osgFX

A brief and informal introduction for users and developers

Marco Jez
September 2003

OpenSceneGraph

Introduction to osgFX

What is osgFX?

- osgFX is:
  - An add-on library for OpenSceneGraph
  - A framework for implementing consistent, self-contained, reusable effects that can be applied to OSG nodes
  - A (small) set of predefined special effects

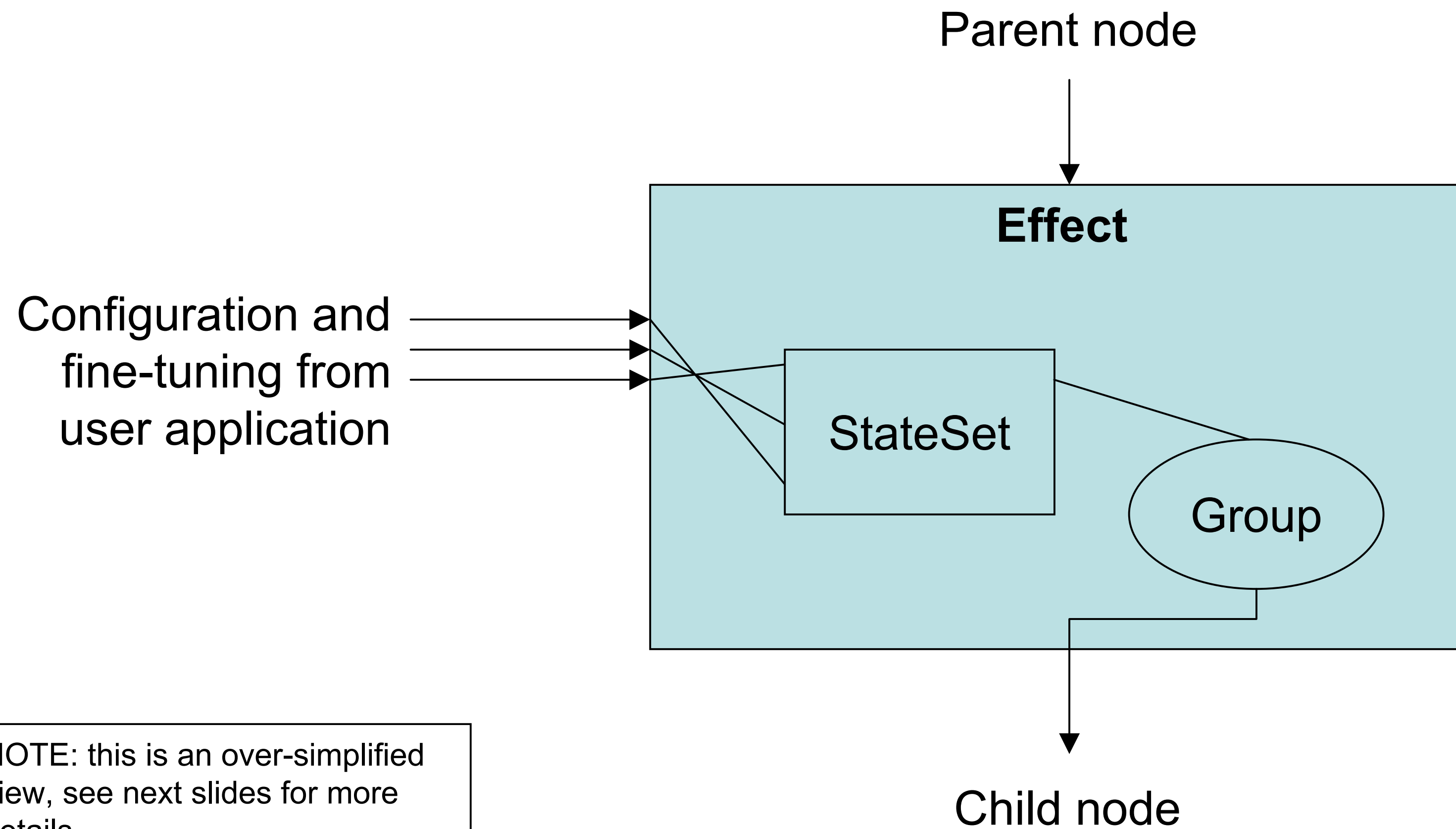
What are effects?

- With term “effect” we refer to a set of visual attributes and behaviors “packed up” in a single object. An effect class may expose a public interface to allow fine-tuning and configuration while staying focused on the effect purpose.
- An effect can be thought as a “bridge” between the problem space (how an object should look like) and the solution space (what state attributes and other properties should be set up).

What are effects? (2)

- From the C++ point of view an effect is an instance of class osgFX::Effect. Well, actually it is an instance of one of its derived classes, because osgFX::Effect inherits directly from osg::Node, so it is abstract.
- From the OSG point of view, an effect is a Node. It behaves like any other node classes and you can attach it anywhere in your scene graph.

Effect functional diagram



How do effects interact with the scene graph?

- The Effect class is a single-child group. It has a method called setChild() which attachs a node to itself. The difference between osgFX::Effect and osg::Group is that Effect can not have multiple children.
- The visual attributes defined within an effect are applied to the child node, just like a Transform node applies a coordinate transform to its children. Effect attributes don't propagate outside the child node.

Using osgFX

How can I apply my favorite effect to existing nodes?

- If you want to apply an effect to a subgraph, follow these steps:
  - Create an instance of the desired effect, for example osgFX::Scribe
  - If necessary, configure the effect using its interface methods
  - Attach your subgraph to the effect node by calling Effect::setChild()
  - Attach the effect node to the scene graph

How can I apply my favorite effect to existing nodes? (example)

- A simple example using the Scribe effect:
  ```
  osg::ref_ptr<osg::Node> my_node =
          osgDB::readNodeFile(“cow.osg”);
  osg::ref_ptr<osgFX::Scribe> scribe_fx =
          new osgFX::Scribe;
  scribe_fx->setChild(my_node.get());
  root->addChild(scribe_fx.get());
  ```
- Next slide shows what you get with the above code

How can I apply my favorite effect to existing nodes? (screenshot)

Getting deeper: Techniques and Passes

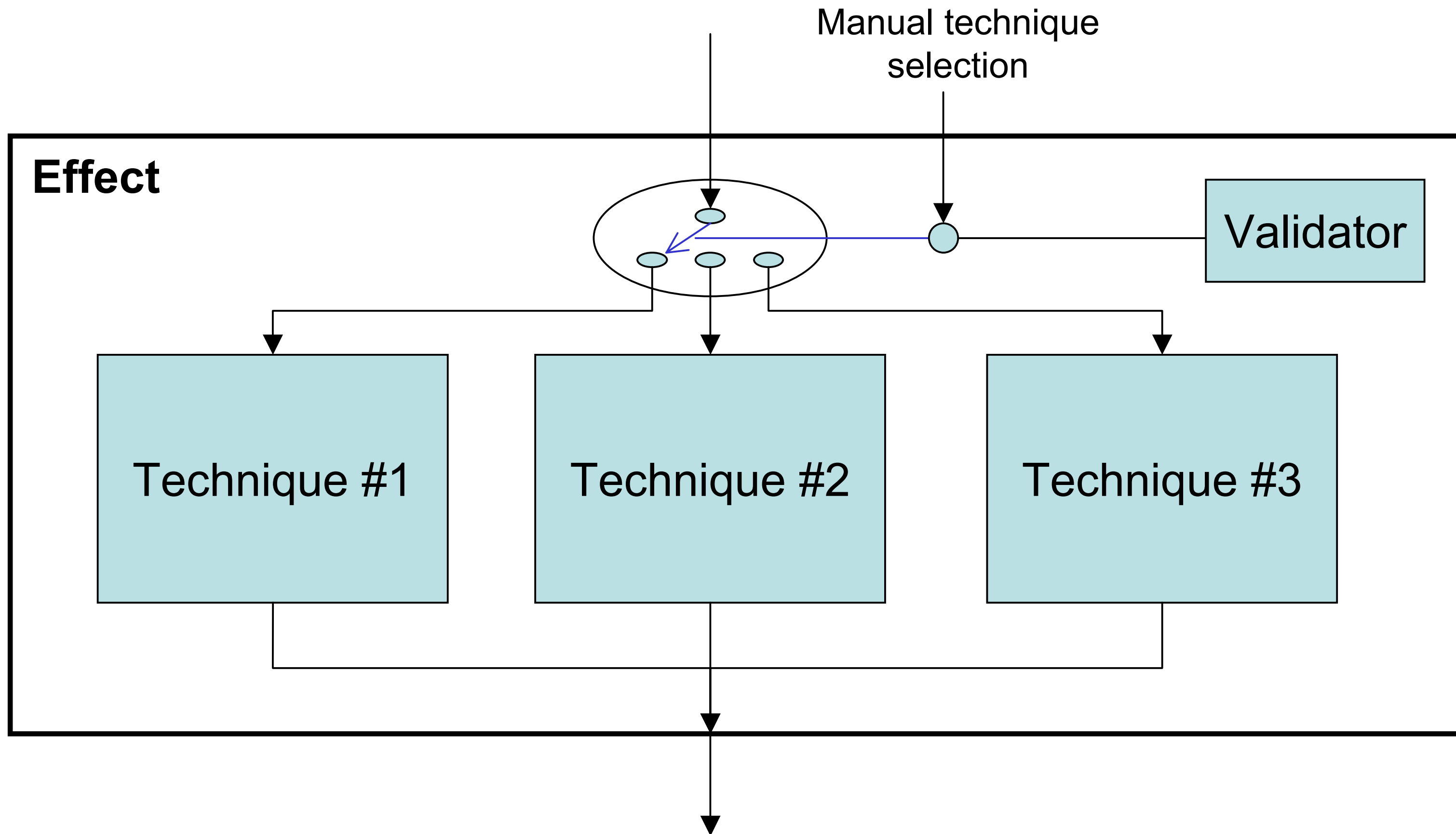
What is a “technique”?

- A technique is one of the possible methods that can be used to implement an effect.
- The wide variety of graphics hardware and OpenGL extensions makes it nearly impossible to implement a complex effect in a “standard” way: you may need different implementations for different types of hardware and OpenGL environments.
- An effect may contain one or more techniques, each one trying to implement the same effect in a different fashion.

Choosing techniques

- By default the Effect class uses a custom (private) StateAttribute object to perform run-time validation of techniques, then it chooses the best one
- The effect developer defines the techniques in decreasing priority order, so that preferred techniques are validated first
- The Effect class chooses the highest priority technique that could be validated in all active rendering contexts at run-time.
- The user can override this default behavior anytime if he wants to

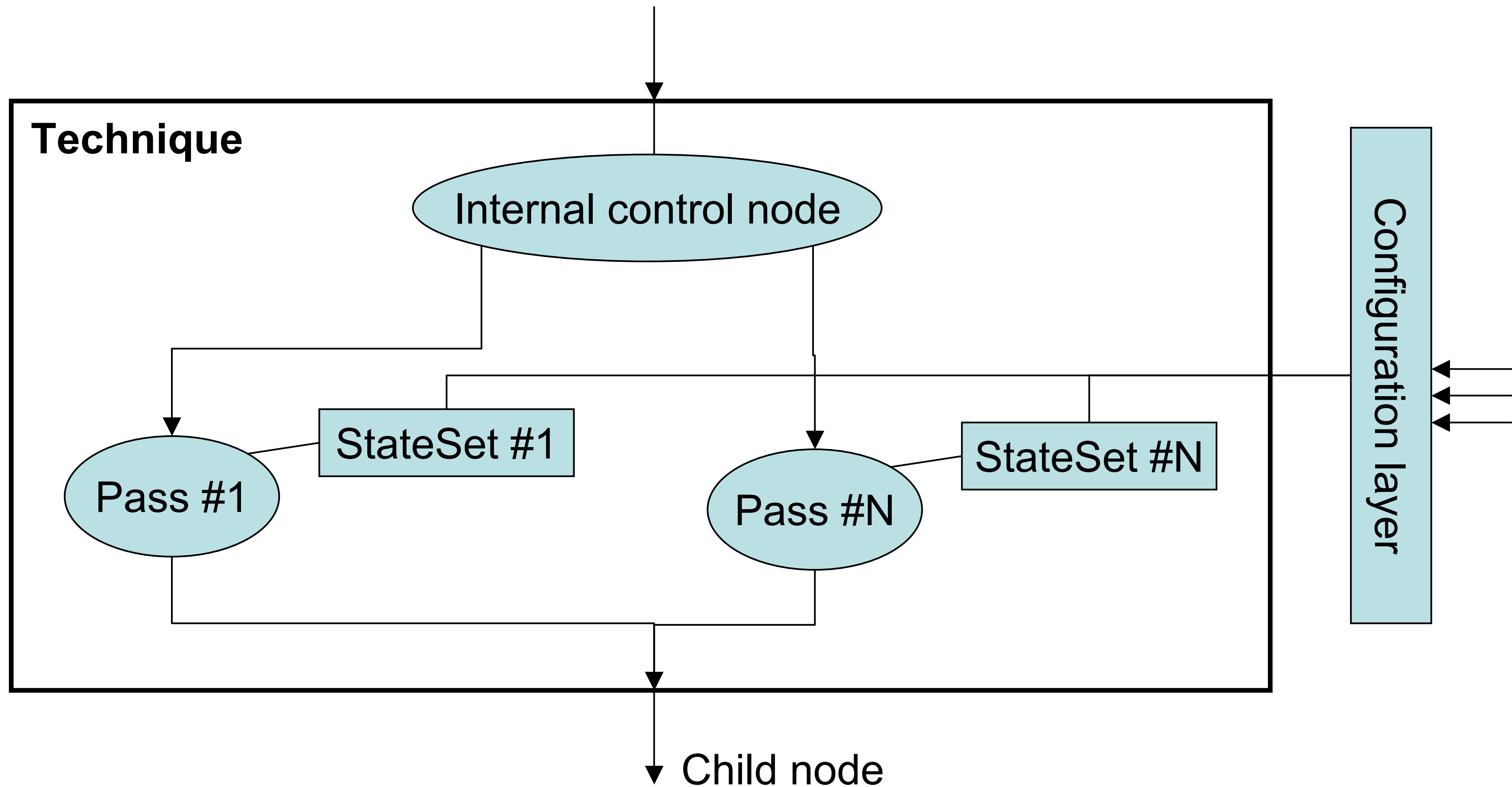
Effect with techniques functional diagram



What is a “pass”?

- Multi-pass rendering means drawing the same object multiple times using a different set of visual attributes each time to get a final image which is the combination of all passes
- A technique may require more than one pass to produce the desired output
- Technique classes create one StateSet object for each rendering pass, then osgFX manages multi-pass rendering transparently

Technique with passes functional diagram



Extending osgFX

Basic steps to create a new effect

- An effect is a class derived from osgFX::Effect, so you need to create your own subclass, let's call it TestFx
- Implement abstract methods like effectName() and effectDescription(), possibly using the META_Effect macro
- Register the effect class by creating a static instance of a registry proxy:
  - osgFX::Registry::Proxy proxy(new TestFx);
- Implement the protected abstract method define_techniques() to create the required effect techniques (see next slide)

Implementing techniques

- For each technique to be implemented you must write a class derived from osgFX::Technique; such class should be private
- In your effect class' define_techniques() method you have to create instances of the above technique classes and add them to the effect by calling Effect::addTechnique() in decreasing priority order

Implementing techniques (continued)

- Provide a validation point for the newly-created techniques. The easiest (but less flexible) way to do that is to override the Technique::getRequiredExtensions() method and specify the OpenGL extensions that are required for that technique.
- Implement the Technique::define_passes() method to create the rendering passes (see next slide)

Defining rendering passes

- A rendering pass is internally implemented as a Group with an associated StateSet. The effect's child node will be automatically added to pass groups at run-time.
- The technique's define_passes() method defines rendering passes by creating one StateSet object for each pass and adding it to the technique by calling Technique::addPass(). The pass group will be automatically generated and linked to the state set.

Creating effects: schematic view

Class TestFX (public)

```
META_Effect(…….);

bool define_techniques()
{
  addTechnique(new FirstTechnique);
  // add other techniques if necessary
}
```

Class FirstTechnique (private)

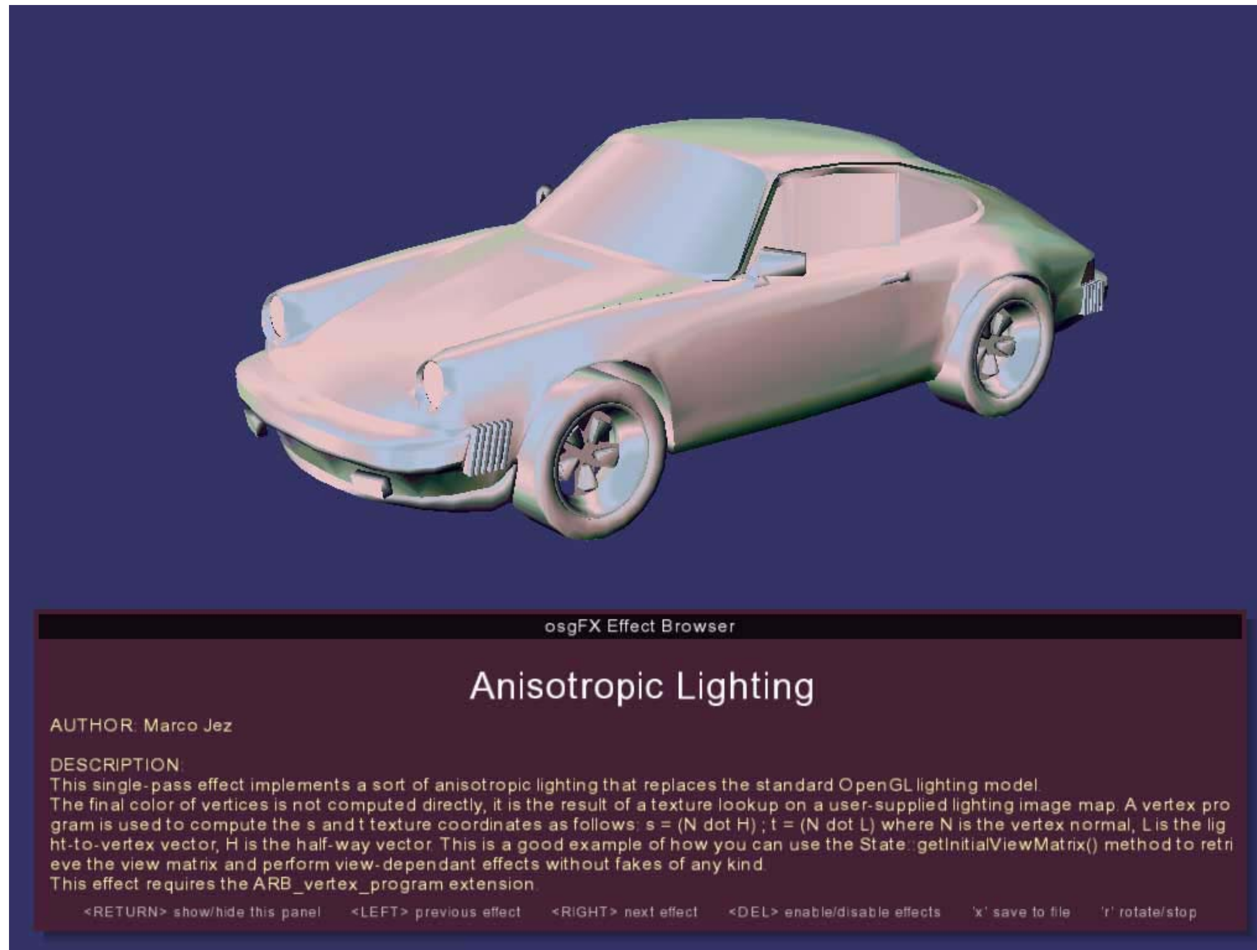
```
void getRequiredExtensions(…….) const
{
  // specify which GL extensions are required
}

void define_passes()
{
  osg::ref_ptr<osg::StateSet> ss1 = new osg::StateSet;
  // add attributes to the state set ss1
  addPass(ss1.get());

  osg::ref_ptr<osg::StateSet> ss2 = new osg::StateSet;
  // add attributes to the state set ss2
  addPass(ss2.get());
}
```

Creating effects: summary

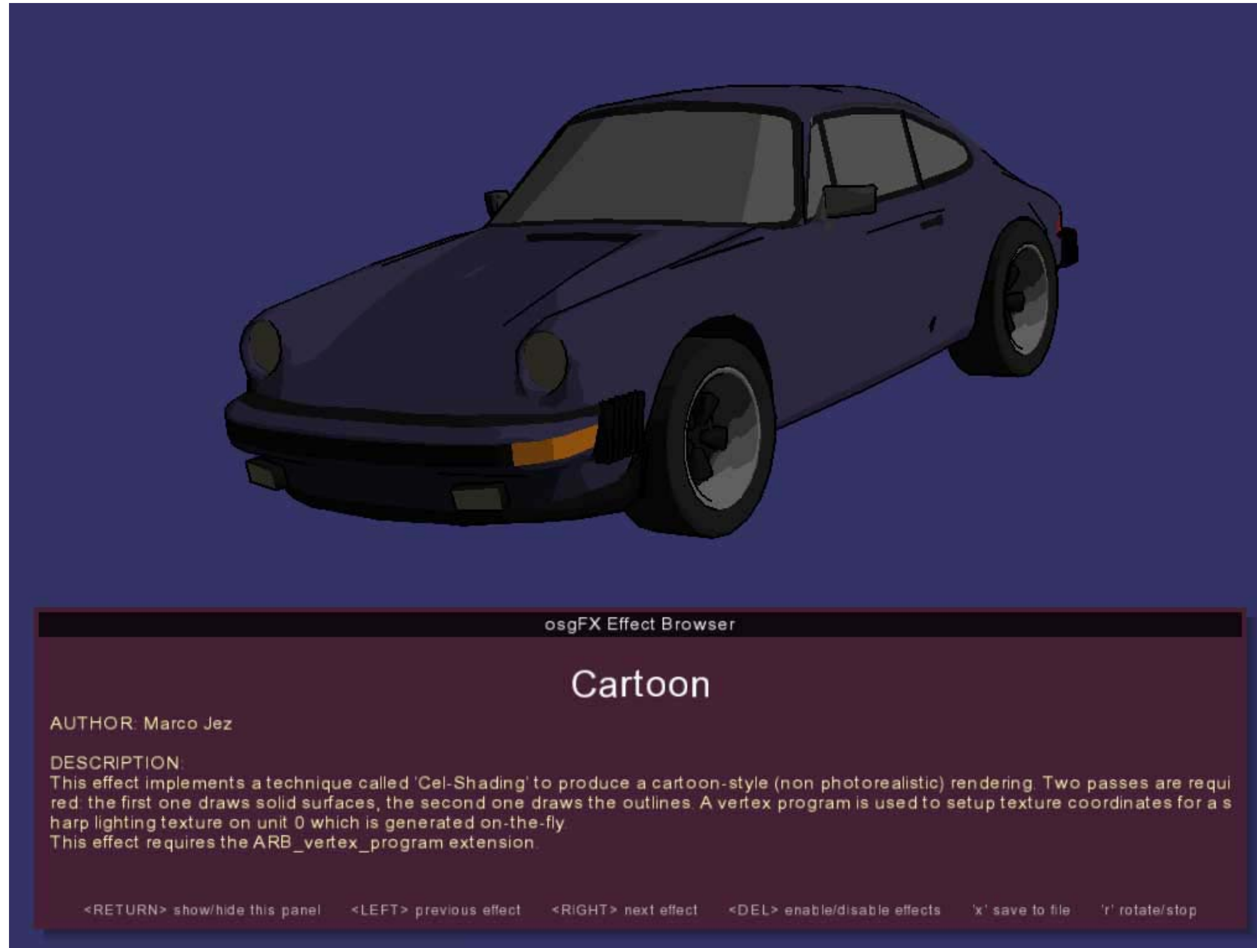
1. Create a class (for example TestFx) derived from osgFX::Effect, fill it with informations like name and description, then register it with a registry proxy
2. Create a (private) class for each technique you want to implement and define a validation behavior for all of them
3. In TestFx::define_techniques() create instances of technique classes and add them to the effect by calling addTechnique()
4. In each technique class' define_passes() method create one or more StateSet objects (one for each rendering pass) and add them to the technique by calling addPass()

Screenshots from osgfxbrowser

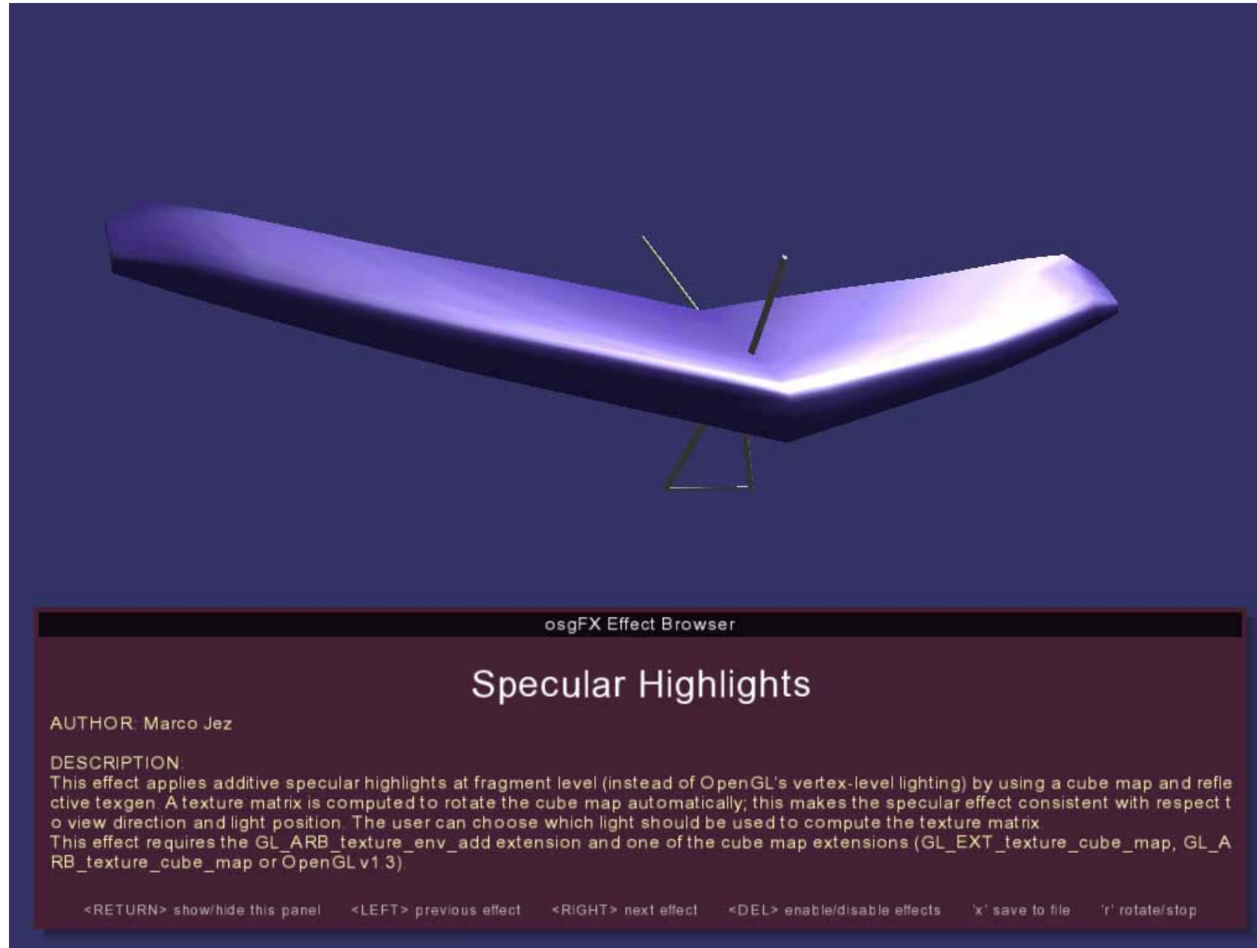
Anisotropic Lighting



Cartoon (cel shading)



Cubemap-based Specular Highlights



Bump Mapping

Questions?

- The OpenSceneGraph mailing list is there for you
  - visit http://www.openscenegraph.org
- If you need more help you can contact the author directly:
  - Marco Jez <marco.jez@arsenal.it>
  - Please try the mailing list first! ☺