Using Hardware Shaders in VTK: A small tutorial

By: Harry Seip, Andy Man
Delft University of Technology, The Netherlands

1 Introduction

This small tutorial provides a step-by step introduction of the use of hardware shaders in combination with VTK. It was written as a part of a B.Sc. project at the Delft University of Technology. We have chosen Cg as the shading language and C++ as the developing for our tutorial since we have the most experience with this combination. VTK also supports GLSL as a shading language but the results of this tutorial should be easily extended to GLSL and to the other languages that VTK supports. The version of VTK that we used was a CVS checkout of November 11, 2005, built with Cg support. Finally we tested our shaders with a Nvidia Geforce 5200FX.
In the next section we will take a quick look at our shading language. Then we will shortly discuss linking a Cg program with a vtk object. In section four we will discuss the core subject of this tutorial, the xmlMaterialfile. Finally we will give some reccomendations in section five.

2 Cg in combination of VTK

Our shaders are written in the language Cg which offers a high-level programming environment of developing shaders. The shaders in Cg are then linked with the object being shaded via the XMLMaterial file which will be discussed in the next section.
In our version of VTK there were several shaders included for testing and demonstration. We also reviewed Cg shaders discussed in tutorials of the Cg user manual. For excellent resources concerning Cg programming we refer to Nvidia's developers website: developer.nvidia.com.

3 Linking VTK with Cg

In this section we will look at the XMLMaterial file that is used to load shaders from within the VTK pipeline. We will discuss the syntax and its semantics. The xml file which we will call the material file from now on is the bridge between the VTK environment and the hardware shaders, in our case written in Cg. The main function of the material file is to locate the shader code, initialize and pass on required parameters and arguments. The code in a C++ program using VTK could be as follows:

```
vtkProperty *prop = anActor->GetProperty();
prop->LoadMaterial("F:\\My Documents\\Vakken\\MsC CG\\minivtk\\Shaders.xml");
prop->ShadingOn();
prop->AddShaderVariable(“Rate”,1, 1.0);
```

We see that vtkProperty is first linked with the shader through the 'LoadMaterial' function and only loads the shader through the 'ShadingOn' function. We can pass on user-defined variables to the shader using "AddShaderVariable". The instantiation and binding of the shader is handled by the vtkShader class and read from the XML file.

4 XML Materialfile Format

The material file is written in standard xml 1.0 format. The material file is accessed by an actor property, so properties and therefore indirectly actors are associated with a given shader. The different tags in the material file are subsequently interpreted by the vtkShader class that acts as a wrapper class for the language specific shader instantiating classes. In our experiment this will be the vtkCgShader class. This class will load, compile and bind the shader to the target actor and uses the information from the material file for all shader specific context. Multiple material files can exist for multiple shaders and/or properties, the only constraint is that only one vertex shader and one fragment shader can be active at the same time due to hardware limitations. We will now

examine the syntax and functions of the different parts within the xml file. We have reverse engineered each attribute from the sample material files and the snippets of documentation found on the internet.

minivtk.cxx | FragmentH.cg | vertex.cg | **Shaders.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Harry Seip 1015729
     Andy Man

     Group 3
     2005 DUT

     XMLMaterial file: required to load cG and/or GLSL shaders in VTK
     Realistic depth and texture shader for bone surface
 -->

<Material name="Mat1" NumberOfProperties="1" NumberOfVertexShaders="1" NumberOfFragmentShaders="1">

  <Shader scope="Vertex" name="vertexGroup3" location="vertex.cg" language="Cg" entry="main" args="-DVERTEX_PROGRAM">
    <MatrixUniform name="ModelViewProj" type="State" number_of_elements="2"
                   value="CG_GL_MODELVIEW_PROJECTION_MATRIX CG_GL_MATRIX_IDENTITY"> </MatrixUniform>
    <MatrixUniform name="ModelViewIT" type="State" number_of_elements="2"
                   value="CG_GL_MODELVIEW_MATRIX CG_GL_MATRIX_INVERSE_TRANSPOSE"> </MatrixUniform>
    <LightUniform name="LightVec" value="Position" />
  </Shader>


  <Shader scope="Fragment" name="fragmentGroup3" location="FragmentH.cg" language="Cg" entry="main" args="-DFRAGMENT_PROGRAM">
    <SamplerUniform type="sampler2D" name="bumpTexture" value="0"> </SamplerUniform>
    <SamplerUniform type="sampler2D" name="detailTexture" value="1"> </SamplerUniform>
    <PropertyUniform  value="AmbientColor" name="ambientColor" > </PropertyUniform>
    <PropertyUniform  value="SpecularColor" name="specularColor" > </PropertyUniform>
    <LightUniform name="lightVector" value="Position" />
  </Shader>

</Material>
```

Figure 5.5 XMLMaterial file

For each tag we found we have listed its function and the attributes of the tag that are possible. For each attribute we have discovered its function, the required input type and if the attribute is optional or not. Optional attributes can be omitted without generating errors. We have compiled this information to the best of our knowledge and have tested its workings where possible but we must note here that these results must be used with reservation since we did not have access to any specifications. We have also seen inconsistencies between various material files, probably caused by the fact that shader functionality for VTK is still in the development phase that could invalidate some results. Lastly we note that the final release version of the specification may still be modified from our working copy.

All shader parameters reside in the material tag, *<Material>*, this tag has some attributes that are listed in the table. Its main function is to structure the material file itself.

<Material>		Defines the material we are loading	
Name:	InputType:	Function:	Optional:
name	String	No apparent function, Identification	Yes
NumberOfProperties	Integer	Sets the number of Properties defined	Yes
NumberOfVertexShaders	Integer	Sets the number of VertexShaders defined	Yes
NumberOfFragmentShaders	Integer	Sets the number of FragmentShaders defined	Yes

The VTK developers have provided a property tag, *<Property>* that can be used to set some elements (members) of the current vtkProperty that has loaded the material file. We can use this to set the ambient, diffuse and specular colours for instance. Note that this can also be done in the vtkpipeline by using the appropriate vtkProperty functions. The semantics here are completely transparent with that method and the only difference is that we are sure of the settings of the property since we are setting them ourselves. The elements of the vtkProperty are accessed with the *<Member>* tag.

The *<Shader>* tag defines a shader and depending on its scope attribute this will be either a vertex shader or a fragment shader. The shader tag also provides a pointer to the Cg file (or GLSL depending on the language used) and acts as a wrapper tag for the various shader parameters. The attributes can be found in the following table:

<Shader> Name:	InputType:	Defines a Shader Function:	Optional:
scope	Enumeration:	To distinguish between Fragment and Vertex shaders	No
name	String	ID's the shader	Yes
location	Enumeration:	Defines the location of the file	No
language	String	Defines shader language (cg or GLSL)	No
entry	String	Defines the entry routine of the shader	No
args	String	specifies command line arguments for the compiler	No

Let's look at some of the attributes used with the *<Shader>* tag. *Scope* has two possible values: "vertex" or "fragment". The location attribute stores the location of the (Cg) code and has three modes: Library, Repository and Pathname. When "Library" or "Repository" is specified as the value this means that the shader code has been compiled with VTK and can then be retrieved using the name attribute. This is useful for basic shaders that are used by many applications. Alternatively a pathname can be supplied that points to the Cg file. We have used the latter option for our shaders.

The remaining tags are used to provide the shader with the parameters it needs. We have found the following parameter tags: *<Uniform>* for constant parameters*, <ApplicationUniform>* for runtime parameters*, <CameraUniform>* for parameters coming from a vtkCamera instance like *Eyeposition, <PropertyUniform>* for parameters stemming from a vtkProperty instance like *Specularcolor, <LightUniform>* for parameters stemming from a vtkLight instance like *Lightposition, <MatrixUniform>* for matrix parameters like *ModelviewIT* and *<SamplerUniform>* for texture parameters*.* We will discuss one of these tags, *<MatrixUniform>* the other tags follow a similar format and have similar attributes.

The *<MatrixUniform>* tag is used to pass on matrices, mainly OpenGL state matrices that are used to transform coordinates from one set to another but arbitrary matrices can be constructed. A matter of importance is the value of the 'name' attribute since that value must be exactly the same as the value that is used in the Cg code. This mechanism is then used to bind the input variables.

<MatrixUniform> Name:	InputType:	Defines an uniform Matrix parameter for the shader Function:
name	String	variable name for the matrix, must be the same as used in shader program
type	Enumeration:	Describes the type: can be float, double or state
number_of_elements	Integer	Number of matrix elements
value	String	Origin of matrices(state) or matrix entries

5 Conclusions & Recommendations

Our goals of our initial project were to investigate and document the use of real-time shaders in VTK, construct a realistic surface rendering of a bone model and to learn to use 3D graphics techniques in general and VTK plus Cg specifically. In order to do this we have reverse engineered parts of VTK, learned Cg programming and basically hacked until we got a working proof of concept. A by-product of our work is this report that contains a introduction to hardware shaders in VTK. When we started there was no material on how to link hardware shaders with VTK since this is a fairly new feature. We distilled this tutorial so that other developers can benefit from our work. At this point we have shown how to link HW shaders into the VTK pipeline using the XMLMaterial file.

Future steps that can be taken in the development of hardware shaders in VTK are:

- Streamline and finalize the material file
- Built in extended texture support
- Add support for HLSL
- Provide documentation

Streamline the material file: At this point the VTK developers allow the parameters to be passed on to the shaders through multiple tags and paths. We could for instance pass on the Lightvector via the LightUniform tag, a normal Uniform tag or by setting the PropertyUniform tag. This ambiguity only introduces unnecessary errors and probably stems from the fact that the developers provided tags for all vtk objects like Light, Property and Camera but also provide tags for all Cg parameters like float3, matrix3x3 etc. We believe that a uniform choice should be made so that parameters are passed to the shader in a transparent manner. At this point we have a slight preference for the solution where all parameters will be passed on using their VTK object tags since this is easier for VTK developers. The addition of texture support and ample documentation is evident. As recommendations for the medical imaging groups we can advice to start with building real-time Cg or GLSL shaders based on the offline render programs already in use such as Renderman. This will allow for fast development without sacrificing quality. When fragment shaders and especially bump mapping is satisfactorily implemented, we recommend augmenting the bone models (vtp file) with normal maps, Tangent space vectors or other information that can be computed during generation so that we can utilize this information when performing real-time rendering. HLSL should be added to unlock the vast developing community that works with HLSL, one of the most popular shading languages out there. Regarding our last goal, we can honestly say that we have come along way from never having heard of VTK and from knowing pixel shaders only trough video game literature, to have built an hardware shaded humerus using VTK and Cg. We would like to thank Dr. Charl Botha for the help and enthusiasm and the visualisation group for their assistance and the use of their computers.

6 References

[1] W. J. Schroeder, K. M. Martin, W. E. Lorensen. *The Design and Implementation Of An Object-Oriented Toolkit For 3D Graphics And Visualization.* GE research.

[2] J. Neider, T. Davis, Mason Woo. *OpenGL Programming Guide*. Addison-Wesley, 1993.

[3] J.F. Blinn, *Simulation of wrinkled surfaces*, Caltech/ JPL, 1979

[4] Søren Dreijer, *Bump Mapping Using CG (2nd Edition)*, Blacksmith Studios, 2005

[5] NVidia, *Cg user manual/ Cg Tutorial*, NVIDIA Corporation, 2005

[6] M.J. Kilgard, *Cg in Two pages*, NVIDIA Corporation 2003

[7] P. Shirley, *Fundamentals of Computer Graphics,* A.K. Peters, 2005.

[8] W. Scroeder, K. Martin, B. Lorensen. *The Visualisation Toolkit 2nd edition,* Prentice Hall, 1997