mental mill® & MetaSL™:
Advances in Cross-Platform Shader Authoring

Laura Scholl, Product Manager, mental mill
mental images, Inc.
As GPU’s and rendering algorithms become more powerful it becomes increasingly more difficult and more costly to:

• create and maintain shader assets with currently available shading languages
• share shader assets between different media/content
mental mill® & MetaSL™

mental mill and its underlying MetaSL shading language provide the solution to this problem with:

- Visual shader creation and programming SDK
- High-level shader representation

Based on the object-oriented programming paradigm, together mental mill and MetaSL provide mechanisms for:

- Abstraction
- Modularity
- Encapsulation
Abstraction: MetaSL

- General purpose, high-level shading language

```cpp
void main()
{
  // Sample the height and center on 0.0 (scaled down too)
  Scalar h = tex2D(height_tex, texture_coordinate[1].xy).x;
  h = h*amount - amount*0.5;

  // The view direction in tangent space.
  Vector3 vtan = tangent_space[0]*direction;

  // perturb the vertex coords. Move the texture toward the eye when
  // the height is negative and away when positive
  Vector2 uv = texture_coordinate[1].xy;
  uv.x *= vtan.x*h;
  uv.y *= vtan.y*h;

  // get the normal from the normal map
  Vector3 n = (tex2D(norm_tex, uv).xyz - 0.5) * 2.0;

  // transform the normal out of tangent space and re-normalize
  normal = normalize(n*tangent_space[0]);

  // diffuse and specular results from lighting
  Color diffuse = Color(0.0, 0.0, 0.0);
  Color specular = Color(0.0, 0.0, 0.0);

  // iterate over scene lights
  Light_iterate light:
  foreach (light)
  {
    Vector3 whalf = normalize(light.direction - direction);
    Vector4 illum = illumination(
      light.dot_nl, dot(normal, whalf), specular_shininess);
    diffuse += illum.y * light.contribution;
    specular += illum.z * light.contribution * specular_color;
  }
}
Abstraction: MetaSL

- General purpose, high-level shading language
  - Provides language mechanisms that:
    - allow the same MetaSL shader to be used:
      - on different hardware platforms (CPU/GPU)
      - in different contexts (film, games, graphics...)
      - with different rendering algorithms
    - take advantage of advancements in:
      - GPU’s
      - rendering algorithms
Abstraction: mental mill

- Visual shader creation/programming environment
  - Provides the ability to create shader graphs rather than specify low-level programming details that encourage artists to experiment without:
    - specifying implementation details
    - depending on programmers
mental mill® & MetaSL™

Abstraction: mental mill
- Visual shader creation/programming environment
Modularity: mental mill

- Visual shader creation/programming environment
  - Provides atomic nodes of MetaSL code that can be combined in a shader graph that allows the artist to build complex shader effects from:
    - other shaders
    - functions
    - state modifiers

© 2008 NVIDIA Corporation.
mental mill® & MetaSL™

Shader graph of atomic MetaSL nodes
Encapsulation: mental mill

- Visual shader creation/programming environment
  - Provides Phenomena™ to enclose a shader graph into a single shader to:
    - simplify parameter lists
    - protect a developed look
    - reuse in creating complex shader graphs
mental mill® & MetaSL™

Encapsulation: mental mill
mental mill® & MetaSL™

mental mill

- Platform independent shader development library that provides tools to:
  - parse MetaSL into an Abstract Syntax Tree (AST)
  - write back-end translators/compilers
  - create a graphical user interface

© 2008 NVIDIA Corporation.
mental mill

- The shader development library comprises:
  - UI components:
    - Shader graph and Phenomenon editor
    - Shader parameter editor
    - MetaSL editor and debugger
  - Shader library browser
  - Translator/compiler components:
    - Shader graph
    - Phenomenon
    - MetaSL
MetaSL

- Abstract shader representation:
  - Similar to C++
  - Familiar shader types:
    - Displacement
    - Surface
    - Light
    - Volume
    - Environment
    - Lens
MetaSL

- Abstract shader representation:
- Shader class
- Input and output parameters
- Main method and other shader methods
- Shader state variables and functions

```cpp
shader My_shader
{
    input:
        Color color(1,1,1,1);

    output:
        Color result;

    void main()
    {
        Vector3 n = transform_normal("internal", "camera", normal);
        result = color * saturate(n.z);
    }
};
```
mental mill® & MetaSL™

mental mill data flow overview

MetaSL Code

OEM Application/mental mill GUI

MetaSL, Phen, and Graph Compilers

Back-end Plug-in Interface

Intermediate languages

Cg

HLSL

GLSL

C++ (mental ray)

Other

Native Compilers

Rendering

© 2008 NVIDIA Corporation.
mental mill® & MetaSL™

mental mill

- Visual shader creation/programming environment
mental mill® & MetaSL™

Phong shader in MetaSL render
mental

mental mill® & MetaSL™

Debugger ‘get info’
mental mill® & MetaSL™

Phong shader
MetaSL code

© 2008 NVIDIA Corporation.
mental mill® & MetaSL™

Phong shader debug mode evaluate RS
Phong shader
deploy mode
evaluate specular
Phong shader
debug mode
evaluate specular*RS
mental mill® & MetaSL™

Add a texture to the Phong shader
mental mill® & MetaSL™

mental ray® preview renderer plug-in
mental mill® & MetaSL™

Add a reflection to the Phong shader
mental mill® & MetaSL™

Create a Phenomenon
mental mill® & MetaSL™

Shiny Plastic Phenomenon
Shiny Plastic Phenomenon in shader graph
mental mill® & MetaSL™

Debugger ‘get info’
mental mill® & MetaSL™

Debugger ‘get info’
Generator_water shader
mental mill® & MetaSL™

Flood shader - using Generator_water
The problem of writing a new shader asset for:
• every platform
• different contexts
• rapidly advancing hardware
• evolving rendering algorithms
is solved with the abstract, high-level shading language MetaSL and mental mill’s library of tools that create and translate shaders to all platforms.
mental mill

- **Artist Edition:**
  - Bundled with FX Composer 2.5
  - Without MetaSL editing and debugging
- **Standard Edition:**
  - Integrated shader editing and debugging tools
  - Back-end plug-ins for 3ds Max, Maya, XSI, Catia, SolidWorks, mental ray, C++, and FX Composer
- **Integrator Edition:**
  - Component libraries for integration into:
    - Design and DCC applications
    - shader pipelines
    - Sample code

© 2008 NVIDIA Corporation.
mental mill® & MetaSL™

demonstration of mental mill workflow