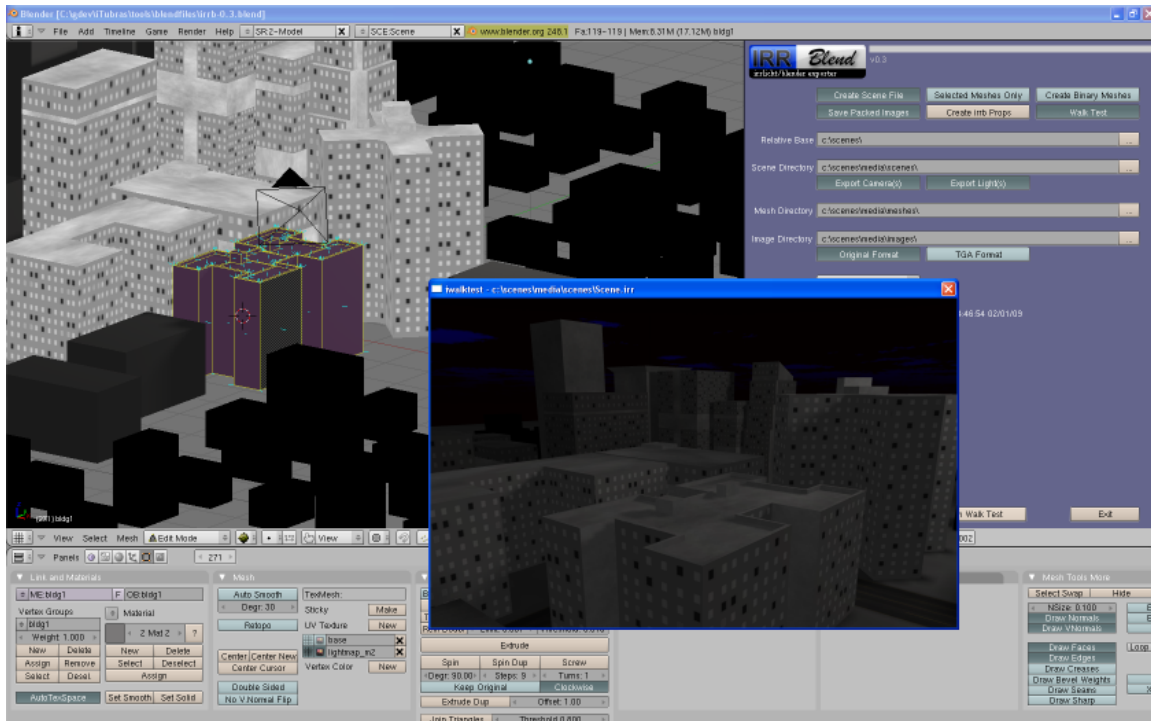




# irrbr

## User Guide

### version 0.3



[irrbr-0.3.blend Link](#)

Copyright 2008-2009 Tubras LTD  
All Rights Reserved

*This document and its contents were created using [OpenOffice.org](#) 3, [GIMP](#) 2.6, and [Inkscape](#) 0.46.*

# Table of Contents

Chapter 1 Features.....	<a href="#">1</a>
Chapter 2 Installation.....	<a href="#">2</a>
Chapter 3 Before You Start.....	<a href="#">3</a>
Chapter 4 Starting The Exporter.....	<a href="#">9</a>
Chapter 5 The irrb Interface.....	<a href="#">10</a>
Chapter 6 Setting Up Output Locations.....	<a href="#">13</a>
Chapter 7 Saving Packed Images.....	<a href="#">16</a>
Chapter 8 Irrlicht Material Generation.....	<a href="#">18</a>
Irrlicht Standard Material Types.....	<a href="#">19</a>
Chapter 9 Running The Exporter.....	<a href="#">22</a>
Chapter 10 iwalktest Integration And Usage.....	<a href="#">23</a>
Chapter 11 Binary Mesh Format.....	<a href="#">24</a>
Chapter 12 ID Properties.....	<a href="#">26</a>
Default Property Attributes.....	<a href="#">28</a>
Blender's ID Property Editor.....	<a href="#">28</a>
Generating ID Properties.....	<a href="#">30</a>
Material ID Properties.....	<a href="#">31</a>
Accessing User Attributes.....	<a href="#">32</a>
Chapter 13 Cameras And Lights.....	<a href="#">33</a>
Chapter 14 Creating A Billboard.....	<a href="#">36</a>
Chapter 15 Creating A Skybox.....	<a href="#">37</a>
Chapter 16 irrb Utilities.....	<a href="#">38</a>
Installation.....	<a href="#">38</a>
Configuration.....	<a href="#">38</a>
iwalktest Usage.....	<a href="#">42</a>
imeshcvr Usage.....	<a href="#">44</a>
Chapter 17 Notes.....	<a href="#">45</a>
Chapter 18 FAQ.....	<a href="#">46</a>

## Chapter 1 Features

The Irrlicht/Blender Exporter exports Blender scene and static node data to the native Irrlicht scene (.irr) and mesh (.irmesh) file formats.

### Implemented Features

- Blender Scene data is exported to Irrlicht native scene format (.irr).
- Blender Mesh data is exported to the Irrlicht native mesh format (.irmesh).
- Export Blender ID Properties as Irrlicht scene UserData.
- Export Blender Lamp, Mesh, Camera, and Empty objects.
- Export Game Engine Materials and corresponding UV data.
- Export Blender Generated Lightmap/Baked Materials.
- Export Object Parent/Child relationships and transforms.
- Export Irrlicht Skyboxes and Billboards.
- Specify Textures as two-sided, transparent, and/or using lighting per face.
- Blender Packed UV Images may optionally be saved to a user specified location and format.
- Automatic translation between Blender and Irrlicht coordinate systems.
- Optionally "walk test" your scene immediately after it has been exported using the Irrlicht render engine.

### Working on for 0.4

- animation

### Todo

- Tangent & Binormal calculation for EVT\_TANGENTS generation
- Sound nodes
- Collision
- Particles
- Triggers

## Chapter 2 Installation

The latest stable download of **irrb** is accessible from both the "Project Home" and "Downloads" tabs of the [Tubras Google Code project](http://tubras.googlecode.com/). On the "Project Home" tab, **irrb** is listed in the "Featured Downloads" section.

The current version is 0.3. It may be downloaded from here:

<http://tubras.googlecode.com/files/irrb-0.3.zip>

The contents of the zip package are:

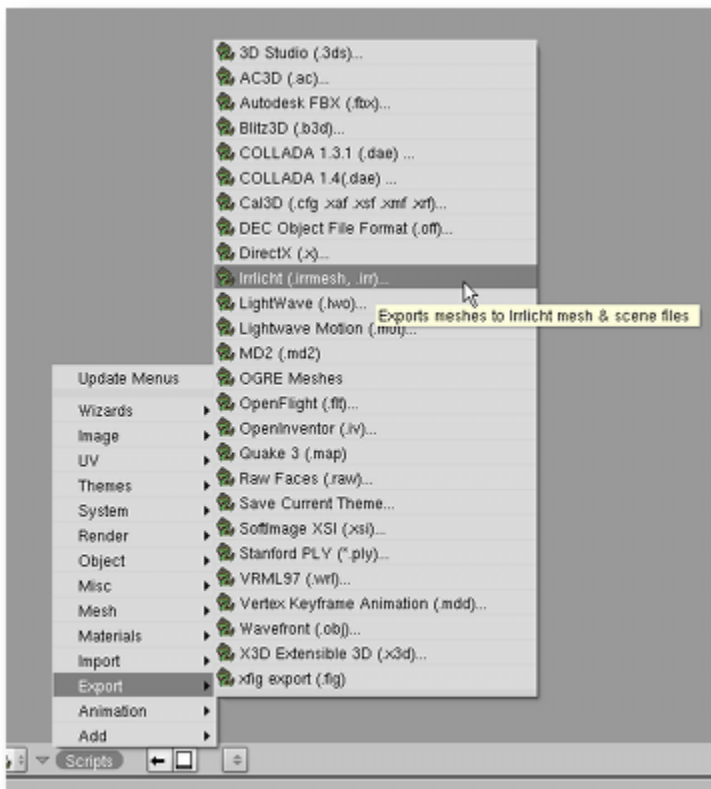
```
irrb.py
irrbmodules/iActionMgr.py
irrbmodules/iConfig.py
irrbmodules/iExporter.py
irrbmodules/iFilename.py
irrbmodules/iGUI.py
irrbmodules/iMaterials.py
irrbmodules/iMesh.py
irrbmodules/iMeshBuffer.py
irrbmodules/irrbblend.png
irrbmodules/iScene.py
irrbmodules/iTGAWriter.py
irrbmodules/iUtils.py
irrbmodules/__init__.py
irrbmodules/docs/changes.txt
irrbmodules/docs/license.html
irrbmodules/docs/UserGuide.pdf
```

Unzip the contents of the zip file into either the Blender scripts directory (".blender/scripts") or the user defined scripts directory. It is important that the "irrbmodules" directory is created as a sub-directory underneath the Blender/User Defined scripts directory.

**Linux users should** install **irrb** to your user defined scripts directory. If you don't have one, create one and set the location in Blenders Preferences (Python:). This directory should also contain the sub-directories: "bpydata/config" in order for **irrb** to successfully save configuration options between Blender sessions. Failing to do so, will likely result in **irrb's** configuration data not being saved due to permission related problems.

After **irrb** has been installed, restart Blender to have **irrb** appear as an option under the Blender Scripts|Export menu

Documentation on installing and configuring "iwalktest" for both Windows and Linux may be found here: [irrb Utilities](#)



## Chapter 3 Before You Start

### Key Points

- Blender Objects map to Irrlicht scene nodes in scene files (.irr).
- Blender Mesh Data Blocks map to mesh files (.irmesh).
- Multiple Blender Objects may link to the same Mesh Data Block. In this case only a single .irmesh file will be generated and multiple nodes will be written to the scene file.
- **irrB** only exports materials that have been UV mapped to images set up in the UV/Image Editor. These are sometimes referred to as "Game Engine Materials".

### Details

Before you start using **irrB** you should at the very least have a fundamental understanding of how to use Blender.

In order to better understand how **irrB** works with Blender, it will help you to look at the relationship between Blenders "Objects" and "DataBlocks".

You can find a technical explanation here: [Blender Architecture](#)

In simpler terms, Blender "Objects" (sometimes referred to as Nodes) specify the Position, Rotation, and Scale of the "DataBlock" that an "Object" links to. You may also have multiple "Objects" link to the same "DataBlock". Let's have a closer look.

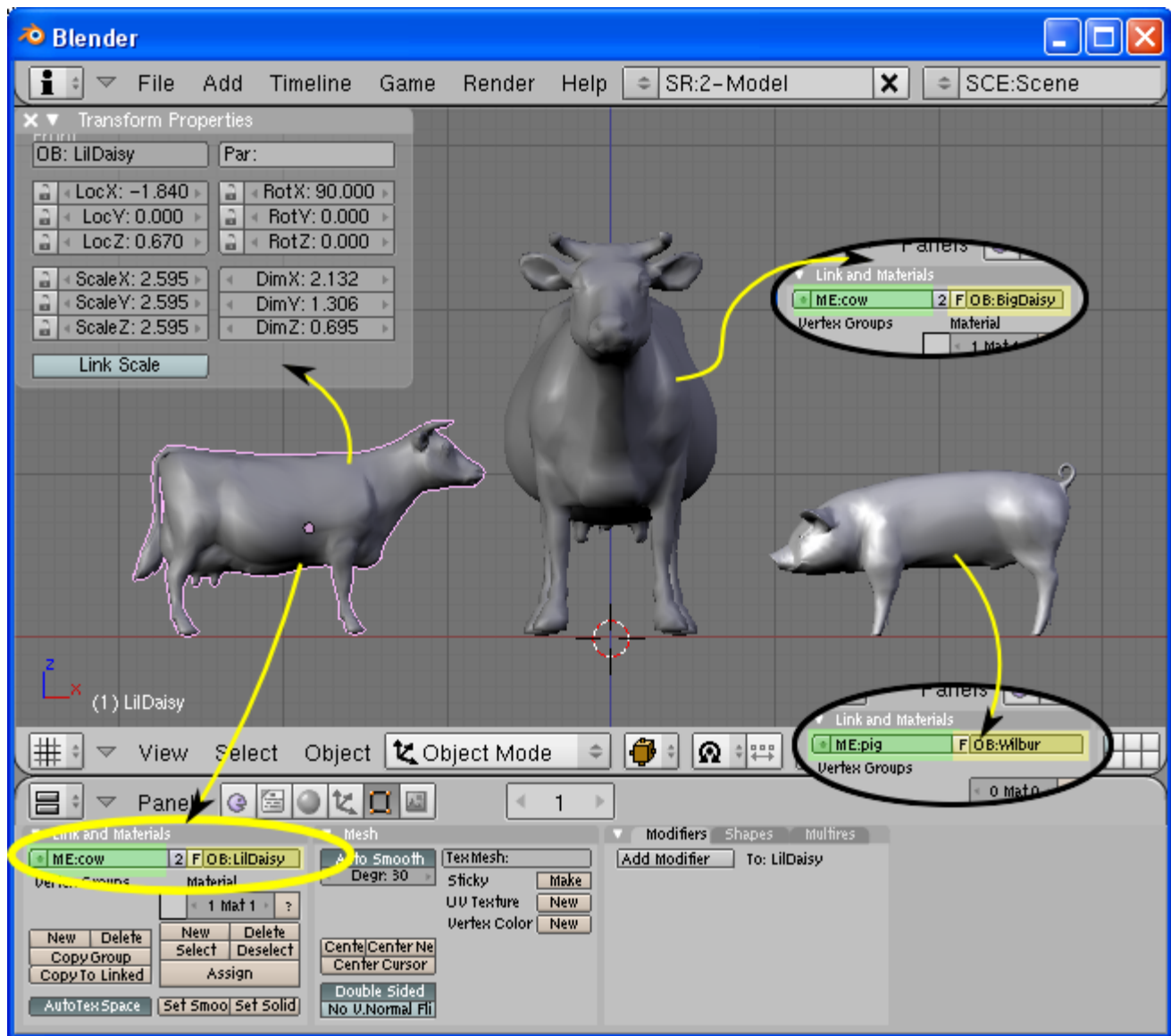


*Illustration 1: LilDaisy, BigDaisy, and Wilbur.*

If the entire scene pictured above were to be exported, how many mesh files would be created and how many Irrlicht scene "nodes" would be written to the "Scene.irr" file?

1. If you answered - 3 mesh files and 3 nodes. Keep reading.
2. If you answered - 2 mesh files and 3 nodes. Keep reading.
3. If you answered - it's not possible to tell from the image. You may skip to the next section.

The reason for not knowing what will be exported is that you must individually select both "cow" objects/nodes to find out if they are linked to the same or different mesh DataBlocks.



*Illustration 2: object and mesh datablock links.*

Selecting the cow on the far left (right click in object mode) and pressing F9 (edit mode window), we can see that this cow object is named "LilDaisy" (OB:LilDaisy) and it is linked to a mesh DataBlock named "cow" (ME:cow).

Selecting the cow in the middle would display an object named "BigDaisy" along with a link to a mesh DataBlock named "cow".

Selecting the pig on the far right would show the pigs object name is "Wilbur" and that it is linked to a mesh DataBlock named "pig".

Now that we know which Objects are linked to which DataBlocks, we can answer the question posed above - "How many mesh files will be created and how many nodes will be written to the Scene.irr file?". The answer is

- 3 nodes will be written to the Scene.irr file: "LilDaisy", "BigDaisy", and "Wilbur".
- 2 .irrmesh files will be created: "cow.irrmesh" and "pig.irrmesh".

The reason only a single "cow.irrmesh" is created is that both cow objects are linked to the same mesh DataBlock.

You may be asking yourself how can both cow objects use the same mesh file? The cow on the left (LilDaisy) is smaller and facing a different direction than BigDaisy. To answer that, let's have a look at the partial contents of the generated Scene.irr file (lamp and camera nodes excluded):

```
<?xml version="1.0"?>
<!-- Created 12/26/2008 10:45 by irr v0.3 - "Irrlicht/Blender Exporter" -->
<irr_scene>
  <node type="mesh">
    <attributes>
      <string name="Name" value="LilDaisy" />
      <int name="Id" value="-1" />
      <vector3d name="Position" value="-0.880993, -0.278409, 0.000000" />
      <vector3d name="Rotation" value="0.000000, -80.097466, 0.000000" />
      <vector3d name="Scale" value="0.519058, 0.519058, 0.519058" />
      <bool name="Visible" value="true" />
      <bool name="AutomaticCulling" value="true" />
      <bool name="DebugDataVisible" value="false" />
      <bool name="IsDebugObject" value="false" />
      <string name="Mesh" value="meshes\cow.irrmesh" />
      <bool name="ReadOnlyMaterials" value="false" />
    </attributes>
  </node>
  <node type="mesh">
    <attributes>
      <string name="Name" value="BigDaisy" />
      <int name="Id" value="-1" />
      <vector3d name="Position" value="0.000000, 0.000000, 0.000000" />
      <vector3d name="Rotation" value="0.000000, 0.000000, 0.000000" />
      <vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
      <bool name="Visible" value="true" />
      <bool name="AutomaticCulling" value="true" />
      <bool name="DebugDataVisible" value="false" />
      <bool name="IsDebugObject" value="false" />
      <string name="Mesh" value="meshes\cow.irrmesh" />
      <bool name="ReadOnlyMaterials" value="false" />
    </attributes>
  </node>
  <node type="mesh">
    <attributes>
      <string name="Name" value="Wilbur" />
      <int name="Id" value="-1" />
      <vector3d name="Position" value="0.840124, -0.324030, -0.066227" />
      <vector3d name="Rotation" value="-90.000000, 180.000000, -0.000000" />
      <vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
      <bool name="Visible" value="true" />
      <bool name="AutomaticCulling" value="true" />
      <bool name="DebugDataVisible" value="false" />
      <bool name="IsDebugObject" value="false" />
      <string name="Mesh" value="meshes\pig.irrmesh" />
      <bool name="ReadOnlyMaterials" value="false" />
    </attributes>
  </node>
</irr_scene>
```

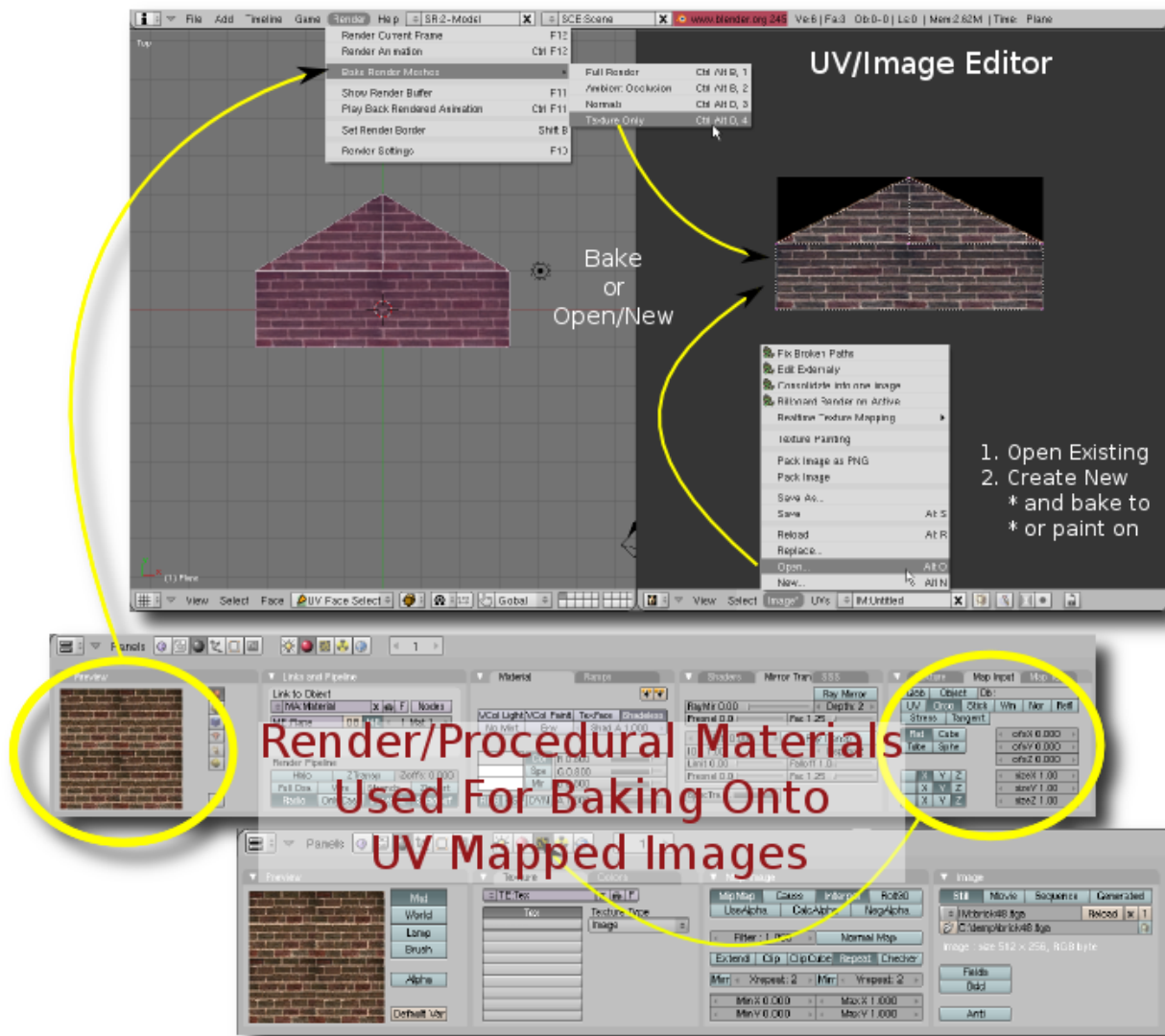
We can see that both the LilDaisy and BigDaisy nodes reference the same mesh by looking at each nodes "Mesh" attribute:

```
<string name="Mesh" value="meshes\cow.irrmesh" />
```

So how can LilDaisy be a different size, position, and orientation? Well, have a look at the "Position",



"Rotation", and "Scale" attributes (transform data) for each node in the Scene.irr file. They are different. Looking back at Illustration 2, we can see that Blender's Object "Transform Properties" are exported to the Scene.irr file. Compare LilDaisy's properties in Image 2 to those that were created in the Scene.irr file. They are the same, except that Blender's Y and Z axis information have been rearranged for your convenience.



*Illustration 3: The lil' big picture.*

**irrb** honors Blender's Object DataBlock relationships. This may cause unexpected results especially when exporting Meshes only. More on this later.

As stated in the key points, **irrb** only exports images that have been assigned in the UV/Image editor. From Illustration 3, you can see that images may be "assigned" from two different sources:

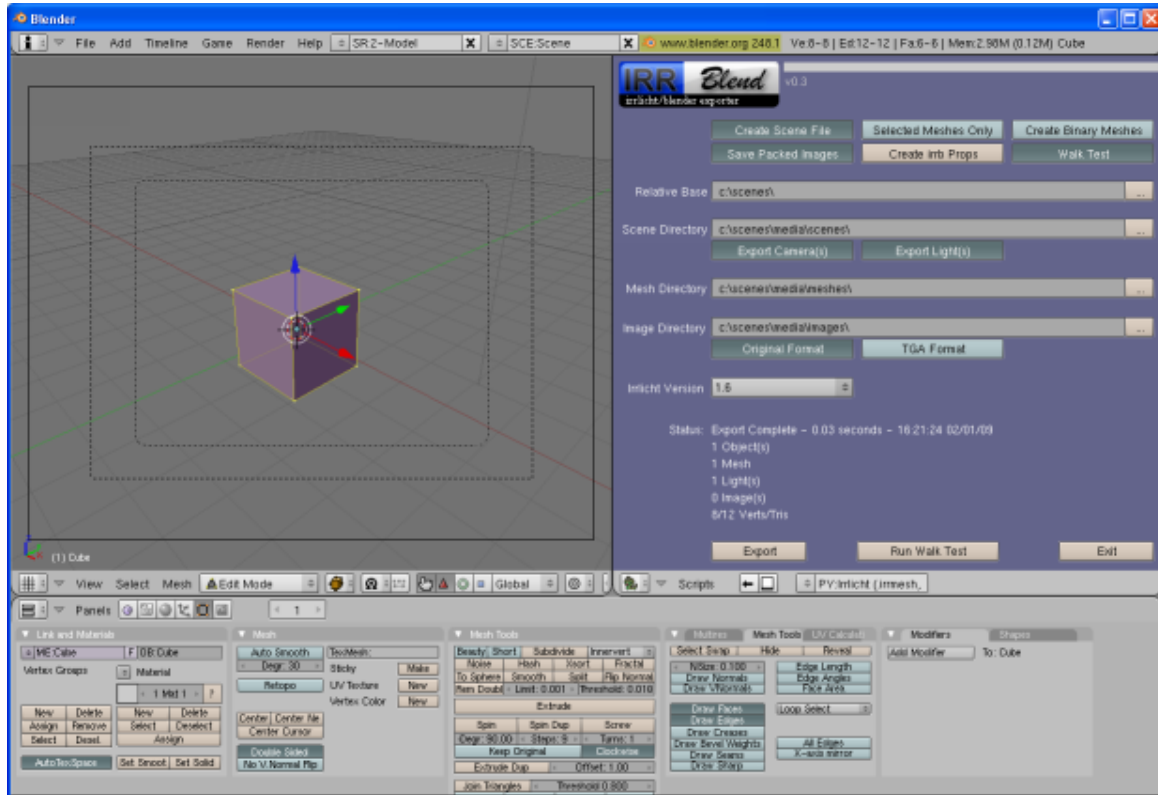
- Open an existing image.
- Create a new blank Image.

Once an image has been assigned to a particular UV layer, you may optionally use Blender's powerful render/procedural material system to create new textures via baking.

It is important to understand that render/procedural materials are NOT exported. They are simply used for creating textures, lightmaps, and/or normal maps.

## Chapter 4 Starting The Exporter

The quickest way to start **irrblender** is to select it from Blender's File menu: "File | Exporters | Irrlicht". Using this method will typically cause the **irrblender** interface to overlay the active window possibly hiding your 3D view. If you repeatedly export and examine your scene using **iwalktest**, you may find it convenient to run the exporter in "split" screen mode:



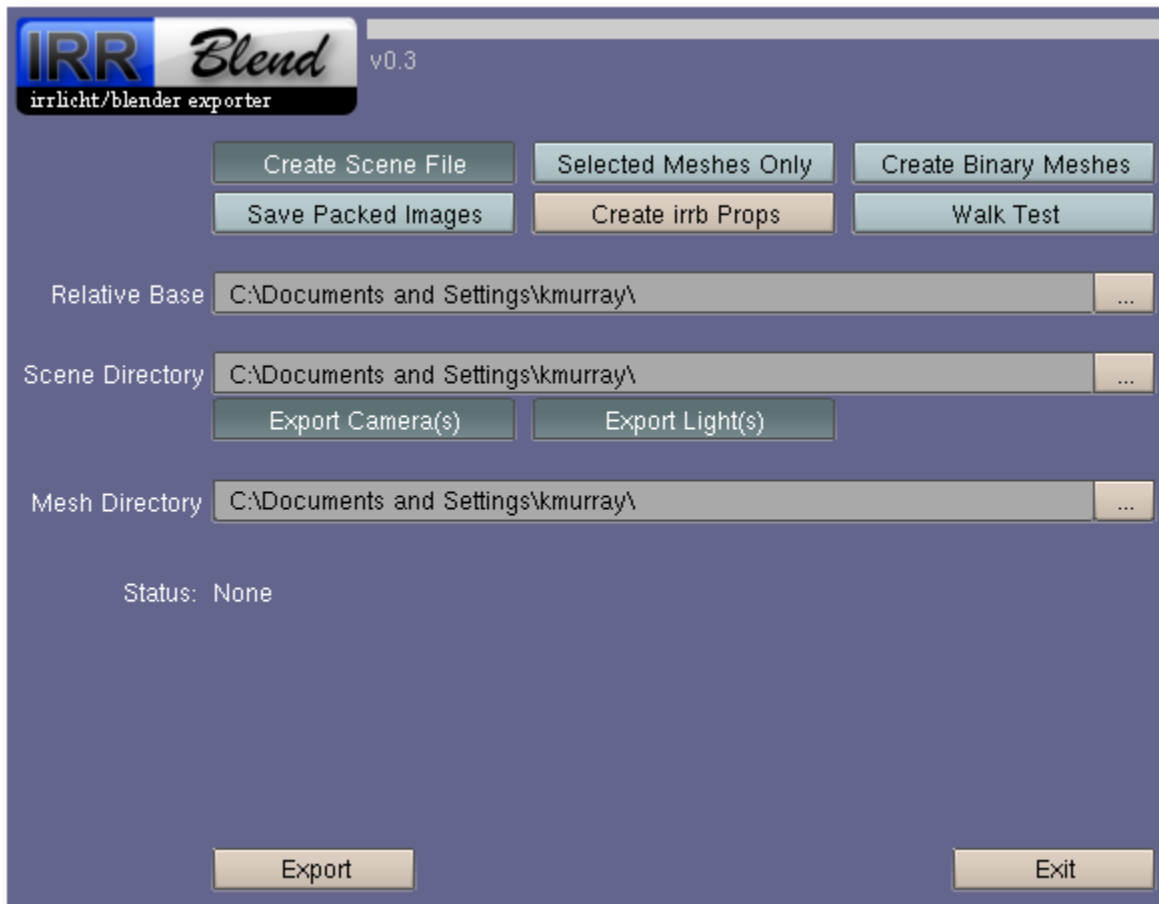
*Illustration 4: Split Screen View*

Steps to create a split screen with **irrblender** in the right half of the Blender window:

1. Right click the top border of the 3D view window.
2. Select the "Split Area" menu option.
3. Move the divider to the middle of the 3D view and left click to anchor it.
4. Select "Scripts Window" for the window type (lower left corner of the right half of the screen).
5. In the "Scripts" menu, select "Export" and then "Irrlicht".

## Chapter 5 The irrbl Interface

First time **irrbl** users will see Illustration 5 with the default options:

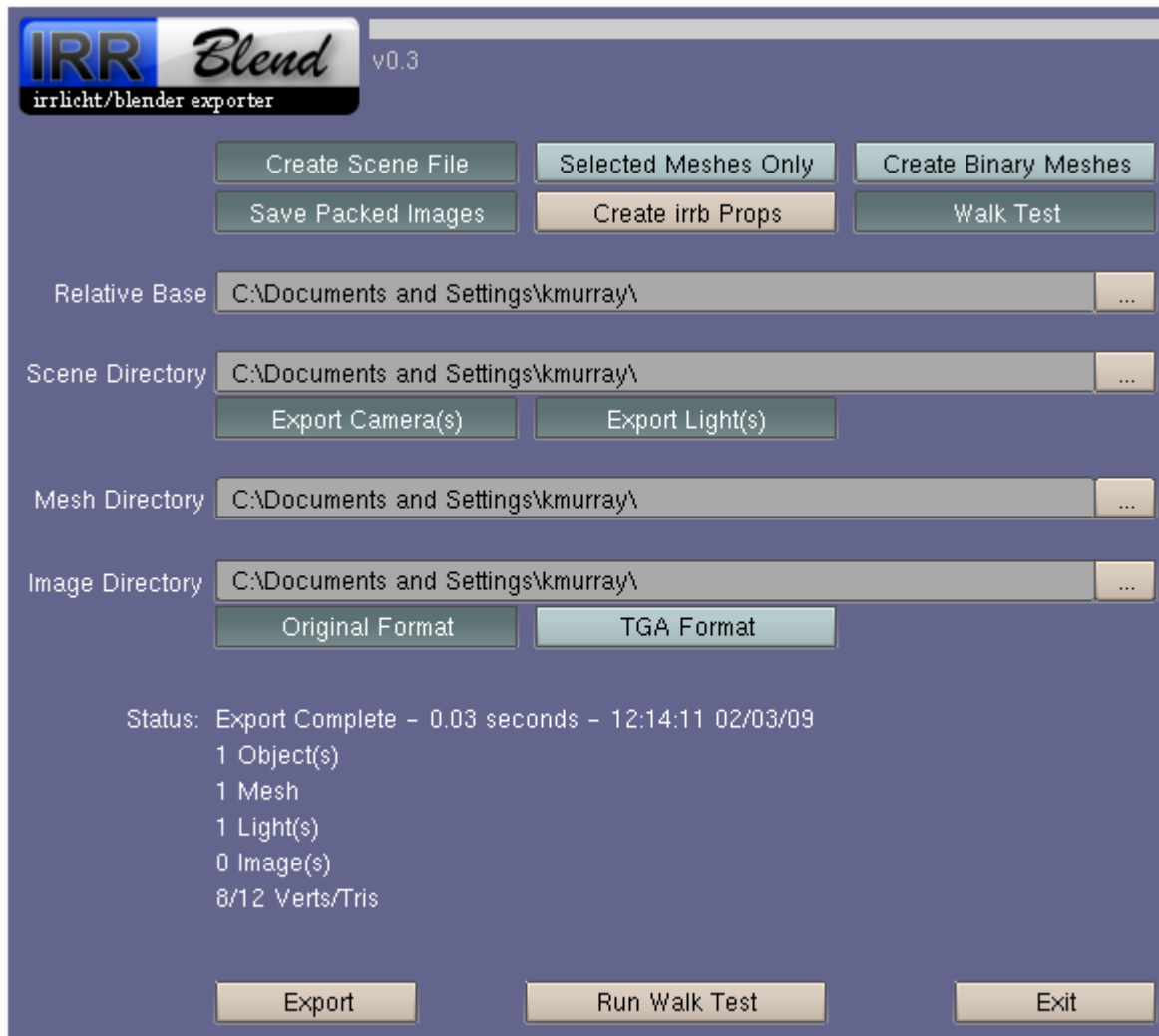


*Illustration 5: irrbl Default Options.*

For both Windows and Linux, the directories will default to the user “home” directory.

**irrbl** stores your preferences and directory locations in a configuration file so your favorite options are remembered between Blender sessions.

Let's take a quick look at all of the **irrbl** options:



*Illustration 6: irrB Options.  
(note that light/dark green buttons are "toggle" buttons)*

**Create Scene File** - Toggle button used to control whether or not a scene file (".irr") is generated. If selected, scene related options will appear.

**Selected Meshes Only** - This toggle button will only export meshes that are selected in the current scene. When un-selected, all nodes and linked meshes will be exported.

**Create Binary Meshes** - This toggle button will only appear if you have the [imeshcv](#) utility installed and properly configured. When selected, the meshes will be exported using the experimental ".irrbmesh" (binary .irrmesh) format. [imeshcv](#) is the program used to convert from .irrmesh to .irrbmesh. The original .irrmesh files are left intact. For more information on the .irrbmesh format, see the section named "[Binary Mesh Format](#)".

**Save Packed Images** – Blender allows UV Texture Images to be packed (saved) inside a .blend file. This option tells irrB to save packed images to a file so the images will be accessible by your Irrlicht application.

**Create irrB Properties** – This push button creates "irrB" default ID Properties for selected objects and Blender materials. When clicked, the "Status" area will display information on the number of objects that were updated. For more information on creating "irrB" ID Properties, see the section named: [ID Properties](#).

**Walk Test** – This toggle button that will appear if you have the IWALKTEST OS environment variable defined.

When selected, the program defined by the IWALKTEST environment variable is executed with your exported scene file as parameter immediately after a successful export. The "[irrB Utilities](#)" contains a program that you may use to examine an exported scene.

**Relative Base** – This directory is used as the base directory when determining relative locations of mesh and texture/image files. In other words, **irrB** uses this location to calculate relative file paths that are stored in the scene and mesh files. More information on how to use the "Relative Base" field may be found in the section named "[Setting Up Output Locations](#)"

**Scene Directory** - This is the directory location that the ".irr" scene file will be saved to. The name of the scene file is generated using the current Blender scene name that is being exported. For example, if you named your scene "barnyard", the scene file that is created will be named "barnyard.irr".

**Export Camera(s)** - Toggle button to indicate whether or not to write camera objects to the ".irr" scene file.

**Export Light(s)** - Toggle button to indicate whether or not to write light objects to the ".irr" scene file.

**Mesh Directory** - This is the directory location where all exported ".irrmesh" files will be saved to.

**Image Directory** - This is the directory location where packed images will be saved to if the "Save Packed Images" option is selected.

**Original Format** – When saving packed images, save using the original format of the packed image.

**TGA Format** – When saving packed images, you may optionally have the packed images converted to .TGA before they are saved..

**Irrlicht Version** – This experimental option allows you to select the target Irrlicht version for binary meshes. It will only appear if the "Create Binary Meshes" option is selected.

**Export** - Makes the magic happen.

**Run Walk Test** - Button that re-runs the **iwalktest** application with the last scene exported.

**Exit** - Exits the **irrB** export script.

## Chapter 6 Setting Up Output Locations

### Key Points

- References to mesh and image locations are **ALL** relative.
- The relative location is based on a single, “base” location specified in the exporters “Relative Base” field.

### Details

Before you export a Blender scene, you will need to set up the locations of where your output (scene, mesh, and optional packed image files) will be written to.

An Irrlicht scene (.irr) file contains references to mesh and image file locations (Skybox & Billboards). For example, a mesh scene node contains the following information:

```
<node type="mesh">
  <attributes>
    <string name="Name" value="Cube" />
    <int name="Id" value="-1" />
    <vector3d name="Position" value="0.000000, 0.000000, 0.000000" />
    <vector3d name="Rotation" value="0.000000, -0.000000, 0.000000" />
    <vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
    <bool name="Visible" value="true" />
    <enum name="AutomaticCulling" value="frustum_box" />
    <bool name="DebugDataVisible" value="false" />
    <bool name="IsDebugObject" value="false" />
    <bool name="ReadOnlyMaterials" value="false" />
    <string name="Mesh" value="Cube.irrmesh" />
  </attributes>
  <userData>
    <attributes>
    </attributes>
  </userData>
</node>
```

Notice how the “**Mesh**” attribute is used to specify the file name that contains the actual mesh data (vertex, face, & material info). The question that needs to be addressed is “**How does the Irrlicht scene loader know where to find Cube.irrmesh?**”.

The answer is fairly simple:

1. First, it looks in the loading application's current directory.
2. Second, it looks in any folder/file that has been previously defined/added by your application via the [\*\*IFileSystem\*\*](#) interface.

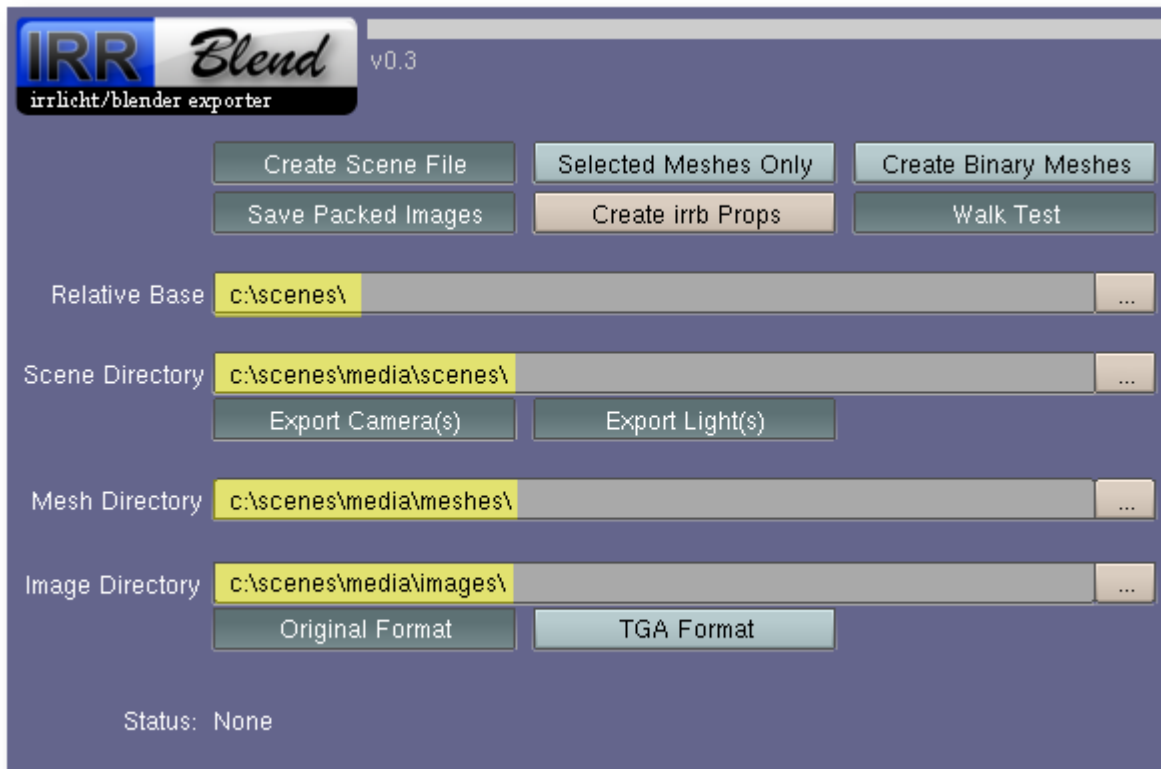
So, unless you plan to have all of your assets (scene, meshes, images, sounds, etc.) located in the same directory as your application, you will need to do a little preparation. The best way to explain this is to work through an example.

### Example Layout 1

For our first example we'll make the following assumptions:

- Our assets are going to be stored in a sub-directory underneath our application directory. We'll name this directory “**media**”.
- Underneath the “media” directory we'll store scene, mesh, and image data separately - each in their own

respective sub-directories (“scenes”, “meshes”, and “images”):



*Illustration 7: Output Directory Setup*

If you were to export Blender's default scene, which contains a single, untextured cube mesh, the following files would be generated:

```
c:\scenes\media\scenes\Scene.irr      <-- Irrlicht scene file
c:\scenes\media\meshes\Cube.irrmesh  <-- Cube mesh file
```

Opening the scene file (Scene.irr) in a text editor, would show the following:

```
<attributes>
  <string name="Name" value="Cube" />
  <int name="Id" value="-1" />
  <vector3d name="Position" value="0.000000, 0.000000, 0.000000" />
  <vector3d name="Rotation" value="0.000000, -0.000000, 0.000000" />
  <vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
  <bool name="Visible" value="true" />
  <enum name="AutomaticCulling" value="frustum_box" />
  <bool name="DebugDataVisible" value="false" />
  <bool name="IsDebugObject" value="false" />
  <bool name="ReadOnlyMaterials" value="false" />
  <string name="Mesh" value="media\meshes\Cube.irrmesh" />
</attributes>
```

Notice how the **Mesh** attribute is set relative to the value specified in **irrblt's** “Relative Base” field (“c:\scenes\”).

If your application was located in and executed from the “c:\scenes\” directory:

```
c:\scenes\myapp.exe
```

you wouldn't need to do anything extra before loading the scene file “media\scenes\Scene.irr”, because when



the scene loader attempts to load the Cube.irmesh file, it will do so using the Mesh attribute value of “media\meshes\Cube.irmesh”, which exists right off of our application directory of “c:\scenes\”.

What if our application was being executed from a directory other than “c:\scenes\” (as is likely the case for iwalktest.exe utility)? Well, before our application loads the scene file, it would need to add the “Relative Base” directory to Irrlicht's file system with a call to:

```
IrrlichtDevice* device; // previously created/assigned
device->getFileSystem()->addFolderFileArchive("c:\\scenes");
// now we can load the scene file
SceneManager->loadScene("c:\\scenes\\media\\scenes\\Scene.irr");
```

What would the mesh attribute value be if the **irrB's** “Relative Base” field were set to:

c:\scenes\media\scenes\

The correct answer is:

```
<string name="Mesh" value="..\meshes\Cube.irmesh" />
```

[\*\*iwalktest\*\*](#) can be very useful for determining why a mesh isn't appearing in a scene and/or why a texture isn't appearing properly. The first question to ask yourself: "Is the mesh/texture being loaded?". **iwalktest** will log mesh/texture loading errors to its log file "iwalktest.log" which is located in the same directory as the **iwalktest** executable. Windows users can also check for load errors in the **iwalktest** console window if it is configured to be displayed (default).

## Chapter 7 Saving Packed Images

### Key Points

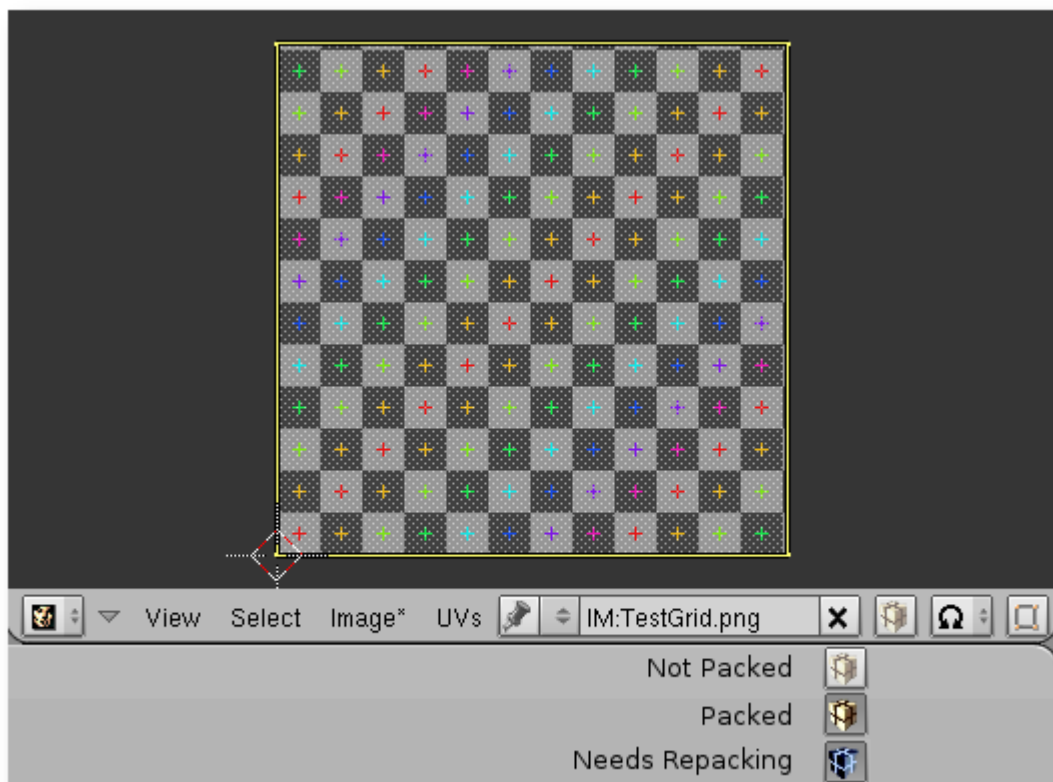
- Blender allows you to create UV mapped images within Blender itself. The newly created images can be saved to disk or optionally “packed” and saved within the .blend file.
- Packed images are a convenient method for exchanging .blend files with other artists without having to worry about keeping external images/textures in sync.
- **irrb** allows you to optionally save packed images to disk when exporting.

### Details

Blender allows you to “pack” texture images. This means the images are physically stored inside of the .blend file. Images that are stored in the .blend file will **NOT** be accessible to the application that is loading an exported scene or mesh file.

Why pack images then? As mentioned above - convenience. Especially if you exchange .blend files with someone else working on a scene/mesh.

Regardless of the image source (externally opened or created within Blender), you can **pack** and **unpack** an image by clicking on the **packing icon**:



*Illustration 8: Blender Image Pack Icons*

- **Top** icon indicates the current image is **NOT** packed.
- **Middle** icon indicates the current image is **packed**.
- **Bottom** icon indicates the current image **needs to be re-packed**.

Images will only need to be re-packed if the original image was previously **packed** and then the image is later modified via baking or texture painting.

Images that are created by Blender and neither packed or saved, are stored internally in the .blend file as 24 bit TGA images. It is important to note that if you modify (texture paint or bake) an image created within Blender, the modifications are not saved internally in the .blend file. In order to save modifications, you must either pack the image or save it to an external file.

If you choose to NOT save packed images then you are responsible for making sure Blender generated images are saved to disk. If you forget, **irrdb** will display a warning message during the export.

If you choose to make use of Blender's packed images feature, you will need to select the “Save Packed Images” option button and also choose a directory where the saved images will be stored.

## Chapter 8 Irrlicht Material Generation

### Key Points

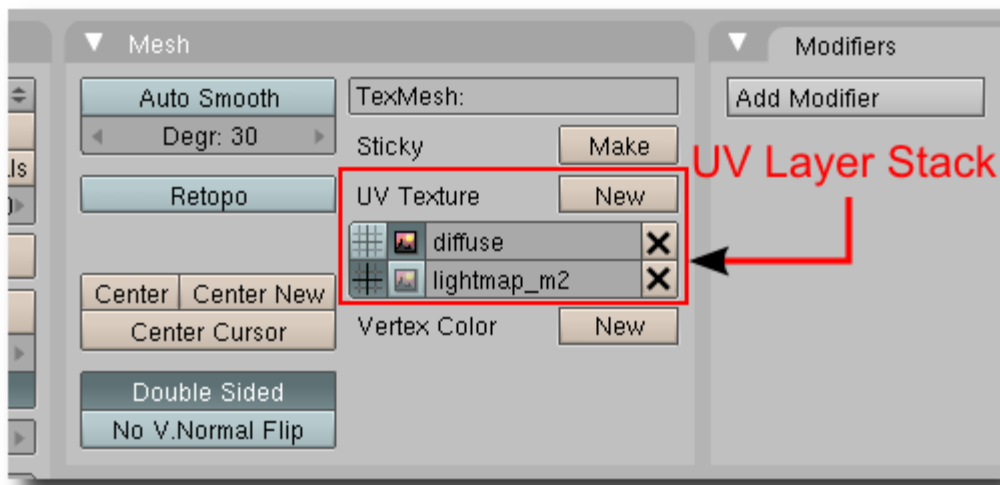
- **irrb** generates Irrlicht materials based on a Blender UV Layer name.
- UV Images map directly to the Irrlicht material "Texture#" attribute. Top UV image is written to "Texture1", Second UV image is written to "Texture2"...

### Details

**irrb** generates Irrlicht materials based on Blender's UV layer stack. All of Irrlicht's standard Materials require either one or two images to be assigned to a material's "texture" attributes located in the .irmesh file:

```
<material bmat="uvmat:untitled:100">
  <enum name="Type" value="lightmap_m2"/>
  ...
  <texture name="Texture1" value="cowuvmap.tga"/>
  <texture name="Texture2" value="lightmap.tga"/>
  <texture name="Texture3" value=""/>
  <texture name="Texture4" value=""/>
  ...
</material>
```

**irrb** maps the images assigned in Blender's UV layer stack to the Irrlicht Texture# attribute one to one:



The image assigned to the top layer is written to the Texture1 attribute. The image assigned to the second layer is written to the Texture2 attribute.

From the example above you can see a material section in the .irmesh file format allows up to four texture layers to be defined even though the standard Irrlicht materials only require up to two.

So how does **irrb** determine the type of material to generate? (enum name="Type" value="**lightmap\_m2**")

Easy, you assign one of Blender's UV layer names an Irrlicht material name that matches one of Irrlicht's built-in standard material names. The built-in standard Irrlicht material names may be found in the ["ematerialtypes.h"](#) file. The material names are also defined below for your convenience.

Starting with Blender's top layer in the UV layer stack and working its way down, **irrb** looks for a match against Irrlicht's material names. If it finds a match, the matching material name will be used as the material type value.

If no match is found, **irrb** defaults the material type to **"solid"**.

**irrb** is also capable of generating material types that match your custom material renderer which you register with either **addMaterialRenderer()** or **addAndDropMaterialRenderer()**. To do this, simply prefix your custom material name with a "\$" in the Blender UV Layer Name. For example, if you have a custom material renderer named "MyMaterial", you would enter "\$MyMaterial" in one of Blender's UV layer names. **irrb** will remove the \$ and generate:

```
<material bmat="uvmat:untitled:100">
  <enum name="Type" value="MyMaterial"/>
  ...
</material>
```

## Irrlicht Standard Material Types

**solid** - Standard solid material. Only first texture is used as the diffuse texture.

**solid\_2layer** - 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, but it works with DirectX.

**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. Vertex light is ignored.

**lightmap\_add** - Material type with standard lightmap technique but lightmap and diffuse texture are not modulated, but added instead.

**lightmap\_m2** - 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. Vertex light is ignored. The texture colors are effectively multiplied by 2 for brightening.

**lightmap\_m4** - 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. Vertex light is ignored. The texture colors are effectively multiplied by 4 for brightening.

**lightmap\_light** - Like **lightmap**, but also supports dynamic lighting.

**lightmap\_light\_m2** - Like **lightmap\_m2**, but also supports dynamic lighting.

**lightmap\_light\_m4** - Like **lightmap\_m4**, but also supports dynamic lighting.

**detail\_map** - Detail mapped material. The first texture is diffuse color map, the second is added to this and usually displayed with a bigger scale value so that it adds more detail. The detail map is added to the diffuse map using ADD\_SIGNED, so that it is possible to add and subtract color from the diffuse map. For example a value of (127,127,127) will not change the appearance of the diffuse map at all. Often used for terrain rendering.

**sphere\_map** - Environment reflection Material. The "sphere reflection map" must be Texture 1.

**reflection\_2layer** - A reflecting material with an optional additional non reflecting texture layer. The reflection map should be set as Texture 1.

**trans\_add** - A transparent material. 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.

**trans\_alphach** - 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 (`video::IVideoDriver::setTextureCreationFlag()`). Also, an alpha ref is used, which can be manipulated using `SMaterial::MaterialTypeParam`. If set to 0, the alpha ref gets its default value which is 0.5f and which means that pixels with an alpha value >127 will be written, others not. In other, simple words: this value controls how sharp the edges become when going from a transparent to a solid spot on the texture. Note that **irrbb** sets Param1 to 0.000001.

**trans\_alphach\_ref** - Makes the material transparent based on the 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 **trans\_alphach**. It is ideal for drawing stuff like leaves of plants, 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 (`video::IVideoDriver::setTextureCreationFlag()`).

**trans\_vertex\_alpha** - Makes the material transparent based on the vertex alpha value.

**trans\_reflection\_2layer** - A transparent reflecting material with an optional additional non reflecting texture layer. The reflection map should be set as Texture 1. The transparency depends on the alpha value in the vertex colors. A texture which will not reflect can be set als Texture 2. Please note that this material type is currently not 100% implemented in OpenGL. It works in Direct3D.

**normalmap\_solid** - A solid normal map renderer. First texture is the color map, the second should be the normal map. Note that you should use this material only when drawing geometry consisting of vertices of type `S3DVertexTangents` (`EVT_TANGENTS`). You can convert any mesh into this format using `IMeshManipulator::createMeshWithTangents()` (See SpecialFX2 Tutorial). This shader runs on vertex shader 1.1 and pixel shader 1.1 capable hardware and falls back on 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. Currently, this shader is only implemented for the D3D8 and D3D9 renderers.

**normalmap\_trans\_add** - A transparent normal map renderer. First texture is the color map, the second should be the normal map. Note that you should use this material only when drawing geometry consisting of vertices of type `S3DVertexTangents` (`EVT_TANGENTS`). You can convert any mesh into this format using `IMeshManipulator::createMeshWithTangents()` (See SpecialFX2 Tutorial). This shader runs on vertex shader 1.1 and pixel shader 1.1 capable hardware and falls back on 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. Currently, this shader is only implemented for the D3D8 and D3D9 renderers.

**normalmap\_trans\_vertexalpha** - A transparent (based on the vertex alpha value) normal map renderer. First texture is the color map, the second should be the normal map. Note that you should use this material only when drawing geometry consisting of vertices of type `S3DVertexTangents` (`EVT_TANGENTS`). You can convert any mesh into this format using `IMeshManipulator::createMeshWithTangents()` (See SpecialFX2 Tutorial). This shader runs on vertex shader 1.1 and pixel shader 1.1 capable hardware and falls back on 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. Currently, this shader is only implemented for the D3D8 and D3D9 renderers.

**parallaxmap\_solid** - Just like **normalmap\_solid**, but uses parallax mapping too, which looks a lot more realistic. This only works when the hardware supports at least vertex shader 1.1 and pixel shader 1.4. 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. The `IVideoDriver::makeNormalMapTexture()` method writes this value automaticly when creating normal maps from a heightmap when using a 32 bit texture. The height scale of the material (affecting the bumpiness) is being controlled by the `SMaterial::MaterialTypeParam` member. If set to zero, the default value (0.02f) will be applied. Otherwise the value set in `SMaterial::MaterialTypeParam` is taken. This value depends on with which scale the texture is mapped on the material. Too high or low values of `MaterialTypeParam` can result in strange artifacts.

**parallaxmap\_trans\_add** - A material just like **parallaxmap\_solid**, but it is transparent, using EMT\_TRANSPARENT\_ADD\_COLOR as base material.

**parallaxmap\_trans\_vertexalpha** - A material just like EMT\_PARALLAX\_MAP\_SOLID, but it is transparent, using EMT\_TRANSPARENT\_VERTEX\_ALPHA as base material.

**onetexture\_blend** - BlendFunc = source **sourceFactor** + **dest** destFactor ( E\_BLEND\_FUNC ) Using only Textures(0). generic Blender

## Chapter 9 Running The Exporter

Press the "Export" button to initiate the exporting of your scene and mesh files.

While **irrb** is exporting, the main GUI screen is hidden and status information is displayed in it's place. The status line contains information about the current mesh being exported as well image saving stats if the "Save Packed Images" option was selected.

You may choose to cancel the export at any time by pressing the Escape key.

After the export has completed, the main GUI screen will again be displayed with a summary of the information that was exported:



If **irrb** detects potential problems while exporting, it will display a list of warnings after the export completes. Click the "Back" button to return to the main GUI window.



## Chapter 10 iwalktest Integration And Usage

**irrb** integrates an optional utility named "iwalktest" which allows you to view and walk through your freshly exported scene using the Irrlicht engine. iwalktest is currently available to execute on both the Windows and Linux platforms.

Instructions on iwalktest installation and usage may be found here: [irrb Utilities](#)

The "Walk Test" GUI button will only appear in **irrb**'s GUI if the following conditions are met:

1. The "IWALKTEST" OS environment variable is defined.
2. The **irrb** "Create Scene File" toggle button is selected.

With these conditions met, it's simply a matter of selecting the "Walk Test" toggle button to view your scene immediately after exporting.

After you export a scene for the first time, a "**Run Walk Test**" push button will appear at the bottom of the **irrb** interface. Clicking on this button will re-execute iwalktest with the last scene exported.

If you prefer to run a different application other than the Tubras supplied "iwalktest.exe", you may do so by setting the "IWALKTEST" environment variable to point to your own application.

A couple of notes on how **irrb** executes the external application:

- **irrb** sets the current directory to the directory in which the external application resides before running the application.
- **irrb** passes the full scene file path and file name to the external application. If the environment variable contains a "\$1", the scene file is substituted in its place. A "\$2" substitution variable is replaced with value of the "Relative Base" path.

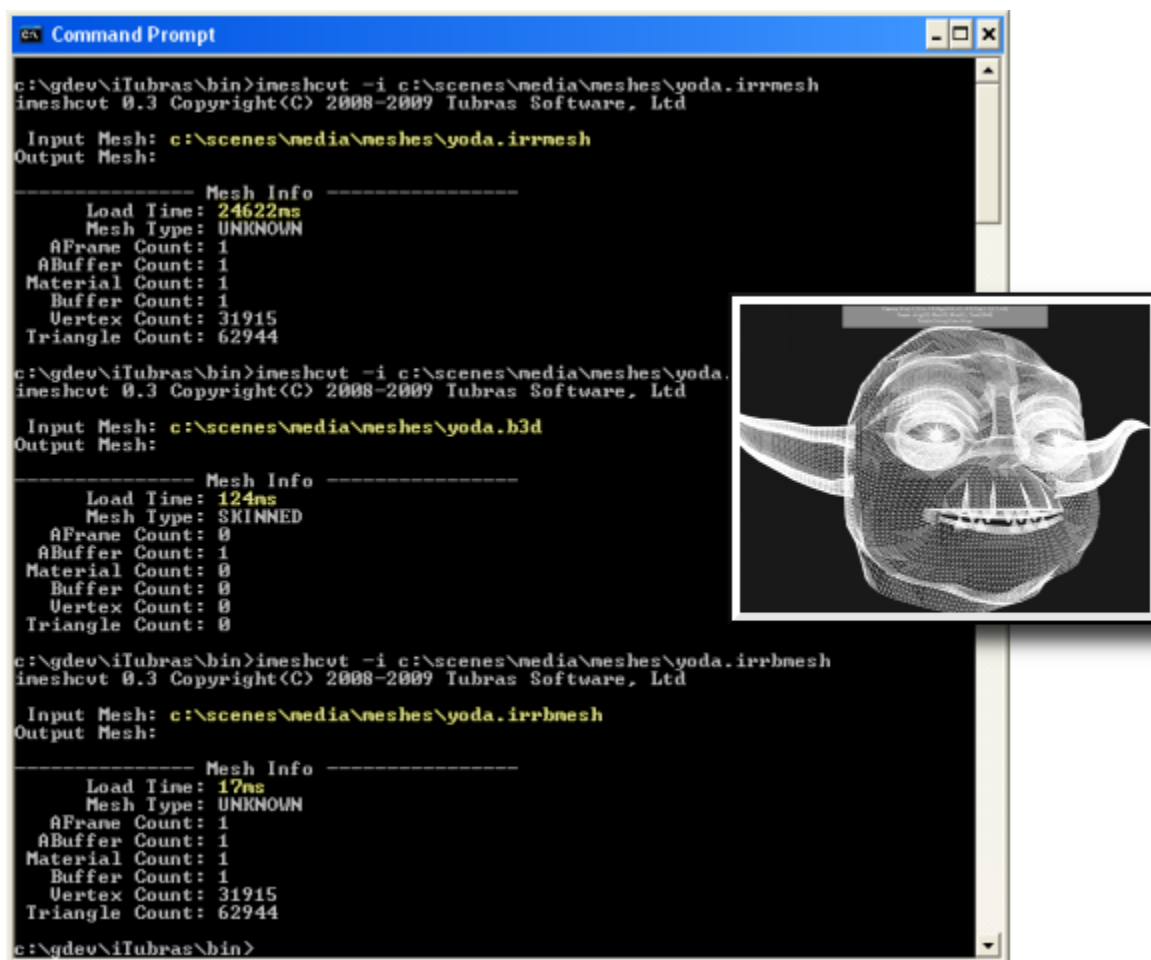
## Chapter 11 Binary Mesh Format

### Key Points

- The binary mesh format is currently experimental.
- irrmesh files (binary mesh) load into your application very quickly.
- The code to read/write the .irrmesh format is freely available to you under the same license as Irrlicht.

### Details

The experimental .irrmesh binary mesh format is fast loading. Irrlicht's native .irrmesh format is based on XML and all though it's nice for exchanging mesh information, it can be painfully slow to load especially for larger scenes and meshes. The following screen shot displays the load times for a high-poly "yoda" mesh exported as .irrmesh, .b3d, and .irrbmesh:



```

c:\gdev\iTubras\bin>imeshcut -i c:\scenes\media\meshes\yoda.irrmesh
imeshcut 0.3 Copyright(C) 2008-2009 Tubras Software, Ltd

Input Mesh: c:\scenes\media\meshes\yoda.irrmesh
Output Mesh:

----- Mesh Info -----
Load Time: 24622ms
Mesh Type: UNKNOWN
AFrame Count: 1
ABuffer Count: 1
Material Count: 1
Buffer Count: 1
Vertex Count: 31915
Triangle Count: 62944

c:\gdev\iTubras\bin>imeshcut -i c:\scenes\media\meshes\yoda.b3d
imeshcut 0.3 Copyright(C) 2008-2009 Tubras Software, Ltd

Input Mesh: c:\scenes\media\meshes\yoda.b3d
Output Mesh:

----- Mesh Info -----
Load Time: 124ms
Mesh Type: SKINNED
AFrame Count: 0
ABuffer Count: 1
Material Count: 0
Buffer Count: 0
Vertex Count: 0
Triangle Count: 0

c:\gdev\iTubras\bin>imeshcut -i c:\scenes\media\meshes\yoda.irrbmesh
imeshcut 0.3 Copyright(C) 2008-2009 Tubras Software, Ltd

Input Mesh: c:\scenes\media\meshes\yoda.irrbmesh
Output Mesh:

----- Mesh Info -----
Load Time: 17ms
Mesh Type: UNKNOWN
AFrame Count: 1
ABuffer Count: 1
Material Count: 1
Buffer Count: 1
Vertex Count: 31915
Triangle Count: 62944

c:\gdev\iTubras\bin>
  
```

The respective files sizes and load times are:

- yoda.irrmesh - 4,713,164 bytes loaded in 24.6 seconds.
- yoda.b3d - 1,521,446 bytes loaded in 124 **milli**-seconds.
- yoda.irrbmesh - 2,548,107 bytes loaded in 17 **milli**-seconds.

The yoda mesh contains approximately 60k faces in a single mesh buffer. This particular test was run several times in sequence in order to allow Windows disk caching to "work itself out". The main reason the .irrbmesh format performs so well is that it mimics Irrlicht's internal mesh buffer and material formats. Very little parsing

is required.

### Using .irrvmesh In Your Application

It is relatively easy to enable your application(s) to read and write .irrvmesh files. Simply save the following files to your applications source directory.

For Loading:

- [CIrrMeshFileLoader.h](#)
- [CIrrBMeshFileLoader.cpp](#)

For Writing:

- [CIrrMeshWriter.h](#)
- [CIrrBMeshWriter.cpp](#)

Then add the following code snippet to your application somewhere in your initialization code:

```
// define _IRR_COMPILE_WITH_IRRB_MESH_LOADER_ to compile the loader
// loader depends on writers header
#include "CIrrBMeshWriter.h"
#include "CIrrBMeshLoader.h"
// code for loading .irrvmesh files
//
// ISceneManager* sceneManager      - previously initialized.
// IFileSystem*   fileSystem        - previously initialized.
CIrrBMeshFileLoader* loader = new CIrrBMeshFileLoader(sceneManager,fileSystem);
sceneManager->addExternalMeshLoader(loader);
loader->drop();
```

Now you're ready to load .irrvmesh files:

```
IAnimatedMesh* pmesh = sceneManager->getMesh("mymesh.irrvmesh");
```

To write .irrvmesh files:

```
#include "CIrrBMeshWriter.h"
// code for writing .irrvmesh files
//
// IvideoDriver* videoDriver - previously initialized.
// IFileSystem*   fileSystem - previously initialized.
// IMesh*        mesh       - previously initialized.
IMeshWriter* writer = new CIrrBMeshWriter(m_videoDriver,m_fileSystem);
((CIrrBMeshWriter*)writer)->setVersion(0x0106); - Irrlicht version target
((CIrrBMeshWriter*)writer)->setCreator("myapp"); - Creator

IWriteFile* file;
file = fileSystem->createAndWriteFile("test.irrvmesh");
writer->writeMesh(file,mesh);
```

## Chapter 12 ID Properties

### Key Points

- Blender's “ID Property” system allows users to attach arbitrary data to Blender Objects, Data Blocks, and Materials.
- **irrb** uses Object and DataBlock ID Properties when generating .irr scene node Standard and User attributes.
- **irrb** optionally uses Material ID Properties when generating .irmesh material attributes.

### Details

Blender allows custom “string”, “int”, “float”, “array”, and “group” data values to be assigned to any data block and/or object. This type of data is known as “ID Properties” in Blender. **irrb** makes use of ID Properties (when available) to generate Standard and User attributes for scene nodes and Material attributes for Irrlicht mesh buffers located in .irmesh files.

Id Properties are “key”, “value” pairs where a “key” may be any arbitrary identifier and the “value” may be any of the data types mentioned above. A “group” value type is simply a group of additional properties. The following is an example of ID Properties that may be assigned to an object:

Key	Value	Type
"irrb"	group	group
--"StandardAttributes"	group	group (sub-group of 'irrb')
----"Visible"	0	int
----"IsDebugObject"	1	int
--"UserAttributes"	group	group (sub-group of 'irrb')
----"Static"	1	int
----"Collider"	"box"	string
----"testFloat"	1.234	float

This example starts with a top level group (think Python dictionary) named “irrb” which contains all of the other properties. This group is required by the **irrb** exporter in order to differentiate and avoid conflicts with other scripts that may also choose to use ID Properties for storing script specific data.

Underneath the “irrb” group, there are two “sub-groups” named “StandardAttributes” and “UserAttributes”. Let's have a look at these two sub-groups and how **irrb** uses them when generating scene node data .

Every scene node that is written to an .irr scene file contains standard and optional user attributes. Standard attributes include:

```
<vector3d name="Position" value="0.000000, 0.000000, 0.000000" />
<vector3d name="Rotation" value="0.000000, 0.000000, 0.000000" />
<vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
<bool name="Visible" value="true" />
<bool name="IsDebugObject" value="false" />
```

The position, rotation, and scale are automatically created and assigned by **irrb** according to the object (mesh, light, camera) being exported. These attributes may not be overridden using ID Properties. However, attributes such as “Visible”, “IsDebugObject”, “Id”, etc. may be overridden using ID Properties. When **irrb** exports an object like the “Cube” in this example, it performs the following checks when generating the scene node

Standard attributes:

1. Set the Cube attributes from the “**defaults**”.
2. Does the Cube **DataBlock** contain a top level ID Property group named “irrb”. If so, override the current attributes with those present in the **DataBlock** “StandardAttributes” properties.
3. Does the Cube **Object** contain a top level ID Property group named “irrb”. If so, override the current attributes with those present in the **Object** “StandardAttributes” properties.

This same logic is also used for “UserAttributes”. Again, using our example above, the following scene node would be created in the scene .irr file:

```
<node type="mesh">
  <attributes>
    <string name="Name" value="Cube" />
    <int name="Id" value="-1" />
    <vector3d name="Position" value="0.000000, 0.000000, 0.000000" />
    <vector3d name="Rotation" value="0.000000, 0.000000, 0.000000" />
    <vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
    <bool name="Visible" value="false" />
    <enum name="AutomaticCulling" value="frustum_box" />
    <bool name="DebugDataVisible" value="false" />
    <bool name="IsDebugObject" value="true" />
    <bool name="ReadOnlyMaterials" value="false" />
    <string name="Mesh" value="meshes/Cube.irrmesh" />
  </attributes>
  <userData>
    <attributes>
      <int name="Static" value="1" />
      <string name="Collider" value="box" />
      <float name="testFloat" value="1.234000" />
    </attributes>
  </userData>
</node>
```

So where do the values for “AutomaticCulling”, “DebugDataVisible”, and “ReadOnlyMaterials” come from? The answer: the “global” default values, which are also configurable by you, the user. Before we discuss default values, let's have a look at the order in which irrb extracts attributes from ID Properties:

1. Default Properties
2. DataBlock Properties (mesh, camera, light, etc.)
3. Object Properties

For every object exported, it is assigned the “**Default**” properties. The Default properties are then overridden by **DataBlock** properties, if they exist. And finally, DataBlock properties are overridden by the **Object** properties, if they exist.

## Default Property Attributes

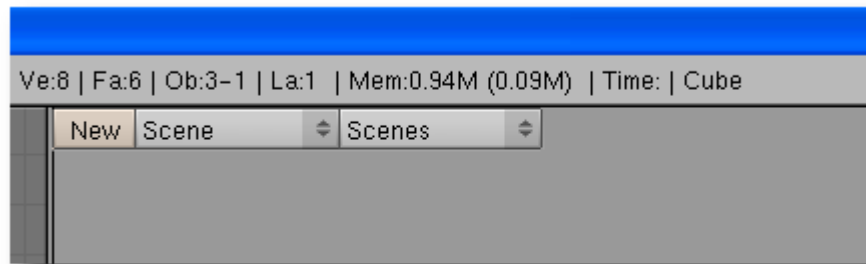
When **irrb** is loaded by Blender, it reads the default scene node and material attributes from the “iConfig” module (**iConfig.py**) located in the “irrbmodules” sub-directory. This module contains a group of Python dictionaries which hold the default attribute values for Scene Nodes and Materials.

You could modify this file to change the defaults (don't do it), but your changes would be overwritten the next time you upgrade **irrb** to the latest version. Instead of modifying “iConfig.py”, simply copy it to a file named “**UserConfig.py**” in the same directory (irrbmodules) as “iConfig.py”. Any changes you make to “UserConfig.py” will then become the global “defaults”.

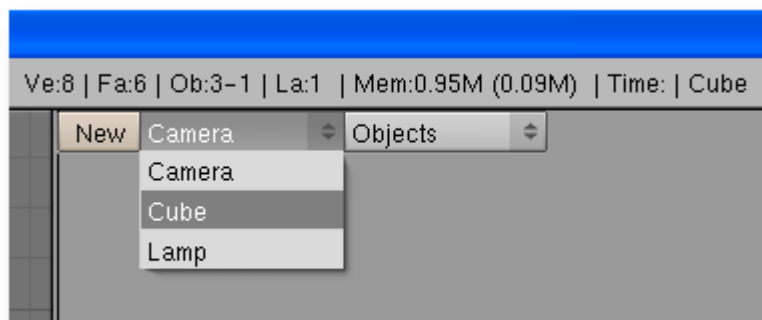
## Blender's ID Property Editor

Blender contains a crude ID Property editor which allows you to add, update, and/or delete properties at the Object and DataBlock level. Let's walk through the process of **manually** adding a UserData attribute to the default Cube Object that typically appears when Blender is first started.

Start the ID Property editor by selecting the Blender Menu Item: “Help|ID Property Browser”. The “empty” attribute window will look like this:



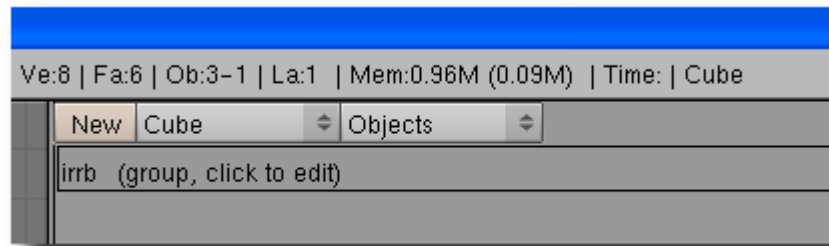
Because we want to add a user attribute to the “Cube” **Object**, select “Objects”, and then select “Cube” from the list of available objects on the left:



The first property we need to add is the top level “**irrb**” group property:

1. Click the “New” button.
2. Set the “Name” field to “irrb”.
3. Select the “Subgroup” button.
4. Click the “OK” button.

The top level “irrb” group will now look like this:

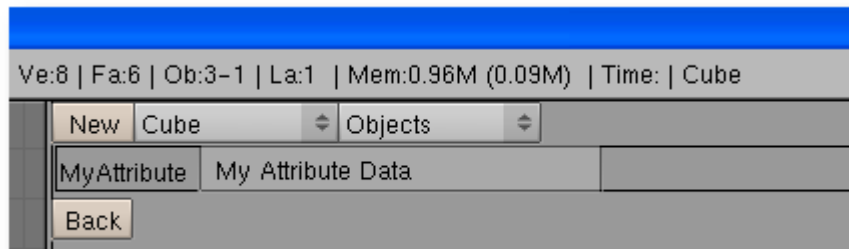


Within the “**irrb**” group we are now ready to create the “**UserAttributes**” group:

1. Click “irrb” group - “irrb (group, click to edit)”.
2. Add another group with the name “UserAttributes” following the same steps used to create the “irrb” group.

Move into the “UserAttributes” group by clicking on “UserAttributes (group, click to edit)”. Now we are ready to add one or more user attributes. Let's add a String attribute named “MyAttribute”:

1. Click the “New” button.
2. Set the “Name” field to “MyAttribute”.
3. Select the “String” button.
4. Click the “OK” button.



This creates a String property named “MyAttribute” with an empty value. Update the value by typing in the value data and then press the Enter key:

Now whenever we export this scene, the Cube scene node will always contain the following UserData attribute:

```
<userData>
  <attributes>
    <string name="MyAttribute" value="My Attribute Value" />
  </attributes>
</userData>
```

## Generating ID Properties

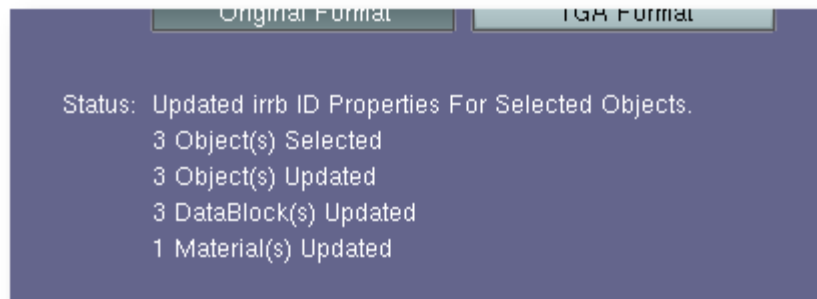
From the previous section, it is easy to see that manually creating ID properties using Blender's property editor can be tedious at best. A quicker more efficient method is to use the “**Create irrb Props**” push button located in the **irrb** exporter interface:



Clicking on “Create irrb Props” will perform two operations:

1. Create the top-level “irrb” group and sub-groups: “StandardAttributes” and “UserAttributes” for all **selected** objects.
2. Create the top-level “irrb” group and default Irrlicht material attributes for all Blender Materials.

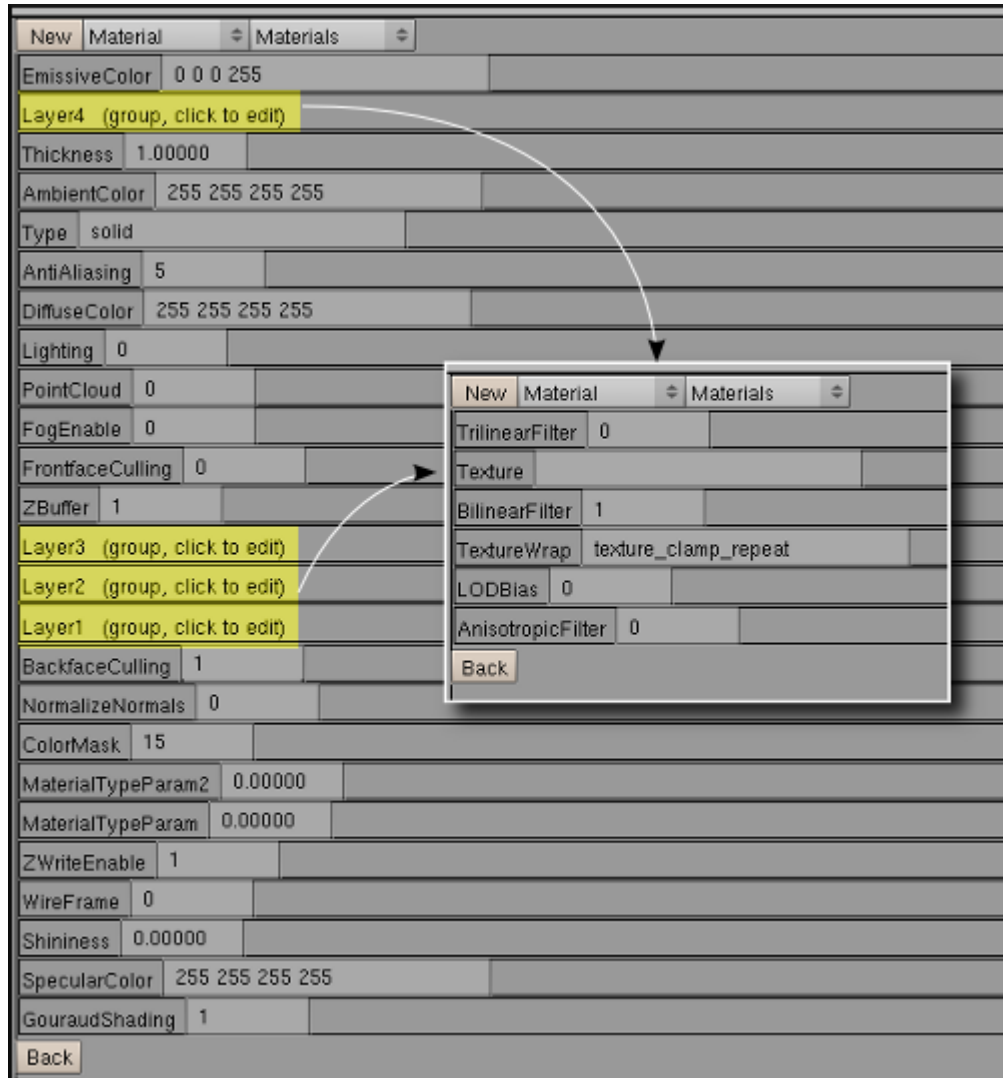
Note that if an Object, DataBlock, or Material already contains a top-level “irrb” group, the property generator skips over it. It does NOT replace existing properties. When the property generator finishes its work, it displays update information in the status area of the **irrb** exporter:





## Material ID Properties

ID Properties assigned to Blender materials may be used to control the generated .irrmesh material attributes. Using the “Create irrB Props” button, will automatically generate ID Properties:



*Illustration 9: Generated ID Properties for a Blender Material*

Notice that Material **Layer** attributes are contained within individual sub-groups named “Layer#”. To modify a specific layer attribute value (TextureWrap, BilinearFilter, LODBias, etc.), simply click on the group name to display the corresponding properties.

When the property generator creates “irrB” properties for new materials, it sets the values to those located in the global defaults (iConfig.py or iUserConfig.py).

All “Color” related property values are specified as “string” values entered as “r g b a” integers 0-255.

## Accessing User Attributes

In order to access scene node UserData properties in your application, you will need a class that implements the [ISceneUserDataSerializer](#) interface overriding the **OnReadUserData** method:

```
void SerializerClass::loadScene(const c8* fileName)
{
    //
    // ISceneManager* sceneManager (previously initialized)
    //
    // The 2nd parameter to "loadScene" is a pointer a ISceneUserDataSerializer
    // interface.
    //
    sceneManager->loadScene(fileName, this);
}
//
// this will be invoked for every scene node that contains UserData attributes.
//
void SerializerClass::OnReadUserData(ISceneNode* forSceneNode,
    io::IAttributes* userData)
{
    bool value=false;
    if(userData->existsAttribute("Dynamic"))
        value = userData->getAttributeAsBool("Dynamic");
    if(value)
    {
        ESCENE_NODE_TYPE type = forSceneNode->getType();
        if(type == ESNT_MESH)
        {
            //
            // do something to make the mesh "dynamic"
            //
        }
    }
}
```

Custom scene node properties aren't limited to just meshes. They may also be defined for lights and cameras.

## Chapter 13 Cameras And Lights

### Key Points

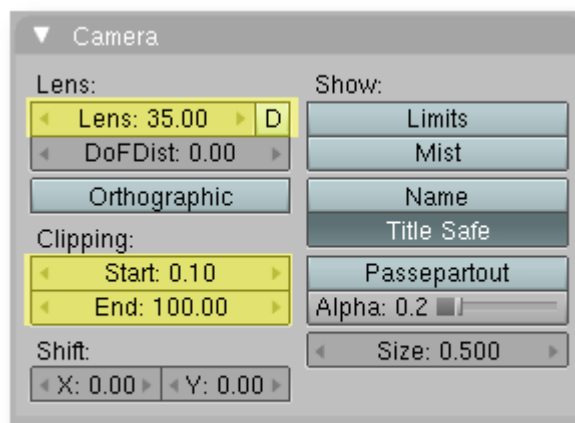
- **irr** allows you to export camera and light node data to scene files.
- Camera/Light node attributes are based on some of Blenders settings but may be overridden by assigning specific Blender ID Properties.

### Camera Details

A **camera** node in a .irr scene file will appear as follows:

```
<node type="camera">
  <attributes>
    <string name="Name" value="Camera" />
    <int name="Id" value="-1" />
    <vector3d name="Position" value="7.481132, 5.343665, -6.507640" />
    <vector3d name="Rotation" value="26.440704, -46.691944, 0.000000" />
    <vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
    <bool name="Visible" value="true" />
    <enum name="AutomaticCulling" value="frustum_box" />
    <bool name="DebugDataVisible" value="false" />
    <bool name="IsDebugObject" value="false" />
    <bool name="ReadOnlyMaterials" value="false" />
    <vector3d name="Target" value="0.000000, 0.000000, 0.000000" />
    <vector3d name="UpVector" value="0.000000, 1.000000, 0.000000" />
    <float name="Fovy" value="0.857556" />
    <float name="Aspect" value="1.250000" />
    <float name="ZNear" value="0.10" />
    <float name="ZFar" value="100.00" />
  </attributes>
</node>
```

The camera specific attributes are: Target, UpVector, Fovy, Aspect, ZNear, and ZFar. All but the Target and UpVector may be set using the Camera properties (F9):



**Fovy.** Set to  $(2 * \text{atan}(16.0 / (\text{Blenders Lens} \rightarrow \text{Lens value})))$ . May be overridden with a "float" logic property named "Fovy".

**Aspect.** Set to the constant 1.25. May be overridden with a "float" logic property named "Aspect".

**ZNear.** Set to Clipping -> Start value.

**ZFar.** Set to Clipping -> End value.

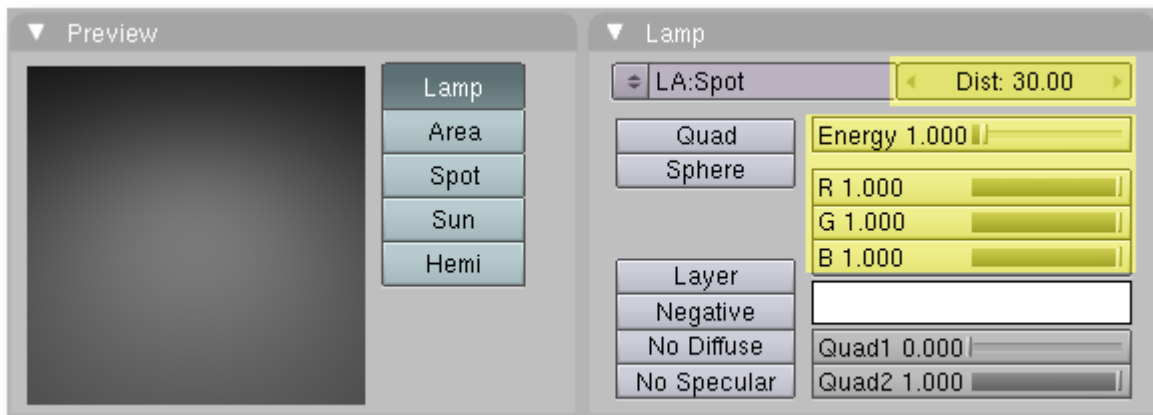
For now, the Target and UpVector attributes are set to (0,0,0) and (0,1,0) respectively.

### Light Details

A **light** node in a .irr scene file will appear as follows:

```
<node type="light">
  <attributes>
    <string name="Name" value="Lamp" />
    <int name="Id" value="-1" />
    <vector3d name="Position" value="4.076245, 5.903862, 1.005454" />
    <vector3d name="Rotation" value="52.738953, -16.936317, 0.000000" />
    <vector3d name="Scale" value="1.000000, 1.000000, 1.000000" />
    <bool name="Visible" value="true" />
    <enum name="AutomaticCulling" value="frustum_box" />
    <bool name="DebugDataVisible" value="false" />
    <bool name="IsDebugObject" value="false" />
    <bool name="ReadOnlyMaterials" value="false" />
    <enum name="LightType" value="Point" />
    <colorf name="AmbientColor" value="0.000000,0.000000, 0.000000, 1.000000" />
    <colorf name="DiffuseColor" value="1.000000, 1.000000, 1.000000 1.000000" />
    <colorf name="SpecularColor" value="1.000000,1.000000, 1.000000, 1.000000" />
    <vector3d name="Attenuation" value="0.000000 0.500000 0.000000" />
    <float name="Radius" value="60.00" />
    <bool name="CastShadows" value="true" />
  </attributes>
</node>
```

The light specific attributes are: LightType, DiffuseColor, SpecularColor, Attenuation, Radius, and CastShadows.



**LightType.** Based on Blenders lamp type:

- lamp -> Point
- area -> Directional
- spot -> Spot
- sun -> Directional
- hemi -> Directional

**AmbientColor.** Constant (0,0,0).

**DiffuseColor.** Set to the R,G,B components of the light color.

**SpecularColor.** Constant (1,1,1).

**Attenuation.** Set to (0.5 / (Blenders Energy Value)).

**Radius.** Set to (2.0 \* (Blenders Distance Value)).

**CastShadows.** Constant "True".

## Chapter 14 Creating A Billboard

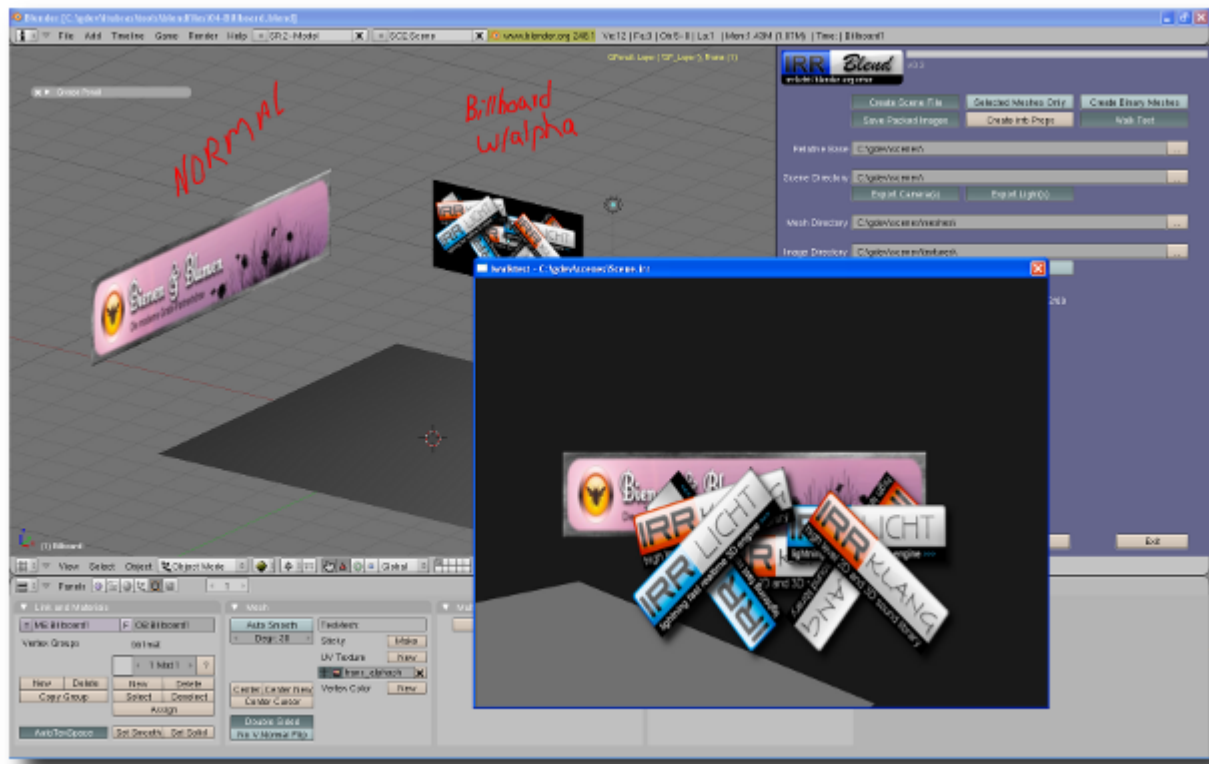


Illustration 10: Sample Billboard .blend file – [04-Billboard.blend](#)

Creating a Billboard in Blender that will be exported as an Irrlicht Billboard scene node is a matter of 3 steps:

1. Add a plane (quad) to your scene. The position and scale of the plane are relevant. The rotation is ignored.
2. Assign an image to the Billboard using Blender's UV/Image Editor.
3. Add and “irr” object ID property (“inodetype” = “billboard”) to the plane indicating that the object should be exported as a Billboard scene node.

The Billboard size is automatically calculated by irr using the Plane's vertex coordinates as well as the scale factor.

Irrlicht Billboard scene nodes contain two optional color parameters named "Shade\_Top" and "Shade\_Down". The default values are White (0xFFFFFFFF). You may optionally override the colors by assigning color values to Billboard object's ID Properties:

"irr" | "Shade\_Top" | "r g b a"

"irr" | "Shade\_Down" | "r g b a"

## Chapter 15 Creating A Skybox

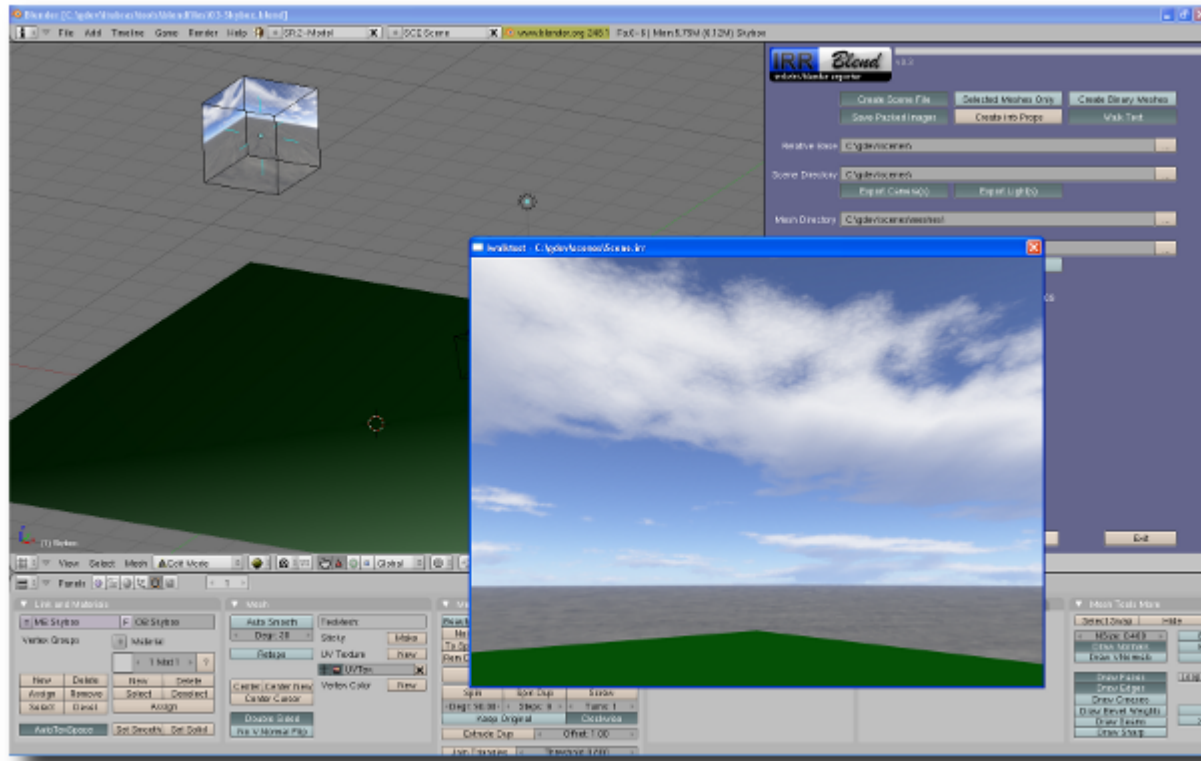
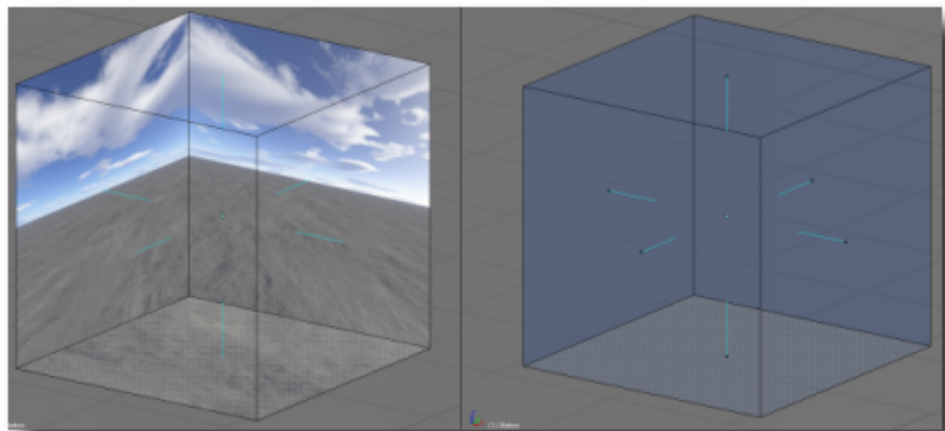


Illustration 11: Sample Skybox .blend file – [03-Skybox.blend](#)

Creating a Skybox in Blender that will be exported as an Irrlicht Skybox scene node requires 4 steps:

1. Add a Cube mesh to your scene. The position and scale are not relevant. However, the rotation is important. The Cube should **NOT** be arbitrarily rotated. In order for irrbb to determine the which images are the front/back, left/right, and top/bottom, each side of the cube must be perpendicular to the Blender axes.
2. Each one of the six Skybox images must be individually mapped to the appropriate face on the cube. The easiest method for doing this is to use “face select mode” CTRL-TAB, “faces”. Select the appropriate face to map, unwrap (U), and then assign the corresponding image.
3. Flip the face normals so that all of normals are facing inward – Select all of the faces (CTRL-A), W, “flip normals”.
4. Add an “irrbb” object ID property (“inodetype” = “skybox”) to the Cube indicating that the object should be exported as an Irrlicht Skybox scene node.



## Chapter 16 irrB Utilities

The utilities **iwalktest** and **imeshcv** may be optionally installed to enhance the functionality of **irrB**.

**iwalktest** is used to view exported Blender scenes. **imeshcv** is used to convert meshes to an experimental binary mesh format (.irrbmesh).

### Installation

The latest stable download of **irrBUtilities** is accessible from both the "Project" and "Downloads" tabs of this [Tubras Google Code project](#). On the "Projects" tab, **irrBUtilities** is listed in the "Featured Downloads" section.

The current version is 0.3. It may be downloaded from here:

Windows:

[irrbUtils-0.3-win32.zip](#)

Linux:

[irrbUtils-0.3-linux32.zip](#)

The contents of the zip package are:

```
iwalktest.exe           // windows executable
iwalktest               // linux executable
imeshcv.exe            // windows executable
imeshcv                // linux executable
data/cfg/irrlicht.lsl   // irrlicht configuration definitions
data/cfg/iwalktest.lsl  // iwalktest configuration
data/fnt/defaults.zip   // fonts used by iwalktest
data/tex/lamp.tga       // texture used for debug lights
```

Unzip the contents to a user **writable** directory because **iwalktest** creates a log file in the directory it is executed from.

### Configuration

In order to integrate **iwalktest** and **imeshcv** with **irrB**, you need to set up environment variables which point to the locations of their respective executables.

**Windows** If installed/unzipped to "c:\irrbutils", then you would set the following environment variables:

```
IWALKTEST=c:\irrbutils\iwalktest.exe -i "%1" -a "%2"
IMESHCVT=c:\irrbutils\imeshcv.exe
```

**Linux** if installed to "/home/kmurray/irrbutils", then you would set the following environment variables:

```
IWALKTEST=/home/kmurray/irrbutils/iwalktest -i "%1" -a "%2"
IMESHCVT=/home/kmurray/irrbutils/imeshcv
```

The "\$1" and "\$2" substitution variables inform irrB where to insert the "Scene File" and "Base Directory" command line parameters.

Regardless of the OS you use, you may wish to run a different application instead of **iwalktest** (maybe your own or [irrEdit](#)). To do this simply set the appropriate value in the **IWALKTEST** environment variable. For example to run [irrEdit](#) under Windows you would set the value to (assuming [irrEdit](#) is installed to "c:\irredit"):



```
IWALKTEST=c:\irredit\irredit.exe "$1"
```

This will cause the scene you export with **irrB** to be opened in **irrEdit** immediately after the export finishes (the scene file name is passed to irrEdit as a parameter).

If you want to run an application that requires information other than the scene file name you may set up the environment variable using the "\$1" substitution variable. For example, your application (myapp.exe) requires a "-i" in front of the scene file name:

```
IWALKTEST=c:\myappdir\myapp.exe -i "$1" -u
```

**irrB** will make the appropriate substitution before invoking your application.

After the environment variables have been properly set up, you may choose to configure **iwalktest** by tweaking its configuration file "iwalktest.lsl":

```
require 'irrlicht'
options =
{
    debug = 1,
    console = true,
    velocity = 4.0,
    angularvelocity = 100.0,
    maxvertangle = 80,
    showcursor = true,
    defcampos = {0, 5, -50},
    defcamtarget = {0, 0, 0},
    -- loadscene = '/temp/scene.irr',
}

video =
{
    driver = EDT_OPENGL,
    -- driver = EDT_DIRECT3D9,
    resolution = {1024, 728},
    -- resolution = {800, 600},
    -- resolution = {640, 480},
    colordepth = 32,
    fullscreen = false,
    vsync = true,
    stencilbuffer = false,
    -- antialias = 0, 2, 4, 8, 16
    antialias = 8,
    bgcolor = {25,25,25,255},
    guiskin = 'gui/tubras2.lsl',
}

filesystems =
{
    -- folders = {'c:/scenes/'},
    zipfiles = {},
    pakfiles = {},
}

keybindings =
{
    ['key.down.w'] = 'frwd 1',
    ['key.up.w'] = 'frwd 0',
    ['key.down.a'] = 'left 1',
}
```

```

['key.up.a'] = 'left 0',
['key.down.s'] = 'back 1',
['key.up.s'] = 'back 0',
['key.down.d'] = 'rght 1',
['key.up.d'] = 'rght 0',
['key.down.e'] = 'mvup 1',
['key.up.e'] = 'mvup 0',
['key.down.c'] = 'mvdn 1',
['key.up.c'] = 'mvdn 0',
['key.down.right'] = 'rotr 1',
['key.up.right'] = 'rotr 0',
['key.down.left'] = 'rotl 1',
['key.up.left'] = 'rotl 0',
['key.down.up'] = 'rotf 1',
['key.up.up'] = 'rotf 0',
['key.down.down'] = 'rotd 1',
['key.up.down'] = 'rotd 0',
['key.down.lshift'] = 'avel 3.5',
['key.up.lshift'] = 'avel 1.0',
['key.down.i'] = 'invert-mouse',
['key.down.f1'] = 'help',
['key.down.f2'] = 'idbg',
['key.down.f3'] = 'wire',
['key.down.f4'] = 'pdbg',
['key.down.f5'] = 'cdbg',
['key.down.f6'] = 'xfrm',
['key.down.prtscr'] = 'sprt',
['key.down.esc'] = 'quit',
}

```

**iwalktest** field descriptions:

**-options-**

**debug** (int).

- The amount of debug information written to "iwalktest.log".

**console** (bool).

- Whether or not iwalktest should create a console window that displays debug information. Windows only.

**velocity** (float).

- The camera movement speed in units/second.

**angularvelocity** (float).

- The camera rotational speed in °/second.

**maxvertangle** (float).

- The maximum angle (up/down) a camera may rotate to.

**showcursor** (bool).

- Whether or not the OS cursor should be visible. Linux users should leave this set to "false".

**loadscene** (string).

- The scene that should be loaded if a scene isn't passed on the command line.

**defcampos** (vector3f).

- The initial position of the default camera created by **iwalktest**.

**defcamtarget** (vector3f).

- The initial target of the default camera created by **iwalktest**.

#### **-video-**

**driver** (int).

- The Irrlicht video driver to use. EDT\_OPENGL – OpenGL, EDT\_DIRECT3D9 – DirectX.

**bgcolor** (vector3d).

- The RGB colour values (0-255) to use for the background colour.

**antialias** (int).

- Anti-aliasing level.

**colordepth** (int).

- The colour depth to use.

**fullscreen** (bool).

- Whether or not to run in full screen mode.

**resolution** (int int).

- The X,Y screen resolution to use.

**vsync** (bool).

- Whether or not to run in vsync mode.

#### **-media-**

**fontfolder** (string).

- Used to control the location of the default fonts used by **iwalktest**.

#### **-filesystems-**

This sections contains the filesystems that **iwalktest** should load/mount during initialization. Possible values include:

<folder>, <zipfile>, and/or <pakfile>

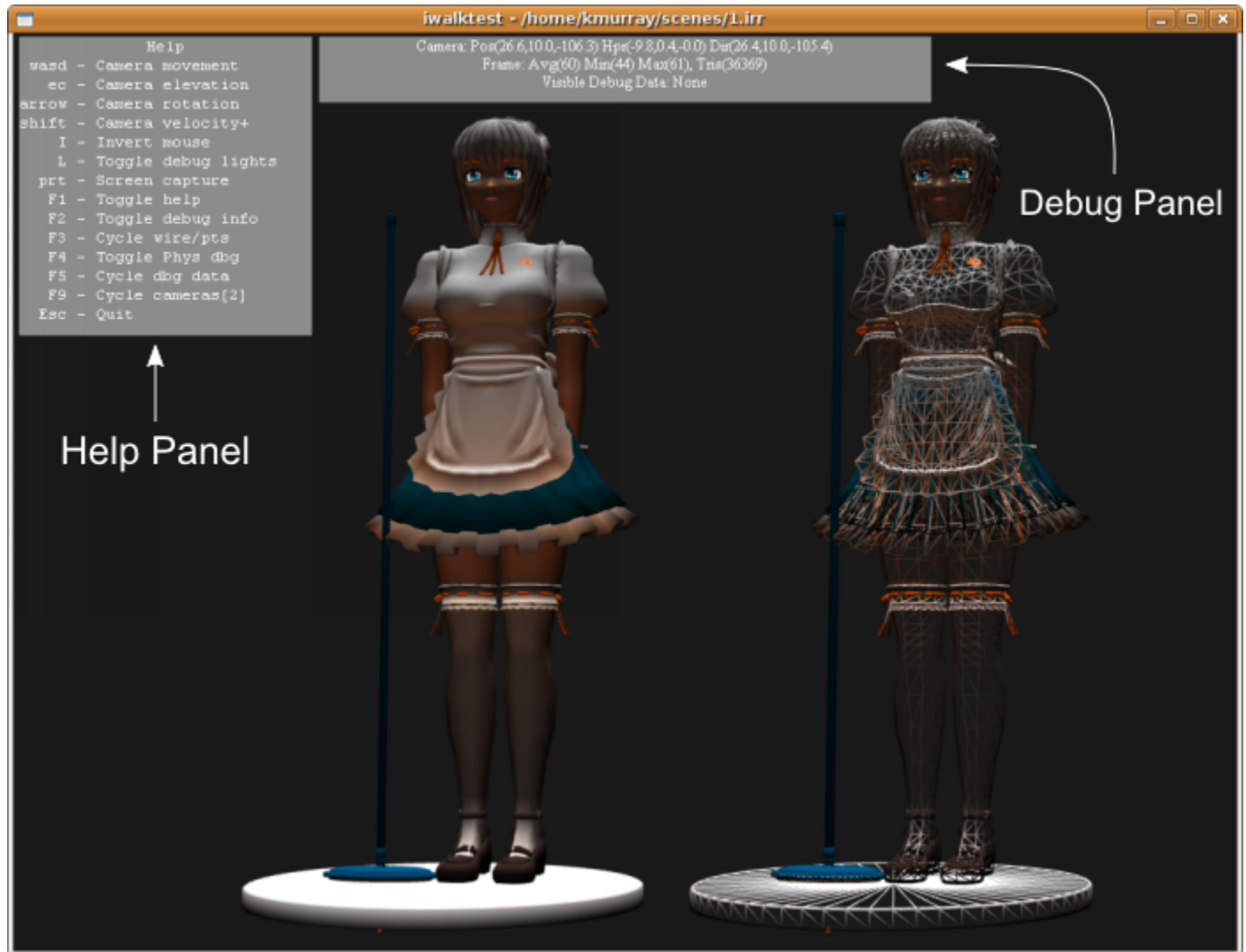
#### **-keybindings-**

This section is used for setting up key bindings to "commands" used by **iwalktest**. These bindings primarily control camera movement. The defaults are "wasd" for movement and arrow keys for rotation. The mouse may also be used for rotation.

## iwalktest Usage

The default application used to view scenes exported by **irrB** is **iwalktest**.

**iwalktest** is a simple Irrlicht application that allows you to freely (God camera) move around and view your scene. It also allows you to easily switch (F9 key) between the various cameras you choose to export.



From the screen shot above you can see that **iwalktest** is capable of displaying two overlay panels - the help panel and the debug panel.

When **iwalktest** initially starts, the help panel is visible and the debug panel isn't. The help panel identifies the keys and functionality that is available while **iwalktest** is running:

**wasd** - Camera movement keys. Holding the Shift key while moving will accelerate the movement by the amount defined in **iwalktest.lsl** ("avel" command). Note the "e" and "c" keys will move the camera **up** and **down** respectively.

### Help Panel

**arrow** – The up/down/left/right arrow keys control camera rotation.

**I** - Inverts the up-down mouse movement.

**L** – Toggles visual light debug data.

**prt** - (Print Screen) will save a screen shot of the view to a file named "cap###.png" (## - screen shot number). The file will be save into the same directory **iwalktest** is located in.

**F1** - Toggles the visibility of the Help panel.

**F2** - Toggles the visibility of the Debug panel.

**F3** - Cycles the display mode (textured, wireframe, point cloud).

**F4** - Not implemented.

**F5** - When the Debug Panel is visible, cycles through the various visual debug modes: vertex normals, bounding box, half transparency etc.

**F9** - If more than one camera exists in the scene, F9 cycles the active camera.

### Debug Panel

The **first line** of the debug panel displays camera information:

```
[camera name]: Pos(x,y,z), Hpr(x,y,z), Dir(x,y,z)
```

"Hpr" - Heading, Pitch, Roll

iwalktest always creates a default scene camera named "tcam".

The **second line** of the debug panel displays frame rate info and the number of triangles that are being rendered.

The **third line** of the debug panel displays what visual debug data is currently visible (if any).

## imeshcvT Usage

**imeshcvT** is a command line utility that allows you to:

- Display information about a mesh.
- Convert a mesh to another format.
- Manipulate a mesh using Irrlicht's [IMeshManipulator](#).

**imeshcvT** options:

usage: imeshcvT <options> -i[input file] -o<output file>

<options> - Generic options:

-a : folder archive

-v : target mesh version (.irrBmesh only)

<options> - Mesh Manipulator options:

-f : flip surfaces

-n : recalculate normals

-s : recalculate normals smooth

-t : create tangents

[input file]      Input mesh file to convert or report on. if no output mesh is specified, info is displayed for the input mesh. Required.

<output file>      Output mesh file to convert to.

If no output file is given, **imeshcvT** will simply display information about the input mesh:

```
Input Mesh: \test\meshes\sphere.irrmesh
Output Mesh:
----- Mesh Info -----
    Load Time: 76ms
    Mesh Type: UNKNOWN
    AFrame Count: 1
    ABuffer Count: 1
Material Count: 1
    Buffer Count: 1
    Vertex Count: 994
Triangle Count: 1984
```

## Chapter 17 Notes

- add iwalktest export to /etc/environment on Ubuntu.
- if permissions problem saving irrbexport.cfg on Linux, create a writeable user defined scripts directory and point Blender's user defined scripts to it ("Python" in preferences). This newly created directory must contain a sub-directory named "bpydata/config".
- to view **irrb** stdout debug info on Linux, start Blender from a console window: "gnome-terminal/xterm -e blender -w". "-w" runs Blender in windowed mode.
- images/copying. When Blender creates an image internally, it is created as a 24 bit image with an internal type of Targa (tga). If the image is saved, it is saved as a 24 bit .tga file even if the file extension is set to some other extension. The bpy "images" data module allows a bit depth to be specified when creating a new image, but the docs say 32 bit is currently unsupported. Because of this, we still allow images to be copied, but not "converted" to another type. Originally, conversion appeared to work because the "reading" software was smart enough to recognize the image type even though it's extension didn't match the actual data. Conversion won't happen unless we include an external Pythonic imaging library or Blender allows us to set the internal image type.
- it is possible to generate an image within Blender and save it without entering a file extension. Blender will save the image to a file with an extension that relates to the image type specified during the save. However, it doesn't set the extension properly on the internal name. So,
  - a generated image with a name of "Untitled"
  - Manually saved as a TGA type (to c:\temp), extension isn't manually entered
  - File will be saved as c:\temp\untitled.tga
  - The internal filename will be set to c:\temp\untitled
  - The image "name" will be set to untitled.tga

irrb attempts to deal with this, but it is better handled by entering the file extension at the time the image is saved.

- Blender lamp type conversions lamp->point, area->direction, spot->spot, sun->directional, and hemi->directional.
- Concave quads should be converted to Triangles before exporting. Show example...

## Chapter 18 FAQ

**Q:** I have a question, where do I go?

**A:** You may post **irrb** related questions in this [Irrlicht thread](#). Please read the appropriate section of this document before posting a question.

**Q:** Can I export animated meshes?

**A: Not Yet.** The .irrmesh file format currently doesn't support animation.

**Q:** Can **irrb** invoke my own application instead of **iwalktest** after exporting?

**A: Yes.** See the section "[iwalktest Integration And Usage](#)"

**Q:** Where can I report bugs?

**A:** [In the irrb Bug thread](#).

**Q:** Where can I submit a patch?

**A:** [In the irrb Patch thread](#).