NewAge Shader Language v0.5 Specification

Christopher Olsen
http://www.upl.cs.wisc.edu/~colsen/
colsen@cs.wisc.edu

September 30th, 2004

Contents

1 Introduction

The NewAge Shader Language is a high level language that can be used with any graphics API. I will assume the use of OpenGL throughout this specification. NASL allows complex materials to be easily defined and loaded from shader files external to an application.

1.1 Terminology

1.1.1 Shaders

A shader contains all textures, colors, operations, and tests for a particular material. When a shader is "bound" these textures, operations, tests, etc. are set in OpenGL. In short a shader is an encapsulation of the OpenGL textures and state of a material. A shader contains only three things: A name, a list of shader operations, and a list of stages.

1.1.2 Stages

Each stage in a shader corresponds to a texture stage on the graphics hardware. It is a layer where various properties can be set. When the surface is rendered all the layers are combined together to produce the final look of the material. The number of possible stages per shader is dependant on the graphics hardware. Here is a list of current hardware and it's capabilities:

```
Geforce 1 or 2:  2 stages
Geforce 3 or 4:  4 stages
Geforce FX:      8 stages
radeon9800:      8 stages
Geforce 6800:    16 stages
Radeon X800:     16 stages
```

A multi-pass implementation for rendering shaders with more stages than the graphics hardware can handle in one pass may be a part of a future version of NASL.

1.2 History of NASL

NASL v0.1 (simply called shader) was first developed for a game engine called Xerosis I was working on in my spare time. It was modelled after Quake 3 shader files. Eventually it got to a working state and was incorporated into Age, a game engine I wrote as part of a group for a Games Technology course at the University of Wisconsin. This version of NASL was quite crippled (little more than two stages and tcmods) and horribly designed due to the time constraints of the class, student life, and myriad other things to be done to finish the game engine.

NASL v0.2 was a complete rewrite of NASL v0.1 and was part of a complete rewrite of the Age engine, cleverly titled NewAge (hence, NewAge Shading Language). With the benefit of a total rewrite NASL v0.2 was much better designed and more powerful than it's predecessor. Everything I wanted in a shading language was there.

NASL v0.3 was a minor update to NASL v0.2. With NASL v0.3 texture and

shader management were incorporated along with NASL into a convenient library that could be easily added to any OpenGL project instantly. Cg vertex shaders were also added to the language.

NASL v0.4 is mostly a rewrite of the parsing code, using a more canonical way of parsing the shader files, making it easier to add new operations to the NASL language. NASL v0.4 also simplifies the language a bit, getting rid of redundant keywords and OpenGL keywords in the language. Additions include the agen operator, and support for setting the textures directory and default shader file for NASL.

NASL v0.5 has completely dropped the texture and shader management in favor of a more simplified definition language that can be used to create shaders in any API.

2 The NASL Language

2.1 Syntax

Shaders are defined with the following syntax:

```
name
{
  <shaderOps>

  {
    <stage1 properties>
  }

  {
    <stage2 properties>
  }
  .
  .
  .
}
```

Nearly every operation in NASL is optional, IE they will use a defined default if the operation is not used in the shader.

2.2 Shader Operations

2.2.1 alphaTest

```
Fragments that fail the alpha test are not rendered.

alphaTest <func> <refValue>

refValue is a value against which the fragment alpha value is tested

func is one of the following:
gt
gte
lt
lte
```

```
eq
neq
never
always

default: alphaTest always 0.0
```

2.2.2 blendFunc

```
The blendFunc operation defines how this surface blends with the frame
buffer.

blendFunc <srcBlend> <dstBlend> <blendEquation>

where srcBlend and dstBlend can be any one of the following:
zero
one
src_alpha
one_minus_src_alpha
src_color
one_minus_src_color
dst_alpha
one_minus_dst_alpha
dst_color
one_minus_dst_color

...and the blendEquation can be:
add
subtract
revsubtract

default: blendFunc one zero add
```

2.3 Stage Operations

2.3.1 combiner

```
The combiner operation defines how this stage combines with the stage
below it.

combiner <replace | modulate | add | subtract | dot3>

default: combiner modulate
```

2.3.2 texmap

```
A 24 or 32 bit TGA file can be used as a stage texture.

texmap <textureName>
```

2.3.3 wrapMode

```
The wrap mode defines how a texture is rendered beyond the range
[0,1].
```

```
wrapMode <clamp | repeat>

default: wrapMode repeat
```

2.3.4 filterMode

```
The filter mode defines how a texture is filtered, whether bilinear or
nearest neighbor.

filterMode <nearest | linear>

default: linear
```