Xtreme3D 2.1

by Gecko

(new functions are highlighted with purple)

EngineCreate()
Creates the 3D engine. After creation other Xtreme3D commands can be called.
CHANGES FROM 2.0.X: window handle now is set with Viewer creation

EngineDestroy()
Destroys the 3D engine, freeing all resources. Don't call Xtreme3D functions after it.

EngineSetCulling(vc)
Sets default visibility culling for objects.
VisibilityCulling:

EngineSetObjectsSorting(os)
Sets sorting mode of objects.

Update(delta)
Updates animations, collisions and so on and resets TrisRendered variable. Usually called before the render function.
CHANGES FROM 2.0.X: added delta value to control the update step size

SetPakArchive(filename)
Activates a pak archive. The engine loads resources from this archive then.

ViewerCreate(x,y,width,height,window_handle)
Creates a viewer; a window that shows the 3D scene.
The function returns a viewer object which is used in other (viewer) functions.
CHANGES FROM 2.0.X: window handle now is set with Viewer creation

ViewerSetCamera(viewer,camera)
Sets/activates the camera for the viewer.
Without a camera the viewer does not show anything, so set one after creation.

ViewerEnableVSync(viewer,vsm)
Enables or disables Vertical Synchronization.

ViewerRender(viewer)
Renders the scene in the viewer through the camera.

ViewerResize(viewer,x,y,width,height)
Sets a new position and size of the viewer.

ViewerSetVisible(viewer,mode)
Shows or hides the viewer.

ViewerGetPixelColor(viewer,x,y)
Returns the pixel color at x, y.

ViewerGetPixelDepth(viewer,x,y)
Returns the pixel depth at x, y.

ViewerSetLighting(viewer,mode)
Mode: true for lights affecting the scene, false for not.

ViewerSetBackgroundColor(viewer,color)
Sets the background color of the viewer.

ViewerSetAmbientColor(viewer,color)
Set the viewer's ambient color.
This ambient color is defined independantly from all lightsources, which can have their own ambient components.

ViewerEnableFog(viewer,mode)
Enables/Disables fog (true/false)

ViewerSetFogColor(viewer,color)
Sets the color of the fog.

ViewerSetFogDistance(viewer,start,end)
start: Minimum distance for fog, what is closer is not affected.
end: Maximum distance for fog, what is farther is at 100% fog intensity.

ViewerScreenToWorld(viewer,x,y,ind)
Converts screen coordinates to 3D world coordinates and returns them.
x, y: screen coordinates
index: 0 returns x coordinate, 1 returns y coordinate, 2 returns z coordinate

ViewerWorldToScreen(viewer,x,y,z,ind)
Converts world coordinates to 2D screen coordinates and returns them.
x, y, z: world coordinates
index: 0 for x, 1 for y
screen coordinate 0,0 is the lower left corner

ViewerCopyToTexture(viewer,material)
Copies the viewer's rendered scene to a material's texture.

ViewerRenderToFile(viewer,filename,width,height)
Renders the scene in the viewer to external *.bmp file.
This function may be incompatible with some other viewer related effects.

ViewerGetFramesPerSecond(viewer)
Returns the viewer's FPS. You are recommended to use this function as it is more accurate than Game Maker's built-in "fps" variable.

ViewerGetPickedObject(viewer,x,y)
Returns the handle of an object that the (x,y) pixel belongs to.

ViewerGetPickedObjectsList(viewer,x,y,width,height,num,ind)
Takes the rectangular area on the screen and searches objects in there. Then it builds an object list consisiting of /num/ objects, and returns an object with a given index in the list.

ViewerScreenToVector(viewer,x,y,ind)
Returns a 3d vector from the viewer's camera position to a given screen point.
/index/ specifies a coordinate to return: 0=x, 1=y, 2=z.

ViewerVectorToScreen(viewer,x,y,z,ind)
Returns a screen point that the given 3d vector passes through. The vector is normalized and begins from the viewer's camera position.
/index/ specifies a coordinate to return: 0=x, 1=y.

ViewerPixelToDistance(viewer,x,y)
Returns the distance to pixel on the screen calculated from it's depth value.

ViewerSetAntiAliasing(viewer,aa)
Sets the anti-aliasing mode:
aaDefault = 0
aaNone = 1
aa2x = 2
aa2xHQ = 3
aa4x = 4
aa4xHQ = 5

DummycubeCreate(parent)
Returns a Dummycube object.
This is an empty, usually non-visible object used for building hierarchies or groups

DummycubeAmalgamate(Dummycube, mode)
Amalgamate the dummy's children in a single OpenGL entity.
This activates a special rendering mode, which will compile the rendering of all of the dummycube's children objects into a single display list. This may provide a significant speed up in some situations, however, this means that changes to the children will be ignored untill you call StructureChanged on the dummy cube.
Some objects, that have their own display list management, may not be compatible with this behaviour.
This will also prevents sorting and culling to operate as usual.
In short, this features is best used for static, non-transparent geometry, or when the point of view won't change over a large number of frames.

DummycubeSetCameraMode(Dummycube, CameraInvarianceMode)
These options allow to "deactivate" sensitivity to camera, f.i. by
centering the object on the camera or ignoring camera orientation.

DummycubeSetVisible(dummy,mode)
Toggles the dummycube visibility. Dummycube is shown as a wireframe cube.

DummycubeSetEdgeColor(dummy,color)
Sets the dummycube edge color.

DummycubeSetCubeSize(dummy,size)
Sets the dummycube size in world units. Default size is 1.

CameraCreate(parent)

Returns a camera object.

CameraSetStyle(camera, CameraStyle)

CameraStyle:
csPerspective, the default value for perspective projection
csOrthogonal, for orthogonal (or isometric) projection.
csOrtho2D, setups orthogonal 2D projection in which 1 unit (in x or y) represents 1 pixel.

CameraSetFocal(camera, FocalLength)

Focal Length of the camera
Adjusting this value allows for lens zooming effects (use SceneScale for linear zooming). This property affects near/far planes clipping.

CameraSetSceneScale(camera, scale)

Scene scaling for camera point
This is a linear 2D scaling of the camera's output, allows for linear zooming (use FocalLength for lens zooming).

CameraScaleScene(camera, step)

Scales it relative.

CameraSetViewDepth(camera, depth)

Adjusts the maximum distance, beyond which objects will be clipped(ie. not visisble).
You must adjust this value if you are experiencing disappearing objects (increase the value) or Z-Buffer crawling (decrease the value).
Z-Buffer crawling happens when depth of view is too large and the Z-Buffer precision cannot account for all that depth
accurately : objects farther overlap closer objects and vice-versa.
Note that this value is ignored in cSOrtho2D mode.

CameraSetTargetObject(camera, object)

If set, camera will point to this object.
When camera is pointing an object, the Direction vector is ignored and the Up vector is used as an absolute vector to the up.
0 for no target object.

CameraMoveAroundTarget (camera, pitch, turn)

Moves camera around the target object

CameraSetDistanceToTarget(camera, distance)

Adjusts the distance of the camera to the target object.

CameraGetDistanceToTarget(camera)

Returns the distance of the camera to the target object.

CameraCopyToTexture(camera, material, width, height)

Copies the camera's content to the material's textue.
The material applied to objects shows the scene of the camera.
camera: a camera object, does not have to be assigned to a viewer

material: material name
width, height: sizes must be power of 2

CameraGetNearPlane(camera)
Nearest clipping plane for the frustum.
This value depends on the Focal and ViewDepth fields and is calculated to minimize Z-Buffer crawling as suggested by the OpenGL documentation.

CameraSetNearPlaneBias(camera,bias)
Scaling bias applied to near-plane calculation.
Values inferior to one will move the nearplane nearer, and also reduce medium/long range Z-Buffer precision, values superior to one will move the nearplane farther, and also improve medium/long range Z-Buffer precision.

CameraAbsoluteVectorToTarget(camera,ind)
Computes the absolute normalized vector to the camera target.
If no target is defined, AbsoluteDirection is returned.

CameraAbsoluteRightVectorToTarget(camera,ind)
Computes the absolute normalized right vector to the camera target.
If no target is defined, AbsoluteRight is returned.

CameraAbsoluteUpVectorToTarget(camera,ind)
Computes the absolute normalized up vector to the camera target.
If no target is defined, AbsoluteUp is returned.

CameraZoomAll(camera)
Position the camera so that the whole scene can be seen

CameraScreenDeltaToVector(camera,deltax,deltay,ratio,normx,normy,normz,ind)
Calculate an absolute translation vector from a screen vector.
Ratio is applied to both screen delta, (normx, normy, normz) should be the translation plane's normal.

CameraScreenDeltaToVectorXY(camera,deltax,deltay,ratio,ind)
Same as ScreenDeltaToVector but optimized for XY plane.

CameraScreenDeltaToVectorXZ(camera,deltax,deltay,ratio,ind)
Same as ScreenDeltaToVector but optimized for XZ plane.

CameraScreenDeltaToVectorYZ(camera,deltax,deltay,ratio,ind)
Same as ScreenDeltaToVector but optimized for YZ plane.

CameraAbsoluteEyeSpaceVector(camera,fordist,rightdist,updist,ind)

CameraSetAutoLeveling(camera,factor)

CameraMoveInEyeSpace(camera,fordist,rightdist,updist)

CameraMoveTargetInEyeSpace(camera,fordist,rightdist,updist)

CameraPointInFront(camera,x,y,z)

CameraGetFieldOfView(camera)

LightCreate(LightStyle, parent)

Returns a light object. The standard Xtreme3D light source covers spotlights, omnidirectionnal and parallel sources. Lights are colored, have distance attenuation parameters and are turned on/off through their Shining property. The maximum number of light source in a scene is limited by the OpenGL implementation (8 lights are supported under most ICDs), though the more light you use, the slower rendering may get. If you want to render many more light/lightsource, you may have to resort to other techniques like lightmapping.
LightStyle:
lsSpot : a spot light, oriented and with a cutoff zone (note that if cutoff is 180, the spot is rendered as an omni source)
lsOmni : an omnidirectionnal source, punctual and sending light in all directions uniformously
lsParallel : a parallel light, oriented as the light source is.

LightSetAmbientColor(light, color)

LightSetDiffuseColor(light, color)

LightSetSpecularColor(light, color)

LightSetAttenuation(light, const, linear, quadratic)

LightSetShining(light, mode)

Switch light on or off (true/false)

LightSetSpotCutoff(light, spotcuttoff)

LightSetSpotExponent(light, exponent)

LightSetSpotDirection(light, x, y, z)

LightSetStyle(light, LightStyle)

BmpfontCreate(CharWidth, CharHeight, HSpace, VSpace, IntervalX, IntervalY, RangeStart, RangeEnd)

Returns a BitmapFont object. The bitmap has to be loaded with BmpfontLoad

BmpfontLoad(font, filename)

Loads a bitmap into the Bmpfont object.

WindowsBitmapfontCreate(fontname, fontsize, RangeStart, RangeEnd)

Returns a WindowsBitmapfont object.
CHANGES FROM 2.0.X: added RangeStart and RangeEnd

HUDTextCreate(font, text, parent)

Returns a HUDText object - a 2D text displayed and positionned in 2D coordinates.
The HUDText uses a character font defined and stored by a BitmapFont object. The text can be scaled and rotated.

HUDTextSetRotation(text, rotation)
Sets rotation.

HUDTextSetFont(htext,font)
Refers the BitmapFont to use.
The referred bitmap font component stores and allows access to individual character bitmaps.

HUDTextSetColor(htext,color,alpha)
Text color and alpha.

HUDTextSetText(htext,string)
Text to render.
Be aware that only the characters available in the bitmap font will be rendered.

FlatTextCreate(font, text, parent)
Returns a FlatText object - a 2D text displayed and positionned in 3D coordinates.
The FlatText uses a character font defined and stored by a BitmapFont object. Default character scale is 1 font pixel = 1 space unit.

FlatTextSetFont(ftext,font)
Refers the BitmapFont to use.
The referred bitmap font component stores and allows access to individual character bitmaps.

FlatTextSetColor(ftext,color,alpha)
Text color and alpha.

FlatTextSetText(ftext,string)
Text to render.
Be aware that only the characters available in the bitmap font will be rendered.

SpaceTextCreate(font, text, extrusion, parent)
Returns a SpaceText object that renders a text in 3D.
Arguments are the bitmap font, the text it should show, extrusion and parent.

SpaceTextSetExtrusion(text, extrusion)
Adjusts the SpaceText extrusion.
If Extrusion=0, the characters will be flat, values >0 will give them a third dimension.

SpaceTextSetFont(stext,winbmpfont)
Refers the BitmapFont to use.
The referred bitmap font component stores and allows access to individual character bitmaps.

SpaceTextSetText(stext,string)
Text to render.
Be aware that only the characters available in the bitmap font will be rendered.

HUDSpriteCreate(material, width, height, parent)
Returns a HUDSprite, a 2 dimensional sprite on top of everything.
Sprite functions below work for HUD sprites, too.

SpriteCreate(material, width, height, parent)
Returns a 3D sprite always facing the camera

SpriteSetSize(sprite, width, height)
Sets new size of sprite.

SpriteScale(sprite, x, y)
Sets new size relative.

SpriteSetRotation(sprite, angle)
Sets rotation.

SpriteRotate(sprite, angle)
Rotates relative.

SpriteMirror(sprite, horizontal, vertical)
horizontal: true for mirrored horizontally
vertical: true for mirrored vertically

SpriteNoZWrite(sprite,mode)
If True, sprite will not write to Z-Buffer.
Sprite will STILL be maskable by ZBuffer test.

PlaneCreate(style, width, height, xTiles, yTiles, parent)
Creates a plane object.
style: 0=multi quad, 1=single quad

AnnulusCreate(innerRadius, outerRadius, height, slices, stacks, loops, parent)
Creates an annulus object. An annulus is a cylinder that can be made hollow (pipe-like).

ConeCreate(bottomRadius, height, slices, stacks, loops, parent)
Creates a cone object.

CubeCreate(width, height, depth, parent)
Creates a cube object.

CubeSetNormalsDirection(cube,nd)

CylinderCreate(topRadius, bottomRadius, height, slices, stacks, loops, parent)
Creates a cylinder object. It can also be used to make truncated cones

DiskCreate(innerRadius, outerRadius, startAngle, sweepAngle, loops, slices,parent)
Creates a disk object. The disk may not be complete, it can have a hole (controled by the innerRadius argument) and can only be a slice (controled by the startAngle and sweepAngle properties).

SphereCreate(radius, slices, stacks, parent)

Creates a sphere object.

SphereSetAngleLimits(sphere,startangle,stopangle,topangle,bottomangle)

TorusCreate(majorRadius, minorRadius, rings, sides, parent)
Creates a torus object.

FrustrumCreate(basewidth,basedepth,apexheight,cutheight,parent)
Creates a frustrum (truncated pyramid) object.

DodecahedronCreate(parent)
Creates a dodecahedron object. The dodecahedron has no texture coordinates defined, so no texture will be mapped.

IcosahedronCreate(parent)
Creates an icosahedron object.

TeapotCreate(parent)
Creates a teapot object.

ActorCreate(filename,matlib,parent)
Actor is an object class specialized in animated meshes.
Actor provides an interface to animated meshes based on morph or skeleton frames, it is capable of performing frame interpolation and animation blending (via AnimationBlender functions).
CHANGES FROM 2.0.X: added material library parameter

ActorCopy(actor,parent)

ActorSetAnimationRange(actor,start,end)
Sets initial and final frames of the animation.

ActorGetCurrentFrame(actor)
Current animation frame.

ActorSwitchToAnimation(actor,animation,smooth)
CHANGES FROM 2.0.X: added smooth parameter

ActorSwitchToAnimationName(actor,animname,smooth)

ActorSynchronize(actor1,actor2)
Synchronize actor animation with an other actor.
Copies Start/Current/End frames, frame delta, AnimationMode and FrameInterpolation.

ActorSetInterval(actor,interval)
Interval between frames, in milliseconds.

ActorSetAnimationMode(actor,aam)

ActorSetFrameInterpolation(actor,afp)

ActorAddObject(actor,filename)

ActorGetCurrentAnimation(actor)

ActorGetFrameCount(actor)

ActorGetBoneCount(actor)

ActorGetBoneByName(actor,bonename)

ActorGetBoneRotation(actor,boneindex,index)

ActorGetBonePosition(actor,boneindex,index)

ActorShowSkeleton(actor,mode)

ActorBoneExportMatrix(actor,boneindex,object)
Copies an actor bone's local matrix into another object's local matrix. If you want, for example, put a weapon in character's hand, you should make it a child of an actor, adjust transformations if needed, and use this function every step.

ActorMakeSkeletalTranslationStatic(actor,animation)
Linearly removes the translation component between skeletal frames.
This function will compute the translation of the first bone (index 0) and linearly subtract this translation in all frames between start frame and end frame. Its purpose is essentially to remove the 'slide' that exists in some animation formats such as SMD.

ActorMakeSkeletalRotationDelta(actor,animation)
Removes the absolute rotation component of the skeletal frames.
Some formats will store frames with absolute rotation information, if this correct if the animation is the "main" animation. This function removes that absolute information, making the animation frames suitable for blending purposes.

AnimationBlenderCreate()
Creates and returns an actor animation blender. You can blend two or more SMD animations with this object.

AnimationBlenderSetActor(animblender,actor)
Binds the blender to the actor.

AnimationBlenderSetAnimation(animblender,name)
Assigns an animation to the blender.
name: *.SMD animation file name without extension

AnimationBlenderSetRatio(animblender,ratio)
Blending along the animation is just a matter of adjusting the ratio, with 0 = first frame and 1 = last frame.

TagListCreate(filename)
Loads and returns an animation tag list from *.MD3 file. These are used to locally transform the separate parts of the MD3 model into the correct places.

TagExportMatrix(actor,taglist,tagname,object)
Copies a tag's local matrix to an object's local matrix. This is commonly used to reposition the MD3 model parts each frame. You can use it with any other object as well.

ActorLoadQ3Animations(actor,filename,classname)
This function populates an animation data from the *.CFG file. The last parameter tells it which class of animation is to be loaded.

ActorMeshObjectsCount(actor)

ActorFaceGroupsCount(actor,meshobject)

ActorFaceGroupGetMaterialName(actor,meshobject,facegroup)

ActorFaceGroupSetMaterial(actor,meshobject,facegroup,material)

FreeformCreate(filename, matlib, lmmatlib, parent)
Creates a Freeform - container objects for an external mesh model.
Freeforms allows loading and rendering model files (like 3DStudio ".3DS" file) in Xtreme3D.
A Freeform may contain more than one mesh, but they will all be handled as a single object in a scene.
matlib - MaterialLibrary where primary materials will be loaded
lmmatlib - MaterialLibrary where secondary (lightmap) materials will be loaded, if there are any
CHANGES FROM 2.0.X: added material library parameters

FreeformMeshObjectsCount(freeform)
Returns number of meshes in the freeform.

FreeformMeshSetVisible(freeform,mesh,mode)

FreeformMeshSetSecondCoords(freeform1,mesh1,freeform2,mesh2)

FreeformMeshTriangleCount(freeform,mesh)
Returns number of triangles in the mesh.

FreeformFaceGroupsCount(freeform,mesh)
Returns number of face groups in the mesh.

FreeformFaceGroupTriangleCount(freeform,mesh,facegroup)
Returns number of triangles in the face group.

BmpHDSCreate(filename)
BmpHDSSetInfiniteWarp(hds, mode)
BmpHDSSetInverted(hds, inverted)
TerrainCreate(parent)
TerrainSetHeightData(terrain,hds)
TerrainSetTileSize(terrain,size);
TerrainSetTilesPerTexture(terrain,tiles);
TerrainSetQualityDistance(terrain,distance);

TerrainSetQualityStyle(terrain,hrs);
hrs: hrsFullGeometry = 0, hrsTesselated = 1.
TerrainSetMaxCLodTriangles(terrain,triangles);
TerrainSetCLodPrecision(terrain,precision);
TerrainSetOcclusionFrameSkip(terrain,frames);
TerrainSetOcclusionTesselate(terrain,tot);
tot: totTesselateAlways = 0, totTesselateIfVisible = 1.
TerrainGetHeightAtObjectPosition(terrain,object);
TerrainGetLastTriCount(terrain);

ObjectSetMaterial(object, material)
Assigns a material to the object.

ObjectSetPosition(object, x, y, z)
Sets the position in local coordinates.

ObjectMove(object, speed)

ObjectStrafe(object, speed)

ObjectRotate(object, pitch, turn, roll)

ObjectGetPitch(object)

ObjectSetScale(object, x, y, z)

ObjectSetPositionY(object, x, y, z)
Sets the position by Y-axis in local coordinates.

ObjectExportMatrix(object1,object2)
Copies the local matrix of object1 to the local matrix of object2.

ObjectExportAbsoluteMatrix(object1,object2)
Copies the absolute matrix of object1 to the local matrix of object2.

ObjectSetEffect(object,fx)

MaterialLibraryCreate()

MaterialLibraryActivate(matlib)

MaterialLibrarySetTexturePaths(matlib,path)

MaterialCreate(matname, filename)

MaterialSetBlendingMode(material, BlendingMode)

MaterialSetOptions(material, IgnoreFog, NoLighting)

MaterialSetShininess(material,shininess)

MaterialSetSpecularColor(material,color)

MaterialSetSecondTexture (material1, material2)
Sets material1 second texture from material2 primary texture.

MaterialSetShader(material, shader)
Assigns a shader to the material.

MaterialTextureSaveToFile(material,filename)
Saves the material's texture to an external *.bmp file

MaterialNoiseCreate(material)
Creates a so-called procedural Perlin noise material. It is useful to render such textures like clouds, marbles etc on the fly. Noise material can even be animated!

MaterialNoiseAnimate(material,speed)
Updates an animation increasing step size by speed value

MaterialNoiseSetDimensions(material,width,height)
Sets noise material resolution

MaterialNoiseSetMinCut(material,mincut)
Sets the minimal possible size of the noise.

MaterialNoiseSetSharpness(material,sharpness)
Sets sharpness parameter

MaterialNoiseSetSeamless(material,mode)
True for seamless texture, false for not

MaterialNoiseRandomSeed(material,seed)
Sets the seed number to generate a noise texture

ShaderEnable(shader,mode)

PhongShaderCreate()
The PhongShader implements simple phong shading through the use of an ARB vertex and fragment program. So far only the material and light properties are supported, some form of texture support will be added in future updates.

BumpShaderCreate()
The BumpShader utilizes OpenGL extensions to perform object space bump mapping. The bump shader runs an ambient light pass and a pass for each light shining in the scene.
The normal map is expected as the primary texture. Diffuse textures are supported through the secondary texture and can be enabled using the diffusetex2 bump option.

BumpShaderSetMethod(shader,bm)
There are currently 2 bump methods: a dot3 texture combiner and a basic ARB fragment program.

The dot3 texture combiner only supports diffuse lighting but is fast and works on lower end graphics adapters. The basic ARBFP method supports diffuse and specular lighting. Both methods pick up the light and material options through the OpenGL state.
bm: bmDot3TexCombiner = 0, bmBasicARBFP = 1

BumpShaderSetSpecularMode(shader,sm)

Sets the specular component.
sm: smOff = 0, smBlinn = 1, smPhong = 2

BumpShaderSetSpace(shader,bs)

bs: bsObject = 0, bsTangentExternal = 1, bsTangentQuaternion = 2

BumpShaderSetParallaxOffset(shader,offset)

BumpShaderSetOptions(shader,diffusetex2,usesectexcoords,lightattenuation,parallaxmapping)

diffusetex2 - use the second texture as a diffuse texture
usesectexcoords - use second texture coordinates
lightattenuation - the shader will use the light attenuation coefficients when calculating light intensity.
parallaxmapping - use the parallax mapping

CelShaderCreate( )
CelShaderSetLineColor(shader,color)
CelShaderSetLineWidth(shader,width)
CelShaderSetOptions(shader,outlines,textured)

TexCombineShaderCreate(matlib)

The TexCombineShader allows to declare how each texture unit should be used, and how each should be combined with the others. You can compose your own basic "shaders" with it.
TexCombineShader supports up to 4 texture units. The first two are loaded automatically: first unit is the first material in a library, second unit should be it's second texture (you assign it yourself). Third and fourth units are specified with TexCombineShaderMaterial3 and TexCombineShaderMaterial4.
matlib : Material Library to work with.

TexCombineShaderAddCombiner(tcs, procedure_string)

Appends a new texture combiner procedure. A texture combiner "code" defines how each texture should be combined, knowing that the result of the last texture unit (the one with the higher index) defines the final output.
Syntax:
Pascal-like, one instruction per line, use '//' for comment.
Accepted tokens:
"Tex0", "Tex1", "Tex2", "Tex3": texture unit
"PrimaryColor" (or "Col") : the primary color
"ConstantColor' (or "EnvCol") : texture environment constant color
Tokens can be qualified with ".a" or ".alpha" to specify the alpha channel explicitly, and ".rgb" to specify color channels (default). You cannot mix alpha and rgb tokens in the same line.
Examples:
1) replace texture 1 with texture 0:
"Tex1:=Tex0;"
2) additive blending between textures 0 and 1:
"Tex1:=Tex0+Tex1;"

3) subtractive blending between textures 0 and 1:
"Tex1:=Tex0-Tex1;"
4) modulation betwwen textures 0 and 1:
"Tex1:=Tex0*Tex1;"
5) signed additive blending between textures 0 and 1:
"Tex1:=Tex0+Tex1-0.5;"
6) interpolation between textures 0 and 1 using primary color as factor:
"Tex1:=Interpolate(Tex0,Tex1,PrimaryColor);"
7) dot3 product between textures 0 and 1:
"Tex1:=Dot3(Tex0,Tex1);"

TexCombineShaderMaterial3(tcs, material)
Sets the third texture unit.

TexCombineShaderMaterial4(tcs, material)
Sets the fourth texture unit.

ThorFXManagerCreate()

ThorFXCreate(manager)
CHANGES FROM 2.0.X: you should set an effect to object with ObjectSetEffect

ThorFXSetColor(manager, inner_c, inner_a, outer_c, outer_a, core_c, core_a)

ThorFXEnableCore(manager,mode)

ThorFXEnableGlow(manager,mode)

ThorFXSetMaxParticles(manager,maxparticles)

ThorFXSetGlowSize(manager,size)

ThorFXSetVibrate(manager,vibrate)

ThorFXSetWildness(manager,wildness)

ThorFXSetTarget(manager,x,y,z)

MemoryViewerCreate(width,height)
Creates a memory viewer.
Memory viewer is similar to an ordinary viewer, but it renders into memory buffer instead of screen. It can be used to perform specific rendering operations, such as Z-buffer shadowing.

MemoryViewerSetCamera(memviewer,camera)

MemoryViewerRender(memviewer)

ZShadowsCreate(viewer,memviewer,width,height)
Creates a Z-buffer shadow object to render a high-quality soft real-time shadows. Any visible object casts shadows onto any other objects, and even onto itself. The main idea behind this tehnique is rendering a

scene into memory viewer and then generating a HUD image from its Z-buffer. At this stage various texture operations, such as pixel sampling, can be performed.
viewer: specifies main Viewer object
memviewer: specifies Memory Viewer (viewer's camera is taken as a spot lightsource).

ZShadowsSetColor(zshadows,color,alpha)

Sets color and alpha for the shadows.

ZShadowsSetSoft(zshadows,mode)

Gives shadows soft edges, by sampling 4 pixels on the lightsource z-buffer.

ZShadowsSetTolerance(zshadows,tolerance)

Specify the tolerance between the z-value and lightsource distance.
Too small values cause ugly dot patterns, caused when surfaces
casts shadows onto itself, due to small inaduracies in calculations.
Too large values cause light to shine too far through a surface.

ZShadowsSetDepthFade(zshadows,mode)

Causes the brightness of the lightsource to fade with distance.
For now, this uses linear fading, just like the fog-algorythm.

ZShadowsSetFrustShadow(zshadows,mode)

Specify if the area outside the lightsource view frustrum is in shadow, or light.

ZShadowsSetSkyShadow(zshadows,mode)