WorldSim3D 0.8

Table of contents

Description

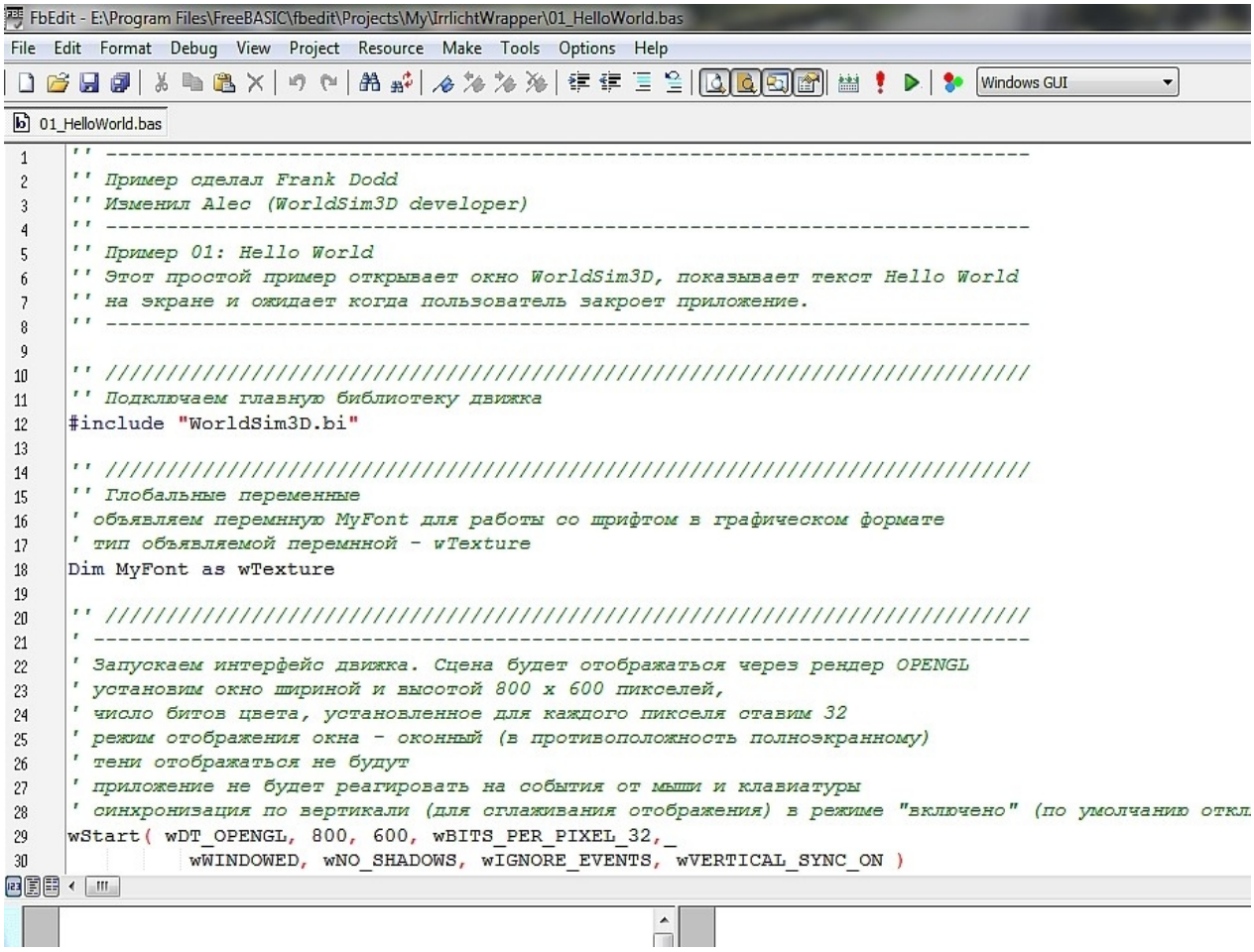
Getting started

Getting Started

The article is written by Frank Dodd
Adapted by Alec

Overview

The first thing I must do is acknowledge the fantastic work that has been carried out by the many different authors whos amazing work makes it possible for this package to enable BASIC programmers to build rich and immersive 3D applications and games. You can find a list of acknowledgements at the end of this document.



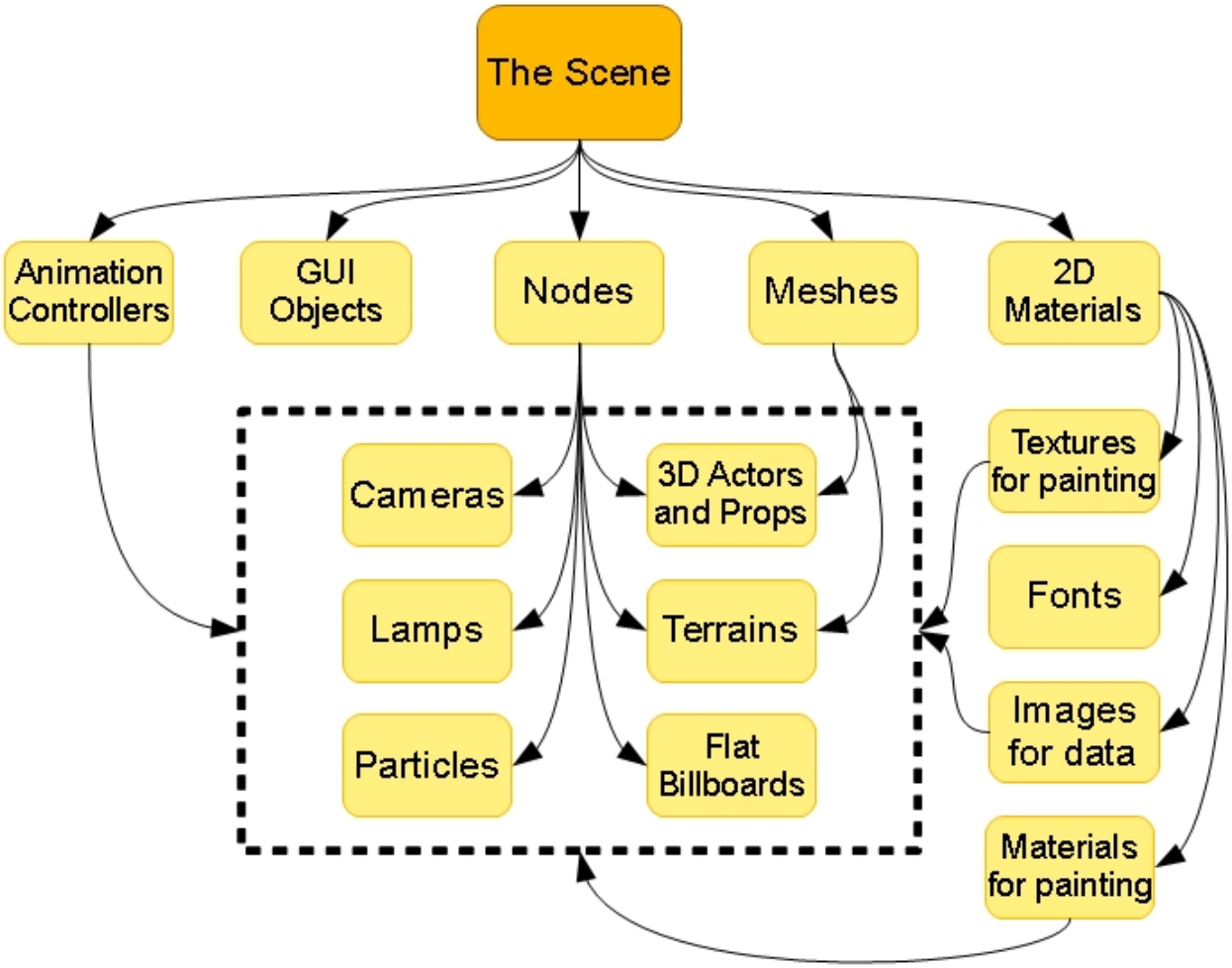
The WorldSim3D provides a simple and partial gateway from non object-oriented languages like FreeBasic into the object oriented 3D graphics Engine Irrlicht. Although the WorldSim3D only exposes part of the functionality of Irrlicht it endevours to expose the most important parts to give you access to Irrlichts powerful features. It does not directly expose the Irrlicht commands but provides a seperate syntax to encapsulate the engine.

In total it provides 361 commands that cover Bitmaps, 3D models, Maps, Terrains, Cameras, Lights and more.

This document serves as an introduction into the WorldSim3D 0.8.

The Scene

When you start WorldSim3D with the wStart command you create a scene that represents your 3D world, this scene is hidden behind the WorldSim3D and is used to load and store all of your resources: Bitmap Textures, 3D Meshes, Fonts and the Nodes in your world.



The Texture and Image Objects

The texture and the image objects are both 2D bitmaps that are loaded from a bitmap file. Textures are used to paint the surface of nodes (3D objects) in the scene or can be drawn directly to the screen as an image for counters or logos. Textures are stored in the computers main memory but, where possible, they are also stored in the Graphics Cards video memory to speed up 3D drawing. When you get a new texture WorldSim3D will return the object to you as an **wTexture** type

Images are usualy used as a source of information and are not drawn to the screen and are instead used to set the height of terrains or to describe how grass is distributed. Images are stored only in the computers main memory. When you get a new image file WorldSim3D will return the object to you as an **wImage**.

You need to save and manage both of these objects so you can use them later. Once you are finished with them it is important that you remove them from memory as images can occupy a lot of memory and you can easily run out of video memory in particular.

The Font Object

An WorldSim3D font is a monochromatic bitmap font that can be used to draw simple text onto

the display. When you get a new font WorldSim3D will return the object to you as an **wFont** type.

Hello, World!

The Mesh Object

The Mesh can be thought of like a blue print and is not actually displayed in your scene. It is a list of triangular geometry that fits together to describe a 3D model, this can be a static prop, a complex map or a character. The mesh can also store animation information and a description of which parts of a texture are to be painted onto its surface.

Where a mesh describes a map it will also have references to dozens of bitmaps that it uses to paint its surfaces.. When you get a new mesh object WorldSim3D will return it to you as an **wMesh** type.

The Node Object

A node is a physical object in your world a 3D Model, Camera, Light, Terrain, Billboard or Particle System. It is an element that has a position, rotation and scale, it will usually be rendered as a visible element if it is in front of the active camera when the canvas is drawn.

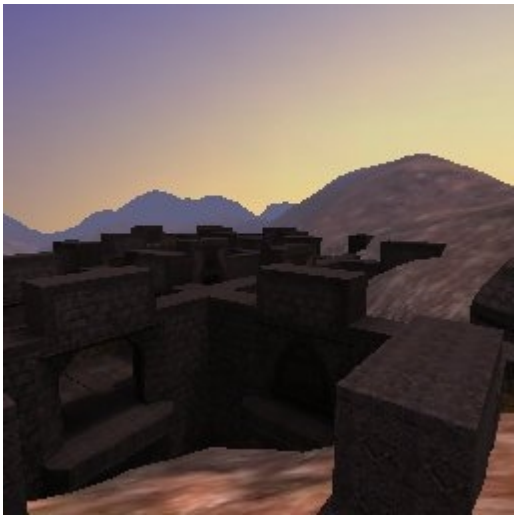
Nodes can usually be painted with a texture a process that is referred to as applying a material, they can be moved, rotated and scaled, hidden from view and deleted from the scene.

A 3D model is usually created by adding a mesh object to the scene. When you create a new node it will be returned to you as an **wNode** type.

The Camera Object

A camera object is a special node type and represent your vantage point into the 3D world, the camera has a viewpoint and a target at which it looks. Cameras have a series of special commands that can alter the appearance of the display however they can also be copied directly into variables of type wNode and be manipulated with all of the node commands. When a camera is created it is returned to you an **wCamera** type.

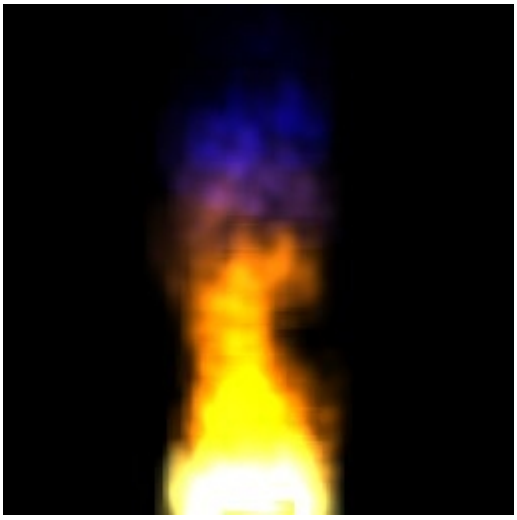
The Terrain Object



A terrain object is a special node type and consists of a large square mesh that is pulled and lowered to represent hills, valleys and mountains.

The terrain can be textured to give the appearance of a realistic landmass and with careful design it can even have map objects buried into it. Terrains have some special commands however they can also be copied directly into variables of type wNode and be manipulated with all of the node commands. When a terrain is created it is returned to you an **wTerrain** type.

The Particle System Object



A particle system is a special node type that can be set up to spit out hundreds of tiny little Billboard like objects and simulate effects like, fire, explosions, fountains, waterfalls, rain etc ...

The particle system has an item called an emitters attached to spit out particles and items called affectors attached to fade then out and change their direction.

Particle systems have some special commands however they can also be copied directly into variables of type wNode and be manipulated with all of the node commands. When a particle system is created it is returned to you an **wParticleSystem** type.

The Animator Object

Animator objects are attached to nodes and are used to delete, rotate or move the object in some way without you having to control the animation yourself step by step. When you create an animator it will be returned to you as an **wAnimator** type.

The Selector Object

Selector objects are used to gather together groups of triangular geometry in the scene that can be used for special functions, at the moment WorldSim3D only supports using them for collision detection. When you create a selector object it will be returned to you as a **wSelector** type.

Collision

Collision

The article is written by Frank Dodd
Adapted by Alec

Introduction
A very important feature of almost every 3D application is knowing when objects collide with other objects in the virtual world, these collisions are used for many things like:

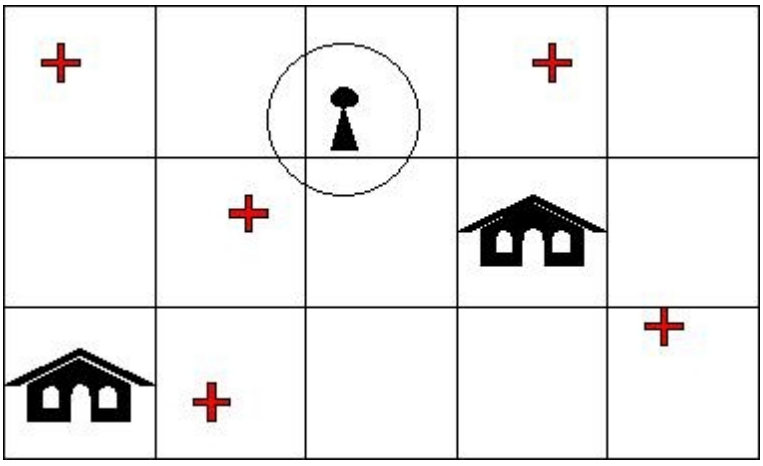
Preventing a camera moving through walls and objects.
Entering special areas such as an elevator.
Knowing when you can pick up an object.
Activating a virtual control.
Detecting the point of impact of a projectile.
Accurately simulating falling objects.

WorldSim3D has a wide range of ways that you can test for colliding objects without using extra physics libraries. And if that isn't enough for your project you can use ODE, which linked in WorldSim3D.

WorldSim3D performs its collision through an object known as a triangle selector, this is represented with the wSelector type. A triangle selector is essentially a simplified version of your model, it contains only the geometry either constructed from the surface of your model or from the bounding box of your model (a simple cube that defines the volume your model occupies). These triangle selectors can be joined together into groups to construct a simple representation of the surfaces in your scene that you wish to collide against.

In this paper we are going to examine methods for solving each of the problems we have listed above.

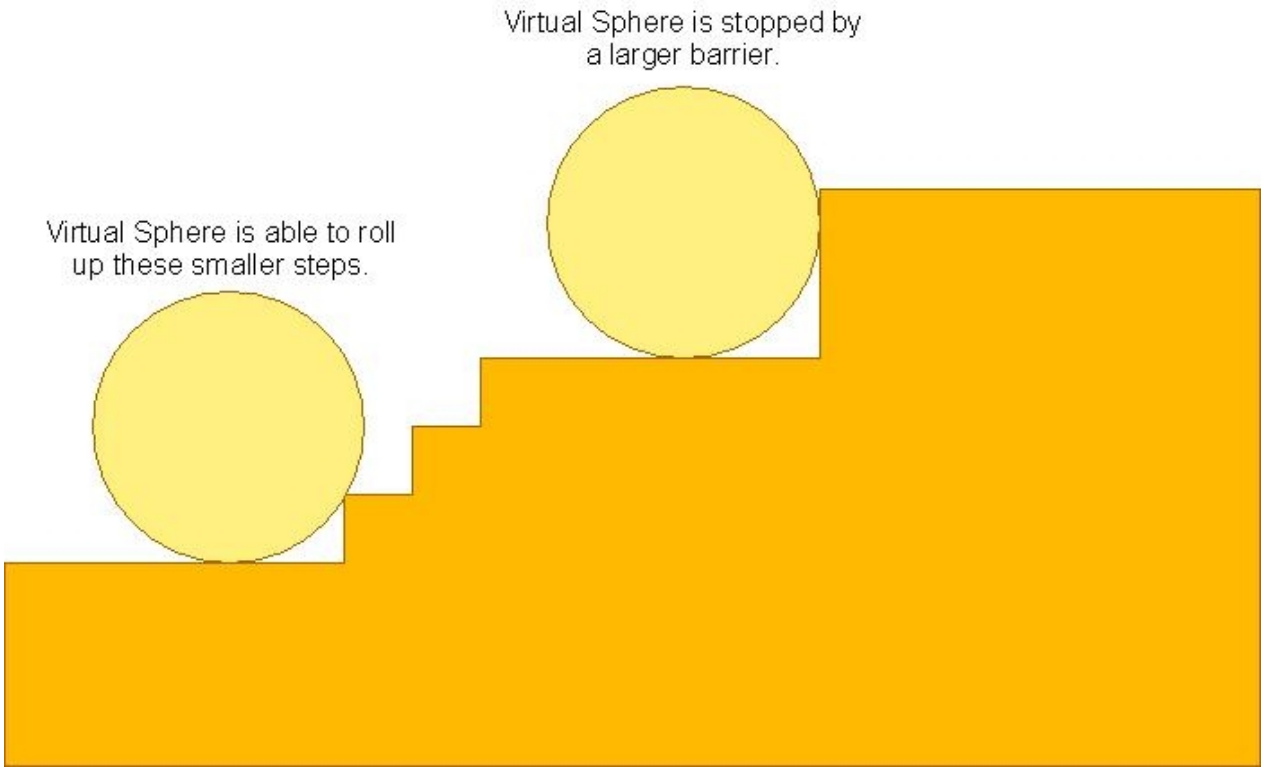
World Segmentation
Before collision is examined, it is essential to address segmentation of the world. In a large and complex environment there may be many hundreds even thousands of objects that you can collide with however only a few of these may actually be in your locallity, testing against every object in your scene will be wasteful and can reduce the performance of your game. This is where segmentation of your environment becomes vital.



In this very simple diagram the avatar (encircled) has a number of 'medical' objects that they can collect and two structures that they can collide with. This world has been divided into 15 areas and clearly the avatars 'area of influence' only overlaps four of those areas. When testing for the collection of medical objects there is only one object that needs to be checked. When preventing the camera moving through walls in the building neither of the potentially large structures needs to be checked potentially saving thousands of

calculations.

Preventing a camera moving through walls and objects
The easiest way to stop your camera moving though objects in the world is with WorldSim3D built-in collision system. This tests for collision between a virtual sphere around your node and all of the polygons that you place into the collision system. It enables you to move the sphere around, even up and down steps and ladders.



The surfaces in a collision are stored in a special object called an **wSelector**. This contains a simplified version of a model in your scene that is used purely for testing collisions.

You can create them simply by supplying the mesh and your node to a single call:

```
DIM as  wSelector mycollision = wGetCollisionGroupFromMesh( roomMesh, roomNode )
```

Or when you are dealing with a complex mesh like a BSP game level you should use the call:

```
DIM as wSelector mycollision = wGetCollisionGroupFromComplexMesh( levelMesh, levelNode )
```

Once you have your collision object you need something that will test for the collisions, for this you can use the special collision animator. This tests for collisions between the sphere and your collision objects and is applied with a single call:

```
wAddCollisionAnimator( mycollision, myCamera, _
                       sizeX, sizeY, sizeZ, _
                       gravityX, gravityY, gravityZ, _
                       offsetX, offsetY, offsetZ )
```

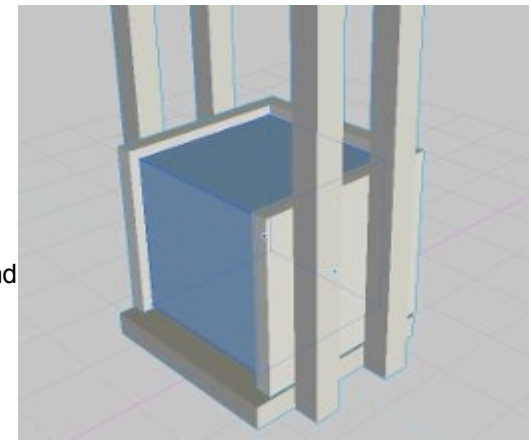
If you make the size of the sphere too small you will find it difficult to walk up steps, if you make the size of the sphere too large you might get stuck in doorways. Experiment to find a good size for your scene.

Gravity can pull in any direction but it will probably be a negative value applied to the Y component.

The offset can be used to offset the position of the object from the center of the virtual sphere, so you can raise a camera to appear to be at eye level for example.

The animator can be applied to any object to provide collision animation with other objects in the scene, controlled vehicles for example. However you should remember that this is purely for simple collisions it is not a physics simulator.

Entering special areas
Often in many games and applications you will want to enter special areas: elevators, spring boards or simply areas that trigger other actions.

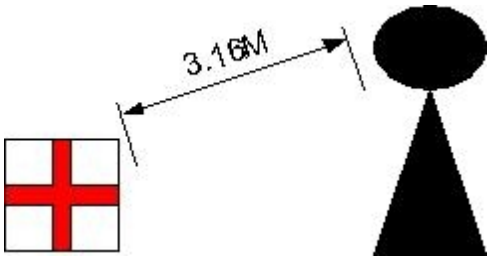


You can test for these conditions with a simple call to wIsPointInsideNode.
As long as your nodes model encompasses the whole of the area you want to test
against.

If you wish to test against a virtual or invisible area you can simply create a node perhaps even a box shaped test node and make it invisible by making a call to wSetNodeVisibility.

While your character is inside the elevator you might alter the position of your avatar with the same forces that you apply to the elevator.

Picking up objects



Picking up objects within a world is a simple task, you can simply judge if the distance between your character and the object, if the distance is short enough the object can be automatically collected or collected with a key press.

The function wGetDistanceBetweenNodes can be used to determine the distances between your camera and a collectable item for example.

Activating a virtual control: Triggers

Often within a game you will want to press a button, open a door or activate an object.. Usually this will be achieved by clicking on it on the screen or adjusting the camera so a crosshair is on the object before a button is pressed.

Detecting the impact of a projectile

Many games need to detect the impact of a projectile against other objects or the surface of a map, whether that is the strike of gunshot, a paint ball or a thrown object. Thanks to 'The Car' we now have a method that we can use to extract all of this important information in a single call.

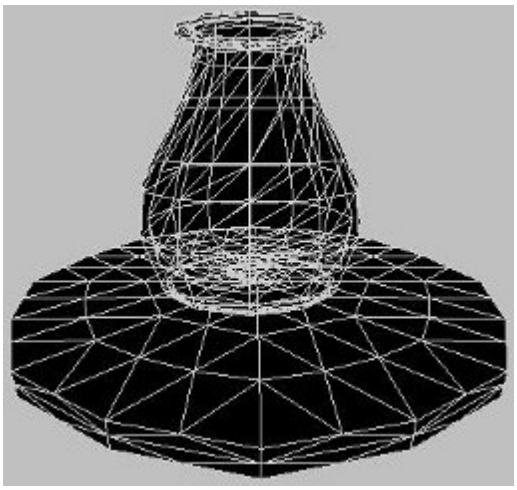
Accurately simulating objects

Acurately simulating collisions and the motions of objects in those collisions is actually beyond the what the WorldSim3D is intended to do, however thanks to 'Siskinedge' we now have example integration and a useable interface for working with physics libraries like Newton and the Open Dynamics Engine.

Lighting

Lighting

The article is written by Frank Dodd
Adapted by Alec

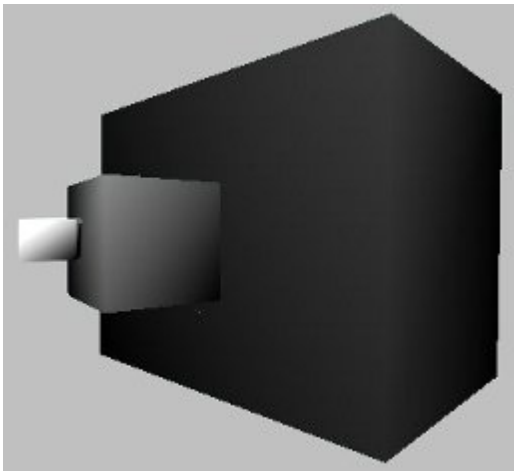


An extremely important aspect of creating a pleasing and realistic scene is getting the lighting right, while this can become a very complex subject there are many simple things you can do to make you scene look better.

Face Sizes

Lighting is applied to a model on a per vertex basis, when you are not using shaders, so having a good even spread of vertices should help you produce a good even lighting spread around the model. Avoiding long polygons to help stop long streaks of light on flat diffuse surfaces.

Scale and Normals



The normals in your model determine the direction that your vertices are facing, and are a very important as a part of lighting because the angle between the light, the surface and the camera determines the brightness of the vertex. One point of great concern when working with Normals is the fact that the normals have to be 'Unit Length' that is when the X, Y and Z of the normals are added up they should result in the number 1 (or very near to it).

In the example to the right the center node has the correct illumination, but the other nodes have been scaled with a call to wSetNodeScale and their normals have also been scaled the smaller nodes normals have become smaller and the node has therefore become brighter, the larger nodes normals have become larger and it has therefore become darker.
There are two ways to counter this effect when using dynamic lighting. After scaling your node with wSetNodeScale you can then make the following call for the node.

```
wSetNodeMaterialFlag ( SceneNode, wMF_normalize_normals, wON)
```

This call will make sure your normals are the correct length during rendering, however it may have a performance impact on your system. Another way is to use the new wScaleMesh call to scale the mesh instead without effecting the nodes. Scaling the mesh however will effect the scaling on all of your nodes.

Vertex Lighting

Vertex lighting lights your model by calculating how brightly illuminated each vertex should be. This depends on the angle between the light source and the camera, it also depends on the

direction the vertex is pointing and the colour and material properties of that vertex. There are four main types of light sources that you can introduce to your scene.

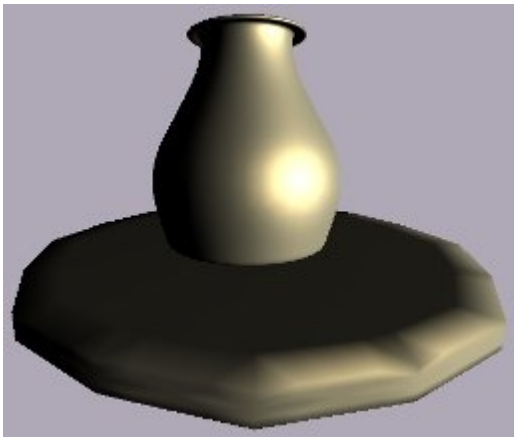
Point Lighting - Radiates lighting out from a point in all directions. This is the default light time in Irrlicht.

Directional Lighting - Casts light in the direction that the light source is facing as if the lightsource was an infinate distance away. This light illuminates your objects from the same direction no matter where they are in the scene and is a good choice for sunlight.

Spot Lighting - Casts light in the direction that the light source is facing in a cone that gets larger the further away you are from the light. This is a good choice for stage lights, flashlights etc ...

Ambient Lighting - This is a special light setting that effects all points in the model no matter which direction they are facing, it simulates the lighting that bounces around the environment from object to object making sure that some light is cast on surfaces that are facing away from the lightsource and looking unnaturally dark.

Equally as important are the materials that are applied to the object, these control how the lighting is reflected off your object as can be seen in the high detail object in the image below.



Diffuse Color - Is the color of light reflected normally from the surface.

Specular Color - Is the color of the highlight where the lightsource is most brightly reflected. Specular color is added to diffuse color so to remove highlights set your specular color to black.

Ambient Color - Is the color of the light that is reflected ambiently from the ambient light setting in the scene. You could for example set a dim grey ambient light and then make the object reflect only blue ambient light, giving the object a blue tint.

Emissive Color - Is the color of the light that the object emits. Although the object itself does not become a lightsource you can make it appear as though it is glowing. This is useful for neon signs and very hot objects.

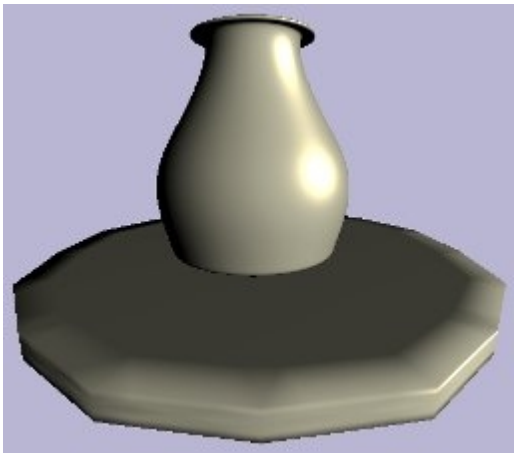
Shininess - This defines how small the highlights on your object are, the larger the number the smaller the highlights.

Each vertex in your model can also have a color and by default this vertex color overrides the diffuse color you set with an wMaterialSetDiffuseColor call. You can change which property is effected byt he vertex color or ensure that it effects no properties with a call to

```
wMaterialVertexColorAffects( Material, wCM_NONE )
```

Diffuse lighting works well even with low resoloution objects but specular lighting is most effect with objects that contain lots of small polygons.

Shader Lighting



Another way to light your model is through the use of advanced shaders, GLSL and HLSL are two examples that can perform advanced lighting effects that programmatically control how light is reflected from the surface of your object.

In the example shown to the right a basic GLSL lighting shader has been added that is creating high quality highlights on the object. In the vertex lighting example above the highlights were improved by using a 9,000 vertex object. The pot to the left achieves the same level of detail on an object with just 1,000 verticies.

While many quality effects are now available you should consider that fact that not all PC graphics support shaders (this is especially true for integrated graphics).

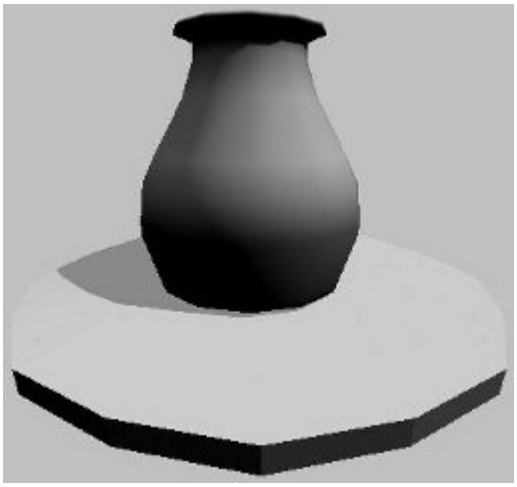
It is always good practice to check the capabilities of your users system and to provide the best experience that you can for their hardware.

Shadows

When drawing an object WorldSim3D determines the amount of light that hits the surface as if the whole scene was transparent, it does not perform the expensive task of detecting if that light is blocked by another object in the scene. This means shadows are an effect that need to be calculated seperately. There are currently four ways to create shadows within Irrlicht and each has its own benefits and its own drawbacks.

Shadow Baking
Shadow Decals
Stencil Shadows
Shader Shadows

Shadow Baking



Shadow baking works by applying a texture to the object, a texture that has the shadows already rendered into the image. This technique can only be applied to static objects and static lights but is perfect for things like buildings. As the shadows are pre-computed they can contain very expensive and realistic shadow calculations.

Shadows can either be baked into a texture along with the color information or it can be stored in its own image and applied with a seperate set of texture co-ordinates allowing color textures to be tiled for more detail, this is the technique used in many loaded BSP maps.

Here you can see that the object is casting a shadow onto the pedestal, there is a more defined shadow lighting down the side of the pot and the underneath of the rim of the pot is darkened into shadow.

The shadow map can also be mixed with dynamic lighting however to add dynamic tones to the object., this allows you to still use colored lights and to change the intensity of the light but you still cannot move the object or the light.

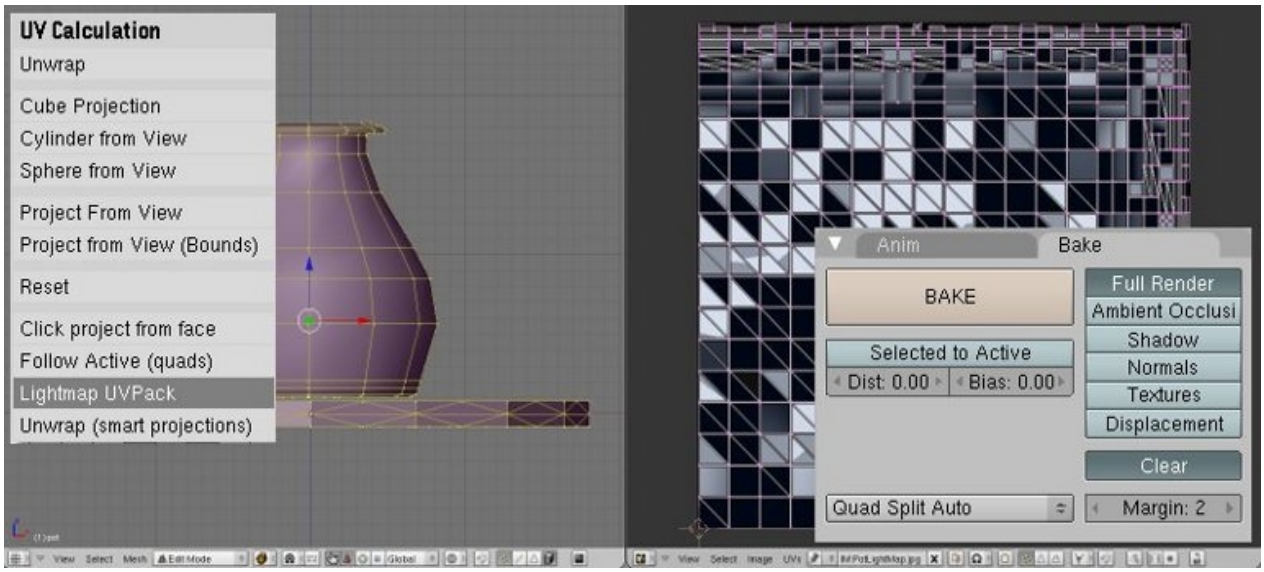
Many 3D packages support texture baking and your Graphics Artist should be familiar with the feature, we will examine implementation of the technique in Blender v2.49.

1. Cleanly UV map your object. You can UV map it anyway you like automatically or by hand but you cannot have overlapping faces. Each face must occupy a seperate location on the texture. The quickest way to do this is to use the Lightmap UVPack feature.
2. With your object selected and in edit mode, create a new image from a UV/Image Editor window. The higher the resoloution the better the shadow quality but the more memory that will be used.
3. From the scene tab select the Bake pane and click on Bake. You will now see your shadow map rendered to the UV/Image editor window.

Before you load your texture it is advisible to switch off mipmaps for the shadow maps. Shadowmaps and Mipmapping do not work well together.

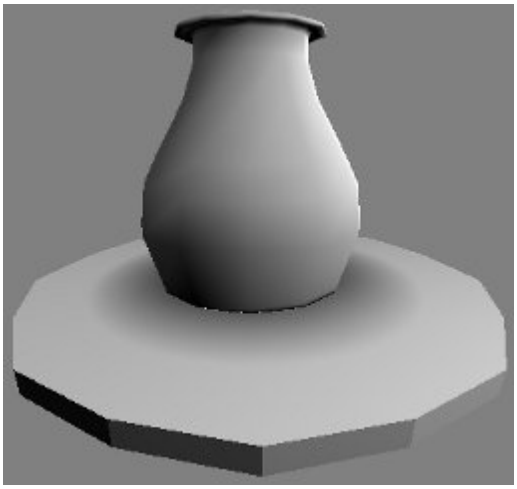
```
wSetTextureCreationFlag( wTCF_CREATE_MIP_MAPS, wOFF )
```

That is all there is to it. You can now save your image from UV/Image Editor 'image' menu and use this to texture your model. Someone that is experienced with the 3D package can bake in textures, radiosity and other features.



Shadow Decals

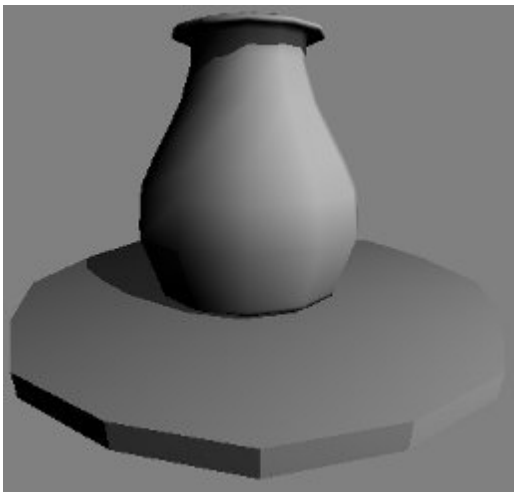
Shadow decals are small rectangles with a transparent texture that are usually attached just beneath the node and above the ground. They are best created as a darkened spot that simulates the loss of ambient lighting on the ground.



Although they are not a realistic shadow effect they are simple quick to render, they do have the appearence of a shadow and can dynamically move to track the object and the light. If your object is flying above the ground they can give the observer a perception of how high the object is from the ground too.

When associated with static objects these masks can even be batched into a single mesh making them a fast and memory efficient way to apply simple static shadows.

Stencil Shadows



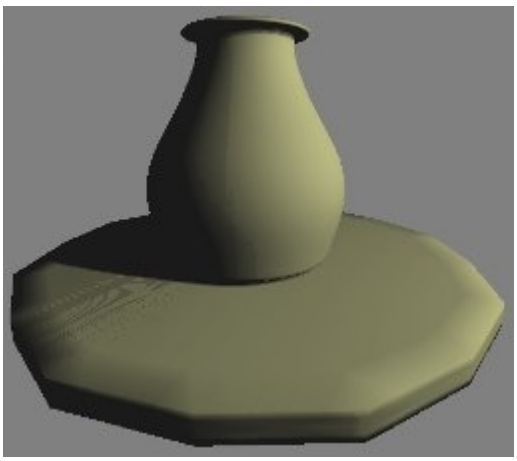
Stencil Shadows are a very commonly used technique and have the advantage that most hardware supports them including older graphics cards and integrated graphics chips. They appear to be realistic and accuratly render the shadows of moving and animated objects.

They do have disadvantages though, they are expensive to render as the effect has to be calculated by the CPU and not the graphics card. The more polygons you have enabled for shadows the worse it gets. The shadows will always have a hard edge also.

However if you use them sensibly and sparingly the effect works extreamly well and you can be confident that the vast majority of your audience will be able to see the effect.

To enable the effect you must enable shadows in your wStart call, call wAddNodeShadow for each node you want to cast a shadow and then finally set the color of your shadows with a call to wSetShadowColor.

Shader Shadows



Shader shadows are the ultimate in dynamic shadows and cast true rendered shadows across the objects they affect. The effect is achieved by using modern shader programs that run on the graphics card.

The effect is striking and very realistic it can show shadows from multiple objects, simulate light shone from flashlights and cast moving shadows even from animated objects.

The effect can be expensive however, each light is a separate render pass that requires a separate and large texture to save the shadow into. It also requires shader support so although it will run on many graphics cards it is unlikely to run on any PC using an older graphics card or an integrated graphics chip. So it should only be supplied as an option.

Shader shadows in WorldSim3D are achieved through Bitplanes XEffect Reloaded module and to enable the effect you must enable the XEffects system with an wXEffectsStart call, then for each node that you want to cast and receive a shadow you should call wXEffectsAddShadowToNode and then finally spot lights are added with a call to wXEffectsAddShadowLight. These set the color and quality of your shadows along with the position, target and angle of the spotlight. This can take some work to get right but the results

can be quite striking.

The Art of Lighting

While this discussion has examined the lighting tools that are available to does not examine how to use them effectively, however it would be remiss of me to not at least mention it. Lighting is an artform in itself that I find similar to sound, it adds emotional content to a scene, it can add drama, tension or any number of feelings to your actors, props and sets.

You will find many good discussions on lighting by simply search for terms like "3D lighting 101", "stage lighting 101", "Portrait Lighting 101", etc ...

Just dont underestimate it and if you have an artist creating your models discuss the lighting with him too and work out how enchancing your lighting can make a good model into a great piece of art.

Game Design

Functional Game Design

The article is written by Frank Dodd
Adapted by Alec

Here we explore a functional game design, a design that describes the features and functions of your game. This design helps you build up a plan of your game that can not only convay your ideas clearly to other people but will also explore things you may not have thought about and give you a better idea of your own project

Whether your project is large or small a good design is going to make it much easier to develop. There are many schools of thought on game design but there are some common rules that I believe are key to the design process:

Be unambiguous. Always be clear and positive don't use the words 'should', 'could' or may.

Be clear. Provide reasonable explainations and references, don't assume the reader already understands any concept in your game design. Where a diagram would be useful try to provide one.

Be professional. I avoid humor and jokes in the document except where it is essential to the design or concept, not everyone has the same sense of humor and they may not be in a humorous mood when reading your document.

Be a salesman. Pitch your project well. Your design may be seen by managers, marketeers and sponsors; these people are interested in a financial return from your project so address that. Your design may be seen by developers who want a an interesting and attractive project to work on, so address that too.

The Example Game Design

I am going to demonstrate my approach to the design process by providing an example design for a simple free game that will be pitched as a platform for delivering advertising.

Whirlybird
Frank Dodd 2010

1. Table of Contents (*This section will provide a table of contents for every major feature of the game design document*)

2. Product Perspective (This important section is the readers introduction to your design it will be short, clear and impressive)

2.1. Concept Artwork

2.2. Overview

Whirlybird is a free to market application for the PC that will be able to carry one or more marketing messages to its audience on in game objects.

The Player will control a simple helicopter and be challenged with taking off from a landing pad, moving to a number of waypoints within a time limit before returning to and landing safely at the landing pad.

At the end of each challenge a new challenge will be presented that increases the difficulty by placing the waypoints at more distant and dangerous locations.

2.3. Vision

- Whirlybird will provide a casual game experience where the player can join for a level and then pick up the same challenge later on.
- Whirlybird will start at a simple and accessible level of ability and gradually build up in difficulty as the player improves.
- Whirlybird will provide an attractive and entertaining game experience through an effective low budget media environment.

- Whirlybird will present advertising messages in a clear, dynamic and unobtrusive manner.

3. Core Gameplay (This section will provide a clear idea of the main concepts in the gameplay and will leave your reader with a detailed idea of how the game will play)

3.1. Overview
The game will be constructed from a series of challenges and each challenge from a series of collection tasks where the player collects waypoints distributed through the environment.

3.2. Challenges
The Challenge will begin with the helicopter at rest on the landing pad.

The player will start the helicopter and manouver it into the air.

The player will complete the challenges collection goals.

Once all collection goals are complete the Challenge is complete

The player will return to and bring their helicopter to rest on the landing pad.

3.3. Collection Goals
Once in the air the player will be able to rotate the helicopter and manouver it to the next waypoint, as the helicopter passes through the waypoint the waypoint will be collected and the collection goal will be complete.

3.4. End Game
The game will end when all challenges have been completed.

4. Helicopter (One key functional aspect that needs describing is your Avatar, here that avatar is the helicopter)

4.1. Overview
The helicopter is the players representation in the game world in this instance the vehicle that the player will be piloting.

4.2. Flight Dynamics
The helicopter will respond in a simple but realistic manner. Based upon the physical forces of:

Gravity - A constant downward force accellerating the vehicle to the ground
Drag - A force proportional to the velocity of the vehicle that works to bring the velocity to zero.
Lift - A controllable upward force lifting the vehicle into the air
Thrust - A controllable directional force that moves the vehicle in a particular direction

4.3. Controls
The vehicle will simulate the major flight controls of a helicopter:

Cyclic - A four way control that determines the tilt of the rotor blades and the direction of travel.
Collective - A two way control that determines the lift derived from the rotator and the thrust produced both vertically and in direction of the cyclic.
Anti-torque pedals - A two way control that changes tail rotor thrust and is able to spin the vehicle left or right (a force called Yaw).
Throttle - A two way control that increases or decreases engine power to maintain an acceptable rotor speed.

W - Cyclic forward
A - Cyclic left
S - Cyclic backward
D - Cyclic right

Q - Anti-torque Left

E - Anti-torque Right

- - Throttle Down
= - Throttle Up

[- Collective Down
] - Collective Up

The player will throttle up the helicopter to apply power, apply collective to lift it safely into the air. They can then apply cyclic to move the travel of the helicopter. To land they will release collective to reduce lift and decend to the surface.

4.4. Collision and Damage
The vehicle will be able to collide with both the terrain and trees placed upon the terrain.

Striking the terrain with a vertical force of over 1m/s will cause a failure of the vehicle.
Striking the terrain with a horizontal force of over 0.5m/s will cause a roll and a failure of the vehicle.
Striking a tree with any horizontal force will cause a failure of the vehicle.
Striking water at any velocity will cause a failure of the vehicle.

Striking a surface outside of the above criteria will bring the helicopter to rest.

4.5. Flight Time
The vehicle will be able to maintain throttle to its Rotors for exactly two minutes at the end of two minutes throttle will reduce to zero.

If the helicopter is at rest and there is no throttle the game will end.

5. Environment (Another functional aspect that needs describing is your environment, an overview of the general environment would be useful and then descriptions of each level or area)

5.1. Overview
A single environment represents the game world in all challenges, it provides a hazard which the helicopter can strike and end the challenge.

5.2. Terrain
The terrain is an island location with a flat spacious central area that provides a simple piloting challenge. Away from the center of the island hills begin to appear until out at the edge of the island hazardous mountain areas decend into the surrounding water

5.3. Water
The water surrounds the island and any locations on the island where the terrain decends below sea level.

The water provides another collision hazard.

5.4. Trees
Trees are scattered sparsely on the island surface and provide a collision hazard. A number of variety of trees are provided for and increase envrionmental detail.

6. Waypoints (Add in and describe in detail each of the main functional components of your game)

6.3.1. Overview
Waypoints are locations that the player must reach to collect and move on a step in the challenge. Once the player reaches a waypoint the next will become visible

6.3.2. Representation
The waypoint is represented by a red circle physically at the location of the waypoint and a yellow circle that overlays the red circle but is a constant size and is always visible on the screen.

If the waypoint is not physically visible on the screen the yellow circle will stop at the edge of the screen

closest to the waypoint.

6.3.3. Locations
The waypoints will be created at a series of predefined locations that are presented to the player as each waypoint is completed.

6.3.4. Collection
The player is able to collect a waypoint simply by flying to within 5 meters of the location. The waypoint will then be collected and the representation will move to the next waypoint.

If there are no more waypoints in this challenge. The representation will be removed and the player must locate the landing pad on their own.

7. Landing Pad

7.1. Overview
The Landing pad is a solid location at which the helcoper starts and at which the helicopter must land to complete the challenge

7.2. Advertising
The base of the landing pad will be decorated with advertising messages that will spin rotate around the circumference.

7.3. Location
The Landing pad will always be found at the center of the map on the surface of the terrain

7.4. Starting Point
The helicopter will always start a challenge at rest on the Landing Pad with the throttle low.

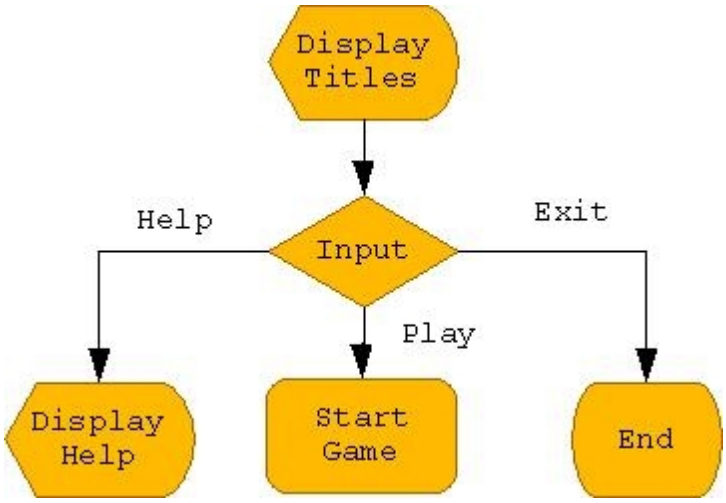
7.5. Landing Point
The helicopter will end a challenge when it is at rest on the Landing Pad with the throttle low and no waypoints to collect.

8. Structure (This section will provide an overview of the structure of the game and how the game will progress it will be complete but it will not go into too much technical detail, that will be in a higher level design document)

8.1. Overview
The game will be designed to progress in a linear manner through the stages described in this section.

8.2. Start Up
When the game is started it will display a title screen, three depressable buttons and play an introductionary musical track.

The game will wait for the player to click on one of three options and to either display a help page, start the game or exit the game.

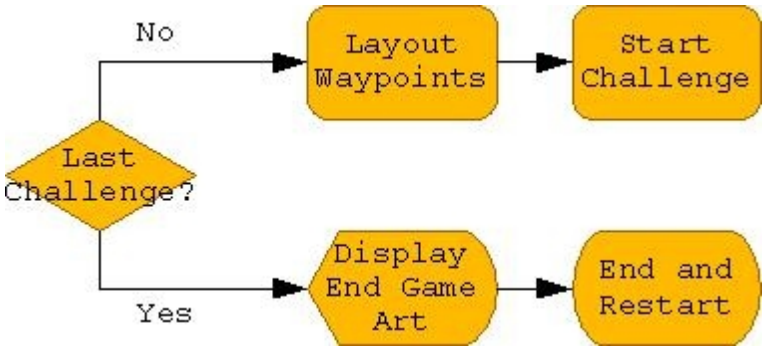


8.3. Game Cycle

The game cycle determines if this is the last challenge in the game.

While there are more challenges the game will layout waypoints and start the next challenge.

When ther are no more challenges the game will display End Game Art and restart the game at the titles.



8.4. Challenge Cycle

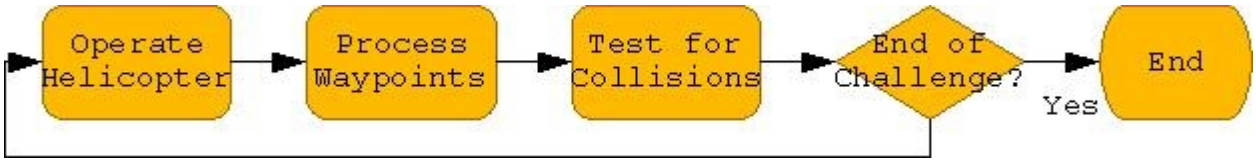
During the challenge cycle the player will operate the helicopter.

The game will check for arrival at waypoints and display the next waypoint when they arrive.

The game will check for collisions between the helicopter and environmental objects.

The game will check for conditions that indicate the challenge has ended and will at that point end the challenge.

If conditions to end the challenge have not been met the challenge cycle will continue.



9. Media Assets

9.1. 3D Models

The game has been designed to use a small number of three dimentional assets and associated texturing:

1 Helicopter

1 Landing Pad
1 Terrain
1 Water
4 Trees

9.2. 2D Graphics
The game has been designed to require a small number of two dimentional images:

1 Title Overlay
1 Help screen Overlay
1 Waypoint Collected Overlay
1 Challenge Complete Overlay
1 Crashed Overlay
3 Depressable image buttons (Play, Help and Exit)
1 Font
2 Waypoint Images

9.3. Sound Effects
The game has been designed to use a small number of sound effects:

1 Helicopter Engine Start
1 Helicopter Engine Running.
1 Helicopter Landing
1 Helicopter Crash
1 Helicopter Splash into water.

9.4. Music
The game has been designed to use a small number of musical tracks:

1 Introduction track
1 Ambient gameplay track

Licence

END-USER LICENSE AGREEMENT ("EULA")

This EULA is a legal agreement between you (either an individual or a single entity) and Alec ("THE AUTHOR"), for the software accompanying this EULA ("THIS SOFTWARE"). THIS SOFTWARE includes executables (both the editor and the compiler), associated media and electronic documentation. By installing and using copies of THIS SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, you may not install and use THIS SOFTWARE.

ALL entities that install and use THIS SOFTWARE agree to be bound by Law of the Russian Federationin respect to any legal actions that may arise.

THIS SOFTWARE is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties.

All copyrights to THIS SOFTWARE are exclusively owned by THE AUTHOR. All rights not expressly granted here are reserved by THE AUTHOR.

The license is personal, non-exclusive and cannot be rented, leased or transferred to anyone.

Any unauthorized use of THIS SOFTWARE shall result in immediate and automatic termination of this license and may result in criminal and/or civil prosecution.

This EULA grants you:

- the right to install and use THIS SOFTWARE for unlimited time, for commercial and non-commercial purposes, without providing either credits or additional fees to THE AUTHOR.
- the right to distribute, for commercial and non-commercial purposes, any number of copies of any work created by using THIS SOFTWARE, without providing either credits or additional fees to THE AUTHOR.
- the right to include to your works created with THIS SOFTWARE the 3d models, images, sound effects entirely created by THE AUTHOR and included to the installation file for THIS SOFTWARE, without providing either credits or additional fees to THE AUTHOR. You may or may not have the right to also include contents (also included to the installation file for THIS SOFTWARE) if it was created entirely or in part by other authors. Please see the 'artwork_readme.txt' file included to the installtion file for THIS SOFTWARE for details.
- the right to make one copy of THIS SOFTWARE for archive or backup purposes.

You may NOT:
- remove or alter any proprietary notices, labels or marks on or in THIS SOFTWARE.
- modify, translate, reverse engineer, decompile, disassemble THIS SOFTWARE.
- rent, transfer, or grant any rights (including possession) in the original or backup copy of THIS SOFTWARE, in full or in part, to any person.

THIS SOFTWARE is provided "as is" and with all faults. THE AUTHOR explicitly and expressly disclaims all warranties and guarantees and does not make any representations with respect to THIS SOFTWARE, whether express, implied, or statutory, including, but not limited to any (if any) warranties of or related to: fitness for a particular purpose, title, non-infringement, lack of viruses, accuracy or completeness of responses, results, lack of negligence or lack of workmanlike effort, and correspondence to description. The entire risk arising out of use or performance of THIS SOFTWARE remains with you.

In no event shall THE AUTHOR be liable for any direct, indirect, consequential, incidental, special, punitive, or other damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, claims of third parties, damages as a result of injury to any person, or any other loss) arising out of or in connection with the license granted under this license agreement or the use of or inability to use THIS SOFTWARE, even if THE AUTHOR has been advised of the possibility of such damages.

Acknowledgements

Many thanks to the following people whose work the WorldSim3D is built upon or who have helped with contributions to the engine or with technical assistance for various features and problems.

Irrlicht http://irrlicht.sourceforge.net/author.html
FreeBasic http://www.freebasic.net/index.php/about?section=credits
Frank Dodd created the Irrlicht wrapper (fantastic work), which is a base for WorldSim3D
Simple Terrain Spattering With invaluable help from the OpenGL Splat tutorial by Jarno van der Linden (http://www.cs.auckland.ac.nz/~jvan006/multitex/multitex.html)
Grass SceneNode Released under the Irrlicht License by G Davidson
Clouds SceneNode Released under the Irrlicht License by G Davidson
Lens Flare Scene Node Placed into the public domain by Paulo Oliveira
6DOF Camera With lots of help from Colin MacDonald and theoretical advice from Aleofjax
Color and Spline particle affectors Released under the Irrlicht License by Dark Kilauea
Open Dynamics Engine Physics support Bindings developed by D.J Peters and the library provided by the Open Dynamics Engine team.
Many Irrilicht commands and support Provided by Agamemnus and The Car
FreeType truetype font support Provided by the FreeType team (www.freetype.org)
Batching Meshes Provided by Gaz Davidson (Bitplane)
Beam Node Provided by Gaz Davidson (Bitplane)
Bolt Node Provided by Sudi and Trivtn under the Irrlicht License
XEffects - Reloaded Provided by Blindside
Many thanks for the development tools Code::Blocks (www.codeblocks.org); GCC Compiler (gcc.gnu.org);

FBIde (www.freebasic.net); FBEdit (fbedit.freebasic.net); NVu
Lots of thanks for help in the Forums Eponasoft; Daiwa; Crocodudule, John K, AlecZ, Alvaro Victor; thebignic and everyone that has offered their support. Thanks.
And many thanks to the thousands of mails posts, examples, replies and comments scattered across the Internet that provided insights into many of the questions that needed to be solved in creating this library.

Functions reference

WorldSim3D Functions Reference

System

These calls deal with starting, running and stopping the WorldSimulator3D engine, it also includes calls that get system metrics and some other miscellaneous tools.

30 functions

wStart

Syntax:

wStart (device type, screen width as integer, screen height as integer, bits per pixel, full screen, use shadows, capture mouse and keyboard, vertical syncronisation)

Description:

Starts the WorldSimulator3D interface and opens a window for rendering.

device type specifies the renderer to use when drawing to the display this may be one of the following types:

wDT_NULL
A NULL device with no display
wDT_SOFTWARE
Irrlichts default software renderer
IRR_EDT_SOFTWARE2
An improved quality software renderer
wDT_OPENGL
Hardware accelerated OpenGL renderer
wEDT_DIRECT3D8
Hardware accelerated DirectX 8 renderer (not included in the Wrappers 'Irlicht.dll' distribution)
wDT_DIRECT3D9
Hardware accelerated DirectX 9 renderer (not included in the Wrappers 'Irlicht.dll'

distribution)

screen width specifies the width of the viewport in pixels

screen height specifies the height of the viewport in pixels

The number of color bits that is used for each pixel 32 bit color gives 24 million different colors whereas 16 bit color gives only 32,000 colors. However the advantage of 16 bit color is that some operations use half the memory and can run at up to twice the speed. This setting can be either of: -
wBITS_PER_PIXEL_16
wBITS_PER_PIXEL_32

Full screen specifies whether the display is to opened in full screen mode or in a window
wWINDOWED
For window mode
wFULLSCREEN
For fullscreen mode. When using full screen mode you will need to adjust the window size to the same dimensions as a supported screen resolution on the target display 640x400 for example.

Use shadows starts the engine in a mode that supports the rendering of stencil shadows.
wNO_SHADOWS
For a display that does not support shadows.
wSHADOWS
For a display that supports shadows.

Capture mouse and keyboard specified whether you want to capture keyboard and mouse events, if you choose to ignore them they will be handled by WorldSim3D for FPS camera control. This parameter should be either of:
wIGNORE_EVENTS
wCAPTURE_EVENTS

vertical syncronisation specifies whether the display of each new frame is syncronised with vertical refresh of the graphics card. This produces a smoother display and avoids 'tearing' where the viewer can see parts of two different frames at the same time. The setting can be either of :-
wVERTICAL_SYNC_OFF
wVERTICAL_SYNC_ON

Example

wStart(wDT_OPENGL, screen_width, screen_height, wBITS_PER_PIXEL_32, wWINDOWED, wSHADOWS, wIGNORE_EVENTS, wVERTICAL_SYNC_ON)

wStartAdvanced

Syntax

```
integer = wStart ( _
   drivertype as wDEVICE_TYPES, _
   scrWidth as integer, _
   scrHeight as integer, _
   bits as uinteger, _
   fullscreen as uinteger, _
   shadows as uinteger, _
   dontignoreinput as uinteger, _
   vsyncenabled as uinteger = wOFF, _
   devicetype as uinteger = 0, _
   doublebufferenabled as uinteger = wON, _
   antialiasenabled as uinteger = 0, _
   highprecisionfpu as uinteger = wOFF )
```

Description

An advanced call for starting the WorldSim3D interface and opens a window for rendering.

device type specifies the renderer to use when drawing to the display this may be one of the following types: -

wDT_NULL
A NULL device with no display
wDT_SOFTWARE
Irrlichts default software renderer
wDT_SOFTWARE2
An improved quality software renderer
wDT_OPENGL
Hardware accelerated OpenGL renderer
wDT_DIRECT3D8
Hardware accelerated DirectX 8 renderer (not included in the Wrappers 'Irlicht.dll' distribution)
wDT_DIRECT3D9
Hardware accelerated DirectX 9 renderer (not included in the Wrappers 'Irlicht.dll' distribution)

screen width specifies the width of the viewport in pixels
screen height specifies the height of the viewport in pixels

The number of color bits that is used for each pixel 32 bit color gives 24 million different colors whereas 16 bit color gives only 32,000 colors. However the advantage of 16 bit color is that some operations use half the memory and can run at up to twice the speed. This setting can be either of: -
wBITS_PER_PIXEL_16
wBITS_PER_PIXEL_32

Full screen specifies whether the display is to opened in full screen mode or in a window
wWINDOWED
For window mode
wFULLSCREEN
For fullscreen mode. When using full screen mode you will need to adjust the window size to the same dimensions as a supported screen resolution on the target display 640x400 for example.

Use shadows starts the engine in a mode that supports the rendering of stencil shadows.
wNO_SHADOWS
For a display that does not support shadows.
wSHADOWS
For a display that supports shadows.

Capture mouse and keyboard specified whether you want to capture keyboard and mouse events, if you choose to ignore them they will be handled by Irrlicht for FPS camera control. This parameter should be either of:
wIGNORE_EVENTS
wCAPTURE_EVENTS

vertical syncronisation specifies whether the display of each new frame is syncronised with vertical refresh of the graphics card. This produces a smoother display and avoids 'tearing' where the viewer can see parts of two different frames at the same time. The setting can be either of :-
wVERTICAL_SYNC_OFF
wVERTICAL_SYNC_ON

devicetype allows a specific type of device for example a windows screen or a console to be selected. For the time being this should be set to 0 which automatically selects the best device

doublebufferenabled is used to control whether double buffering is used. When double buffering

is used two drawing surfaces are created one for display and the other that is used for drawing too. Double buffering is required for anit-aliasing the options are: wON or wOFF

antialiasenabled is used to enable the antialiasing effect, this effect produces a blurring at the edges of object giving their lines a smooth natural appearence. There is usually a big penalty for using this effect though sometimes as high as 30% of the frame rate or more. This is a value for the anti-aliasing and should be a power of 2. (e.g: 2, 4, 8, 16)

highprecisionfpu is used to enable high precision Floating point calculations, that produce more accurate result at the expense of a slower operating speed.

Example

```
wStartAdvanced ( _
   wDT_OPENGL, _      ' Use OpenGL
   800, 600, _          ' in a window 800x600
   wBITS_PER_PIXEL_32, _' using 32 bit true color
   wWINDOWED, _       ' in a window
   wNO_SHADOWS, _     ' without stencil shadows
   wIGNORE_EVENTS, _  ' dont capture keystrokes and mouse
   wON, _             ' sync to the monitor refresh rate
   0, _               ' 0 = use the most appropriate window device
   wON, _             ' Switch on double buffering of the display
   4, _               ' Anti-aliasing level 4
   wON )              ' use high precision floating point math
```


wTransparentZWrite

Syntax
wTransparentZWrite

Description
Allow transparency to write to the z buffer, this is nessecary sometimes to correct problems with the ordering of transparent objects in the scene, it may also have an effect of performance however.

Example
wTransparentZWrite

wRunning

Syntax
wRunning

Description
Used to determine if the WorldSim3D engine is still running.

Example
```
wStart( wDT_OPENGL, screen_width, screen_height, wWINDOWED, wSHADOWS, wIGNORE_EVENTS )
While wRunning
Wend
wStop
```


wSetViewPort

Syntax
```
wSetViewPort( topX as integer, topY as integer, bottomX as integer, bottomY as integer )
```

Description
Define the area of the screen into which elements are going to be drawn. This can be used to draw the scene multiple times for split screen effects.

Example
```
wSetActiveCamera( FirstCamera )
wSetViewPort( 0,0, 200,200 )
wDrawScene
```


wBeginScene

Syntax
```
wBeginScene( Red as integer, Green as integer, Blue as integer )
```

Description
Starts to draw a frame, erasing the canvas with the specified color. The colors are integer values in the range from 0 (black) to 255 (full intensity)

Example
```
wStart( wDT_OPENGL, screen_width, screen_height, wWINDOWED, wSHADOWS, wIGNORE_EVENTS )
While wRunning
    wBeginScene( 255, 255, 255 )
    wDrawScene
    wEndScene
Wend
wStop
```


wBeginSceneAdvanced

Syntax
```
wBeginSceneAdvanced( _
    sceneBackgroundColor As Uinteger, _
    clearBackBuffer As Ubyte = 1, _
    clearZBuffer As Ubyte = 1)
```

Description
Readies a scene for rendering, erasing the canvas and setting a background color.
Unlike the wBeginScene function, this function can clear the back- and the z-buffer.

Parameters:
    backBuffer   Specifies if the back buffer should be cleared, which means that the screen is filled with the color specified. If this parameter is false, the back buffer will not be cleared and the color parameter is

ignored.
zBuffer Specifies if the depth buffer (z buffer) should be cleared. It is not nesesarry to do so if only 2d drawing is used.
color The color used for back buffer clearing

Example
```
wStart( wDT_OPENGL, screen_width, screen_height, wWINDOWED, wSHADOWS, wIGNORE_EVENTS )
While wRunning
  wBeginSceneAdvanced ( RGBA( 128, 128, 128, 0 ), wON, wON )
   wDrawScene
   wEndScene
Wend
wStop
```


wDrawScene

Syntax
wDrawScene

Description
This renders the 3D scene to the canvas, drawing all 3D elements: nodes, particles, billboards, etc ....

Example
```
wStart( wDT_OPENGL, screen_width, screen_height, wWINDOWED, wSHADOWS, wIGNORE_EVENTS )
While wRunning
   wBeginScene( 255, 255, 255 )
   wDrawScene
   wEndScene
Wend
wStop
```


wDrawSceneToTexture

Syntax
wDrawSceneToTexture(render_texture as wTexture)

Description
Draw scene manager objects to a texture surface, the texture must have been created with a call to wCreateRenderTargetTexture. This is useful for creating textures from 3D objects in your scene perhaps nameplates in the interface for characters for example. Note: the target texture must be smaller than the view window as some resources are shared between the two.

Example
```
wSetActiveCamera ( StaticCamera )
wDrawSceneToTexture ( RenderTexture )

wBeginScene( 240, 255, 255 )

wSetActiveCamera ( FPSCamera )
wDrawScene
```


wSetRenderTarget

Syntax
wSetRenderTarget (texture As wTexture, sceneBackgroundColor As Uinteger = 0, clearBackBuffer As Ubyte

= 1, clearZBuffer As Ubyte = 1)

Description
Set the target surface for rendering, this allows objects to be rendered to a texture that can then be drawn to the screen or displayed on other objects. Calling this function with texture set to 0 sets the drawing target back to the screen,.

Texture is a texture created with the special wCreateRenderTargetTexture call.
scene background color is generated with the FreeBasic RGBA call and defines the colour used in any clear operation.
clean back buffer when set to wON erases the background of the texture
clear z buffer when set to wON erases the depth buffer (used by stencil shadows and some shaders)

Example
```
Texture = wCreateRenderTargetTexture( 512, 512 )
wSetRenderTarget( Texture, RGBA( 0,0,0,0), wON, wON )
wDrawScene
```


wDrawGUI

Syntax
wDrawGUI

Description
This renders the 2D graphical user interface that has been created to the scene. At the moment the engine supports these GUI objects: window, static text , button, scrollbar, listbox, editbox, image, checkbox and modal file open dialog.

Example
```
wStart( wDT_OPENGL, screen_width, screen_height, wWINDOWED, wSHADOWS, wIGNORE_EVENTS )
While wRunning
    wBeginScene( 255, 255, 255 )
    wDrawScene
    wDrawGUI
    wEndScene
Wend
wStop
```


wEndScene

Syntax
wEndScene

Description
This renders the 3D scene to the canvas, drawing all 3D elements: nodes, particles, billboards, etc ....

Example
```
wStart( wDT_OPENGL, screen_width, screen_height, wWINDOWED, wSHADOWS, wIGNORE_EVENTS )
While wRunning
    wBeginScene( 255, 255, 255 )
    wDrawScene
    wEndScene
Wend
wStop
```

wStop

Syntax
wStop

Description
Stop the WorldSim3D Engine freeing all of the resources and closing the display window.

Example
```
wStart( wDT_OPENGL, screen_width, screen_height, wINDOWED, wSHADOWS, wIGNORE_EVENTS )
While wRunning
    wBeginScene( 255, 255, 255 )
    wDrawScene
    wEndScene
Wend
wStop
```


wQueryFeature

Syntax
uinteger wQueryFeature(Feature as wVIDEO_FEATURE_QUERY)

Description
Used to determine if a particular video feature is supported by the graphics card. The function will return (1) if the feature is supported and (0) if it isn't. The feature parameter should be either of the following values:
wVDF_RENDER_TO_TARGET
Is driver able to render to a surface?
wVDF_HARDWARE_TL
Is hardeware transform and lighting supported?
wVDF_MULTITEXTURE
Are multiple textures per material possible?
wVDF_BILINEAR_FILTER
Is driver able to render with a bilinear filter applied?
wVDF_MIP_MAP
Can the driver handle mip maps?
wVDF_MIP_MAP_AUTO_UPDATE
Can the driver update mip maps automatically?
wVDF_STENCIL_BUFFER
Are stencil buffers switched on and does the device support stencil buffers?
wVDF_VERTEX_SHADER_1_1
Is Vertex Shader 1.1 supported?
wVDF_VERTEX_SHADER_2_0
Is Vertex Shader 2.0 supported?
wVDF_VERTEX_SHADER_3_0
Is Vertex Shader 3.0 supported?
wVDF_PIXEL_SHADER_1_1
Is Pixel Shader 1.1 supported?
wVDF_PIXEL_SHADER_1_2
Is Pixel Shader 1.2 supported?
wVDF_PIXEL_SHADER_1_3
Is Pixel Shader 1.3 supported?
wVDF_PIXEL_SHADER_1_4
Is Pixel Shader 1.4 supported?
wVDF_PIXEL_SHADER_2_0
Is Pixel Shader 2.0 supported?
wVDF_PIXEL_SHADER_3_0
Is Pixel Shader 3.0 supported?
wVDF_ARB_VERTEX_PROGRAM_1

Are ARB vertex programs v1.0 supported?
wVDF_ARB_FRAGMENT_PROGRAM_1
Are ARB fragment programs v1.0 supported?
wVDF_ARB_GLSL
Is GLSL supported?
wVDF_HLSL
Is HLSL supported?
wVDF_TEXTURE_NPOT
Are non-power-of-two textures supported?
wVDF_FRAMEBUFFER_OBJECT
Are framebuffer objects supported?
wVDF_VERTEX_BUFFER_OBJECT
Are vertex buffer objects supported?
wVDF_ALPHA_TO_COVERAGE
Is alpha to coverage supported?
wVDF_COLOR_MASK
Are color masks supported?
wVDF_MULTIPLE_RENDER_TARGETS
Are multiple render targets supported?
wVDF_MRT_BLEND
Are seperate blend settings for render targets supported?
wVDF_MRT_COLOR_MASK
Are seperate color masks for render targets supported?
wVDF_MRT_BLEND_FUNC
Are seperate blend functions for render targets supported?
wVDF_GEOMETRY_SHADER
Are geometry shaders supported?

Example
```
if wQueryFeature( wVDF_MULTITEXTURE ) = 0 then
  ? "MultiTexture is NOT supported"
End if
```


wDisableFeature

Syntax
uinteger wDisableFeature(Feature as wVIDEO_FEATURE_QUERY, state as uinteger)

Description
Used to disable a particular video feature on the graphics card. The feature parameter is identical to wQueryFeature.

State should be either wON or wOFF

Example
```
wDisableFeature( wVDF_MULTITEXTURE, wOFF )
```


wGetTime

Syntax
unsigned_integer = wGetTime

Description
Get the current time in milliseconds.

Example

time = wGetTime

wSetTime

Syntax
wGetTime(time as uinteger)

Description
Set the current animation time in milliseconds.

Example
wSetTime(2500)

wGetFPS

Syntax
Integer_variable = wGetFPS

Description
Get the current frame rate. This is determined by the number of times the wEndScene is called per second.

Example
frame_rate = wGetFPS
wStop()
Print “Frame Rate was “;frame_rate
Sleep

wGetPrimitivesDrawn

Syntax
unsigned_integer = wGetPrimitivesDrawn

Description
Get the number of primitives (mostly triangles) drawn in the last frame.

Example
polygons = wGetPrimitivesDrawn
wStop()
Print “The system drew about “;polygons;" triangles"
Sleep

wSetWindowCaption

Syntax
wSetWindowCaption(caption text as wide string)

Description
Set the caption in the WorldSim3D window title bar.

Example
wSetWindowCaption(“My WorldSim3D project”)

wMakeARGB

Syntax
unsigned_integer = wMakeARGB (Alpha, Red, Green, Blue)

Description
Takes four values representing a colors Alpha, Red, Green and Blue intensity and returns them as a 32bit unsigned integer. Typically used for working with colors in wVECT structures.

Example
vcolor = wMakeARGB(0, 255, 128, 128)

wGetScreenSize

Syntax
wGetScreenSize(width as integer, height as integer)

Description
Gets the screen side into the two supplied variables.

Example
wGetScreenSize(ScreenWidth, ScreenHeight)

wIsFullscreen

Syntax
wIsFullscreen() as integer

Description
Checks if the WorldSim3D window is running in fullscreen mode. Returns 0 if the application is windowed any other value indicates full screen mode

Example
if wIsFullscreen = wOFF Then Print "Windowed Mode"

wIsWindowActive

Syntax
wIsWindowActive() as integer

Description
Checks if the WorldSim3D window is active. Returns 0 if the application is windowed any other value indicates full screen mode

Example
if wIsWindowActive > 0 Then Print wDrawScene

wIsWindowFocused

Syntax
wIsWindowFocused() as integer

Description
Checks if the WorldSim3D window has focus. Returns 0 if the application is windowed any other value indicates full screen mode

Example
if wIsWindowFocused > 0 Then Print wDrawScene

wIsWindowMinimized

Syntax
wIsWindowMinimized() as integer

Description
Checks if the WorldSim3D window is minimized. Returns 0 if the application is windowed any other value indicates full screen mode

Example
if wIsWindowMinimized = 0 Then Print wDrawScene

wMaximizeWindow

Syntax
wMaximizeWindow()

Description
Maximizes the window if possible.

Example
wMaximizeWindow

wMinimizeWindow

Syntax
wMinimizeWindow()

Description
Minimizes the window if possible.

Example
wMinimizeWindow

wRestoreWindow

Syntax
wRestoreWindow()

Description
Restore the window to normal size if possible.

Example
wRestoreWindow

wSetResizableWindow

Syntax
wResizableWindow()

Description
Make the WorldSim3D window resizable by dragging on the corner of the window.

Example
wResizableWindow

Keyboard and Mouse

Input Event Functions

These calls allow you recover keyboard events and mouse actions that the user creates.

' find out if there is a key event ready to be read. returns 1 if there is an
' event available (the event receiver must have been started when the system
' was initialised)

10 functions

wKeyEventAvailable

Syntax
wKeyEventAvailable

Description
Determine if there are any keystrokes waiting to be read..

Example
```
while wKeyEventAvailable
    KeyEvent = wReadKeyEvent
Wend
```


wReadKeyEvent

Syntax
key_event_pointer = wReadKeyEvent

Description
Read a key event from the WorldSim3D window. The properties of the key event are stored in the returned type.

Example
```
While wKeyEventAvailable
    KeyEvent = wReadKeyEvent
    If KeyEvent->key = wKEY_DOWN then
        Movement = DOWN
```

```
    End If
Wend
```


wMouseEventAvailable

Syntax
wMouseEventAvailable

Description
Determine if there are any mouse actions waiting to be read.

Example

```
while wMouseEventAvailable
    MouseEvent = wReadMouseEvent
Wend
```


wReadMouseEvent

Syntax
mouse_event_pointer = wReadMouseEvent

Description
Read a mouse event from the WorldSim3D window. The properties of the mouse event are stored in the returned type.

Example

```
while wMouseEventAvailable
    ' read the mouse event out
    MouseEvent = wReadMouseEvent
    if MouseEvent->action = wME_MOUSE_MOVED then
        SPIN = MouseEvent->x
    endif
wend
```


wSetMousePosition

Syntax
wSetMousePosition(x as single, y as single)

Description
Set the position of the mouse pointer and return the relative change in position.

Example
wSetMousePosition(XPosition, YPosition)

wGetAbsoluteMousePosition

Syntax
wGetAbsoluteMousePosition(x as single, y as single)

Description
Gets the absoloute position of the mouse pointer.

Example
wGetAbsoluteMousePosition(XPosition, YPosition)

wHideMouse

Syntax
wHideMouse

Description
Hide the mouse pointer

Example
wHideMouse

wShowMouse

Syntax
wShowMouse

Description
Shows the mouse pointer

Example
wShowMouse

wGUIEvents

Syntax
wGUIEvents (eventsForGUI as integer)

Description
Lets the GUI system handle the events that is whether keyboard and mouse events should be used by the GUI

Example:
wGUIEvents(1)

wGUIEventAvailable

Syntax
wGUIEventAvailable ( )

Description
Find out if there is a GUI event ready to be read. Returns 1 if there is an event available (the event receiver must have been started when the system was initialised)

Example:
```
WHILE wRunning
    ' begin the scene, erasing the canvas to white before rendering
    wBeginScene( redBackground, greenBackground, 255 )

    ' draw the Graphical User Interface
```

```
    wDrawGUI

    ' if there are GUI events available
    If wGUIEventAvailable then

        ' read the GUI event out
        GUIEvent = wReadGUIEvent

        ' process the particular control
        select case GUIEvent->id

        case MY_GUI_BUTTON_FILE
            ' if the button has been pressed
            If GUIEvent->event = wGUI_ET_BUTTON_CLICKED then
                ' open a file open dialog
                wAddFileOpen( "Select an image file", MY_GUI_FILEOPEN, wGUI_MODAL  )
            End If

        case MY_GUI_BUTTON_WINDOW
            ' if the button has been pressed
            If GUIEvent->event = wGUI_ET_BUTTON_CLICKED then
                ' open a small modal window
                guiWindow = wAddWindow( "Window", 80, 80, 160, 144, wGUI_MODAL )
                ' with a button that can close the window
                wAddButton( 16, 32, 64, 48, 104, "Close me", "", guiWindow )
            End If

        case MY_GUI_BUTTON_CLOSE
            ' if the button has been pressed
            If GUIEvent->event = wGUI_ET_BUTTON_CLICKED then
                ' remove the window and its child button from the display
                wGUIRemove( guiWindow )
            End If
        End Select
    End if

    ' end drawing the scene and render it
    wEndScene
WEND
```


wReadGUIEvent

Syntax
wReadGUIEvent ( )

Description
Read a GUI event out

Example:

```
WHILE wRunning
    ' begin the scene, erasing the canvas to white before rendering
    wBeginScene( redBackground, greenBackground, 255 )

    ' draw the Graphical User Interface
    wDrawGUI

    ' if there are GUI events available
    If wGUIEventAvailable then
```

```
        ' read the GUI event out
        GUIEvent = wReadGUIEvent

        ' process the particular control
        select case GUIEvent->id

        case MY_GUI_BUTTON_FILE
            ' if the button has been pressed
            If GUIEvent->event = wGUI_ET_BUTTON_CLICKED then
                ' open a file open dialog
                wAddFileOpen( "Select an image file", MY_GUI_FILEOPEN, wGUI_MODAL  )
            End If
        End Select
    End if

    ' end drawing the scene and render it
    wEndScene
WEND
```


Filing System

wAddZipFile

Syntax
wAddZipFile(zip file as zstring, ignore case, ignore paths)

Description
Adds a zip archive to the filing system allowing you to load files straight out of the zip file. Common pk3 files are simply zip files

Ignore case should be one of the following values:
wUSE_CASE
wIGNORE_CASE

Ignore paths allows you to simply use the filename without the path, the filename should always be unique in the archive when using this option. The value should be one of the following:
wUSE_PATHS
wIGNORE_PATHS

Example
wAddZipFile("data.pk3", wIGNORE_CASE, wIGNORE_PATHS)

wChangeWorkingDirectory

Syntax
wChangeWorkingDirectory(New directory as zstring)

Description
Change the working directory of the WorldSim3D Environment.

Example
wChangeWorkingDirectory("c:\media")

wGetWorkingDirectory

Syntax
string = wGetWorkingDirectory

Description
Get the current working directory of the WorldSim3D Environment.

Example
CurrentDirectory = wGetWorkingDirectory

2D Functions

wSetTextureCreationFlag

Syntax
wSetTextureCreationFlag(flag_to_set as wTEXTURE_CREATION_FLAG, flag_value as uinteger)

Description
Sets texture creation flags controlling how textures are handled when they are created. The following flags can be set:

wTCF_ALWAYS_16_BIT
Forces the driver to always create 16 bit textures, independently of which format the file on disk has. When choosing this you may loose some color detail, but gain speed and save memory. 16 bit textures can be transferred twice as quickly as 32 bit textures and only use half of the memory space. When using this flag, it does not make sense to use the flags wTCF_ALWAYS_32_BIT, wTCF_OPTIMIZED_FOR_QUALITY, or wTCF_OPTIMIZED_FOR_SPEED at the same time.

wTCF_ALWAYS_32_BIT
Forces the driver to always create 32 bit textures, independently of which format the file on disk has. Please note that some drivers (like the software device) will ignore this, because they are only able to create and use 16 bit textures. When using this flag, it does not make sense to use the flags wTCF_ALWAYS_16_BIT, wTCF_OPTIMIZED_FOR_QUALITY, or wTCF_OPTIMIZED_FOR_SPEED at the same time.

wTCF_OPTIMIZED_FOR_QUALITY
Lets the driver decide in which format the textures are created and tries to make the textures look as good as possible. Usually it simply chooses the format in which the texture was stored on disk. When using this flag, it does not make sense to use the flags wTCF_ALWAYS_16_BIT, wTCF_ALWAYS_32_BIT, or wTCF_OPTIMIZED_FOR_SPEED at the same time.

wTCF_OPTIMIZED_FOR_SPEED
Lets the driver decide in which format the textures are created and tries to create them maximizing render speed. When using this flag, it does not make sense to use the flags wTCF_ALWAYS_16_BIT, wTCF_ALWAYS_32_BIT, or wTCF_OPTIMIZED_FOR_QUALITY, at the same time.

wTCF_CREATE_MIP_MAPS
Automatically creates mip map levels for the textures.

wTCF_NO_ALPHA_CHANNEL
Discard any alpha layer and use non-alpha color format.

Example

wSetTextureCreationFlag(wTCF_ALWAYS_32_BIT, wON)

wGetImage

Syntax
wTexture = wGetImage(Texture file name as zstring)

Description
Load a 2D texture from a bitmap file into main memory that can then be used to supply a heightmap to a terrain or other similar CPU based operations. The images can not be used to texture 3D objects.

Example
TerrainMap = wGetImage("heightmap.bmp")

wGetTexture

Syntax
wTexture = wGetTexture(Texture file name as zstring)

Description
Load a 2D texture from a bitmap file into video memory that can then be used to texture a model or to draw onto the screen.

Example
health_sprite = wGetTexture("Health.bmp")

wCreateTexture

Syntax
wTexture = wCreateTexture(texture_name as zstring, x_size as integer, y_size as integer, format as wCOLOR_FORMAT)

Description
Creates a blank texture. The format of the texture can be one of the following: -

wCF_A1R5G5B5
16 bit color format used by the software driver, and thus preferred by all other WorldSim3D engine video drivers. There are 5 bits for every color component, and a single bit is left for alpha information.

wCF_R5G6B5
Standard 16 bit color format.

wCF_R8G8B8
24 bit color, no alpha channel, but 8 bit for red, green and blue.

wCF_A8R8G8B8
Default 32 bit color format. 8 bits are used for every component: red, green, blue and alpha.

Example
radar_sprite = wCreateTexture("mytexture", 128, 128, wCF_A8R8G8B8)

wCreateImage

Syntax
wImage = wCreateImage(x_size as integer, y_size as integer, format as wCOLOR_FORMAT)

Description
Creates a blank image, does not use video memory. The format of the image can be one of the following:

wCF_A1R5G5B5
16 bit color format used by the software driver, and thus preferred by all other irrlicht engine video drivers. There are 5 bits for every color component, and a single bit is left for alpha information.

wCF_R5G6B5
Standard 16 bit color format.

wCF_R8G8B8
24 bit color, no alpha channel, but 8 bit for red, green and blue.

wCF_A8R8G8B8
Default 32 bit color format. 8 bits are used for every component: red, green, blue and alpha.

Example
BlankPicture = wCreateImage(128, 128, wCF_A8R8G8B8)

wCreateRenderTargetTexture

Syntax
wTexture = wCreateRenderTargetTexture(x_size as integer, y_size as integer)

Description
Create a texture that is suitable for the scene manager to use as a surface to which it can render its 3d object. Each of the dimensions must be of a power of two for example 128x128 or 256x256.

This function is very important when producing texture maps for special effects for example a rendering of a model for a 2D image displayed in the HUD, the rendering of a model for display on a 3D surface for example a video display of virtual camera, the rendering of the texture for the reflection of a mirror, the rendering of the environment for use in a water or chrome shader. Most cards, even old cards, will support this very important function.

Example
RenderTexture = wCreateRenderTargetTexture (256, 256)

wRemoveTexture

Syntax
wRemoveTexture(texture as wTexture)

Description
Removes the texture from memory freeing up the space it occupied. You should ensure that the texture is not in use by materials assigned to nodes.

Example
DIM MyTexture as wTexture
MyTexture = wGetTexture("Spaceship.bmp")
wRemoveTexture(MyTexture)

wRemoveImage

Syntax
wRemoveImage(image as wImage)

Description
Removes the image from memory freeing up the space it occupied. You should ensure that the image is not in use by other functions.

Example
```
DIM MyImage as wImage
MyImage = wGetImage( "Enemy.bmp" )
wRemoveImage( MyImage )
```


wMakeNormalMapTexture

Syntax
wMakeNormalMapTexture(Texture object as wTexture, Amplitude as single)

Description
Create a normal map from a gray-scale height map texture. Normal maps are used to add a high level of surface lighting detail to what are normally low resolution models. They can have a massive effect on the realism of an object, the model you create will have to be created in "tangent" space to support this.

Example
```
wMakeNormalMapTexture( WallBumps, 0.9 )
```


wLockTexture

Syntax
pixels_ptr = wLockTexture(texture as wTexture)

Description
Locks the texture and returns a pointer to the pixels.

Example
```
DIM texture_pixels as uinteger ptr
texture_pixels = wLockTexture( MyTexture )
```


wUnlockTexture

Syntax
wUnlockTexture(texture as wTexture)

Description
Unlock the texture, presumably after it has been modified and recreate the mipmap levels.

Example
```
wUnlockTexture( MyTexture )
```

wLockImage

Syntax
pixels_ptr = wLockImage(image as wImage)

Description
Locks the image and returns a pointer to the pixels.

Example
DIM image_pixels as uinteger ptr
image_pixels = wLockImage(MyImage)

wUnlockImage

Syntax
wUnlockImage(image as wImage)

Description
Unlock the image, presumably after it has been modified.

Example
wUnlockImage(MyImage)

wBlendTextures

Syntax
```
wBlendTextures  (destination_texture as wTexture,_
                 source_texture as wTexture,_
                 x_offset as integer,_
                 y_offset as integer,_
                 operation as integer ) as integer
```

Description
Blend the source texture into the destination texture to create a single texture

Example
Blend the two loaded textures onto the created surface
```
wBlendTextures( TextureD, TextureB, 0,0, wBLEND_ADD )
wBlendTextures( TextureD, TextureC, 0,0, wBLEND_SCREEN )
```


wColorKeyTexture

Syntax
wColorKeyTexture(Texture object as wTexture, Red as integer, Green as integer, Blue as integer)

Description
Copies any parts of the texture that are the same as the specified color into the textures alpha channel. This can then be used for special effects or to make these regions transparent.

Example
wColorKeyTexture(message, 255, 255, 255)

wDraw2DImage

Syntax
wDraw2DImage(Texture to draw as wTexture, X position as integer, Y position as integer)

Description
Draws the texture to the display at the supplied co-ordinates.

Example
wDraw2DImage(Station_title, 4, 4)

wDraw2DImageElement

Syntax
wDraw2DImageElement(Texture to draw as wTexture, X position as integer, Y position as integer, Source top X as integer, Source top Y as integer, Source bottom X as integer, Source bottom Y as integer, whether to use alpha as integer)

Description
Draws the texture to the display at the supplied co-ordinates, the image is copied from the specified rectangle in the source texture, this enables you to put many images onto a single texture. This function also supports the alpha channel when drawing the image to the display and can draw the image transparently.

The value for whether or not to use the alpha channel should be one of the following values: -
wIGNORE_ALPHA
wUSE_ALPHA

Example
wDraw2DImageElement(my_image, screen_width - 60 - 4, 4,0,0,60,31, wUSE_ALPHA)

wDraw2DImageElementStretch

Syntax
wDraw2DImageElementStretch (texture as wTexture, destination top X as integer, destination top Y as integer, destination bottom X as integer, destination bottom Y as integer, source top X as integer, source top Y as integer, source bottom X as integer, source bottom Y as integer, use Alpha as integer)

Description
Draws the texture to the display into the supplied rectangle, the image is copied from the specified rectangle in the source texture, this enables you to put many images onto a single texture. If the rectangles are different sizes this function will scale the images appropriately. This function also supports the alpha channel when drawing the image to the display and can draw the image transparently.

The value for whether or not to use the alpha channel should be one of the following values: -
wIGNORE_ALPHA
wUSE_ALPHA

Example
wDraw2DImageElementStretch(my_game_logo, 16, 16, 80, 80, 0, 0, 32, 32, wUSE_ALPHA)

wGetFont

Syntax

wFont = wGetFont(Filename of the bitmap font file as zstring)

Description
Loads a bitmap containing a bitmap font.

Example
BitmapFont = wGetFont ("bitmapfont.bmp")

w2DFontDraw

Syntax
w2DFontDraw (Font Object as wTexture, the text to display as wstring ptr, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer)

Description
Draws the text into the supplied rectangular area using the supplied font object.

Example
w2DFontDraw (BitmapFont, "SIMPLE MONOCHROME FONT", 120, 80, 250, 96)

wSaveScreenShot

Syntax
wSaveScreenShot(filename as zstring)

Description
Save a screenshot out to a file, the image format is defined by the extension applied to the filename. WorldSim3D currently supports: bmp, png, tga, ppm and jpg

Example
wSaveScreenShot("c:\myscreen.bmp")

wGetScreenShot

Syntax
texture = wGetScreenShot(x as uinteger, y as uinteger, width as uinteger, height as uinteger)

Description
Return a pointer to a texture containing a rectangular portion of a screenshot.

Example
DIM texture as wTexture = wGetScreenShot(0,0, 256,256)

wGetTextureInformation

Syntax
texture = wGetTextureInformation (texture as wTexture, textureWidth as unsigned integer, textureHeight as unsigned integer, texturePitch as unsigned integer,textureFormat as wCOLOR_FORMAT)

Description
Get information on a texture. The width, height, pitch and color format is returned in the supplied variables.

Example

wGetTextureInformation (selectedTexture, width, height, pitch, col_format)

wGetImageInformation

Syntax
texture = wGetImageInformation (image as wImage, textureWidth as unsigned integer, textureHeight as unsigned integer, texturePitch as unsigned integer,textureFormat as wCOLOR_FORMAT)

Description
Get information on an image. The width, height, pitch and color format is returned in the supplied variables.

Example
wGetImageInformation (selectedImage, width, height, pitch, col_format)

Shaders

Built-in functions

Advanced Materials use GPU programs to create sophisticated texturing effects that can greatly add to the realism of the scene but are only supported by modern graphics cards with Pixel and Vertex shader support. Currently Irrlicht which is a core engine for WorldSim3D supports Vertex Shaders, Pixel Shaders, ARB Vertex programs, ARB Fragment programs, HLSL (DirectX 9) and GLSL (OpenGL).

wCreateNamedVertexShaderConstant

wCreateNamedVertexShaderConstant

Syntax

result = wCreateNamedVertexShaderConstant (shader as wShader, const_name as zstring ptr, const_preset as integer, const_data as single, data_count as integer)

Description

Creates a Vertex shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array

Returns: 1 if the constant was sucessfully created

Example

wCreateNamedVertexShaderConstant (shader, "Time", byval wNO_PRESET, @time, 1)

wCreateNamedPixelShaderConstant

Syntax
result = IrrCreateNamedPixelShaderConstant (shader as wShader, const_name as zstring ptr, const_preset as integer, const_data as single, data_count as integer)

Description
Creates a Pixel shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array

Returns: 1 if the constant was sucessfully created

Example
```
dim color(4) as Single => { 1.0, 1.0, 1.0, 1.0 }
wCreateNamedPixelShaderConstant ( shader, "color", wNO_PRESET, @color, 4 )
```


wCreateAddressedVertexShaderConstant

Syntax
result = wCreateAddressedVertexShaderConstant (shader as wShader, const_address as integer, const_preset as integer, const_data as single, data_count as integer)

Description
Creates a Vertex shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array

Returns: 1 if the constant was sucessfully created

Example
```
wCreateAddressedVertexShaderConstant ( shader, 4, wNO_PRESET, @time, 1 )
```


wCreateAddressedPixelShaderConstant

Syntax
result = wCreateAddressedPixelShaderConstant (shader as wShader, const_address as integer, const_preset as integer, const_data as single, data_count as integer)

Description
Creates a Pixel shader constant that allows you to change the value of a constant inside a shader during the execution of the program, simply assign one of the preset constants to the constant name or attach the constant to an array of floats and change the constant simply by changing the values in your array

Returns: 1 if the constant was sucessfully created

Example
```
dim position(3) as Single => { 0.0, 0.0, 0.0 }
wCreateAddressedPixelShaderConstant ( shader, 2, wNO_PRESET, @position, 3 )
```


wAddHighLevelShaderMaterial

Syntax
wShader = wAddHighLevelShaderMaterial (vertex_program as zstring ptr, vertex_start_function as zstring

ptr, vertex_prog_type as uinteger, pixel_program as zstring ptr, pixel_start_function as zstring ptr, pixel_prog_type as uinteger, material_type as uinteger)

Description
Creates a new material using a high level shading language.

vertex program: String containing the source of the vertex shader program. This can be 0 if no vertex program shall be used.
vertex_start_function: Name of the entry function of the vertex shader program
vertex_program_type: Vertex shader version used to compile the GPU program
pixel_program: String containing the source of the pixel shader program. This can be 0 if no pixel shader shall be used.
pixel_start_function: Entry name of the function of the pixel shader program
pixel_program_type: Pixel shader version used to compile the GPU program
baseMaterial: Base material which renderstates will be used to shade the material.

Returns a type that contains a material_type number that can be used to shade nodes with this new material. If the shader could not be created it will return 0

Example
```
My_shader = wAddHighLevelShaderMaterial ( _
    vertex_program, "main", wVS_1_1, _
    pixel_program, "main", wPS_1_1, _
    wMT_SOLID )
```


wAddHighLevelShaderMaterialFromFiles

Syntax
wShader = wAddHighLevelShaderMaterialFromFiles (vertex_program_filename as zstring ptr, vertex_start_function as zstring ptr, vertex_prog_type as uinteger, pixel_program_filename as zstring ptr, pixel_start_function as zstring ptr, pixel_prog_type as uinteger, material_type as uinteger)

Description
Creates a new material using a high level shading language stored in files.

vertex program_filename: String containing the filename of the vertex shader program. This can be 0 if no vertex program shall be used.
vertex_start_function: Name of the entry function of the vertex shader program
vertex_program_type: Vertex shader version used to compile the GPU program
pixel_program_filename: String containing the filename of the pixel shader program. This can be 0 if no pixel shader shall be used.
pixel_start_function: Entry name of the function of the pixel shader program
pixel_program_type: Pixel shader version used to compile the GPU program
baseMaterial: Base material which renderstates will be used to shade the material.
Returns a type that contains a material_type number that can be used to shade nodes with this new material. If the shader could not be created it will return 0.

Example
```
My_shader = wAddHighLevelShaderMaterialFromFiles ( _
   ".\media\wood.vertex", "main", wVS_1_1, _
   ".\media\wood.pixel", "main", wPS_1_1, _
   wMT_SOLID )
```


wAddShaderMaterial

Syntax

wShader = wAddShaderMaterial (vertex_program as zstring ptr, pixel_program as zstring ptr, material_type as uinteger)

Description
Creates a new material using a shading language program stored in a string.

vertex program: String containing the source of the vertex shader program. This can be 0 if no vertex program shall be used. For DX8 programs, the will always input registers look like this: v0: position, v1: normal, v2: color, v3: texture cooridnates, v4: texture coordinates 2 if available. For DX9 programs, you can manually set the registers using the dcl_ statements.
pixel_program: String containing the source of the pixel shader program. This can be 0 if no pixel shader shall be used.
baseMaterial: Base material which renderstates will be used to shade the material.

Return: Returns a type that contains a material_type number that can be used to shade nodes with this new material. If the shader could not be created it will return 0

Example
My_shader = wAddShaderMaterial (vertex_program, pixel_program, wMT_SOLID)

wAddShaderMaterialFromFiles

Syntax
wShader = wAddShaderMaterialFromFiles (vertex_program_filename as zstring ptr, pixel_program_filename as zstring ptr, material_type as uinteger)

Description
Creates a new material using a shading language program stored in files.

vertex program: String containing the source of the vertex shader program. This can be 0 if no vertex program shall be used. For DX8 programs, they will always input registers look like this: v0: position, v1: normal, v2: color, v3: texture cooridnates, v4: texture coordinates 2 if available. For DX9 programs, you can manually set the registers using the dcl_ statements.

pixel_program: String containing the source of the pixel shader program. This can be 0 if no pixel shader shall be used.
baseMaterial: Base material which renderstates will be used to shade the material.
Return: Returns a type that contains a material_type number that can be used to shade nodes with this new material. If the shader could not be created it will return 0

Example
My_shader = wAddShaderMaterialFromFiles (".\media\wood_low.vtx", ".\media\wood_low.pxl" wMT_SOLID)

XEffects

17 functions

wXEffectsStart

Syntax
```
wXEffectsStart (
        vsm as integer = wOFF,
        softShadows as integer = wOFF,
        bitdepth32 as integer = wOFF )
```

Description
Starts the XEffects advanced shader extension provided by Bitplane from the Irrlicht Forums. This must be called before any other XEffects calls.

The first parameter 'vsm' is used to turn on the 'Variance Shadow Maps' feature. VSM is an advanced form of shading used to avoid aliasing problems that can be seen with the other shadowing function. It can create clear sharp shadowing. Use wON to enable this feature.

The second parameter 'soft shadows' provides blurred shadows, similar as those cast by a large source. Use wON to enable this feature.

The last parameter 'bit depth 32' enables 32 bit buffers for the internal processes. While this will use more video memory it can produce improved results.

Example
wXEffectsStart (wOFF, wON)

wXEffectsEnableDepthPass

Syntax
wXEffectsEnableDepthPass(enable as integer)

Description
Enables a depth rendering pass. This is required for shaders that rely on depth information. Use wON to enable the function.

Example
wXEffectsEnableDepthPass (wON)

wXEffectsAddPostProcessingFromFile

Syntax
wXEffectsAddPostProcessingFromFile(name as zstring ptr, effectType as integer = 0)

Description
Adds a shader feature to the display from a GLSL or HLSL program stored in a file. Shaders do need some programming support so only the XEffects Shaders are supported through the XEffects calls.

The first parameter is the path and file name for the shader program. If you are operating in OpenGL you should use the GLSL extension and when operating in DirectX you shouhld use the HLSL extension.

The second parameter can usually be omitted or set to 0. Only when loading the SSAO shader (not the SSAO composite shader) should it be set to 1.

Example
wXEffectsAddPostProcessingFromFile ("./media/shaders/ssao.glsl", 1)

wXEffectsSetPostProcessingUserTexture

Syntax
wXEffectsSetPostProcessingUserTexture(texture as wTexture)

Description
Sets the user defined post processing texture. This is used internally for the SSAO shader but is used

primarily for the water shader where it defines the specular surface pattern of the water.

You can change the texture through a sequence of images to produce an animated effect.

Example
wXEffectsSetPostProcessingUserTexture (waterTexture(i))

wXEffectsAddShadowToNode

Syntax
wXEffectsAddShadowToNode(node as wNode,
filterType as wFILTER_TYPE = wFT_NONE,
shadowType as wSHADOW_MODE = wSM_BOTH)

Description
Adds the shadowing effect to a node. This controls both recieving and casting shadows.

The filterType defines the amount of sampling that is to be carried out on the node. This can be one of the following settings, increasing the filter increases the quality and also the cost of rendering.

wFT_NONE
wFT_4PCF
wFT_8PCF
wFT_12PCF
wFT_16PCF

The shadow type specifies the type of shadowing applied to the node. This can be set to one of the following settings: -

wSM_RECEIVE
wSM_CAST
wSM_BOTH
wSM_EXCLUDE

Example
wXEffectsAddShadowToNode (roomNode)

wXEffectsRemoveShadowFromNode

Syntax
wXEffectsRemoveShadowFromNode(node as wNode)

Description
Removes the shadowing effect from a node.

Example
wXEffectsRemoveShadowFromNode (roomNode)

wXEffectsExcludeNodeFromLightingCalculations

Syntax
wXEffectsExcludeNodeFromLightingCalculations(node as wNode)

Description
Excludes a node from shadowing calculations.

Example
wXEffectsExcludeNodeFromLightingCalculations (particleNode)

wXEffectsAddNodeToDepthPass

Syntax
wXEffectsAddNodeToDepthPass(node as wNode)

Description
Adds a node to the list of nodes used for calculating the depth pass.

Example
wXEffectsAddNodeToDepthPass (barrierNode)

wXEffectsSetAmbientColor

Syntax
wXEffectsSetAmbientColor(R as uinteger, G as uinteger, B as uinteger, Alpha as uinteger)

Description
Sets the ambient lighting procuded in the scene by the XEffects system.

Example
wXEffectsSetAmbientColor (32,32,32,0)

wXEffectsSetClearColor

Syntax
wXEffectsSetClearColor(R as uinteger, G as uinteger, B as uinteger, Alpha as uinteger)

Description
The XEffects system uses a different background color to the one specified in the wBeginScene call use this call to set this default background color.

Example
wXEffectsSetClearColor (255,250,32,0)

wXEffectsAddShadowLight

Syntax
```
wXEffectsAddShadowLight(
    shadowDimen as uinteger,
    posX as single, byVal posY as single, byVal posZ as single,
    targetX as single, byVal targetY as single, byVal targetZ as single,
    R as single, byval G as single, byval B as single, byval Alpha as single,
    lightNearDist as single, byval lightFarDist as single,
    angleDegrees as single )
```

Description
Adds a special dynamic shadow casting light to the scene, for each of these lights that you add there is a seperate shadow map created and a seperate render pass so for each light you add the scene takes more memory and gets slower.

The first parameter specifies the shadow map resolution for the shadow light. The shadow map is always square, so you need only pass 1 dimension, preferably a power of two between 512 and 2048, maybe larger depending on your quality requirements and target hardware.
The pos parameters specify the lights initial position
The target parameters is the (look at) target for the light
The color setting are the floating point color intensity values of the light
The near and far distance of the light are very important values for determining the reach of the light.
The last parameter is the FOV (Field of view), since the light is similar to a spot light, the field of view will determine its area of influence. Anything that is outside of a lights frustum (Too close, too far, or outside of it's field of view) will be unlit by this particular light, similar to how a spot light works.

Example
```
wXEffectsAddShadowLight ( 512, 200,200,0, 0,0,0, _
                          0.7,0.7,0.6,0.0, 1.0, 1200.0, 89.99 )
```


wXEffectsSetShadowLightPosition

Syntax
```
wXEffectsSetShadowLightPosition( lightIndex as uinteger,
   posX as single, byVal posY as single, byVal posZ as single )
```

Description
Set the position of a shadow light. the index refers to the numerical order in which the lights were added.

Example
```
wXEffectsSetShadowLightPosition ( 0, 200,200,0 )
```


wXEffectsGetShadowLightPosition

Syntax
```
wXEffectsGetShadowLightPosition( lightIndex as uinteger,
   posX as single, byVal posY as single, byVal posZ as single )
```

Description
Get the position of a shadow light. the index refers to the numerical order in which the lights were added.

Example
```
wXEffectsGetShadowLightPosition ( 0, x, y, z )
```


wXEffectsSetShadowLightTarget

Syntax
```
wXEffectsSetShadowLightTarget( lightIndex as uinteger,
   targetX as single, byVal targetY as single, byVal targetZ as single )
```

Description
Set the target location of a shadow light. the index refers to the numerical order in which the lights were added.

Example
```
wXEffectsSetShadowLightTarget ( 0, 25,15,0 )
```

wXEffectsGetShadowLightTarget

Syntax
wXEffectsGetShadowLightTarget(lightIndex as uinteger,
targetX as single, byVal targetY as single, byVal targetZ as single)

Description
Get the target location of a shadow light. the index refers to the numerical order in which the lights were added.

Example
wXEffectsGetShadowLightTarget (0, x, y, z)

wXEffectsSetShadowLightColor

Syntax
wXEffectsSetShadowLightColor(lightIndex as uinteger,
R as single, byval G as single, byval B as single, byval Alpha as single)

Description
Set the target location of a shadow light. the index refers to the numerical order in which the lights were added.

Example
wXEffectsSetShadowLightColor (0, 1.0, 0.75, 0.2, 0.0)

wXEffectsGetShadowLightColor

Syntax
wXEffectsGetShadowLightColor(lightIndex as uinteger,
R as single, byval G as single, byval B as single, byval Alpha as single)

Description
Get the target location of a shadow light. the index refers to the numerical order in which the lights were added.

Example
wXEffectsGetShadowLightColor (0, r, g, b, a)

Materials

Calls for creating and manipulating materials that can be applied to a node to color and texture the object. Basic Materials set common properties like the shininess and reflective color of the objects.

wSetNodeAmbientColor

Syntax
wSetNodeAmbientColor (node As wNode, uColor As Uinteger)

Description
Sets the ambient color of all materials in a node. This color value is created with the FreeBasic RGBA call. The ambient color is a color applied to the whole node as a simulation of ambient lighting reflected from the objects around it.

Example
wSetNodeAmbientColor (object_material, RGBA(128,0,0,0))

wSetNodeDiffuseColor

Syntax
wSetNodeDiffuseColor (node As wNode, uColor As Uinteger)

Description
Sets the diffuse color of all materials in a node. This color value is created with the FreeBasic RGBA call. The diffuse color is the indirectly lit surface colour.

Example
wSetNodeDiffuseColor (object_material, RGBA(128,0,0,0))

wSetNodeSpecularColor

Syntax
wSetNodeSpecularColor (node As wNode, uColor As Uinteger)

Description
Sets the specilar color of all materials in a node. This color value is created with the FreeBasic RGBA call. The specular color is the color of the highlights on the node representing reflections of light sources.

Example
wSetNodeSpecularColor (object_material, RGBA(128,0,0,0))

wSetNodeEmissiveColor

Syntax
wSetNodeEmissiveColor (node As wNode, uColor As Uinteger)

Description
Sets the emissive color of all materials in a node. This color value is created with the FreeBasic RGBA call. The emissive colour is the light 'generated within' the node. Setting this to 255,255,255,255 will make the node appear as though it has the no lighting effect applied to it.

Example
wSetNodeEmissiveColor (object_material, RGBA(128,0,0,0))

wSetNodeColorByVertex

Syntax
wSetNodeColorByVertex (material as wMaterial, affected_property as wCOLOR_MATERIAL)

Description
Sets which aspect of all of the materials in a node is affected by the vertex colour.

affected_property can be one of:

wCM_NONE

Dont use vertex color for lighting

wCM_DIFFUSE
Use vertex color for diffuse light, (the default value)

wCM_AMBIENT
Use vertex color for ambient light

wCM_EMISSIVE
Use vertex color for emissive light

wCM_SPECULAR
Use vertex color for specular light

wCM_DIFFUSE_AND_AMBIENT
Use vertex color for both diffuse and ambient light

Example
wSetNodeColorByVertex (object_material, wCM_NONE)

wMaterialVertexColorAffects

Syntax
wMaterialVertexColorAffects (material as wMaterial, affected_property as wCOLOR_MATERIAL)

Description
Sets which aspect of the material is affected by the vertex colour.

affected_property can be one of:

wCM_NONE
Dont use vertex color for lighting

wCM_DIFFUSE
Use vertex color for diffuse light, (the default value)

wCM_AMBIENT
Use vertex color for ambient light

wCM_EMISSIVE
Use vertex color for emissive light

wCM_SPECULAR
Use vertex color for specular light

wCM_DIFFUSE_AND_AMBIENT
Use vertex color for both diffuse and ambient light

Example
wMaterialVertexColorAffects (object_material, wCM_NONE)

wMaterialSetShininess

Syntax
wMaterialSetShininess (material as wMaterial, shininess as single)

Description
Set how shiny the material is, the higher the value the more defined the highlights.

Example
wMaterialSetShininess (object_material, 20.0)

wMaterialSetSpecularColor

Syntax
wMaterialSetSpecularColor (material as wMaterial, Alpha as uinteger, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set the color of specular highlights on objects with this material applied.

Example
wMaterialSetSpecularColor (object_material, 0, 255, 128, 128)

wMaterialSetDiffuseColor

Syntax
wMaterialSetDiffuseColor (material as wMaterial, Alpha as uinteger, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set the color of diffuse lighting on objects with this material applied.

Example
wMaterialSetDiffuseColor (object_material, 0, 255, 128, 255)

wMaterialSetAmbientColor

Syntax
wMaterialSetAmbientColor (material as wMaterial, Alpha as uinteger, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set the color of ambient light reflected by objects with this material applied.

Example
wMaterialSetAmbientColor (object_material, 0, 64, 128, 255)

wMaterialSetEmissiveColor

Syntax
wMaterialSetEmissiveColor (material as wMaterial, Alpha as uinteger, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set the color of light emitted by objects with this material applied.

Example

wMaterialSetEmissiveColor (object_material, 0, 64, 128, 255)

wMaterialSetMaterialTypeParam

Syntax
wMaterialSetMaterialTypeParam(material as wMaterial, value as single)

Description
Set material specific parameter. Used in a couple of vertex alpha and normal mapping material types.

Example
wMaterialSetMaterialTypeParam(object_material, 0.357)

wSetMaterialBlend

Syntax
wSetMaterialBlend (material as wMaterial, source as wBLEND_FACTOR, destination as wBLEND_FACTOR)

Description
Sets the source and destination surface blend factors for the ONETEXTURE_BLEND material. This is mainly useful in multi-pass rendering, where you render the scene to the display and then render the scene a second time with the ONETEXTURE_BLEND material setting which mixes the existing pixels and the new pixels using the blend setting defined here.

wBLEND_FACTOR can be one of the following values:

BF_ZERO
A fixed value of zero

BF_ONE
A fixed value of one

BF_DST_COLOR
The destination color

BF_ONE_MINUS_DST_COLOR
The inverted destination color

BF_SRC_COLOR
The source color

BF_ONE_MINUS_SRC_COLOR
The inverted source color

BF_SRC_ALPHA
The source alpha value

BF_ONE_MINUS_SRC_ALPHA
The inverted source alpha value

BF_DST_ALPHA
The destination alpha value

BF_ONE_MINUS_DST_ALPHA
The inverted destination alpha value

BF_SRC_ALPHA_SATURATE

Example
wSetMaterialBlend (object_material, BF_SOURCE_COLOR, BF_DST_COLOR)

wSetMaterialLineThickness

Syntax
wSetMaterialLineThickness(material as wMaterial, thickness as single)

Description
Sets the line thickness of none 3D elements associated with this material.

Example
wSetMaterialLineThickness(object_material, 2.0)

Scene

wGetMesh

Syntax
wMesh = wGetMesh(Filename of the mesh object to load as zstring)

Description
Loads the specified mesh ready to be added to the scene. The WorldSim3D engine supports a wide range of mesh types including BSP, MD2, 3DS, Direct X, etc...

Example
DolphinMesh = wGetMesh("Dolphin.x")

wCreateMesh

Syntax
wCreateMesh (mesh_name as zstring ptr, vertex_count as integer, vertices as wVERT, indices_count as integer, indices as ushort) as wMesh

Description
Create a new mesh. You must supply a list of vertices of type wVECT and an array of indices that refer to these vertices. The indices are taken in groups of three joining up the dots defined by the veticies and forming a collection of triangles.

Example
PyramidMesh = wCreateMesh("Pyramid", 5, vertices(0), 18, indicies(0))

wAddHillPlaneMesh

Syntax
wAddHillPlaneMesh (mesh_name As zString Ptr, tileSizeX As Single, tileSizeY As Single, tileCountX As

Integer, tileCountY As Integer, material As uInteger Ptr = 0, hillHeight As Single = 0, countHillsX As Single = 0, countHillsY As Single = 0, textureRepeatCountX As Single = 1, textureRepeatCountY As Single = 1) as wMesh

Description
Creates a hill plane mesh that represents a simple terrain. Many properties have default values allowing a mesh to be created with a simple call

Example
TerrainMesh = wAddHillPlaneMesh("Terrain", 1.0, 1.0, 10, 10)

wCreateBatchingMesh

Syntax
wCreateBatchingMesh ( )

Description
Create a batching mesh that will be a collection of other meshes into a single source mesh. The function of the batching mesh is to avoid the use of large numbers of nodes that adds an overhead to the rendering process that can significantly slow it down. Where you have a forest with a thousand trees you will see a significant increase in performance by batching all of those trees into a smaller number of node.

Returns: A bactching mesh, while this is handled as an wMesh it should only be used with batching mesh commands.

Example
batchingMesh = WCreateBatchingMesh( )

wAddToBatchingMesh

Syntax
wAddToBatchingMesh (
meshBatch as wMesh,
mesh as wMesh,
posX as single = 0.0f, posY as single = 0.0f, posZ as single = 0.0f,
rotX as single = 0.0f, rotY as single = 0.0f, rotZ as single = 0.0f,
scaleX as single = 1.0f, scaleY as single = 1.0f, scaleZ as single = 1.0f)

Description
Adds a mesh to the batching mesh at the specified position, rotation and scale. If each of your meshes requires a different texture you should call wSetMeshMaterialTexture for the mesh you are about to add prior to adding the mesh to the batch.

Example
wAddToBatchingMesh(batchingMesh, treeMesh)

wFinalizeBatchingMesh

Syntax
wFinalizeBatchingMesh (mesh as wMesh, frame number as integer)

Description
Finalises the batching mesh, this should be called once all of the meshes have been added to the batching mesh. The function returns a new mesh object that can be used in all standard mesh calls..

Example
Dim as wMesh newMesh = wFinalizeBatchingMesh(BatchingMesh)

wWriteMesh

Syntax
IrrWriteMesh(mesh as irr_mesh, file_format as wMESH_FILE_FORMAT, save_filename as zstring) as uinteger

Description
Write the first frame of the supplied animated mesh out to a file using the specified file format. The following file formats are supported by WorldSim3D:

Irrlicht Native mesh writer, for static .irrmesh files.
wMW_IRR_MESH

COLLADA mesh writer for .dae and .xml files.
wMW_COLLADA

STL mesh writer for .stl files.
wMW_STL

The function will return the follow error codes:

(0) Could not get mesh writer object
(1) Could not open file
(2) Unable to write the mesh to the file
(3) Successfully wrote file

Example
```
if wWriteMesh( custom_mesh, wMW_IRR_MESH, "mymesh.irr" ) = 3
    ? "Wrote the mesh to file successfully"
```


wRemoveMesh

Syntax
wRemoveMesh(mesh as wMesh)

Description
Removes a mesh from the scene cache, freeing up resources.

Example
wRemoveMesh(my_mesh)

wRenameMesh

Syntax
wRenameMesh(mesh as wMesh, name as zstring ptr)

Description
Rename a loaded mesh through the scene cache, the mesh can then subsequently be loaded again as a different mesh

Example
wRenameMesh(my_mesh, "New Name")

wClearUnusedMeshes

Syntax
wClearUnusedMeshes()

Description
Clears all meshes that are held in the mesh cache but not used anywhere else. Any references to these meshes will become invalid.

Example
wClearUnusedMeshes()

wSetMeshHardwareAccelerated

Syntax
wSetMeshHardwareAccelerated (mesh as wMesh, frame number as integer)

Description
Set the supplied mesh as a Hardware Accelerated object, this offloads the verticies and indicies to hardware support on the graphics card, making the process of rendering those meshes much faster. The feature must be supported on the graphics card and the object must contain over 500 vertices for the operation to be successful. This operation is applied to all mesh buffers in the mesh.

Example
wSetMeshHardwareAccelerated(ShipMesh, 0)

wSetMeshMaterialTexture

Syntax
wSetMeshMaterialTexture(
    mesh as wMesh,
    byval texture as wTexture,
    byval material_index as integer,
    byval buffer as integer = 0)

Description
Apply the supplied texture the specified mesh. Upto four textures can be applied to the material by applying them to different material indicies, these textures can be used by materials or shader functions. Setting a mesh texture will apply the texture to all nodes that use that mesh it can also used for texturing a mesh before it is added to a batch mesh.

Example
wSetMeshMaterialTexture(StatueMesh, stoneTexture, 0)

wGetMeshFrameCount

Syntax
integer = wGetMeshFrameCount (mesh as wMesh)

Description
Gets the number of frames in the supplied mesh.You can use this value to traverse the indicies and vertices in a mesh containing a number of frames.

Example
MeshFrameCount = wGetMeshFrameCount(WolfMesh)

wGetMeshBufferCount

Syntax
integer = wGetMeshBufferCount (mesh as wMesh, frame number as integer)

Description
Gets the number of mesh buffers in the supplied mesh.You can use this value to traverse the indicies and vertices in a mesh containing a number of mesh buffers. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0.

Most meshes only contain one mesh buffer however the artist creating the mesh may decide to break the mesh up into a number of groups of meshes, for example a house might have a roof mesh buffer and a walls mesh buffer.

Example
MeshBufferCount = wGetMeshBufferCount(HouseMesh, 0)

wGetMeshIndexCount

Syntax
integer = wGetMeshIndexCount (mesh as wMesh, frame number as integer, mesh_buffer as integer)

Description
Gets the number of Indicies in the supplied mesh.You can use this value to allocate an array for reading out the list of indicies in a mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Example
MeshIndexCount = wGetMeshIndexCount(MapMesh, 0,0)

wGetMeshIndices

Syntax
wGetMeshIndices (mesh as wMesh, frame number as integer , indicies as ushort, mesh_buffer as integer)

Description
Gets the list of indices in a mesh and copies them into the supplied buffer. Each index references a vertex in the mesh the indices are grouped into three's and together form a triangular surface. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Indices should be the first element of an array or the first integer in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the indices otherwise an overflow will occur and memory will be corrupted.

Example
wGetMeshIndices(MapMesh, 0, Indicies(0),0)

wSetMeshIndices

Syntax
wSetMeshIndices(mesh as wMesh, frame number as integer , indicies as ushort, mesh_buffer as integer)

Description
This sets the value of the list of indicies in a mesh copying them from the supplied buffer. Each index references a vertex in the mesh the indices are grouped into three's and together form a triangular surface. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Indices should be the first element of an array or the first integer in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the indices otherwise an overflow will occur and erroneous values will be written into the mesh causing unpredictable results.

Example
wSetMeshIndices(MapMesh, 0, Indicies(0),0)

wGetMeshVertexCount

Syntax
integer = wGetMeshVertexCount (mesh as wMesh, frame number as integer, mesh_buffer as integer)

Description
Gets the number of Vertices in the supplied mesh.You can use this value to allocate an array for reading out the list of vertices in a mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Example
MeshVertexCount = wGetMeshVertexCount(MapMesh, 0)

wGetMeshVertices

Syntax
wGetMeshVertices (mesh as wMesh, frame number as integer , vertices as wVERT, mesh_buffer as integer)

Description
Gets the list of vertices in a mesh and copies them into the supplied buffer. Each vertex represents a point in the mesh that is the corner of one of the grou pof triangles that is used to construct the mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Vertices should be the first element of an array or the first IRR_VERT structure in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the vertices otherwise an overflow will occur and memory will be corrupted.

Example
wGetMeshVertices(MapMesh, 0, Verticies(0), 0)

wSetMeshVertices

Syntax
wSetMeshVertices(mesh as wMesh, frame number as integer , indicies as IRR_VERT, mesh_buffer as integer)

Description
This sets the value of the list of vertices in a mesh copying them from the supplied buffer. Each vertex represents a point in the mesh that is the corner of one of the grou pof triangles that is used to construct the mesh. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Vertices should be the first element of an array or the first wVERT structure in a pool of allocated memory, it is passed into the wrapper by reference as a pointer. You must ensure that the array you supply is large enough to contain all of the vertices otherwise an overflow will occur and erroneous values will be written into the mesh causing unpredictable results.

Example
wSetMeshVertices(MapMesh, 0, Vertices(0), 0)

wGetMeshVertexMemory

Syntax
wGetMeshVertexMemory (mesh as wMesh, frame number as integer , mesh_buffer as integer)

Description
Get a memory pointer to the vertex memory for the supplied mesh operations can be carried out very quickly on vertices through this function but object sizes and array access needs to be handled by the caller.

Example
Dim as wVERT verts = wGetMeshVertexMemory(MapMesh, 0, 0)

wScaleMesh

Syntax
wScaleMesh(mesh as wMesh, scale as single, frame number as integer = 0, mesh_buffer as integer = 0, source mesh as wMesh = 0)

Description
Scales the verticies in a mesh without affecting the normals, tangents or texture co-ordinates. This is particularly useful for enlarging a mesh without affecting lighting. It should be noted though that scaling the mesh will scale all of the nodes that use it as their source. The scaling is applied unformly to all axis.

Example
wScaleMesh(StatueMesh, 2.0)

wSetMeshVertexColors

Syntax
wSetMeshVertexColors(mesh as wMesh, frame number as integer , vertexColour as uinteger ptr, vertexGroupStartIndicies as uinteger ptr, vertexGroupEndIndicies as uinteger ptr, numberOfGroups as uinteger, mesh_buffer as integer)

Description
This sets the color of groups of verticies in a mesh. You can define any number of groups of verticies and set the color of those group invividually. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Example
```
DIM color(0 to 2) as uinteger
color(0) = RGBA(255,0,0,0)
color(1) = RGBA(255,0,0,0)
color(2) = RGBA(255,0,0,0)
DIM start as uinteger = 0
DIM end as uinteger = 2
wSetMeshVertexColors( MapMesh, 0, @color, @start, @end, 1, 0 )
```


wSetMeshVertexCoords

Syntax
wSetMeshVertexCoords(mesh as wMesh, frame number as integer , vertexCoordinates as wVECTOR Ptr, vertexGroupStartIndicies as uinteger ptr, vertexGroupEndIndicies as uinteger ptr, numberOfGroups as uinteger, mesh_buffer as integer)

Description
This sets the co-ordinates of groups of verticies in a mesh. You can define any number of groups of verticies and set the color of those group invividually. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter mesh buffer 0 will be used.

Example
```
DIM pos(0 to 2) as wVECTOR
pos(0).x = 0 : pos(0).y = 0 : pos(0).z = 0
pos(1).x = 1 : pos(1).y = 0 : pos(1).z = 0
pos(2).x = 0 : pos(2).y = 1 : pos(2).z = 0
DIM start as uinteger = 0
DIM end as uinteger = 2
wSetMeshVertexCoords( MapMesh, 0, @color, @start, @end, 1, 0 )
```


wSetMeshVertexSingleColor

Syntax
wSetMeshVertexSingleColor(mesh as wMesh, frame number as integer , vertexColour as uinteger ptr, vertexGroupStartIndicies as uinteger ptr, vertexGroupEndIndicies as uinteger ptr, numberOfGroups as uinteger, mesh_buffer as integer)

Description
This sets the color of groups of verticies in a mesh. You can define any number of groups of verticies and set the color of those group invividually. If the mesh is animated frame number indicates the number of the frame to recover mesh data for if it is not animated this value should be set to 0. If the mesh contains a number of mesh buffers you can specific which mesh buffer you want to access, if you omit this parameter

mesh buffer 0 will be used.

Example
DIM start as uinteger = 0
DIM end as uinteger = 2
wSetMeshVertexSingleColor(MapMesh, 0, RGBA(255,255,255,255), @start, @end, 1, 0)

wGetMeshBoundingBox

Syntax
wGetMeshBoundingBox(mesh as wMesh, min X as single, min Y as single, min Z as single, min X as single, min Y as single, min Z as single)

Description
Gets the bounding box of a mesh into the supplied variables, the six paramters define the corners of an axis aligned cube that contains the whole mesh.

Example
wGetMeshBoundingBox(MapMesh, topX, topY, topZ, bottomX, bottomY, bottomZ)

wGetRootSceneNode

Syntax
wNode = wGetRootSceneNode( )

Description
Gets the scenes root node, all scene nodes are children of this node

Example
TheScene = wGetRootSceneNode()

wAddMeshToScene

Syntax
wNode = wAddMeshToScene(Mesh object as wMesh)

Description
Adds a mesh to the scene as a new 3D 'node'.

Example
DolphinMesh = wGetMesh("Dolphin.x")
SceneNode = wAddMeshToScene(DolphinMesh)

wAddMeshToSceneAsOcttree

Syntax
wNode = IrrAddMeshToSceneAsOcttree (Mesh object as wMesh)

Description
Adds a mesh to the scene as a new 3D 'node'. This method optimise's the mesh with an Octtree, this is particularly useful for maps where there is a lot of geometry in the mesh but little of it can be seen at any one time. Optimizing your node with this function will result in a large increase in performance.

Example
MapMesh = IrrGetMesh("planet_large.bsp")
MapNode = wAddMeshToSceneAsOcttree(MapMesh)

wAddStaticMeshForNormalMappingToScene

Syntax
wNode = IrrAddStaticMeshForNormalMappingToScene(Mesh object as wMesh)

Description
Adds a mesh to the scene as a static object, the mesh is altered so that it is suitable for the application of a Normal or Parallax mapping material, any animation information is lost.

Example
StoneRoomMesh = wGetMesh("StoneRoom.x")
SceneNode = wAddStaticMeshForNormalMappingToScene(StoneRoomMesh)
wSetNodeMaterialTexture(SceneNode, colorMap, 0)
wSetNodeMaterialTexture(SceneNode, normalMap, 1)
wMaterialSetSpecularColor(wGetMaterial(SceneNode), 0, 0, 0)
wSetNodeMaterialType(SceneNode, wMT_PARALLAX_MAP_SOLID)
' adjust the height of the paralax effect
wMaterialSetMaterialTypeParam(wGetMaterial(SceneNode), 0.035f)

wLoadScene

Syntax
wLoadScene (file_name As zString Ptr)

Description
Loads all meshes and creates nodes for a scene defined within a file created by IrrEdit.

Example
wLoadScene("Map1.wscn")

wSaveScene

Syntax
wSaveScene (file_name As zString Ptr)

Description
Saves the current scene into a file that can be loaded by irrEdit.

Example
wSaveScene("MyScene.wscn")

wGetSceneNodeFromId

Syntax
wNode = wGetSceneNodeFromId(id as integer)

Description
Get a scene node based on its ID and returns null if no node is found. This is particularly useful for obtaining references to nodes created automatically when using wLoadScene.

Example
My_Node = lrrGetSceneNodeFromId(15)

wGetSceneNodeFromName

Syntax
wNode = wGetSceneNodeFromId(id as zstring ptr)

Description
Get a scene node based on its name and returns null if no node is found. This is particularly useful for obtaining references to nodes created automatically when using lrrLoadScene.

Example
My_Node = wGetSceneNodeFromName("Box")

wAddBillBoardGroupToScene

Syntax
wNode = wAddBillBoardToScene (X size of the node as integer, Y size of the node as integer, X position as integer, Y position as integer, Z position as integer)

Description
Adds a billboard to the scene of the specified size and at the specified position. A billboard is a flat 3D textured sprite that always faces towards the camera. You need to texture this element with a separate command.

Example
Billboard = wAddBillBoardToScene(10.0,8.0, 0,0,0)

wAddBillBoardToGroup

Syntax
```
BillboardAddress = wAddBillBoardToGroup (
    group as wNode, _
    sizex as single, sizey as single, _
    x as single = 0, y as single = 0, z as single = 0, _
    roll as single = 0, _
    A as uinteger = 255, R as uinteger = 255, G as uinteger = 255, B as uinteger = 255 )
```

Description
Adds a billboard to a billboard group. There are a number of properties that are used to specify the billboard.

group is the billboard group node
sizex and sizey are the x and y sizes of the billboard
x, y and z define the position of the billboard
roll specifies the number of degrees that the billboard is spun around its center.
A, R, G and B specify the color used for the billboard

Example
```
BillboardAddress = wAddBillBoardToGroup( BillboardGroup,_
    200.0, 200.0, _
    0.0, 0.0, 0.0, _
    0.0, _
```

```
    0, 255, 255, 255 )
```


wAddBillBoardByAxisToGroup

Syntax

```
BillboardAddress = wAddBillBoardByAxisToGroup (
    group as wNode, _
    sizex as single, sizey as single, _
    x as single = 0, y as single = 0, z as single = 0, _
    roll as single = 0, _
    A as uinteger = 255, R as uinteger = 255, G as uinteger = 255, B as uinteger = 255, _
    axis_x as single = 0, axis_y as single = 0, axis_z as single = 0 )
```

Description

Adds a billboard to a billboard group that is fixed to a particular axis these billboards are particularly useful for things like grass..There are a number of properties that are used to specify the billboard.

group is the billboard group node
sizex and sizey are the x and y sizes of the billboard
x, y and z define the position of the billboard
roll specifies the number of degrees that the billboard is spun around its center.
A, R, G and B specify the color used for the billboard
axis_x, axis_y, axis_z a direction around which the billboard is spun to face the camera

Example

```
BillboardAddress = wAddBillBoardByAxisToGroup( BillboardGroup,_
    200.0, 200.0, _
    0.0, 0.0, 0.0, _
    0.0, _
    0, 255, 255, 255, _
    0.0, 1.0, 0.0 )
```


wRemoveBillBoardFromGroup

Syntax

```
wRemoveBillBoardFromGroup  ( group as wNode, billboardAddress as any ptr )
```

Description

Removes the specified billboard from the billboard group

Example

```
wRemoveBillBoardFromGroup ( BillboardGroup, BillboardAddress )
```


wBillBoardGroupShadows

Syntax

```
wBillBoardGroupShadows  ( group as wNode, _
    x as single = 1.0, y as single = 0, z as single = 0, _
    intensity as single = 1.0, ambient as single = 0.0 )
```

Description

Applies lighting to the billboards in a cluster of billboards. This can be used for example to shade the particles in a group of billboards representing a cloud.

group is the group of billboards to which the lighting is to be applied.

x, y and z is the direction from which the light is arriving
intensity is the strength of the light
ambient is the strength of ambient light in the billboard group

Example
wBillBoardGroupShadows(BillBoardGroup, 1.0, 0.0, 0.0, 1.0, 0.5)

wGetBillBoardGroupCount

Syntax
uinteger = wGetBillBoardGroupCount (group as wNode)

Description
Get the number of billboards in the billboard group.

Example
count = wGetBillBoardGroupCount (BillboardGroup)

wBillBoardForceUpdate

Syntax
wBillBoardForceUpdate (group as wNode)

Description
Unlike regular billboards the billboard group does not always update the orientation of the billboards every frame. If you are a long distance away from the billboard group the camera needs to travel a significant distance before the angle has changed enough to warrent an update of all of the billboards verticies to make them point to the camera once more. You may want to force a refresh at some point with this call.

Example
wBillBoardForceUpdate (BillboardGroup)

wAddBillBoardToScene

Syntax
wNode = wAddBillBoardToScene (X size of the node as integer, Y size of the node as integer, X position as integer, Y position as integer, Z position as integer)

Description
Adds a billboard to the scene of the specified size and at the specified position. A billboard is a flat 3D textured sprite that always faces towards the camera. You need to texture this element with a separate command.

Example
Billboard = wAddBillBoardToScene(10.0,8.0, 0,0,0)

wSetBillBoardColor

Syntax
wNode = wSetBillBoardColor (node as wNode, topColor as uinteger, bottomColor as integer)

Description
Set the color of the top and bottom verticies in a billboard applying a vertical graduated shade to it. The

colors should be generated with the FreeBasic RGBA function

Example
wSetBillBoardColor(Billboard, RGBA(255,255,255,255), RGBA(0,0,0,0))

wSetBillBoardSize

Syntax
wNode = wSetBillBoardSize (node as wNode, BillWidth as single, BillHeight as single)

Description
Adds a billboard to the scene of the specified size and at the specified position. A billboard is a flat 3D textured sprite that always faces towards the camera. You need to texture this element with a separate command.

Example
wSetBillBoardSize(Billboard, 10.0, 8.0)

wAddBillboardTextSceneNode

Syntax
wNode = wAddBillboardTextSceneNode (font as wFont, text as wstring, X size of the node as integer, Y size of the node as integer, X position as integer, Y position as integer, Z position as integer, parent as wNode, topColor as uinteger, bottomColor as uinteger)

Description
Adds a text billboard to the scene of the specified size and at the specified position. A text billboard is a flat 3D textured sprite that always faces towards the camera and has the supplied text written onto it. You should not texture this element.

font defines the font that is used to generate the text.

text is a wide character string containing the text you want to display on the billboard.

X_size and Y_size define the width and height of the billboard

X, Y and Z define the position of the billboard.

Parent defines the object that is the parent to this billboard, if there is no parent this should be set to wNO_OBJECT

topColor is the colour value of the top of the text on the billboard. This can be created with hte RGBA command.

bottomColor is the colour value of the bottom of the text on the billboard. This can be created with hte RGBA command.

Example

```
Billboard = wAddBillboardTextSceneNode( _
        font, "Hello World", _
        64.0, 12.0, _
        0.0, 40.0, 0.0, _
        parentNode, _
        RGBA(255,255,0,0), _
        RGBA(255,0,0,255))
```

wAddParticleSystemToScene

Syntax
node as particle_system =wAddParticleSystemToScene (add_emitter)

Description
Adds a particle system to the scene as a node, a particle system is an object that creates and manages hundreds of small billboard like objects that are used to represent smoke, rain and other natural effects. Once created you then need to add emitters and affecters to create and control the particles.

Add emitter can be one of the following values:
wNO_EMITTER
For no default emitter (this is probably the option you will use and you will then add a specific emitter later)
wDEFAULT_EMITTER
To create a default emitter that ejects a thin vertical stream of particles.

Example
Smoke = wAddParticleSystemToScene(wNO_EMITTER)

wAddSkyBoxToScene

Syntax
wNode = wSkyBoxToScene (up_texture as wTexture, down_texture as wTexture, left_texture as wTexture, right_texture as wTexture, front_texture as wTexture, back_texture as wTexture)

Description
Adds a skybox node to the scene this is huge hollow cube that encapsulates the entire scene and has a different texture applied to each of its six surfaces to represent a distant sky or matte scene.

Example
```
SkyBox = wAddSkyBoxToScene( _
    wGetTexture("./media/Skybox_up.jpg"),_
    wGetTexture("./media/Skybox_dn.jpg"),_
    wGetTexture("./media/Skybox_lf.jpg"),_
    wGetTexture("./media/Skybox_rt.jpg"),_
    wGetTexture("./media/Skybox_ft.jpg"),_
    wGetTexture("./media/Skybox_bk.jpg"))
```


wAddSkyDomeToScene

Syntax
wNode = wAddSkyDomeToScene (texture as wTexture, horizontal_res as uinteger, vertical_res as uinteger, texture_percentage as double, sphere_percentage as double, sphere_radius as single)

Description
Adds a skydome node to the scene this is huge hollow sphere (or part of a sphere) that encapsulates the entire scene to represent a distant sky or matte scene. The horizontal and vertical resolution define the number of segments in the mesh of the sphere (setting these too high can quickly produce a very costly mesh). Texture percentage defines the amount of the texture that is mapped to the scene, this should be a value between 0 and 1 (0 being non of the texture and 1 being the whole texture). Finally sphere percentage defines how much of a sphere is created and should be a value between 0 and 2 (0 being none of a sphere, 1 being a hemi-sphere and 2 being a full sphere).

Example

SkyBox = wAddSkyDomeToScene(wGetTexture("./media/domesky.jpg"), 8, 8, 1.0, 2.0, 10000.0);

wAddEmptySceneNode

Syntax
wNode = wAddEmptySceneNode

Description
Adds an empty node to the scene. This is required if you wish to add custom OpenGL commands with no WorldSoim3D Objects.

Example
EmptyNode = wAddEmptySceneNode

wAddTestSceneNode

Syntax
wNode = wAddTestSceneNode

Description
Adds a simple cube object to the scene. This is particularly useful for testing and is a quick and easy way of playing objects into the scene for testing placement.

Example
TestBox = wAddTestSceneNode

wAddCubeSceneNode

Syntax
wNode = wAddCubeSceneNode(size as single)

Description
Adds a simple cube object to the scene with the specified dimensions.

Example
MyCube = wAddCubeSceneNode(10.0)

wAddSphereSceneNode

Syntax
wNode = wAddSphereSceneNode(radius as single, poly_count as integer)

Description
Adds a simple sphere object to the scene of the specified radius and with the specified level of detail. A reasonable value for poly_count would be 16 setting this value too high could produce a very high density mesh and affect your frame rate adversely.

Example
MySphere = wAddSphereSceneNode(0.5, 16)

wAddSphereSceneMesh

Syntax
wAddSphereSceneMesh (name as zstring ptr, ByVal redius as single, ByVal polyCount as integer) as wMesh

Description
Adds a simple sphere node to the scene of the specified radius, with the specified level of detail and with the specified name. A reasonable value for poly_count would be 16 setting this value too high could produce a very high density mesh and affect your frame rate adversely.

Experimental function. Not tested.

wAddWaterSurfaceSceneNode

Syntax
wNode = wAddWaterSurfaceSceneNode (mesh as wMesh, waveHeight as Single = 2.0, waveSpeed as Single = 300.0, waveLength as Single = 10.0, parent as irr_scene_node = 0, id as Integer = -1, positionX as Single = 0, positionY as Single = 0, positionZ as Single = 0, rotationX as Single = 0, rotationY as Single = 0, rotationZ as Single = 0, scaleX as Single = 1.0, scaleY as Single = 1.0, scaleZ as Single = 1.0)

Description
Adds a mesh with a water animator applied to it, the mesh is animated automatically to simulate a water effect across its surface. Many properties are predefined for this node and a convincing water effect can be created simply by supplying the parameter for the mesh, however the node can be positioned, rotated and scaled by this call and the appearance of the waves on its surface can be adjusted.

Example
WaterNode = wAddWaterSurfaceSceneNode(pond_mesh)

wAddClouds

Syntax
wNode = wAddClouds (texture as wTexture, lod as uinteger, depth as uinteger, density as uinteger)

Description
Adds a set of clouds to the scene. These clouds are most appropriate to a cloud effect experienced by a vehicle flying through them and could be of particular use in masking the transition of a spacecraft from an orbital vantage point to a flat terrain object. They do make a nice ordernary cloud effect too but can appear unrealistic when they are directly over the observer.

LOD defeines the level of detail in the cloud, more detail is added into the cloud depending on the distance of the observer from the object. depth defines the depth of recursion when creating the cloud and finally density defines the number of clouds in the cloud object.

Example
CloudNode = wAddClouds(CloudTexture, 3, 1, 500)

wAddCloud

Syntax
```
wAddCloud  (cloudNode as wNode,_
            X as single,_
            Y as single,_
            Z as single  )
```

Description

Add a cloud to a cloud scene node

Experimental function. Not tested.

wAddLensFlare

Syntax
wNode = wAddLensFlare (texture as wTexture)

Description
Adds a lens flare patch object to the scene, this object simulates the effect of bright lights on the optics of a camera., the position of the lens flare can be set and changed with the IrrSetNodePosition command. The lens flare obejct uses a bitmap containing a series of 128x128 images representing stages of the the lens flare effect.

Example
SceneNode = wAddLensFlare(LensTexture)

wAddGrass

Syntax
wNode = wAddGrass (terrain as wTerrain, x as integer, y as integer, patchSize as integer, fadeDistance as single, crossed as integer, grassScale as single, maxDensity as uinteger, dataPositionX as integer, dataPositionY as integer, heightMap as wImage, textureMap as irr_image, grassMap as wImage, grassTexture as wTexture)

Description
Adds a grass object to the scene. Grass objects are associated with terrain and tile terrain objects and are used to place small billboard objects into the scene representing folliage, this implementation of grass creates a large number of grass objects already positioned across the terrain and then dynamically shows or hides them depending on where the camera is within the scene. The grass is also affected with a wind modifier that gently moves the grass as if it were caught in the wind (by setting the speed of the wind to zero the grass will become static and you will see an increase in performance).

The position and size of the patch of grass can be set with x, y, patchSize and grassScale.
FadeDistance controls the distance at which the number of displayed grass elements in that patch are reduced. If this is set to 1.0 then when the cameral is inside the patch all of grass will be displayed but once outside less and less will be shown. By increasing this to 2.0 then all of the grass is shown until the camera is two patches distant. This gives a better appearence but reduces performance as more grass has to be drawn.
crossed can be set to either IRR_ON or IRR_OFF. When off each piece of grass is a separate entity with its own position and rotation. When On grass is paired up and placed to form a cross. Crossed grass can have a better appearance as you rotate around it. However individual grass can give the impression that there is more of it and you can therefore reduce the number of grass blades and increase performance.
MaxDensity controls the number of individual clumps of folliage that are created.
Dataposition X and Y can be used with a large bitmap associated with a tiled terrain and allow the color information to be taken from an offset position on the bitmap.
Heightmap is an image that contains the height of the terrain onto which the grass is placed.
TextureMap is the color map used to color the verticies of the grass and allow you to create areas of dark of light grass, you can use the terrain color map here.
GrassMap is an image used to adjust the height and density of the grass. For example you might have a patch where you dont want to see any grass or a barren patch where you want short stubble.
GrassTexture is the actually texture used for the grass. This RGBA image is automatically broken up into a number of sections that are used to texure different clumps of grass.

Grass usually looks best when it is closely matched to the color of the terrain and to assist with this a new Material Type has been added wMT_TRANSPARENT_ADD_ALPHA_CHANNEL_REF that adds the color of grass texture to the color of the grass which is automatically set to the color of the terrain that it lies upon.

Example
grassNode = wAddGrass (Terrain, x, y, 1024, 1.0, 250, 0, 0, terrainHeight, terrainColor, grassMap, grassTexture)

wAddZoneManager

Syntax
wNode = wAddZoneManager (initialNearDistance as single, initialFarDistance as single)

Description
Adds a zone/distance management node to the scene. This simple but very powerful object has no visible geometry in the scene, it is used by attaching other nodes to it as children. When the camera is further away than the far distance and closer than the near distance to the zone manager all of the zones child objects are made invisible. This allows you to group objects together and automatically have them hidden from the scene when they are too far away to see. By using the near distance you could have two sets of objects in the scene one with high detail for when you are close and another with low detail for when you are far away.

Another way to use the zone manager would be to test when your camera is inside the zones bounding box and switch its visibility on and off manually.

Example
zone = wAddZoneManager(100,300)

wSetZoneManagerProperties

Syntax
wAddZoneManager (zoneManager as irr_node, newNearDistance as single, newFarDistance as single, accumulateChildBoxes as uinteger)

Description
Sets the draw distances of nodes in the zone/distance management node and whether or not the zone manager is to accumulate the bounding boxes of its children as they are added.

Example
wSetZoneManagerProperties(zone, 0, 600, wON)

wSetZoneManagerBoundingBox

Syntax
wSetZoneManagerBoundingBox (zoneManager as wNode, x as single, y as single, z as single, boxWidth as single, boxHeight as single, boxDepth as single)

Description
Allows the user to manually set the bounding box of a zone manager node.

Example
wSetZoneManagerBoundingBox(zone, 0, 0, 0, 100, 100, 100)

wSetZoneManagerAttachTerrain

Syntax
wSetZoneManagerAttachTerrain (zoneManager as wNode, terrain as wTerrain, structureMapFile as zstring ptr, colorMapFile as zstring ptr, detailMapFile as zstring ptr, ImageX as integer, ImageY as integer, sliceSize as integer)

Description
A special feature of the zone manager is its ability to manage tiled terrain nodes, a zone does this by taking position of an attached terrain object that it shares with other zone objects whenever the camera starts to come into range. The terrain object is loaded with new height information, color and detail ready for when it becomes visible to the camera.

The structureMapFile is the name of an RGBA bitmap file that is to be used to set the structure of the terrain. The Alpha channel is used to set the height and the RGB channels are used to set the color of the vertex at that position. This can be used to load lighting into the scene or to load detail map blending into the scene for simple terrain spattering (discussed in the tile terrain section).

The optional color and detail maps are loaded to apply new color and detail maps to the terrain. If either is not used they should be replaced with wNO_OBJECT.

The Image X and Y define the X and Y position of this tile on the structure and color images, so you could load a 1024x1024 structure image and a 1024x1024 detail image in and have your zones form a grid across these large bitmaps.

Finally SilceSize allows you to only process a slice of the terrain on each frame, as a tile is swapped loading in bitmaps and then adjusting what could be 65,000 vertices in a single frame will cause a noticable hiccup in the smooth running of the scene, so by setting the SliceSize you can define how many rows of the terrain are to be processed on each frame.for example if your tile is 128x128 you might process 32 rows, the tile would then be restructured over 4 frames instead of trying to do it all in one.

Note: You can load your images manually to save them with IrrGetImage and IrrGetTexture and let them stay in memory to avoid having to load images while the scene is running however you should stay aware of how much memory you are using especially the graphics card memory used by wGetTexture.

Example
wSetZoneManagerAttachTerrain (Zone(X + Y*ROW_SIZE), Terrain(index), "SunnyValley.tga", "SunnyValley.bmp", wNO_OBJECT, X*112, Y*112, 32)

wSetGrassDensity

Syntax
wSetGrassDensity (grass as irr_node, density as integer, distance as single)

Description
Set grass density, density being the number of grass nodes visible in the scene and distance being the distance at which they can be seen.

Example
wSetGrassDensity (grassNode, 300, 4000)

wSetGrassWind

Syntax
wNSetGrassWind (grass as wNode, strength as single, resoloution as single)

Description
Set the grass wind effect, the strength being the strength of the wind, the resoloution being how often the effect is calculated. By setting the resoloution to zero the wind effect will be stopped and there will be a performance increase however the wind effect adds significantly to the subtle atmosphere of the scene.

Example
wSetGrassWind (grassNode, 3.0, 1.0)

wGetGrassDrawCount

Syntax
uinteger = wGetGrassDrawCount (grass as wNode)

Description
Get the number of grass objects drawn.

Example
VisibleGrass = wGetGrassDrawCount(Grass)

wSetFlareScale

Syntax
wSetFlareScale (flare as wNode, source as single, optics as single)

Description
Sets the scale of optics in the scene. The source is the texture used to surround the light source while the options are the scale of textures in the optics of the camera. Sometimes it is effected to make the scale of the source considerably larger than those of the optics and to scale the effect in the optics down so that their appearence is more subtle.

Example
wSetFlareScale (FlareNode, 2.0, 1.0)

wSetShadowColor

Syntax
wSetShadowColor (Alpha as integer, Red as integer, Green as integer, Blue as integer)

Description
Sets the color of shadows cast by objects in the scene. If you are observing a bright scene you might use a light grey shadow instead of a heavy black shadow to add to realism.

Example
wSetShadowColor(0, 128, 128, 128)

wSetFog

Syntax
wNode = wSetFog (Red as integer, Green as integer, Blue as integer, fogtype as integer, fog_start as single, fog_end as single, density as single)

Description

Set the properties of fog in the scene.

Red, Green and Blue define the fog color, you should set this to the same color as your sky so the scene fogs out nicely into nothing. These are integer values in the range of 0 to 255

Fogtype specifies whether you want the fog to increase in a linear mannar or exponentially - exponential fog usually
looks more atmospheric while linear looks more like a dense sea fog. This may be specified as either
wLINEAR_FOG
wEXPONENTIAL_FOG

Fog start and end specify the distance at which the fog starts and the distance at which the fog reaches its maximum density. The values here will depend on the size and scale of the scene.

Density is only used with exponential fog and determines how quickly the exponential change takes place, good values for this range from 0 to 1

Example
ThinFog = wSetFog (240,255,255, wEXPONENTIAL_FOG, 0.0,8000.0, 0.5)

wDraw3DLine

Syntax
wDraw3DLine(x_start as single, y_start as single, z_start as single, x_end as single, y_end as single, z_end as single, Red as integer, Green as integer, Blue as integer)

Description
Draws a line onto the display using 3D co-ordinates and a specified color.

Example
wBeginScene(240, 255, 255)
wDraw3DLine(0.0, 0.0, 0.0, 0.0, 50.0, 0.0, 0, 255, 0)
wDrawScene

wSetSkyDomeColor

Syntax
wSetSkyDomeColor(dome as wNode, horizontalRed as uinteger, horizontalGreen as uinteger, horizontalBlue as uinteger, zenithRed as uinteger, zenithGreen as uinteger, zenithBlue as uinteger)

Description
Set the color of the verticies in the skydome. Two colors are defined one for the horizon and another for the top of the sky dome, this simulates the type of coloring effects you see in the sky. If you are using a full spherical skydome the horizontal color will be the color at the bottom of the skydome.

Example
' color the skydome so that it is brighter at the horizon and a darker blue at the top of the sky
wSetSkyDomeColor(SkyDome, 128, 128, 255, 64, 64, 255)

wSetSkyDomeColorBand

Syntax
wSetSkyDomeColorBand(dome as wNode, horizontalRed as uinteger, horizontalGreen as uinteger, horizontalBlue as uinteger, bandVerticalPosition as integer, bandFade as single, addative as uinteger)

Description
Creates a horizontal band of color in the skydome, this is mainly useful for creating additional bands of color at the horizon, where your sky is a graduation of blues and then in the morning you have a brighter golden band as the sun rises. The vertical position in the vertex at which you wish to create the band, bandFade defines the amount that the band is faded into the existing skydome color, addative can be wON to add the color of the band to the existing color of the skydome or wOFF to replace it.

Example
```
' add a band of golden color at the horizon
wSetSkyDomeColorBand ( SkyDome, 240,220,128, 24, 0.25, wON )
```


wSetSkyDomeColorPoint

Syntax
```
wSetSkyDomeColorPoint( dome as wNode, Red as uinteger, Green as uinteger, Blue as uinteger,
pointXPosition as single, pointYPosition as single, pointZPosition as single, pointRadius as single,
pointFade as single, addative as uinteger )
```

Description
Set the color of the verticies in the skydome radiating out from a point. This is powerful effect that can be used to color parts of the skydome and create effects to represent the glows of the rising sun or the moon in the sky. The radius is used to limit the distance of the coloring, pointFade defines the amount that the band is faded into the existing skydome color and addative can be wON to add the color of the band to the existing color of the skydome or wOFF to replace it.

Example
```
' add a bright golden circle of light at the same point as the rising sun
wSetSkyDomeColorPoint ( SkyDome, 255,220,96, 1000.0, -250.0, 0.0, 1500.0, 0.75, IRR_ON )
```


wAddLODManager

Syntax
```
node = wAddLODManager ( fadeScale as uinteger = 4, useAlpha as uinteger = wON, callback as any ptr =
0 )
```

Description
Adds a level of detail manager to the scene. The primary use for this node is to add other scene nodes to it as children and have their level of detail controlled automatically. If those nodes are made from loaded meshs different meshes containing different amounts of detail can be displayed at different distances.

The other function of the LOD manager is to fade nodes in and out at a specific distance so they gradually fade rather than disappear abruptly. This is achieved by applying a distance without supplying a mesh.

fadeScale is the number of 1/4 seconds that the node takes to fade out or in. 4 units equals 1 second.
useAlpha specifies whether or not the Alpha color of the object is faded too.
the callback function is called whenever a node is made invisible or visible. this allows you to stop processing hidden nodes.

Example
```
LODManager = wAddLODManager( 4, wON, @NodeChangeCallback )
wAddLODMesh( LODManager,   0.0, LOD1Mesh )
wAddLODMesh( LODManager, 400.0, wNO_OBJECT )
```

wAddLODMesh

Syntax
wAddLODMesh (node as wNode, distance as single, mesh as wMesh)

Description
Set the distance at which a particular mesh is to be applied to child mesh nodes. if no mesh is supplied it specifies the distance at which the node should be faded in an out.

node is the LOD manager node
distance is the distance at which this effect will be applied
mesh is the mesh used at this distance and beyond or null to specifiy the limit of visibility for this node.

Example
```
LODManager = wAddLODMesh( 4, wON, @NodeChangeCallback )
wAddLODMesh( LODManager,   0.0, LOD1Mesh )
wAddLODMesh( LODManager, 400.0, wNO_OBJECT )
```


wSetLODMaterialMap

Syntax
wSetLODMaterialMap (node as wNode, source as wMATERIAL_TYPES, target as wMATERIAL_TYPES)

Description
Specifies which material is used to apply the fade effect for another material type. How this is used will depend on the effect that you want to achieve. By default fading is applied with the wMT_TRANSPARENT_VERTEX_ALPHA material.

node is the LOD manager node
source is the material type your node uses
target is the material type used for the fade effect.

Example
```
wSetLODMaterialMap( LODManager, wMT_TRANSPARENT_ADD_COLOR,
wMT_TRANSPARENT_ADD_COLOR)
```


wAddBoltSceneNode

Syntax
wNode = wAddBoltSceneNode ( )

Description
The bolt is a special scene node that can be used to replicate electrical effects. This command simply adds the bolt you should then make a call to set the bolts properties. This node can be used to simulate lightning and other electrical effects.

Example
```
ElectricNode = wAddBoltSceneNode( )
wSetBoltProperties ( SceneNode, _
        0,90,0, _               ' the start point for the bolt
        0,0,0, _                ' the end point for the bolt
        50, _                   ' the bolt updates every 50 miliseconds
        10, _                   ' the bolt is 10 units wide
        5, _                    ' the bolt is 5 units thick
        10, _                   ' there are 10 sub parts in each bolt
        4, _                    ' there are 4 individual bolts
```

```
    wON, _              ' the end is not connected to an exact point
    RGBA( 255, 255, 255, 0 )) ' Lighting color
```


wSetBoltProperties

Syntax

```
wSetBoltProperties (
    bolt as wNode, _
    startX as single, startY as single, startZ as single, _
    endX as single, endY as single, endZ as single, _
    updateTime as uinteger = 50, _
    radius as uinteger = 10, _
    thickness as single = 5.0, _
    parts as uinteger = 10, _
    bolts as uinteger = 6, _
    steadyend as uinteger = wOFF, _
    boltColor as uinteger = RGBA(0,0,255,255))
```

Description

This sets the properties of a bolt node that simulates an electrical effect. There are a number of properties that control many aspects of the bolt to produce a wide range of appearences..

Start X, Y and Z define the point that the bolt origionates from.

End X,Y and Z define the terminating point for the bolt.

Update time specifies the number of miliseconds between updates to the appearence of the bolt.

Radius is the radius of the entire bolt effect.

Thickness is the thickness of a single electrical element in the bolt.

Parts defines the number of segments the bolt is divided into.

Bolts represents the number of individual electrical arcs that are rendered.

SteadyEnd when set to wON ends in a tight point, when set to IRR_OFF it ends with the same width as the rest of the bolt.

Color specifies the diffuse color that is applied to the bolt.

Example

```
ElectricNode = wAddBoltSceneNode( )
wSetBoltProperties ( SceneNode, _
    0,90,0, _             ' the start point for the bolt
    0,0,0, _              ' the end point for the bolt
    50, _                 ' the bolt updates every 50 miliseconds
    10, _                 ' the bolt is 10 units wide
    5, _                  ' the bolt is 5 units thick
    10, _                 ' there are 10 sub parts in each bolt
    4, _                  ' there are 4 individual bolts
    wON, _                ' the end is not connected to an exact point
    RGBA( 255, 255, 255, 0 )) ' Lighting color
```

wAddBeamSceneNode

Syntax
wNode = wAddBeamSceneNode ( )

Description
The beam is a special scene node that can be used to replicate beam effects like lasers and tracer gun fire. This command simply adds the beam you should then make calls to set the beams properties.
Example
BeamNode = wAddBeamSceneNode ( )
wSetBeamSize (BeamNode, 5.0)
wSetBeamPosition (BeamNode, X,Y,Z, X+100,Y,Z)

wSetBeamSize

Syntax
wSetBeamSize (beam as wNode, size as single)

Description
This call sets the width of a beam node

Example
wSetBeamSize (BeamNode, 5.0)

wSetBeamPosition

Syntax
wSetBeamPosition (beam as wNode, _
startX as single, startY as single, startZ as single, _
endX as single, endY as single, endZ as single)

Description
This call sets the start and end positions of a beam node. The beam will stretch between the two nodes.

Start X, Y and Z define the point that the bolt origionates from.

End X,Y and Z| define the terminating point for the bolt.

Example
wSetBeamPosition (BeamNode, X,Y,Z, X+100,Y,Z)

Nodes

wGetMaterialCount

Syntax
integer = wGetMaterialCount (node as wNode)

Description
Get the number of materials associated with a node.

Example

nummaterials = wGetMaterialCount(StatueNode)

wGetMaterial

Syntax
wMaterial = wGetMaterial(node as wNode, material_index as integer)

Description
Get the material associated with the node at the particular index

Example
current_material = wGetMaterial(StatueNode, index)

wSetNodeMaterialTexture

Syntax
wSetNodeMaterialTexture(node as wNode, texture as wTexture, material_index as integer)

Description
Applys a texture to a node in the scene, how the texture is applied across the surface of the node will depend on the texturing co-ordinates in each of the vectors of the mesh and how they are plotted across the surface of the texture.
Some nodes can have several textures applied to them to create special material effects.

Node refers to a node you have added to the scene.

Texture refers to a texture you have loaded from an image file.

Material is the index number of the material layer, this will usually be 0 or 1.

Example
wSetNodeMaterialTexture(DolphinNode, DolphinTexture, 0)

wSetNodeMaterialFlag

Syntax
wSetNodeMaterialFlag(node as wNode, flag as wMATERIAL_TYPES, value as uinteger)

Description
Sets material properies of a node that will effect its appearance on the screen, each of these properties can be either switched on or off.

Node refers to a node that has been added to the scene.

Flag is one of the following properties:

wMF_WIREFRAME
Render as wireframe outline
wMF_POINTCLOUD
Draw a point cloud instead of polygons
wMF_GOURAUD_SHADING
Render smoothly across polygons
wMF_LIGHTING
Material is effected by lighting
wMF_ZBUFFER

Enable z buffer
wMF_ZWRITE_ENABLE
Can write as well as read z buffer
wMF_BACK_FACE_CULLING
Cull polygons facing away
wMF_FRONT_FACE_CULLING
Cull polygons facing front
wMF_BILINEAR_FILTER
Enable bilinear filtering
wMF_TRILINEAR_FILTER
Enable trilinear filtering
wMF_ANISOTROPIC_FILTER
Reduce blur in distant textures
wMF_FOG_ENABLE
Enable fogging in the distance
wMF_NORMALIZE_NORMALS
Use when scaling dynamically lighted models
wMF_TEXTURE_WRAP
Gives access to all layers texture wrap settings. Overwrites separate layer settings.
wMF_ANTI_ALIASING
Anti-aliasing mode
wMF_COLOR_MASK
ColorMask bits, for enabling the color planes
wMF_COLOR_MATERIAL
ColorMaterial enum for vertex color interpretation

The value should be one of the following to switch the property on or off:
wON
wOFF

Example
wSetNodeMaterialFlag(CharacterNode, wMF_GOURAUD_SHADING, wON)

wSetNodeMaterialType

Syntax
wSetNodeMaterialType(node as wNode, mat_type as wMATERIAL_FLAGS)

Description
Set the way that materials are applied to the node.

Node refers to a node that has been added to the scene.

Mat_type is one of the following properties that is applied to the node:

wMT_SOLID
Standard solid rendering. Only one (first) texture is used, which is supposed to be the diffuse material.

wMT_SOLID_2_LAYER
Solid material with 2 texture layers.
The second is blended onto the first using the alpha value of the vertex colors.
This material is currently not implemented in OpenGL.

wMT_LIGHTMAP
Material type with standard lightmap technique.
There should be 2 textures: The first texture layer is a diffuse map,
the second is a light map. Dynamic light is ignored.

wMT_LIGHTMAP_ADD
... as above but adds levels instead of modulating between them

wMT_LIGHTMAP_M2
... as above but color texture levels are multiplied by 2 for brightening.
Like known in DirectX as D3DTOP_MODULATE2X. Dynamic light is ignored.

wMT_LIGHTMAP_M4
... as above but color texture levels are multiplied by 4 for brightening.
Like known in DirectX as D3DTOP_MODULATE4X. Dynamic light is ignored.

wMT_LIGHTMAP_LIGHTING
Like wMT_LIGHTMAP, but also supports dynamic lighting.

wMT_LIGHTMAP_LIGHTING_M2
Like wMT_LIGHTMAP_M2, but also supports dynamic lighting.

wMT_LIGHTMAP_LIGHTING_M4
Like wMT_LIGHTMAP_M4, but also supports dynamic lighting.

wMT_DETAIL_MAP
2 blended textures: the first is a color map the second at a different scale adds and subtracts from the color to add detail.
Often used for terrain rendering.

wMT_SPHERE_MAP
Look like a reflection of the environment around it.

wMT_REFLECTION_2_LAYER
A reflecting material with an optional non reflecting texture layer.
The reflection map should be set as first texture.

wMT_TRANSPARENT_ADD_COLOR
A transparency effect that simply adds a color texture to the background. The darker the color the more transparent it is.
Only the first texture is used. The new color is calculated by simply adding the source color and the dest color. This means if for example a billboard using a texture with black background and a red circle on it is drawn with this material, the result is that only the red circle will be drawn a little bit transparent, and everything which was black is 100% transparent and not visible. This material type is useful for particle effects.

wMT_TRANSPARENT_ALPHA_CHANNEL
A transparency effect that uses the color textures alpha as a transparency level. That is it makes the material transparent based on the texture alpha channel.
The final color is blended together from the destination color and the texture color, using the alpha channel value as blend factor. Only first texture is used. If you are using this material with small textures, it is a good idea to load the texture in 32 bit mode.

wMT_TRANSPARENT_ALPHA_CHANNEL_REF
Makes the material transparent based on the color texture alpha channel. If the alpha channel value is greater than 127, a pixel is written to the target, otherwise not. This material does not use alpha blending and is a lot faster than wMT_TRANSPARENT_ALPHA_CHANNEL. It is ideal for drawing stuff like leaves of plants, grass etc., because the borders are not blurry but sharp.
Only first texture is used. If you are using this material with small textures and 3d object, it is a good idea to load the texture in 32 bit mode.

wMT_TRANSPARENT_VERTEX_ALPHA
A transparency effect that uses the vertex alpha value.

wMT_TRANSPARENT_REFLECTION_2_LAYER
A transparent & reflecting effect. The first texture is a
reflection map, the second a color map. Transparency is from vertex
alpha. The reflection map should be set as first texture. The transparency
depends on the alpha value in the vertex colors. A texture which will not
reflect can be set as second texture. Please note that this material type is
currently not 100% implemented in OpenGL.

wMT_NORMAL_MAP_SOLID
A solid normal map renderer. First texture is color, second is normal
map. Only use nodes added with wAddStaticMeshForNormalMappingToScene.
This shader runs on vertex shader 1.1 and pixel shader 1.1 capable hardware
and falls back to a fixed function lighted material if this hardware is not
available. Only two lights are supported by this shader, if there are more,
the nearest two are chosen.

wMT_NORMAL_MAP_TRANSPARENT_ADD_COLOR
... as above only with a transparency effect that simply adds
the color to the background. The darker the color the more transparent it is.

wMT_NORMAL_MAP_TRANSPARENT_VERTEX_ALPHA
... as above only with a transparency effect that uses the vertex alpha value

wMT_PARALLAX_MAP_SOLID
Just like wMT_NORMAL_MAP_SOLID, but uses parallax mapping. Looks a lot more more
realistic providing virtual displacement of the surface. Uses the alpha channel
of the normal map for height field displacement. First texture is the color map,
the second should be the normal map. The normal map texture should contain the
height value in the alpha component. Requires vertex shader 1.1 and pixel shader 1.4.

wMT_PARALLAX_MAP_TRANSPARENT_ADD_COLOR
... as above only with a transparency effect that simply adds the color
to the background. The darker the color the more transparent it is.
Using wMT_TRANSPARENT_ADD_COLOR as base material.

wMT_PARALLAX_MAP_TRANSPARENT_VERTEX_ALPHA
... as above only with a transparency effect that uses the vertex alpha value
Using wMT_TRANSPARENT_VERTEX_ALPHA as base material.

wMT_ONE_TEXTURE_BLEND
(not yet added)

wMT_FOUR_DETAIL_MAP
4 grayscale images in the channels of the first texture mixed with the vertex
channels as alpha images

wMT_TRANSPARENT_ADD_ALPHA_CHANNEL_REF
(not yet added)

wMT_TRANSPARENT_ADD_ALPHA_CHANNEL
(not yet added)

wMT_FORCE_32BIT = &h7fffffff
This value is not used. It only forces this enumeration to compile to 32 bit.

Example

wSetNodeMaterialType(WaterNode, wMT_LIGHTMAP)

wSetNodePosition

Syntax
wSetNodePosition(node as wNode, X as single, Y as single, Z as single)

Description
Moves the node to the new position.

Example
wSetNodePosition(CharacterNode, 500.0, 100.7, -192.6)

wSetNodeRotation

Syntax
wSetNodeRotation(node as wNode, X as single, Y as single, Z as single)

Description
Rotate a node to the specified orientaion through its X, Y and Z axis

Example
wSetNodeRotation(CharacterNode, 34.5 0.76, -67.3)

wSetNodeScale

Syntax
wSetNodeScale(node as wNode, X as single, Y as single, Z as single)

Description
Change the scale of a node in the scene making it bigger or smaller in the X, Y and Z axis

Example
wSetNodeScale(CharacterNode, 1.2,1.5,1.2)

wSetNodeRotationPositionChange

Syntax
wSetNodeRotationPositionChange(node as wNode, roll as single, pitch as single, yaw as single, drive as single, strafe as single, elevate as single, forwardVector as wVECTOR ptr, upVector as wVECTOR ptr, offsetVectorCount as integer, offsetVectors as wVECTOR ptr)

Description
Apply a change in rotation and a directional force. we can also optionally recover pointers to a series of vectors. The first is a pointer to a vector pointing forwards the second is a pointer a vector pointing upwards following this are any number of points that will also be rotated (the effect on these points is NOT accumulative so the points should be initialised with their origonal values each time this is called)

Example
wSetNodeRotationPositionChange(SceneNode, roll, pitch, yaw, drive, strafe, elevate, @forwardVector, @upVector, 2, @cameraVector(0))

wDebugDataVisible

Syntax
wDebugDataVisible (node as wNode, visible as integer)

Description
Displays debugging data around a node, this typically means drawing the bounding box around the edges of the node.

There are a series of values for displaying different types of debugging information and not all of them are supported on all node types

0 No Debugging
1 Bounding Box
2 Normals
4 Skeleton
8 Wireframe
16 Transparency
32 Bounding Box Buffers
&hffffffff Everything

Example
wDebugDataVisible (PyramidNode, 1)

wGetNodePosition

Syntax
wGetNodePosition(node as wNode, X as single, Y as single, Z as single)

Description
Gets the position of a node in the scene and stores its X, Y and Z co-ordinates into the supplied variables.

Example
wGetNodePosition(CharacterNode, XPosition, YPosition, ZPosition)

wGetNodeAbsolutePosition

Syntax
wGetNodeAbsolutePosition(node as wNode, X as single, Y as single, Z as single)

Description
Get the absoloute position of the node in the scene this position includes the position changes of all of the nodes parents too.

Example
wGetNodeAbsolutePosition(CharacterNode, XPosition, YPosition, ZPosition)

wGetNodeRotation

Syntax
wGetNodeRotation(node as wNode, X as single, Y as single, Z as single)

Description
Get the rotation of a node in the scene and stores the X, Y and Z rotation values in the supplied variables..

Example
wGetNodeRotation(CharacterNode, XRotation, YRotation, ZRotation)

wGetNodeScale

Syntax
wGetNodeScale(node as wNode, X as single, Y as single, Z as single)

Description
Get the scale of a node in the scene and stores the X, Y and Z scale values in the supplied variables..

Example
wGetNodeScale(CharacterNode, XScale, YScale, ZScale)

wAddChildToParent

Syntax
wAddChildToParent (child as wNode, parent as irr_node)

Description
Attaches the child node to the parent node, whenever you change the parent node the child node changes too. This is useful for putting a cup in a characters hand for example. You can move and rotate the child node to move the object into position against its parent.

Example
wAddChildToParent(CupNode, CharacterNode)

wAddNodeShadow

Syntax
wNode = wAddNodeShadow (node as wNode, mesh as wMesh = 0)

Description
Adds shadows to a node that are cast across other nodes in the scene, shadowing need to be turned on when you call IrrStart. You should analyse the performance of your scene carefully when using this function as it can have a significant effect on your frame rate. You can supply a different mesh to the one used to display the node, this shadow mesh could be a much lower resoloution than that used for your model thereby improving performance.

Example
wAddNodeShadow (CharacterNode)

wSetNodeVisibility

Syntax
wSetNodeVisibility (node as wNode, visible as integer)

Description
This allows you to hide nodes from the display so you can quickly and easily switch objects out to improve performance or create effects liek one node transforming into another node (perhaps in a puff of particle

smoke).

Visible can be one of the following values: -
wINVISIBLE
wVISIBLE

Example
wSetNodeVisibility(CharacterNode, wVISIBLE)

wRemoveNode

Syntax
wRemoveNode(node as wNode)

Description
Removes a node from the scene deleting it.

Example
wRemoveNode(CharacterNode)

wRemoveAllNodes

Syntax
wRemoveAllNodes()

Description
Clears the entire scene, any references to nodes in the scene will become invalid.

Example
wRemoveAllNodes()

wGetNodeParent

Syntax
wNode = wGetNodeParent (node as wNode)

Description
Gets the parent of the specified node.

Example
ParentNode = wGetNodeParent(ChildNode)

wSetNodeParent

Syntax
wSetNodeParent (node as wNode, parent as wNode)

Description
Sets the parent of the specified node.

Example
ParentNode = wSetNodeParent(ChildNode, ParentNode)

wGetNodeFirstChild

Syntax
wNode = wGetNodeFirstChild (node as wNode, searchPosition as any ptr)

Description
Get the first child node of this node, returns 0 if there is no child.

Example
ChildNode = wGetNodeFirstChild (SectorNode, position)

wGetNodeNextChild

Syntax
wNode = wGetNodeNextChild (node as wNode, searchPosition as any ptr)

Description
Get the next child node of this node, returns 0 if there is no child.

Example
ChildNode = wGetNodeNextChild(SectorNode, position)

wIsNodeLastChild

Syntax
integer = wIsNodeLastChild (child as wNode, parent as wNode)

Description
Attaches the child node to the parent node, whenever you change the parent node the child node changes too. This is useful for putting a cup in a characters hand for example. You can move and rotate the child node to move the object into position against its parent.

Example
```
if wIsNodeLastChild( SectorNode, position ) = 0 then
    LastNode = YES
end if
```


wGetNodeID

Syntax
integer = wGetNodeID (node as wNode)

Description
Each node can have a 32 bit signed identification number assigned to them this can be used in collision operations to filter out particular classes of object.

Example
NodeID = wGetNodeID(TreeNode)

wSetNodeID

Syntax
wSetNodeID (node as wNode, id as integer)

Description
Adds a simple cube object to the scene. This is particularly useful for testing and is a quick and easy way of playing objects into the scene for testing placement.

Example
wSetNodeID (TreeNode, 8)

wGetNodeName

Syntax
const zstring ptr = wGetNodeName (node as wNode)

Description
Get the name of the node.

Example
NodeName = wGetNodeName(StatueNode)

wSetNodeName

Syntax
wSetNodeName (node as wNode, name as zstring ptr)

Description
Set the name of a node

Example
wSetNodeName(StatueNode, "HeroStatue")

wGetNodeMesh

Syntax
wMesh = wGetNodeMesh (node as wNode)

Description
Get the mesh that is associated with a node

Example
myMesh = wGetNodeMesh(StatueNode)

wSetNodeMesh

Syntax
wSetNodeMesh (node as wNode, mesh as wMesh)

Description
Sets the mesh used by a node creaed from a mesh model.

Example
wSetNodeMesh(BuildingNode, LowDetailMesh)

wGetNodeBoundingBox

Syntax
integer = wGetNodeBoundingBox (node as wode, x1 as single, y1 as single, z1 as single, x2 as single, y2 as single, z2 as single,)

Description
Gets the coordiantes describing the bounding box of the node into the six supplied variables.

Example
NodeID = wGetNodeBoundingBox(BuildingNode, Xa, Ya, Za, Xb, Yb, Zb)

wGetNodeTransformedBoundingBox

Syntax
integer = wGetNodeTransformedBoundingBox (node as wNode, x1 as single, y1 as single, z1 as single, x2 as single, y2 as single, z2 as single,)

Description
Gets the transformed (absolute value) bounding box of a node into the six supplied variables. So if your node has been moved hundreds of units away from the origion the co-ordinates of its bounding box will also be hundreds of units away corisponding to its real location in the scene.

Example
NodeID = wGetNodeTransformedBoundingBox(BuildingNode, Xa, Ya, Za, Xb, Yb, Zb)

Node Animation

Bone based

wSetNodeAnimationRange

Syntax
wSetNodeAnimationRange(node as wNode, Start Frame as integer, End Frame as integer)

Description
Sets the range of animation that is to be played in the node. An anaimation sequences might run from 0 to 200 frames and a sequence where your character is running might only occupy a portion of this.

Example
wSetNodeAnimationRange(CharacterNode, 50, 75)

wPlayNodeMD2Animation

Syntax
wPlayNodeMD2Animation (node as wNode, sequence as uinteger)

Description
MD2 format models have specific animation sequences contained within them that can be played back with a simple call.

sequence should be one of the following values: -
wMD2_STAND
wMD2_RUN
wMD2_ATTACK
wMD2_PAIN_A
wMD2_PAIN_B
wMD2_PAIN_C
wMD2_JUMP
wMD2_FLIP
wMD2_SALUTE
wMD2_FALLBACK
wMD2_WAVE
wMD2_POINT
wMD2_CROUCH_STAND
wMD2_CROUCH_WALK
wMD2_CROUCH_ATTACK
wMD2_CROUCH_PAIN
wMD2_CROUCH_DEATH
wMD2_DEATH_FALLBACK
wMD2_DEATH_FALLFORWARD
wMD2_DEATH_FALLBACKSLOW
wMD2_BOOM

Example
wPlayNodeMD2Animation(CharacterNode, wMD2_STAND)

wGetJointNode

Syntax
wNode = IrrGetJointNode (node as wNode, joint_name as zstring ptr)

Description
This supplies you with an invisible node that follows the motion of a particular joint in an animated models skeleton. You can use this to attach child nodes that represent objects a person is carrying for example. It can now also be used to manually move the joint.

The name should refer to the name of a joint in the model.

Example
HandNode = wGetJointNode(CharacterNode, "LeftHand")

wSetJointMode

Syntax
wSetJointMode (node as wNode, mode as uinteger)

Description
Sets the animation mode of joints in a node. When using the control mode wAnimateJoints must be called before wDrawScene.

wJOINT_MODE_NONE will result in no animation of the model based on bones
wJOINT_MODE_READ will result in automatic animation based upon the animation defined with calls like wSetNodeAnimationRange
wJOINT_MODE_CONTROL will allow the position of the bones to be set through code

Example
wSetJointMode(CharacterNode, wJOINT_MODE_CONTROL)

wSetNodeAnimationSpeed

Syntax
wSetNodeAnimationSpeed (node as wNode, speed as integer)

Description
Change the speed at which an animation is played for a node. You could use this to make a character run slowly or quickly and still keep its feet on the ground.

Example
wSetNodeAnimationSpeed(CharacterNode, 25)

wGetNodeAnimationFrame

Syntax
uinteger = wGetNodeAnimationFrame(node as wNode)

Description
Get the frame number that is currently being played by the node.

Example
CurrentFrame = wGetNodeAnimationFrame(AnimNode)

wSetNodeAnimationFrame

Syntax
wSetNodeAnimationFrame(node as wNode, frame as integer)

Description
Set the current frame number being played in the animation

Example
wSetNodeAnimationFrame(CharacterNode, 75)

wSetTransitionTime

Syntax
wSetTransitionTime (node as wNode, speed as single)

Description
Sets the transition time across which two poses of an animated mesh are blended. For example a character in a sitting pose can be switched into a lying down pose by blending the two frames, this will provide a more convincing smooth transition instead of a snap change in position. wAnimateJoints must be called before wDrawScene if blending is used.

Example
wSetTransitionTime(CharacterNode, 0.75)

wAnimateJoints

Syntax
wAnimateJoints(node as wNode)

Description
Animates the mesh based on the position of the joints, this should be used at the end of any manual joint operations including blending and joints animated using wJOINT_MODE_CONTROL and wSetNodeRotation on a bone node.

Example
wAnimateJoints(CharacterNode)

Various

wAddCollisionAnimator

Syntax
wAnimator = wAddCollisionAnimator (selector as wSelector, node as wNode, radius x as single, radius y as single, radius z as single, gravity x as single, gravity y as single, gravity z as single, offset x as single, offset y as single, offset z as single)

Description
This adds a collision animator to a node that applies collision detection and gravity to the object. The collision detection will stop the object penetrating through a surface in the objects it is colliding against and will also press it against the surface using gravity.

Selector represents a selection of triangles in the scene, this is usually all of the triangles in a map for instance. Please refer to the section on collision for further details of how to obtain this object.

Radius X, Radius Y and Radius Z define an ellipsoid that defines the area of collision this eliptical shape allows the collision detection to slide the object up steps and even ladders. If you make it too big you might be too large to get through a doorway but if you make it too small you may not be able to climb steps. You should play with these values and find the best ones for your scene.

Gravity X, Gravity Y and Gravity Z work together to specify the force that is applied to the node to make it drop to the ground. Other values could be used to simulate wind effects.

Offset X, Offset Y and Offset Z are used to offset the node by a specific distance from the center of the collision, as the center of the object and the size of your collision ellipsoid vary you can use this to adjust the position of the node and to bring it into contact with the ground.

Example
CollisionAnimator = wAddCollisionAnimator(MapCollision, CameraNode, 30.0,30.0,30.0, 0.0,-3.0,0.0, 0.0,50.0,0.0)

wAddDeleteAnimator

Syntax
wAnimator = wAddDeleteAnimator (node as wNode, milliseconds to deletion as integer)

Description
This animator deletes the node it is attached to after the specified number of milliseconds (1/1000ths of a second). You could use this animator to delete a falling rock for example, all you would need to do is attach the delete animator, a movement animator and then forget about it.

Example
DeleteAnimator = wAddDeleteAnimator(RockNode, 3000)

wAddFlyCircleAnimator

Syntax
wAnimator = wAddFlyCircleAnimator (node as wNode, center x as single, center y as single, center z as single, radius as single, speed as single)

Description
This animator moves the node it is attached to in a circular path.

Center X, Center Y and Center Z define the center of the circular path.

Radius defines the radius of the path

Speed defines the rate the node moves around the circular path

Example
CircleAnimator = wAddFlyCircleAnimator(PowerNode, 0,0,0 50, 20)

wAddFlyStraightAnimator

Syntax
wAnimator = wAddFlyStraightAnimator (node as wNode, start x as single, start y as single, start z as single, end x as single, end y as single, end z as single, time to complete as uinteger, loop path as integer)

Description
This animator makes the node it is attached to move in a straight line from the start to the end point. It would be useful for objects moving on a conveyor belt for example

Start X, Start Y and Start Z specify the start point of the path.

End X, End Y and End Z specify the end point of the path.

Time to complete specifies the number of milliseconds the animator will take to move the node from the start to the end point

Loop path determines if the node will be moved from the start to the end and then stopped or whether the animation will be looped this parameter should be either:
wONE_SHOT
For a single animation and then stop
wLOOP

To continuously repeat the animation

Example
FlyAnimator = wAddFlyStraightAnimator(AnimatedBox, 0,50,-300, 0,50,300, 3000, wLOOP)

wAddRotationAnimator

Syntax
wAnimator = wAddRotationAnimator (node as wNode, x as single, y as single, z as single)

Description
This animator makes the node it is attached to spin around.

X, Y and Z specify the number of radians the object is spun around each axis

Example
RotationAnimator = wAddRotationAnimator(DisplayCaseNode, 0, 0.1, 0)

wAddSplineAnimator

Syntax
wAnimator = wAddSplineAnimator (node as wNode, array size as integer, x as single, y as single, z as single, time to start as integer, speed as single, tightness as single)

Description
This is one of the more difficult to set up of the animators but is very natural looking and powerful. A spline is a curved line that passes through or close to a list of co-ordinates, creating a smooth flight. This animator needs a list of coordinates stored in three arrays, one array each for the X, Y and Z locations of all the points. A good way to get co-ordinates for these arrays is to load in the camera position example program and move your camera to a point and write down its co-ordinates.

Array size specifies how many points there are in your spline motion.

The three arrays X, Y and Z containing co-ordinates are passed by reference as a pointer therefore you should ensure that the array is the correct size otherwise unpredictable results will be obtained.

Time to start specifies the number of milliseconds that must pass before the animation starts.

Speed defines the rate the node moves along the spline curve.

Tightness specifies how tightly the curve is tied to the points (0 is angular and 1 is very loose)

Example
SplineX(0) = -100 : SplineY((0) = 50 : SplineZ((0) = 0
SplineX(1) = 0 : SplineY((1) = 100 : SplineZ((1) = -100
SplineX(2) = 100 : SplineY((2) = 50 : SplineZ((2) = 0
SplineX(3) = 0 : SplineY((3) = 100 : SplineZ((3) = 100
SplineAnimator = wAddSplineAnimator(CameraNode, 4, SplineX(0), SplineY(0), SplineZ(0), 0, 0.5, 1)

wAddFadeAnimator

Syntax
wAnimator = wAddFadeAnimator (node as wNode, milliseconds to deletion as integer, scale as single)

Description
This animator deletes the node it is attached to after the specified number of milliseconds (1/1000ths of a second). During the time while it is waiting to delete it the node is slowly faded to invisibility and is also scaled by the specified amount. You could use this animator to fade and delete an object from a scene that was no longer required like a used medical pack, all you would need to do is attach the fade animator and forget about it.

Example
FadeAnimator = wAddFadeAnimator(MedicalNode, 3000, 0.0)

wRemoveAnimator

Syntax
wRemoveAnimator (node as wNode, node as wAnimator)

Description
This removes an animator from a node. Stopping the animation or cleaning an animator up so you can apply a new one.

Example
wRemoveAnimator(DoorNode, FlyAnimator)

Collision

wGetCollisionGroupFromMesh

Syntax
wSelector = wGetCollisionGroupFromMesh (mesh as wMesh, node as wNode)

Description
Creates a collision object from the triangles contained within the specified mesh as applied to the position, rotation and scale of the supplied node.

Example
ObjectSelector = wGetCollisionGroupFromMesh(SimpleBuildingMesh, MyBuilding)

wGetCollisionGroupFromComplexMesh

Syntax
wSelector = lrrGetCollisionGroupFromComplexMesh (mesh as wMesh, node as wNode)

Description
Creates an optimized triangle selection group from a large complex mesh like a map. This group can then be used in collision functions to collide objects against this node. You need to supply both the mesh the node was created from and the node itself.

Example
MapSelector = wGetCollisionGroupFromComplexMesh(MapMesh, MapNode)

wGetCollisionGroupFromBox

Syntax
wSelector = wGetCollisionGroupFromBox (node as wNode)

Description
Creates a collision object from the bounding box of a node.

Example
ObjectSelector = wGetCollisionGroupFromBox(CharacterNode)

wGetCollisionGroupFromTerrain

Syntax
wSelector = wGetCollisionGroupFromTerrain (node as wNode, level of detail as integer)

Description
Creates a collision object from a terrain node. A higher level of detail improves the collision detection but consumes more resources and can effect the speed of the process.

Example
TerrainSelector = wGetCollisionGroupFromTerrain(TerrainNode, 1)

wAttachCollisionGroupToNode

Syntax
wAttachCollisionGroupToNode (collisionGroup as wSelector, node as wNode)

Description
Attaches a collision group that you have already created from a mesh and a node to another node without duplicating the collision geometry.

Example
wAttachCollisionGroupToNode(boxCollision, anotherBoxNode)

wRemoveCollisionGroup

Syntax
wRemoveCollisionGroup (collisionGroup as wSelector, node as wNode)

Description
Remove the collision selector from memory. This collision selector must not be attached to another collision group when it is removed, the collision group is first removed from the node you supply.

Example
wRemoveCollisionGroup(buildingCollision, buildingNode)

wSetNodeTriangleSelector

Syntax
wSetNodeTriangleSelector (node as wNode, collisionGroup as wSelector)

Description

Assigns a collision group to a specific node..

Example
wSetNodeTriangleSelector(newBuilding, buildingCollision)

wCreateCombinedCollisionGroup

Syntax
wSelector = wCreateCombinedCollisionGroup

Description
Creates a collision object that can be used to combine several collision objects together so you could add a couple of maps and a terrain for example. Initially the combined collision object is empty.

Example
SelectorGroup = wCreateCombinedCollisionGroup

wAddCollisionGroupToCombination

Syntax
wAddCollisionGroupToCombination (combined collision group as wSelector, collision group as wSelector)

Description
Adds a collision object to group of collision objects.

Example
wAddCollisionGroupToCombination(SelectorGroup, MapSelector)
wAddCollisionGroupToCombination(SelectorGroup, TerrainSelector)

wRemoveAllCollisionGroupsFromCombination

Syntax
wRemoveAllCollisionGroupsFromCombination (combined collision group as wSelector)

Description
Empty a collision group object so that you can add different collision groups to it.

Example
wRemoveAllCollisionGroupsFromCombination(SelectorGroup)

wRemoveCollisionGroupFromCombination

Syntax
wRemoveCollisionGroupFromCombination (combined collision group as irr_selector, collision group as wSelector)

Description
Remove a single specified collision object from a group of collision objects.

Example
wRemoveCollisionGroupFromCombination(SelectorGroup, TerrainSelector)

wGetCollisionPoint

Syntax
integer = wGetCollisionPoint (start as wVECTOR, line_end as wVECTOR, collision group as wSelector, collision point as wVECTOR)

Description
Detect the collision point of a ray in the scene with a collision object if a collision was detected 1 is returned and vector collision contains the co-ordinates of the point of collision

Start defines the start point of the ray and End defines the endpoint

Collision group is a selector object created with one of the above functions.

Collision point is the co-ordinates in 3D space of the collision object the ray and the selector object.

Example
collided = wGetCollisionPoint (StartVector, EndVector, CharacterSelector, CollisionVector)

wGetCollisionNodeFromCamera

Syntax
wNode = wGetCollisionNodeFromCamera (camera as wCamera)

Description
A ray is cast through the camera and the nearest node that is hit by the ray is returned. If no node is hit zero is returned for the object

Example
TargetedNode = wGetCollisionNodeFromCamera (CameraNode)

wGetCollisionNodeFromRay

Syntax
wNode = wGetCollisionNodeFromRay (start as wVECTOR, line_end as wVECTOR)

Description
A ray is cast through the supplied coordinates and the nearest node that is hit by the ray is returned. If no node is hit zero is returned for the object

Example
TargetedNode = wGetCollisionNodeFromRay(RayStartVector, RayEndVector)

wGetChildCollisionNodeFromRay

Syntax
wNode = wGetChildCollisionNodeFromRay (node as wNode, idMask as integer, recurse as uinteger, start as wVECTOR, line_end as wVECTOR)

Description
A ray is cast through the supplied coordinates and the nearest node that is hit by the ray is returned. if no node is hit zero is returned for the object, only a subset of objects are tested, i.e. the children of the supplied node that match the supplied id.lif the recurse option is enabled the entire tree of child objects connected to this node are tested.

Example
wGetChildCollisionNodeFromRay (SectorNode, 100, wOFF, StartPoint, EndPoint)

wGetChildCollisionNodeFromPoint

Syntax
wNode = wGetChildCollisionNodeFromPoint (node as wNode, idMask as integer, recurse as uinteger, point as wVECTOR)

Description
The node and its children are recursively tested and the first node that contains the matched point is returned. if no node is hit zero is returned for the object, only a subset of objects are tested, i.e. the children of the supplied node that match the supplied id. if the recurse option is enabled the entire tree of child objects connected to this node are tested.

Example
wGetChildCollisionNodeFromPoint (SectorNode, 100, wON, TestPoint)

wGetCollisionNodeFromScreenCoordinates

Syntax
wNode = IrrGetCollisionNodeFromScreenCoordinates (screen x as integer, screen y as integer)

Description
A ray is cast through the screen at the specified co-ordinates and the nearest node that is hit by the ray is returned. If no node is hit zero is returned for the object.

Example
SelectedNode = wGetCollisionNodeFromScreenCoordinates(MouseX, MouseY)

wGetNodeAndCollisionPointFromRay

Syntax
wNode = wGetNodeAndCollisionPointFromRay (vectorStart as wVECTOR, vectorEnd as wVECTOR, node as wNode, posX as single, posY as single, posZ as single, normalX as single, normalY as single, normalZ as single, id as integer = 0, rootNode as wNode = wNO_OBJECT)

Description
A ray is cast through the specified co-ordinates and the nearest node that has a collision selector object that is hit by the ray is returned along with the coordinate of the collision and the normal of the triangle that is hit. if no node is hit zero is returned for the object. If a node is supplied for the rootNode that tests for collision start from that node and are only tested against that node and its children.

Example
```
wGetRayFromScreenCoordinates ( screen_x, screen_y, CameraNode, StartVector, EndVector )
wGetNodeAndCollisionPointFromRay ( StartVector, EndVector, collidedNode, hitX, hitY, hitZ, normalX, normalY, normalZ, 0, myRoom )
if NOT collidedNode = wNO_OBJECT then
    Print "We hit something"
end if
```

wSetupWsSceneCollision

Syntax
wSetupWsSceneCollision (camera As wCamera)

Description
An all-inclusive collision routine for .wscn scenes.

Experimental function. Not tested.

wGetCollisionResultPosition

Syntax
wGetCollisionResultPosition (selector As wSelector, ellipsoidPosition As wVECTOR, ellipsoidRadius As wVECTOR, velocity As wVECTOR, gravity As wVECTOR, slidingSpeed as single, outPosition As wVECTOR, outHitPosition As wVECTOR, outFalling As Integer)

Description
Collides a moving ellipsoid with a 3d world with gravity and returns the resulting new position of the ellipsoid, the point at which the elipsoid collided with the surface and whether the ellipsoid is falling through the air.

This can be used for moving a character in a 3d world: The character will slide at walls and is able to walk up stairs. The method used how to calculate the collision result position is based on the paper "Improved Collision detection and Response" by Kasper Fauerby.

Example
```
wGetCollisionResultPosition ( _
	collisionGroup, _
	vectPosition, _
	vectRadius, _
	vectVelocity, _
	vectGravity, _
	0.00005, _
	vectResultPosition, _
	vectHitPosition
	areFalling )
wSetNodePosition( rockNode, vectPosition.X, vectPosition.Y, vectPosition.Z )
```


Math

wGetRayFromScreenCoordinates

Syntax
wGetRayFromScreenCoordinates (screen x as integer, screen y as integer, camera as wCamera, ray start as wVECTOR, ray end as wVECTOR)

Description
Gets a ray that goes from the specified camera and through the screen coordinates the information is copied into the supplied start and end vectors. You can then use this ray in other collision operations.

Example

wGetRayFromScreenCoordinates (screen_x, screen_y, CameraNode, StartVector, EndVector)

wGetScreenCoordinatesFrom3DPosition

Syntax
wGetScreenCoordinatesFrom3DPosition (screen x as integer, screen y as integer, at position as wVECTOR)

Description
Screen co-ordinates are returned for the position of the specified 3D co-ordinates as if an object were drawn at them on the screen, this is ideal for drawing 2D bitmaps or text around or on your 3D object on the screen for example in the HUD of an aircraft. After the call Screen X and Screen Y will contain the co-ordinates.

Example
wGetScreenCoordinatesFrom3DPosition (XPosition, YPosition, RocketVector)

wGet3DPositionFromScreenCoordinates

Syntax
wGet3DPositionFromScreenCoordinates (screenx as integer, screeny as integer, x as single, y as single, z as single, camera as irr_camera, normalX as single = 0.0, normalY as single = 0.0, normalZ as single = 1.0, distanceFromOrigin as single = 0.0)

Description
Calculates the intersection between a ray projected through the specified screen co-ordinates and a plane defined from a normal and the distance of that plane from the world origin.

The Parameters X, Y and Z will recieve the 3D position where the line through the screen intersects with the plane.

Example
wGet3DPositionFromScreenCoordinates (ScreenX, ScreenY, XPosition, YPosition, ZPosition, MyCamera)
wSetNodePosition(MyModel, XPosition, YPosition, ZPosition)

wGet2DPositionFromScreenCoordinates

Syntax
wGet2DPositionFromScreenCoordinates (screenx As integer, screeny As integer, x As Single, y As Single, camera As wCamera)

Description
Calculates the intersection between a ray projected through the specified screen co-ordinates and a plane at the world origin.

The Parameters X, Y and Z will recieve the 2D position where the line through the screen intersects with the plane.

Example
wGet2DPositionFromScreenCoordinates (256, 256, x, y, OurCamera)
wSetNodePosition(MyCursor, XPosition, YPosition, ZPosition)

wGetDistanceBetweenNodes

Syntax
distance = wGetDistanceBetweenNodes (nodeA as wNode, nodeA as wNode)

Description
The distance between two nodes is measured using fast maths functions that will show inaccuracies. Useful for when it is nessecary to test distances between many nodes.

Example
Dim As Single Distance = wGetDistanceBetweenNodes(nodeA, nodeB)

wAreNodesIntersecting

Syntax
test = wAreNodesIntersecting (nodeA as wNode, nodeA as wNode)

Description
Tests whether the bounding boxes are two nodes are intersecting. Bounding boxes are axis aligned and do not rotate when you rotate the nodes. This should be kept in mind when testing for collisions.

Example
```
If NOT wAreNodesIntersecting ( nodeA, nodeB ) = 0 Then
    Print "Collision"
End If
```


wIsPointInsideNode

Syntax
wNode = wIsPointInsideNode (node as wNode, X as Single, Y as Single, Z as Single)

Description
Determine if the specified point is inside the bounding box of the node.

Example
```
If NOT wIsPointInsideNode ( node, X, Y, Z ) = 0 Then
    Print "Point is inside Node"
End If
```


Camera

wAddFPSCamera

Syntax
wCamera = wAddFPSCamera (ByVal parent As wSceneNode = 0, ByVal rotateSpeed As Single = 100.0, ByVal moveSpeed As Single = 0.1, ByVal id As Integer = -1, ByVal keyMapArray As SKeyMap Ptr = 0, ByVal keyMapSize As Integer = 0, ByVal noVericalMovement As Byte = 0, ByVal jumpSpeed As Single = 0.0)

Description
Adds a 'first person shooter' style camera into the scene that will be used to define the view point and target point and other attributes of the view into the 3D scene. If you haven't captured mouse and keyboard events

this camera can be controlled with the cursor keys and the mouse.

Example
FPSCamera = wAddFPSCamera

wAddCamera

Syntax
wCamera = wAddCamera (camera X as single, camera Y as single, camera Z as single, target X as single, target Y as single, target Z as single)

Description
Adds a camera to into the scene that will be used to define the view point and target point and other attributes of the view into the 3D scene. Animators and other node functions can be applied to this node.

Camera X, Camera Y and Camera Z define the view point of the camera.

Target X, Target Y and Target Z define the target of the camera,

Example
CameraObject = wAddCamera(100,0,0, 0,-10,0)

wAddMayaCamera

Syntax
wCamera = wAddMayaCamera (parent as wNode, rotateSpeed as single, zoomSpeed as single, moveSpeed as single)

Description
Adds a Maya style camera to into the scene the user can click with the left, middle and right mouse buttons to move, zoom and rotate the camera.

rotateSpeed the speed at which the camera revolves
zoomSpeed the speed at which the camera zooms in and out
moveSpeed the speed at which the camera moves

Example
CameraObject = wAddMayaCamera(wNO_OBJECT, 100.0, 100.0, 100.0)

wSetCameraTarget

Syntax
wSetCameraTarget (camera as wCamera, X as single, Y as single, Z as single)

Description
The camera view point can be moved by simply using the lrrSetNodePosition function but this operation will change the point that the camera is pointing at.

Example
wSetCameraTarget (CameraObject, 0, 50, 0)

wGetCameraTarget

Syntax
wGetCameraTarget (camera as wCamera, X as single, Y as single, Z as single)

Description
Get the point in space that the camera is looking at. The point is copied into the supplied X, Y and Z variables

Example
wGetCameraTarget (CameraObject, LookAtX, LookAtY, LookAtZ)

wGetCameraUpDirection

Syntax
wGetCameraUpDirection (camera as wCamera, X as single, Y as single, Z as single)

Description
Get the up vector of a camera object into the supplied variables, this controls the upward direction of the camera and allows you to roll it for free flight action. This specifies a point in space at which the top of the camera points.

Example
wGetCameraUpDirection (CameraObject, TopOfCamPointsAtX, TopOfCamPointsAtY, TopOfCamPointsAtZ)

wSetCameraUpDirection

Syntax
wSetCameraUpDirection (camera as wCamera, X as single, Y as single, Z as single)

Description
Set the up vector of a camera object, this controls the upward direction of the camera and allows you to roll it for free flight action. This specifies a point in space at which the top of the camera points.

Example
wSetCameraUpDirection (CameraObject, TopOfCamPointsAtX, TopOfCamPointsAtY, TopOfCamPointsAtZ)

wGetCameraOrientation

Syntax
wGetCameraOrientation (camera as wCamera, X as wVECTOR, Y as wVECTOR, Z as wVECTOR)

Description
Gets the vectors describing the camera direction useful after the camera has been revolved.

Example
IrrGetCameraOrientation (CameraObject, VectorX, VectorY, VectorZ)

wSetCameraClipDistance

Syntax

wSetCameraClipDistance (camera as wCamera, distance as single)

Description
A camera clips objects in the distance that may be a part of the scene to increase rendering performance without requiring you to manage adding and deleting the objects from the view. This defines the distance beyond which no polygons will be drawn.

Example
wSetCameraClipDistance (CameraObject, 12000)

wSetActiveCamera

Syntax
wSetActiveCamera (camera as wCamera)

Description
When you have several camera objects in the scene you can use this call to define which of them is to be used to look through when drawing the scene.

Example
wSetActiveCamera(CameraObject)

wSetCameraFOV

Syntax
wSetCameraFOV (camera as wCamera, fov as single)

Description
Sets the field of vision of the camera a wide field of vision will give a distorted perspective, if the angle is too narrow the display will feel restricted. The value is in radians and has a default value of PI / 2.5

Example
wSetCameraFOV(CameraObject, PI / 2)

wSetCameraAspectRatio

Syntax
wSetCameraAspectRatio (camera as wCamera, aspectRatio as single)

Description
Sets the aspect ratio of the camera in the same way you think of standard screens and widescreens. A widescreen usually has an aspect ratio of 16:9 or 16/9 = 1.78. The camera apect ratio is set up automatically however if you are using split screen effects you may need to change the camera aspect ratio.

Example
wSetCameraAspectRatio(CameraObject, 1.78)

wRevolveCamera

Syntax
wRevolveCamera (camera as wCamera, yaw as single, pitch as single, roll as single, drive as single, strafe as single, elevate as single)

Description
Revolve the camera using quaternion calculations, this will help avoid gimbal lock associated with normal Rotations and is ideal for spacecraft and aircraft.

The command takes six parameters that control yaw (turning left and right), pitch (tilting up and down), roll (rolling left and right), drive (moving forwards and backward), strafe (moving left and right) and finally elevate (moving up and down)

Many thanks to RogerBorg for this.

Example
wRevolveCamera (CameraObject, CameraYaw, CameraPitch, CameraRoll, CameraDrive, CameraDrive, CameraStrafe, CameraElevate)

wSetCameraUpAtRightAngle

Syntax
wSetCameraUpAtRightAngle (camera as wCamera)

Description
Set the camera up at a right angle to the camera vector.

Example
wSetCameraUpAtRightAngle (CameraObject)

wSetCameraOrthagonal

Syntax
wSetCameraOrthagonal (camera as wCamera, distanceX as single, distanceY as single, distanceZ as single)

Description
Set the projection of the camera to an orthagonal view, where there is no sense of perspective. The distance to the target adjusts the width and height of the camera view, essentially the smaller it is the larger the object will appear.

Example
wGetNodePosition(MyTarget, tarX, tarY, tarZ)
wGetNodePosition(MyCamera, camX, camY, camZ)
wSetCameraOrthagonal (MyCamera, camX-tarX, camY-tarY, camZ-tarZ)

Lighting

wAddLight

Syntax
wNode = wAddLight (x as single, y as singlez as single, red as single, green as single, blue as single, size as single)

Description
Adds a light into scene to naturally illuminate your scene.

X, Y and Y defines the coordinates of the light in the scene.

Red, Green and Blue define the intensities of the lighting for those colors. This is a fractional number ranging from 0 upwards the higher the value the brighter the light.

Size specifies the radius of effect of the light

Example
WarningLight = wAddLight (0, 100, 50, 0.5,0.5,0.5, 50)

wSetAmbientLight

Syntax
wSetAmbientLight (Red as single, Green as single, Blue as single)

Description
Sets the ambient lighting level in the scene, ambient light casts light evenly across the entire scene and can be used to increase the overall lighting level. If should never be greater that the brightness of the darkest area of your scene, it can however reduce the number of lights you need in the scene.

The Red, Green and Blue components of this lighting is supplied as integers in the range or 0 to 255

Example
wSetAmbientLight(72, 64, 64)

wSetLightAmbientColor

Syntax
wSetLightAmbientColor(Light as wNode, Red as single, Green as single, Blue as single)

Description
Ambient color emitted by the light, ambient light casts light evenly across the entire scene and can be used to increase the overall lighting level. If should never be greater that the brightness of the darkest area of your scene, it can however reduce the number of lights you need in the scene.

The Red, Green and Blue components of this lighting is supplied as singles specifying the brightness in each color channel

Example
wSetLightAmbientColor(SceneLight, 1.0, 0.1, 0.7)

wSetLightAttenuation

Syntax
wSetLightAttenuation(Light as wNode, Red as single, Green as single, Blue as single)

Description
Changes the light strength fading over distance. Good values for distance effects use (1.0, 0.0, 0.0) and simply add small values to the second and third element.

Example
wSetLightAttenuation(SceneLight, 1.0, 0.08, 0.07)

wSetLightCastShadows

Syntax
wSetLightCastShadows(Light as wNode, cast_shadows as uinteger)

Description
Specifies whether the light casts shadows in the scene or not. Shadowing must be enabled in the IrrStart call and also on the nodes in the scene.

Example
wSetLightCastShadows(SceneLight, wON)

wSetLightDiffuseColor

Syntax
wSetLightDiffuseColor(Light as wNode, Red as single, Green as single, Blue as single)

Description
wSetLightDiffuseColor

The Red, Green and Blue components of this lighting is supplied as singles specifying the brightness in each color channel

Example
wSetLightDiffuseColor(SceneLight, 1.0, 1.0, 0.8)

wSetLightFalloff

Syntax
wSetLightFalloff(Light as wNode, Falloff as single)

Description
The light strength's decrease between Outer and Inner cone.

Example
wSetLightFalloff(SceneLight, 0.8)

wSetLightInnerCone

Syntax
wSetLightInnerCone(Light as wNode, InnerCone as single)

Description
The angle of the spot's inner cone. Ignored for other lights.

Example
wSetLightInnerCone(SceneLight, 0.4)

wSetLightOuterCone

Syntax
wSetLightOuterCone(Light as wNode, OuterCone as single)

Description
The angle of the spot's outer cone. Ignored for other lights.

Example
wSetLightOuterCone(SceneLight, 0.9)

wSetLightRadius

Syntax
wSetLightRadius(Light as wNode, Radius as single)

Description
Radius of light. Everything within this radius be be lighted. If some artefacts can be seen when the radius is changed in this instance simply make the radius a little large

Example
wSetLightRadius(SceneLight, 50.2)

wSetLightSpecularColor

Syntax
wSetLightSpecularColor(Light as wNode, Red as single, Green as single, Blue as single)

Description
Sets the ambient lighting level in the scene, ambient light casts light evenly across the entire scene and can be used to increase the overall lighting level. If should never be greater that the brightness of the darkest area of your scene, it can however reduce the number of lights you need in the scene.

The Red, Green and Blue components of this lighting is supplied as singles specifying the brightness in each color channel

Example
wSetLightSpecularColor(SceneLight, 1.0, 1.0, 1.0)

wSetLightType

Syntax
wSetLightType(Light as wNode, Light_type as wLIGHT_TYPE)

Description
The type of the light. All lights default to a point light but can be changed with this setting to one of the following values:

wLT_POINT
wLT_SPOT
wLT_DIRECTIONAL

Example
wSetLightType(SceneLight, wLT_SPOT)

Terrain

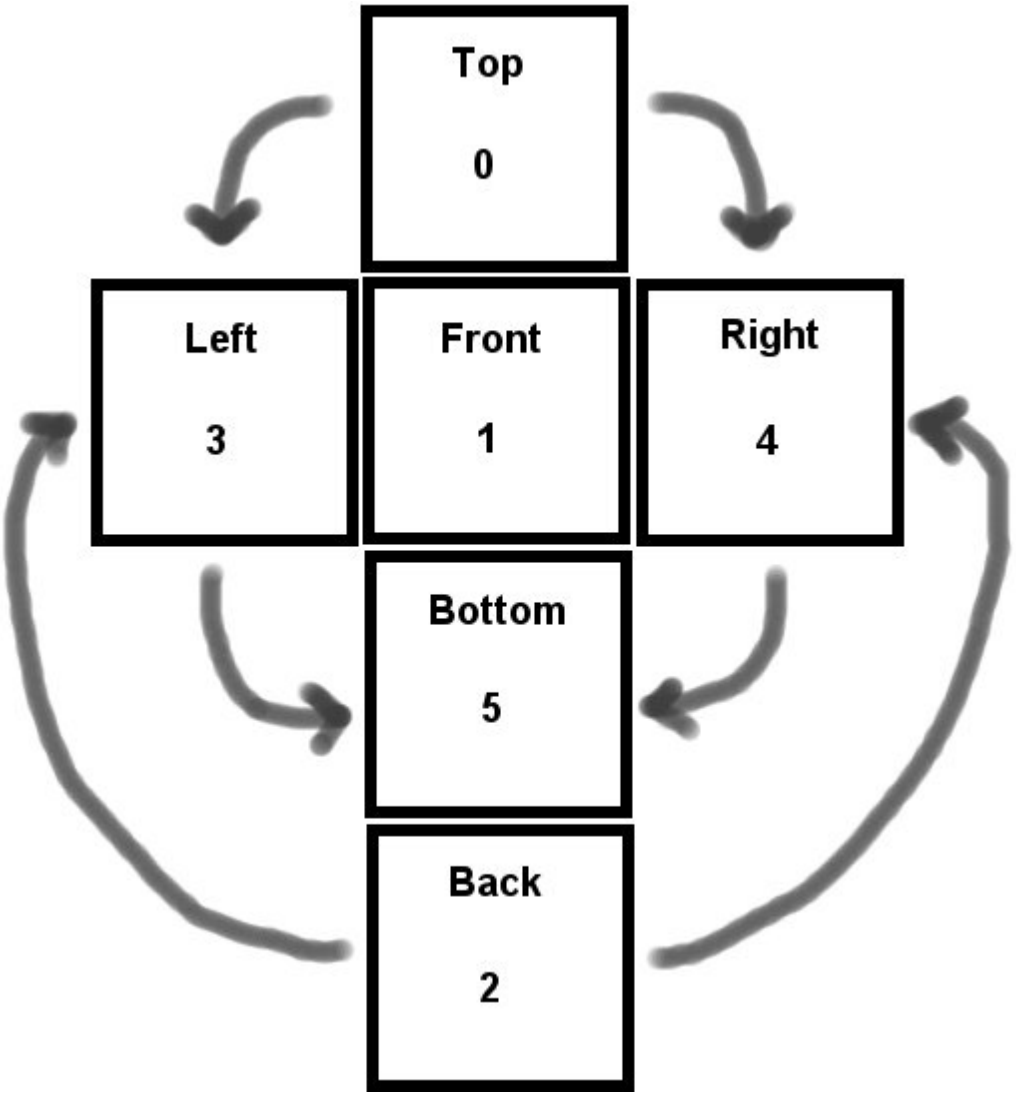
wAddSphericalTerrain

Syntax
wTerrain = wAddSphericalTerrain (topPath as zstring ptr, frontPath as zstring ptr, backPath as zstring ptr, leftPath as zstring ptr, rightPath as zstring ptr, bottomPath as zstring ptr, xPosition as single = 0.0, yPosition as single = 0.0, zPosition as single = 0.0, xRotation as single = 0.0, yRotation as single = 0.0, zRotation as single = 0.0, xScale as single = 1.0, yScale as single = 1.0, zScale as single = 1.0, vertexAlpha as integer = 255, vertexRed as integer = 255, vertexGreen as integer = 255, vertexBlue as integer = 255, smoothing as integer = 0, spherical as integer = 0, maxLOD as integer = 5, patchSize as wTERRAIN_PATCH_SIZE = wTPS_17)

Description
Creates a spherical terrain that represents a planetary body. When using this terrain it is better to think of it as a cube rather than a sphere, in fact it is a cube that is distorted so that its surface becomes spherical, like a cube it has a top, bottom, left, right, front and back and co-ordinates are thought of as being at position X,Y on cube face N. In someways this makes working with placing things on the object simpler as you can think of it as six flat surfaces.

The first six paths are the path of six gray scale bitmaps where bright pixels are high points on the terrain and black pixels are low points.
The position, rotation and scale of the terrain are specified with the next series of parameters.
Four parameters are used to set the vertex color of all the verticies in the terrain.
Smoothing is used to smooth out the contours of the hills in the terrain.
maxLOD and patchSize allow you to adjust the level of detail within the terrain although it is usually best to leave these to default values.

When creating heightmaps for the faces of the terrain you will need to ensure that the height of pixels at the edge of adjoining sides of the terrain are the same otherwise large visible cracks will appear at the edges of the faces, the easiest way to do this is to create terrain texture and then copy and/or rotate it onto its adjacent face. You can get some suprisingly effective planets and asteroids with textures as small as 32x32 but the object also runs well with a terrain size at the maximum 256 x 256.



Example
```
Terrain = wAddSphericalTerrain( _
          "moonbase_top.bmp", _
          "moonbase_front.bmp", _
          "moonbase_back.bmp", _
          "moonbase_left.bmp", _
          "moonbase_right.bmp", _
          "moonbase_bottom.bmp", _
          px,py,pz,  rx,ry,rz, 64.0,64.0,64.0, _
          0, 255, 255, 255,  -30, 0, 4, wTPS_17 )
```


wAddTiledTerrain

Syntax
```
wAddTiledTerrain alias "IrrAddTiledTerrain" ( _
      heightMapPath as zstring ptr, _
      texturePath as zstring ptr, _
      followNode as wNode, _
      terrainWidth as integer = 1000, _
      terrainHeight as integer = 1000, _
      tileSize as single = 1.0, _
      meshSize as integer = 100, _
```

heightmapScale as single = 10.0, _
smoothing as integer = 0.0, _
materialType as wMATERIAL_TYPES = wMT_DETAIL_MAP) as wNode

Description
Create a tiled terrain node from a highfield map

Experimental function. Not tested.

wAddTerrain

Syntax
wTerrain = wAddTerrain (path as zstring ptr, xPosition as single = 0.0, yPosition as single = 0.0, zPosition as single = 0.0, xRotation as single = 0.0, yRotation as single = 0.0, zRotation as single = 0.0, xScale as single = 1.0, yScale as single = 1.0, zScale as single = 1.0, vertexAlpha as integer = 255, vertexRed as integer = 255, vertexGreen as integer = 255, vertexBlue as integer = 255, smoothing as integer = 0, maxLOD as integer = 5, patchSize as wTERRAIN_PATCH_SIZE = wTPS_17)

Description
Creates a terrain object from a gray scale bitmap where bright pixels are high points on the terrain and black pixels are low points. You will inevitablly have to rescale the terrain during the call or after it is created. The Terrain object is a special dynamic mesh whose resoloution is reduced in the distance to reduce the number of triangles it consumes.

Path is the filename of a gray scale image used to define the contours of the surface.
xPosition, yPosition and zPosition define the position of the terrain
xRotation, yRotation and zRotation define the rotation of the terrain
xScale, xScale and xScale define the scale of the terrain
vertexAlpha, vertexRed, vertexGreen, vertexBlue, define the vertex color of all points in the terrain.
smoothing allows you to define whether the contours of the surface of the terrain are smoothed over.
maxLOD and patchsize control the properties of the level of detail calculations applied to the terrain, it is recommended that these are left at default values.

Example
TerrainNode = wAddTerrain("CanyonsHeightField.bmp")

wScaleTerrainDetailTexture

Syntax
wScaleTerrainDetailTexture (terrain as wTerrain, X as single, Y as single)

Description
As a terrain object is a particularly huge mesh when textured are applied to it they look extremely pixelated. To get over this effect a terrain object can have two materials applied to it, one to give general surface color and a second that is copied across the surface like tiles to give a rough detailed texture. This call specifies the scaling of this detail texture.

Example
wScaleTerrainDetailTexture (TerrainNode, 20, 20)

wSetSphericalTerrainTexture

Syntax
wSetSphericalTerrainTexture (terrain as wTerrain, topTexture as wTexture, frontTexture as wTexture,

backTexture as wTexture, leftTexture as wTexture, rightTexture as wTexture, bottomTexture as wTexture, materialIndex as uinteger)

Description
Apply six textures to the surface of a spherical terrain. By using the material index you can set the color or the detail maps

Example
```
wSetSphericalTerrainTexture ( TerrainNode, _
                              "moobbase_col_top.bmp", _
                              "moobbase_col_front.bmp", _
                              "moobbase_col_back.bmp", _
                              "moobbase_col_left.bmp", _
                              "moobbase_col_right.bmp", _
                              "moobbase_col_bottom.bmp", _
                              0 )
```


wLoadSphericalTerrainVertexColor

Syntax
wLoadSphericalTerrainVertexColor (terrain as wTerrain, topMap as wImage, frontMap as wImage, backMap as wImage, leftMap as wImage, rightMap as wImage, bottomMap as wImage)

Description
Apply six images to the vertex colors of the faces, this is useful for setting the verticies so that they can be used with simple terrain spattering described in the section on tiled terrains above.

Example
```
wLoadSphericalTerrainVertexColor ( TerrainNode, _
                                   "moobbase_vert_top.bmp", _
                                   "moobbase_vert_front.bmp", _
                                   "moobbase_vert_back.bmp", _
                                   "moobbase_vert_left.bmp", _
                                   "moobbase_vert_right.bmp", _
                                   "moobbase_vert_bottom.bmp" )
```


wScaleSphericalTexture

Syntax
wScaleSphericalTexture (terrain as wTerrain, X as single, Y as single)

Description
As the surfaces of a sphereical terrain object are a particularly huge mesh when textures are applied to them they look extremely pixelated. To get over this effect a spherical terrain object can have two materials applied to it, one to give general surface color and a second that is copied across the surface like tiles to give a rough detailed texture. This call specifies the scaling of this detail texture.

Example
```
wScaleSphericalTexture ( SphericalTerrainNode, 20, 20 )
```


wGetTerrainHeight

Syntax
single = wGetTerrainHeight (terrain as wTerrain, X as single, Y as single)

Description
Get the height of a point on a terrain. This can be a particularlly fast and accurate way to move an object over a terrain.

Example
Y = wGetTerrainHeight (TerrainNode, X, Z)

wGetTiledTerrainHeight

Syntax
single = wGetTiledTerrainHeight (terrain as wTerrain, X as single, Z as single)

Description
Get the height of a point (or coordinate) on a tiled terrain. This can be a particularlly fast and accurate way to move an object over a tiled terrain.

Example
Y = wGetTiledTerrainHeight (TerrainNode, X, Z)

Experimental function. Not tested.

wGetSphericalTerrainSurfacePosition

Syntax
wGetSphericalTerrainSurfacePosition (terrain as wTerrain, face as integer, logicalX as single, logicalZ as single, X as single, Y as single, Z as single)

Description
Get the surface position of a logical point on the terrain. You supply a face number and a logical X, Y position on that face and this call will return the height of that point on the terrain sphere inside the X, Y, Z parameters.

Note: By subtracting the center of the sphere from this co-ordinate and converting this vector to angles you can find the upward direction of the point on the surface.

Example
wGetSphericalTerrainSurfacePosition (TerrainNode, wTOP_FACE, buggyX, buggyZ, X, Y, Z)

wGetSphericalTerrainSurfacePositionAndAngle

Syntax
wGetSphericalTerrainSurfacePosition (terrain as wTerrain, face as integer, logicalX as single, logicalZ as single, X as single, Y as single, Z as single, RotationX as single, RotationY as single, RotationZ as single)

Description
Get the surface position and angle of a logical point on the terrain. This is not the normal of the surface but essentially the angles to the gravitational center.

Example
wGetSphericalTerrainSurfacePositionAndAngle (Terrain, F, I, J, PX,PY,PZ, RX,RY,RZ)

wGetSphericalTerrainLogicalSurfacePosition

Syntax
wGetSphericalTerrainSurfacePosition (terrain as wTerrain, face as integer, logicalX as single, logicalZ as single, X as single, Y as single, Z as single)

Description
Convert a co-ordinate into a logical Spherical terrain position.

Please note that this calculation is not 100% accurate, it is advised that the translation is done at altitude and the difference either ignored or blended as the observer decends.

Note: The height above the surface can be calculated simply by calculating the length of the center of the planet to the surface and then the center of the planet to the space coordinate and subracting the two
Note: The momentum could be calculated by converting two samples and then measing the difference in height and X and Z on the face

Example
wGetSphericalTerrainLogicalSurfacePosition (Terrain, X, Y, Z, face, LX, LZ)

wAddTerrainTile

Syntax
wTerrain = wAddTerrainTile (image as wImage, tileSize as integer = 256, dataX as integer = 0, dataY as integer = 0, xPosition as single = 0.0, yPosition as single = 0.0, zPosition as single = 0.0, xRotation as single = 0.0, yRotation as single = 0.0, zRotation as single = 0.0, xScale as single = 1.0, yScale as single = 1.0, zScale as single = 1.0, smoothing as integer = 1, maxLOD as integer = 5, patchSize as wTERRAIN_PATCH_SIZE = wTPS_17)

Description
Creates a tilable terrain object from a gray scale bitmap where bright pixels are high points on the terrain and black pixels are low points. You will inevitablly have to rescale the terrain during the call or after it is created. The Terrain object is a special dynamic mesh whose resoloution is reduced in the distance to reduce the number of triangles it consumes.

Unlike the origonal terrain object the tileable terrain object can be attached to other terrain tile objects without being affected by cracks between tiles caused by the level of detail mechanism. When working with tile terrains it should be noted that the terrain is internally divided up into patches that are patchSize - 1 and there is always one invisible row of patches at the top and left of the terrain. Essentially this means that if your tileSize is 128 x 128 the visible size of your terrain will be 112 x 112 (with a patchSize of wTPS_17)

Note: Tiled Terrain object can be automatically control with the Zone Manager objects please refer to them for further details.

Image is an image file loaded with IrrGetImage and containing a gray scale image used to define the contours of the surface.
TileSize defines the size of the terrain independantly of the size of the image used to create it
xPosition, yPosition and zPosition define the position of the terrain
xRotation, yRotation and zRotation define the rotation of the terrain
xScale, xScale and xScale define the scale of the terrain
smoothing allows you to define whether the contours of the surface of the terrain are smoothed over.
maxLOD and patchsize control the properties of the level of detail calculations applied to the terrain, it is recommended that these are left at default values.

Example
TerrainNode = wAddTerrainTile(EasterIslandImage, 128)

wGetTerrainTileHeight

Syntax
single = wGetTerrainTileHeight (terrain as wTerrain, X as single, Y as single)

Description
Get the height of a point on a terrain tile. This can be a particularlly fast and accurate way to move an object over a terrain.

Example
Y = wGetTerrainTileHeight (TerrainNode, X, Z)

wScaleTileTexture

Syntax
wScaleTileTexture (terrain as wTerrain, X as single, Y as single)

Description
As a tile terrain object is a particularly huge mesh when textured are applied to it they look extremely pixelated. To get over this effect a terrain object can have two materials applied to it, one to give general surface color and a second that is copied across the surface like tiles to give a rough detailed texture. This call specifies the scaling of this detail texture.

Example
wScaleTileTexture (TerrainNode, 20, 20)

wAttachTile

Syntax
wAttachTile (terrain as wTerrain, neighbouring_terrain as wTerrain, edge as integer)

Description
Set the adjacent tile to this tile node. To avoid cracks appearing between tiles, tiles need to know which tiles are their neighbours and which edges they are attached too.

Example
wAttachTile(TerrainNorth, TerrainSouth, TOP_EDGE)
wAttachTile(TerrainSouth, TerrainNorth, BOTTOM_EDGE)

Макросы:
#define TOP_EDGE 0
#define BOTTOM_EDGE 1
#define LEFT_EDGE 2
#define RIGHT_EDGE 3

wSetTileColor

Syntax
wSetTileColor(terrain as wTerrain, image as wImage, x as integer, y as integer)

Description
Loads the tile vertex colors from the supplied image file. The RGB values are used to set the color of the verticies. This can either be for loading precalculated lighting into the scene or it can be used with the new wMT_FOUR_DETAIL_MAP material type to define the weight of each of the greyscale detail maps in the

RGB channels of the detail map. The x and y values can be used to load the structure from a specific point on the bitmap.

Example
wSetTileColor(TerrainCove, CoveStructure, 0, 0)

wSetTileStructure

Syntax
wSetTileStructure (terrain as wTerrain, image as wImage, x as integer, y as integer)

Description
Loads the tile structure from the supplied image file. Unlike the image in the origonal call to create a terrain tile this image has a different structure. The image should be in RGBA format, the alpha value is used to set the height of the terrain and the RGB values are used to set the color of the verticies. This can either be for loading precalculated lighting into the scene or it can be used with the new wMT_FOUR_DETAIL_MAP material type to define the weight of each of the greyscale detail maps in the RGB channels of the detail map. The x and y values can be used to load the structure from a specific point on the bitmap.

Example
wSetTileStructure(TerrainCove, CoveStructure, 0, 0)

Particles

wAddParticleEmitter

Syntax
wEmitter = wAddParticleEmitter (particle system as wParticleSystem, settings as wPARTICLE_EMITTER)

Description
Adds a particle emitter to the particle system, this creates particles and controls how they move and when they are to be removed. It requires a very large number of parameters to define this flexible effect and as such these parameters are stores in a special wPARTICLE_EMITTER structure.

Example
MyEmitter = wAddParticleEmitter(SmokeParticles, SmokeEmitter)

wAddAnimatedMeshSceneNodeEmitter

Syntax
wEmitter = wAddAnimatedMeshSceneNodeEmitter(particle_system as wParticleSystem, node as wNode, use_normal_direction as uinteger, normal_direction_modifier as single, emit_from_every_vertex as integer, settings as wPARTICLE_EMITTER)

Description
Creates a particle emitter for an animated mesh scene node

Parameters:
node - Pointer to the animated mesh scene node to emit particles from
use_normal_direction - If true, the direction of each particle created will be the normal of the vertex that it's emitting from. The normal is divided by the normalDirectionModifier parameter, which defaults to 100.0f.

normal_direction_modifier - If the emitter is using the normal direction then the normal of the vertex that is being emitted from is divided by this number.
emit_from_every_vertex - If true, the emitter will emit between min/max particles every second, for every vertex in the mesh, if false, it will emit between min/max particles from random vertices in the mesh.

A large number of additional parameters are also required to define this flexible effect and as such these parameters are stores in a special wPARTICLE_EMITTER structure. The box size properties of this structure are unused in this call

Example
MyEmitter = wAddAnimatedMeshSceneNodeEmitter (SmokeParticles, SceneNode, 1, 0.25, 0, SmokeEmitter)

wAddFadeOutParticleAffector

Syntax
wAffector = wAddFadeOutParticleAffector (particle_system as wParticleSystem)

Description
Adds a fade out affector to the particle system, this fades the particles out as they come to the end of their lifespan and stops them 'popping' out of existance. This creates a convincing effect for fire and smoke in particular.

Example
MyAffector = wAddFadeOutParticleAffector(SmokeParticles)

wAddGravityParticleAffector

Syntax
wAffector = wAddGravityParticleAffector (particle system as wParticleSystem, x as single, y as single, z as single)

Description
Adds a gravity affector to the particle system, this gradually pulls the particles in the direction of the effect, although it is called a gravity effector it can be used to make a wind effect and have the particles drift off to the side.

X, Y and Z define the force that is applied to the particles over time.

Example
MyAffector = wAddGravityParticleAffector(SmokeParticles, -0.1, 0, 0)

wAddParticleAttractionAffector

Syntax
wAffector = wAddParticleAttractionAffector(particle_system as wParticleSystem, x as Single, y as Single, z as Single, speed as Single = 1.0, attract as uinteger = 1, affectX as uinteger = 1, affectY as uinteger = 1, affectZ as uinteger = 1)

Description
Creates a point attraction affector. This affector modifies the positions of the particles and attracts them to a specified point at a specified speed per second.

Parameters:
x,y,z - Point to attract particles to.

speed - Speed in units per second, to attract to the specified point.
attract - Whether the particles attract or detract from this point use the constants wATTRACT or wREPEL (defaults to wATTRACT)
affectX - Whether or not this will affect the X position of the particle, use 1 to effect the position and 0 to leave it unaffected (defaults to true).
affectY - Whether or not this will affect the Y position of the particle, use 1 to effect the position and 0 to leave it unaffected (defaults to true).
affectZ - Whether or not this will affect the Z position of the particle, use 1 to effect the position and 0 to leave it unaffected (defaults to true).

Example
MyAffector = wAddParticleAttractionAffector(SmokeParticles, 0.0, 10.0, 0.0, 20.0, wATTRACT, 1, 1, 1)

wAddRotationAffector

Syntax
wAffector = wAddRotationAffector (particle_system as wParticleSystem, Speed_X as Single, Speed_Y as Single, Speed_Z as Single, pivot_X as Single, pivot_Y as Single, pivot_Z as Single)

Description
Creates a rotation affector. This affector modifies the positions of the particles and attracts them to a specified point at a specified speed per second.

Parameters:
speed x,y,z - Rotation in degrees per second
pivot x,y,z - Point to rotate the particles around

Example
MyAffector = wAddRotationAffector(SmokeParticles, -120.0, 0.0, 0.0, 0.0, 0.0, 0.0)

wAddStopParticleAffector

Syntax
wAffector = wAddStopParticleAffector (particle_system as wParticleSystem, time as uinteger, emitter as wEmitter)

Description
The stop particle affector waits for the specified period of time to elapse and then stops the specified emitter emitting particles by setting its minimum and maximum particle emission rate to zero. The emitter can easily be started up again by changing its emission rate.

Parameters:
Time - The number of milliseconds to elapse before the particles are stopped
Emitter - The particle generating object to stop

Example
MyAffector = wAddStopParticleAffector(SmokeSystem, 1000, smoke_emitter)

wAddParticlePushAffector

Syntax
wAffector = wAddParticlePushAffector (particle_system as wParticleSystem, x as single, y as single, z as single, speedX as single, speedY as single, speedZ as single, far as single, near as single, column as single, distant as integer)

Description
Creates a point push affector. This affector modifies the positions of the particles and pushes them toward or away from a specified point at a specified speed per second. The strength of this effect is adjusted by a near and a far distance. Beyond the far distance the particle is not effected at all and the closer you get to the center of the effect the stronger the force is.

If a near distance is defined (a value greater than 0.0) the effect is somewhat different, particles closer to the center than the near distance are not effected at all, and the stongest point of the effect is always halfway between the near and far limits, for example if your near distance was 25.0 and your far distance was 75.0 the strongest force would be applied to particles at a distance of 50.0

If a column width is defined the effect will only take place in a vertical column that is that wide, this is useful for fountains of water where as the water spreads out of the column a gravity affector can take over.

By adjusting these parameters and the strength you can create columns, spheres, shells and rings of effect that can, in combination, push particles in complex motions

Parameters:
x, y, z - Point to attract particles to or repel particles away from
speedX, speedY, speedZ - A vector describing the strength of the effect
Far - Furthest distance of effect
Near - Closest distance of effect
Column - The width of a vertical column in which the push affector has influence, somewhat like the column of water in a fountain
Distant - Use wON to apply the same force in the same directionto all particles and use wOFF to apply a force that radiates away from the center of the effect

Example
```
MyAffector = wAddParticlePushAffector ( ColumnOfSmoke, 0, 0, 0,  0, 100, 0,  100, 0.0, 0.0, wOFF )
```


wAddColorMorphAffector

Syntax
```
wAffector = wAddColorMorphAffector  ( particle_system as wParticleSystem, numberOfParticles as uinteger, particlecolors as uinteger ptr, particletimes as uinteger ptr, smooth as uinteger )
```

Description
This clever effect that allows you to provide an array of colors and an optional array of times that effect the color of the particle over its lifetime, the particle could start off bright orange and fade away into grey and then black for example.

Parameters:
numEntries - the number of entries in the supplied table
colors - the table of colors
time - the table of times at which each color becomes active
smooth - Use wON to smoothly blend between colors and use wOFF to sharply switch between colors

Example
```
DIM colors(0 to 4) as uinteger = {  wMakeARGB(0,255,255,128), _ ' yellow white
                                    wMakeARGB(0,255,128,0), _ ' yellow
                                    wMakeARGB(0,128,64,0), _ ' orange
                                    wMakeARGB(0,0,0,128), _ ' slight blue
                                    wMakeARGB(255,0,0,0) } ' black and faded
DIM times(0 to 4) as uinteger = {500, 800, 1250, 1500, 2000 }

MyAffector = wAddColorMorphAffector( Fire.particles, 5, @colors(0), @times(0), wON )
```

wAddSplineAffector

Syntax
wAffector = wAddSplineAffector (particle_system as wParticleSystem, VertexCount as uinteger, verticies as wVERT ptr, speed as single, tightness as single, attraction as single, deleteAtEnd as uinteger)

Description
This clever effect that allows you to create an affector that moves the particles along the path of a spline for very controled and complex particle motion.

Parameters:
VertexCount - Is the number of points in your spline
Verticies - Is an array of wVERT objects defining the X,Y and Z position of points
Speed - is the speed with which particles are moved along the spline
tightness - is the tightness of the curve of the spline
attraction - is how closely the particles are atracted to the curve of the spline
deleteAtEnd - Use wON to delete the particles when they reach the end of the spline and use wOFF to allow the particles to be deleted naturally.

Example
```
DIM splineverticies(0 to 3) as wVERT

splineverticies(0).x = 0.0 : splineverticies(0).y = 0.0 : splineverticies(0).z = 0.0
splineverticies(1).x = 0.0 : splineverticies(1).y = 20.0 : splineverticies(1).z = 25.0
splineverticies(2).x = 0.0 : splineverticies(2).y = 40.0 : splineverticies(2).z = -25.0
splineverticies(3).x = 0.0 : splineverticies(3).y = 60.0 : splineverticies(3).z = 0.0

wAddSplineAffector ( NeonLight.particles, 4, @splineverticies(0),  2.0, 1.0, 5.0,  wON )
```


wRemoveAffectors

Syntax
wRemoveAffectors (particle system as wParticleSystem)

Description
Removes all affectors from a particle system, you might use this if you wanted to change the direction or strength of the wind for example.

Example
```
wRemoveAffectors( SmokeParticles )
```


wSetParticleEmitterDirection

Syntax
wSetParticleEmitterDirection(particle_emitter as wEmitter, x as single, y as single, z as single)

Description
Set direction the emitter emits particles.

Example
```
wSetParticleEmitterDirection( MyEmitter, 0.0, 0.4, 0.0 )
```

wSetParticleEmitterMinParticlesPerSecond

Syntax
wSetParticleEmitterMinParticlesPerSecond(particle_emitter as wEmitter, min_particles_per_second as uinteger)

Description
Set minimum number of particles the emitter emits per second.

Example
wSetParticleEmitterMinParticlesPerSecond(MyEmitter, 32)

wSetParticleEmitterMaxParticlesPerSecond

Syntax
wSetParticleEmitterMaxParticlesPerSecond(particle_emitter as wEmitter, max_particles_per_second as uinteger)

Description
Set maximum number of particles the emitter emits per second.

Example
wSetParticleEmitterMaxParticlesPerSecond(MyEmitter, 100)

wSetParticleEmitterMinStartColor

Syntax
wSetParticleEmitterMinStartColor(particle_emitter as wEmitter, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set minimum starting color for particles.

Example
wSetParticleEmitterMinStartColor(MyEmitter, 255, 192, 128)

wSetParticleEmitterMaxStartColor

Syntax
wSetParticleEmitterMaxStartColor(particle_emitter as wEmitter, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Set maximum starting color for particles.

Example
wSetParticleEmitterMaxStartColor(MyEmitter, 255, 192, 128)

wSetParticleAffectorEnable

Syntax
wSetParticleAffectorEnable(particle_affector as wAffector, Enable as uinteger)

Description
Enable or disable an affector. Setting the value to 1 enables the affector, setting it to 0 disables it. wON and wOFF can be used also.

Example
wSetParticleAffectorEnable(MyAffector, wOFF)

wSetFadeOutParticleAffectorTime

Syntax
wSetFadeOutParticleAffectorTime(particle_affector as wAffector, FadeFactor as float)

Description
Alter the fadeout affector changing the fade out time.

Example
wSetFadeOutParticleAffectorTime(MyAffector, 2000.0)

wSetFadeOutParticleAffectorTargetColor

Syntax
wSetFadeOutParticleAffectorTargetColor(particle_affector as wAffector, Red as uinteger, Green as uinteger, Blue as uinteger)

Description
Alter the fadeout affector changing the target color to the affector fades to over time.

Example
wSetFadeOutParticleAffectorTargetColor(MyAffector, 16, 8, 0)

wSetGravityParticleAffectorDirection

Syntax
wSetGravityParticleAffectorDirection(particle_affector as wAffector, x as single, y as single, z as single)

Description
Alter the direction and force of gravity for a gravity affector.

Example
wSetGravityParticleAffectorDirection(MyAffector, 0.2, 0.1, 0.0)

wSetGravityParticleAffectorTimeForceLost

Syntax
wSetGravityParticleAffectorTimeForceLost(particle_affector as wAffector, time_force_lost as single)

Description
Set the time in milliseconds when the gravity force is totally lost and the particle does not move any more.

Example
wSetGravityParticleAffectorTimeForceLost(MyAffector, 800.0)

wSetParticleAttractionAffectorAffectX

Syntax
wSetParticleAttractionAffectorAffectX(particle_affector as wAffector, affect as uinteger)

Description
Set whether or not an atraction affector will affect particles in the X direction.. Setting the value to 1 enables the effect, setting it to 0 disables it. IRR_ON and IRR_OFF can be used also.

Example
wSetParticleAttractionAffectorAffectX(MyAffector, wON)

wSetParticleAttractionAffectorAffectY

Syntax
wSetParticleAttractionAffectorAffectY(particle_affector as wAffector, affect as uinteger)

Description
Set whether or not an atraction affector will affect particles in the Y direction.. Setting the value to 1 enables the effect, setting it to 0 disables it. wON and wOFF can be used also.

Example
wSetParticleAttractionAffectorAffectY(MyAffector, wON)

wSetParticleAttractionAffectorAffectZ

Syntax
wSetParticleAttractionAffectorAffectZ(particle_affector as wAffector, affect as uinteger)

Description
Set whether or not an atraction affector will affect particles in the Z direction.. Setting the value to 1 enables the effect, setting it to 0 disables it. wON and wOFF can be used also.

Example
wSetParticleAttractionAffectorAffectZ(MyAffector, wON)

wSetParticleAttractionAffectorAttract

Syntax
wSetParticleAttractionAffectorAttract(particle_affector as wAffector, affect as uinteger)

Description
Set whether or not the particles are attracted or repelled from an attractor effector.. Use the values wATTRACT and wREPEL for convienience.

Example
wSetParticleAttractionAffectorAttract(MyAffector, wATTRACT)

wSetParticleAttractionAffectorPoint

Syntax

wSetParticleAttractionAffectorPoint(particle_affector as wAffector, x as single, y as single, z as single)

Description
Set the point that particles will attract to when affected by this attractor affector.

Example
wSetParticleAttractionAffectorPoint(MyAffector, wATTRACT)

wSetRotationAffectorPivotPoint

Syntax
wSetRotationAffectorPivotPoint(particle_affector as wAffector, x as single, y as single, z as single)

Description
Set the point that particles will rotate about when affected by this rotation affector.

Example
wSetRotationAffectorPivotPoint(MyAffector, wATTRACT)

wSetFurthestDistanceOfEffect

Syntax
wSetFurthestDistanceOfEffect(particle_affector as wAffector, newDistance as single)

Description
Set the furthest distance of effect on particles affected by the push affector.

Example
wSetFurthestDistanceOfEffect(MyAffector, 100.0)

wSetNearestDistanceOfEffect

Syntax
wSetNearestDistanceOfEffect(particle_affector as wAffector, newDistance as single)

Description
Set the nearest distance of effect on particles affected by the push affector.

Example
wSetNearestDistanceOfEffect(MyAffector, 10.0)

wSetColumnDistanceOfEffect

Syntax
wSetColumnDistanceOfEffect(particle_affector as wAffector, newDistance as single)

Description
Set the column distance of effect on particles affected by the push affector.

Example
wSetColumnDistanceOfEffect(MyAffector, 20.0)

wSetCenterOfEffect

Syntax
wSetCenterOfEffect(particle_affector as wAffector, x as single, y as single, z as single)

Description
Set the center of the effect of particles affected by the push affector.

Example
wSetCenterOfEffect(MyAffector, 0.0, PushHeight, 0.0)

wSetStrengthOfEffect

Syntax
wSetStrengthOfEffect(particle_affector as wAffector, x as single, y as single, z as single)

Description
Set the strength of the effect of particles affected by the push affector.

Example
wSetStrengthOfEffect(MyAffector, PipeVelocity, 0.0, 0.0)

GUI

wGUISetFont

Syntax
wGUISetFont (font as wFont)

Description
Sets the font used by the GUI.

Example
```
' set the font used by the GUI
BitmapFont = wGetFont ( "./media/Bitmap_font.bmp" )
wGUISetFont( BitmapFont )
```


wGUISetColor

Syntax
```
wGUISetColor ( font as wFont_
                element as wGUI_COLOR_ELEMENT, _
                red as integer, _
                green as integer, _
                blue as integer, _
                alpha as integer )
```

Description
Sets the color of an element used by the GUI.

Example

wGUISetColor(wGUI_CE_3D_FACE, 192, 255, 255, 255)
wGUISetColor(wGUI_CE_BUTTON_TEXT, 0, 64, 64, 255)

wGUIClear

Syntax
wGUIClear ()

Description
Clears all GUI objects from the display.

Example
wGUIClear()

wGUIRemove

wGUIRemove
Syntax
wGUIRemove (object as guiElement)

Description
Removes the specified GUI object from the display.

Example
wGUIRemove(myButton)

wGUIGetText

wGUIGetText
Syntax
wGUIGetText (object as guiElement)

Description
Gets the text associated with a GUI object.

Example
DIM myString as wstring = wGUIGetText(myEditBox)

wGUISetText

Syntax
wGUISetText (object as guiElement, text as wstring)

Description
Sets the text of a GUI object.

Example
DIM fpsString as wstring * 256
fpsString = "FPS: " + Str(wGetFPS)
wGUISetText(myButton, fpsString)

wAddWindow

Syntax
wAddWindow (title as wstring ptr, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, modal as uinteger, parent as guiElement) as guiElement

Description
Creates an empty window that can form the frame to contain other controls.

Title is a wide string that contains the title of the window.

Top X, Top Y, Bottom X and Bottom Y define a box in which the window is drawn

Modal determines if the window locks out the rest of the interface until it is closed:
wGUI_MODAL
wGUI_NOT_MODAL

Parent defines the parent object of this window. This can be ommited if the window has no parent.

Example
windowObject = wAddWindow("A Window", 4,0,200,64, wGUI_MODAL)

wAddStaticText

Syntax
wAddStaticText (text as wstring ptr, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, border as uinteger, wordwrap as uinteger, parent as guiElement) as guiElement

Description
Creates a static text object on the Graphical User Interface, this simply displays the specifed text in the specified box.

Text is a wide string that contains the text you want to display.

Top X, Top Y, Bottom X and Bottom Y define a box in which the text is drawn

Border is used to draw a visible box around the text, its value should be either of: -
wGUI_NO_BORDER
wGUI_BORDER

Word wrap is used to define whether text is to be wrapped around into a second line when it fills the width of the text box, its value should be either of:
wGUI_NO_WRAP 0
wGUI_WRAP 1

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example
statictextObject = wAddStaticText("Hello World", 4,0,200,16, wGUI_NO_BORDER, wGUI_NO_WRAP, windowObject)

wAddButton

Syntax
wAddButton (Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, text as wstring ptr, parent as guiElement) as guiElement

Description
Add a clickable button object to the gui display.

Top X, Top Y, Bottom X and Bottom Y define a box in which the button is drawn

id specifies a unique numerical reference for the button so events can be identified as coming from this object

text specified the label assigned to the button

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example
buttonObject = wAddButton(16,16,96,32, 120, "My Button", windowObject)

wAddScrollBar

Syntax
wAddScrollBar (horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, currentValue as integer, maxValue as integer, parent as guiElement) as guiElement

Description
Add a scrollbar object to the GUI display.

Horizontal defines if the scrollbar is horizontal or vertical, acceptable values for this field are:

wGUI_HORIZONTAL
wGUI_VERTICAL

Top X, Top Y, Bottom X and Bottom Y define a box in which the scrollbar is drawn

id specifies a unique numerical reference for the scrollbar so events can be identified as coming from this object

currentValue specified the current setting of the scrollbar

maxValue specifies the maximum setting of the scrollbar

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example
scrollbarObject = wAddScrollBar(wGUI_HORIZONTAL, 16,16,96,32, 120, 128, 156, windowObject)

wAddListBox

Syntax
wAddListBox (horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, background as integer, parent as guiElement) as guiElement

Description
Add a listbox object containing a list of items to the gui display.

horizontal specifies whether the scrollbar is oriented horizontally or vertically. acceptable values are:

wGUI_HORIZONTAL
wGUI_VERTICAL

Top X, Top Y, Bottom X and Bottom Y define a box in which the listbox is drawn

id specifies a unique numerical reference for the listbox so events can be identified as coming from this object

background specifies whether the background of the listbox should be drawn. acceptable values are: -

wGUI_DRAW_BACKGROUND
wGUI_EMPTY_BACKGROUND

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example
listboxObject = wAddListBox(16,16,96,32, 120, wGUI_DRAW_BACKGROUND, windowObject)

wAddListBoxItem

Syntax
wAddListBoxItem (listbox as guiElement, text as wstring)

Description
Add a text element to a list box.

listbox defines the listbox gui object to add the string too.

text is the string containing the new item

Example
wAddListBoxItem(listboxObject, "Apples")

wInsertListBoxItem

Syntax
wInsertListBoxItem (parent as guiElement, text as wstring, index as integer)

Description
Insert a text element to a list box.

listbox defines the listbox gui object to insert the string into.

text is the string containing the new item

index is the position at which to insert the item

Example
wInsertListBoxItem(listboxObject, "Pears", 3)

wRemoveListBoxItem

Syntax
wRemoveListBoxItem (parent as guiElement, index as integer)

Description
Remove a text element from a list box.

listbox defines the listbox gui object to remove the string from.

index is the position of the item to be removed

Example
wRemoveListBoxItem(listboxObject, 2)

wSelectListBoxItem

Syntax
wSelectListBoxItem (parent as guiElement, index as integer)

Description
Select a text element in a list box.

listbox defines the listbox gui object to select the item within.

index is the position of the item to be removed

Example
wSelectListBoxItem(listboxObject, 1)

wAddEditBox

Syntax
wAddEditBox (text as wstring, horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, border as integer, password as integer, parent as guiElement) as guiElement

Description
Add an editbox object containing a list of items to the GUI display.

text is the string that is inserted into the editbox

Top X, Top Y, Bottom X and Bottom Y define a box in which the editbox is drawn

id specifies a unique numerical reference for the editbox so events can be identified as coming from this object

border specifies whether the object has a border drawn around it. acceptable values are:

wGUI_NO_BORDER
wGUI_BORDER

password specifies whether the editbox is a password field that hides the text typed into it. acceptable values are:

wGUI_PASSWORD
wGUI_NOT_PASSWORD

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example

editboxObject = wAddEditBox("My String", 16,16,96,32, 120, wGUI_BORDER, wGUI_NOT_PASSWORD, windowObject)

wAddImage

Syntax
wAddImage (texture as wTexture, horizontal as integer, X as integer, Y as integer, useAlpha as integer, id as integer, parent as TEXTURE) as guiElement

Description
Add an image object to the GUI display.

texture is a loaded texture object that is to be displayed

X, Y define a position at which the image is drawn

useAlpha specifies whether the alpha channel of the texture is to be used. acceptable values are:

wIGNORE_ALPHA
wUSE_ALPHA

id specifies a unique numerical reference for the image so events can be identified as coming from this object

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example
imageObject = wAddImage(texture, 16,16, wIGNORE_ALPHA, 120, windowObject)

wAddCheckBox

Syntax
wAddCheckBox (text as wstring, horizontal as integer, Top X as integer, Top Y as integer, Bottom X as integer, Bottom Y as integer, id as integer, checked as integer, parent as guiElement) as guiElement

Description
Add a checkbox object to the GUI display.

text is the string that is used to label the checkbox

Top X, Top Y, Bottom X and Bottom Y define a box in which the checkbox is drawn

id specifies a unique numerical reference for the checkbox so events can be identified as coming from this object

checked specifies whether the object starts in the checked state. acceptable values are: -

wOFF
wON

Parent defines the parent object of this window. This can be ommited if the object has no parent.

Example
checkboxObject = wAddCheckBox("My Checkbox", 16,16,96,32, 120, wOFF, windowObject)

wCheckCheckBox

Syntax
wCheckCheckBox (checkbox as guiElement, checked as integer)

Description
Set the checked state of a checkbox.

checkbox defines the checkbox GUI object to check or uncheck.

checked specifies whether the object starts in the checked state. acceptable values are:

wOFF
wON

Example
wCheckCheckBox(checkboxObject, wON)

wAddFileOpen

Syntax
wAddFileOpen (title as wstring, id as integer, checked as integer, modal as integer, parent as guiElement) as guiElement

Description
Open a modal file open dialog so that a file can be selected.

title is the string that is displayed in the titlebar of the file selector window.

id specifies a unique numerical reference for the button so events can be identified as coming from this object

Modal determines if the window locks out the rest of the interface until it is closed: -
wGUI_MODAL
wGUI_NOT_MODAL

Parent defines the parent object of this window. This can be ommited if the window has no parent.

Example
fileOpenObject = wAddFileOpen("Select a bitmap", 120, wGUI_MODAL, windowObject)

wGetLastSelectedFile

Syntax
wGetLastSelectedFile (fileopenobject as guiElement, checked as integer)

Description
Get the last file name selected from a file selection dialog.

Example
fileName = wGetLastSelectedFile()

Sounds

A sound library linked in, but its functions not included in WorldSim3D 0.8.
It will be available in the next version of WorldSim3D.

ODE

ODE (Open Dynamics Engine) linked in, but its functions not included in WorldSim3D 0.8
It will be available in the next version of WorldSim3D.