Table of Contents

1 Overview

1.1 Description

A planeshift effect is defined by a single CS library xml file saved as with a .eff file extension. This CS library defines all materials, textures, and meshes that the effect will depend on. The effect portion of the library is defined by a custom addon xml block inside the library file.

An effect is mode of one or more Effect Objs. These Effect Objs are everything you can see and hear about the effect. Every Effect Obj is attached to an Effect Anchor, which is a means to provide a base position for your Effect Objs.

1.2 Effect file example

```
<?xml version="1.0" encoding="utf-8" ?>
<library>
      <library>/path/to/a/library/dependency</library>


      <textures>
            <texture name="texture1">
                  <file>/this/art/effects/texture1.dds</file>
            </texture>
```

```
            <texture name="texture2">
                  <file>/this/art/effects/texture2.dds</file>
            </texture>
      </textures>


      <materials>
            <material name="material1">
                  <texture>texture1</texture>
            </material>
            <material name="material2">
                  <material>texture2</material>
            </material>
      </materials>


      <addon plugin="PSEffects">
            <effect name="effect_name">
                  <!-- Place effect anchors here -->

                  <!-- Place effect objs here -->
            </effect>
      </addon>
</library>
```
The anchor and objs sections not shown above are the heart of the effect and will be detailed below.

1.3 KeyFramed Animation

Each Effect Anchor and Effect Obj uses keyframes to animate itself. A keyframe consists of a time value and a set of actions. While animating, the Effect Manager will take the current time, find the keyframes that occur just before and just after that time, and then interpolate the action values between the two keyframes.

Every action in a keyframe is also optional. That is, you do not explicity need to specify a value for each action you use for every keyframe. To illustrate, lets say you have a set of three keyframes:

```
<keyFrame time="0">
      <action name="scale" value="1.0" />
      <action name="position" x="0" y="0" z="0" />
</keyFrame>
<keyFrame time="500">
      <action name="scale" value="2.0" />
</keyFrame>
<keyFrame time="1000">
      <action name="scale" value="1.0" />
      <action name="position" x="1" y="1" z="1" />
</keyFrame>
```

Notice how a position action was not specified for the middle keyframe? Since it was not specified in the middle keyframe, the position will be interpolated between the previous and next keyframes that have specified a position action. So if the current time were 500ms, then the current position value would be x="0.5" y="0.5" z="0.5", since it would interpolate correctly across keyframes with the position action missing.

If a keyframe action has multiple values, then each value is optional. For example, a position action can be specified as:

```
<action name="position" y="1" />
```
omitting the x, and z values. These values will use defaults and be interpolated as needed.

2 Effect Anchors

The effect anchors are key to positioning effect components; these are animated using keyframes. The basic format for an effect anchor is as follows:

```
<anchor type="anchor_type" name="enter_id_used_to_refer_to_this_anchor">
      <dir>dir_type</dir>
      <!-- other non-keyframe options go here -->
      <keyFrame time="0">
            <action name="action_name" action values go here />
      </keyFrame
      <keyFrame time="1000"><!-- A keyframe for a second later -->
            ...
      </keyFrame>
</anchor>
```

2.1 Global Anchor Options

2.1.1 dir

The dir option is not specified inside a keyframe. This is an option that remains constant across all keyframes in an anchor.

The dir dictates the orientation of the position defined by the keyframes.

Possible dir_type values:

- **target** specifies that the direction should be the direction that the target is facing
- **origin** specifies that the direction should be in the direction that the origin is facing (ie, maybe a casting effect that shoots a beam from the player's chest or something)
- **none** no direction for the anchor; with no direction any position's x, y, and z values will remain untransformed.

2.2 Anchor KeyFrame Actions

2.2.1 Position

Position is a keyframe action type. It dictates the position of the anchor while following the rules set by the dir option. The position action accepts values for x, y, and z. z specifies a position along the forward direction, x specifies a position along the sideways direction, and y specifies a position along the upwards direction.

2.2.2 ToTarget

ToTarget is a keyframe action type. This dictates the position of the anchor similar to the position action, but the ToTarget action is always oriented towards the effect's target, and the values given in ToTarget will be multiplied by the distance between the effect's origin and target.

For example, a ToTarget action defined as

```
<action name="ToTarget" x="0" y="0" z="1" />
```
will place the anchor directly on the effect's target.

2.3 Basic Anchor

2.3.1 Overview

The basic anchor type is the simplest anchor available. It takes the global options and keyframe actions and uses them exactly as specified with no modification or transformation.

2.3.2 Examples

```
<!-- An anchor centered on the effect origin -->
<anchor type="basic" name="origin">
      <dir>none</dir>
</anchor>


<!-- An anchor centered on the effect target -->
<anchor type="basic" name="target">
      <dir>none</dir>
      <keyFrame time="0">
            <action name="totarget" z="1" />
      </keyFrame>
</anchor>

<!-- An anchor that moves from the effect origin to the effect target over the
period of one second-->
<anchor type="basic" name="origin_to_target">
      <dir>none</dir>
      <keyFrame time="0">
            <action name="totarget" z="0" />
      </keyFrame>
      <keyFrame time="1000">
            <action name="totarget" z="1" />
      </keyFrame>
</anchor>
```

2.4 Spline Anchor

2.4.1 Overview

The spline anchor type works just like the basic anchor type, except all movement will be smoothly interpolated along a spline. Because there are extra calculations required, this is less efficient than a

basic anchor and should only be used when smooth interpolation is explicitly desired.

2.4.2 Examples

```
<!-- An anchor that moves an a parabolic arc similar to throwing a rock from
origin to target-->
<anchor type="spline" name="parabolic_trajectory">
      <dir>none</dir>
      <keyFrame time="0">
            <action name="totarget" y="0" z="0" />
      </keyFrame>
      <keyFrame time="500">
            <action name="totarget" y="0.5" />
      </keyFrame>
      <keyFrame time="1000">
            <action name="totarget" z="1" />
      </keyFrame>
</anchor>
```

2.5 Socket Anchor

2.5.1 Overview

The socket anchor as an anchor that follows an sprcal3d socket. You could use this anchor to anchor a flame to the hand of a spell caster for example.

2.5.2 Examples

```
<!-- An anchor that attaches to the left hand of the sprcal3d this effect is
attached to -->
<anchor type="socket" name="left_hand" socket="lefthand">
      <dir>none</dir>
</anchor>
```

3 Effect Objs

Everything that you can see or hear in an effect is done through an effect obj. Each effect obj is attached to a single effect anchor that dictates its base position. The basic format for an effect obj is as follows:

```
<obj type="obj_type" name="obj_name">
      <birth>100</birth>
      <death>2000</death>
      <attach>anchor_attachment</attach>
      <priority>render_priority</priority>
      <z_func_definition />
      <mixmode>mixmode_type</mixmode>
      <dir>dir_type</dir>
      <keyFrame time="0">
            <action name="action_name" action values go here />
      </keyFrame>
      <keyFrame time="1000">
            ...
```

```
    </keyFrame>
</obj>
```

3.1 Global Obj Options

3.1.1 birth

The birth option controls when the effect obj will be born. When the obj is born, its keyframe animations will begin as if the obj had been alive for the entire life of the effect. For example, if the obj has a birth of 1000 and there is a keyframe at time=”1000” then after 1 second of the effect's creation, the obj will become visible and inherit the actions as specified in the keyframe with time=”1000” and not the keyframe at time=”0”.

3.1.2 death

Death controls the time when this effect obj will die. If the death is specified as “none”, then this obj will live forever until the effect is specifically killed. Immortal objs should be used only when you're sure that the obj should live forever. THERE MUST ALWAYS BE A DEATH in a spell effect, because spell effects rely on all objs killing themselves.

3.1.3 attach

This specifies the effect anchor that this obj is attached to.

3.1.4 priority

This dictates when the effect obj will be rendered. The only thing you should know about this is that it should be omitted unless you're using alpha values (which you should avoid), and in the case of using alphas you should set priority to “alpha”.

3.1.5 z_func

Specifies how this effect obj will make use of the z-buffer. Possible tags are: <znone />, <zfill />, <zuse />, <ztest />. Just omit this unless you know what a z-buffer is.

3.1.6 mixmode

Specifies how the effect obj is mixed with its background. Possible values include:

- **copy** No blending with the background is done.
- **mult** The effect obj is multiplied with its background. White color values will be invisible, and dark values will darken.
- **mult2** The effect obj is multiplied with its background and also by 2. This is similar to multiply, but it will turn out brighter and more suitable for spell effects
- **alpha** Rely completely on the alpha value of the effect obj to govern transparency.
- **transparent** AAR TODO: Figure out what this mode is for

- **destalphaadd** AAR TODO: Figure out what this mode is for
- **srcalphaadd** AAR TODO: Figure out what this mode is for
- **premultalpha** AAR TODO: Figure out what this mode is for

3.1.7 dir

This is an option that remains constant across all keyframes in an obj.

The dir dictates the orientation of the position defined by the keyframes.

Possible dir_type values:

- **target** specifies that the direction should be the direction that the target is facing
- **origin** specifies that the direction should be in the direction that the origin is facing (ie, maybe a casting effect that shoots a beam from the player's chest or something)
- **none** no direction for the anchor; with no direction any position's x, y, and z values will remain untransformed.
- **totarget** specifies that the direction should be the direction from the origin to target
- **camera** specifies a direction from the anchor position to the camera position
- **billboard** similar to camera, but specifies a billboard transformation (the orientation will always be in the inverse of the camera's forward direction)

3.2 Obj KeyFrame Actions

3.2.1 scale

This specifies the overall scaling factor of the obj. A value of 1 will leave the obj in its original size, and 10 will make the obj 10 times its original size.

Available attribute values: value

3.2.2 volume

Volume specifies the volume of an audio obj.

Available attribute values: value

3.2.3 topscale

Specifies the scaling for the top of the obj. It can be used to expend or shrink the top of the obj to control the coned shape. This is only used for certain effect objs.

Available attribute values: value

3.2.4 position

Specifies the position offset from the obj's attached anchor.

Available attribute values: x, y, z

3.2.5 rotate

Specifies the rotation around the obj's attached anchor. This will rotate using the anchor as the origin of rotation.

Available attribute values: x, y, z

3.2.6 spin

Specifies the rotation around the obj's own center. Even if you offset the position of the obj, the spin value will still rotate around the obj's own center.

Available attribute values: x, y, z

3.2.7 colour

Specifies the colour of the obj.

Available attribute values: r, g, b, a

3.2.8 height

Specifies the vertical scale of the obj.

Available attribute values: value

3.2.9 padding

This is used by some obj types to add padding between components of the obj.

Available attribute values: value

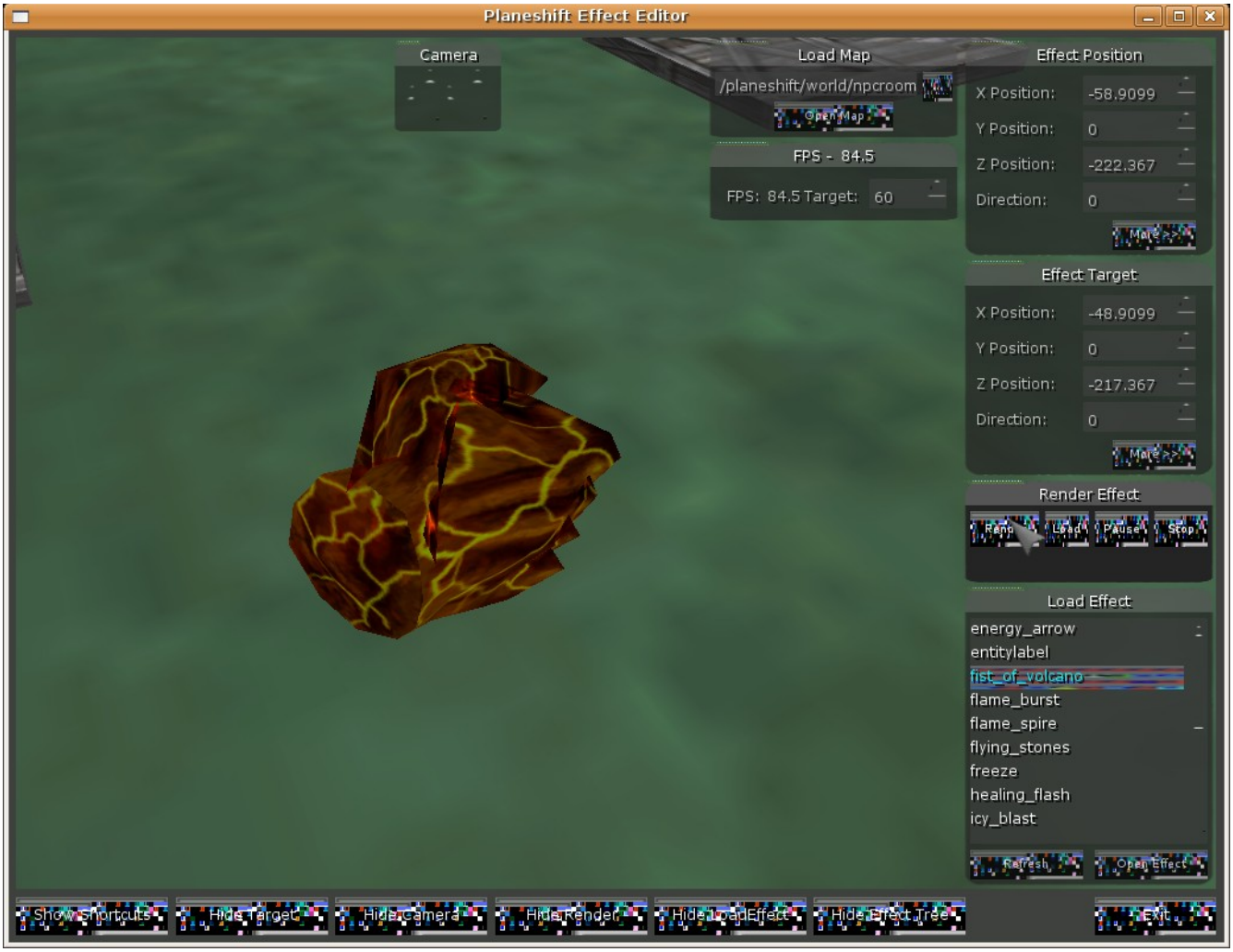
3.2.10 animate

I think this is used for particle systems to control whether it's emitting particles or something.

Available attribute values: value=”yes” or value=”no”

// AAR TODO: research this.

3.3 Mesh Obj



The fist in fist_of_volcano is a mesh obj.

3.3.1 Overview

The mesh obj is a general effect obj that allows you to add any CS mesh to an effect

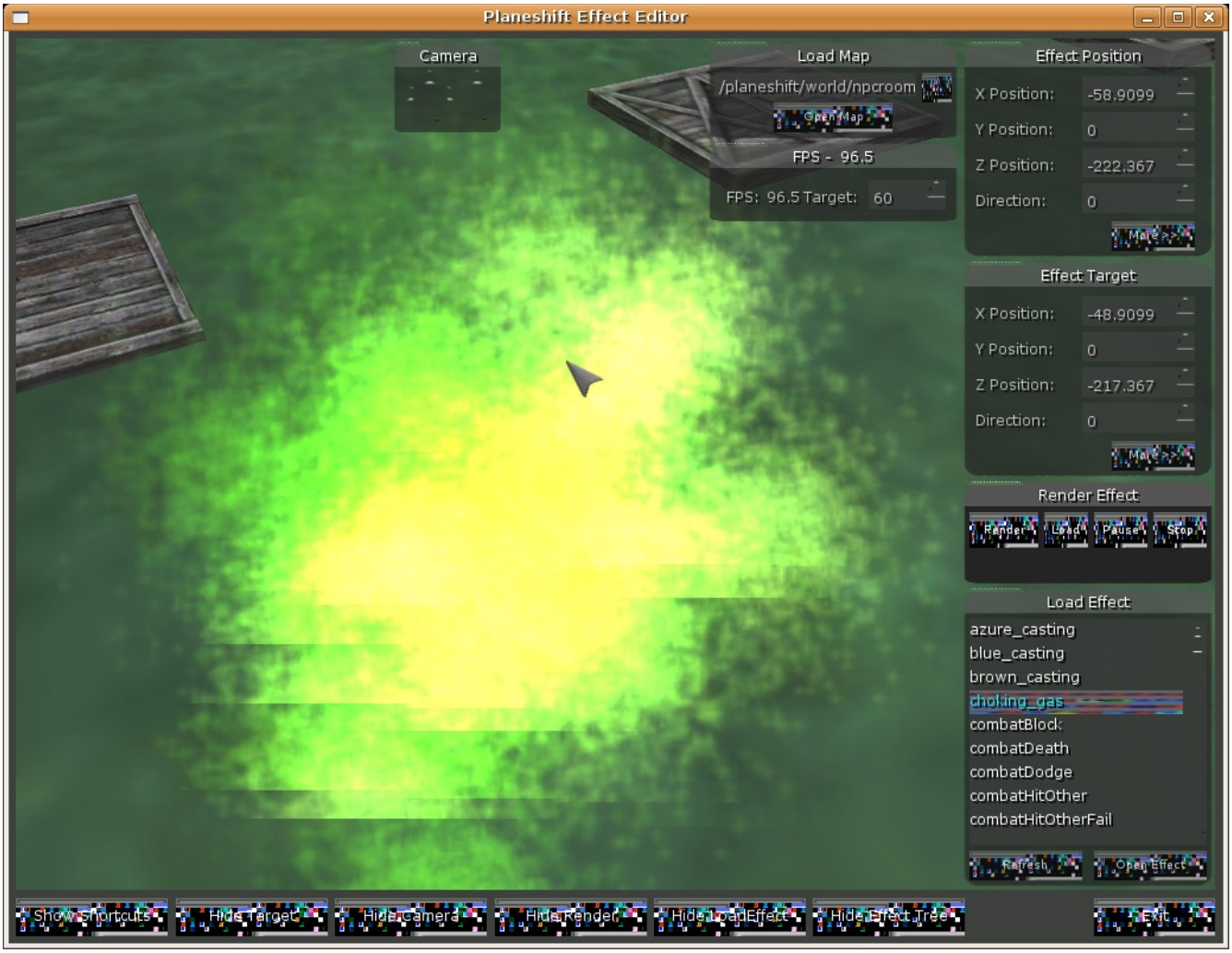
To create a mesh obj, place the mesh in its own library file, reference it at the top of your effect file using the <library>/path/to/library/file</library> tags, and add the mesh obj as you would any other effect obj with the addition of a fact=”name of your mesh's factory” attribute.

3.3.2 Example

```
<obj type=”mesh” name=”fist” fact=”fist”>
      <attach>pos</attach>
      <death>23000</death>
      <mixmode>mult2</mixmode>
      <dir>totarget</dir>
      <keyFrame time=”0”>
            <action name=”position” z=”0.7” />
```

```
      </keyFrame>
</obj>
```

3.4 Particles Obj



The green gas in choking_gas uses a particles obj.

3.4.1 Overview

The particles obj is a way to add any particle system to your effect.

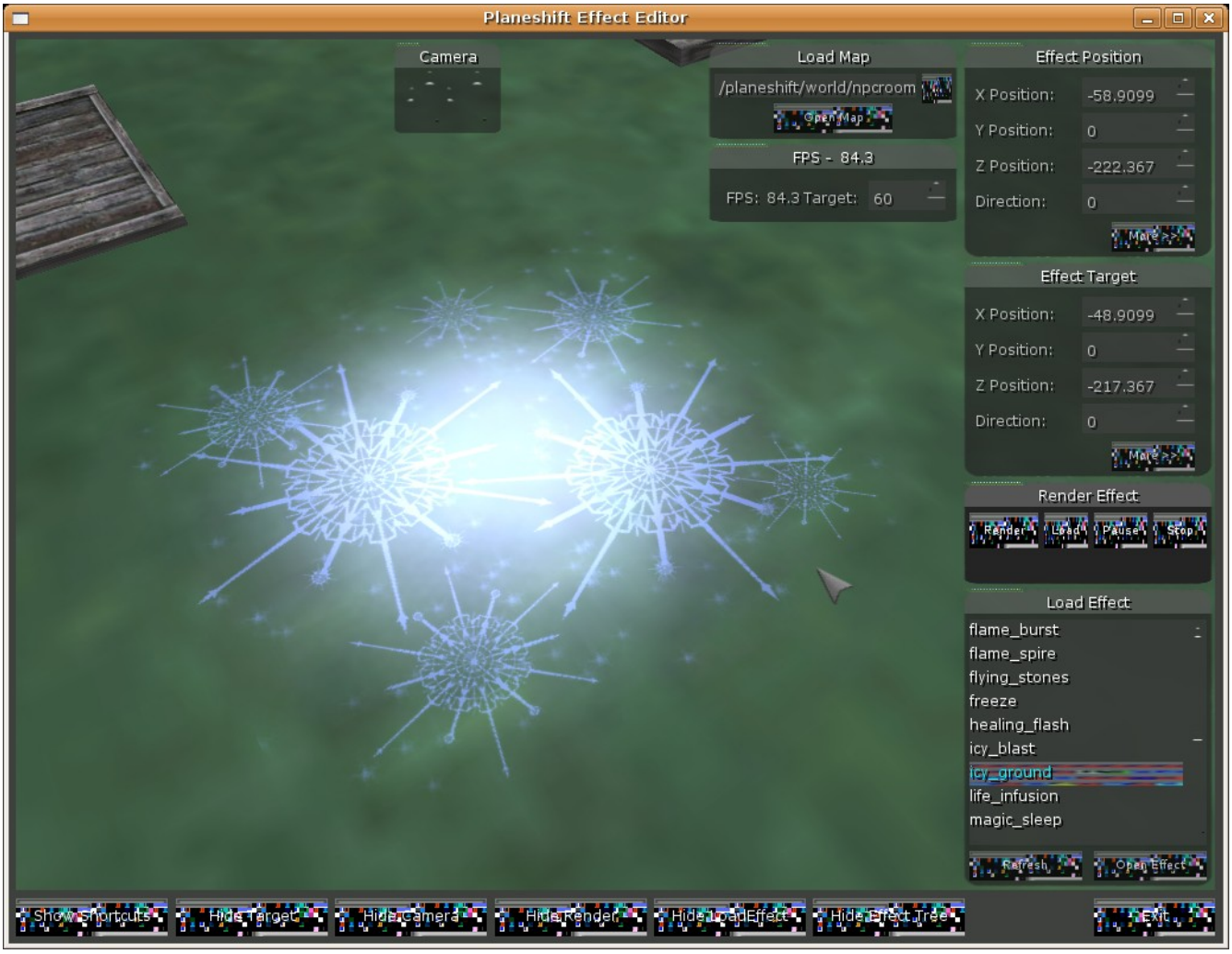
To create a particles obj, place the create a library file containing your particles mesh, reference it at the top of your effect file using the <library>/path/to/library/file</library> tags, and add the particles obj as you would any other effect obj with the addition of a fact=”name of your particles's factory” attribute.

3.4.2 Example

```
<obj type="particles" name="gas" fact="gas">
      <attach>target</attach>
      <death>35000</death>
      <keyFrame time="0">
```

```
            <action name="animate" value="true" />
      </keyFrame>
      <keyFrame time="30000">
            <action name="animate" value="false" />
      </keyFrame>
</obj>
```

3.5 Quad Obj



The ice crystals in icy_ground are displayed using a quad obj.

3.5.1 Overview

The quad obj adds a flat, textured 3D square to your effect. The only dependency of a quad obj is the need for a material; otherwise, there are no mesh or particle system dependencies.

3.5.2 Example

```
<obj type="quad" name="icyfloor" material="dot">
      <attach>pos</attach>
      <death>30000</death>
      <dir>none</dir>
```

```
      <keyFrame time="0">
            <action name="alpha" value="0" />
            <action name="position" x="0" y="0.1" z="0" />
            <action name="scale" value="6" />
            <action name="height" value="1.0" />
            <action name="colour" r="160" g="160" b="230" />
      </keyFrame>
      <keyFrame time="28900">
            <action name="colour" r="160" g="160" b="230" />
      </keyFrame>
      <keyFrame time="29900">
            <action name="colour" r="0" g="0" b="0" />
      </keyFrame>
</obj>
```

3.6 SimpMesh Obj

I believe this obj type is deprecated and should not be used. Use the Mesh Obj instead.

3.7 Sound Obj

3.7.1 Overview

The Sound Obj adds sound effects to an effect. The sound file needs to be a sound resource defined in the PS Sound Manager.

The Sound Obj has the following extra tags:

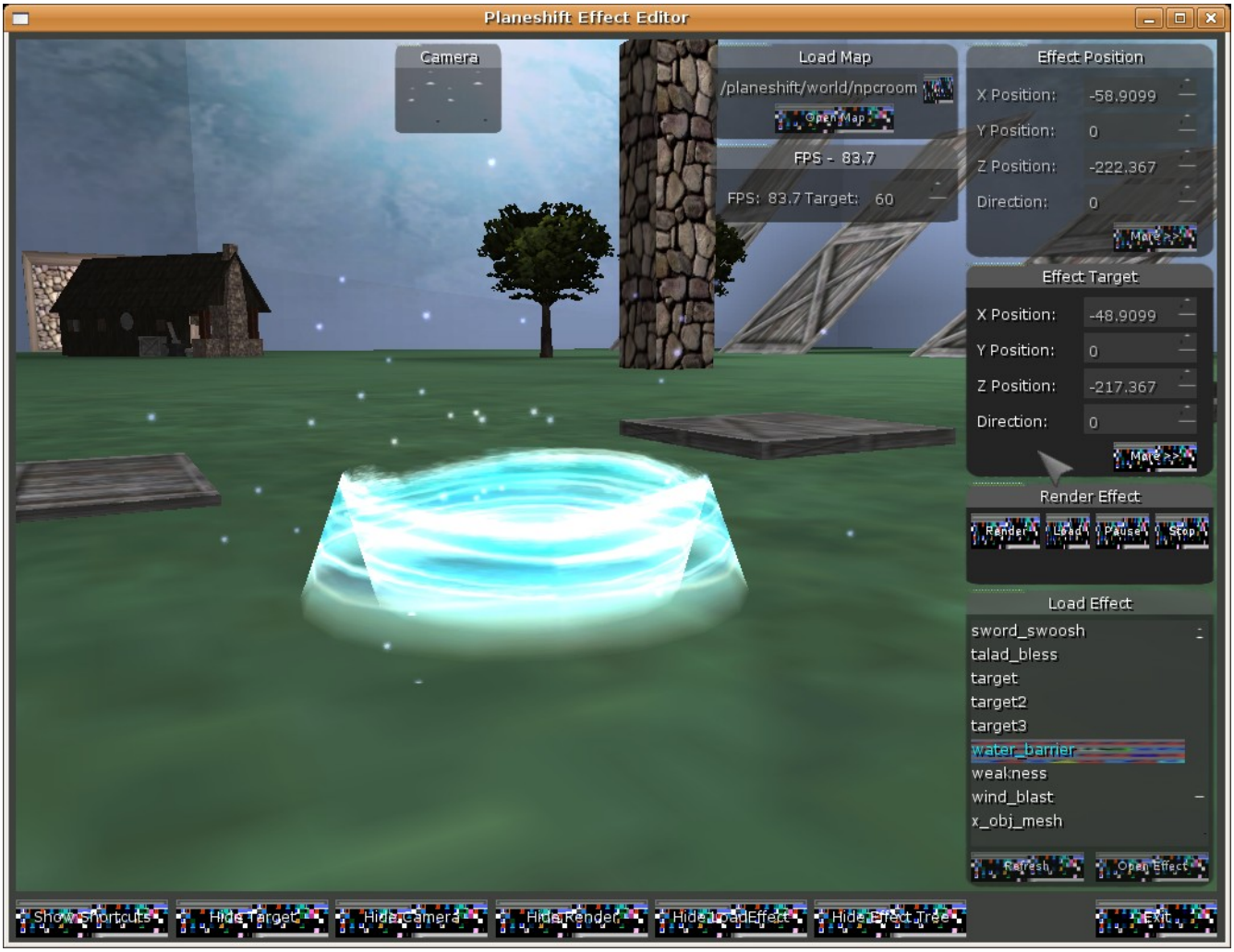
- **mindist** if the camera is within this radius of the sound's position, then the sound will be at full volume (as specified by the volume action)
- **maxdist** if the camera is outside this radius from the sound's position, then the sound will be too distant to be heard.
- **loop** if true, this sound will loop back to the beginning once it has finished
- **resource** the name of the sound resource to play as specified in the PS Sound Manager

3.7.2 Example

```
<obj type="sound" name="Water_Barrier_Loop" resource="water" loop=”true”>
      <attach>pos</attach>
      <birth>50</birth>
      <death>120000</death>
      <mindist>2</mindist>
      <maxdist>30</maxdist>
      <keyFrame time="0">
            <action  name="volume" value="0.1" />
      </keyFrame>
      <keyFrame time="700">
            <action  name="volume" value="0.8" />
      </keyFrame>
      <keyFrame time="110000">
            <action  name="volume" value="0.6" />
```

```
      </keyFrame>
      <keyFrame time="120000">
            <action  name="volume" value="0.1" />
      </keyFrame>
</obj>
```

3.8 Spire Obj



The waves of water in water_barrier use a Spire Obj.

3.8.1 Overview

The Spire Obj adds a vertical spire to an effect. The only dependency of a spire obj is the need for a material; otherwise, there are no mesh or particle system dependencies.

The Spire Obj has the following extra tags:

- **shape** dictates the shape of the spire as seen from above. This can be one of circle, asterix, star, and layered.
- **segments** specifies the number of sides for the spire. A higher number of segments will make for a rounder spire, but will be less efficient.

3.8.2 Example

```
<obj type="spire" name="river" material="wave" shape=”circle” segments="20">
      <attach>pos</attach>
      <dir>none</dir>
      <death>120000</death>
      <keyFrame time="0">
            <action name="scale" value="0.67" />
            <action name="height" value="0.75" />
            <action name="topscale" value="1.2" />
      </keyFrame>
      <keyFrame time="119900">
            <action name="spin" x="0" y="64800" z="0" />
      </keyFrame>
      <keyFrame time="120000">
            <action name="spin" x="0" y="0" z="0" />
      </keyFrame>
</obj>
```

3.9 Star Obj

No effects use this yet.

3.9.1 Overview

The intent of the Star Obj is to shoot out beams of light in random directions. Despite my horrible art in the screenshot, I'm convinced it will look good in an effect with the right art touch.

Like the Quad Obj and Spire Obj, the Star Obj is only dependent on being given a material; no mesh or particles object is required.

The Star Obj has the following extra tags:

- **segments** the number of beams to project outwards.

3.9.2 Example

```
<obj type="star" name="star_typical" material="star_example" segments="15">
      <attach>pos</attach>
      <death>none</death>
      <keyFrame time="0">
            <action name="colour" r="255" g="255" b="0" a="255" />
            <action name="position" y="3" />
            <action name="height" value="20" />
            <action name="topscale" value="1" />
      </keyFrame>
</obj>
```

3.10 Text Obj

Entity Labels in game use the Text Obj.

3.10.1 Overview

The Text Obj provides a means of displaying text. It works like the Quad Obj, except uses given text as a material.

The Text Obj has the following extra tags:

- **static** set this to true if the text of this material will not be modified in code. This is usually true, except for special cases (such as entity labels) where the text must be changed in the game.
- **antialias** set this to true or false to control antialiasing for the rendered text.

The Text Obj uses a set of tags to dictate display behaviour. The below example attempts to demonstrate them all.

3.10.2 Example

```
<obj type="text" name="text">
      <attach>pos</attach>
      <death>none</death>
      <dir>billboard</dir>
```

```
      <text>
            This is some normal text[br /]
            [colour:00ff00]This is some green text.[/colour][br /]
            [shadow:0000ff]This is some blue shadowed text.[/shadow][br /]
            [left]This is some left-aligned text[/left]
            [right]Right aligned[/right]
            [center]Centered text[/center]
            [outline:00ff00]This is some green outlined text.[/outline][br /]
      </text>
</obj>
```

3.11 Trail Obj

This effect obj creates a trail following current position of the obj. Right now I'm working on this for sword swooshes, but let me work on this a bit more before you use it.

4 Eedit

4.1 Overview

Eedit is planeshift's effect editor. At the moment it provides a way for you to dynamically preview your effects and find errors as you edit/create the effect.

4.2 Usage

Basic usage in creating/editing an effect:

1. Open EEdit
2. If you're creating a new effect, create a .eff file somewhere in your effect directory, and then click “Refresh” in EEdit's effect list to refresh the list of effects.
3. Double click on the desired effect in EEdit's effect list to load it.
4. Click “Render” in the render dialog to display the effect (right now you have to look down from the starting camera position to actually see your effect).
5. To edit your effect, edit the .eff file, then click the “Load” button beside render button to reload the .eff file, and finally click “Render” to display it.

There is much more to eedit that you can play around with, but for now this simple overview will have to do for documentation.