# tsDirectGraphics

tsDirectGraphics は Microsoft DirectX9 の描画機能を簡単に使うためのライブラリです
tsDirectGraphics では２Ｄ描画を担当します
３Ｄ描画が必要な場合は派生クラスの tsDirectGraphics3D を利用してください

tsDirectGraphics は非常にシンプルなコードの記述だけで簡単に DirectX を使った描画ができますが、その気になればマルチスクリーンはテクスチャレンダリング、シェーダーを使った高度なプログラミングもできる様に設計されています
弊社の正式なツール開発やゲーム開発で 10 年以上使用されている実績のあるライブラリです

ts ライブラリシリーズは主にゲーム作成を目的に作られています
tsDirectGraphics 以外には以下のライブラリがあります

- tsDirectGraphics
  本ライブラリで２Ｄ描画を担当します

- tsDirectGraphics3D
  tsDirectGraphics の派生クラスで３Ｄ描画を担当します

- tsDirectInput
  ジョイスティックやキーボードの入力を担当します

- tsDirectSound
  wav ファイルによる音楽、効果音の再生を担当します

- tsAVI
  avi ファイルによる動画の再生を担当します
  動画をテクスチャに張り込む事もできます

- tsSocket
  TCP によるクライアント、サーバモデルの通信を担当します

その他に、これらのヘルパークラスとして tsGgCtrl や tsGameInput などがあります

# 目次

## 内容

## 動作環境

- DirectX9 が動作する WindowsPC
- ライブラリの Lib は VisualStudio2013(32bit ビルド)と C++BuilderXE(32bit ビルド)様にそれぞれビルドされています
- WindowsXP(32bit,SP3)、Windows7(64bit)、Window8.1(64bit)で動作確認しています
- ライブラリは DirectX9c(June 2010)の 32bit 版でビルドされています

## 必要なソフトウェア

・Microsoft DirectX9 以上
(以下の Microsoft のサイトから無料でダウンロードできます)
http://www.microsoft.com/japan/directx/default.mspx

## 開発準備

本ライブラリに対して、インクルードパスを通して lib ファイルをリンクすればそのまま使用できます

DirectX9 SDK へのインクルードパスとライブラリパスの指定は DirectX9 SDK のドキュメントをご覧ください

# チュートリアル

## クラスの宣言

tsDirectGraphics のヘッダーファイルをインクルードし、実態の宣言をします
この場合の引数は 1 の場合、エラー等が発生した場合にメッセージボックスを開いてエラーメッセージを表示します
引数が 0 の場合はエラー表示をしませんので、それぞれの関数の戻り値によってユーザーがエラー表示をする事になります

```
#include “tsDirectGraphics.h”
tsDirectGraphics dxg(1);
```

サンプルでは直接実態の宣言をしていますが、new 端子を使って宣言しても構いません

```
tsDirectGraphics *dxg=new tsDirectGraphics(1);
```

必要な DirectX ライブラリのインクルードは tsDirectGraphics 内で行われていますので必要ありません

## 初期化処理

作成するプログラム内でウインドウが作成されてウインドウハンドルが取得可能になった時点で一度だけ初期化処理を呼び出します

```
dxg.initialize(hwnd,800,600,false);
```

サンプルは最も単純な初期化です
第一引数にウインドウハンドル、続いてフレームバッファの幅と高さ、最後の引数はフルスクリーンモードで動作させる場合は true を指定します

なお、ウインドウ自体のフルスクリーン指定はメニューの有る無しやウインドウスタイルに依存しますので、ユーザー側で行ってください

基本的に作成したウインドウの描画範囲とフレームバッファのサイズは同じ方が好ましいですが、違う場合はフレームバッファの内容がスケーリングされて表示される事になります

もっと詳細な初期化パラメータに関しては、関数一覧をご覧ください

## メインループでの処理

プログラム内で毎回呼び出されるメインループ内で以下の様にして描画をします

```
dxg.beginScene();          //シーンの描画開始
dxg.clear();               //フレームバッファのクリア

ここに各種描画処理を書く

dxg.endScene();            //シーンの描画終了
dxg.drawScene();           //バックバッファとスワップして表示
```

基本的にはこれだけで画面描画ができます

フレームバッファのクリアは必要に応じて行ってください

## サンプルプログラム

以下は、テクスチャを１枚表示する簡単なサンプルです

VisualStudio で作成していますが、Win32API を使って作成されています

```
/*************************************************************/
/* tsDirectGraphicsサンプルプログラム
/*************************************************************/
#include <windows.h>
#include <tchar.h>
#include <tsDirectGraphics.h>

//定数宣言
#define CAPTION _T("sample program")
#define APPNAME _T("sample")
#define FULLSCREEN 0
#define SCREEN_WIDTH     800
#define SCREEN_HEIGHT    600

//関数のプロトタイプ宣言
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void appInit(HWND hwnd, HINSTANCE hInstance);
void appQuit(void);
void appMain(void);

//変数宣言
bool endFlag = false;
tsDirectGraphics dxg(1);

//ウインドウズメイン
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
        WNDCLASS wc;
        //ウインドウクラス構造体
        //HWND hwnd;
                //ウインドウハンドル
        MSG msg;
        //メッセージ構造体
        int width, height;
        //画面サイズ
        //ウインドウの作成
        ZeroMemory(&wc, sizeof(WNDCLASS));
        wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hInstance = hInstance;
        wc.lpfnWndProc = WndProc;
        wc.lpszClassName = APPNAME;
        wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

```
	RegisterClass(&wc);
	if (fullScreen){
		width = SCREEN_WIDTH;
		height = SCREEN_HEIGHT;
	}else{
		//width = SCREEN_WIDTH + GetSystemMetrics(SM_CXDLGFRAME) * 2;
		//height = SCREEN_HEIGHT + GetSystemMetrics(SM_CYDLGFRAME) * 2 +
GetSystemMetrics(SM_CYCAPTION);
		RECT clientRect = { 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT };
		AdjustWindowRectEx(&clientRect, WS_BORDER | WS_CAPTION |
WS_SYSMENU | WS_MINIMIZEBOX | WS_VISIBLE, FALSE, 0);
		width = clientRect.right - clientRect.left;
		height = clientRect.bottom - clientRect.top;
	}
	windowHandle = CreateWindow(APPNAME, CAPTION, WS_OVERLAPPED | WS_SYSMENU
| WS_MINIMIZEBOX | WS_VISIBLE,
		CW_USEDEFAULT, CW_USEDEFAULT, width, height, NULL, NULL,
hInstance, NULL);
	//初期化
	appInit(windowHandle, hInstance);
	//メッセージループ
	while (!endFlag)
	{
		if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
		else{
			appMain();
		}
	}
	return 0;
}
```

```
//ウインドウプロシージャ
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
	switch (message)
	{
	case WM_DESTROY:
		//ウインドウが破棄された時の処理
		appQuit();
		endFlag = true;
		return 0;
	case WM_KEYDOWN:
		//キー入力
		switch (wParam){
		case VK_ESCAPE:
			endFlag = true;
			break;
		}
		break;
	}
	//デフォルトの処理
	return DefWindowProc(hWnd, message, wParam, lParam);
}

//===========================================
//	appInit
//	概要：	アプリケーションの初期化処理
//	引数：	hwnd		ウインドウハンドル
//			hInstance	インスタンス
//	戻値：	無し
//===========================================
void appInit(HWND hwnd, HINSTANCE hInstance)
{
	dxg.initialize(hwnd, SCREEN_WIDTH, SCREEN_HEIGHT,
FULLSCREEN);
	dxg.loadTexture(0,_T("..¥¥data¥¥test.png"));
}
```

```
//=============================================
//		appQuit
//		概要：　アプリケーションの終了処理
//		引数：　無し
//		戻値：　無し
//=============================================
void appQuit(void)
{
}

//=============================================
//		appMain
//		概要：　アプリケーションのメイン処理
//		引数：　無し
//		戻値：　無し
//=============================================
void appMain(void)
{
	dxg.beginScene();
	dxg.clear();

	dxg.drawSprite(0,&D3DXVECTOR2(0,0));

	dxg.endScene();
	dxg.drawScene();
}
```

サンプルではウインドウの表示のため多少コードが長くなっていますが、実際に tsDirectGraphics を使うための処理は appInit 関数と appMain 関数の中だけです
ロードしたテクスチャリソースの解放などは tsDirectGraphics が行うので意識する必要はありません

## テクスチャ読み込み時の注意

現在の GPU のほとんどがテクスチャへの参照を効率化するためにテクスチャ画像を読み込んだ時点で幅及び高さを２のべき乗にします
このため元のテクスチャサイズが２のべき乗以外の時は少し工夫が必要になります

## drawSprite による描画

drawSprite で描画する場合は loadTexture 関数でテクスチャを読み込む際に filter 引数に D3DX_FILTER_NONE を指定します
通常は２のべき乗に引き伸ばされますが、D3DX_FILTER_NONE を指定して読み込んだ場合は元画像の右側と下側に余白を作って２のべき乗にします
このまま drawSprite で描画すると正常に描画されます
逆に drawPolygon を使った場合は UV が 0～1 に指定されている場合、余白部分まで含んだ画像になってしまいます
この場合は読み込まれたテクスチャサイズと元画像のテクスチャサイズから適切な UV 座標を指定する必要があります

## drawPolygon による描画

drawPolygon で描画する場合には各頂点に対して UV 値 0～1 でテクスチャ座標を指定するため２のべき乗に引き伸ばされたテクスチャでの正常に表示できます
逆に D3DX_FILTER_NONE を指定して読み込んだテクスチャでは余白部分を含んでしまいますので、適切な UV を指定する必要があります

## スプライトの回転拡大

通常はテクスチャの左上を原点にスプライトの描画がされますが、拡大回転などを行う際にはスプライトの中心を原点にした方が楽な場合が多いです
drawSprite 関数による描画で原点を指定するには引数 center にスプライトの中心を入れます
もちろんわざとずらしてピボットの様に使う事もできます
スプライトのサイズが分かっている場合はスプライトサイズを２で割れば良いですが、自動的に計算できれば楽になります
この場合、loadTexture 関数の引数 srcInfo を指定します
デフォルトでは NULL が指定されていますが、D3DXIMAGE_INFO へのポインタを指定する事で、スプライトの元サイズなどの情報が取得できます
以下はスプライトの中心を原点に１フレームに１度づつ回転させるサンプルです

```
読み込み部分
D3DXIMAGE_INFO info;
dxg.loadTexture(0,”test.png”,0, D3DFMT_UNKNOWN,&info,
D3DX_FILTER_NONE);

描画部分
D3DXVECTOR2 center;
center.x=info.Width/2.0f;
center.y=info.Height/2.0f;
dxg.drawSprite(0, &D3DXVECTOR2(100,100),NULL,NULL,center,r++);
```

## ポリゴンを使った描画

ポリゴンを使った描画ではテクスチャ読み込み時に D3DX_FILTER_NONE を指定すると余白ができてかえって面倒なのでそのまま読み込みます
あとは UV を 0～1 に指定して描画すれば正常に表示できます
ただしポリゴンの場合は各頂点を自分で指定しないといけないので元画像と同じサイズで表示する場合はテクスチャ読み込み時に引数 srcInfo を指定して元のテクスチャサイズを取得しておきます

```
読み込み部分
D3DXIMAGE_INFO info;
dxg.loadTexture(0,”test.png”,0, D3DFMT_UNKNOWN,&info);

描画部分
WDTVERTEX vtx[4]={
   {0,0,0,1,D3DCOLOR_XRGB(255,255,255),0,0},
   {info.Widht,0,0,1,D3DCOLOR_XRGB(255,255,255),1,0},
   {info.Width,info.Height,0,1,D3DCOLOR_XRGB(255,255,255),1,1},
   {0.info.Height,0,1,D3DCOLOR_XRGB(255,255,255),0,1}
};
dxg.drawPolygon(4,0,vtx);
```

## レンダリングテクスチャを使う

レンダリングテクスチャとはテクスチャ自体をフレームバッファの様にして、シーン自体をテクスチャにレンダリングして、レンダリングされたテクスチャを別のシーンで再利用する方法です

cleateRenderTexture 関数でレンダリング用テクスチャを作成し、selectRenderTexture でレンダリングターゲットを変更してあとは普通にシーンの描画をすればレンダリングテクスチャの出来上がりです

レンダリングテクスチャは普通のテクスチャと同じ様に使う事ができます

テクスチャの管理番号は通常のテクスチャとレンダリングテクスチャでは独立しています

```
dxg.loadTexture(0,”test.png”);　　//テクスチャ番号 0 に”test.png”を読み込む
レンダリングテクスチャ作成
dxg.createRenderTexture(0,256,256); //レンダリングテクスチャ番号 0 に 256×256 のテクスチャを作成

レンダリングテクスチャへの描画部分
dxg.selectRenderTexture(0);　　//レンダリングターゲットを切り替える
dxg..beginScene();
dxg.clear(D3DCOLOR_XRGB(0,255,0));　　//背景を緑でクリア
dxg.drawSprite(0,&D3DXVECTOR2(0,0);　//スプライトを 3 枚描画
dxg.drawSprite(0,&D3DXVECTOR2(128,128);
dxg.drawSprite(0,&D3DXVECTOR2(256,256);
dx.endScene();
dxg.selectRenderTexture(-1);　　//レンダリングターゲットをフレームバッファに戻す



//レンダリングテクスチャを描画
dxg.drawSprite(0,&D3DXVECTOR(0,0),NULL,NULL,NULL,0,0xffffffff,true);
```

## ビューポートを使ったクリッピング

本来、ビューポートは３Ｄ描画時のスクリーン座標変換のための物ですが、これを使って２Ｄ描画の際にもクリッピングする事ができます
例えば 800×600 ドットの画面で左上 1/4 だけ描画してあとはクリッピングしたい場合は以下の様に指定します

```
D3DVIEWPORT9 viewData;
viewData.X=0;
viewData.Y=0;
viewData.Width=400;
viewData.Height=300;
dx9g.setViewport(&viewData);
```

クリッピング領域を画面全体に戻したい時は、引数なしで dx9g.setViewport 関数を呼び出します

## テクスチャの管理

loadTexture や createTexture を使って作成されたテクスチャリソースの解放などは tsDirectGraphics が自動で行います
特にレンダリングテクスチャを作成する場合は深度バッファを共有するために深度バッファの計算などを行わないといけないので、極力 tsDirectGraphics の関数を使った方が楽です
tsDirectGraphics は管理番号でテクスチャを管理しますが、通常のテクスチャとレンダリングテクスチャの管理番号は独立しているので、番号が被っても問題ありません
使用できるテクスチャ数は tsDirectGraphics.h に記述されています
通常テクスチャ：0～16383
レンダリングテクスチャ：0～11

## 構造体一覧

tsDirectGraphics では以下の構造体が使用されます

### WDVERTEX

テクスチャ無しの変換済み２Ｄ頂点です

頂点を指定してポリゴンを描画する際に仕様します

WDVERTEX は以下の様に定義されています

```
typedef struct{
        float x,y,z;
        float rhw;
        D3DCOLOR color;
}WDVERTEX;
```

### WDTVERTEX

テクスチャ付の変換済み２Ｄ頂点です

頂点を指定してテクスチャ付ポリゴンを描画する際に仕様します

WDTVERTEX は以下の様に定義されています

```
typedef struct{
        float x,y,z;
        float rhw;
        D3DCOLOR color;
        float tu,tv;
}WDVERTEX;
```

### RENDERTEX_PARAM

レンダリング用テクスチャの情報です

RENDERTEX_PARAM は以下の様に定義されています

```
typedef struct{
        int w,h;
        D3DFORMAT format;
} RENDERTEX_PARAM;
```

## 関数一覧

### checkResult

宣言

bool checkResult(void);

引数

無し

戻値

初期化終了済みの場合は true

未初期化の場合は false

説明

tsDirectGraphics によるデバイスの初期化が終了しているか調べます

タイマー割り込み関数などで描画を行う場合など、初期化が終了する前に描画関数が呼び出される可能性がある場合は、この関数でチェックして true が帰った場合のみ描画関数を呼び出します

## initialize

### 宣言

bool initialize(HWND hwnd,int width,int height,bool fullScreen,int sync=1,int refresh=60,int dispMode=-1,D3DSWAPEFFECT swapEffect=D3DSWAPEFFECT_COPY,UINT adapter=D3DADAPTER_DEFAULT,bool stencile=false);

### 説明

tsDirectGraphics の初期化を行います
最小限で３つの引数を指定する必要があります

### 戻値

初期化に成功した場合は true
失敗した場合は false

### 引数

**hwnd**
　描画対象となるウインドウのウインドウハンドルを指定します
**width**
　フレームバッファの幅を指定します
**height**
　フレームバッファの高さを指定します
**fullScreen**
　フルスクリーンモードの場合は true、ウインドウモードの場合は false を指定します
**sync**
　フレームバッファのフリップ時に垂直同期を待つ場合は 1、待たない場合は 0 を指定します
　デフォルトは 1 です
**Refresh**
　フルスクリーンモードの場合のリフレッシュレートを指定します
　デフォルトは 60 です
**dispMode**
　フルスクリーンモードの場合の画面モードを指定します
　R5G6B5 の場合は 0、X8R8G8B8 は 1、現在の画面モードを維持する場合は-1 を指定します

デフォルトは-1 です

swapEffect

画面フリップの際の動作を以下の３つから指定します

D3DSWAPEFFECT_DISCARD　最も効率的なスワップチェーンをドライバが決定

D3DSWAPEFFECT_FLIP　バックバッファとフロントバッファをフリップする

D3DSWAPEFFECT_COPY　バックバッファをフロントバッファにコピーする

デフォルトは D3DSWAPEFFECT_COPY で、これ以外が指定されている場合、フリップ関数である drawScene 関数でコピーする矩形を指定できないのでデフォルトで使用する必要があります

**adapter**

使用するディスプレイアダプタ(グラフィックカード)が複数枚ある場合に使用するディスプレイアダプタを指定します

デフォルトは D3DADAPTER_DEFAULT で、デフォルトのディスプレイアダプタが使用される様に指定されています

**stencil**

ステンシルバッファを使用する場合は true、使用しない場合は false を指定します

デフォルトでは false が指定されています

## addScene

### 宣言

bool addScene(int scene,HWND hwnd,int width,int height,int sync=1,int refresh=60,int dispMode=-1,D3DSWAPEFFECT swapEffect=D3DSWAPEFFECT_COPY,UINT adapter=D3DADAPTER_DEFAULT);

### 説明

マルチウインドウ等、複数のウインドウを制御する際にフレームバッファを追加します

### 引数

**scene**

追加するシーン番号を指定します

0 番はメインのシーンになるので 1～7 番を指定してください

**hwnd**

追加されるウインドウのウインドウハンドルを指定します

**width**

追加されるシーンのフレームバッファの幅を指定します

**height**

追加されるシーンのフレームバッファの高さを指定します

**sync**

フレームバッファのフリップ時に垂直同期を待つ場合は 1、待たない場合は 0 を指定します

**refresh**

フルスクリーンモード（マルチモニター）の場合のリフレッシュレートを指定します

デフォルトは 60 です

**dispMode**

フルスクリーンモードの場合の画面モードを指定します

R5G6B5 の場合は 0、X8R8G8B8 は 1、現在の画面モードを維持する場合は-1 を指定します

デフォルトは-1 です

**swapEffect**

画面フリップの際の動作を以下の 3 つから指定します

D3DSWAPEFFECT_DISCARD　最も効率的なスワップチェーンをドライバが決定

D3DSWAPEFFECT_FLIP　バックバッファとフロントバッファをフリップする

D3DSWAPEFFECT_COPY　バックバッファをフロントバッファにコピーする

デフォルトは D3DSWAPEFFECT_COPY で、これ以外が指定されている場合、フリッ

プ関数である drawScene 関数でコピーする矩形を指定できないのでデフォルトで使用する必要があります

**adapter**

使用するディスプレイアダプタ(グラフィックカード)が複数枚ある場合に使用するディスプレイアダプタを指定します

デフォルトは D3DADAPTER_DEFAULT で、デフォルトのディスプレイアダプタが使用される様に指定されています

戻値

シーンの追加に成功した場合は true

失敗した場合は false

## selectScene

宣言

```
bool selectScene(int scene);
```

説明

addScene で複数のシーンを追加している場合、描画対象になるシーンを切り替えます

引数

**scene**

0～7 を指定します

0 はメインのシーンです

戻値

シーンの切り替えに成功した場合は true

失敗した場合は false

## deviceResetFunc;

### 宣言

void deviceResetFunc(void(*cleanFunc)(void)=NULL,void(*initFunc)(void)=NULL);

### 説明

DirectX がデバイスの消失などによりリセットする必要がある場合に処理する関数を指定します

通常は tsDirectGraphics 内で自動的に処理されるのでこの関数で登録する必要はありませんが、ユーザーが独自に確保したテクスチャポインタなどがある場合は、リセット動作のためのクリーンアップ及びリセット後の初期化処理を記述した関数を登録してください

### 引数

**cleanFunc**

デバイスリセットに必要なリソースのクリーンアップを記述した関数

**initFunc**

デバイスリセット後にリソースの初期化を記述した関数

### 戻値

無し

## deviceClean

### 宣言

void deviceClean(void);

### 説明

DirectX がデバイスを消失等によりリセットする必要がある場合に呼び出されます

通常は tsDirectGraphics 内で自動的に呼び出されるのでユーザーが意識する必要はありません

### 引数

無し

### 戻値

無し

## deviceInit

### 宣言

bool deviceInit(void);

### 説明

DirectX がデバイスを消失等によりリセット後の初期化動作をする場合に呼び出されます
通常は tsDirectGraphics 内で自動的に呼び出されるのでユーザーが意識する必要はありません

### 引数

無し

### 戻値

初期化に成功した場合は true
失敗した場合は false

## resizeFrameBuffer

### 宣言

bool resizeFrameBuffer(int width,int height);

### 説明

フレームバッファのリサイズを行います
ウインドウのリサイズイベント等が発生した場合に呼び出してください
※この関数はデバイスのリセット動作を行います

### 引数

**width**
　フレームバッファの新しい幅を指定します
**height**
　フレームバッファの新しい高さを指定します

### 戻値

フレームバッファのリサイズに成功した場合は true
失敗した場合は false

## changeWindowMode

### 宣言

bool changeWindowMode(int mode=-1,UINT refresh=D3DPRESENT_RATE_DEFAULT,int dispMode=-1)

### 説明

ウインドウモードとフルスクリーンモードを切り替える場合に呼び出します
※この関数は DirectX のステータスの切り替えのみ行います
ウインドウ自体の切り替えは、メニューの有る無しなどに依存するためユーザー側で行ってください

### 引数

**mode**

0：ウインドウモード
1：フルスクリーンモード
-1(default)：ウインドウモード⇔フルスクリーンモードで自動切り替え

**refresh**

フルスクリーンモード（マルチモニター）の場合のリフレッシュレートを指定します
デフォルトは 60 です

**dispMode**

フルスクリーンモードの場合の画面モードを指定します
R5G6B5 の場合は 0、X8R8G8B8 は 1、現在の画面モードを維持する場合は-1 を指定します
デフォルトは-1 です

### 戻値

ウインドウモードの切り替えに成功した場合は true
失敗した場合は false を返します

## clear

### 宣言

void clear(D3DCOLOR color=D3DCOLOR_XRGB(0,0,0),int rectNum=0,D3DRECT *rect=NULL,DWORD flag=D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER);

### 説明

フレームバッファ及び深度バッファをクリアします

### 引数

**color**

クリア色を指定します

デフォルトは黒です

**rectNum**

クリアする範囲の数を指定します

デフォルトではフレームバッファ全体をクリアするため 1 が指定されています

**rect**

クリアする矩形範囲を指定する D3DRECT 構造体の配列を指定します

rectNum との組み合わせで、フレームバッファをクリアする範囲を複数指定する事ができます

デフォルトは画面全体です

**flag**

クリアするバッファを指定します

デフォルトではフレームバッファと深度バッファをクリアします

### 戻値

無し

## clearZ

### 宣言

void clearZ(int rectNum=0,D3DRECT *rect=NULL,float z=1.0f);

### 説明

深度バッファのみクリアします

### 引数

**rectNum**

クリアする範囲の数を指定します

デフォルトではフレームバッファ全体をクリアするため 1 が指定されています

**rect**

クリアする矩形範囲を指定する D3DRECT 構造体の配列を指定します

rectNum との組み合わせで、フレームバッファをクリアする範囲を複数指定する事ができます

デフォルトは画面全体です

**z**

クリアする Z 値を指定します

デフォルトは 1.0 です

### 戻値

無し

## clearStencil

### 宣言

void clearStencil(int rectNum=0,D3DRECT *rect=NULL,DWORD stencil=0);

### 説明

ステンシルバッファをクリアします

### 引数

**rectNum**

クリアする範囲の数を指定します

デフォルトではフレームバッファ全体をクリアするため 1 が指定されています

**rect**

クリアする矩形範囲を指定する D3DRECT 構造体の配列を指定します

rectNum との組み合わせで、フレームバッファをクリアする範囲を複数指定する事ができます

デフォルトは画面全体です

**stencil**

クリアする値

### 戻値

無し

## beginScene

### 宣言

void beginScene(void);

### 説明

シーンの描画を開始します

全ての描画関数は必ず beginScene と endScene の間で実行します

### 引数

無し

### 戻値

無し

## endScene

### 宣言

void endScene(void);

### 説明

シーンの描画を終了します

### 引数

無し

### 戻値

無し

## drawScene

### 宣言

bool drawScene(RECT *srect=NULL,RECT *drect=NULL);

### 説明

バックバッファをフリップして画面に表示します
矩形を指定して一部のみ表示する事もできます
関数実行時にデバイスが消失している場合は自動的にデバイスのリセット及び管理リソースの復帰を行います
この際に deviceResetFunc 関数によって解放及び初期化用の関数が登録されている場合は呼び出されます
また、デバイスの消失によって関数が失敗した場合は false が帰りますので、この戻り値によって処理する事もできます

### 引数

**srect**
コピー元の矩形
デフォルトではフレームバッファ全体です

**drect**
コピー先の矩形
デフォルトでは画面全体です

### 戻値

関数が成功した場合は true、失敗した場合は false が返ります
関数が失敗するのは主に、スクリーンセーバーの起動などにより DirectX がデバイスを消失した場合です
この場合は管理しているリソースに対して自動的に復帰処理が行われます

## setViewport

### 宣言

bool setViewport(void);

### 説明

ビューポートをデフォルト設定にします

デフォルトは描画範囲がフレームバッファ全体、Z 深度が 0～1 の状態です

### 引数

無し

### 戻値

ビューポートの設定に成功した場合は true

失敗した場合は false を返します

## setViewport

### 宣言

bool setViewport(D3DVIEWPORT9 *viewport);

### 説明

ユーザー指定のビューポートを指定します

### 引数

**viewport**

ビューポート

### 戻値

ビューポートの設定に成功した場合は true

失敗した場合は false を返します

## backClip

### 宣言

void backClip(int state);

### 説明

裏面クリッピング（カリング）の有効/無効を指定します

２Ｄの場合はほとんど指定する必要はありません

### 引数

**state**

0:クリッピング無し

1:右回りでクリッピング

2:左回りでクリッピング

### 戻値

無し

## enableZ

### 宣言

void enableZ(bool state);

### 説明

Ｚチェックの有効/無効を指定します

### 引数

**state**

true：有効

false：無効

### 戻値

無し

## enableMask

### 宣言

void enableMask(void);

### 説明

アルファチャンネルによるマスクを有効にします
通常は透過部分を持ったテクスチャを表示する事が大半なので、この関数は常に呼び出しておいた方が良いです
基本的には tsDirectGraphics 初期化後の一度呼び出せば良いですが、この設定はデバイスのリセットによって初期化されるため毎フレーム呼び出した方が良いでしょう

### 引数

無し

### 戻値

無し

## setAlphaState

### 宣言

void setAlphaState(int mode);

### 説明

半透明処理を指定します

### 引数

**mode**

0：通常表示
1：半透明
2：加算半透明
3：減算半透明

### 戻値

無し

## setAlphaTest

### 宣言

void setAlphaTest(bool flag,DWORD alpha=0);

### 説明

アルファテストの有効/無効の指定をします

### 引数

**flag**

true：アルファテストを有効にする

false：アルファテストを無効にする

**alpha**

アルファテストに合格するしきい値

### 戻値

無し

## setZWrite

### 宣言

void setZWrite(bool flag);

### 説明

深度バッファへの書き込みの有効/無効を指定します

### 引数

flag

true：深度バッファへの書き込みを有効にします

false：深度バッファへの書き込みを無効にします

### 戻値

無し

## setTextureFilter

### 宣言

void setTextureFilter(DWORD filter=D3DTEXF_LINEAR);

### 説明

描画時のテクスチャフィルタを指定します

### 引数

**filter**

D3DTEXTUREFILTERTYPE 列挙体から選択してください
それぞれの意味は DirectX SDK のドキュメントを参照してください
デフォルトは D3DTEXF_LINEAR でバイリニアルフィルタです

### 戻値

無し

## setPointSize

### 宣言

void setPointSize(float size=1.0f);

### 説明

drawLine 時の線の太さを指定します

### 引数

**size**

線の太さ
デフォルトは 1.0f で 1 ドットです

### 戻値

無し

## selectTexture

### 宣言

void selectTexture(int stage,int num,bool rend=false);

### 説明

指定したテクスチャステージに指定したテクスチャを割り当てます…が、
tsDirectGraphics の drawPolygon 関数などはテクスチャ番号を指定して直接割り当てる事ができるのであまり使う必要はありません
テクスチャステージ間のオペレーションを指定し、ユーザーが直接 DrawPrimitive 関数などを呼び出して特殊な描画をする場合などに呼び出してください

### 引数

**state**

テクスチャステージ番号

**num**

テクスチャ管理番号

**rend**

レンダリングテクスチャを使用する

### 戻値

無し

## getDevice

### 宣言

LPDIRECT3DDEVICE9 getDevice(void);

### 説明

DIRECT3DDEVICE9 へのポインタを取得します
tsDirectGraphics に実装されていない機能を使う場合はこの関数でデバイスを取得して直接、利用したいメソッドを呼び出してください

### 引数

無し

### 戻値

DIRECT3DDEVICE9 へのポインタ

## getPParam

### 宣言

D3DPRESENT_PARAMETERS *getPParam(void);

### 説明

selectScene 関数で選択されているシーンの D3DPRESENT_PARAMETERS を取得します

### 引数

無し

### 戻値

D3DPRESENT_PARAMETERS へのポインタ

### getSelectScene

宣言

int getSelectScene(void);

説明

selectScene 関数で選択されているシーンの番号を返します

引数

無し

戻値

無し

### getTexture

宣言

LPDIRECT3DTEXTURE9 *getTexture(int texture,bool rend=false);

説明

tsDirectGraphics が管理しているテクスチャへのポインタを返します

引数

**texture**

テクスチャ管理番号

**rend**

レンダリングテクスチャを指定

戻値

DIRECT3DTEXTURE9 へのポインタ

## loadTexture

### 宣言

bool loadTexture(int num,LPCTSTR fname,D3DCOLOR colKey=0,D3DFORMAT format=D3DFMT_UNKNOWN,D3DXIMAGE_INFO *srcInfo=NULL,DWORD filter=D3DX_FILTER_LINEAR,UINT mipLevel=1/*D3DX_DEFAULT*/,int w=0,int h=0,D3DPOOL pool=D3DPOOL_MANAGED);

### 説明

ファイル名を指定して、ファイルからテクスチャを読み込みます

### 引数

**num**

テクスチャ管理番号(0～16383)

fname

ファイル名

**colKey**

カラーキーを指定すると、指定した色を透過色にする事ができます

アルファチャンネルで透過させている場合は必要ありません

デフォルトは無効の 0 が指定されています

**format**

読み込んだ後のテクスチャフォーマットを指定します

デフォルトは画像ファイルから決められる D3DFMT_UNKNOWN が指定されます

**srcInfo**

元画像の情報を取得する D3DXIMAGE_INFO へのポインタを指定します

元画像の情報が必要ない場合は NULL を指定できます

**filter**

画像を読み込んだ際に DirectX が画像サイズを２のべき乗になる様に調整します

この際に使用されるフィルタを指定します

デフォルトは D3DX_FILTER_LINEAR が指定されており、バイリニアフィルタによる補完がされます

**mipLevel**

テクスチャ読み込み時に作成されるミップマップレベルを指定します

２Ｄではミップマップは殆ど必要ないので１が指定されています

**w**

作成されるテクスチャの幅を指定できます

デフォルトは画像サイズです(２のべき乗に調整される)

**h**

作成されるテクスチャの高さを指定できます

デフォルトは画像サイズです(２のべき乗に調整される)

**pool**

テクスチャメモリの管理方式

デフォルトは D3DPOOL_MANAGED が指定されておりメモリの管理は DirectX が行います

| 戻値 |
| --- |

テクスチャの読み込みが成功した場合は true

失敗した場合は false を返します

## loadTexture

### 宣言

bool loadTexture(int num,void *ptr,UINT size,D3DCOLOR colKey=0,D3DFORMAT format=D3DFMT_UNKNOWN,D3DXIMAGE_INFO *srcInfo=NULL,DWORD filter=D3DX_FILTER_LINEAR,UINT mipLevel=1/*D3DX_DEFAULT*/,int w=0,int h=0,D3DPOOL pool=D3DPOOL_MANAGED);

### 説明

メモリ上のデータアドレスとデータサイズを指定して、メモリ上からテクスチャを読み込みます
BITMAPINFOHEARER など画像データの本体アドレスを指定します

### 引数

**num**
　テクスチャ管理番号(0～16383)

ptr
　画像データが格納されているポインタ

**aize**
　画像データ本体のサイズ

**colKey**
　カラーキーを指定すると、指定した色を透過色にする事ができます
　アルファチャンネルで透過させている場合は必要ありません
　デフォルトは無効の 0 が指定されています

**format**
　読み込んだ後のテクスチャフォーマットを指定します
　デフォルトは画像ファイルから決められる D3DFMT_UNKNOWN が指定されます

**srcInfo**
　元画像の情報を取得する D3DXIMAGE_INFO へのポインタを指定します
　元画像の情報が必要ない場合は NULL を指定できます

**filter**
　画像を読み込んだ際に DirectX が画像サイズを 2 のべき乗になる様に調整します
　この際に使用されるフィルタを指定します
　デフォルトは D3DX_FILTER_LINEAR が指定されており、バイリニアフィルタによる補完がされます

**mipLevel**
　テクスチャ読み込み時に作成されるミップマップレベルを指定します

２Ｄではミップマップは殆ど必要ないので１が指定されています

**w**

作成されるテクスチャの幅を指定できます

デフォルトは画像サイズです(２のべき乗に調整される)

**h**

作成されるテクスチャの高さを指定できます

デフォルトは画像サイズです(２のべき乗に調整される)

**pool**

テクスチャメモリの管理方式

デフォルトは D3DPOOL_MANAGED が指定されておりメモリの管理は DirectX が行います

| 戻値 |
| --- |

テクスチャの読み込みが成功した場合は true

失敗した場合は false を返します

## loadTexture

### 宣言

bool loadTexture(int num,LPD3DXBUFFER ptr,D3DCOLOR colKey=0,D3DFORMAT format=D3DFMT_UNKNOWN,D3DXIMAGE_INFO *srcInfo=NULL,DWORD filter=D3DX_FILTER_LINEAR,UINT mipLevel=1/*D3DX_DEFAULT*/,int w=0,int h=0,D3DPOOL pool=D3DPOOL_MANAGED);

### 説明

メモリ上に保存された画像ファイルのアドレスを指定して、メモリ上からテクスチャを読み込みます

BITMAPFILEHEADER など画像ファイル自体のアドレスを指定します

### 引数

**num**

テクスチャ管理番号(0～16383)

**ptr**

画像ファイルが格納されているメモリへのポインタ

**colKey**

カラーキーを指定すると、指定した色を透過色にする事ができます

アルファチャンネルで透過させている場合は必要ありません

デフォルトは無効の 0 が指定されています

**format**

読み込んだ後のテクスチャフォーマットを指定します

デフォルトは画像ファイルから決められる D3DFMT_UNKNOWN が指定されます

**srcInfo**

元画像の情報を取得する D3DXIMAGE_INFO へのポインタを指定します

元画像の情報が必要ない場合は NULL を指定できます

**filter**

画像を読み込んだ際に DirectX が画像サイズを２のべき乗になる様に調整します

この際に使用されるフィルタを指定します

デフォルトは D3DX_FILTER_LINEAR が指定されており、バイリニアフィルタによる補完がされます

**mipLevel**

テクスチャ読み込み時に作成されるミップマップレベルを指定します

２Ｄではミップマップは殆ど必要ないので１が指定されています

**w**

作成されるテクスチャの幅を指定できます
デフォルトは画像サイズです(２のべき乗に調整される)

**h**

作成されるテクスチャの高さを指定できます
デフォルトは画像サイズです(２のべき乗に調整される)

**pool**

テクスチャメモリの管理方式
デフォルトは D3DPOOL_MANAGED が指定されておりメモリの管理は DirectX が行います

| 戻値 |
| --- |

テクスチャの読み込みが成功した場合は true
失敗した場合は false を返します

## createTexture

### 宣言

bool createTexture(int num,D3DXIMAGE_INFO *imageInfo,D3DPOOL pool=D3DPOOL_MANAGED);

### 説明

新しい空のスチャを作成します

### 引数

**num**

テクスチャ管理番号

**imageInfo**

新しいテクスチャを作成する際のサイズ、フォーマット等指定する D3DXIMAGE_INFO へのポインタ

**pool**

テクスチャメモリの管理方式

デフォルトは D3DPOOL_MANAGED が指定されておりメモリの管理は DirectX が行います

### 戻値

テクスチャの作成が成功した場合は true

失敗した場合は false を返します

## releaseTexture

### 宣言

void releaseTexture(int num);

### 説明

テクスチャを解放します
tsDirectGraphics 終了時や同じテクスチャ番号に読み込みを行う場合には
tsDirectGraphics が自動的に解放しますが、アプリ中で巨大なテクスチャを大量に読み込み直したりしてビデオメモリが不足する様な場合はユーザーが任意に未使用テクスチャを解放してください

### 引数

**num**
　解放するテクスチャ管理番号

### 戻値

　無し

## checkTexture

### 宣言

bool checkTexture(int num);

### 説明

テクスチャ管理番号を指定して、使用中か空きかを調べる

### 引数

**num**
　使用中かを調べるテクスチャ管理番号

### 戻値

指定されたテクスチャが使用中なら true
未使用なら false

## createRenderTexture

### 宣言

bool createRenderTexture(int num,int w,int h,D3DFORMAT format=D3DFMT_UNKNOWN);

### 説明

レンダリング用テクスチャを作成します

### 引数

**num**

テクスチャ管理番号

**w**

作成されるレンダリングテクスチャの幅

**h**

作成されるレンダリングテクスチャの高さ

**format**

作成されるレンダリングテクスチャのフォーマット

デフォルトでは D3DFMT_UNKNOWN が指定されていて、フレームバッファのフォーマットに合わせたフォーマットを使用します

### 戻値

レンダリングテクスチャの作成に成功した場合は true

失敗した場合は false を返します

## releaseRenderTexture

### 宣言

void releaseRenderTexture(int num);

### 説明

レンダリングテクスチャを解放します

### 引数

**num**

　解放するレンダリングテクスチャの管理番号

### 戻値

無し

## selectRenderTexture

### 宣言

bool selectRenderTexture(int num);

### 説明

レンダリングターゲットを指定されたレンダリングテクスチャに切り替えます

-1 を指定するとレンダリングターゲットをフレームバッファに戻します

### 引数

**num**

　レンダリングターゲットに指定するレンダリングテクスチャの管理番号

### 戻値

レンダリングターゲットに切り替えに成功した場合は true

失敗した場合は false を返します

## beginSprite

### 宣言

void beginSprite(DWORD flag=D3DXSPRITE_ALPHABLEND);

### 説明

スプライト描画の開始を宣言します

endSprite 関数との間で呼び出されたスプライト描画の関数は、この関数で指定されたパラメータの影響を受けます

drawSprite 関数はこの関数が呼び出されていない場合は自動的に呼び出すので普段はあまり気にする必要はありません

この関数の影響を受けるのは drawSprite 関数及び drawText 関数です

### 引数

**flag**

アルファブレンドを使用する際の処理

デフォルトでは D3DXSPRITE_ALPHABLEND が指定されており、通常の半透明処理が行われます

### 戻値

無し

## endSprite

### 宣言

void endSprite(void);

### 説明

スプライト描画の終了を宣言します

beginSprite 関数を呼び出した場合は、必ずこの関数を呼び出してスプライトの描画を終了してください

### 引数

無し

### 戻値

無し

## setFont

### 宣言

bool setFont(int width,int height,LPCTSTR font,int style=0,DWORD charSet=SHIFTJIS_CHARSET);

### 説明

drawText 関数で使用するフォントを設定します

drawText 関数を使用する前に必ずこの関数でフォントの設定をしてください

### 引数

width

　フォントの幅

height

　フォントの高さ

font

　使用するフォント名

　(例：”ＭＳ　ゴシック”)などコンピュータにインストールされているフォントを指定してください

style

　フォントスタイル

　0:通常

　1:太文字

　2:極太

　3:極々太

　4～7:+斜体

charset

　使用するキャラクターセット

　デフォルトでは SHIFTJIS_CHARSET が使用されます

### 戻値

フォントの選択に成功した場合は true

失敗した場合は false を返します

## drawText

### 宣言

void drawText(int x,int y,D3DCOLOR color,LPCTSTR str,...);

### 説明

文字列を描画します

使用する前に必ず setFont 関数でフォントの設定を行ってください

### 引数

**x**

表示するＸ座標

**y**

表示するＹ座標

**color**

文字列の色

**str**

表示する文字列(printf に準拠します)

**...**

追加引数

### 戻値

無し

## drawText

### 宣言

void drawText(LPRECT rect,D3DCOLOR color,DWORD format,LPCTSTR str,...);

### 説明

矩形を指定して文字列を描画します

使用する前に必ず setFont 関数でフォントの設定を行ってください

### 引数

**rect**

描画範囲を指定する矩形へのポインタ

color

文字列の色

format

表示方法を指定する DT_***の組み合わせ

DT_LEFT,DT_CENTER,DT_RIGHT,DT_TOP,DT_VCENTER,DT_BOTTOM,DT_NCLIP,DT_WORD_ELLIPSIS,DT_WORDBREAK があります

詳しくは DirectX9 SDK のドキュメントを参照してください

**str**

表示する文字列(printf に準拠します)

**…**

追加引数

### 戻値

無し

## drawSprite

### 宣言

void drawSprite(int num,D3DXVECTOR2 *pos,RECT *rect=NULL,D3DXVECTOR2 *scale=NULL,D3DXVECTOR2* center=NULL,float rot=0,D3DCOLOR color=0xffffffff,bool rend=false);

### 説明

テクスチャ番号を指定してスプライトを描画します

### 引数

**num**

表示するテクスチャ管理番号

**pos**

表示する座標を指定する D3DXVECTOR2 へのポインタ

**rect**

使用するテクスチャの範囲を指定する事でテクスチャの一部をスプライトとして描画できます

デフォルトは NULL が指定されており、テクスチャ全体を使用します

**scale**

描画するスケーリング値を指定します

デフォルトは NULL が指定されており元のサイズのまま描画されます

**center**

回転、拡縮、スケーリング、描画などの中心位置を指定します

デフォルトは NULL が指定されており、テクスチャの左上を原点にします

**rot**

回転角度の（度）で指定します

デフォルトは 0 です

**color**

スプライト描画時の色を指定します

この値を設定する事で色調変換や透過処理が行えます

デフォルトは D3DCOLOR_XRGB(255,255,255)が指定されており、元画像がそのまま表示されます

**rend**

true を指定するとレンダリングテクスチャを使用します

| 戻値 |
| --- |
| 無し |

## drawSprite

### 宣言

void drawSprite(LPDIRECT3DTEXTURE9 texture,D3DXVECTOR2 *pos,RECT *rect=NULL,D3DXVECTOR2 *scale=NULL,D3DXVECTOR2* center=0,float rot=0,D3DCOLOR color=0xffffffff);

### 説明

テクスチャへのポインタを指定してスプライトを描画します
おもにユーザーが独自に取得したテクスチャを描画する際に使用します

### 引数

**texture**

DIRECT3DTEXTURE9 へのポインタ

**pos**

表示する座標を指定する D3DXVECTOR2 へのポインタ

**rect**

使用するテクスチャの範囲を指定する事でテクスチャの一部をスプライトとして描画できます

デフォルトは NULL が指定されており、テクスチャ全体を使用します

**scale**

描画するスケーリング値を指定します

デフォルトは NULL が指定されており元のサイズのまま描画されます

**center**

回転、拡縮、スケーリング、描画などの中心位置を指定します

デフォルトは NULL が指定されており、テクスチャの左上を原点にします

**rot**

回転角度の（度）で指定します

デフォルトは 0 です

**color**

スプライト描画時の色を指定します

この値を設定する事で色調変換や透過処理が行えます

デフォルトは D3DCOLOR_XRGB(255,255,255)が指定されており、元画像がそのまま表示されます

### 戻値

無し

## drawLine

### 宣言

void drawLine(int num,WDVERTEX *vtx);

### 説明

ラインを描画します

頂点を２個以上指定した場合は、連続した直線を描画します

### 引数

**num**

描画する頂点数

**vtx**

頂点配列(WDVERTEX)へのポインタ

### 戻値

無し

## drawPolygon

### 宣言

void drawPolygon(int num,WDVERTEX *vtx);

### 説明

テクスチャ無しポリゴンを描画します

### 引数

**num**

描画する頂点数

**vtx**

頂点配列(WDVERTEX)へのポインタ

### 戻値

無し

## drawPolygon

### 宣言

void drawPolygon(int num,int texture,WDTVERTEX *vtx,bool rend=false);

### 説明

テクスチャ番号を指定して、テクスチャ付ポリゴンを描画します

### 引数

**num**

描画する頂点数

**texture**

テクスチャ番号

**vtx**

頂点配列(WDTVERTEX)へのポインタ

**rend**

true の場合はレンダリングテクスチャを使用する

### 戻値

無し

## drawPolygon

### 宣言

void drawPolygon(int num,LPDIRECT3DTEXTURE9 texture,WDTVERTEX *vtx);

### 説明

テクスチャデータ(DIRECT3DTEXTURE9)へのポインタを指定してポリゴンを描画します

### 引数

**num**

描画する頂点数

**texture**

テクスチャデータ(DIRECT3DTEXTURE9)へのポインタ

**vtx**

頂点配列(WDTVERTEX)へのポインタ

### 戻値

無し

## draw2DLine

### 宣言

void draw2DLine(int num,D3DXVECTOR2 *vtx,D3DCOLOR color,float width=1,DWORD pattern=0xffffffff);

### 説明

２Ｄラインを描画する

頂点を２個以上指定した場合は、連続した直線を描画します

### 引数

**num**

描画する頂点数

**vtx**

頂点配列(D3DXVECTOR2)へのポインタ

**color**

描画色

**width**

線の太さ

デフォルトは１です

### 戻値

無し

## saveFrameBuffer

### 宣言

bool saveFrameBuffer(LPCTSTR fName,D3DXIMAGE_FILEFORMAT format=D3DXIFF_BMP);

### 説明

フレームバッファをファイルに保存します

### 引数

**fName**

保存するファイル名

**format**

保存するフォーマット

デフォルトは D3DXIFF_BMP で、bmp 形式で保存されます

### 戻値

保存に成功した場合は true

失敗した場合は false を返します

## saveFrameBuffer

### 宣言

bool saveFrameBuffer(LPD3DXBUFFER &mptr,D3DXIMAGE_FILEFORMAT format=D3DXIFF_BMP);

### 説明

フレームバッファをメモリに保存します

### 引数

**mptr**

保存するメモリへのポインタ

**format**

保存するフォーマット

デフォルトは D3DXIFF_BMP で、bmp 形式で保存されます

### 戻値

保存に成功した場合は true

失敗した場合は false を返します

## saveTexture

### 宣言

bool saveTexture(int num,LPCTSTR fName,RECT *rect,D3DXIMAGE_FILEFORMAT format=D3DXIFF_BMP,bool rend=false);

### 説明

テクスチャをファイルに保存します

### 引数

**num**

テクスチャ番号

**fName**

保存するファイル名

**rect**

保存範囲を指定する RECT 構造体へのポインタ

デフォルトは NULL でテクスチャ全体です

**format**

保存するフォーマット

デフォルトは D3DXIFF_BMP で、BMP 形式で保存されます

**rend**

true を指定するとレンダリングテクスチャを保存します

### 戻値

保存に成功した場合は true

失敗した場合は false を返します

## saveTexture

### 宣言

bool saveTexture(int num,LPD3DXBUFFER &mptr,RECT *rect=NULL,D3DXIMAGE_FILEFORMAT format=D3DXIFF_BMP,bool rend=false);

### 説明

テクスチャをメモリに保存します

### 引数

**num**

テクスチャ番号

**mptr**

保存するメモリへのポインタ

**rect**

保存範囲を指定する RECT 構造体へのポインタ

デフォルトは NULL でテクスチャ全体です

**format**

保存するフォーマット

デフォルトは D3DXIFF_BMP で、BMP 形式で保存されます

**rend**

true を指定するとレンダリングテクスチャを保存します

### 戻値

保存に成功した場合は true

失敗した場合は false を返します

## setAlphaStateEx

### 宣言

void setAlphaStateEx(int mode);

### 説明

半透明処理を指定します

Adobe の AfterEffects の透過処理に準拠した透過処理を実現しますが、完全には再現できません

AfterEffects の完全な再現をするにはピクセルシェーダーによるオペレーションが必要です

### 引数

**mode**

0:通常（ブレンドなし）○

1:半透明○

2:加算合成○

3:加算半透明合成○

4:減算合成○

5:乗算合成○

6:乗算合成 2

7:スクリーン合成○

8:リバース(除外)○

9:覆い焼き（リニア）○

10:較(明)○

11:比較(暗)

12:覆い焼き（カラー)

13:減算(ソース-デスティネーション)

14:減算(デスティネーション-ソース)

### 戻値

無し

## loadEffect

### 宣言

bool loadEffect(LPCTSTR fName,int num=0);

### 説明

エフェクトファイルをロードします

ファイル名の拡張子によって内部的に loadEffectObj 関数か loadEffectFx 関数を呼び出します

### 引数

**fName**

エフェクトファイルのファイル名

**num**

エフェクト管理番号(1～7)

デフォルトは 0 です

### 戻値

エフェクトファイルの読み込みに成功した場合は true

失敗した場合は false を返します

## loadEffectObj

### 宣言

bool loadEffectObj(LPCTSTR fName,int num=0);

### 説明

コンパイル済みエフェクトファイル(拡張子 obj)を読み込みます

### 引数

**fName**

コンパイル済みエフェクトファイルのファイル名

**num**

エフェクト管理番号(1～7)

デフォルトは 0 です

### 戻値

エフェクトファイルの読み込みに成功した場合は true

失敗した場合は false を返します

## loadEffectFx

### 宣言

bool loadEffectFx(LPCTSTR fName,int num=0);

### 説明

エフェクトファイル(拡張子 fx)を読み込みます

### 引数

**fName**

エフェクトファイルのファイル名

**num**

エフェクト管理番号(1～7)

デフォルトは 0 です

### 戻値

エフェクトファイルの読み込みに成功した場合は true

失敗した場合は false を返します

### releaseEffect

宣言

void releaseEffect(int num = 0);

説明

エフェクトを解放します

引数

**num**

エフェクト管理番号(1～7)

デフォルトは 0 です

戻値

無し

### getEffect

宣言

LPD3DXEFFECT getEffect(int num=0);

説明

エフェクトのポインタを取得します

引数

**num**

エフェクト管理番号(1～7)

デフォルトは 0 です

戻値

エフェクトへのポインタ(LPD3DXEFFECT)

## getTechnique

### 宣言

D3DXHANDLE getTechnique(LPCTSTR name,int num=0);

### 説明

指定されたエフェクト内のテクニックのハンドルを取得します

### 引数

**name**

テクニック名

**num**

エフェクト管理番号(1～7)

デフォルトは 0 です

### 戻値

テクニックのハンドル

## getParameter

### 宣言

D3DXHANDLE getParameter(LPCTSTR name,int num=0,D3DXHANDLE param=NULL);

### 説明

パラメータのハンドルを取得する

### 引数

**name**

テクニック名

**num**

エフェクト管理番号

**param**

パラメータ

### 戻値

パラメータのハンドル

## selectTechnique

### 宣言

bool selectTechnique(D3DXHANDLE handle,int num=0);

### 説明

テクニックを選択して開始する

### 引数

**handle**

開始するテクニックのハンドル

**num**

エフェクト管理番号

デフォルトは 0

### 戻値

テクニックの開始に成功した場合は true

失敗した場合は false を返します

## endTechnique

### 宣言

bool endTechnique(int num=0);

### 説明

テクニックを終了する

### 引数

**num**

エフェクト管理番号

デフォルトは 0

### 戻値

指定したテクニックの終了に成功した場合は true

失敗した場合は false を返します

## selectPass

### 宣言

bool selectPass(UINT pass,int num=0);

### 説明

実行するシェーダーパスを指定する

### 引数

**pass**

　実行するパス

num

　エフェクト管理番号

### 戻値

パスの選択に成功した場合は true

失敗した場合は false を返します

## endPass

### 宣言

bool endPass(int num=0);

### 説明

実行中のシェーダーパスを終了する

### 引数

**num**

　エフェクト管理番号

### 戻値

シェーダーパスを終了に成功した場合は true

失敗した場合は false を返します

## drawRect

### 宣言

void drawRect(RECT rect,D3DCOLOR color);

### 説明

矩形を描画します

### 引数

**rect**

描画する矩形

**color**

描画色

### 戻値

無し

## drawBrokenRect

### 宣言

void drawBrokenRect(RECT rect,int ofs,D3DCOLOR color);

### 説明

破線の矩形を描画します

### 引数

**rect**

描画する矩形

**ofs**

破線を描画する際のオフセット

この値を変化させる事で破線のアニメーションができます

**color**

描画色

### 戻値

無し

## drawBrokenCircle

### 宣言

void drawBrokenCircle(D3DXVECTOR2 *pos,int r,int ofs,D3DCOLOR color);

### 説明

破線で円を描画します

### 引数

**pos**

描画する円の中心

**r**

描画する円の半径

**ofs**

破線を描画する際のオフセット

この値を変化させる事で破線のアニメーションができます

**color**

描画色

### 戻値

無し

## drawBrokenTextureEdge

### 宣言

void drawBrokenTextureEdge(D3DXVECTOR2 *pos,int texture,int ofs,DWORD color,D3DXVECTOR2 *scale=NULL,D3DXVECTOR2* center=NULL,float rot=0);

### 説明

テクスチャ番号を指定してテクスチャの周りに破線の選択領域を描画します

### 引数

**pos**

表示座標

**texture**

選択範囲を作成するテクスチャの管理番号

**ofs**

破線を描画する際のオフセット

この値を変化させる事で破線のアニメーションができます

**color**

破線の色

**scale**

元テクスチャの拡縮率

デフォルトは NULL でスケーリング無し

**center**

元テクスチャの中心

デフォルトは NULL で左上原点

**rot**

元テクスチャの回転角度

デフォルトは 0 で回転無し

### 戻値

無し

## drawBrokenTextureEdge

### 宣言

void drawBrokenTextureEdge(D3DXVECTOR2 *pos,LPDIRECT3DTEXTURE9 texture,int ofs,DWORD color,D3DXVECTOR2 *scale=NULL,D3DXVECTOR2* center=NULL,float rot=0);

### 説明

テクスチャデータへのポインタを指定して、テクスチャの周りに破線の選択領域を描画します

### 引数

**pos**

表示座標

**texture**

選択範囲を作成するテクスチャへのポインタ(LPDIRECT3DTEXTURE9)

**ofs**

破線を描画する際のオフセット

この値を変化させる事で破線のアニメーションができます

**color**

破線の色

**scale**

元テクスチャの拡縮率

デフォルトは NULL でスケーリング無し

**center**

元テクスチャの中心

デフォルトは NULL で左上原点

**rot**

元テクスチャの回転角度

デフォルトは 0 で回転無し

### 戻値

無し

## getTexturePixel

### 宣言

D3DCOLOR getTexturePixel(int texture,int x,int y);

### 説明

テクスチャ番号を指定して、指定されたテクスチャのピクセル色を取得します

### 引数

**texture**

ピクセル色を取得するテクスチャの管理番号

**x**

ピクセル色を取得するＸ座標

**y**

ピクセル色を取得するＹ座標

### 戻値

ピクセル色

## getTexturePixel

### 宣言

D3DCOLOR getTexturePixel(LPDIRECT3DTEXTURE9 texture,int x,int y);

### 説明

テクスチャデータへのポインタを指定して、指定されたテクスチャのピクセル色を取得します

### 引数

**texture**

ピクセル色を取得するテクスチャへのポインタ(LPDIRECT3DTEXTURE9)

x

ピクセル色を取得するX座標

**y**

ピクセル色を取得するY座標

### 戻値

ピクセル色

## getTextureSize

### 宣言

POINT getTextureSize(int texture,bool rend=false);

### 説明

テクスチャ番号を指定して、テクスチャのサイズを取得する

### 引数

**texture**

サイズを取得するテクスチャ番号

**rend**

true の場合はレンダリングテクスチャのサイズを取得する

### 戻値

テクスチャのサイズ

## getTextureSize

### 宣言

POINT getTextureSize(LPDIRECT3DTEXTURE9 texture);

### 説明

テクスチャデータへのポインタを指定して、テクスチャのサイズを取得する

### 引数

**texture**

サイズを取得するテクスチャへのポインタ(LPDIRECT3DTEXTURE9)

### 戻値

テクスチャサイズ

## getTextureFormat

### 宣言

D3DFORMAT getTextureFormat(int texture,bool rend=false);

### 説明

テクスチャ番号を指定して、テクスチャのフォーマットを取得する

### 引数

**texture**

フォーマットを取得するテクスチャ番号

**rend**

true の場合はレンダリングテクスチャのサイズを取得する

### 戻値

テクスチャフォーマット(D3DFORMAT)

## getTextureFormat

### 宣言

D3DFORMAT getTextureFormat(LPDIRECT3DTEXTURE9 texture);

### 説明

テクスチャデータへのポインタを指定して、テクスチャのフォーマットを取得する

### 引数

**texture**

フォーマットを取得するテクスチャへのポインタ(LPDIRECT3DTEXTURE9)

### 戻値

テクスチャフォーマット(D3DFORMAT)

## 履歴

- 2015-03-10　1.0.0.0　　正式版リリース

## 連絡先

E-Mail:info@thinksheep.com
http://www.thinksheep.com/

## 最後に

使用中に不具合が発生した場合はや操作に関する不明点は、上記連絡先にご連絡ください