RenderIt 3D!

User's Guide

By Steven J. Eschweiler

Programmed by Steven J. Eschweiler
Indotek Software Productions
www.indotek.com

Table of Contents

Getting Started

Requirements

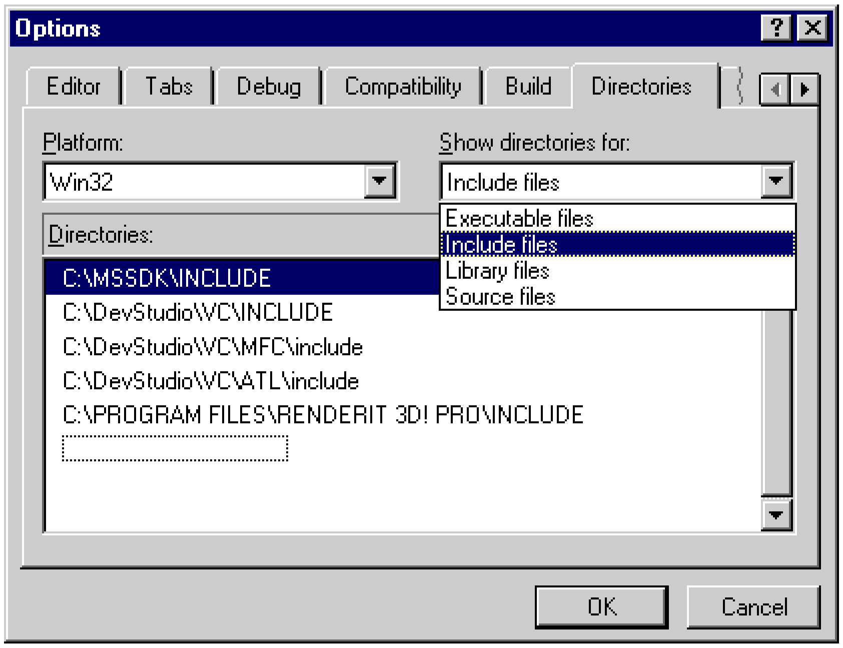
Microsoft Visual C++ 5 or later
DirectX 6.1 SDK or later
Pentium processor or later
32 MB of RAM
Windows 9x / 2000
C++ programming knowledge
Windows programming knowledge
A basic understanding of 3D programming

Setting Include Paths

In order for RenderIt 3D applications to compile and link properly, the DirectX SDK include files and the RenderIt 3D include files must be properly set within Microsoft Visual C++. If you do not have Microsoft Visual C++ 5.0 Professional, then you need to refer to the documentation provided with your development environment for information on changing these settings.

Include search paths

Make sure the search path for header files is correct, and that the directory for DirectX header files is the first path that the compiler searches. To check the include path, choose "Options" from the "Tools" menu, then select the "Directories" tab. The following dialog box will appear:



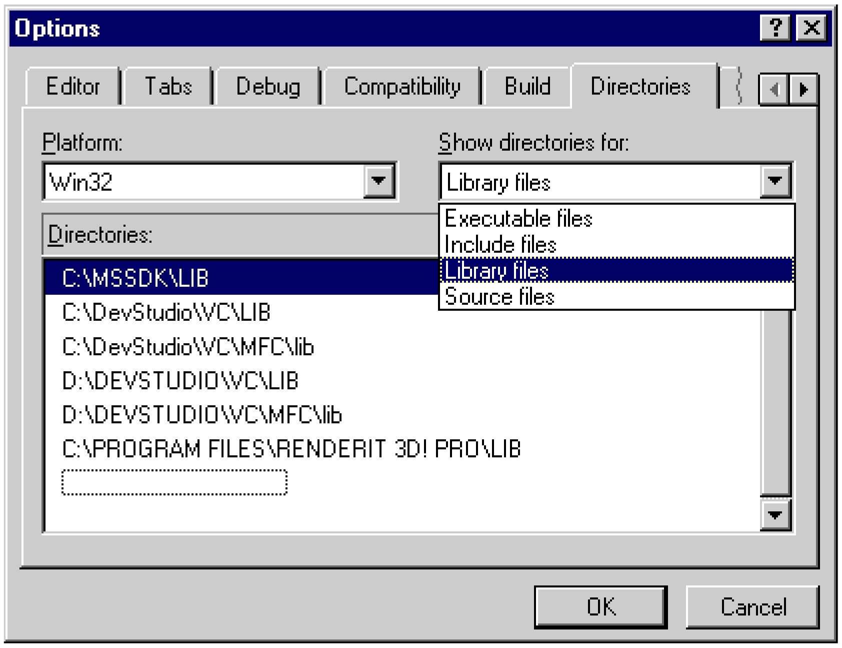
Platform: Win32
Show directories for: Include files
Directories:
C:\MSSDK\INCLUDE
C:\DevStudio\VC\INCLUDE
C:\DevStudio\VC\MFC\include
C:\DevStudio\VC\ATL\include
C:\PROGRAM FILES\RENDERIT 3D! PRO\INCLUDE

The topmost path should indicate the folder that contains the latest DirectX header files. The default path is C:\mssdk\include. If the path is not present, add it to the list and move it to the top of the search list by using the toolbar controls within the Directories tab.

The RenderIt 3D INCLUDE folder must also appear in the list as well. The search order of the RenderIt 3D INCLUDE folder is not important.

Linker search paths

Check the search paths and search order that the linker uses to search for link libraries. The link search paths are also listed on the Directories tab; choose Options from the Tools menu, then select the Directories tab. When the dialog appears, choose the "Library files" option in the "Show directories for:" drop-down list box.



Platform: Win32

Show directories for: Library files

Directories:

C:\MSSDK\LIB
C:\DevStudio\VC\LIB
C:\DevStudio\VC\MFC\lib
D:\DEVSTUDIO\VC\LIB
D:\DEVSTUDIO\VC\MFC\lib
C:\PROGRAM FILES\RENDERIT 3D! PRO\LIB

The topmost path should be the folder that contains the latest DirectX link libraries. The default path is C:\mssdk\lib.

The RenderIt 3D LIB folder must also appear in the list as well. The search order of the RenderIt 3D LIB folder is not important.

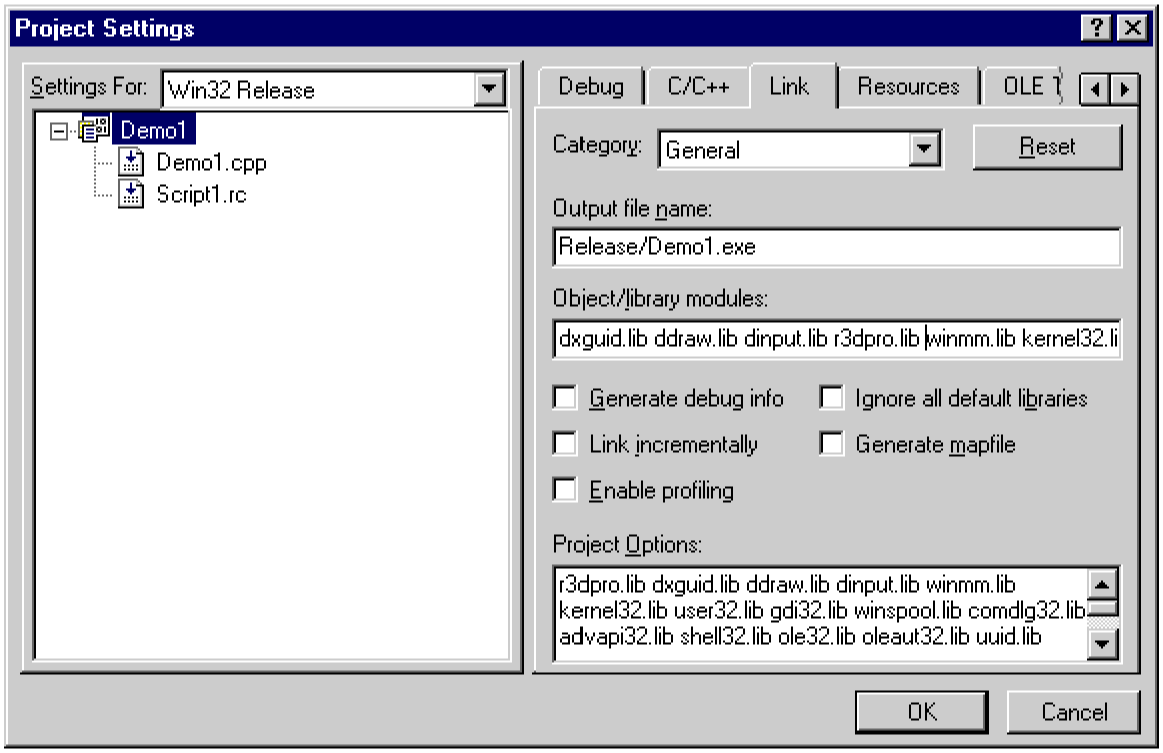
Building RenderIt 3D! Applications

RenderIt 3D applications can be built using the steps provided in this section assuming that the include paths are set correctly as outlined above. If you do not have Microsoft Visual C++ 5.0 Professional, then you need to refer to the documentation provided with your development environment for information on changing these settings.

To build a RenderIt 3D application, start Microsoft Visual C++ Developer Studio and click "File->New…". Select the "Projects" tab in the dialog box, and select "Win32 Application". Enter a name for your project in the "Project name:" edit box and click "OK".

Open the "Project Settings" dialog box. In Visual C++ 5.0 Professional, the "Project Settings" dialog box is accessed by selecting "Settings…" from the "Project" Menu.

To set up the required options for all build modes, select "All Configurations" from the "Settings For:" drop down list box.



Category: General

Output file name: Release/Demo1.exe

Object/library modules: dxguid.lib ddraw.lib dinput.lib r3dpro.lib winmm.lib kernel32.li

- [ ] Generate debug info
- [ ] Ignore all default libraries
- [ ] Link incrementally
- [ ] Generate mapfile
- [ ] Enable profiling

Project Options:

```
r3dpro.lib dxguid.lib ddraw.lib dinput.lib winmm.lib
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib
```

Now, add the libraries into your project. Click on the "Link" tab and add the following libraries in the "Object/library modules:" edit box.

ddraw.lib
dxguid.lib
r3d.lib or **r3deval.lib** (Full Version or Evaluation Version of RenderIt 3D)

Since RenderIt 3D requires a Pentium or later, your application will also require a Pentium or later. Click on the "C/C++" tab and select "Code Generation" from the "Category:" drop down list box. Now select "Pentium" from the "Processor:" drop down list box. Click "OK" to close the "Project Settings…" dialog box.

For this tutorial we will set a "Release" build mode. Select "Set Active Configuration…" from the "Build" menu and select "Win32 Release".

Now you're ready to add a source file. Select "New…" from the "File" menu and click on the "Files" tab. Make sure that the "Add to project:" check box has a check mark next to it and then click on "C++ Source File". Enter a filename for the new source file in the "File name:" edit box.

Included with RenderIt 3D is a source code template which needs to be copied to the windows clipboard using RenderIt 3D's Template program. The Template program can be run from the Programs folder (ex: "Start -> Programs -> RenderIt 3D Evaluation -> Template"). Once complete, simply paste the clipboard contents into your source file (ex: Edit -> Paste).

RenderIt 3D applications should show an "enumeration" dialog box prior to calling the r3d_Initialize function. The template file contains the source code for accessing the dialog box, but you also need the dialog box resource in your project. To copy the resource files, open Windows Explorer and copy all of the files from the RenderIt 3D Template Folder to your new project directory. The files required are SCRIPT1.RC, POWERED.BMP, and RESOURCE.H.

Once these files are copied, you need to add the resource script file to your project. From the "Project" menu, select "Add To Project" and select "Files…". Select "SCRIPT1.RC". Now select "Save Workspace" from the "File" menu and you're all set.

Now you're ready to start programming your RenderIt 3D application.

Working with RenderIt 3D!

It is recommended that your application be built from the RenderIt 3D Source Code Template as outlined in the *Getting Started* section of this manual. The template has been tested and handles some of the basic tasks involved in creating a RenderIt 3D application such as enumerating devices, setting up windows, displaying an enumeration dialog box and initializing and shutting down the RenderIt 3D system.

Of course, you are not restricted to using the source code template or the supplied dialog box. For example, you may want to add a check box to the enumeration dialog box to remember the user's favorite settings.

Enumeration, Video Modes and Windowed Modes

RenderIt 3D enumerates all 16, 24 and 32 bit video modes available on the target system. If the desktop is set to one the supported color depths, RenderIt 3D provides windowed mode rendering as well. If the desktop is set to 256 colors, your application can still run in full screen mode and all compatible video modes will be enumerated.

Direct3D devices are only enumerated if they support multi-pass alpha blending and vertex fog. Most video cards support this minimal set of features. The Hardware Emulation Layer of DirectX supports these features and so your application will work on a wide variety of systems.

Limits

The limit on the number of models, materials, textures, virtual surfaces and DirectDraw surfaces is, for all practical purposes, limited by the amount of memory available on a particular system. This data is allocated as needed by r3d_ListLoad and the r3d_SurfaceCreate and r3d_DDSurfaceCreate series of functions. By using r3d_ListUnload, r3d_SurfaceDestroyAll, and r3d_DDSurfaceDestroyAll prior to loading a new rendering environment, you can use minimal system resources. It is important that you test your application prior to distribution on a system equipped with your minimum memory requirements.

The limit on the number of attached models is 128 models per model. The limit on texture size is 1024 x 1024 unless the hardware has a smaller limit, typically 256 x 256.

Debugging RenderIt 3D! Applications

Since RenderIt 3D allows you to render in a window under 3D Hardware Emulation, you can debug your RenderIt 3D applications. If you set the build mode to Debug in Microsoft Visual C++ and you get an error related to the LIBC.LIB file, you may need to make sure that your linker contains the correct search path for your Microsoft Visual C++ CD-ROM and that the CD-ROM is loaded.

Another useful tool for debugging is Reference Rasterizer. RenderIt 3D will enumerate the Reference Rasterizer provided with the DirectX SDK provided that it is installed on your system. To install the Reference Rasterizer from the default DirectX SDK directory, enter the following in the RUN dialog from the Windows START menu:

"C:\Mssdk\Samples\Multimedia\D3dim\Bin\Enablerefrast.reg".

The Reference Rasterizer allows you to see what your application should look like on a system equipped with compatible 3D hardware accelerator regardless of the whether the system contains a 3D hardware accelerator or not. If you find that your code works properly with the Reference Rasterizer but not on your 3D Hardware Accelerator, the problem is not related to RenderIt 3D but is related to your 3D Hardware Accelerator or it's drivers or how it's drivers are installed. For more information about obtaining the proper drivers, please refer to the Indotek Driver Page on the web at:

http://www.indotek.com/driver.html

Textures

RenderIt 3D expects textures as well as "alpha" textures to be in BMP format. They need to be uncompressed and they must be 256 color or 24-bit color BMP images. If you provide a texture that is not square, a power of 2 in size, or not within the supported size limits, RenderIt 3D will scale the texture, with anti-aliasing, to an appropriate size automatically.

RenderIt 3D also has the ability to create mipmap levels for you automatically through the r3d_ListAdd3Dmodel and r3d_ListAddMaterial functions. Mipmapping is an efficient method of reducing visual artifacts in rendered textures. Mipmapping and texture filtering may be enabled or disabled at runtime with r3d_SetFilteringAndMipMapOptions.

Textures are never loaded twice. For example, if more than one material references the same texture, all subsequent textures will simply point to the previous loaded texture. This helps reduce the amount of texture swapping overhead on systems equipped with a 3D Hardware Accelerator.

RenderIt 3D allows textures to contain an alpha channel. By including an "alpha" texture, RenderIt 3D will automatically create a Direct3D alpha texture containing various levels of alpha transparency. Alpha textures must be the same size as the color texture.

When your application executes under the Hardware Emulation Layer, RenderIt 3D will use a color-keyed texture when appropriate. If you supply an alpha texture with solid white and solid black texels only, then various levels of alpha transparency are not needed and rendering performance and graphics quality will increase by using a color-keyed texture.

Alpha textures will only be loaded if a texture is also present. Alpha textures should not be grayscale BMP images but rather 256 color or 24-bit color BMP images. Alpha textures should contain colors that range from black to white. Each color is interpreted to describe the amount of texel transparency in the textures' alpha channel. A black pixel indicates a fully transparent texel and a white pixel indicates a fully opaque texel.

RenderIt 3D textures can not handle textures that are larger than 1024 x 1024. Some hardware can not handle textures that are larger than 256 x 256 in size. Because of this, RenderIt 3D automatically scales textures with anti-aliasing when required. If this is an issue, you can use the r3d_GetMinTextureWidthHeight and r3d_GetMaxTextureWidthHeight functions to selectively load textures of the desired size.

Surfaces

RenderIt 3D supports two types of surfaces, Virtual Surfaces and DirectDraw Surfaces.

Virtual surfaces do not provide hardware acceleration capabilities but are ideal for title screens, menus, or blitting with various levels of alpha transparency. RenderIt 3D also provides a means for stretching a virtual surface with anti-aliasing. Since virtual surfaces do not rely on hardware, special effects provided by virtual surfaces are guaranteed to work regardless of whether or not the hardware supports them.

DirectDraw surfaces have the potential of being accelerated by hardware but they usually only provide one level of transparency and do not perform anti-aliasing when stretching.

The type of surface you use should depend on what you need the surface for. For example, if you want to display a title screen, you would be better off using a virtual surface. But if you wanted to blit an image at every single frame, you would be better off using a DirectDraw surface.

You are not restricted to using one type of surface or the other. For example, if you were running your application in different video modes and you wanted to display the dashboard of a car at every frame and you wanted that dashboard to look really good, you could load the image onto a virtual surface, stretch the image with anti-aliasing to the appropriate screen dimensions, and then transfer the anti-aliased image onto a DirectDraw surface. This way you can blit the DirectDraw surface with fast results and it will look better then if you did a stretched blit with DirectDraw.

Back Buffer Access and 2D Rendering

RenderIt 3D now provides a way to improve the performance of 2D rendering. By placing all of your r3d_2D…() series of functions inside of an r3d_2DBegin and r3d_2DEnd function pair, you will avoiding locking and unlocking a surface each time the back buffer is accessed. For example, if your application needs to draw several lines and pixels, you only need to lock and unlock the back buffer once with r3d_2DBegin and r3d_2DEnd.

The r3d_2DGetPixelBlueMacro, r3d_2DGetPixelGreenMacro, r3d_2DGetPixelRedMacro, and r3d_2DSetPixelMacro functions must be placed inside of an r3d_2DBegin and r3d_2DEnd function pair. If the remaining r3d_2D…() series of functions are not placed inside of an r3d_2DBegin and r3d_2DEnd function pair, the back buffer will be locked and unlocked each time these functions are called.

Working with Matrices

RenderIt 3D matrices are 12 element arrays of type float. The first nine elements of the array (0 to 8) describe the rotation of the matrix and the last three elements (9 to 11) describe the translation (or offset position) of the matrix. Fortunately, you don't need to know what the first nine elements are when using RenderIt 3D.

The last three elements of a RenderIt 3D matrix describe the translation as X, Y, and Z respectively. If the matrix is a view matrix, then the last three elements describe your location of view in world space. If the matrix is a model matrix, then the last three elements describe the model location in world space.

There are three matrix manipulation functions in the RenderIt 3D library; r3d_MakeIdentityMatrix, r3d_RotateMatrixByQuaternion, and r3d_ConcatenateMatrices. When you initialize a matrix, you call r3d_MakeIdentityMatrix. To perform rotations on the matrix, you can call r3d_RotateMatrixByQuaternion. If you want to perform an X, Y, Z translation on the matrix, you simply set the last three elements of the matrix. If you want to multiply two matrices, then you can call r3d_ConcatenateMatrices.

By using r3d_RotateMatrixByQuaternion, the matrix is guaranteed to be stable since the function ensures that the matrix is properly normalized. If rotations are performed on a matrix manually, there is a very good chance that the matrix will lose precision over time resulting in distorted rendering.

The r3d_RotateMatrixByQuaternion function is a little more powerful than it may initially appear and can provide any rotation or series of rotations required by your application. You could, for example, rotate a model in its local axis system instead of world coordinates making it useful for responding to user input, or some derivative of user input. For example, let's say we don't know what the current orientation is of an aircraft and we only want to rotate the aircraft based on user input. If the user pulls back on the flight stick, the pitch of the aircraft would increase in its local system, not in the world system.

To pitch the aircraft in the world system, we would do this:

r3d_RotateMatrixByQuaternion(ModelMatrix, 1.0F, 0.0F, 0.0F, angle_to_rotate_in_radians);

However, since the current orientation of the aircraft is unknown and the pitch effect should occur in the aircraft system, we would actually want to do this:

r3d_RotateMatrixByQuaternion(ModelMatrix, ModelMatrix[0], ModelMatrix[3], ModelMatrix[6], angle_to_rotate_in_radians);

If you don't get the results you are looking for with the r3d_RotateMatrixByQuaternion function, try changing the system used for the axis of rotation as shown above.

Also, the order in which you perform rotations with r3d_RotateMatrixByQuaternion is important. For our flight simulator example, you would want to perform Yaw rotations first, which is a rotation on the Y-axis. The next rotation to perform would be the Pitch, which is on the X-axis, and finally the Roll, which is on the Z-axis.

An example of how you could use r3d_RotateMatrixByQuaternion to create a matrix from the known Euler angles of Yaw, Pitch, and Roll, is shown in the following example:

```
float fMatrix[12], YawAngle, PitchAngle, RollAngle;
// Aircraft is facing east
YawAngle = 90 * 0.0174532925199432957692369076848861F;
// Aircraft is pitch upwards by 45 degrees
PitchAngle = 45 * 0.0174532925199432957692369076848861F;
// Aircraft is rolled over to its right side
RollAngle = 90 * 0.0174532925199432957692369076848861F;
r3d_MakeIdentityMatrix( fMatrix );
r3d_RotateMatrixByQuaternion( fMatrix, 0,1,0, YawAngle );
r3d_RotateMatrixByQuaternion( fMatrix, 1,0,0, PitchAngle );
r3d_RotateMatrixByQuaternion( fMatrix, 0,0,1, RollAngle );
```

If the code above performed the rotations in a different order, the result would be different.

Working with Models

RenderIt 3D models passed to the r3d_ListAdd3DModel function are expected to be in RenderIt 3D (*.R3D) format. RenderIt 3D models are exported from Materialize 3D, which is bundled with this release. Materialize 3D can import 3D Studio *.3DS, AutoCAD *.DXF, and

Direct3D Text *.X models. Materialize 3D also allows you to modify models and add textures. For more information, see the Materialize 3D User's Guide.

Texture Animation

RenderIt 3D provides rudimentary support for texture animation which can be useful for displaying explosion and smoke sequences, displaying a video sequence on a texture surface, or anything else that your application requires.

The r3d_TextureAnimateScroll function allows an offset to be applied to the texture coordinates of a texture on a 3D model. The function is useful for dynamically scrolling a texture across a polygon or group of polygons.

The r3d_TextureAnimateSwap function is faster than r3d_TextureAnimateScroll. If you need to quickly flip a series of textures for animation purposes, r3d_TextureAnimateSwap is the right choice.

Using r3d_TextureAnimateScroll for achieving the animation of a texture containing multiple images per texture is not recommended since precision errors can occur over time. In addition, some 3D hardware has trouble coping with texel coordinates that are very large.

Instead, it is recommended that you use either r3d_TextureAnimateSwap or multiple 3D models for rendering a texture containing multiple images per texture.

The r3d_TextureAnimateSwap function can be used to change textures on a model and is very fast. For example, let's say that you have a texture that is 256 x 256 in size and it contains 16 smaller textures in a 4 x 4 grid. Using your bitmap editor, you could break the texture into 16 separate bitmaps of size 32 x 32. Then it's just a simple matter of loading the 16 images and using r3d_TextureAnimateSwap to select the appropriate texture required by your animation sequence.

As an alternative, you can use 16 models where each model has the same texture, but contains the appropriate texture coordinates for each texture in the sequence. This method is the fastest method and is much more reliable than trying to use r3d_TextureAnimateScroll.

If you need to display a video sequence on a texture or modify a texture surface once in a while, the r3d_TextureAnimateSurface series of functions are a good choice. However, the r3d_TextureAnimateSurface series of functions are the slowest. Every time you modify a texture surface with the r3d_TextureAnimateSurface series of functions, the texture needs to be reloaded into video memory as required by Direct3D. This results in decreased performance under 3D hardware acceleration. The r3d_TextureAnimateSurface series of functions will not restore a copy of an altered texture surface when the r3d_WindowSize function is called and they will not work with mipmapped textures when the bMipMap parameter is set to TRUE in the r3d_ListAdd3DModel and r3d_ListAddMaterial functions.

Rendering Custom Polygons

Although RenderIt 3D can handle 3D model rendering directly, you may want to render a single polygon or a series of polygons. With RenderIt 3D, rendering custom polygons is as simple as calling r3d_SetMatrix and r3d_SetMaterial followed by one or more DrawPrimitive calls. Polygon rendering should take place within the r3d_RenderBegin and r3d_RenderEnd function pair.

The following is small sample showing how you can implement polygon rendering.

```
//
// Somewhere in the global data area:
//
D3DVERTEX Front[4] = {
   {-1.0F,-1.0F,0.0F,0.0F,0.0F,-1.0F,0.0F,0.0F},
   {-1.0F, 1.0F,0.0F,0.0F,0.0F,-1.0F,0.0F,1.0F},
   { 1.0F, 1.0F,0.0F,0.0F,0.0F,-1.0F,1.0F,1.0F},
   { 1.0F,-1.0F,0.0F,0.0F,0.0F,-1.0F,1.0F,0.0F},
};
D3DVERTEX Back[4] = {
   { 1.0F,-1.0F,0.0F,0.0F,0.0F, 1.0F,1.0F,0.0F},
   { 1.0F, 1.0F,0.0F,0.0F,0.0F, 1.0F,1.0F,1.0F},
   {-1.0F, 1.0F,0.0F,0.0F,0.0F, 1.0F,0.0F,1.0F},
   {-1.0F,-1.0F,0.0F,0.0F,0.0F, 1.0F,0.0F,0.0F},

};
long hMaterial[1];
float MatrixView[12], MatrixPolygons[12];
//
// Somewhere in the initialization code:
//
r3d_ListAddMaterial( 1.0F, 0.0F, 0.0F, 1.0F, 32.0F, 0.5F, 0.5F, 0.5F, 0.0F, 0.0F,
0.0F, TRUE, "MyTexture.bmp", FALSE, "", FALSE);
if ( r3d_ListLoad( NULL, &hMaterial[0] ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "Error", MB_OK );
return FALSE;
}
r3d_MakeIdentityMatrix( MatrixView );
MatrixView[11] = -5.0F;
r3d_MakeIdentityMatrix( MatrixPolygons );
//
// Somewhere in your rendering loop between the
// r3d_RenderBegin and r3d_RenderEnd functions.
//
//r3d_RenderBegin( MatrixView );
r3d_SetMaterial( hMaterial[0] );
r3d_SetMatrix( MatrixPolygons );
r3d_GetDevice()->DrawPrimitive( D3DPT_TRIANGLEFAN, D3DFVF_VERTEX, &Front, 4, NULL );
r3d_GetDevice()->DrawPrimitive( D3DPT_TRIANGLEFAN, D3DFVF_VERTEX, &Back, 4, NULL );
//r3d_RenderEnd(NULL);
```

You should try to group polygons by material for optimal rendering performance. For more information on the DrawPrimitive methods, please refer to your DirectX SDK documentation.

RenderIt 3D! Programming Guidelines

This section provides some basic guidelines for programming the RenderIt 3D library.

* You should always use error checking on the RenderIt 3D functions that return success or failure information. The error will most likely be caused by a failure to follow the rules.

* RenderIt 3D will only load RenderIt 3D (*.R3D) models. In order to use r3d_ListAdd3DModel you need to convert your models to the R3D file format with Materialize 3D, which is bundled with this package.

* Don't assume that a particular video mode or windowed mode is supported on every system. RenderIt 3D allows your application to run in any desired video mode without changing any of your code with the exception of 2D drawing and blitting functions. However, by using the values of width and height obtained from r3d_GetDisplayMode, you can avoid this problem.

* If you are a beginner, use the template as the starting point for your new application and modify it as necessary. Information on building a RenderIt 3D application is provided in the Getting Started section of this manual.

* There should only be one call to the r3d_Initialize function.

* Before your application exits, a call to r3d_Finished must take place but only if r3d_Initialize has been called.

* All 3D models and materials required for a virtual environment must be loaded once with r3d_ListAdd3DModel and r3d_ListAddMaterial followed by r3d_ListLoad. If, for example, you have different levels of a game or would like to render a new virtual environment, you should call r3d_ListUnload and then proceed loading the new models and materials again.

* The following functions must always be located between an r3d_RenderBegin and r3d_RenderEnd function pair:

r3d_AlphaPolygonSort
r3d_Render3Dmodel
r3d_SetMaterial
r3d_SetMatrix
r3d_SetViewMatrix
r3d_ZBlit3DModel1
r3d_ZBlit3DModel2

* Only the following functions can be both inside and outside of an r3d_RenderBegin and r3d_RenderEnd function pair:

r3d_EnableSpecular
r3d_EnableFog
r3d_SetFilteringAndMipMapOptions
r3d_MakeIdentityMatrix
r3d_ConcatenateMatrices
r3d_RotateMatrixByQuaternion

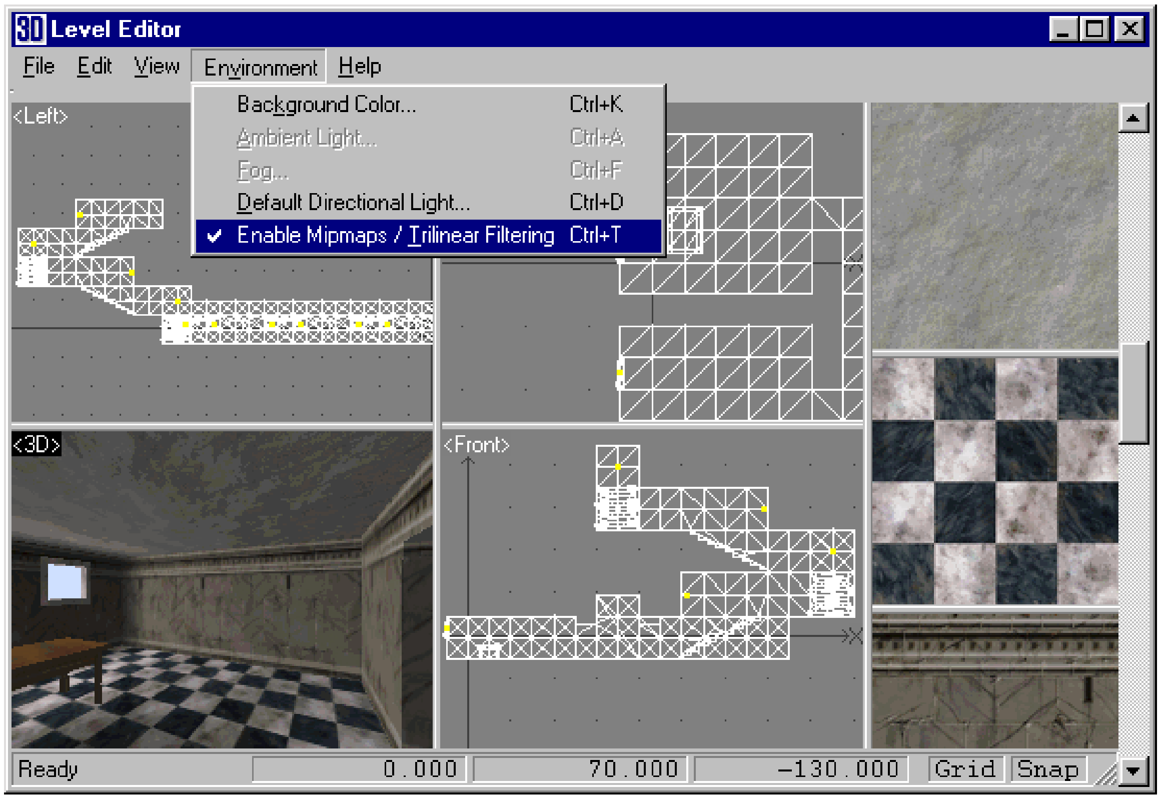
* If you want to render polygons or models, do so only between an r3d_RenderBegin and r3d_RenderEnd function pair.

* It is not recommended that you call r3d_RenderBegin and r3d_RenderEnd more than once per frame. You should restrict the number of these calls to once per frame, or once for every viewport you set by r3d_SetRenderArea.

* Never create your own materials directly through Direct3D. Use r3d_ListAddMaterial instead.

* If you experience problems with your application when you select 3D Hardware Acceleration, I'd say that there is a 99.9% chance that your drivers are not 100% compatible with DirectX or they are not installed properly. For more information, please visit the Indotek driver page on the web at http://www.indotek.com/driver.html.

The Level Editor


Level Editor
File Edit View Environment Help
Background Color... Ctrl+K
Ambient Light... Ctrl+A
Fog... Ctrl+F
Default Directional Light... Ctrl+D
✓ Enable Mipmaps / Trilinear Filtering Ctrl+T
<Left>
<3D>
<Front>
Ready
0.000
70.000
-130.000
Grid
Snap

(Actual Screen Shot)

The source code to the Level Editor is free with your purchase of RenderIt 3D. Since it is a freebie, Indotek will not provide technical support regarding the Level Editor or its source code. The Level Editor is by no means complete. But the source code can serve as a good starting point for further customization.

Anywho, with a few more modifications to the source code of the Level Editor, you could export optimized 3D data along with portal and sector data and get an extremely fast and professional quality game out of it. In fact, I see no reason why you couldn't get 60 frames-per-second with mipmapping, tri-linear filtering and music and sound enabled on a good 3D accelerator. If anyone tells you that it can't be done, I'm here to tell you that they're wrong!

Believe it or not, most of the programming involved for importing 3DS models including portals and sectors is already coded into the Level Editor. All you need is a modeler that is capable of exporting properly to 3DS format. If you have 3D Studio MAX, you're in luck. - You can model entire levels within 3D Studio MAX complete with textures, portals and sectors and simply import it into the Level Editor. Basically, all you need in this case is to write some code to export the data to your proprietary file format.

RenderIt 3D is designed at a lower level and anything you can do in Direct3D, you can do with RenderIt 3D. If you see a really cool looking game that runs under Direct3D, you can clone it with RenderIt 3D. You are not restricted to the RenderIt 3D API. You get complete control and you save a lot of time instead of trying to figure out Direct3D in the first place.

Technical Support

I provide free e-mail technical support to anyone who is using RenderIt 3D whether registered or not. Please restrict your questions solely to the RenderIt 3D library. Thank you!

Steve Eschweiler
support@indotek.com
steve@indotek.com

Function Reference

Function Reference Overview

2D Rendering Functions

r3d_2DBegin
r3d_2DBlit
r3d_2DEnd
r3d_2DFadeIn
r3d_2DGetPixelBlue
r3d_2DGetPixelBlueMacro
r3d_2DGetPixelGreen
r3d_2DGetPixelGreenMacro
r3d_2DGetPixelRed
r3d_2DGetPixelRedMacro
r3d_2DLine
r3d_2DLineMask
r3d_2DSetPixel
r3d_2DSetPixelMacro

DirectDraw Surface Functions

r3d_DDSurfaceBlit
r3d_DDSurfaceCreateFromSurface
r3d_DDSurfaceDestroy
r3d_DDSurfaceDestroyAll
r3d_DDSurfaceGet
r3d_DDSurfaceGetHeight
r3d_DDSurfaceGetWidth

Virtual Surface Functions

r3d_SurfaceBlit
r3d_SurfaceColorGetA
r3d_SurfaceColorGetB
r3d_SurfaceColorGetG
r3d_SurfaceColorGetR
r3d_SurfaceColorMakeRGBA
r3d_SurfaceCreate
r3d_SurfaceCreateFromBackBuffer
r3d_SurfaceCreateFromBMP
r3d_SurfaceDestroy
r3d_SurfaceDestroyAll
r3d_SurfaceExportToBMP
r3d_SurfaceGet
r3d_SurfaceGetHeight
r3d_SurfaceGetWidth

r3d_SurfaceSetAlphaFromBMP
r3d_SurfaceSetColorKey
r3d_SurfaceStretchSmooth

Texture Surface Functions

r3d_TextureAnimateScroll
r3d_TextureAnimateSurfaceBlit
r3d_TextureAnimateSurfaceLock
r3d_TextureAnimateSurfaceTexel0
r3d_TextureAnimateSurfaceTexel1
r3d_TextureAnimateSurfaceTexel2
r3d_TextureAnimateSurfaceTexel3
r3d_TextureAnimateSurfaceTexel4
r3d_TextureAnimateSurfaceTexel5
r3d_TextureAnimateSurfaceUnlock
r3d_TextureAnimateSwap

File Access Functions

r3d_FileClose
r3d_FileOpen
r3d_FileRead
r3d_FileSeek
r3d_FileSize
r3d_FileWrite

Timing Functions

r3d_GetCurrentTick
r3d_SetCurrentTick

Windowed Mode Functions

r3d_WindowInit
r3d_WindowMove
r3d_WindowSize

DirectX Interface Functions

r3d_GetBackBufferSurface
r3d_GetDevice
r3d_GetDeviceDesc
r3d_GetDirect3D
r3d_GetDirectDraw
r3d_GetDisplayMode
r3d_GetFrontBufferSurface
r3d_GetLight

r3d_GetViewport
r3d_GetZBufferSurface

Matrix Manipulation Functions

r3d_ConcatenateMatrices
r3d_MakeIdentityMatrix
r3d_RotateMatrixByQuaternion

Model / Material / Texture Management Functions

r3d_GetMaxTextureWidthHeight
r3d_GetMinTextureWidthHeight
r3d_ListAdd3DModel
r3d_ListAddMaterial
r3d_ListLoad
r3d_ListUnload

3D Model Manipulation Functions

r3d_Attach3DModels
r3d_Detach3DModels
r3d_Get3DModelMaterialCount
r3d_Get3DModelMaterialGeometry
r3d_Get3DModelMaterialHandle
r3d_GetBoundingBoxCoords
r3d_GetBoundingSphereRadius
r3d_PreLight3DModel

3D Rendering Functions

r3d_AlphaPolygonSort
r3d_EnableFog
r3d_EnableSpecular
r3d_Render3DModel
r3d_RenderBegin
r3d_RenderEnd
r3d_SetFilteringAndMipMapOptions
r3d_SetMaterial
r3d_SetMatrix
r3d_SetViewMatrix
r3d_ZBlit3DModel1
r3d_ZBlit3DModel2

Misc. Functions

r3d_3DVertexToScreenVertex
r3d_Enumerate

r3d_EnumerateSpecific
r3d_Finished
r3d_ForceAlphaRendering
r3d_GetLastErrorString
r3d_GetSelectionInfo
r3d_GetVersion
r3d_Initialize
r3d_LightRemoveDefault
r3d_LightSetAmbient
r3d_MessageBox
r3d_PageFlip
r3d_RestoreSurfaces
r3d_Set2DClipRegion
r3d_SetBackgroundColor
r3d_SetFogAttributes
r3d_SetLightLocationColor
r3d_SetRenderArea
r3d_ViewportClear

r3d_2DBegin

The r3d_2DBegin function locks the back buffer. It can also be used to optimize back buffer access with the r3d_2D… series of functions. r3d_2DBegin should not be placed within an r3d_RenderBegin and r3d_RenderEnd function pair.

```
BOOL r3d_2DBegin(
        unsigned char **pMem,
        int *Pitch,
        int *PixelFormat);
```

Parameters

pMem is the address of a pointer of type char variable that will be assigned the address of the back buffer surface. You may pass NULL if you do not need this information.

Pitch is a pointer to a variable that will be assigned the pitch of the back buffer surface. You may pass NULL if you do not need this information.

PixelFormat is a pointer to a variable that will be assigned the pixel format code used by RenderIt 3D for the back buffer surface. You may pass NULL if you do not need this information.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
unsigned char *pMem;
int Pitch,PixelFormat;
r3d_2DBegin( &pMem, &Pitch, &PixelFormat );
r3d_2DSetPixelMacro( pMem, Pitch, PixelFormat, 0, 0, 255, 255, 255 );
r3d_2DEnd();
```

See Also

r3d_2DEnd

r3d_2DBlit

The r3d_2DBlit function will transfer a specified region from a virtual surface to a specified region on the back buffer. Clipping and stretching will be performed if required.

```
BOOL r3d_2DBlit(
    int hSrc,
    int SrcMinx,
    int SrcMaxx,
    int SrcMiny,
    int SrcMaxy,
    int DstMinx,
    int DstMaxx,
    int DstMiny,
    int DstMaxy,
    BOOL BlendAlphaChannel);
```

Parameters

hSrc is the handle of the virtual surface.

SrcMinx,SrcMaxx,SrcMiny,SrcMaxy describes the source region of the virtual surface to be copied.

DstMinx,DstMaxx,DstMiny,DstMaxy describes the destination region on the back buffer.

BlendAlphaChannel should be set to TRUE to blend the virtual surface with the back buffer using the alpha channel information contained in the virtual surface.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, ”Bitmap.bmp“ );
r3d_2DBlit( hBitmap, 0, r3d_SurfaceGetWidth(hBitmap)-1, 0,
r3d_SurfaceGetHeight(hBitmap)-1, 0, Width-1, 0, Height-1, FALSE );
r3d_SurfaceDestroy(hBitmap);
```

See Also

r3d_2DFadeIn

r3d_2DEnd

The r3d_2DEnd function should follow a previous call to r3d_2DBegin.

BOOL r3d_2DBegin(void);

Parameters

None

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
unsigned char *pMem;
int Pitch,PixelFormat;
r3d_2DBegin( &pMem, &Pitch, &PixelFormat );
r3d_2DSetPixelMacro( pMem, Pitch, PixelFormat, 0, 0, 255, 255, 255 );
r3d_2DEnd();
```

See Also

r3d_2DBegin

r3d_2DFadeIn

The r3d_2DFadeIn function can be used to gradually fade in a virtual surface on top of the current image on the back buffer. The contents of the front buffer will be destroyed.

```
BOOL r3d_2DFadeIn(
    int hSrc,
    int SrcMinx,
    int SrcMaxx,
    int SrcMiny,
    int SrcMaxy,
    int DstMinx,
    int DstMaxx,
    int DstMiny,
    int DstMaxy,
    int StepSize);
```

Parameters

hSrc is the handle of the virtual surface.

SrcMinx,SrcMaxx,SrcMiny,SrcMaxy describes the source region of the virtual surface to be fade in.

DstMinx,DstMaxx,DstMiny,DstMaxy describes the destination region on the back buffer.

StepSize determines the speed of the fade and should be between 1 and 255. A value of 255 produces the fastest fade in effect. A value of 1 produces the slowest possible fade in effect.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, ”Bitmap.bmp“ );
r3d_2DFadeIn( hBitmap, 0, r3d_SurfaceGetWidth(hBitmap)-1, 0,
r3d_SurfaceGetHeight(hBitmap)-1, 0, Width-1, 0, Height-1, 15 );
r3d_SurfaceDestroy(hBitmap);
```

See Also

r3d_2DBlit

r3d_2DGetPixelBlue

The r3d_2DGetPixelBlue function will obtain the blue color component of a specified pixel from the back buffer.

```
unsigned char r3d_2DGetPixelBlue(
        int x,
        int y);
```

Parameters

x,y is the screen coordinate of the pixel.

Return Value

Returns the blue color component of the desired pixel on the back buffer in the range of 0-255. If x,y is outside the range of the back buffer surface, the return value is undefined.

Source Code Sample

```
unsigned char red,green,blue;
r3d_2DBegin( NULL, NULL, NULL );
red = r3d_2DGetPixelRed( 0, 0 );
green = r3d_2DGetPixelGreen( 0, 0 );
blue = r3d_2DGetPixelBlue( 0, 0 );
r3d_2DEnd();
```

See Also

r3d_2DGetPixelRed, r3d_2DGetPixelGreen, r3d_2DGetPixelRedMacro, r3d_2DGetPixelGreenMacro, r3d_2DGetPixelBlueMacro

r3d_2DGetPixelBlueMacro

The r3d_2DGetPixelBlueMacro will obtain the blue color component of a specified pixel from the back buffer.

```
unsigned char r3d_2DGetPixelBlue(
        unsigned char *pMem,
        int Pitch,
        int PixelFormat,
        int x,
        int y);
```

Parameters

pMem, Pitch, PixelFormat is obtained from a previous call to r3d_2DBegin.

x,y is the screen coordinate of the pixel.

Return Value

Returns the blue color component of the desired pixel on the back buffer in the range of 0-255. If x,y is outside the range of the back buffer surface, the return value is undefined.

Source Code Sample

```
unsigned char *pMem,red,green,blue;
int Pitch,PixelFormat;
r3d_2DBegin( &pMem, &Pitch, &PixelFormat );
red = r3d_2DGetPixelRedMacro( pMem, Pitch, PixelFormat, 0, 0 );
green = r3d_2DGetPixelGreenMacro( pMem, Pitch, PixelFormat, 0, 0 );
blue = r3d_2DGetPixelBlueMacro( pMem, Pitch, PixelFormat, 0, 0 );
r3d_2DEnd();
```

See Also

r3d_2DGetPixelRed, r3d_2DGetPixelGreen, r3d_2DGetPixelBlue, r3d_2DGetPixelRedMacro, r3d_2DGetPixelGreenMacro

r3d_2DGetPixelGreen

The r3d_2DGetPixelGreen function will obtain the green color component of a specified pixel from the back buffer.

```
unsigned char r3d_2DGetPixelBlue(
        int x,
        int y);
```

Parameters

x,y is the screen coordinate of the pixel.

Return Value

Returns the green color component of the desired pixel on the back buffer in the range of 0-255. If x,y is outside the range of the back buffer surface, the return value is undefined.

Source Code Sample

```
unsigned char red,green,blue;
r3d_2DBegin( NULL, NULL, NULL );
red = r3d_2DGetPixelRed( 0, 0 );
green = r3d_2DGetPixelGreen( 0, 0 );
blue = r3d_2DGetPixelBlue( 0, 0 );
r3d_2DEnd();
```

See Also

r3d_2DGetPixelRed, r3d_2DGetPixelBlue, r3d_2DGetPixelRedMacro, r3d_2DGetPixelGreenMacro, r3d_2DGetPixelBlueMacro

r3d_2DGetPixelGreenMacro

The r3d_2DGetPixelGreenMacro will obtain the green color component of a specified pixel from the back buffer.

```
unsigned char r3d_2DGetPixelGreen(
        unsigned char *pMem,
        int Pitch,
        int PixelFormat,
        int x,
        int y);
```

Parameters

pMem, Pitch, PixelFormat is obtained from a previous call to r3d_2DBegin.

x,y is the screen coordinate of the pixel.

Return Value

Returns the green color component of the desired pixel on the back buffer in the range of 0-255. If x,y is outside the range of the back buffer surface, the return value is undefined.

Source Code Sample

```
unsigned char *pMem,red,green,blue;
int Pitch,PixelFormat;
r3d_2DBegin( &pMem, &Pitch, &PixelFormat );
red = r3d_2DGetPixelRedMacro( pMem, Pitch, PixelFormat, 0, 0 );
green = r3d_2DGetPixelGreenMacro( pMem, Pitch, PixelFormat, 0, 0 );
blue = r3d_2DGetPixelBlueMacro( pMem, Pitch, PixelFormat, 0, 0 );
r3d_2DEnd();
```

See Also

r3d_2DGetPixelRed, r3d_2DGetPixelGreen, r3d_2DGetPixelBlue, r3d_2DGetPixelRedMacro, r3d_2DGetPixelBlueMacro

r3d_2DGetPixelRed

The r3d_2DGetPixelRed function will obtain the red color component of a specified pixel from the back buffer.

```
unsigned char r3d_2DGetPixelBlue(
        int x,
        int y);
```

Parameters

x,y is the screen coordinate of the pixel.

Return Value

Returns the red color component of the desired pixel on the back buffer in the range of 0-255. If x,y is outside the range of the back buffer surface, the return value is undefined.

Source Code Sample

```
unsigned char red,green,blue;
r3d_2DBegin( NULL, NULL, NULL );
red = r3d_2DGetPixelRed( 0, 0 );
green = r3d_2DGetPixelGreen( 0, 0 );
blue = r3d_2DGetPixelBlue( 0, 0 );
r3d_2DEnd();
```

See Also

r3d_2DGetPixelGreen, r3d_2DGetPixelBlue, r3d_2DGetPixelRedMacro, r3d_2DGetPixelGreenMacro, r3d_2DGetPixelBlueMacro

r3d_2DGetPixelRedMacro

The r3d_2DGetPixelRedMacro will obtain the red color component of a specified pixel from the back buffer.

```
unsigned char r3d_2DGetPixelGreen(
    unsigned char *pMem,
    int Pitch,
    int PixelFormat,
    int x,
    int y);
```

Parameters

pMem, Pitch, PixelFormat is obtained from a previous call to r3d_2DBegin.

x,y is the screen coordinate of the pixel.

Return Value

Returns the red color component of the desired pixel on the back buffer in the range of 0-255. If x,y is outside the range of the back buffer surface, the return value is undefined.

Source Code Sample

```
unsigned char *pMem,red,green,blue;
int Pitch,PixelFormat;
r3d_2DBegin( &pMem, &Pitch, &PixelFormat );
red = r3d_2DGetPixelRedMacro( pMem, Pitch, PixelFormat, 0, 0 );
green = r3d_2DGetPixelGreenMacro( pMem, Pitch, PixelFormat, 0, 0 );
blue = r3d_2DGetPixelBlueMacro( pMem, Pitch, PixelFormat, 0, 0 );
r3d_2DEnd();
```

See Also

r3d_2DGetPixelRed, r3d_2DGetPixelGreen, r3d_2DGetPixelBlue, r3d_2DGetPixelGreenMacro, r3d_2DGetPixelBlueMacro

r3d_2DLine

The r3d_2DLine function will draw a line between two specified points on the back buffer. Clipping will be performed if necessary.

```
BOOL r3d_2DLine(
    int x01,
    int y01,
    int x02,
    int y02,
    unsigned char r,
    unsigned char g,
    unsigned char b);
```

Parameters

x01,y01,x02,y02 should describe the two endpoints of the line in screen coordinates.

r,g,b should contain the red, green, and blue color components of the line in the range of 0-255.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_2DBegin( NULL, NULL, NULL );
r3d_2DLine( 0, 0, 100, 100, 255, 255, 255 );
r3d_2DEnd();
```

See Also

r3d_2DLineMask

r3d_2DLineMask

The r3d_2DLine function will draw a line between two specified points on the back buffer. Clipping will be performed if necessary.

```
BOOL r3d_2DLineMask(
    int x01,
    int y01,
    int x02,
    int y02,
    unsigned char r,
    unsigned char g,
    unsigned char b,
    int *Mask,
    int NumMaskElements);
```

Parameters

x01,y01,x02,y02 should describe the two endpoints of the line in screen coordinates.

r,g,b should contain the red, green, and blue color components of the line in the range of 0-255.

Mask should be the address of an array. If an element in the array is non-zero, the resulting pixel will be solid. If an element in the array is zero, the resulting pixel will not be rendered.

NumMaskElements should contain the number of elements in the mask array.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int Mask[8] = { 1,1,1,1,0,0,0,0 };
r3d_2DBegin( NULL, NULL, NULL );
r3d_2DLineMask( 0, 0, 100, 100, 255, 255, 255, &Mask[0], 8 );
r3d_2DEnd();
```

See Also

r3d_2DLine

r3d_2DSetPixel

The r3d_2DSetPixel function will set a pixel on the back buffer to a specified color. Clipping and alpha blending will be performed if necessary.

```
BOOL r3d_2DSetPixel(
        int x,
        int y,
        unsigned char r,
        unsigned char g,
        unsigned char b,
        unsigned char a);
```

Parameters

x,y is the screen coordinate of the pixel.

r,g,b,a should contain the red, green, blue and alpha color components of the pixel in the range of 0-255. If alpha is set to 255 the pixel will not be blended.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_2DBegin( NULL, NULL, NULL );
r3d_2DSetPixel( 0, 0, 255, 255, 255, 128 );
r3d_2DEnd();
```

See Also

r3d_2DSetPixelMacro

r3d_2DSetPixelMacro

The r3d_2DSetPixelMacro will set a pixel on the back buffer to a specified color. Clipping will be performed if necessary.

```
void r3d_2DSetPixelMacro(
    unsigned char *pMem,
    int Pitch,
    int PixelFormat,
    int x,
    int y,
    unsigned char r,
    unsigned char g,
    unsigned char b);
```

Parameters

pMem, Pitch, PixelFormat is obtained from a previous call to r3d_2DBegin.

x,y is the screen coordinate of the pixel.

r,g,b should contain the red, green, and blue color components of the pixel in the range of 0-255.

Return Value

None.

Source Code Sample

```
unsigned char *pMem;
int Pitch,PixelFormat;
r3d_2DBegin( &pMem, &Pitch, &PixelFormat );
r3d_2DSetPixelMacro( pMem, Pitch, PixelFormat, 0, 0, 255, 255, 255 );
r3d_2DEnd();
```

See Also

r3d_2DSetPixel

r3d_3DVertexToScreenVertex

The r3d_3DVertexToScreenVertex function transforms a 3D vertex into screen coordinates.

```
BOOL r3d_3DVertexToScreenVertex(
    float x,
    float y,
    float z,
    float *pf4x3VertexMatrix,
    int *sx,
    int *sy,
    BOOL *bOffScreen);
```

Parameters

x,y,z are the 3D coordinates of the vertex.

pf4x3VertexMatrix is the address of the matrix array describing the 3D world coordinates of the 3D vertex.

sx,sy will be set to the screen coordinates of the transformed vertex unless the bOffScreen parameter is set to TRUE.

bOffScreen will be set to TRUE if the transformation places the 2D vertex outside the screen area.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
float fMatrix[12];
int sx,sy;
BOOL bOffScreen;
r3d_MakeIdentityMatrix(fMatrix);
r3d_3DVertexToScreenVertex( 0, 0, 10, &fMatrix[0], &sx, &sy, &bOffScreen )
if ( bOffScreen == FALSE )
r3d_SetPixel( sx, sy, 255, 255, 255, 255 );
```

r3d_AlphaPolygonSort

The r3d_AlphaPolygonSort function should only be called prior r3d_RenderEnd and only if you wish to perform custom sorting of alpha polygons. If you don't call this function prior to r3d_RenderEnd, RenderIt 3D will sort any alpha polygons stored in it's internal buffer automatically in a back to front order using the centroid of the alpha polygons as a reference. If you call r3d_AlphaPolygonSort but do not sort the alpha polygons, alpha polygons will be rendered without sorting.

void r3d_AlphaPolygonSort(
    int **TriangleCount,
    D3DVERTEX **VertexArray,
    int **TriangleSortArray);

Parameters

TriangleCount is a pointer to an integer which points to the number of triangles that are currently in the TriangleSortArray parameter. The number of vertices will be this value times three.

VertexArray is a pointer to a D3DVERTEX array containing information about the vertices of the alpha polygons. The first three vertices describe the first triangle, the next three vertices describe the second triangle, and so on.

TriangleSortArray is a pointer to an array of integers that contains references to the triangle sort order.

Return Value

None.

Source Code Sample

```
// Simple bubble sort for a front to back drawing order
// Sorts triangles by their center
int LoopVar,LoopVar2,Temp;
float x01,y01,z01,x02,y02,z02,DistanceSquared01,DistanceSquared02;
int *TriangleCount;
D3DVERTEX *VertexArray;
int *TriangleSortArray;
r3d_AlphaPolygonSort( &TriangleCount, &VertexArray, &TriangleSortArray);
for (LoopVar=0; LoopVar<(*TriangleCount)-1; LoopVar++)
{
for (LoopVar2=LoopVar+1; LoopVar2<(*TriangleCount); LoopVar2++)
{
// Calculate distance squared to centroid of
// two polygons to compare.
x01 = (( VertexArray[(TriangleSortArray[LoopVar]*3)+0].x +
VertexArray[(TriangleSortArray[LoopVar]*3)+1].x +
VertexArray[(TriangleSortArray[LoopVar]*3)+2].x ) * 0.33333333333333333333F);
y01 = (( VertexArray[(TriangleSortArray[LoopVar]*3)+0].y +
VertexArray[(TriangleSortArray[LoopVar]*3)+1].y +
VertexArray[(TriangleSortArray[LoopVar]*3)+2].y ) * 0.33333333333333333333F);
z01 = (( VertexArray[(TriangleSortArray[LoopVar]*3)+0].z +
VertexArray[(TriangleSortArray[LoopVar]*3)+1].z +
VertexArray[(TriangleSortArray[LoopVar]*3)+2].z ) * 0.3333333333333333333333F);
DistanceSquared01 = (x01*x01) + (y01*y01) + (z01*z01);
x02 = (( VertexArray[(TriangleSortArray[LoopVar2]*3)+0].x +
VertexArray[(TriangleSortArray[LoopVar2]*3)+1].x +
VertexArray[(TriangleSortArray[LoopVar2]*3)+2].x ) * 0.3333333333333333333333F);
y02 = (( VertexArray[(TriangleSortArray[LoopVar2]*3)+0].y +
VertexArray[(TriangleSortArray[LoopVar2]*3)+1].y +
VertexArray[(TriangleSortArray[LoopVar2]*3)+2].y ) * 0.3333333333333333333333F);
z02 = (( VertexArray[(TriangleSortArray[LoopVar2]*3)+0].z +
VertexArray[(TriangleSortArray[LoopVar2]*3)+1].z +
VertexArray[(TriangleSortArray[LoopVar2]*3)+2].z ) * 0.3333333333333333333333F);
DistanceSquared02=(x02*x02)+(y02*y02)+(z02*z02);
// Swap triangle references based on relative
// distance to centroid, if needed.
if (DistanceSquared01 < DistanceSquared02)
{
Temp=TriangleSortArray[LoopVar];
TriangleSortArray[LoopVar]=TriangleSortArray[LoopVar2];
TriangleSortArray[LoopVar2]=Temp;
}
}
}
// This is where you call r3d_RenderEnd() which renders the alpha polygons.
```

See Also

r3d_ForceAlphaRendering

r3d_Attach3DModels

The r3d_Attach3DModels function attaches two models together. Attached models may be linked together in a chain and transformed independently as parts. The maximum limit of attached models is 128 models per model.

```
BOOL r3d_Attach3DModels(
        long lDstModelHandle,
        long lSrcModelHandle,
        float *pf4x3SrcModelMatrix);
```

Parameters

lDstModelHandle is a valid handle returned by a previous call to r3d_ListLoad and describes the destination model (the model that will contain the attached model).

lSrcModelHandle is a valid handle returned by a previous call to r3d_ListLoad and describes the source model (the model to attach).

pf4x3SrcModelMatrix is the address of a 12 element array describing the matrix of the model to attach.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Let's load 2 models - a car model without wheels and a single wheel model
r3d_ListAdd3DModel("Car.r3d",FALSE)
r3d_ListAdd3DModel("Wheel.r3d",FALSE)
r3d_ListLoad(hModel,NULL)
// Declare the matrices in the global data area!
float CarMatrix[12];
float FrontLeftWheelMatrix[12];
float FrontRightWheelMatrix[12];
float BackLeftWheelMatrix[12];
float BackRightWheelMatrix[12];
// Create the matrices
r3d_MakeIdentityMatrix(CarMatrix);
r3d_MakeIdentityMatrix(FrontLeftWheelMatrix);
r3d_MakeIdentityMatrix(FrontRightWheelMatrix);
r3d_MakeIdentityMatrix(BackLeftWheelMatrix);
r3d_MakeIdentityMatrix(BackRightWheelMatrix);
// Now we set the transformation of the four wheels in relation to the car
// We assume the wheel model's rotation coincides to the right wheels
FrontRightWheelMatrix[9]=2.0F;
FrontRightWheelMatrix[10]=1.0F;
FrontRightWheelMatrix[11]=3.0F;
BackRightWheelMatrix[9]=2.0F;
BackRightWheelMatrix[10]=1.0F;
```

```
BackRightWheelMatrix[11]=-3.0F;
FrontLeftWheelMatrix[9]=-2.0F;
FrontLeftWheelMatrix[10]=1.0F;
FrontLeftWheelMatrix[11]=3.0F;
BackLeftWheelMatrix[9]=-2.0F;
BackLeftWheelMatrix[10]=1.0F;
BackLeftWheelMatrix[11]=-3.0F;
r3d_RotateMatrixByQuaternion( FrontLeftWheelMatrix, 0,1,0, 180.0F*0.01745329251994F
);
r3d_RotateMatrixByQuaternion( BackLeftWheelMatrix, 0,1,0, 180.0F*0.01745329251994F );
// Now we attach all four wheels to the car
r3d_Attach3DModels( hModel[CAR], hModel[WHEEL], FrontRightWheelMatrix );
r3d_Attach3DModels( hModel[CAR], hModel[WHEEL], BackRightWheelMatrix );
r3d_Attach3DModels( hModel[CAR], hModel[WHEEL], FrontLeftWheelMatrix );
r3d_Attach3DModels( hModel[CAR], hModel[WHEEL], BackLeftWheelMatrix );
// Now that we have loaded all models and set up the car model with the wheels
attached, we only need to render the car with a single function call to
r3d_Render3DModel() when it's time to render:
// r3d_RenderBegin(. . .);
// . . .
// r3d_Render3DModel( hModel[CAR], CarMatrix );
// . . .
// r3d_RenderEnd(. . .);
// It is important that you understand that any transformations performed on the
wheels (by their matrix) will now be local to the car's transformation. To turn the
front wheels as if steering the car, you would simply do the following:
// r3d_RotateMatrixByQuaternion(FrontLeftWheelMatrix, 0,1,0, angle_in_radians );
// r3d_RotateMatrixByQuaternion(FrontRightWheelMatrix, 0,1,0, angle_in_radians );
// The next time you render the car, the wheels will appear as if they were turned.
// It is also important to remember that the wheel matrices should not be declared
locally. In other words the wheel matrices will need to be valid every time
r3d_Render3DModel(hModel[CAR]); is called. Of course, the best way to avoid any
problems is to declare the matrices in the global data area.
```

See Also

r3d_Detach3DModels

r3d_ConcatenateMatrices

The r3d_ConcatenateMatrices function will multiply two RenderIt 3D matrices together. The first matrix will be multiplied by the second matrix and placed into the destination matrix.

```
void r3d_ConcatenateMatrices(
        float *pf4x3DstMatrix,
        float *pf4x3Matrix1,
        float *pf4x3Matrix2);
```

Parameters

pf4x3DstMatrix is the address of a 12 element array that will be filled with the resulting matrix of pf4x3Matrix1 * pf4x3Matrix2.

pf4x3Matrix1 is the address of a 12 element array that describes the first matrix.

pf4x3Matrix2 is the address of a 12 element array that describes the second matrix.

Return Value

None.

Source Code Sample

```
float MatrixView[12], MatrixAirplane[12], MatrixPilotHead[12];
r3d_MakeIdentityMatrix( MatrixAirplane );
r3d_MakeIdentityMatrix( MatrixPilotHead );
// Pilot is looking left out of the airplane window
r3d_RotateMatrixByQuaternion( MatrixPilotHead, 0,1,0, -90.0F*0.01745329251994F );
// Aircraft is banked at 45 degrees
r3d_RotateMatrixByQuaternion( MatrixAirplane, 0,0,1, 45.0F*0.01745329251994F );
// Create the appropriate view matrix based on the orientation of the pilots head and
the orientation of the aircraft.
r3d_ConcatenateMatrices( MatrixView, MatrixAirplane, MatrixPilotHead );
r3d_RenderBegin( MatrixView );
// . . .
```

See Also

r3d_MakeIdentityMatrix, r3d_RotateMatrixByQuaternion

r3d_DDSurfaceBlit

The r3d_DDSurfaceBlit will transfer any DirectDraw surface to any other DirectDraw surface including front and back buffers. Clipping and stretching will be performed if necessary. The r3d_DDSurfaceBlit function will use DirectDraw's BltFast method automatically if possible, otherwise it will use DirectDraw's Blt method.

```
BOOL r3d_DDSurfaceBlit(
    LPDIRECTDRAWSURFACE4 Src,
    LPDIRECTDRAWSURFACE4 Dst,
    int SrcWidth,
    int SrcHeight,
    int DstWidth,
    int DstHeight,
    int SrcMinx,
    int SrcMaxx,
    int SrcMiny,
    int SrcMaxy,
    int DstMinx,
    int DstMaxx,
    int DstMiny,
    int DstMaxy,
    BOOL bSrcColorKey,
    BOOL bDstColorKey);
```

Parameters

Src,Dst are pointers to the source and destination DirectDraw surfaces respectively.

SrcWidth,SrcHeight,DstWidth,DstHeight are the actual dimensions of the source and destination surfaces.

SrcMinx,SrcMaxx,SrcMiny,SrcMaxy describes the region of the source surface to transfer from.

DstMinx,DstMaxx,DstMiny,DstMaxy describes the region of the destination surface to transfer to.

bSrcColorKey should be set to TRUE if you have used DirectDraw's SetColorKey method to set a source color key on the source surface and you wish to perform a color keyed source blit.

bDstColorKey should be set to TRUE if you have used DirectDraw's SetColorKey method to set a destination color key on the source surface and you wish to perform a color keyed destination blit.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,hDDSurface,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
// Attempt to create the DirectDraw surface in video memory. If that fails we simply
create the surface in system memory.
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,1))
r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0);
r3d_SurfaceDestroy(hBitmap);
// Blit to back buffer
r3d_DDSurfaceBlit( r3d_DDSurfaceGet(hDDSurface), r3d_GetBackBufferSurface(),
r3d_DDSurfaceGetWidth(hDDSurface), r3d_DDSurfaceGetHeight(hDDSurface), Width, Height,
0, r3d_DDSurfaceGetWidth(hDDSurface)-1, 0, r3d_DDSurfaceGetHeight(hDDSurface)-1, 0,
Width-1, 0, Height-1, FALSE, FALSE);
r3d_DDSurfaceDestroy(hDDSurface);
```

See Also

r3d_2DBlit

r3d_DDSurfaceCreateFromSurface

The r3d_DDSurfaceCreateFromSurface function will create a DirectDraw surface from a previously created virtual surface. The dimensions will be set to the dimensions of the virtual surface. Once created, the virtual surface image will be copied to the DirectDraw surface.

```
BOOL r3d_DDSurfaceCreateFromSurface(
    int *handle,
    int hSrc,
    int MemoryType);
```

Parameters

handle is a pointer to a variable that will be assigned a surface handle.

hSrc is the handle of a previously created virtual surface.

MemoryType specifies where you want the DirectDraw surface created. A value of 1 indicates that you want to attempt to create the surface in video memory. A value of 0 indicates that you want to create the surface in system memory.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// If you are running in 3D Hardware Acceleration with RenderIt 3D, you will get
optimal results by setting the last parameter of r3d_DDSurfaceCreateFromSurface to 1,
however the function may fail to create the surface due to video memory constraints.
Always check the return value from r3d_DDSurfaceCreateFromSurface.
int hBitmap,hDDSurface,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
// Attempt to create the DirectDraw surface in video memory. If that fails we simply
create the surface in system memory.
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,1))
{
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0))
{
r3d_MessageBox( r3d_GetLastErrorString(), "Error", MB_OK );
}
}
r3d_SurfaceDestroy(hBitmap);
```

r3d_DDSurfaceCreateFromSurface

The r3d_DDSurfaceCreateFromSurface function will create a DirectDraw surface from a previously created virtual surface. The dimensions will be set to the dimensions of the virtual surface. Once created, the virtual surface image will be copied to the DirectDraw surface.

BOOL r3d_DDSurfaceCreateFromSurface(
int **handle*,
int *hSrc*,
int *MemoryType*);

Parameters

handle is a pointer to a variable that will be assigned a surface handle.

hSrc is the handle of a previously created virtual surface.

MemoryType specifies where you want the DirectDraw surface created. A value of 1 indicates that you want to attempt to create the surface in video memory. A value of 0 indicates that you want to create the surface in system memory.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// If you are running in 3D Hardware Acceleration with RenderIt 3D, you will get
optimal results by setting the last parameter of r3d_DDSurfaceCreateFromSurface to 1,
however the function may fail to create the surface due to video memory constraints.
Always check the return value from r3d_DDSurfaceCreateFromSurface.
int hBitmap,hDDSurface,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
// Attempt to create the DirectDraw surface in video memory. If that fails we simply
create the surface in system memory.
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,1))
{
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0))
r3d_MessageBox( r3d_GetLastErrorString(), "Error", MB_OK );
}
r3d_SurfaceDestroy(hBitmap);
// Blit to back buffer
r3d_DDSurfaceBlit( r3d_DDSurfaceGet(hDDSurface), r3d_GetBackBufferSurface(),
r3d_DDSurfaceGetWidth(hDDSurface), r3d_DDSurfaceGetHeight(hDDSurface), Width, Height,
0, r3d_DDSurfaceGetWidth(hDDSurface)-1, 0, r3d_DDSurfaceGetHeight(hDDSurface)-1, 0,
Width-1, 0, Height-1, FALSE, FALSE);
r3d_DDSurfaceDestroy(hDDSurface);
```

See Also

r3d_DDSurfaceDestroy, r3d_SurfaceCreateFromBMP

r3d_DDSurfaceDestroy

The r3d_DDSurfaceDestroy function will free the memory of a surface previously allocated with a call to r3d_DDSurfaceCreateFromSurface.

```
BOOL r3d_DDSurfaceDestroy(
        int handle);
```

Parameters

handle is the handle of the DirectDraw surface you wish to release.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,hDDSurface;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0);
r3d_SurfaceDestroy(hBitmap);
r3d_DDSurfaceDestroy(hDDSurface);
```

See Also

r3d_DDSurfaceDestroyAll, r3d_DDSurfaceCreateFromSurface

r3d_DDSurfaceDestroyAll

The r3d_DDSurfaceDestroyAll function will free the memory of all surfaces previously allocated with calls to r3d_DDSurfaceCreateFromSurface.

BOOL r3d_DDSurfaceDestroyAll(void);

Parameters

None.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,hDDSurface;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0);
r3d_SurfaceDestroy(hBitmap);
r3d_DDSurfaceDestroyAll();
```

See Also

r3d_DDSurfaceDestroy, r3d_DDSurfaceCreateFromSurface

r3d_DDSurfaceGet

The r3d_DDSurfaceGet function will return a pointer to the DirectDraw surface memory previously created with r3d_DDSurfaceCreateFromSurface.

```
LPDIRECTDRAWSURFACE4 r3d_DDSurfaceGet(
    int hDDSurface);
```

Parameters

hDDSurface is the handle of the DirectDraw surface you wish to obtain.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,hDDSurface,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
// Attempt to create the DirectDraw surface in video memory. If that fails we simply
create the surface in system memory.
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,1))
r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0);
r3d_SurfaceDestroy(hBitmap);
// Blit to back buffer
r3d_DDSurfaceBlit( r3d_DDSurfaceGet(hDDSurface), r3d_GetBackBufferSurface(),
r3d_DDSurfaceGetWidth(hDDSurface), r3d_DDSurfaceGetHeight(hDDSurface), Width, Height,
0, r3d_DDSurfaceGetWidth(hDDSurface)-1, 0, r3d_DDSurfaceGetHeight(hDDSurface)-1, 0,
Width-1, 0, Height-1, FALSE, FALSE);
r3d_DDSurfaceDestroy(hDDSurface);
```

See Also

r3d_DDSurfaceGetWidth, r3d_DDSurfaceGetHeight, r3d_DDSurfaceCreateFromSurface

r3d_DDSurfaceGetHeight

The r3d_DDSurfaceGetHeight function will return the height of a DirectDraw surface previously created with r3d_DDSurfaceCreateFromSurface.

```
int r3d_DDSurfaceGetHeight(
      int handle);
```

Parameters

handle is the handle of the DirectDraw surface you wish to obtain the information from.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,hDDSurface,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
// Attempt to create the DirectDraw surface in video memory. If that fails we simply
create the surface in system memory.
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,1))
r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0);
r3d_SurfaceDestroy(hBitmap);
// Blit to back buffer
r3d_DDSurfaceBlit( r3d_DDSurfaceGet(hDDSurface), r3d_GetBackBufferSurface(),
r3d_DDSurfaceGetWidth(hDDSurface), r3d_DDSurfaceGetHeight(hDDSurface), Width, Height,
0, r3d_DDSurfaceGetWidth(hDDSurface)-1, 0, r3d_DDSurfaceGetHeight(hDDSurface)-1, 0,
Width-1, 0, Height-1, FALSE, FALSE);
r3d_DDSurfaceDestroy(hDDSurface);
```

See Also

r3d_DDSurfaceGetWidth, r3d_DDSurfaceCreateFromSurface

r3d_DDSurfaceGetWidth

The r3d_DDSurfaceGetWidth function will return the width of a DirectDraw surface previously created with r3d_DDSurfaceCreateFromSurface.

```
int r3d_DDSurfaceGetWidth(
        int handle);
```

Parameters

handle is the handle of the DirectDraw surface you wish to obtain the information from.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap,hDDSurface,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
// Attempt to create the DirectDraw surface in video memory. If that fails we simply
create the surface in system memory.
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,1))
r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0);
r3d_SurfaceDestroy(hBitmap);
// Blit to back buffer
r3d_DDSurfaceBlit( r3d_DDSurfaceGet(hDDSurface), r3d_GetBackBufferSurface(),
r3d_DDSurfaceGetWidth(hDDSurface), r3d_DDSurfaceGetHeight(hDDSurface), Width, Height,
0, r3d_DDSurfaceGetWidth(hDDSurface)-1, 0, r3d_DDSurfaceGetHeight(hDDSurface)-1, 0,
Width-1, 0, Height-1, FALSE, FALSE);
r3d_DDSurfaceDestroy(hDDSurface);
```

See Also

r3d_DDSurfaceGetHeight, r3d_DDSurfaceCreateFromSurface

r3d_Detach3DModels

The r3d_Detach3DModels function allows you to detach any models that were attached with the r3d_Attach3DModels function.

```
BOOL r3d_Detach3DModels(
        long lDstModelHandle,
        long lSrcModelHandle,
        float *pf4x3SrcModelMatrix);
```

Parameters

lDstModelHandle,lSrcModelHandle,pf4x3SrcModelMatrix should be identical to the parameters that you passed to r3d_Attach3DModels.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
#define BODY 0
#define ARM 1
long hModel[2];
r3d_ListAdd3DModel( "Body.r3d", FALSE);
r3d_ListAdd3DModel( "Arm.r3d", FALSE );
r3d_ListLoad( hModel, NULL );
// Declare the matrices in the global data area!
float MatrixBody[12], MatrixRightArm[12], MatrixLeftArm[12];
// Create the matrices
r3d_MakeIdentityMatrix( MatrixBody );
r3d_MakeIdentityMatrix( MatrixRightArm );
r3d_MakeIdentityMatrix( MatrixLeftArm );
MatrixRightArm[9] = 0.7F;
MatrixLeftArm[9] = -0.7F;
// Attach models
r3d_Attach3DModels( hModel[BODY], hModel[ARM], MatrixRightArm );
r3d_Attach3DModels( hModel[BODY], hModel[ARM], MatrixLeftArm );
// BOOM! Someone blows your arm off!
r3d_Detach3DModels( hModel[BODY], hModel[ARM], MatrixRightArm );
// But then, out of nowhere, a doctor appears and surgically re-attaches your arm
r3d_Attach3DModels( hModel[BODY], hModel[ARM], MatrixRightArm );
```

See Also

r3d_Attach3DModels

r3d_EnableFog

The r3d_EnableFog function will enable or disable fog effects.

```
BOOL r3d_EnableFog(
      BOOL bEnable);
```

Parameters

bEnable should be set to TRUE to enable fog or FALSE to disable it.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_SetFogAttributes( 50.0F, 100.0F, 1.0F, 0.0F, 0.0F );
r3d_EnableFog( TRUE );
```

See Also

r3d_SetFogAttributes, r3d_EnableSpecular

r3d_EnableSpecular

The r3d_EnableSpecular function will enable or disable specular reflection effects.

```
BOOL r3d_EnableSpecular(
      BOOL bEnable);
```

Parameters

bEnable should be set to TRUE to enable specular or FALSE to disable it.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_EnableSpecular( TRUE );
```

See Also

r3d_EnableFog

r3d_Enumerate

The r3d_Enumerate function enumerates all RenderIt 3D compatible DirectDraw and Direct3D devices. It should be the first function called in any RenderIt 3D application. As an alternative, you can call the r3d_EnumerateSpecific function. RenderIt 3D compatible devices must support vertex fog, multipass alpha blending, and at least one of six supported back buffer pixel formats including 16:RGB:565, 16:RGB:555, 24:RGB:888, 24:BGR:888, 32:ARGB:8888, and 32:ABGR:8888. Most systems support these minimal requirements.

```
BOOL r3d_Enumerate(
      char Driver[32][40],
      char Device[32][32][40],
      char Mode[32][32][32][40]);
```

Parameters

Driver will be filled with descriptions all of the display devices available on the target system. A value of NULL indicates the end of available descriptions.

Device will be filled with descriptions all of the Direct3D devices available on the target system for the corresponding display device. A value of NULL indicates the end of available descriptions.

Mode will be filled with descriptions all of the available video modes for a particular Direct3D device for a particular display device. A value of NULL indicates the end of available descriptions.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
char gDriver[32][40];
char gDevice[32][32][40];
char gMode[32][32][32][40];
if ( r3d_Enumerate( gDriver, gDevice, gMode ) == FALSE )
   {
   r3d_MessageBox( r3d_GetLastErrorString(), "Program Aborting", MB_OK );
   return FALSE;
   }
```

See Also

r3d_EnumerateSpecific, r3d_GetSelectionInfo, r3d_Initialize

r3d_EnumerateSpecific

The r3d_EnumerateSpecific function enumerates all RenderIt 3D compatible DirectDraw and Direct3D devices that meet a specified criteria. It should be the first function called in any RenderIt 3D application. As an alternative, you can call the r3d_Enumerate function. RenderIt 3D compatible devices must support vertex fog, multipass alpha blending, and at least one of six supported back buffer pixel formats including 16:RGB:565, 16:RGB:555, 24:RGB:888, 24:BGR:888, 32:ARGB:8888, and 32:ABGR:8888. Most systems support these minimal requirements.

```
BOOL r3d_EnumerateSpecific(
      char Driver[32][40],
      char Device[32][32][40],
      char Mode[32][32][32][40],
      BOOL bWindowedModesOnly,
      BOOL bVideoModesOnly,
      BOOL b3DEmulationOnly,
      BOOL b3DAccelerationOnly);
```

Parameters

Driver will be filled with descriptions all of the display devices available on the target system. A value of NULL indicates the end of available descriptions.

Device will be filled with descriptions all of the Direct3D devices available on the target system for the corresponding display device. A value of NULL indicates the end of available descriptions.

Mode will be filled with descriptions all of the available video modes for a particular Direct3D device for a particular display device. A value of NULL indicates the end of available descriptions.

bWindowedModesOnly should be set to TRUE if you want to enumerate all devices that support rendering to a window. You can not set both bWindowedModesOnly and bVideoModesOnly to FALSE.

bVideoModesOnly should be set to TRUE if you want to enumerate all devices that support rendering to a full screen video mode. You can not set both bWindowedModesOnly and bVideoModesOnly to FALSE.

b3DEmulationOnly should be set to TRUE if you want to enumerate all devices that support rendering in software. You can not set both b3DEmulationOnly and b3DAccelerationOnly to FALSE.

b3DAccelerationOnly should be set to TRUE if you want to enumerate all devices that support rendering with a 3D hardware accelerator. You can not set both b3DEmulationOnly and b3DAccelerationOnly to FALSE.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
char gDriver[32][40];
char gDevice[32][32][40];
char gMode[32][32][32][40];
if ( r3d_EnumerateSpecific( gDriver, gDevice, gMode, TRUE, FALSE, TRUE, TRUE ) ==
FALSE )
   {
   r3d_MessageBox( r3d_GetLastErrorString(), "Program Aborting", MB_OK );
   return FALSE;
   }
```

See Also

r3d_Enumerate, r3d_GetSelectionInfo, r3d_Initialize

r3d_FileClose

The r3d_FileClose function is provided with RenderIt 3D to simplify file access. It will close a file opened with r3d_FileOpen.

```
BOOL r3d_FileClose(
      HANDLE handle);
```

Parameters

handle is returned from a previous call to r3d_FileOpen.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
void
CopyFile(char *SrcFilename,char *DstFilename)
{
HANDLE hFile;
DWORD HelloFileSize;
char *buffer = NULL;
if ( (hFile = r3d_FileOpen( SrcFilename, R3D_READ)) == INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", SrcFilename, MB_OK );
HelloFileSize = r3d_FileSize( hFile );
if ( NULL != (buffer = new unsigned char[HelloFileSize]) )
{
r3d_FileRead( hFile, buffer, HelloFileSize );
r3d_FileSeek( hFile, 0, R3D_BEGINNING);
r3d_FileClose( hFile );
if ( (hFile = r3d_FileOpen( DstFilename, R3D_WRITE|R3D_CREATE|R3D_TRUNCATE )) ==
INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", DstFilename, MB_OK );
r3d_FileWrite( hFile, buffer, HelloFileSize );
r3d_FileClose( hFile );
delete( buffer );
buffer = NULL;
}
}
```

See Also

r3d_FileOpen, r3d_FileRead, r3d_FileSeek, r3d_FileSize, r3d_FileWrite

r3d_FileOpen

The r3d_FileOpen function is provided with RenderIt 3D to simplify file access. It will open a file for reading or writing.

HANDLE r3d_FileOpen(
    char *filename,
    int flags);

Parameters

filename is a string containing the pathname of the file to open.

flags can only be one of the following combination of constants:

R3D_READ
Opens the file for reading only.

R3D_WRITE | R3D_CREATE | R3D_TRUNCATE
Opens the file for writing only. If it doesn't exist, it gets truncated and overwritten, otherwise it is created.

R3D_WRITE | R3D_CREATE | R3D_APPEND
Opens the file for writing only. All written information will be appended to the end of the file or if it does not exist, it will be created.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
void
CopyFile(char *SrcFilename,char *DstFilename)
{
HANDLE hFile;
DWORD HelloFileSize;
char *buffer = NULL;
if ( (hFile = r3d_FileOpen( SrcFilename, R3D_READ)) == INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", SrcFilename, MB_OK );
HelloFileSize = r3d_FileSize( hFile );
if ( NULL != (buffer = new unsigned char[HelloFileSize]) )
{
r3d_FileRead( hFile, buffer, HelloFileSize );
r3d_FileSeek( hFile, 0, R3D_BEGINNING);
r3d_FileClose( hFile );
if ( (hFile = r3d_FileOpen( DstFilename, R3D_WRITE|R3D_CREATE|R3D_TRUNCATE )) ==
INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", DstFilename, MB_OK );
r3d_FileWrite( hFile, buffer, HelloFileSize );
r3d_FileClose( hFile );
delete( buffer );
buffer = NULL;
}
}
```

See Also

r3d_FileClose, r3d_FileRead, r3d_FileSeek, r3d_FileSize, r3d_FileWrite

r3d_FileRead

The r3d_FileRead function is provided with RenderIt 3D to simplify file access. It will read a file opened with r3d_FileOpen.

```
BOOL r3d_FileRead(
    HANDLE handle,
    void *buffer,
    DWORD count);
```

Parameters

handle is returned from a previous call to r3d_FileOpen.

buffer is a pointer to a variable length buffer containing the data to be read.

count should contain the number of bytes to read from the file.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
void
CopyFile(char *SrcFilename,char *DstFilename)
{
HANDLE hFile;
DWORD HelloFileSize;
char *buffer = NULL;
if ( (hFile = r3d_FileOpen( SrcFilename, R3D_READ)) == INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", SrcFilename, MB_OK );
HelloFileSize = r3d_FileSize( hFile );
if ( NULL != (buffer = new unsigned char[HelloFileSize]) )
{
r3d_FileRead( hFile, buffer, HelloFileSize );
r3d_FileSeek( hFile, 0, R3D_BEGINNING);
r3d_FileClose( hFile );
if ( (hFile = r3d_FileOpen( DstFilename, R3D_WRITE|R3D_CREATE|R3D_TRUNCATE )) ==
INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", DstFilename, MB_OK );
r3d_FileWrite( hFile, buffer, HelloFileSize );
r3d_FileClose( hFile );
delete( buffer );
buffer = NULL;
}
}
```

See Also

r3d_FileClose, r3d_FileOpen, r3d_FileSeek, r3d_FileSize, r3d_FileWrite

r3d_FileSeek

The r3d_FileSeek function is provided with RenderIt 3D to simplify file access. It will change the file position in a file previously opened with r3d_FileOpen.

```
BOOL r3d_FileSeek(
	HANDLE handle,
	long offset,
	int flags);
```

Parameters

handle is returned from a previous call to r3d_FileOpen.

offset should be the number of bytes to move from a location specified by one of the flags defined below.

flags must be one of the following constants:

R3D_BEGINNING
Move "offset" bytes from the beginning of the file for reading or writing.

R3D_END
Move "offset" bytes from the end of the file for reading or writing.

R3D_CURRENT
Move "offset" bytes from the current file position in the file for reading or writing.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
void
CopyFile(char *SrcFilename,char *DstFilename)
{
HANDLE hFile;
DWORD HelloFileSize;
char *buffer = NULL;
if ( (hFile = r3d_FileOpen( SrcFilename, R3D_READ)) == INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", SrcFilename, MB_OK );
HelloFileSize = r3d_FileSize( hFile );
if ( NULL != (buffer = new unsigned char[HelloFileSize]) )
{
r3d_FileRead( hFile, buffer, HelloFileSize );
r3d_FileSeek( hFile, 0, R3D_BEGINNING);
r3d_FileClose( hFile );
if ( (hFile = r3d_FileOpen( DstFilename, R3D_WRITE|R3D_CREATE|R3D_TRUNCATE )) ==
INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", DstFilename, MB_OK );
r3d_FileWrite( hFile, buffer, HelloFileSize );
r3d_FileClose( hFile );
delete( buffer );
buffer = NULL;
}
}
```

See Also

r3d_FileClose, r3d_FileOpen, r3d_FileRead, r3d_FileSize, r3d_FileWrite

r3d_FileSize

The r3d_FileSize function is provided with RenderIt 3D to simplify file access. It will return the size of a file previously opened with r3d_FileOpen.

```
DWORD r3d_FileSize(
    HANDLE handle);
```

Parameters

handle is returned from a previous call to r3d_FileOpen.

Return Value

The size of the file in bytes.

Source Code Sample

```
void
CopyFile(char *SrcFilename,char *DstFilename)
{
HANDLE hFile;
DWORD HelloFileSize;
char *buffer = NULL;
if ( (hFile = r3d_FileOpen( SrcFilename, R3D_READ)) == INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", SrcFilename, MB_OK );
HelloFileSize = r3d_FileSize( hFile );
if ( NULL != (buffer = new unsigned char[HelloFileSize]) )
{
r3d_FileRead( hFile, buffer, HelloFileSize );
r3d_FileSeek( hFile, 0, R3D_BEGINNING);
r3d_FileClose( hFile );
if ( (hFile = r3d_FileOpen( DstFilename, R3D_WRITE|R3D_CREATE|R3D_TRUNCATE )) ==
INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", DstFilename, MB_OK );
r3d_FileWrite( hFile, buffer, HelloFileSize );
r3d_FileClose( hFile );
delete( buffer );
buffer = NULL;
}
}
```

See Also

r3d_FileClose, r3d_FileOpen, r3d_FileRead, r3d_FileSeek, r3d_FileWrite

r3d_FileWrite

The r3d_FileWrite function is provided with RenderIt 3D to simplify file access. It will write data to a file previously opened with r3d_FileOpen.

```
BOOL r3d_FileWrite(
      HANDLE handle,
      void *buffer,
      DWORD count);
```

Parameters

handle is returned from a previous call to r3d_FileOpen.

buffer is a pointer to a buffer containing the data to be written.

count should contain the number of bytes to write to the file.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
void
CopyFile(char *SrcFilename,char *DstFilename)
{
HANDLE hFile;
DWORD HelloFileSize;
char *buffer = NULL;
if ( (hFile = r3d_FileOpen( SrcFilename, R3D_READ)) == INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", SrcFilename, MB_OK );
HelloFileSize = r3d_FileSize( hFile );
if ( NULL != (buffer = new unsigned char[HelloFileSize]) )
{
r3d_FileRead( hFile, buffer, HelloFileSize );
r3d_FileSeek( hFile, 0, R3D_BEGINNING);
r3d_FileClose( hFile );
if ( (hFile = r3d_FileOpen( DstFilename, R3D_WRITE|R3D_CREATE|R3D_TRUNCATE )) ==
INVALID_HANDLE_VALUE)
r3d_MessageBox( "Error Opening File", DstFilename, MB_OK );
r3d_FileWrite( hFile, buffer, HelloFileSize );
r3d_FileClose( hFile );
delete( buffer );
buffer = NULL;
}
}
```

See Also

r3d_FileClose, r3d_FileOpen, r3d_FileRead, r3d_FileSeek, r3d_FileSize

r3d_Finished

The r3d_Finished function releases all DirectX objects created by RenderIt 3D as well as all DirectDraw surfaces and virtual surfaces. You should call r3d_Finished whenever your application exits provided that you have previously called r3d_Initialize.

void r3d_Finished(void);

Parameters

None.

Return Value

None.

Source Code Sample

```
case WM_DESTROY:
r3d_Finished();
PostQuitMessage(0);
return 0L;
```

See Also

r3d_Initialize

r3d_ForceAlphaRendering

The r3d_ForceAlphaRendering function allows RenderIt 3D to render the alpha polygons from 3D models as they are passed to the renderer. By default, RenderIt 3D stores alpha polygons, sorts them in a back to front order and then renders them within the r3d_RenderEnd function.

```
void r3d_ForceAlphaRendering(
        BOOL bEnable);
```

Parameters

bEnable should be set to TRUE to force 3D model alpha polygons to be rendered as they are passed to the renderer.

Return Value

None.

Source Code Sample

```
r3d_ForceAlphaRendering( TRUE );
```

See Also

r3d_AlphaPolygonSort, r3d_RenderEnd

r3d_Get3DModelMaterialCount

The r3d_Get3DModelMaterialCount function provides a means of obtaining the number of materials within a specified 3D model.

```
BOOL r3d_Get3DModelMaterialCount(
        int hModel,
        int *MaterialCount);
```

Parameters

hModel is the handle of a model obtained from a previous call to r3d_ListLoad.

MaterialCount is a pointer to variable that will contain the number of materials on the model.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Indirectly altering geometry
//
// Indirectly altering the geometric data is useful for
// altering a models' shape without introducing any new
// 3D data or adding new vertices and triangles. You
// simply alter the Vertex and Triangle data through
// their pointers.
//
// Note: It is not safe to increase TriangleCount or
// VertexCount using this method.
//
// This method is destructive in that once you change
// the data, you can not change it back unless you have
// loaded two copies of the model and copy the model
// data from the non-destructed model to the destructed
// model.
//
// This example assumes that hModel[H_MODEL] is a valid
// model handle.
//
int MaterialCount, LoopVar;
BOOL *bPreLit;
unsigned long *VertexCount, *TriangleCount;
D3DLVERTEX *Vertex;
unsigned short *Triangle;
r3d_Get3DModelMaterialCount(hModel[H_MODEL],&MaterialCount);
for ( LoopVar=0; LoopVar<MaterialCount; LoopVar++ )
{
// Get material offset information
r3d_Get3DModelMaterialGeometry( hModel[H_MODEL], LoopVar, &bPreLit, &VertexCount,
(D3DVERTEX **)&Vertex, &TriangleCount, &Triangle );
// You could alter your vertex information here
```

```
    // example: Vertex[0]-=5;
    }
    // You can now render the altered model at handle hModel[H_MODEL]
    r3d_Render3DModel(hModel[H_MODEL],MatrixMainCharacter);
```

See Also

r3d_Get3DModelMaterialGeometry, r3d_Get3DModelMaterialHandle

r3d_Get3DModelMaterialGeometry

The r3d_Get3DModelMaterialGeometry function provides pointers to the internal vertex and triangle data associated with a particular material of a 3D model. This function is useful for the implementation of 3D model animation. For example, you could use this function as a basis for key-frame animation for animating 3D characters, or you could use this function for dynamically processing geometry.

```
BOOL r3d_Get3DModelMaterialGeometry(
     int hModel,
     int ModelMaterialOffset,
     BOOL **bPreLit,
     unsigned long **VertexCount,
     D3DVERTEX **Vertex,
     unsigned long **TriangleCount,
     unsigned short **Triangle);
```

Parameters

hModel is the model handle obtained with a previous call to r3d_ListLoad.

ModelMaterialOffset is the offset of a material within the model. This parameter can not equal or exceed the number of materials supplied by r3d_Get3DModelMaterialCount. All vertex and polygon data are grouped by material. If you passed an offset of zero for this parameter, you would be indexing the first material of the model as well as their corresponding polygons and vertices. To find out which material offset corresponds to a particular material or texture of a particular RenderIt 3D model (*.R3D), load the RenderIt 3D model into Materialize 3D! and inspect the material numbers shown in the "Edit Materials" dialog box. The material numbers are equal to the parameter that should be passed to this function. If you save the model within Materialize 3D! after inspection, the material numbers may be re-assigned possibly making them invalid.

BPreLit should be passed an address of a pointer to type BOOL. The pointer will be assigned to the address of a type BOOL value which is TRUE if the vertex data is using D3DLVERTEX vertices instead of D3DVERTEX vertices.

VertexCount should be passed an address of a pointer to type unsigned long. The pointer will be assigned to the address of a type unsigned long which will contain the number of vertices in the Vertex parameter.

Vertex should be passed an address of a pointer to type D3DVERTEX. The pointer will be assigned to the base address of a D3DVERTEX array where each element describes one vertex. If the bPreLit parameter is TRUE, this pointer will be assigned to the base address of a D3DLVERTEX array. Typecasting D3DVERTEX to type D3DLVERTEX is permitted when the bPreLit parameter is TRUE.

TriangleCount should be passed an address of a pointer to type unsigned long. The pointer will be assigned to the address of a type unsigned long which will contain the number of triangles (times three) in the Triangle parameter.

Triangle should be passed an address of a pointer to type unsigned short. The pointer will be assigned to the base address of an unsigned short array where each element will contain an index into the Vertex array. The first three elements describe the first triangle, the second three elements describe the second triangle, and so on.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Indirectly altering geometry
//
// Indirectly altering the geometric data is useful for
// altering a models' shape without introducing any new
// 3D data or adding new vertices and triangles. You
// simply alter the Vertex and Triangle data through
// their pointers.
//
// Note: It is not safe to increase TriangleCount or
// VertexCount using this method.
//
// This method is destructive in that once you change
// the data, you can not change it back unless you have
// loaded two copies of the model and copy the model
// data from the non-destructed model to the destructed
// model.
//
// This example assumes that hModel[H_MODEL] is a valid
// model handle.
//
int MaterialCount, LoopVar;
BOOL *bPreLit;
unsigned long *VertexCount, *TriangleCount;
D3DLVERTEX *Vertex;
unsigned short *Triangle;
r3d_Get3DModelMaterialCount( hModel[H_MODEL], &MaterialCount );
for ( LoopVar=0; LoopVar<MaterialCount; LoopVar++ )
{
// Get material offset information
r3d_Get3DModelMaterialGeometry( hModel[H_MODEL], LoopVar, &bPreLit, &VertexCount,
(D3DVERTEX **)&Vertex, &TriangleCount, &Triangle );
// You could alter your vertex information here
// example: Vertex[0]-=5;
}
// You can now render the altered model at handle hModel[H_MODEL]
r3d_Render3DModel( hModel[H_MODEL], MatrixMainCharacter );
```

See Also

r3d_Get3DModelMaterialHandle, r3d_Get3DModelMaterialCount

r3d_Get3DModelMaterialHandle

r3d_Get3DModelMaterialHandle is useful for obtaining a valid material handle from a RenderIt 3D Model. The material handle may then be used for subsequent calls to r3d_SetMaterial, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSurfaceUnlock, or r3d_TextureAnimateSwap.

```
BOOL r3d_Get3DModelMaterialHandle(
    int hModel,
    int ModelMaterialOffset,
    int *hMaterial);
```

Parameters

hModel is the model handle obtained with a previous call to r3d_ListLoad.

ModelMaterialOffset is the offset of a material within the model. This parameter can not equal or exceed the number of materials supplied by r3d_Get3DModelMaterialCount. If you passed an offset of zero for this parameter, you would be indexing the first material of the model. To find out which material offset corresponds to a particular material or texture of a particular RenderIt 3D model (*.R3D), load the RenderIt 3D model into Materialize 3D! and inspect the material numbers shown in the "Edit Materials" dialog box. The material numbers are equal to the parameter that should be passed to this function. If you save the model within Materialize 3D! after inspection, the material numbers may be re-assigned possibly making them invalid.

hMaterial should be the address of an integer which will be assigned a valid handle of the material.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL] is a valid
// model handle.
//
int MaterialCount, hMaterial;
r3d_Get3DModelMaterialCount( hModel[H_MODEL], &MaterialCount );
r3d_Get3DModelMaterialHandle( hModel[H_MODEL], MaterialCount-1, &hMaterial );
```

See Also

r3d_Get3DModelMaterialGeometry, r3d_Get3DModelMaterialCount

r3d_GetBackBufferSurface

The r3d_GetBackBufferSurface function returns an LPDIRECTDRAWSURFACE4 pointer to the back buffer surface.

LPDIRECTDRAWSURFACE4 r3d_GetBackBufferSurface(void);

Parameters

None.

Return Value

Returns a DIRECTDRAWSURFACE4 pointer to the back buffer surface.

Source Code Sample

```
int hBitmap,hDDSurface,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
// Attempt to create the DirectDraw surface in video memory. If that fails we simply
create the surface in system memory.
if (!r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,1))
r3d_DDSurfaceCreateFromSurface(&hDDSurface,hBitmap,0);
r3d_SurfaceDestroy(hBitmap);
// Blit to back buffer
r3d_DDSurfaceBlit( r3d_DDSurfaceGet(hDDSurface), r3d_GetBackBufferSurface(),
r3d_DDSurfaceGetWidth(hDDSurface), r3d_DDSurfaceGetHeight(hDDSurface), Width, Height,
0, r3d_DDSurfaceGetWidth(hDDSurface)-1, 0, r3d_DDSurfaceGetHeight(hDDSurface)-1, 0,
Width-1, 0, Height-1, FALSE, FALSE);
r3d_DDSurfaceDestroy(hDDSurface);
```

See Also

r3d_GetFrontBufferSurface

r3d_GetBoundingBoxCoords

The r3d_GetBoundingBoxCoords function provides the bounding box information that encompasses a specified 3D model. It is sometimes useful for quickly determining whether or not further collision detection is required at the polygon level.

```
BOOL r3d_GetBoundingBoxCoords(
    long hModel,
    float *pf4x3ModelMatrix,
    float *fMinX,
    float *fMinY,
    float *fMinZ,
    float *fMaxX,
    float *fMaxY,
    float *fMaxZ);
```

Parameters

hModel is the model handle obtained with a previous call to r3d_ListLoad.

pf4x3ModelMatrix should be set to NULL if you only want to obtain the bounding box coordinates in a non-transformed state (in the models local system). If you would like to obtain the bounding box information of a 3D model in a transformed state (in the world system), you should supply the address of a 12 element matrix array for this parameter which is usually the model's matrix. It will be easier to implement your own collision detection routine by providing the matrix of the model.

fMinX,fMinY,fMinZ,fMaxX,fMaxY,fMaxZ are pointers to variables of type float that will be filled with the minimum x, y, z and the maximum x, y, z coordinates describing the bounding box of the 3D model.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL] is a valid
// model handle and that MatrixModel[H_MODEL] an
// appropriate matrix.
//
float _3DExtents[6];
r3d_GetBoundingBoxCoords( hModel[H_MODEL], MatrixModel[H_MODEL], &_3Dextents[0],
&_3Dextents[1], &_3Dextents[2], &_3Dextents[3], &_3Dextents[4], &_3Dextents[5] );
```

See Also

r3d_GetBoundingSphereRadius

r3d_GetBoundingSphereRadius

The r3d_GetBoundingSphereRadius function provides the radius information of a sphere that encompasses a specified 3D model. It is sometimes useful for quickly determining whether or not further collision detection is required at the polygon level.

```
BOOL r3d_GetBoundingSphereRadius(
    long hModel,
    float *fRadius);
```

Parameters

hModel is the model handle obtained with a previous call to r3d_ListLoad.

fRadius is a pointer to variable of type float that will be filled with the radius of the sphere.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL] is a valid
// model handle.
//
float fRadius;
r3d_GetBoundingBoxCoords( hModel[H_MODEL], &fRadius );
```

See Also

r3d_GetBoundingBoxCoords

r3d_GetCurrentTick

The r3d_GetCurrentTick function returns RenderIt 3D's high-resolution tick counter variable running at 1024 hertz.

unsigned long r3d_GetCurrentTick(void);

Parameters

None.

Return Value

The r3d_GetCurrentTick function returns the high-resolution timer variable running at 1024 hertz as type unsigned long.

Source Code Sample

```
float FrameTime;
unsigned long CurrentTick,StartTick,NumTicks;
// Somewhere in your main loop:
{
CurrentTick = r3d_GetCurrentTick();
NumTicks = ( CurrentTick-StartTick );
// Avoid problems (ex. user switches to another application)
if (NumTicks<1 || NumTicks>4096)
NumTicks = 1024;
StartTick = CurrentTick;
FrameTime = (float)NumTicks/1024.0F;
// FrameTime now contains the number of seconds between now and the previous frame.
}
```

See Also

r3d_SetCurrentTick

r3d_GetDevice

The r3d_GetDevice function returns a pointer to the Direct3D device. This is useful if you want to access it directly.

LPDIRECT3DDEVICE3 r3d_GetDevice(void);

Parameters

None.

Return Value

Returns a DIRECT3DDEVICE3 pointer to the Direct3D device.

Source Code Sample

```
r3d_GetDevice()->SetRenderState( D3DRENDERSTATE_ZWRITEENABLE, (DWORD)FALSE );
```

See Also

DirectX SDK Documentation, r3d_GetDeviceDesc

r3d_GetDeviceDesc

The r3d_GetDeviceDesc function returns a pointer to the currently selected Direct3D device description.

LPD3DDEVICEDESC r3d_GetDeviceDesc(void);

Parameters

None.

Return Value

Returns a D3DDEVICEDESC pointer to the current Direct3D device.

Source Code Sample

```
BOOL gbClampingSupported;
if ( r3d_GetDevice()->dwFlags & D3DDD_TRICAPS )
{
if ( r3d_GetDevice()->dpcTriCaps.dwTextureAddressCaps & D3DPTADDRESSCAPS_CLAMP )
gbClampingSupported = TRUE;
}
```

See Also

DirectX SDK Documentation, r3d_GetDevice

r3d_GetDirect3D

The r3d_GetDirect3D function returns a pointer to the Direct3D object. This is useful if you want to access it directly.

LPDIRECT3D3 r3d_GetDirect3D(void);

Parameters

None.

Return Value

Returns a DIRECT3D3 pointer to the Direct3D object.

Source Code Sample

```
// Global data area
LPDIRECT3DVERTEXBUFFER g_pvbVertexBuffer = NULL;
BOOL gb3Daccelerated = FALSE;
char gDriver[32][40];
char gDevice[32][32][40];
char gMode[32][32][32][40];
int gDriverSelection=0;
int gDeviceSelection=0;
int gModeSelection=0;
// Somewhere in the WinMain function . . .
if ( r3d_Enumerate( gDriver, gDevice, gMode ) == FALSE )
{
r3d_MessageBox(r3d_GetLastErrorString(),"Program Aborting",MB_OK);
return FALSE;
}
// gDriverSelection, gDeviceSelection, gModeSelection initialized in dialog box
if ( DialogBox( hInst, MAKEINTRESOURCE(IDD_RENDEROPTIONS), NULL,
(DLGPROC)DialogSelectDeviceProc) == 0 )
return FALSE;
int Width, Height;
r3d_GetSelectionInfo( gDriverSelection, gDeviceSelection, gModeSelection, &Width,
&Height, NULL, &gb3Daccelerated );
// . . .
// . . .
// . . .
// Create a vertex buffer
D3DVERTEXBUFFERDESC vbdesc;
ZeroMemory(&vbdesc, sizeof(D3DVERTEXBUFFERDESC));
vbdesc.dwSize = sizeof(D3DVERTEXBUFFERDESC);
vbdesc.dwCaps = 0L;
vbdesc.dwFVF = D3DFVF_VERTEX;
vbdesc.dwNumVertices = NUM_FLAG_VERTICES;
// If this is not a hardware device, make sure the
// vertex buffer uses system memory
if ( gb3Daccelerated == FALSE )
vbdesc.dwCaps |= D3DVBCAPS_SYSTEMMEMORY;
// Create a clipping-capable vertex buffer.
r3d_GetDirect3D()->CreateVertexBuffer( &vbdesc, &g_pvbVertexBuffer, 0L, NULL );
```

See Also

DirectX SDK Documentation

r3d_GetDirectDraw

The r3d_GetDirectDraw function returns a pointer to the DirectDraw object. This is useful if you want to access it directly.

LPDIRECTDRAW4 r3d_GetDirectDraw(void);

Parameters

None.

Return Value

Returns a DIRECTDRAW4 pointer to the DirectDraw object.

Source Code Sample

```
DDSCAPS2 ddsCaps2;
DWORD dwTotalMem, dwFreeMem;
ZeroMemory( &ddsCaps2, sizeof(ddsCaps2) );
ddsCaps2.dwCaps = DDSCAPS_TEXTURE;
r3d_GetDirectDraw()->GetAvailableVidMem( &ddsCaps2, &dwTotalMem, &dwFreeMem );
```

See Also

DirectX SDK Documentation

r3d_GetDisplayMode

The r3d_GetDisplayMode function returns information about the current display mode or the windowed mode.

```
void r3d_GetDisplayMode(
        int *Width,
        int *Height,
        unsigned long *lBPP,
        unsigned long *RBitMask,
        unsigned long *GBitMask,
        unsigned long *BBitMask,
        int *PixelFormatCode);
```

Parameters

Width,Height are pointers to variables of type int that will be filled with the width and height of the back buffer. You may pass NULL if you do not need this information.

lBPP is a pointer to a variable of type unsigned long that will be filled with the number of bits-per-pixel of the back buffer. You may pass NULL if you do not need this information.

RBitMask,GBitMask,BBitMask are pointers to variables of type unsigned long that will be filled with the bit masks of the red, green and blue color components of the back buffer. You may pass NULL if you do not need this information.

PixelFormatCode is a pointer to a variable of type int that will be filled with the pixel format code of the back buffer surface. RenderIt 3D uses pixel format codes internally. The following list shows how the PixelFormatCode should be interpreted:

PixelFormatCode = 0:
BPP: 16
RBitMask: 0x0000F800
GBitMask: 0x000007E0
BBitMask: 0x0000001F

PixelFormatCode = 1:
BPP: 16
RBitMask: 0x00007C00
GBitMask: 0x000003E0
BBitMask: 0x0000001F

PixelFormatCode = 2:
BPP: 24
RBitMask: 0x00FF0000
GBitMask: 0x0000FF00
BBitMask: 0x000000FF

PixelFormatCode = 3:
BPP: 24
RBitMask: 0x000000FF
GBitMask: 0x0000FF00

BBitMask: 0x00FF0000

PixelFormatCode = 4:
BPP: 32
RBitMask: 0x00FF0000
GBitMask: 0x0000FF00
BBitMask: 0x000000FF

PixelFormatCode = 5:
BPP: 32
RBitMask: 0x000000FF
GBitMask: 0x0000FF00
BBitMask: 0x00FF0000

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int gWidth,gHeight;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL );
```

r3d_GetFrontBufferSurface

The r3d_GetFrontBufferSurface function returns an LPDIRECTDRAWSURFACE4 pointer to the front buffer surface - also known as the primary surface.

LPDIRECTDRAWSURFACE4 r3d_GetFrontBufferSurface(void);

Parameters

None.

Return Value

Returns a DIRECTDRAWSURFACE4 pointer to the front buffer surface.

Source Code Sample

```
LPDIRECTDRAWSURFACE4 lpPrimarySurface;
lpPrimarySurface = r3d_GetFrontBufferSurface();
```

See Also

r3d_GetBackBufferSurface

r3d_GetLastErrorString

The r3d_GetLastErrorString returns a pointer to an internal string describing the last error that occurred. It is useful for debugging your application and is easily used in conjunction with r3d_MessageBox.

char *r3d_GetLastErrorString(void);

Parameters

None.

Return Value

Returns a pointer to an internal error string of the last error that occurred.

Source Code Sample

```
if ( !r3d_2DBegin( NULL, NULL, NULL) )
r3d_MessageBox( r3d_GetLastErrorString(), "Error", MB_OK );
```

See Also

r3d_MessageBox

r3d_GetLight

The r3d_GetLight function returns a pointer to the default Direct3D directional light used by RenderIt 3D.

LPDIRECT3DLIGHT r3d_GetLight(void);

Parameters

None.

Return Value

Returns a DIRECT3DLIGHT pointer to the default directional light.

Source Code Sample

```
LPDIRECT3DLIGHT lpLight;
lpLight = r3d_GetLight();
```

See Also

DirectX SDK Documentation, r3d_LightRemoveDefault, r3d_LightSetAmbient

r3d_GetMaxTextureWidthHeight

The r3d_GetMaxTextureWidthHeight function returns the maximum allowable width and height of a texture supported by the current Direct3D device.

int r3d_GetMaxTextureWidthHeight(void);

Parameters

None.

Return Value

Returns a value of type int indicating the maximum allowable width and height of a texture.

Source Code Sample

```
if ( r3d_GetMaxTextureWidthHeight() > 256 )
r3d_ListAdd3DModel( "Model.R3D", FALSE );
else
r3d_ListAdd3DModel( "AlternateModel.R3D", FALSE );
```

See Also

r3d_GetMinTextureWidthHeight

r3d_GetMinTextureWidthHeight

The r3d_GetMinTextureWidthHeight function returns the minimum allowable width and height of a texture supported by the current Direct3D device.

int r3d_GetMinTextureWidthHeight(void);

Parameters

None.

Return Value

Returns a value of type int indicating the minimum allowable width and height of a texture.

Source Code Sample

```
int MinTextureSize;
MinTextureSize = r3d_GetMinTextureWidthHeight();
```

See Also

r3d_GetMaxTextureWidthHeight

r3d_GetSelectionInfo

The r3d_GetSelectionInfo function provides information for a particular configuration provided by the r3d_Enumerate function.

```
BOOL r3d_GetSelectionInfo(
      int DriverSelection,
      int DeviceSelection,
      int ModeSelection,
      int *Width,
      int *Height,
      unsigned long *dwBitDepth,
      BOOL *b3DAccelerated);
```

Parameters

DriverSelection is an index into one of the enumerated display devices.

DeviceSelection is an index into one of the enumerated Direct3D devices.

ModeSelection is an index into one of the enumerated display modes.

Width,Height is a pointer to a variable of type int that will be filled with the dimensions of the video mode. A value of 0 in width or height indicates the "Window" selection.

dwBitDepth is a pointer to a variable of type unsigned long that will be filled with the number of bits-per-pixel of the selection.

b3DAccelerated is a pointer to a variable of type BOOL that will be set to TRUE if the selection corresponds to a 3D hardware accelerated device.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
//
// Somewhere in the global data area
//
BOOL gb3Daccelerated = FALSE;
BOOL gbWindowed = FALSE;
HWND ghWwnd;
char gDriver[32][40];
char gDevice[32][32][40];
char gMode[32][32][32][40];
int gDriverSelection=0;
int gDeviceSelection=0;
int gModeSelection=0;
//
// Somewhere in the WinMain function . . .
//
RECT ViewportRect, WindowRect;
if ( r3d_Enumerate( gDriver, gDevice, gMode ) == FALSE )
{
r3d_MessageBox(r3d_GetLastErrorString(),"Program Aborting",MB_OK);
return FALSE;
}
// gDriverSelection, gDeviceSelection, gModeSelection initialized in dialog box
if ( DialogBox( hInst, MAKEINTRESOURCE(IDD_RENDEROPTIONS), NULL,
(DLGPROC)DialogSelectDeviceProc) == 0 )
return FALSE;
int Width, Height;
r3d_GetSelectionInfo( gDriverSelection, gDeviceSelection, gModeSelection, &Width,
&Height, NULL, &gb3Daccelerated );
if (Width==0 && Height==0)
{
// Selection is windowed, set up for windowed mode.
gbWindowed = TRUE;
WNDCLASS WndClass = { CS_HREDRAW|CS_VREDRAW, WinProc, 0, 0, hInst, LoadIcon( hInst,
IDI_APPLICATION ), LoadCursor( NULL, IDC_ARROW ), (HBRUSH)GetStockObject( BLACK_BRUSH
), APP_NAME, APP_NAME };
RegisterClass( &WndClass );
ghWnd = CreateWindow( APP_NAME, APP_NAME, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
CW_USEDEFAULT, 400, 300+20, 0L, 0L, hInst, 0L );
ShowWindow( ghWnd, SW_SHOWNORMAL );
SetFocus( ghWnd );
UpdateWindow( ghWnd );
// For windowed mode only...
GetClientRect( ghWnd, &ViewportRect );
GetClientRect( ghWnd, &WindowRect );
ClientToScreen( ghWnd, (POINT*)&WindowRect.left );
ClientToScreen( ghWnd, (POINT*)&WindowRect.right );
r3d_WindowInit( WindowRect, ViewportRect );
}
else
{
// Selection is fullscreen, set up for fullscreen mode.
gbWindowed = FALSE;
WNDCLASS WndClass = { CS_HREDRAW|CS_VREDRAW, WinProc, 0, 0, hInst, LoadIcon( hInst,
IDI_APPLICATION ), LoadCursor( NULL, IDC_ARROW ), NULL, APP_NAME, APP_NAME };
RegisterClass( &WndClass );
ghWnd = CreateWindowEx( WS_EX_TOPMOST, APP_NAME, APP_NAME, WS_POPUP, 0, 0,
GetSystemMetrics( SM_CXSCREEN ), GetSystemMetrics( SM_CYSCREEN ), NULL, NULL, hInst,
NULL );
ShowWindow( ghWnd, SW_SHOWNORMAL );
UpdateWindow( ghWnd );
SetFocus( ghWnd );
SetCursor( NULL );
}
if ( r3d_Initialize( ghWnd, 100000.0F, 75.0F, gDriverSelection, gDeviceSelection,
gModeSelection ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "Fatal Error - Program Aborting", MB_OK );
r3d_Finished();
return FALSE;
}
```

See Also

r3d_Enumerate, r3d_Initialize

r3d_GetVersion

The r3d_GetVersion function provides the current version number of RenderIt 3D. For example, if major is 3, tenths is 0 and hundredths is 1, then you are using RenderIt 3D version 3.01.

```
void r3d_GetVersion(
        int *major,
        int *tenths,
        int *hundredths);
```

Parameters

major is the major version number.

tenths is one part of the minor version number.

hundredths is another part of the minor version number.

Return Value

None.

Source Code Sample

```
int major,tenths,hundredths;
r3d_GetVersion( &major, &tenths, &hundredths );
sprintf( buffer, "RenderIt 3D! Version %d.%d.%d", major, tenths, hundredths );
r3d_MessageBox( buffer, "Version #", MB_OK );
```

r3d_GetViewport

The r3d_GetViewport function returns a pointer to the Direct3D viewport object.

LPDIRECT3DVIEWPORT3 r3d_GetViewport(void);

Parameters

None.

Return Value

Returns a DIRECT3DVIEWPORT3 pointer to the Direct3D viewport object.

Source Code Sample

```
LPDIRECT3DVIEWPORT3 lpViewport;
lpViewport = r3d_GetViewport();
```

See Also

DirectX SDK Documentation

r3d_GetZBufferSurface

The r3d_GetZBufferSurface function returns a pointer to the Direct3D z-buffer surface.

LPDIRECTDRAWSURFACE4 r3d_GetZBufferSurface(void);

Parameters

None.

Return Value

Returns a DIRECTDRAWSURFACE4 pointer to the Direct3D z-buffer surface.

Source Code Sample

```
LPDIRECTDRAWSURFACE4 lpZBuffer;
lpZBuffer = r3d_GetZBufferSurface();
```

See Also

DirectX SDK Documentation

r3d_Initialize

The r3d_Initialize function initializes RenderIt 3D and sets up DirectDraw and Direct3D Immediate Mode. The r3d_Initialize function should be preceded by a call to r3d_Enumerate.

```
BOOL r3d_Initialize(
    HWND hWnd,
    float fFarClipPlane,
    float fFov,
    int DriverSelection,
    int DeviceSelection,
    int ModeSelection);
```

Parameters

hWnd should be your applications main window handle.

fFarClipPlane is the distance to the far clipping plane. Any 3D objects that are beyond this point will not be displayed. For best rendering results, you want this value to closely correspond to the maximum view distance of your scene.

fFov is the field-of-view angle expressed in angle measurements and typically ranges anywhere from 60 to 90 degrees. Note: This parameter is not measured in radians.

DriverSelection is an index into one of the enumerated display devices and will be used for initialization.

DeviceSelection is an index into one of the enumerated Direct3D devices and will be used for initialization.

ModeSelection is an index into one of the enumerated display modes and will be used for initialization.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
//
// Somewhere in the global data area
//
BOOL gb3Daccelerated = FALSE;
BOOL gbWindowed = FALSE;
HWND ghWwnd;
char gDriver[32][40];
char gDevice[32][32][40];
char gMode[32][32][32][40];
```

```
int gDriverSelection=0;
int gDeviceSelection=0;
int gModeSelection=0;
//
// Somewhere in the WinMain function . . .
//
RECT ViewportRect, WindowRect;
if ( r3d_Enumerate( gDriver, gDevice, gMode ) == FALSE )
{
r3d_MessageBox(r3d_GetLastErrorString(),"Program Aborting",MB_OK);
return FALSE;
}
// gDriverSelection, gDeviceSelection, gModeSelection initialized in dialog box
if ( DialogBox( hInst, MAKEINTRESOURCE(IDD_RENDEROPTIONS), NULL,
(DLGPROC)DialogSelectDeviceProc) == 0 )
return FALSE;
int Width, Height;
r3d_GetSelectionInfo( gDriverSelection, gDeviceSelection, gModeSelection, &Width,
&Height, NULL, &gb3Daccelerated );
if (Width==0 && Height==0)
{
// Selection is windowed, set up for windowed mode.
gbWindowed = TRUE;
WNDCLASS WndClass = { CS_HREDRAW|CS_VREDRAW, WinProc, 0, 0, hInst, LoadIcon( hInst,
IDI_APPLICATION ), LoadCursor( NULL, IDC_ARROW ), (HBRUSH)GetStockObject( BLACK_BRUSH
), APP_NAME, APP_NAME };
RegisterClass( &WndClass );
ghWnd = CreateWindow( APP_NAME, APP_NAME, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
CW_USEDEFAULT, 400, 300+20, 0L, 0L, hInst, 0L );
ShowWindow( ghWnd, SW_SHOWNORMAL );
SetFocus( ghWnd );
UpdateWindow( ghWnd );
// For windowed mode only...
GetClientRect( ghWnd, &ViewportRect );
GetClientRect( ghWnd, &WindowRect );
ClientToScreen( ghWnd, (POINT*)&WindowRect.left );
ClientToScreen( ghWnd, (POINT*)&WindowRect.right );
r3d_WindowInit( WindowRect, ViewportRect );
}
else
{
// Selection is fullscreen, set up for fullscreen mode.
gbWindowed = FALSE;
WNDCLASS WndClass = { CS_HREDRAW|CS_VREDRAW, WinProc, 0, 0, hInst, LoadIcon( hInst,
IDI_APPLICATION ), LoadCursor( NULL, IDC_ARROW ), NULL, APP_NAME, APP_NAME };
RegisterClass( &WndClass );
ghWnd = CreateWindowEx( WS_EX_TOPMOST, APP_NAME, APP_NAME, WS_POPUP, 0, 0,
GetSystemMetrics( SM_CXSCREEN ), GetSystemMetrics( SM_CYSCREEN ), NULL, NULL, hInst,
NULL );
ShowWindow( ghWnd, SW_SHOWNORMAL );
UpdateWindow( ghWnd );
SetFocus( ghWnd );
SetCursor( NULL );
}
if ( r3d_Initialize( ghWnd, 100000.0F, 75.0F, gDriverSelection, gDeviceSelection,
gModeSelection ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "Fatal Error - Program Aborting", MB_OK );
r3d_Finished();
return FALSE;
}
```

See Also

r3d_Enumerate, r3d_GetSelectionInfo, r3d_Finished

r3d_LightRemoveDefault

The r3d_LightRemoveDefault function will remove the default Direct3D directional light. After this function is called, r3d_SetLightLocationColor will have no effect.

BOOL r3d_LightRemoveDefault(void);

Parameters

None.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_LightRemoveDefault();
```

See Also

r3d_LightSetAmbient, r3d_SetLightLocationColor

r3d_LightSetAmbient

The r3d_LightSetAmbient function sets the amount of ambient light.

```
BOOL r3d_LightSetAmbient(
    float r,
    float g,
    float b);
```

Parameters

r,g,b should describe the red, green and blue color components of the ambient light and should range from 0.0 to 1.0.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_LightSetAmbient( 0.4F, 0.4F, 0.4F );
```

See Also

r3d_SetLightLocationColor

r3d_ListAdd3DModel

The r3d_ListAdd3DModel function adds a RenderIt 3D model to an internal list maintained by RenderIt 3D. All models added to the list with r3d_ListAdd3DModel can be loaded with a subsequent call to r3d_ListLoad.

```
BOOL r3d_ListAdd3DModel(
        char *pFilename,
        BOOL bMipMap);
```

Parameters

pFilename should be filename of the model to load. Only 3D models converted to the *.R3D format by Materialize 3D! can be loaded.

bMipMap should be set to TRUE if you want RenderIt 3D to automatically create mipmap levels for the textures on the model.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Somewhere in the global data area
#define H_BACKDROP              0
#define H_CITYBLOCK             1
#define H_TRAFFICSIGNAL   2
#define H_STOPSIGN              3
#define H_BUILDING              4
#define H_MONSTERTRUCK    5
#define H_TRUCKWHEEL      6
#define H_TREE            7
#define H_TREE            8
#define H_PERSON          9
#define NUM_MODELS              10
char lpModels[NUM_MODELS][_MAX_PATH] = {
   "Backdrop.r3d",
   "CityBlock.r3d",
   "TrafficSignal.r3d",
   "StopSign.r3d",
   "Building.r3d",
   "MonsterTruck.r3d",
   "TruckWheel.r3d",
   "Tree.r3d",
   "Tree2.r3d",
   "Steve.r3d",
};
long hModel[NUM_MODELS];
float MatrixModel[NUM_MODELS];
#define NUM_MATERIALS     1
long hMaterial[NUM_MATERIALS];
```

```
// Somewhere in a LoadLevel type of function:
int LoopVar;
for (LoopVar=0; LoopVar<NUM_MODELS; LoopVar++)
{
if ( r3d_ListAdd3DModel( lpModels[LoopVar], FALSE) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), lpModels[LoopVar], MB_OK );
return FALSE;
}
r3d_MakeIdentityMatrix( MatrixModel[LoopVar] );
}
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F,
0.0F, TRUE, "Texture.bmp", TRUE, "AlphaTexture.bmp", TRUE );
if ( r3d_ListLoad( &hModel[0], &hMaterial[0] ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "r3d_ListLoad", MB_OK );
return FALSE;
}
```

See Also

r3d_ListAddMaterial, r3d_ListLoad, r3d_ListUnload

r3d_ListAddMaterial

The r3d_ListAddMaterial function adds a material to an internal list maintained by RenderIt 3D. All materials added to the list with r3d_ListAddMaterial can be loaded with a subsequent call to r3d_ListLoad. This function is useful if you want to render polygons with Direct3D but need to create one or materials first.

```
BOOL r3d_ListAddMaterial(
        float r,
        float g,
        float b,
        float a,
        float pow,
        float sr,
        float sg,
        float sb,
        float er,
        float eg,
        float eb,
        BOOL bLoadTexture,
        char *TextureFilenameBMP,
        BOOL bLoadAlpha,
        char *AlphaFilenameBMP,
        BOOL bMipMap);
```

Parameters

r,g,b,a are the red, green, blue, and alpha color components of the material and range from 0.0 to 1.0.

pow indicates the power of the specular reflection. The higher the power, the more shinny the material appears.

sr,sg,sb are the red, green, and blue color components of the specular reflection and range from 0.0 to 1.0. A value of 0.0 for sr,sg and sb produces no specular reflection.

er,eg,eb are the red, green, and blue color components of the emissive color and range from 0.0 to 1.0. The higher the values, the more the material will "glow".

bLoadTexture should be set to TRUE if you want this material to have a texture.

TextureFilename is the filename of the texture that you want to assign to the material. The texture must be a 256 color or 24-bit uncompressed Windows bitmap in *.BMP format.

bLoadAlphaTexture should be set to TRUE if you want the texture to have an alpha channel.

AlphaTextureFilename is the filename of the "alpha" texture. The texture must be a 256 color or 24-bit uncompressed Windows bitmap in *.BMP format. The bitmap should contain various levels of Grey which describe the amount of alpha per texel where black texels will be fully transparent and solid white texels will be fully opaque.

bMipMap should be set to TRUE if you want RenderIt 3D to automatically create mipmap levels for the texture.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Somewhere in the global data area
#define H_BACKDROP              0
#define H_CITYBLOCK             1
#define H_TRAFFICSIGNAL  2
#define H_STOPSIGN              3
#define H_BUILDING              4
#define H_MONSTERTRUCK   5
#define H_TRUCKWHEEL     6
#define H_TREE           7
#define H_TREE           8
#define H_PERSON         9
#define NUM_MODELS              10
char lpModels[NUM_MODELS][_MAX_PATH] = {
   "Backdrop.r3d",
   "CityBlock.r3d",
   "TrafficSignal.r3d",
   "StopSign.r3d",
   "Building.r3d",
   "MonsterTruck.r3d",
   "TruckWheel.r3d",
   "Tree.r3d",
   "Tree2.r3d",
   "Steve.r3d",
};
long hModel[NUM_MODELS];
float MatrixModel[NUM_MODELS];
#define NUM_MATERIALS    1
long hMaterial[NUM_MATERIALS];
// Somewhere in a LoadLevel type of function:
int LoopVar;
for (LoopVar=0; LoopVar<NUM_MODELS; LoopVar++)
{
if ( r3d_ListAdd3DModel( lpModels[LoopVar], FALSE) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), lpModels[LoopVar], MB_OK );
return FALSE;
}
r3d_MakeIdentityMatrix( MatrixModel[LoopVar] );
}
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F,
0.0F, TRUE, "Texture.bmp", TRUE, "AlphaTexture.bmp", TRUE );
if ( r3d_ListLoad( &hModel[0], &hMaterial[0] ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "r3d_ListLoad", MB_OK );
return FALSE;
}
```

See Also

r3d_ListAdd3DModel, r3d_ListLoad, r3d_ListUnload

r3d_ListLoad

The r3d_ListLoad function loads all 3D models and materials added by r3d_ListAdd3DModel and/or r3d_ListAddMaterial. The r3d_ListLoad function should only be called once after all models and materials have been added.

```
BOOL r3d_ListLoad(
      long *hModels,
      long *hMaterials);
```

Parameters

hModels should be the address of an array of type long that will be filled with the model handles added by r3d_ListAdd3DModel. If you have not added any models, you should set this parameter to NULL.

hMaterials should be the address of an array of type long that will be filled with the material handles added by r3d_ListAddMaterial. If you have not added any materials, you should set this parameter to NULL.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Somewhere in the global data area
#define H_BACKDROP              0
#define H_CITYBLOCK             1
#define H_TRAFFICSIGNAL   2
#define H_STOPSIGN              3
#define H_BUILDING              4
#define H_MONSTERTRUCK    5
#define H_TRUCKWHEEL      6
#define H_TREE            7
#define H_TREE            8
#define H_PERSON          9
#define NUM_MODELS              10
char lpModels[NUM_MODELS][_MAX_PATH] = {
   "Backdrop.r3d",
   "CityBlock.r3d",
   "TrafficSignal.r3d",
   "StopSign.r3d",
   "Building.r3d",
   "MonsterTruck.r3d",
   "TruckWheel.r3d",
   "Tree.r3d",
   "Tree2.r3d",
   "Steve.r3d",
};
long hModel[NUM_MODELS];
```

```
float MatrixModel[NUM_MODELS];
#define NUM_MATERIALS    1
long hMaterial[NUM_MATERIALS];
// Somewhere in a LoadLevel type of function:
int LoopVar;
for (LoopVar=0; LoopVar<NUM_MODELS; LoopVar++)
{
if ( r3d_ListAdd3DModel( lpModels[LoopVar], FALSE) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), lpModels[LoopVar], MB_OK );
return FALSE;
}
r3d_MakeIdentityMatrix( MatrixModel[LoopVar] );
}
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F,
0.0F, TRUE, "Texture.bmp", TRUE, "AlphaTexture.bmp", TRUE );
if ( r3d_ListLoad( &hModel[0], &hMaterial[0] ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "r3d_ListLoad", MB_OK );
return FALSE;
}
```

See Also

r3d_ListAdd3DModel, r3d_ListAddMaterial, r3d_ListUnload

r3d_ListUnload

The r3d_ListUnload function will release all resources and free all memory allocated with a previous call to r3d_ListLoad.

BOOL r3d_ListUnload(void);

Parameters

None.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Somewhere in the global data area
#define H_BACKDROP              0
#define H_CITYBLOCK             1
#define H_TRAFFICSIGNAL  2
#define H_STOPSIGN              3
#define H_BUILDING              4
#define H_MONSTERTRUCK   5
#define H_TRUCKWHEEL     6
#define H_TREE           7
#define H_TREE           8
#define H_PERSON         9
#define NUM_MODELS              10
char lpModels[NUM_MODELS][_MAX_PATH] = {
   "Backdrop.r3d",
   "CityBlock.r3d",
   "TrafficSignal.r3d",
   "StopSign.r3d",
   "Building.r3d",
   "MonsterTruck.r3d",
   "TruckWheel.r3d",
   "Tree.r3d",
   "Tree2.r3d",
   "Steve.r3d",
};
long hModel[NUM_MODELS];
float MatrixModel[NUM_MODELS];
#define NUM_MATERIALS    1
long hMaterial[NUM_MATERIALS];
// Somewhere in a LoadLevel type of function:
int LoopVar;
for (LoopVar=0; LoopVar<NUM_MODELS; LoopVar++)
{
if ( r3d_ListAdd3DModel( lpModels[LoopVar], FALSE) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), lpModels[LoopVar], MB_OK );
return FALSE;
}
r3d_MakeIdentityMatrix( MatrixModel[LoopVar] );
}
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F, 0.0F,
0.0F, TRUE, "Texture.bmp", TRUE, "AlphaTexture.bmp", TRUE );
if ( r3d_ListLoad( &hModel[0], &hMaterial[0] ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "r3d_ListLoad", MB_OK );
return FALSE;
}
r3d_ListUnload();
```

See Also

r3d_ListAdd3DModel, r3d_ListAddMaterial, r3d_ListLoad

r3d_MakeIdentityMatrix

The r3d_MakeIdentityMatrix function sets a RenderIt 3D matrix to it's identity.

```
void r3d_MakeIdentityMatrix(
        float *pf4x3Matrix);
```

Parameters

pf4x3Matrix is the address of a 12 element array of type float.

Return Value

None.

Source Code Sample

```
float MatrixView[12];
r3d_MakeIdentityMatrix( MatrixView );
```

See Also

r3d_ConcatenateMatrices, r3d_RotateMatrixByQuaternion

r3d_MessageBox

The r3d_MessageBox function calls the MessageBox function in the Windows API but first sets up your DirectX application to display the message box properly.

```
int r3d_MessageBox(
      LPCTSTR lpMessage,
      LPCTSTR lpTitle,
      UINT uType);
```

Parameters

lpMessage points to a null-terminated string containing the message to be displayed.

lpTitle points to a null-terminated string used for the dialog box title. If this parameter is NULL, the default title "Error" is used.

uType specifies a set of bit flags that determine the contents and behavior of the dialog box. The most common are presented here. Please refer to your appropriate documentation for more information.

Specify one of the following flags to indicate the buttons contained in the message box:

MB_ABORTRETRYIGNORE
The message box contains three push buttons: Abort, Retry, and Ignore.
MB_OK
The message box contains one push button: OK. This is the default.
MB_OKCANCEL
The message box contains two push buttons: OK and Cancel.
MB_RETRYCANCEL
The message box contains two push buttons: Retry and Cancel.
MB_YESNO
The message box contains two push buttons: Yes and No.
MB_YESNOCANCEL
The message box contains three push buttons: Yes, No, and Cancel.

Specify one of the following flags to display an icon in the message box:

MB_ICONEXCLAMATION,
MB_ICONWARNING
An exclamation-point icon appears in the message box.
MB_ICONINFORMATION,
MB_ICONASTERISK
An icon consisting of a lowercase letter i in a circle appears in the message box.
MB_ICONQUESTION
A question-mark icon appears in the message box.
MB_ICONSTOP,
MB_ICONERROR,
MB_ICONHAND
A stop-sign icon appears in the message box.

Return Value

The return value is zero if there is not enough memory to create the message box.

If the function succeeds, the return value is one of the following menu-item values returned by the dialog box:

IDABORT
Abort button was selected.
IDCANCEL
Cancel button was selected.
IDIGNORE
Ignore button was selected.
IDNO
No button was selected.
IDOK
OK button was selected.
IDRETRY
Retry button was selected.
IDYES
Yes button was selected.

If a message box has a Cancel button, the function returns the IDCANCEL value if either the ESC key is pressed or the Cancel button is selected. If the message box has no Cancel button, pressing ESC has no effect.

Source Code Sample

```
if ( !r3d_2DBegin( NULL, NULL, NULL) )
r3d_MessageBox( r3d_GetLastErrorString(), "Error", MB_OK );
```

See Also

r3d_GetLastErrorString

r3d_PageFlip

The r3d_PageFlip function exchanges the back buffer surface with the front buffer surface. After the pages have been flipped, the z-buffer and back buffer can be cleared.

```
BOOL r3d_PageFlip(
      BOOL bClearZBuffer,
      BOOL bClearBackBuffer);
```

Parameters

bClearZBuffer should be set to TRUE if you want to clear the z-buffer surface.

bClearBackBuffer should be set to TRUE if you want to clear the back buffer surface. If you have rendered a scene that covers the entire screen area, you can set this parameter to FALSE and increase performance.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_PageFlip( TRUE, FALSE );
```

See Also

r3d_ViewportClear, r3d_SetBackgroundColor , r3d_SetRenderArea

r3d_PreLight3DModel

The r3d_PreLight3DModel function will light a 3D model based on RenderIt 3D's light location and color. This is useful for increasing the rendering performance of static models.

```
BOOL r3d_PreLight3DModel(
      long hModel);
```

Parameters

hModel is the handle of a model obtained from a previous call to r3d_ListLoad.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL] is a valid
// model handle.
//
r3d_PreLight3DModel( hModel[H_MODEL] );
```

See Also

r3d_Render3DModel, r3d_ListLoad, r3d_Attach3DModels

r3d_Render3DModel

The r3d_Render3DModel function renders a 3D model to the current render area on the back buffer surface. This function must be within an r3d_RenderBegin and r3d_RenderEnd function pair.

```
BOOL r3d_Render3DModel(
    long hModel,
    float *pf4x3ModelMatrix);
```

Parameters

hModel is the handle of the model obtained from a previous call to r3d_ListLoad.

pf4x3ModelMatrix is the address of a 12 element array of type float describing the model matrix transform.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL] is a valid
// model handle.
//
float MatrixModel[12],MatrixView[12];
r3d_MakeIdentityMatrix( MatrixView );
r3d_MakeIdentityMatrix( MatrixModel );
r3d_RenderBegin( MatrixView );
r3d_Render3DModel( hModel[H_MODEL], MatrixModel );
r3d_RenderEnd(NULL);
```

See Also

r3d_RenderBegin, r3d_RenderEnd

r3d_RenderBegin

The r3d_RenderBegin function starts a 3D render state. r3d_RenderBegin should be used in conjunction with r3d_RenderEnd. r3d_RenderBegin and r3d_RenderEnd should bracket all 3D rendering calls and should only be called once per frame per rendering area.

```
BOOL r3d_RenderBegin(
    float *pf4x3ViewMatrix);
```

Parameters

pf4x3ViewMatrix is the address of a 12 element array of type float describing the view matrix transform.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL0] and
// hModel[H_MODEL1] are valid model handles.
//
float MatrixModel[2][12],MatrixView[12];
r3d_MakeIdentityMatrix( MatrixView );
r3d_MakeIdentityMatrix( MatrixModel[0] );
r3d_MakeIdentityMatrix( MatrixModel[1] );
r3d_RenderBegin( MatrixView );
r3d_Render3DModel( hModel[H_MODEL0], MatrixModel[0] );
r3d_Render3DModel( hModel[H_MODEL1], MatrixModel[1] );
r3d_RenderEnd(NULL);
```

See Also

r3d_RenderEnd

r3d_RenderEnd

The r3d_RenderEnd function stops a 3D render state. r3d_RenderEnd should be used in conjunction with r3d_RenderBegin. r3d_RenderBegin and r3d_RenderEnd should bracket all 3D rendering calls and should only be called once per frame per rendering area.

```
BOOL r3d_RenderEnd(
      long *pTotalPolygonsRendered);
```

Parameters

pTotalPolygonsRendered is the address of a variable of type long that will be filled with the number of polygons that were passed to the renderer since the last call to r3d_RenderBegin. You may pass NULL if you do not require this information.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL0] and
// hModel[H_MODEL1] are valid model handles.
//
float MatrixModel[2][12],MatrixView[12];
r3d_MakeIdentityMatrix( MatrixView );
r3d_MakeIdentityMatrix( MatrixModel[0] );
r3d_MakeIdentityMatrix( MatrixModel[1] );
r3d_RenderBegin( MatrixView );
r3d_Render3DModel( hModel[H_MODEL0], MatrixModel[0] );
r3d_Render3DModel( hModel[H_MODEL1], MatrixModel[1] );
r3d_RenderEnd(NULL);
```

See Also

r3d_RenderBegin

r3d_RestoreSurfaces

The r3d_RestoreSurfaces function restores any lost front, back, and texture surfaces and automatically loads bitmaps back onto the texture surfaces if required.

BOOL r3d_RestoreSurfaces(void);

Parameters

None.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_RestoreSurfaces();
```

r3d_RotateMatrixByQuaternion

The r3d_RotateMatrixByQuaternion function will perform a rotation on a matrix by a quaternion describing the axis and angle of rotation. Creating a matrix with r3d_MakeIdentityMatrix and modifying it exclusively with r3d_RotateMatrixByQuaternion ensures that the matrix is always normalized and stable over time by avoiding rounding errors. r3d_RotateMatrixByQuaternion does not affect the last 3 elements of the passed matrix which are used for translation.

```
BOOL r3d_RotateMatrixByQuaternion(
    float *pf4x3Matrix,
    float fX,
    float fY,
    float fZ,
    float fRadianAngle);
```

Parameters

pf4x3Matrix is the base address to a 12 element array describing the matrix.

fX is the X component of the vector describing the axis of rotation.

fY is the Y component of the vector describing the axis of rotation.

fZ is the Z component of the vector describing the axis of rotation.

fRadianAngle is the angle of rotation around the fX,fY,fZ axis expressed in radians. To convert from a standard angle measurement to a radian, simply multiply the angle by 0.01745329251994.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
float MatrixView[12], MatrixAirplane[12], MatrixPilotHead[12];
r3d_MakeIdentityMatrix( MatrixAirplane );
r3d_MakeIdentityMatrix( MatrixPilotHead );
// Pilot is looking left out of the airplane window
r3d_RotateMatrixByQuaternion( MatrixPilotHead, 0,1,0, -90.0F*0.01745329251994F );
// Aircraft is banked at 45 degrees
r3d_RotateMatrixByQuaternion( MatrixAirplane, 0,0,1, 45.0F*0.01745329251994F );
// Create the appropriate view matrix based on the orientation of the pilots head and
the orientation of the aircraft.
r3d_ConcatenateMatrices( MatrixView, MatrixAirplane, MatrixPilotHead );
r3d_RenderBegin( MatrixView );
// . . .
```

See Also

r3d_ConcatenateMatrices, r3d_MakeIdentityMatrix

r3d_Set2DClipRegion

The r3d_Set2DClipRegion specifies the clipping region for all primitives rendered with the r3d_2D… series of functions. Any 2D primitives outside this area will not be rendered. By default, the clipping region is set to the extents of the back buffer.

```
void r3d_Set2DClipRegion(
        int MinX,
        int MaxX,
        int MinY,
        int MaxY);
```

Parameters

MinX,MaxX describes the minimum and maximum extents of the clipping region on the X screen axis.

MinY,MaxY describes the minimum and maximum extents of the clipping region on the Y screen axis.

Return Value

None.

Source Code Sample

```
int hBitmap,Width,Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL );
r3d_Set2DClipRegion( 0, Width/2, 0, Height/2 );
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
r3d_2DBlit( hBitmap, 0, r3d_SurfaceGetWidth(hBitmap)-1, 0,
r3d_SurfaceGetHeight(hBitmap)-1, 0, Width-1, 0, Height-1, FALSE );
r3d_SurfaceDestroy(hBitmap);
```

r3d_SetBackgroundColor

The r3d_SetBackgroundColor function sets the color used for clearing the back buffer with the r3d_PageFlip, r3d_ViewportClear, and r3d_SetRenderArea functions.

```
BOOL r3d_SetBackgroundColor(
        float r,
        float g,
        float b);
```

Parameters

r,g,b describe the red, green and blue color components and should range from 0.0 to 1.0.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_SetBackgroundColor( 0.7F,0.7F,0.7F );
```

See Also

r3d_ViewportClear, r3d_PageFlip, r3d_SetRenderArea

r3d_SetCurrentTick

The r3d_SetCurrentTick function will let you set the RenderIt 3D timer variable to a specified value. The RenderIt 3D timer runs at 1024 hertz.

```
BOOL r3d_SetCurrentTick(
        unsigned long tick);
```

Parameters

tick is the value you wish to set for the timer.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_SetCurrentTick(1024);
```

See Also

r3d_GetCurrentTick

r3d_SetFilteringAndMipMapOptions

The r3d_SetFilteringAndMipMapOptions function allows you to set texture filtering states and mipmapping states provided that the Direct3D device supports it. In order for mipmapping to work, the textures must created with mipmaps enabled.

```
BOOL r3d_SetFilteringAndMipMapOptions(
        int Method);
```

Parameters

Method should be set to one of the following values:

Method = 0:
Disable mipmapping
Disable texture filtering

Method = 1:
Disable mipmapping
Enable texture filtering

Method = 2:
Enable mipmapping
Disable texture filtering

Method = 3:
Enable mipmapping
Enable texture filtering

Method = 4:
Enable mipmapping with filtering
Disable texture filtering

Method = 5: (Trilinear filtering)
Enable mipmapping with filtering
Enable texture filtering

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_SetFilteringAndMipMapOptions(5);
```

See Also

r3d_ListAdd3DModel, r3d_ListAddMaterial

r3d_SetFogAttributes

The r3d_SetFogAttributes function allows you to set the Direct3D vertex fog states. By default, the fog color is set to black and extends to the far clip plane as specified in r3d_Initialize.

```
BOOL r3d_SetFogAttributes(
    float fFogZStart,
    float fFogZEnd,
    float r,
    float g,
    float b);
```

Parameters

fFogZStart,fFogZEnd indicate the start and end z-axis extents of the fog. Objects within this range appear with fog. Objects that are at or beyond the distance indicated by fFogZEnd are still rendered, but are rendered with maximum fog.

r,g,b indicate the red, green, and blue color components of the fog color.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_SetFogAttributes( 0.0F, 100.0F, 0.7F, 0.7F, 0.7F );
r3d_EnableFog( TRUE );
```

See Also

r3d_Initialize, r3d_EnableFog, r3d_EnableSpecular, r3d_SetFilteringAndMipMapOptions

r3d_SetLightLocationColor

The r3d_SetLightLocationColor function allows you to modify the default Direct3D directional light created by RenderIt 3D. The r3d_PreLight3DModel function uses the location and color of the light for pre-lighting 3D models.

```
BOOL r3d_SetLightLocationColor(
    float fLightX,
    float fLightY,
    float fLightZ,
    float fLightRed,
    float fLightGreen,
    float fLightBlue);
```

Parameters

fLightX,fLightY,fLightZ is vector from 0,0,0 to the location of the light. The r3d_SetLightLocationColor function will automatically convert this is an appropriate unit vector for Direct3D.

fLightRed,fLightGreen,fLightBlue indicate the red, green and blue color components emitted by the light and should range from 0.0 to 1.0.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Place the sun at 12 noon emitting a sun like color
r3d_SetLightLocationColor( 0.0F, 1.0F, 0.0F, 1.0F, 0.95F, 0.9F );
```

See Also

r3d_PreLight3DModel, r3d_LightRemoveDefault, r3d_LightSetAmbient

r3d_SetMaterial

The r3d_SetMaterial function sets a material created by r3d_ListAddMaterial and r3d_ListLoad. The r3d_SetMaterial function should be called prior to rendering polygon data and can only be called within an r3d_RenderBegin and r3d_RenderEnd function pair.

```
BOOL r3d_SetMaterial(
      int hMaterial);
```

Parameters

hMaterial is the handle of a material obtained from a previous call to r3d_ListLoad or r3d_Get3DModelMaterialHandle.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[] and
// hMaterial[] are valid handles
//
float MatrixView[12], MatrixModel[12], MatrixPolygonData[2][12];
r3d_MakeIdentityMatrix( MatrixView );
r3d_MakeIdentityMatrix( MatrixModel );
r3d_MakeIdentityMatrix( MatrixPolygonData[0] );
r3d_MakeIdentityMatrix( MatrixPolygonData[1] );
r3d_RenderBegin( MatrixView );
r3d_SetMaterial( hMaterial[0] );
r3d_SetMatrix( MatrixPolygonData[0] );
// Render polygons belonging to this material and matrix here
r3d_SetMaterial( hMaterial[1] );
r3d_SetMatrix( MatrixPolygonData[1] );
// Render polygons belonging to this material and matrix here
r3d_Render3DModel( hModel[0], MatrixModel );
r3d_RenderEnd( NULL );
```

See Also

r3d_ListAddMaterial , r3d_ListLoad, r3d_RenderBegin , r3d_RenderEnd

r3d_SetMatrix

The r3d_SetMatrix function sets the Direct3D world transform to the specified matrix. The r3d_SetMatrix function should be called prior to rendering polygon data and can only be called within an r3d_RenderBegin and r3d_RenderEnd function pair.

```
BOOL r3d_SetMatrix(
      float *pf4x3Matrix);
```

Parameters

pf4x3Matrix is the address of a 12 element array of type float describing the matrix transform.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[] and
// hMaterial[] are valid handles
//
float MatrixView[12], MatrixModel[12], MatrixPolygonData[2][12];
r3d_MakeIdentityMatrix( MatrixView );
r3d_MakeIdentityMatrix( MatrixModel );
r3d_MakeIdentityMatrix( MatrixPolygonData[0] );
r3d_MakeIdentityMatrix( MatrixPolygonData[1] );
r3d_RenderBegin( MatrixView );
r3d_SetMaterial( hMaterial[0] );
r3d_SetMatrix( MatrixPolygonData[0] );
// Render polygons belonging to this material and matrix here
r3d_SetMaterial( hMaterial[1] );
r3d_SetMatrix( MatrixPolygonData[1] );
// Render polygons belonging to this material and matrix here
r3d_Render3DModel( hModel[0], MatrixModel );
r3d_RenderEnd( NULL );
```

See Also

r3d_RenderBegin , r3d_RenderEnd

r3d_SetRenderArea

The r3d_SetRenderArea function sets an area on the back buffer where 3D primitives will be rendered.

```
BOOL r3d_SetRenderArea(
      BOOL bClearZBuffer,
      BOOL bClearBackBuffer,
      int MinX,
      int MaxX,
      int MinY,
      int MaxY);
```

Parameters

bClearZBuffer should be set to TRUE if you want to clear the z-buffer surface.

bClearBackBuffer should be set to TRUE if you want to clear the back buffer surface. If you have rendered a scene that covers the entire render area, you can set this parameter to FALSE and increase performance.

MinX,MaxX describes the minimum and maximum extents of the clipping region on the X screen axis.

MinY,MaxY describes the minimum and maximum extents of the clipping region on the Y screen axis.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int gWidth, gHeight;
r3d_GetDisplayMode( &gWidth, &gHeight );
r3d_SetRenderArea( TRUE, TRUE, 0, (gWidth/2)-1, 0, (gHeight/2)-1 );
```

See Also

r3d_PageFlip, r3d_ViewportClear

r3d_SetViewMatrix

The r3d_SetViewMatrix function sets the Direct3D view transform to the specified matrix. The r3d_SetViewMatrix function can only be called within an r3d_RenderBegin and r3d_RenderEnd function pair.

```
BOOL r3d_SetViewMatrix(
    float *pf4x3Matrix);
```

Parameters

pf4x3Matrix is the address of a 12 element array of type float describing the view matrix transform.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[] and
// hMaterial[] are valid handles
//
float MatrixView[12], MatrixModel[12], MatrixPolygonData[2][12];
r3d_MakeIdentityMatrix( MatrixView );
r3d_MakeIdentityMatrix( MatrixModel );
r3d_MakeIdentityMatrix( MatrixPolygonData[0] );
r3d_MakeIdentityMatrix( MatrixPolygonData[1] );
r3d_RenderBegin( MatrixView );
r3d_Render3DModel( hModel[0], MatrixModel );
// We need to alter the view matrix for some reason
r3d_SetViewMatrix( MatrixView );
r3d_SetMaterial( hMaterial[0] );
r3d_SetMatrix( MatrixPolygonData[0] );
// Render polygons belonging to this material and matrix here
r3d_SetMaterial( hMaterial[1] );
r3d_SetMatrix( MatrixPolygonData[1] );
// Render polygons belonging to this material and matrix here
r3d_RenderEnd( NULL );
```

See Also

r3d_RenderBegin , r3d_RenderEnd, r3d_SetMatrix

r3d_SurfaceBlit

The r3d_SurfaceBlit function will transfer a specified region from one virtual surface to another. Clipping and stretching will be performed if required.

```
BOOL r3d_SurfaceBlit(
    int hSrc,
    int hDst,
    int SrcMinx,
    int SrcMaxx,
    int SrcMiny,
    int SrcMaxy,
    int DstMinx,
    int DstMaxx,
    int DstMiny,
    int DstMaxy,
    BOOL bBlendAlphaChannel);
```

Parameters

hSrc is the handle of the source surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

hDst is the handle of the destination surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

SrcMinx,SrcMaxx,SrcMiny,SrcMaxy describes the region on the source surface.

DstMinx,DstMaxx,DstMiny,DstMaxy describes the region on the destination surface.

bBlendAlphaChannel should be set to TRUE to blend the source surface with the destination surface using the alpha channel information contained in the source surface.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmapSrc, hBitmapDst, BackBufferWidth, BackBufferHeight, SrcWidth, SrcHeight;
r3d_SurfaceCreateFromBMP( &hBitmapSrc, "Bitmap.bmp" );
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
SrcWidth = r3d_SurfaceGetWidth(hBitmapSrc);
SrcHeight = r3d_SurfaceGetHeight(hBitmapSrc);
r3d_SurfaceCreate( &hBitmapDst, BackBufferWidth, BackBufferHeight );
r3d_SurfaceBlit( hBitmapSrc, hBitmapDst, 0, SrcWidth-1, 0, SrcHeight-1, 0,
BackBufferWidth-1, 0, BackBufferHeight-1, FALSE );
```

```
r3d_2DBlit( hBitmapDst, 0, BackBufferWidth-1, 0, BackBufferHeight-1, 0,
BackBufferWidth-1, 0, BackBufferHeight-1, FALSE );
r3d_SurfaceDestroy( hBitmapDst );
r3d_SurfaceDestroy( hBitmapSrc );
```

See Also

r3d_2DBlit , r3d_SurfaceStretchSmooth

r3d_SurfaceColorGetA

The r3d_SurfaceColorGetA macro obtains the alpha color component from a virtual surface color. To access a pixel color on a virtual surface, see r3d_SurfaceGet.

unsigned char r3d_SurfaceColorGetA(
    unsigned long dwColor);

Parameters

dwColor is the color of a pixel on a virtual surface.

Return Value

The alpha color component of the specified color in the range of 0 to 255.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceColorGetB, r3d_SurfaceColorGetG, r3d_SurfaceColorGetR, r3d_SurfaceColorMakeRGBA, r3d_SurfaceGet

r3d_SurfaceColorGetB

The r3d_SurfaceColorGetB macro obtains the blue color component from a virtual surface color. To access a pixel color on a virtual surface, see r3d_SurfaceGet.

```
unsigned char r3d_SurfaceColorGetB(
    unsigned long dwColor);
```

Parameters

dwColor is the color of a pixel on a virtual surface.

Return Value

The blue color component of the specified color in the range of 0 to 255.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceColorGetB, r3d_SurfaceColorGetG, r3d_SurfaceColorGetR, r3d_SurfaceColorMakeRGBA, r3d_SurfaceGet

r3d_SurfaceColorGetG

The r3d_SurfaceColorGetG macro obtains the green color component from a virtual surface color. To access a pixel color on a virtual surface, see r3d_SurfaceGet.

```
unsigned char r3d_SurfaceColorGetG(
    unsigned long dwColor);
```

Parameters

dwColor is the color of a pixel on a virtual surface.

Return Value

The green color component of the specified color in the range of 0 to 255.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceColorGetA, r3d_SurfaceColorGetB, r3d_SurfaceColorGetR, r3d_SurfaceColorMakeRGBA, r3d_SurfaceGet

r3d_SurfaceColorGetR

The r3d_SurfaceColorGetR macro obtains the red color component from a virtual surface color. To access a pixel color on a virtual surface, see r3d_SurfaceGet.

```
unsigned char r3d_SurfaceColorGetR(
    unsigned long dwColor);
```

Parameters

dwColor is the color of a pixel on a virtual surface.

Return Value

The red color component of the specified color in the range of 0 to 255.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceColorGetA, r3d_SurfaceColorGetB, r3d_SurfaceColorGetG, r3d_SurfaceColorMakeRGBA, r3d_SurfaceGet

r3d_SurfaceColorMakeRGBA

The r3d_SurfaceColorMakeRGBA macro creates a pixel color for a virtual surface based on red, green, blue, and alpha color components.

```
unsigned long r3d_SurfaceColorMakeRGBA(
    unsigned char r,
    unsigned char g,
    unsigned char b,
    unsigned char a);
```

Parameters

r,g,b,a is the red, green, blue and alpha color components describing the pixel color to create.

Return Value

Returns a pixel color compatible with a virtual surface.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceColorGetA, r3d_SurfaceColorGetB, r3d_SurfaceColorGetG, r3d_SurfaceColorGetR, r3d_SurfaceGet

r3d_SurfaceCreate

The r3d_SurfaceCreate function creates a virtual surface based on a specified width and height.

```
BOOL r3d_SurfaceCreate(
        int *handle,
        int Width,
        int Height);
```

Parameters

handle is the address of an integer that will be assigned a handle for the virtual surface being created.

Width,Height describes the width and height of the virtual surface.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmapSrc, hBitmapDst, BackBufferWidth, BackBufferHeight, SrcWidth, SrcHeight;
r3d_SurfaceCreateFromBMP( &hBitmapSrc, "Bitmap.bmp" );
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
SrcWidth = r3d_SurfaceGetWidth(hBitmapSrc);
SrcHeight = r3d_SurfaceGetHeight(hBitmapSrc);
r3d_SurfaceCreate( &hBitmapDst, BackBufferWidth, BackBufferHeight );
r3d_SurfaceBlit( hBitmapSrc, hBitmapDst, 0, SrcWidth-1, 0, SrcHeight-1, 0,
BackBufferWidth-1, 0, BackBufferHeight-1, FALSE );
r3d_2DBlit( hBitmapDst, 0, BackBufferWidth-1, 0, BackBufferHeight-1, 0,
BackBufferWidth-1, 0, BackBufferHeight-1, FALSE );
r3d_SurfaceDestroy( hBitmapDst );
r3d_SurfaceDestroy( hBitmapSrc );
```

See Also

r3d_SurfaceCreateFromBMP, r3d_SurfaceCreateFromBackBuffer, r3d_SurfaceDestroy

r3d_SurfaceCreateFromBackBuffer

The r3d_SurfaceCreateFromBackBuffer function creates a virtual surface based on the size of the back buffer and copies the back buffer contents to the virtual surface.

```
BOOL r3d_SurfaceCreateFromBackBuffer(
      int *handle);
```

Parameters

handle is the address of an integer that will be assigned a handle for the virtual surface being created.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Create a screen shot of the back buffer
r3d_SurfaceCreateFromBackBuffer( &hBitmap );
r3d_SurfaceExportToBMP( hBitmap, "Captured.bmp" );
r3d_SurfaceDestroy( hBitmap );
```

See Also

r3d_SurfaceCreate, r3d_SurfaceCreateFromBMP, r3d_SurfaceDestroy

r3d_SurfaceCreateFromBMP

The r3d_SurfaceCreateFromBMP function creates a virtual surface and copies the bitmap file to the surface. The bitmap must be an uncompressed 256 color or 24-bit color windows BMP file.

```
BOOL r3d_SurfaceCreateFromBMP(
    int *handle,
    char *BitmapFilename);
```

Parameters

handle is the address of an integer that will be assigned a handle for the virtual surface being created.

BitmapFilename is the pathname of the bitmap file.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap, Width, Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
r3d_2DBlit( hBitmap, 0, r3d_SurfaceGetWidth(hBitmap)-1, 0,
r3d_SurfaceGetHeight(hBitmap)-1, 0, Width-1, 0, Height-1, FALSE );
r3d_SurfaceDestroy( hBitmap );
```

See Also

r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, r3d_SurfaceDestroy

r3d_SurfaceDestroy

The r3d_SurfaceDestroy function destroys a specified virtual surface by freeing the memory allocated for it when the surface was created.

BOOL r3d_SurfaceDestroy(
    int handle);

Parameters

handle is the handle of the virtual surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap, Width, Height;
r3d_GetDisplayMode( &Width, &Height, NULL, NULL, NULL, NULL, NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
r3d_2DBlit( hBitmap, 0, r3d_SurfaceGetWidth(hBitmap)-1, 0,
r3d_SurfaceGetHeight(hBitmap)-1, 0, Width-1, 0, Height-1, FALSE );
r3d_SurfaceDestory( hBitmap );
```

See Also

r3d_SurfaceCreate, r3d_SurfaceCreateFromBMP, r3d_SurfaceCreateFromBackBuffer, r3d_SurfaceDestroyAll

r3d_SurfaceDestroyAll

The r3d_SurfaceDestroyAll function frees the memory allocated for all virtual surfaces created by previous calls to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, and r3d_SurfaceCreateFromBMP.

BOOL r3d_SurfaceDestroyAll(void);

Parameters

None.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmapSrc, hBitmapDst, BackBufferWidth, BackBufferHeight, SrcWidth, SrcHeight;
r3d_SurfaceCreateFromBMP( &hBitmapSrc, "Bitmap.bmp" );
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
SrcWidth = r3d_SurfaceGetWidth(hBitmapSrc);
SrcHeight = r3d_SurfaceGetHeight(hBitmapSrc);
r3d_SurfaceCreate( &hBitmapDst, BackBufferWidth, BackBufferHeight );
r3d_SurfaceBlit( hBitmapSrc, hBitmapDst, 0, SrcWidth-1, 0, SrcHeight-1, 0,
BackBufferWidth-1, 0, BackBufferHeight-1, FALSE );
r3d_2DBlit( hBitmapDst, 0, BackBufferWidth-1, 0, BackBufferHeight-1, 0,
BackBufferWidth-1, 0, BackBufferHeight-1, FALSE );
r3d_SurfaceDestroyAll();
```

See Also

r3d_SurfaceDestroy

r3d_SurfaceExportToBMP

The r3d_SurfaceExportToBMP function exports a surface to a 24-bit color windows BMP file. This is useful for creating screen shots.

```
BOOL r3d_SurfaceExportToBMP(
        int handle,
        char *pFilename);
```

Parameters

handle is the handle of the virtual surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

pFilename is a pointer to a char string containing the filename.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Create a screen shot of the back buffer
r3d_SurfaceCreateFromBackBuffer( &hBitmap );
r3d_SurfaceExportToBMP( hBitmap, "Captured.bmp" );
r3d_SurfaceDestroy( hBitmap );
```

See Also

r3d_SurfaceCreateFromBackBuffer

r3d_SurfaceGet

The r3d_SurfaceGet function returns a pointer to the first pixel on a specified virtual surface. The memory is linear and starts at the top, left pixel. The "pitch" is equal to the width. The width can be obtained from r3d_SurfaceGetWidth. To access a pixel at a particular x,y location on a virtual surface, please refer to the source code example below.

unsigned long *r3d_SurfaceGet(
int handle);

Parameters

handle is the handle of the virtual surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

Return Value

Returns a pointer of type unsigned long which points to the base address of the virtual surface memory.

If you typecast the return value to type unsigned char you can obtain the color components from the virtual surface without using the r3d_SurfaceColorGetA, r3d_SurfaceColorGetB, r3d_SurfaceColorGetG, and r3d_SurfaceColorGetR macros. You can also write pixel colors to the virtual surface without using r3d_SurfaceColorMakeRGBA.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceColorGetA, r3d_SurfaceColorGetB, r3d_SurfaceColorGetG, r3d_SurfaceColorGetR, r3d_SurfaceColorMakeRGBA.

r3d_SurfaceGetHeight

The r3d_SurfaceGetHeight function returns the height of a specified virtual surface.

```
int r3d_SurfaceGetHeight(
      int handle);
```

Parameters

handle is the handle of the virtual surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

Return Value

Returns the height of the virtual surface as type int.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceGetWidth

r3d_SurfaceGetWidth

The r3d_SurfaceGetWidth function returns the width of a specified virtual surface. The width is also equal to the pitch of each "scanline".

```
int r3d_SurfaceGetWidth(
      int handle);
```

Parameters

handle is the handle of the virtual surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

Return Value

Returns the width of the virtual surface as type int.

Source Code Sample

```
int x, y, hBitmap, BitmapWidth, BitmapHeight;
unsigned long *pSurface;
unsigned char r,g,b,a;
r3d_SurfaceCreateFromBMP( &hBitmap, "Bitmap.bmp" );
BitmapWidth = r3d_SurfaceGetWidth( hBitmap );
BitmapHeight = r3d_SurfaceGetHeight( hBitmap );
pSurface = r3d_SurfaceGet( hBitmap );
x = (BitmapWidth/2) - 1;
y = (BitmapHeight/2) - 1;
a = r3d_SurfaceColorGetA( pSurface[(BitmapWidth * y) + x] );
b = r3d_SurfaceColorGetB( pSurface[(BitmapWidth * y) + x] );
g = r3d_SurfaceColorGetG( pSurface[(BitmapWidth * y) + x] );
r = r3d_SurfaceColorGetR( pSurface[(BitmapWidth * y) + x] );
// Set pixel color to half intensity
r /= 2;
g /= 2;
b /= 2;
pSurface[(BitmapWidth * y) + x] = r3d_SurfaceColorMakeRGBA( r, g, b, a);
```

See Also

r3d_SurfaceGetHeight

r3d_SurfaceSetAlphaFromBMP

The r3d_SurfaceSetAlphaFromBMP function sets the alpha channel pixels on a virtual surface from a bitmap file. The bitmap must be an uncompressed 256 color or 24-bit color windows BMP file. The bitmap must be the same size as the surface. The whiter the pixel in the bitmap image, the more solid the resulting alpha channel pixel will be. Solid black pixels indicate full transparency. The r3d_SurfaceSetAlphaFromBMP function will completely overwrite the alpha channel and disregard any previous calls to r3d_SurfaceSetAlphaFromBMP as well as r3d_SurfaceSetColorKey.

```
BOOL r3d_SurfaceSetAlphaFromBMP(
    int handle,
    char *BitmapFilename);
```

Parameters

handle is the handle of the virtual surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

BitmapFilename is the pathname of the bitmap file.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap, BackBufferWidth, BackBufferHeight;
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "SmokeImage.bmp" );
r3d_SurfaceSetAlphaFromBMP( hBitmap, "SmokeAlpha.bmp" );
r3d_2DBlit( hBitmap, 0, r3d_SurfaceGetWidth(hBitmap)-1, 0,
r3d_SurfaceGetHeight(hBitmap)-1, 0, BackBufferWidth-1, 0, BackBufferHeight-1, TRUE );
r3d_SurfaceDestroy( hBitmap );
```

See Also

r3d_SurfaceSetColorKey, r3d_SurfaceCreateFromBMP

r3d_SurfaceSetColorKey

The r3d_SurfaceSetColorKey function sets all alpha channel pixels on a virtual surface to fully transparent when the specified color components match the pixel, otherwise it sets it to fully opaque. The r3d_SurfaceSetColorKey function will completely overwrite the alpha channel and disregard any previous calls to r3d_SurfaceSetAlphaFromBMP as well as r3d_SurfaceSetColorKey.

```
BOOL r3d_SurfaceSetColorKey(
    int handle,
    unsigned char Red,
    unsigned char Green,
    unsigned char Blue);
```

Parameters

handle is the handle of the virtual surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

Red,Green,Blue are the blue, green, and red color components of the color key.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmap, BackBufferWidth, BackBufferHeight;
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
r3d_SurfaceCreateFromBMP( &hBitmap, "SmokeImage.bmp" );
// Set all solid blue pixels in the bitmap to be transparent
r3d_SurfaceSetColorKey( hBitmap, 0, 0, 255 );
r3d_2DBlit( hBitmap, 0, r3d_SurfaceGetWidth(hBitmap)-1, 0,
r3d_SurfaceGetHeight(hBitmap)-1, 0, BackBufferWidth-1, 0, BackBufferHeight-1, TRUE );
r3d_SurfaceDestroy( hBitmap );
```

See Also

r3d_SurfaceSetAlphaFromBMP, r3d_SurfaceCreateFromBMP

r3d_SurfaceStretchSmooth

The r3d_SurfaceStretchSmooth function will stretch one virtual surface onto another and provides anti-aliasing. If you don't want the resulting image to be anti-aliased, you can use r3d_SurfaceBlit. The r3d_SurfaceStretchSmooth function uses the width and height of the source and destination surfaces to determine the scale size.

```
BOOL r3d_SurfaceStretchSmooth(
        int hSrc,
        int hDst);
```

Parameters

hSrc is the handle of the source surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

hDst is the handle of the destination surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
int hBitmapSrc, hBitmapDst, BackBufferWidth, BackBufferHeight, SrcWidth, SrcHeight;
r3d_SurfaceCreateFromBMP( &hBitmapSrc, "Bitmap.bmp" );
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
SrcWidth = r3d_SurfaceGetWidth(hBitmapSrc);
SrcHeight = r3d_SurfaceGetHeight(hBitmapSrc);
r3d_SurfaceCreate( &hBitmapDst, BackBufferWidth, BackBufferHeight );
r3d_SurfaceStretchSmooth( hBitmapSrc, hBitmapDst );
r3d_2DBlit( hBitmapDst, 0, BackBufferWidth-1, 0, BackBufferHeight-1, 0,
BackBufferWidth-1, 0, BackBufferHeight-1, FALSE );
r3d_SurfaceDestroy( hBitmapDst );
r3d_SurfaceDestroy( hBitmapSrc );
```

See Also

r3d_2DBlit, r3d_SurfaceBlit

r3d_TextureAnimateScroll

The r3d_TextureAnimateScroll function is useful for smoothly scrolling a texture across a group of polygons. It modifies the texture's texel coordinates on the polygons containing the texture. The greater the number of polygons that need to be modified, the greater the performance decrease.

```
BOOL r3d_TextureAnimateScroll(
        int hModel,
        int ModelMaterialOffset,
        float fOffset_tu,
        float fOffset_tv);
```

Parameters

hModel is a model handle obtained from a previous call to r3d_ListLoad.

ModelMaterialOffset is the offset of a material within the model. This parameter can not equal or exceed the number of materials supplied by r3d_Get3DModelMaterialCount. All vertex and polygon data are grouped by material. If you passed an offset of zero for this parameter, you would be indexing the first material of the model as well as their corresponding polygons and vertices. To find out which material offset corresponds to a particular material or texture of a particular RenderIt 3D model (*.R3D), load the RenderIt 3D model into Materialize 3D! and inspect the material numbers shown in the "Edit Materials" dialog box. The material numbers are equal to the parameter that should be passed to this function. If you save the model within Materialize 3D! after inspection, the material numbers may be re-assigned possibly making them invalid.

fOffset_tu,fOffset_tv describe the offset amount to apply to the texture coordinates.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// This example assumes that hModel[H_MODEL] is a valid
// model handle.
//
// Scroll the texture contained in the last material of the model
//
int MaterialCount;
r3d_Get3DModelMaterialCount( hModel[H_MODEL], &MaterialCount );
r3d_TextureAnimateScroll( hModel[H_MODEL], MaterialCount-1, 0.01F, 0.01F );
```

See Also

r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceBlit

The r3d_TextureAnimateSurfaceBlit function will transfer a virtual surface to a texture surface. If the texture surface contains an alpha channel, then r3d_TextureAnimateSurfaceBlit will copy the alpha channel information directly from the virtual surface to the texture surface. Clipping and stretching will be performed if required.

```
BOOL r3d_TextureAnimateSurfaceBlit(
        int hSrc,
        int WidthHeight,
        int TextureFormat,
        unsigned char *SurfacePointer,
        int SurfacePitch,
        int SrcMinx,
        int SrcMaxx,
        int SrcMiny,
        int SrcMaxy,
        int DstMinx,
        int DstMaxx,
        int DstMiny,
        int DstMaxy);
```

Parameters

hSrc is the handle of the source surface previously obtained from a call to r3d_SurfaceCreate, r3d_SurfaceCreateFromBackBuffer, or r3d_SurfaceCreateFromBMP.

WidthHeight,TextureFormat,SurfacePointer,SurfacePitch is obtained from a previous call to r3d_TextureAnimateSurfaceLock.

SrcMinx,SrcMaxx,SrcMiny,SrcMaxy describes the source region of the virtual surface to be copied.

DstMinx,DstMaxx,DstMiny,DstMaxy describes the destination region on the texture surface.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Copy the back buffer to a texture surface
// with anti-aliasing.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
```

```
int hBitmapSrc, hBitmapDst, BackBufferWidth, BackBufferHeight;
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
r3d_SurfaceCreateFromBackBuffer( &hBitmapSrc )
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
r3d_SurfaceCreate( &hBitmapDst, WidthHeight, WidthHeight );
r3d_SurfaceStretchSmooth( hBitmapSrc, hBitmapDst );
r3d_TextureAnimateSurfaceBlit( hBitmapDst, WidthHeight, TextureFormat,
SurfacePointer, SurfacePitch, 0, r3d_SurfaceGetWidth(hBitmapDst)-1, 0,
r3d_SurfaceGetHeight(hBitmapDst)-1, 0, WidthHeight-1, 0, WidthHeight-1 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
r3d_SurfaceDestroyAll();
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceLock

The r3d_TextureAnimateSurfaceLock function will lock the texture surface of a material for subsequent calls to r3d_TextureAnimateSurfaceBlit and the r3d_TextureAnimateSurfaceTexel series of functions. The r3d_TextureAnimateSurfaceLock function expects the specified material to contain a texture. Any texture surface locked by r3d_TextureAnimateSurfaceLock must be unlocked after the surface has been accessed using r3d_TextureAnimateSurfaceUnlock.

```
BOOL r3d_TextureAnimateSurfaceLock(
    int hMaterial,
    int *WidthHeight,
    int *TextureFormat,
    unsigned char **SurfacePointer,
    int *SurfacePitch);
```

Parameters

hMaterial is the handle of a material obtained from a previous call to r3d_ListLoad or r3d_Get3DModelMaterialHandle.

WidthHeight is the address of an integer that will be assigned the width and height of the texture surface.

TextureFormat is the address of an integer that will be assigned the RenderIt 3D texture format code of the texture surface.

SurfacePointer is the address of a pointer that will point to the DirectDraw surface of the texture.

SurfacePitch is the address of an integer that will be assigned the pitch of the texture surface.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Sets the center texel of a texture surface to red.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
if ( TextureFormat == 0 )
```

```
   r3d_TextureAnimateSurfaceTexel0 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 1 )
   r3d_TextureAnimateSurfaceTexel1 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 2 )
   r3d_TextureAnimateSurfaceTexel2 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 3 )
   r3d_TextureAnimateSurfaceTexel3 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 4 )
   r3d_TextureAnimateSurfaceTexel4 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 5 )
   r3d_TextureAnimateSurfaceTexel5 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceTexel0

The r3d_TextureAnimateSurfaceTexel0 macro sets a texel color on a texture surface that was previously locked with the r3d_TextureAnimateSurfaceLock function. The r3d_TextureAnimateSurfaceTexel0 macro should be used when the TextureFormat parameter from the r3d_TextureAnimateSurfaceLock function returns 0.

```
void r3d_TextureAnimateSurfaceTexel0(
    unsigned char *SurfacePointer,
    int Pitch,
    float u,
    float v,
    unsigned char a,
    unsigned char r,
    unsigned char g,
    unsigned char b);
```

Parameters

SurfacePointer,Pitch is obtained from a previous call to r3d_TextureAnimateSurfaceLock.

u,v is the texture coordinate of the texel.

a,r,g,b should contain the alpha, red, green, and blue color components of the texel in the range of 0-255.

Return Value

None.

Source Code Sample

```
// Sets the center texel of a texture surface to red.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
if ( TextureFormat == 0 )
   r3d_TextureAnimateSurfaceTexel0 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 1 )
   r3d_TextureAnimateSurfaceTexel1 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 2 )
   r3d_TextureAnimateSurfaceTexel2 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 3 )
   r3d_TextureAnimateSurfaceTexel3 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
```

```
else if ( TextureFormat == 4 )
   r3d_TextureAnimateSurfaceTexel4 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 5 )
   r3d_TextureAnimateSurfaceTexel5 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceTexel1

The r3d_TextureAnimateSurfaceTexel1 macro sets a texel color on a texture surface that was previously locked with the r3d_TextureAnimateSurfaceLock function. The r3d_TextureAnimateSurfaceTexel1 macro should be used when the TextureFormat parameter from the r3d_TextureAnimateSurfaceLock function returns 1.

```
void r3d_TextureAnimateSurfaceTexel1(
      unsigned char *SurfacePointer,
      int Pitch,
      float u,
      float v,
      unsigned char a,
      unsigned char r,
      unsigned char g,
      unsigned char b);
```

Parameters

SurfacePointer,Pitch is obtained from a previous call to r3d_TextureAnimateSurfaceLock.

u,v is the texture coordinate of the texel.

a,r,g,b should contain the alpha, red, green, and blue color components of the texel in the range of 0-255.

Return Value

None.

Source Code Sample

```
// Sets the center texel of a texture surface to red.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
if ( TextureFormat == 0 )
   r3d_TextureAnimateSurfaceTexel0 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 1 )
   r3d_TextureAnimateSurfaceTexel1 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 2 )
   r3d_TextureAnimateSurfaceTexel2 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 3 )
   r3d_TextureAnimateSurfaceTexel3 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
```

```
else if ( TextureFormat == 4 )
   r3d_TextureAnimateSurfaceTexel4 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 5 )
   r3d_TextureAnimateSurfaceTexel5 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceTexel2

The r3d_TextureAnimateSurfaceTexel2 macro sets a texel color on a texture surface that was previously locked with the r3d_TextureAnimateSurfaceLock function. The r3d_TextureAnimateSurfaceTexel2 macro should be used when the TextureFormat parameter from the r3d_TextureAnimateSurfaceLock function returns 2.

```
void r3d_TextureAnimateSurfaceTexel2(
      unsigned char *SurfacePointer,
      int Pitch,
      float u,
      float v,
      unsigned char a,
      unsigned char r,
      unsigned char g,
      unsigned char b);
```

Parameters

SurfacePointer,Pitch is obtained from a previous call to r3d_TextureAnimateSurfaceLock.

u,v is the texture coordinate of the texel.

a,r,g,b should contain the alpha, red, green, and blue color components of the texel in the range of 0-255.

Return Value

None.

Source Code Sample

```
// Sets the center texel of a texture surface to red.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
if ( TextureFormat == 0 )
   r3d_TextureAnimateSurfaceTexel0 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 1 )
   r3d_TextureAnimateSurfaceTexel1 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 2 )
   r3d_TextureAnimateSurfaceTexel2 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 3 )
   r3d_TextureAnimateSurfaceTexel3 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
```

```
else if ( TextureFormat == 4 )
   r3d_TextureAnimateSurfaceTexel4 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 5 )
   r3d_TextureAnimateSurfaceTexel5 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceTexel3

The r3d_TextureAnimateSurfaceTexel3 macro sets a texel color on a texture surface that was previously locked with the r3d_TextureAnimateSurfaceLock function. The r3d_TextureAnimateSurfaceTexel3 macro should be used when the TextureFormat parameter from the r3d_TextureAnimateSurfaceLock function returns 3.

```
void r3d_TextureAnimateSurfaceTexel3(
      unsigned char *SurfacePointer,
      int Pitch,
      float u,
      float v,
      unsigned char a,
      unsigned char r,
      unsigned char g,
      unsigned char b);
```

Parameters

SurfacePointer,Pitch is obtained from a previous call to r3d_TextureAnimateSurfaceLock.

u,v is the texture coordinate of the texel.

a,r,g,b should contain the alpha, red, green, and blue color components of the texel in the range of 0-255.

Return Value

None.

Source Code Sample

```
// Sets the center texel of a texture surface to red.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
if ( TextureFormat == 0 )
   r3d_TextureAnimateSurfaceTexel0 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 1 )
   r3d_TextureAnimateSurfaceTexel1 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 2 )
   r3d_TextureAnimateSurfaceTexel2 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 3 )
   r3d_TextureAnimateSurfaceTexel3 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
```

```
else if ( TextureFormat == 4 )
   r3d_TextureAnimateSurfaceTexel4 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 5 )
   r3d_TextureAnimateSurfaceTexel5 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceTexel4

The r3d_TextureAnimateSurfaceTexel4 macro sets a texel color on a texture surface that was previously locked with the r3d_TextureAnimateSurfaceLock function. The r3d_TextureAnimateSurfaceTexel4 macro should be used when the TextureFormat parameter from the r3d_TextureAnimateSurfaceLock function returns 4.

```
void r3d_TextureAnimateSurfaceTexel4(
      unsigned char *SurfacePointer,
      int Pitch,
      float u,
      float v,
      unsigned char a,
      unsigned char r,
      unsigned char g,
      unsigned char b);
```

Parameters

SurfacePointer,Pitch is obtained from a previous call to r3d_TextureAnimateSurfaceLock.

u,v is the texture coordinate of the texel.

a,r,g,b should contain the alpha, red, green, and blue color components of the texel in the range of 0-255.

Return Value

None.

Source Code Sample

```
// Sets the center texel of a texture surface to red.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
if ( TextureFormat == 0 )
   r3d_TextureAnimateSurfaceTexel0 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 1 )
   r3d_TextureAnimateSurfaceTexel1 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 2 )
   r3d_TextureAnimateSurfaceTexel2 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 3 )
   r3d_TextureAnimateSurfaceTexel3 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
```

```
else if ( TextureFormat == 4 )
   r3d_TextureAnimateSurfaceTexel4 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 5 )
   r3d_TextureAnimateSurfaceTexel5 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceTexel5

The r3d_TextureAnimateSurfaceTexel5 macro sets a texel color on a texture surface that was previously locked with the r3d_TextureAnimateSurfaceLock function. The r3d_TextureAnimateSurfaceTexel5 macro should be used when the TextureFormat parameter from the r3d_TextureAnimateSurfaceLock function returns 5.

```
void r3d_TextureAnimateSurfaceTexel5(
      unsigned char *SurfacePointer,
      int Pitch,
      float u,
      float v,
      unsigned char a,
      unsigned char r,
      unsigned char g,
      unsigned char b);
```

Parameters

SurfacePointer,Pitch is obtained from a previous call to r3d_TextureAnimateSurfaceLock.

u,v is the texture coordinate of the texel.

a,r,g,b should contain the alpha, red, green, and blue color components of the texel in the range of 0-255.

Return Value

None.

Source Code Sample

```
// Sets the center texel of a texture surface to red.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
if ( TextureFormat == 0 )
   r3d_TextureAnimateSurfaceTexel0 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 1 )
   r3d_TextureAnimateSurfaceTexel1 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 2 )
   r3d_TextureAnimateSurfaceTexel2 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 3 )
   r3d_TextureAnimateSurfaceTexel3 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
```

```
else if ( TextureFormat == 4 )
   r3d_TextureAnimateSurfaceTexel4 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
else if ( TextureFormat == 5 )
   r3d_TextureAnimateSurfaceTexel5 ( SurfacePointer, SurfacePitch, WidthHeight/2,
WidthHeight/2, 255, 255, 0, 0 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap, r3d_TextureAnimateSurfaceUnlock

r3d_TextureAnimateSurfaceUnlock

The r3d_TextureAnimateSurfaceUnlock function unlocks a texture surface that was previously locked with r3d_TextureAnimateSurfaceLock. The r3d_TextureAnimateSurfaceUnlock function should be called as soon as access to the texture surface is complete.

BOOL r3d_TextureAnimateSurfaceUnlock(
    int hMaterial);

Parameters

hMaterial is the handle of a material obtained from a previous call to r3d_ListLoad or r3d_Get3DModelMaterialHandle. This parameter should correspond to a texture surface that was previously locked with r3d_TextureAnimateSurfaceLock.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Copy the back buffer to a texture surface
// with anti-aliasing.
//
// This example assumes that hMaterial[0] is a valid
// material handle which contains a texture.
//
int hBitmapSrc, hBitmapDst, BackBufferWidth, BackBufferHeight;
int WidthHeight, TextureFormat, SurfacePitch;
unsigned char *SurfacePointer;
r3d_GetDisplayMode( &BackBufferWidth, &BackBufferHeight, NULL, NULL, NULL, NULL,
NULL);
r3d_SurfaceCreateFromBackBuffer( &hBitmapSrc )
r3d_TextureAnimateSurfaceLock( hMaterial[0], &WidthHeight, &TextureFormat,
&SurfacePointer, &SurfacePitch );
r3d_SurfaceCreate( &hBitmapDst, WidthHeight, WidthHeight );
r3d_SurfaceStretchSmooth( hBitmapSrc, hBitmapDst );
r3d_TextureAnimateSurfaceBlit( hBitmapDst, WidthHeight, TextureFormat,
SurfacePointer, SurfacePitch, 0, r3d_SurfaceGetWidth(hBitmapDst)-1, 0,
r3d_SurfaceGetHeight(hBitmapDst)-1, 0, WidthHeight-1, 0, WidthHeight-1 );
r3d_TextureAnimateSurfaceUnlock( hMaterial[0] );
r3d_SurfaceDestroyAll();
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap

r3d_TextureAnimateSwap

The r3d_TextureAnimateSwap function assigns a new material for the specified material on a 3D model. The r3d_TextureAnimateSwap function provides an efficient method for achieving texture animation. The original material on the 3D model is not required to contain a texture. If you wish to reassign the original material after calling r3d_TextureAnimateSwap, you need to save the material handle using r3d_Get3DModelMaterialHandle prior to calling this function.

```
BOOL r3d_TextureAnimateSwap(
    int hModel,
    int ModelMaterialOffset,
    int hMaterial);
```

Parameters

hModel is the model handle obtained with a previous call to r3d_ListLoad.

ModelMaterialOffset is the offset of a material within the model. This parameter can not equal or exceed the number of materials supplied by r3d_Get3DModelMaterialCount. All vertex and polygon data are grouped by material. If you passed an offset of zero for this parameter, you would be indexing the first material of the model as well as their corresponding polygons and vertices. To find out which material offset corresponds to a particular material or texture of a particular RenderIt 3D model (*.R3D), load the RenderIt 3D model into Materialize 3D! and inspect the material numbers shown in the "Edit Materials" dialog box. The material numbers are equal to the parameter that should be passed to this function. If you save the model within Materialize 3D! after inspection, the material numbers may be re-assigned possibly making them invalid.

hMaterial is the handle of a material obtained from a previous call to r3d_ListLoad or r3d_Get3DModelMaterialHandle.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Render an explosion texture animation
//
//
// Somewhere in the global data area:
//
#define NUM_MODELS      1
#define NUM_MATERIALS   16
long hModel[NUM_MODELS],hMaterial[NUM_MATERIALS];
int ghSavedModelMaterial;
//
// Somewhere in your level loader:
```

```
//
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode01.bmp", TRUE, "ExplodeAlpha01.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode02.bmp", TRUE, "ExplodeAlpha02.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode03.bmp", TRUE, "ExplodeAlpha03.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode04.bmp", TRUE, "ExplodeAlpha04.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode05.bmp", TRUE, "ExplodeAlpha05.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode06.bmp", TRUE, "ExplodeAlpha06.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode07.bmp", TRUE, "ExplodeAlpha07.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode08.bmp", TRUE, "ExplodeAlpha08.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke09.bmp", TRUE, "SmokeAlpha09.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke10.bmp", TRUE, "SmokeAlpha10.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke11.bmp", TRUE, "SmokeAlpha11.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke12.bmp", TRUE, "SmokeAlpha12.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke13.bmp", TRUE, "SmokeAlpha13.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke14.bmp", TRUE, "SmokeAlpha14.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke15.bmp", TRUE, "SmokeAlpha15.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke16.bmp", TRUE, "SmokeAlpha16.bmp", FALSE );
r3d_ListAdd3DModel( "Rectangle.r3d", FALSE );
r3d_ListLoad( &hModel[0], &hMaterial[0] );
//
// We'll save the original model material handle just in case . . .
//
r3d_Get3DModelMaterialHandle( hModel[0], 0, &ghSavedModelMaterial );
//
// Somewhere in your rendering loop:
//
// We animate the texture by swapping material handles
// in the model. We assume that a variable "Frame" has
// been assigned to the appropriate frame number. In
// this case, frame is between 0 and 15.
//
r3d_TextureAnimateSwap( hModel[0], 0, hMaterial[Frame] );
//
// We could blit the 3D model within an r3d_RenderBegin
// and r3d_RenderEnd function pair. fMatrix is assumed
// to be a valid 12 element RenderIt 3D! matrix
// containing the 3D translation of the model in world
// space.
//
r3d_ZBlit3DModel2( hModel[0], &fMatrix[0] );
//
// If for some reason we wanted to restore the original
// material on the model, we could do it like this:
//
r3d_TextureAnimateSwap( hModel[0], 0, ghSavedModelMaterial );
```

See Also

r3d_TextureAnimateScroll, r3d_TextureAnimateSurfaceBlit, r3d_TextureAnimateSurfaceLock, r3d_TextureAnimateSwap

r3d_ViewportClear

The r3d_ViewportClear function will set the z-buffer or back buffer to a cleared state.

```
BOOL r3d_ViewportClear(
      BOOL bClearZBuffer,
      BOOL bClearBackBuffer);
```

Parameters

bClearZBuffer should be set to TRUE if you want to clear the z-buffer surface.

bClearBackBuffer should be set to TRUE if you want to clear the back buffer surface. If you have rendered a scene that covers the entire screen area, you can set this parameter to FALSE and increase performance.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
r3d_ViewportClear( TRUE, TRUE );
```

See Also

r3d_PageFlip, r3d_SetBackgroundColor , r3d_SetRenderArea

r3d_WindowInit

The r3d_WindowInit function should be called if the application is running in a window. The r3d_WindowInit function should be called after setting up the main window but before calling r3d_Initialize.

```
void r3d_WindowInit(
      RECT rcScreen,
      RECT rcViewport);
```

Parameters

rcScreen is a windows RECT structure defining the screen area of the window. The top and left members of this structure should be set to the physical screen coordinates of the top and left pixel of the window. The RECT structure is inclusive - For example, the right member should be equal to left+width instead of left+width-1 and the bottom member should be equal to top+height instead of top+height-1.

rcViewport is a windows RECT structure defining the rendering area of the window. The top and left members of this structure should be set to 0,0. The right and bottom members of this structure should be set to the width and height of the window. To increase performance with software rendering, you can decrease the size of Viewport through the right and bottom members. Decreasing the size of Viewport, is not recommended when running in 3D hardware acceleration since performance may actually decrease.

Return Value

None.

Source Code Sample

```
//
// Somewhere in the global data area
//
BOOL gb3Daccelerated = FALSE;
BOOL gbWindowed = FALSE;
HWND ghWwnd;
char gDriver[32][40];
char gDevice[32][32][40];
char gMode[32][32][32][40];
int gDriverSelection=0;
int gDeviceSelection=0;
int gModeSelection=0;
//
// Somewhere in the WinMain function . . .
//
RECT ViewportRect, WindowRect;
if ( r3d_Enumerate( gDriver, gDevice, gMode ) == FALSE )
{
r3d_MessageBox(r3d_GetLastErrorString(),"Program Aborting",MB_OK);
return FALSE;
}
// gDriverSelection, gDeviceSelection, gModeSelection initialized in dialog box
```

```
if ( DialogBox( hInst, MAKEINTRESOURCE(IDD_RENDEROPTIONS), NULL,
(DLGPROC)DialogSelectDeviceProc) == 0 )
return FALSE;
int Width, Height;
r3d_GetSelectionInfo( gDriverSelection, gDeviceSelection, gModeSelection, &Width,
&Height, NULL, &gb3Daccelerated );
if (Width==0 && Height==0)
{
// Selection is windowed, set up for windowed mode.
gbWindowed = TRUE;
WNDCLASS WndClass = { CS_HREDRAW|CS_VREDRAW, WinProc, 0, 0, hInst, LoadIcon( hInst,
IDI_APPLICATION ), LoadCursor( NULL, IDC_ARROW ), (HBRUSH)GetStockObject( BLACK_BRUSH
), APP_NAME, APP_NAME };
RegisterClass( &WndClass );
ghWnd = CreateWindow( APP_NAME, APP_NAME, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
CW_USEDEFAULT, 400, 300+20, 0L, 0L, hInst, 0L );
ShowWindow( ghWnd, SW_SHOWNORMAL );
SetFocus( ghWnd );
UpdateWindow( ghWnd );
// For windowed mode only...
GetClientRect( ghWnd, &ViewportRect );
GetClientRect( ghWnd, &WindowRect );
ClientToScreen( ghWnd, (POINT*)&WindowRect.left );
ClientToScreen( ghWnd, (POINT*)&WindowRect.right );
r3d_WindowInit( WindowRect, ViewportRect );
}
else
{
// Selection is fullscreen, set up for fullscreen mode.
gbWindowed = FALSE;
WNDCLASS WndClass = { CS_HREDRAW|CS_VREDRAW, WinProc, 0, 0, hInst, LoadIcon( hInst,
IDI_APPLICATION ), LoadCursor( NULL, IDC_ARROW ), NULL, APP_NAME, APP_NAME };
RegisterClass( &WndClass );
ghWnd = CreateWindowEx( WS_EX_TOPMOST, APP_NAME, APP_NAME, WS_POPUP, 0, 0,
GetSystemMetrics( SM_CXSCREEN ), GetSystemMetrics( SM_CYSCREEN ), NULL, NULL, hInst,
NULL );
ShowWindow( ghWnd, SW_SHOWNORMAL );
UpdateWindow( ghWnd );
SetFocus( ghWnd );
SetCursor( NULL );
}
if ( r3d_Initialize( ghWnd, 100000.0F, 75.0F, gDriverSelection, gDeviceSelection,
gModeSelection ) == FALSE )
{
r3d_MessageBox( r3d_GetLastErrorString(), "Fatal Error - Program Aborting", MB_OK );
r3d_Finished();
return FALSE;
}
```

See Also

r3d_WindowMove, r3d_WindowSize

r3d_WindowMove

The r3d_WindowMove function should be called when a WM_MOVE message is sent to the window and only if the application is running in an active window.

```
void r3d_WindowMove(
        short x,
        short y);
```

Parameters

x is the low word of LPARAM which is passed to the window procedure.

y is the high word of LPARAM which is passed to the window procedure.

Return Value

None.

Source Code Sample

```
// gbMainLoop is a BOOL variable that is set to
// TRUE if the main message loop is active
//
// gbActiveApp is a BOOL variable that is set to
// TRUE if the window has been activated
//
// Somewhere in your switch statement inside your
// windows procedure function
//
case WM_MOVE:
if (gbMainLoop && gbActiveApp)
{
r3d_WindowMove((short)LOWORD(lParam),(short)HIWORD(lParam));
}
break;
```

See Also

r3d_WindowInit, r3d_WindowSize

r3d_WindowSize

The r3d_WindowSize function should be called when a WM_SIZE message is sent to the window and only if the application is running in an active window.

```
int r3d_WindowSize(
      RECT rcScreen,
      RECT rcViewport);
```

Parameters

rcScreen is a windows RECT structure defining the screen area of the window. The top and left members of this structure should be set to the physical screen coordinates of the top and left pixel of the window. The RECT structure is inclusive - For example, the right member should be equal to left+width instead of left+width-1 and the bottom member should be equal to top+height instead of top+height-1.

rcViewport is a windows RECT structure defining the rendering area of the window. The top and left members of this structure should be set to 0,0. The right and bottom members of this structure should be set to the width and height of the window. To increase performance with software rendering, you can decrease the size of Viewport through the right and bottom members. Decreasing the size of Viewport, is not recommended when running in 3D hardware acceleration since performance may actually decrease.

Return Value

If the function fails, the return value is 0.

If the function succeeds, the return value is 1.

If the function determines that there is insufficient video memory to resize the rendering surfaces, the return value is 2, and the function automatically uses the previous window size.

Source Code Sample

```
case WM_MOVE:
if (gbMainLoop && gbActiveApp)
{
r3d_WindowMove((short)LOWORD(lParam),(short)HIWORD(lParam));
}
break;

case WM_SIZE:
if( SIZE_MAXHIDE==wParam || SIZE_MINIMIZED==wParam )
gbActiveApp=FALSE;
else
gbActiveApp=TRUE;

if (gbWindowed && gbMainLoop && gbActiveApp)
{
gbMainLoop=FALSE;
GetClientRect(hWnd,&ViewportRect);
GetClientRect(hWnd,&WindowRect);
ClientToScreen(hWnd,(POINT*)&WindowRect.left);
ClientToScreen(hWnd,(POINT*)&WindowRect.right);
if (gb3DAccelerated==FALSE)
{
if (r3d_MessageBox("Would you like to optimize rendering performance at the expense
of visual quality?",APP_NAME,MB_YESNO)==IDYES)
{
// We just set the viewport (or render target) size to 1/4 of the actual window size
ViewportRect.right/=2;
ViewportRect.bottom/=2;
}
}
ret=r3d_WindowSize(WindowRect,ViewportRect);
if (ret==2)
// Not Fatal - Simply not enough memory
// Resort to original window size
r3d_MessageBox("Insufficient video memory available for new window size. Reverting to
previous window size.",APP_NAME,MB_OK);
else if (ret==0)
// Fatal Error
r3d_MessageBox(r3d_GetLastErrorString(),"r3d_WindowSize",MB_OK);
// It a good idea to get the new width and height in your
// global width and height variables.
r3d_GetDisplayMode(&gWidth,&gHeight,NULL,NULL,NULL,NULL,NULL);
gbMainLoop=TRUE;
}
break;

case WM_GETMINMAXINFO:
((MINMAXINFO*)lParam)->ptMinTrackSize.x=320;
((MINMAXINFO*)lParam)->ptMinTrackSize.y=240+20;
break;
```

See Also

r3d_WindowInit, r3d_WindowMove

r3d_ZBlit3DModel1

The r3d_ZBlit3DModel1 function will "blit" a 3D model onto the rendering surface as though it were a 2D bitmap with depth information. The model will always be oriented toward the view. The r3d_ZBlit3DModel1 function ignores the rotation of view on the z-axis.

```
BOOL r3d_ZBlit3DModel1(
    long hModel,
    float *pf4x3ModelMatrix);
```

Parameters

hModel is the handle of a model obtained from a previous call to r3d_ListLoad.

pf4x3ModelMatrix is a pointer to a 12 element array describing the matrix of the model. The first 9 elements of the array describe the rotation of the model and is ignored. The last 3 elements describe the location of the model in world space.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Render an explosion texture animation
//
//
// Somewhere in the global data area:
//
#define NUM_MODELS      1
#define NUM_MATERIALS   16
long hModel[NUM_MODELS],hMaterial[NUM_MATERIALS];
int ghSavedModelMaterial;
//
// Somewhere in your level loader:
//
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode01.bmp", TRUE, "ExplodeAlpha01.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode02.bmp", TRUE, "ExplodeAlpha02.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode03.bmp", TRUE, "ExplodeAlpha03.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode04.bmp", TRUE, "ExplodeAlpha04.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode05.bmp", TRUE, "ExplodeAlpha05.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode06.bmp", TRUE, "ExplodeAlpha06.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode07.bmp", TRUE, "ExplodeAlpha07.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode08.bmp", TRUE, "ExplodeAlpha08.bmp", FALSE );
```

```
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke09.bmp", TRUE, "SmokeAlpha09.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke10.bmp", TRUE, "SmokeAlpha10.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke11.bmp", TRUE, "SmokeAlpha11.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke12.bmp", TRUE, "SmokeAlpha12.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke13.bmp", TRUE, "SmokeAlpha13.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke14.bmp", TRUE, "SmokeAlpha14.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke15.bmp", TRUE, "SmokeAlpha15.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke16.bmp", TRUE, "SmokeAlpha16.bmp", FALSE );
r3d_ListAdd3DModel( "Rectangle.r3d", FALSE );
r3d_ListLoad( &hModel[0], &hMaterial[0] );
// We'll save the original model material handle just in case . . .
r3d_Get3DModelMaterialHandle( hModel[0], 0, &ghSavedModelMaterial );
//
// Somewhere in your rendering loop:
//
// We animate the texture by swapping material handles
// in the model. We assume that a variable "Frame" has
// been assigned to the appropriate frame number. In
// this case, frame is between 0 and 15.
//
r3d_TextureAnimateSwap( hModel[0], 0, hMaterial[Frame] );
//
// We could blit the 3D model within an r3d_RenderBegin
// and r3d_RenderEnd function pair. fMatrix is assumed
// to be a valid 12 element RenderIt 3D! matrix
// containing the 3D translation of the model in world
// space.
//
r3d_ZBlit3DModel1( hModel[0], &fMatrix[0] );
//
// If for some reason we wanted to restore the original
// material on the model, we could do it like this:
//
r3d_TextureAnimateSwap( hModel[0], 0, ghSavedModelMaterial );
```

See Also

r3d_ZBlit3DModel2, r3d_Render3DModel

r3d_ZBlit3DModel2

The r3d_ZBlit3DModel2 function will "blit" a 3D model onto the rendering surface as though it were a 2D bitmap with depth information. The model will always be oriented toward the view. The r3d_ZBlit3DModel2 function does not ignore the rotation of view on the z-axis.

```
BOOL r3d_ZBlit3DModel2(
    long hModel,
    float *pf4x3ModelMatrix);
```

Parameters

hModel is the handle of a model obtained from a previous call to r3d_ListLoad.

pf4x3ModelMatrix is a pointer to a 12 element array describing the matrix of the model. The first 9 elements of the array describe the rotation of the model and is ignored. The last 3 elements describe the location of the model in world space.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE and a call to r3d_GetLastErrorString will return a string identifying the error.

Source Code Sample

```
// Render an explosion texture animation
//
//
// Somewhere in the global data area:
//
#define NUM_MODELS      1
#define NUM_MATERIALS   16
long hModel[NUM_MODELS],hMaterial[NUM_MATERIALS];
int ghSavedModelMaterial;
//
// Somewhere in your level loader:
//
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode01.bmp", TRUE, "ExplodeAlpha01.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode02.bmp", TRUE, "ExplodeAlpha02.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode03.bmp", TRUE, "ExplodeAlpha03.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode04.bmp", TRUE, "ExplodeAlpha04.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode05.bmp", TRUE, "ExplodeAlpha05.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode06.bmp", TRUE, "ExplodeAlpha06.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode07.bmp", TRUE, "ExplodeAlpha07.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Explode08.bmp", TRUE, "ExplodeAlpha08.bmp", FALSE );
```

```
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke09.bmp", TRUE, "SmokeAlpha09.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke10.bmp", TRUE, "SmokeAlpha10.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke11.bmp", TRUE, "SmokeAlpha11.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke12.bmp", TRUE, "SmokeAlpha12.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke13.bmp", TRUE, "SmokeAlpha13.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke14.bmp", TRUE, "SmokeAlpha14.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke15.bmp", TRUE, "SmokeAlpha15.bmp", FALSE );
r3d_ListAddMaterial( 1.0F, 1.0F, 1.0F, 1.0F, 0.0F, 0.0F, 0.0F, 0.0F, 1.0F, 1.0F,
1.0F, TRUE, "Smoke16.bmp", TRUE, "SmokeAlpha16.bmp", FALSE );
r3d_ListAdd3DModel( "Rectangle.r3d", FALSE );
r3d_ListLoad( &hModel[0], &hMaterial[0] );
// We'll save the original model material handle just in case . . .
r3d_Get3DModelMaterialHandle( hModel[0], 0, &ghSavedModelMaterial );
//
// Somewhere in your rendering loop:
//
// We animate the texture by swapping material handles
// in the model. We assume that a variable "Frame" has
// been assigned to the appropriate frame number. In
// this case, frame is between 0 and 15.
//
r3d_TextureAnimateSwap( hModel[0], 0, hMaterial[Frame] );
//
// We could blit the 3D model within an r3d_RenderBegin
// and r3d_RenderEnd function pair. fMatrix is assumed
// to be a valid 12 element RenderIt 3D! matrix
// containing the 3D translation of the model in world
// space.
//
r3d_ZBlit3DModel2( hModel[0], &fMatrix[0] );
//
// If for some reason we wanted to restore the original
// material on the model, we could do it like this:
//
r3d_TextureAnimateSwap( hModel[0], 0, ghSavedModelMaterial );
```

See Also

r3d_ZBlit3DModel1, r3d_Render3DModel