog + alpha * color_to_be_written). The fog color must be in the same format as the current draw surface. Note that fog cannot be used at the same time as alpha blending with vertex sourced alpha values.
- ***S3DTK_RENDERINGTYPE*** - Controls how to render the triangles. Can be one of ***S3DTK_GOURAUD***, which only uses vertex colors, ***S3DTK_LITTEXTURE***, which performs affine texture mapping and blends the texture colors with the vertex colors, ***S3DTK_UNLITTEXTURE***, which performs affine texture mapping while ignoring vertex colors, ***S3DTK_LITTEXTUREPERSPECT***, which performs perspective correct texture mapping and blends the texture colors with the vertex colors or ***S3DTK_UNLITTEXTUREPERSPECT*** which performs perspective correct texture mapping while ignoring vertex colors.
- ***S3DTK_TEXBLENDINGMODE*** - Controls how to combine the texture colors with the vertex colors and can be either ***S3DTK_TEXMODULATE*** which multiplies the colors together or ***S3DTK_TEXDECAL*** which blends the two colors using the alpha value stored in the texture.
- ***S3DTK_TEXFILTERINGMODE*** - Texture filtering mode, can be one of ***S3DTK_NEAREST***, ***S3DTK_LINEAR***, ***S3DTK_MIP_NEAREST***, ***S3DTK_LINEAR_MIP_NEAREST***, ***S3DTK_MIP_LINEAR*** or ***S3DTK_LINEAR_MIP_LINEAR***.
- ***S3DTK_TEXTUREACTIVE*** - Value is a pointer to a *TS3DTK_SURFACE* for the texture to use when drawing triangles with a render type that uses textures (ie. anything other than ***S3DTK_GOURAUD***).
- ***S3DTK_ZBUFFERCOMPAREMODE*** - Controls zbuffer compare mode. Can be one of ***S3DTK_ZNEVERPASS*** (never passes the z test), ***S3DTK_ZSRCGTZFB*** (rendered > already there), ***S3DTK_ZSRCEQZFB*** (rendered = already there), ***S3DTK_ZSRCGEZFB*** (rendered >= already there), ***S3DTK_ZSRCLSZFB*** (rendered < already there), ***S3DTK_ZSRCNEZFB*** (rendered <> already there), ***S3DTK_ZSRCLEZFB*** (rendered <= already there), ***S3DTK_ZALWAYSPASS*** (always pass the z test).
- ***S3DTK_ZBUFFERENABLE*** - Enable or disable z buffering, can be ***S3DTK_ON*** or ***S3DTK_OFF***.
- ***S3DTK_ZBUFFERSURFACE*** - Pointer to a *TS3DTK_SURFACE* to be used as a zbuffer.

- ***S3DTK_ZBUFFERUPDATEENABLE*** - When this is set to ***S3DTK_ON*** (default) the zbuffer is updated with new values as pixels are drawn on the draw buffer. Setting this to ***S3DTK_OFF*** will not disable any writes (comparisons are still being made).

Some state values are actually pointers, in which case the pointers must be casted to ***ULONG***.

Unit: S3DTK

See Also:

S3DTK_GetState (in *TS3DTK_FUNCTION_LIST*)

3.1.8.2. S3DTK_GetState

Get a S3D state.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_GetState: function(
  pFuncStruct: Pointer;
  State: ULONG;
  Value: ULONG): ULONG; cdecl;
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.
- *State* - One of the state constants described below.
- *Value* - Pointer to memory that will receive the data (yes, this is really meant to be a pointer).

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

States:

The valid values for the ***State*** parameter and the potential values the ***Value*** parameter can have are listed in *S3DTK_SetState*. In addition to the values listed there, the following values can also be queried:

- ***S3DTK_VERSION*** - Returns the version of the S3D library in a ***ULONG***. The format is *$MMmm* where MM is the major version and mm is the minor version.

- ***S3DTK_VIDEOMEMORYADDRESS*** - Returns the linear address of the video memory. Use *S3DTK_LinearToPhysical* to convert it to a physical address that can be used for calculating offsets for surfaces in video memory.
- ***S3DTK_DISPLAYADDRESSUPDATED*** - If this is ***S3DTK_YES*** the display address has changed.
- ***S3DTK_GRAPHICS_ENGINE_IDLE*** - Returns ***S3DTK_TRUE*** if the S3D engine is idle. This could be useful for updating textures, however using ***DDLOCK_WAIT*** during locking is actually simpler.

Unit: S3DTK

See Also:

S3DTK_SetState (in *TS3DTK_FUNCTION_LIST*)

3.1.8.3. S3DTK_TriangleSet

Draw triangles.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_TriangleSet: function(
  pFuncStruct: Pointer;
  pVertexSet: PULONG;
  NumVertexes: ULONG;
  SetType: ULONG): ULONG; cdecl;
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.
- *pVertexSet* - A pointer to an array of *pointers* where each pointer points to a *TS3DTK_VERTEX_LIT* or *TS3DTK_VERTEX_TEX*.
- *NumVertexes* - Number of vertices to draw from the array.
- *SetType* - The type of primitives. This can be one of ***S3DTK_TRILIST***, ***S3DTK_TRISTRIP*** or ***S3DTK_TRIFAN***. See below.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Description:

This can be used to draw triangles in the form of a list of N/3 separate triangles, a triangle strip made out of N-2 triangles where after the first triangle the rest are defined by the shared edge following a zig-zag pattern or triangle fan made out of

N-2 triangles where the first vertex is shared by all triangles and the other vertices specify the non-shared edges.

Notice that vertex data is specified as an array of pointers, which allows reuse of shared vertices by having the same vertex pointed to by different pointers.

Unit: S3DTK

See Also:

S3DTK_TriangleSetEx (in *TS3DTK_FUNCTION_LIST*)

3.1.8.4. S3DTK_TriangleSetEx

Set state and draw triangles.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_TriangleSetEx: function(
  pFuncStruct: Pointer;
  pVertexSet: PULONG;
  NumVertexes: ULONG;
  SetType: ULONG
  pSetState: PULONG;
  NumStates: ULONG): ULONG; cdecl;
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.
- *pVertexSet* - A pointer to an array of *pointers* where each pointer points to a *TS3DTK_VERTEX_LIT* or *TS3DTK_VERTEX_TEX*.
- *NumVertexes* - Number of vertices to draw from the array.
- *SetType* - The type of primitives. This can be one of ***S3DTK_TRILIST***, ***S3DTK_TRISTRIP*** or ***S3DTK_TRIFAN***.
- *pSetState* - Pointer to an array of ***ULONG*** key-value pairs.
- *NumStates* - The number of pairs in the array.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Description:

This is the same as a combination of *S3DTK_SetState* followed by *S3DTK_TriangleSet*. The values for the state are the same as the former.

Unit: S3DTK

See Also:

S3DTK_TriangleSet (in *TS3DTK_FUNCTION_LIST*), *S3DTK_SetState* (in *TS3DTK_FUNCTION_LIST*)

3.1.8.5. S3DTK_BitBlt

Copy a rectangular area from one surface to another.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_BitBlt: function(
  pFuncStruct: Pointer;
  pDestSurface: TS3DTK_LPSURFACE;
  pDestRect: TS3DTK_LPRECTAREA;
  pSrcSurface: TS3DTK_LPSURFACE;
  pSrcRect: TS3DTK_LPRECTAREA): ULONG; cdecl;
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.
- *pDestSurface* - Pointer to *TS3DTK_SURFACE* for the destination surface. The destination surface must be in video memory.
- *pDestRect* - Pointer to a *TS3DTK_RECTAREA* for the area in the destination surface to write to.
- *pSrcSurface* - Pointer to a *TS3DTK_SURFACE* for the source surface.
- *pSrcRect* - Pointer to a *TS3DTK_RECTAREA* for the area in the source surface to copy from.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Limitations:

Both surfaces must have the same pixel format and size. Also the destination surface must be in video memory.

Unit: S3DTK

See Also:

S3DTK_BitBltTransparent (in *TS3DTK_FUNCTION_LIST*)

3.1.8.6. S3DTK_BitBltTransparent

Copy a rectangular area from one surface to another with color keying.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_BitBltTransparent: function(
  pFuncStruct: Pointer;
  pDestSurface: TS3DTK_LPSURFACE;
  pDestRect: TS3DTK_LPRECTAREA;
  pSrcSurface: TS3DTK_LPSURFACE;
  pSrcRect: TS3DTK_LPRECTAREA;
  TransparentColor: ULONG): ULONG; cdecl;
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.
- *pDestSurface* - Pointer to a *TS3DTK_SURFACE* for the destination surface. The destination surface must be in video memory.
- *pDestRect* - Pointer to a *TS3DTK_RECTAREA* for the area in the destination surface to write to.
- *pSrcSurface* - Pointer to a *TS3DTK_SURFACE* for the source surface.
- *pSrcRect* - Pointer to a *TS3DTK_RECTAREA* for the area in the source surface to copy from.
- *TransparentColor* - The color key that marks the transparent pixels.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Description:

This works similarly to *S3DTK_BitBlt* but it also performs color keying against the specified color. The value depends on the format of the surface - for 8bit surfaces this is the index of the palette color whereas for 15bit surfaces this is the value to check for (only the lower 15 bits are taken into account).

Limitations:

Both surfaces must have the same pixel format and size. Also the destination surface must be in video memory. This only supports ***S3DTK_VIDEORGB8*** and ***S3DTK_VIDEORGB15*** formats. Other formats are not supported.

Unit: S3DTK

See Also:

S3DTK_BitBlt (in *TS3DTK_FUNCTION_LIST*)

3.1.8.7. S3DTK_RectFill

Fill a rectangular area in video memory with the given value.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_RectFill: function(
  pFuncStruct: Pointer;
  pDestSurface: TS3DTK_LPSURFACE;
  pDestRect: TS3DTK_LPRECTAREA;
  FillPattern: ULONG
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.
- *pDestSurface* - Pointer to a *TS3DTK_SURFACE* for the destination surface. The destination surface must be in video memory.
- *pDestRect* - Pointer to a *TS3DTK_RECTAREA* for the area in the destination surface to fill.
- *FillPattern* - The value to fill. This must match the surface format.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Description:

This can be used to clear the draw buffer and zbuffer or fill parts of them.

Unit: S3DTK

3.1.8.8. S3DTK_GetLastError

Provides details for a failed call.

Method of:

TS3DTK_FUNCTION_LIST

Syntax:

```
S3DTK_GetLastError: function(
  pFuncStruct: Pointer): Integer; cdecl;
```

Parameters:

- *pFuncStruct* - Pointer to the *TS3DTK_FUNCTION_LIST* that contains this method.

Returns:

One of the error codes described below.

Description:

When a function returns ***S3DTK_ERR*** this function can be used right next to provide additional information about the error. Note that this function is only usable with function calls made after *S3DTK_CreateRenderer*.

The possible error codes are:

- ***S3DTK_CANTCONVERT*** - Cannot convert form linear to physical address.
- ***S3DTK_INVALIDFILTERINGMODE*** - Invalid filtering mode.
- ***S3DTK_INVALIDSURFACEFORMAT*** - Invalid surface format.
- ***S3DTK_INVALIDRENDERINGTYPE*** - Invalid rendering type.
- ***S3DTK_INVALIDVALUE*** - Invalid value.
- ***S3DTK_NULLPOINTER*** - Unexpected nil pointer.
- ***S3DTK_RENDERINGSURFISNOTSET*** - There is no surface set for rendering to.
- ***S3DTK_UNSUPPORTEDKEY*** - Unsupported state constant.
- ***S3DTK_UNSUPPORTEDMETHOD*** - Tried to call an unsupported method.
- ***S3DTK_UNSUPPORTEDVIDEOMODE*** - (this is really only available on DOS)

Unit: S3DTK

3.1.9. PS3DTK_LIB_INIT

Pointer to *TS3DTK_LIB_INIT*.

Declaration:

```
PS3DTK_LIB_INIT = ^TS3DTK_LIB_INIT;
```

Unit: S3DTK

3.1.10. PS3DTK_RENDERER_INITSTRUCT

Pointer to *TS3DTK_RENDERER_INITSTRUCT*.

Declaration:

```
PS3DTK_RENDERER_INITSTRUCT = ^TS3DTK_RENDERER_INITSTRUCT;
```

Unit: S3DTK

3.1.11. PS3DTK_SURFACE

Pointer to *TS3DTK_SURFACE*.

Declaration:

```
PS3DTK_SURFACE = ^TS3DTK_SURFACE;
```

Unit: S3DTK

3.1.12. PS3DTK_RECTAREA

Pointer to *TS3DTK_RECTAREA*.

Declaration:

```
PS3DTK_RECTAREA = ^TS3DTK_RECTAREA;
```

Unit: S3DTK

3.1.13. PS3DTK_VERTEX_LIT

Pointer to *TS3DTK_VERTEX_LIT*.

Declaration:

```
PS3DTK_VERTEX_LIT = ^TS3DTK_VERTEX_LIT;
```

Unit: S3DTK

3.1.14. PS3DTK_VERTEX_TEX

Pointer to *TS3DTK_VERTEX_TEX*.

Declaration:

```
PS3DTK_VERTEX_TEX = ^TS3DTK_VERTEX_TEX;
```

Unit: S3DTK

3.1.15. PS3DTK_FUNCTION_LIST

Pointer to *TS3DTK_FUNCTION_LIST*.

Declaration:

```
PS3DTK_FUNCTION_LIST = ^TS3DTK_FUNCTION_LIST;
```

Unit: S3DTK

3.2. Function Reference

- *S3DTK_InitLib* - Initialize the library.
- *S3DTK_ExitLib* - Shut down the library.
- *S3DTK_CreateRenderer* - Create a renderer instance and provide functions for accessing it.
- *S3DTK_DestroyRenderer* - Create a renderer instance and provide functions for accessing it.
- *S3DTK_PhysicalToLinear* - Map a physical address and size to a linear address.
- *S3DTK_LinearToPhysical* - Find the physical address the given linear address is mapped.
- *S3DTK_EnterCritical* - Serialize access to S3D among threads.
- *S3DTK_ReleaseCritical* - Release the previously obtain critical section.

3.2.1. S3DTK_InitLib

Initialize the library.

Syntax:

```
function S3DTK_InitLib(
  lParam: ULONG): ULONG; cdecl;
```

Parameters:

- *lParam* - Pointer to a *TS3DTK_LIB_INIT* with initialization settings or 0 to use the default settings (equivalent to passing just ***S3DTK_INITPIO*** for fullscreen applications).

Returns:

S3DTK_OK if successful, ***S3DTK_3DCAPNOTSUPPORTED*** if the hardware does not support 3D rendering, ***S3DTK_CANTCONVERT*** if the linear address of the video buffer cannot be determined (this shouldn't happen under Windows) or ***S3DTK_NOS3KERNEL*** if the kernel device driver is not installed (in case you distribute the *S3DTKW.DLL* yourself but rely on the driver to be preinstalled.

Note:

After calling this function to initialize the library you should call *S3DTK_CreateRenderer* to create the renderer and obtain the functions for performing rendering calls.

Unit: S3DTK

See Also:

S3DTK_CreateRenderer, *S3DTK_ExitLib*

3.2.2. S3DTK_ExitLib

Shut down the library.

Syntax:

```
function S3DTK_ExitLib: ULONG; cdecl;
```

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Note:

Before calling this function ensure that the renderer has been destroyed by calling *S3DTK_DestroyRenderer*.

Unit: S3DTK

See Also:

S3DTK_DestroyRenderer, *S3DTK_InitLib*

3.2.3. S3DTK_CreateRenderer

Create a renderer instance and provide functions for accessing it.

Syntax:

```
function S3DTK_CreateRenderer(
  Param1: ULONG;
  ppFunctionList: PPointer): ULONG; cdecl;
```

Parameters:

- *Param1* - Pointer to a *TS3DTK_RENDERER_INITSTRUCT* with initialization settings or 0 to use the default settings (equivalent to passing the flags ***S3DTK_FORMAT_FLOAT***, ***S3DTK_FORMAT_UVRANGE*** and ***S3DTK_FORMAT_XYRANGE*** OR'd together - note that this can be suboptimal if the application performs clipping!).
- *ppFunctionList* - Pointer to a *TS3DTK_FUNCTION_LIST* to receive function pointers for performing renderer calls.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Unit: S3DTK

See Also:

S3DTK_InitLib, *S3DTK_DestroyRenderer*

3.2.4. S3DTK_DestroyRenderer

Create a renderer instance and provide functions for accessing it.

Syntax:

```
function S3DTK_DestroyRenderer(
  ppFunctionList: PPointer): ULONG; cdecl;
```

Parameters:

- *ppFunctionList* - The *TS3DTK_FUNCTION_LIST* pointer that *S3DTK_CreateRenderer* returned.

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Unit: S3DTK

See Also:

S3DTK_ExitLib, *S3DTK_CreateRenderer*

3.2.5. S3DTK_PhysicalToLinear

Map a physical address and size to a linear address.

Syntax:

```
function S3DTK_PhysicalToLinear(
  PhysAddr: ULONG;
  Size: ULONG): ULONG; cdecl;
```

Parameters:

- *PhysAddr* - The physical address.
- *Size* - Size of the area to be mapped.

Returns:

The linear address (can be cast to a pointer) or ***S3DTK_ERR***.

Unit: S3DTK

See Also:

S3DTK_LinearToPhysical

3.2.6. S3DTK_LinearToPhysical

Find the physical address the given linear address is mapped.

Syntax:

```
function S3DTK_LinearToPhysical(
  Linear: ULONG): ULONG; cdecl;
```

Parameters:

- *Linear* - The linear address (can be cast from a pointer).

Returns:

The physical address or ***S3DTK_ERR***.

Unit: S3DTK

See Also:

TS3DTK_SURFACE, *S3DTK_PhysicalToLinear*

3.2.7. S3DTK_EnterCritical

Serialize access to S3D among threads.

Syntax:

```
function S3DTK_EnterCritical: ULONG; cdecl;
```

Returns:

S3DTK_OK if successful, ***S3DTK_ERR*** otherwise.

Description:

Together with *S3DTK_ReleaseCritical* this can be used to synchronize calls between threads using S3D. Note that there must not be any OS calls between ***S3DTK_EnterCritical*** and *S3DTK_ReleaseCritical*.

Unit: S3DTK

See Also:

S3DTK_ReleaseCritical

3.2.8. S3DTK_ReleaseCritical

Release the previously obtain critical section.

Syntax:

```
function S3DTK_ReleaseCritical: ULONG; cdecl;
```

Returns:

Always ***S3DTK_OK***.

Description:

Together with *S3DTK_EnterCritical* this can be used to synchronize calls between threads using S3D. Note that there must not be any OS calls between *S3DTK_EnterCritical* and ***S3DTK_ReleaseCritical***.

Unit: S3DTK