Outline

• RenderMonkey™ IDE overview:
  – Design goals
  – IDE overview
  – Effect structure overview

• Shader creation examples

• Overview of advanced RenderMonkey™ features

• Conclusions
RenderMonkey Addresses the Needs of Game Developers

- Shaders are more than just assembly code

- Encapsulating shaders can be complex
  - Cannot deliver shader-based effects using a standard mechanism

- Solve problems for shader development
  - Designing software on emerging hardware is difficult
  - Lack of currently existing tools for full shader development
  - Need a collaboration tool for artists and programmers for shader creation
Designed for Extensibility

- Flexible framework design
- Allows easy incorporation of existing APIs
  - Supports DirectX® 8.1 and DirectX® 9.0 in current versions
  - RenderMonkey version 0.9 Beta already supports Microsoft HLSL
  - Extensible framework to support emerging HLSL standards
    - OpenGL 2.0 Shading Language
RenderMonkey Facilitates Efficient Shader Development

- Simplifies shader creation
- Fast prototyping and debugging of new graphics algorithms
- Helps you share graphics effects with other developers and artists
- Easy integration of effects into existing applications
- Quickly create new components using our flexible framework
- A communication tool between artists and developers
- Provides an open platform for ATI and ISV’s to develop and incorporate future shader tools
Program Your Shaders Using Our Intuitive, Convenient IDE
RenderMonkey Effect Data Organization

- Encapsulate all effect data in a single text file
- Each Effect Workspace consists of:
  - Effect Group(s)
    - Effect(s)
  - Pass(es)
    - Render State
    - Pixel Shader
    - Vertex Shader
    - Geometry
    - Textures
  - Variables and stream mapping nodes
    - Variables can live at any level of the workspace
Uses Standard XML File Format

- Allows easy data representation
  - Industry standard
  - User-extensible
  - Parsers are readily available
  - User-readable file format

- Describes all effect-related information
  - Shader code
  - Render states
  - Models / texture information

- Import and export easily from your native formats
  - Use our parser and run-time format
  - Write an exporter / importer plug-in
Constant Color Shading Effect
Example

• Simple beginner shader
  – Transform vertices into clip space
  – Shade them using constant material color

• Variables used for this effect:
  – Material color variable
  – View projection matrix

• In this example we’ll learn:
  – How to create a new workspace in RenderMonkey
  – How to create variables and edit their values
  – How to setup geometry model
  – How to link variables to constant store registers
  – How to edit shaders in RenderMonkey
Access Your Effect Data

• All effect data organized in a hierarchical Workspace tree view

• Easily identify data node types by their icons
Effect Group Nodes

- Group related effects in one container
- Provides mechanism for dealing with a lot of effects
- Facilitate fallback versions of same effect
- Group shaders by type, pick the first one that validates
- How you group effects is entirely up to you
Effect Nodes

- Encompasses all information needed to implement a real-time visual effect
- Composed of multiple passes
- Inherit from a Default Effect
  - Used to set a known starting state for all effects
Data Scope and Traversal

• Store common effect data in the default effect:
  – All other effects inherit data from it
  – Easy to share data from a common point

• Effects don’t inherit from other effects in the workspace
  – Common data that should be global to the effects:
    • Stream mapping
    • Texture variables
    • Model variables

• Variable scope is similar to C-style variable scope

• Data validation occurs upward through passes in the effect then upward through passes in the default effect
Pass Nodes

• Every pass is a draw call

• Passes inherit data from previous pass within the effect
  – First pass inherits from default effect

• A typical pass contains:
  – A vertex and pixel shader pair (either HLSL or ASM) (a requirement)
  – Render state block
    • Render states are inherited from pass to pass
  – Texture objects
    • Must reference a valid texture variable
    • Each texture object stores associated texture states
  – Geometry model reference (a requirement)
  – Stream mapping reference (a requirement)
  – May also contain nodes of other types (variables, etc)

• Different geometry can be used in each pass
Variable Nodes

- Parameters to your shaders
- More intuitive way of dealing with constant store registers
- Give shader constants meaningful names and types
- Manipulate shader constants using convenient GUI widgets

- Supported variable types:
  - Matrices
  - Vectors
  - Scalars
  - Colors
  - Textures
  - Strings
Pre-defined Variables Help You Set Up Your Shaders Quickly

- RenderMonkey IDE calculates their values at run-time

- Provide a set of commonly used parameters:
  - View projection matrix
  - View matrix
  - Inverse view matrix
  - Projection matrix
  - View direction vector
  - View position vector
  - Time
  - cos_time, sin_time, tan_time
Edit Shader Parameters Using GUI Widgets via Editor plug-ins

- Utilize knowledge of the variable type for editing:
  - Color Editor
  - Vector Editor
  - Matrix Editor
  - Scalar Editor
- Edit variables with custom widgets: easy to create your own
Full Support for DirectX® 9.0 Shader Models

• Assembly shaders
  – Vertex shader versions 1.0/1.1 – 2.0
  – Pixel shader versions 1.0/1.1/1.3/1.4 – 2.0
  – Future support for shader versions 2_0_x, 2_0_sw and 3_0

• Integrated HLSL support
Edit Shaders Using Specialized Editors

• Tabbed view allows editing multiple passes in the effect

• Automatic syntax coloring of shader code
  – Separate syntax rules for HLSL and ASM shaders

• Type your code, compile and see instant results!
HLSL Shader Editor Plug-in

- Allows incredible ease of shader creation
- Simple interface links RenderMonkey nodes to HLSL variables and samplers
  - Link vectors, colors, scalars and matrices to variable parameters
  - Link texture objects to samplers
- Control target and entry points for your shaders
Quickly Learn How to Create HLSL Shaders

- **HLSL Shader Programming with RenderMonkey** GDC presentation

- View *High Level Shading with DirectX® 9 on ATI’s RADEON™ 9700 Series* Net Seminar on ATI Developer website

- Attend *High Level Shader Language Workshop* at Microsoft booth
Assembly Shader Editor Plug-in

- Easily link RenderMonkey variable nodes to constant store registers
- Syntax colored for shader assembly language
Linking Variable Nodes to Constant Store Registers is Easy

• Use constant store editor

• Bind a constant storage register to a variable from the effect workspace

• Preview the incoming values
Read All Application Messages in a Single Place - The Output Module

- Output the results of shader compilation and application messages
- Linked with the shader editor for compilation error highlighting
Integrated Compile Time Error Reporting Simplifies Shader Creation

- Compilation errors displayed in the output module window
- Double-clicking on the error highlights the line containing erroneous code in the editor
Interactively Preview Your Effects in the Viewer Plug-in

- All changes to the shader or its parameters modify the rendered image in real time

- DirectX® 9.0 preview
  - HAL / REF

- Customize the preview:
  - Standard trackball navigation
  - Customizable settings for camera and clear colors
  - A set of preset views (Front / Back / Side / etc)
Viewer Plug-in (cont.)

- Incremental pass preview to visualize shader interaction
Textured Shading Effect Example

- Build upon previous shader example
- Explore ways to apply a texture map to the underlying model using shaders
- In this effect, we’ll learn:
  - How to use more of RenderMonkey pre-defined variables
  - How to setup stream mapping
  - How to work with texture objects
Simplified Stream Mapping Setup

- Setup each stream using the Stream Mapping Editor
- A stream mapping node can be created at any point in the workspace
- Stream mapping nodes can be shared by multiple effects
- Multiple references to a stream mapping node can be created throughout the workspace
- Each pass must have its own stream mapping reference
Using Textures in a RenderMonkey Effect

- Texture variables can be shared between different effects

- Support the following texture types:
  - 1D, 2D texture maps (JPEG, TGA, BMP formats)
  - Cube maps (DDS)
  - Volume textures (DDS)
  - Dynamic renderable textures

- Texture objects belong to a pass node
  - Reference a texture variable to sample from
  - Store all related texture and sampler states
Texture and Sampler State Editing

- Specify texture and sampler state values (filtering, clamping, etc) for each texture node within a pass

- Texture and sampler states are bundled together in one editor

- State changes modify rendered output in real time
Setup All Render States in the Render State Editor Plug-in

- Modify any render state within a particular pass

- Render states are inherited
  - From previous passes in the effect
  - From the default effect

- Great for exploring the results of changing a render state – a useful learning tool
Dynamic Texture Rendering Example

- Direct output of one or more passes to a texture map
- Sample from that texture to create interesting effects
  - Scene post-processing effects
    - Depth of Field
    - Gaussian Blur
    - Tone Mapping
    - HDR Rendering
    - Other image processing techniques
- In this effect, we’ll learn:
  - How to direct pass output to a renderable texture
  - How to sample from a dynamic texture
  - How to edit parameters for a renderable texture / target
Rendering to a Texture Using RenderMonkey

• Simple to setup

• Special texture variable type:
  – Renderable Texture

• Direct pass output to a texture
  – Use Render Target node to link to a Renderable Texture

• Modify parameters using appropriate editors
  – Render Target Editor
  – Renderable Texture Editor
Customize Renderable Texture

- Specify desired texture format
  - Select from a variety of formats

- Control texture dimensions
  - Width and Height
  - Tie the texture size to viewport dimensions
Control Render Target Parameters

- Modify parameters to suit your needs
- Specify whether the texture should be cleared
- Specify clear color
- Toggle whether the depth buffer should be cleared
- Specify the depth clear value
Develop Shaders With Your Artists

• Use the Artist Editor Interface to explore shaders:
  – Expose the power of programmable shaders to artists and designers
  – Programmers and Artists living in harmony!

• View workspace using Art tab
  – Only view data relevant to the artists
  – Programmers can select which data is artist-editable

• Provides look and feel of GUI widgets artists are familiar with

• See changes in real time
Edit All Artist Parameters Using a Single Interface

- Programmer has control of what parameters can be modified
  - Flag variables as ‘artist-editable’ as needed

- Artist modifies only specified variables:
  - Use the Artist Editor interface

- Data organization follows the effect structure:
  - Variables are grouped in tab sheets by passes/effects they belong to

- Artists can tweak parameters and *instantly* see changes:
  - Modify vectors, scalars, colors using convenient controls
Artist Editor Interface
RenderMonkey IDE is built from plug-ins

- IDE ships with standard set of plug-ins
  - Shader Editors
  - Variable Editors
  - Artist Editor
  - Preview Window

- Write your own plug-ins with the RenderMonkey SDK
  - Beta SDK will contain sample code for a set of supported plug-in modules
    - Exporter / Importer
    - Editor
    - Geometry Loader

- Let us know if there are specific problems you are trying to solve – we’ll be happy to work closely with you to solve them!
Examples of Custom Plug-in Modules

• Editors for engine-specific data types
  – Edit custom game engine data
  – Link engine state to shader effects

• Custom Preview Module
  – Preview effects using your engine

• Import / Export Plug-ins
  – Write a custom importer/exporter to support your engine
  – Write a custom model loader to handle your data format

• Texture Creation plug-ins
RenderMonkey Plug-in SDK

- Uses lightweight C plug-in interface

- Message passing system
  - Application messages are passed to the plug-in along with necessary database information

- Allows the plug-in to navigate and modify the effect database directly

- Plug-ins can communicate with other plug-ins through the RenderMonkey IDE
Beta SDK Caveat

• The plug-in interfaces may change from Beta SDK to the full release
  – *Beta plug-in interfaces may not be fully supported in future releases and will need to be updated*

• Compact plug-in interface allows to separate plug-in interface code from plug-in implementation

• RenderMonkey SDK version 1.0 will be supported in subsequent releases
Summary

• RenderMonkey IDE is a powerful, intuitive environment for developing shaders

• Prototype your shaders quickly

• Explore all parameters for your shaders using convenient GUI widgets

• Let your artists tweak shader parameters to find the desired look

• Develop shaders with your artists!
ATI Developer Resources

• Visit the ATI Developer Relations website to view this and other presentations:
  – www.ati.com/developer

• Download RenderMonkey™:
  – Full documentation available

• View *High Level Shading With DirectX 9® on ATI’s RADEON™ 9700 Series* Net Seminar:
  – http://www.ati.com/developer