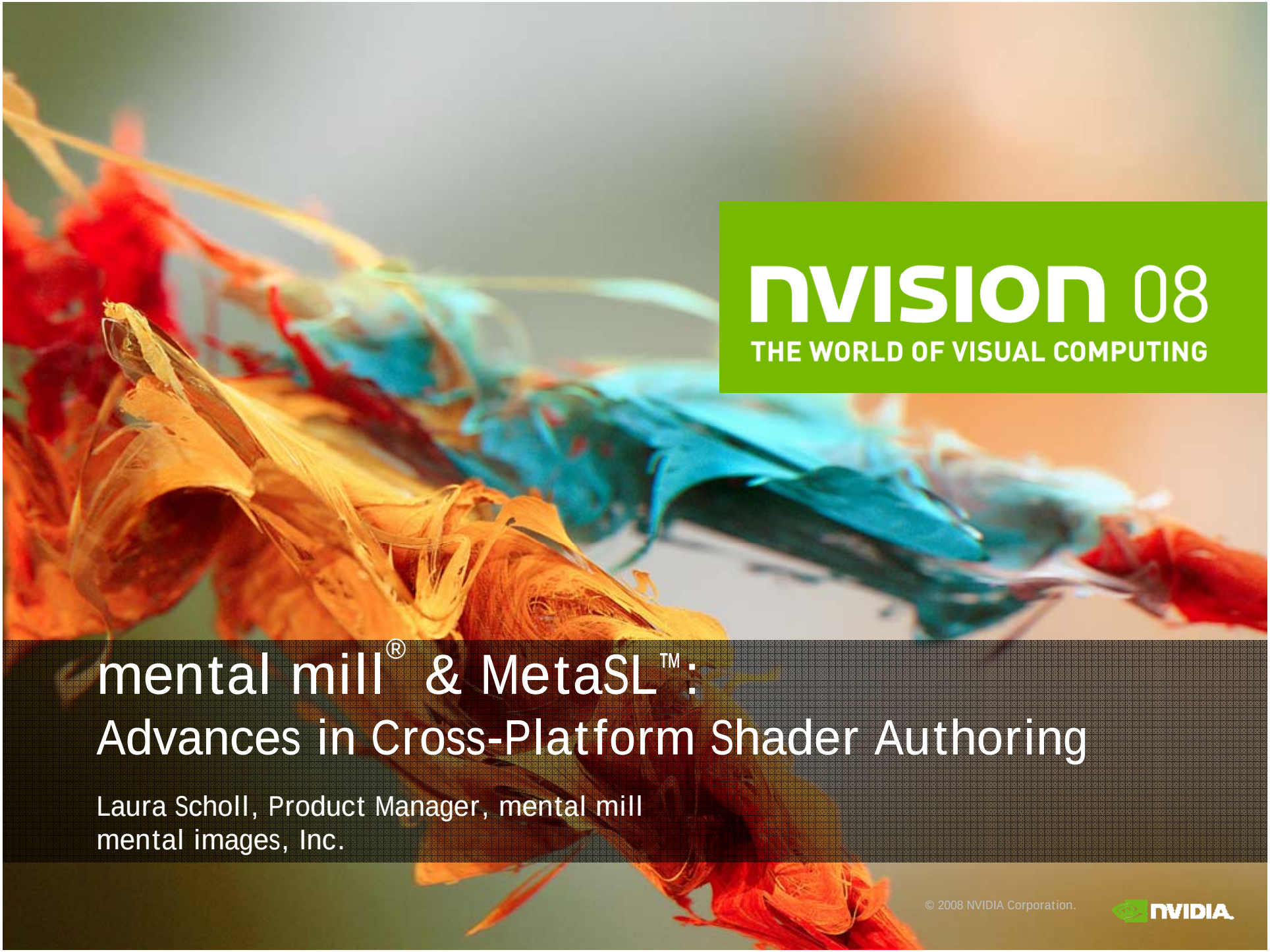
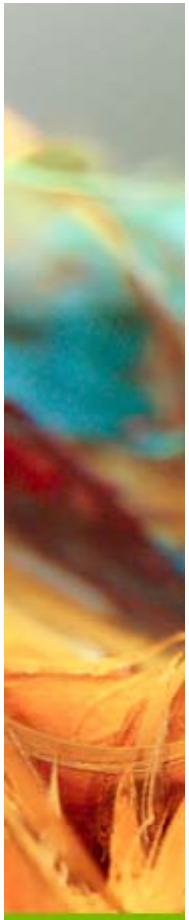


mental mill® & MetaSL™: Advances in Cross-Platform Shader Authoring

Laura Scholl, Product Manager, mental mill
mental images, Inc.

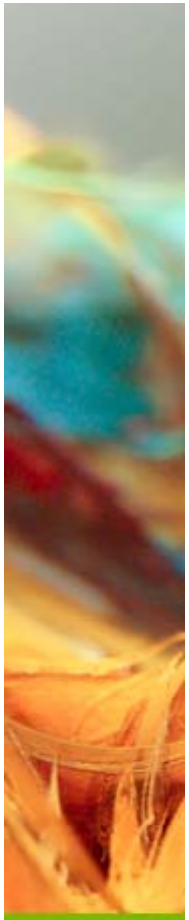
NVIDIA



mental mill® & MetaSL™

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:

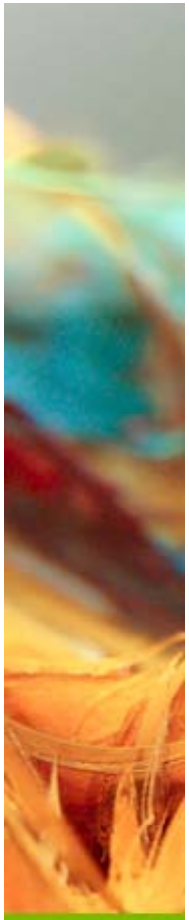
- Abstraction
- Modularity
- Encapsulation

mental mill® & MetaSL™

Abstraction: MetaSL

- General purpose, high-level shading language

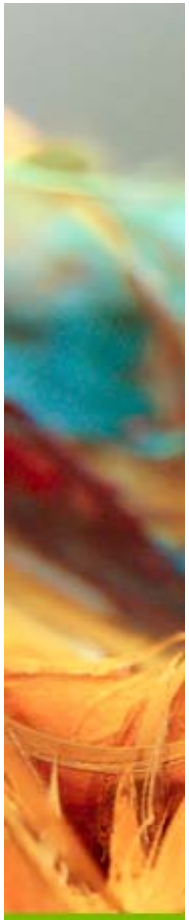
```
47
48      void main()
49      {
50          // Sample the height and center on 0.0 (scaled down too)
51          Scalar h = tex2D(height_tex, texture_coordinate[1].xy).x;
52          h = h*amount - amount*0.5;
53
54          // The view direction in tangent space.
55          Vector3 vtan = tangent_space[0]*direction;
56
57          // perturb the vertex coords. Move the texture toward the eye when
58          // the height is negative and away when positive
59          Vector2 uv = texture_coordinate[1].xy;
60          uv.x -= vtan.x*h;
61          uv.y -= vtan.y*h;
62
63          // get the normal from the normal map
64          Vector3 n = (tex2D(norm_tex, uv).xyz - 0.5) * 2.0;
65
66          // transform the normal out of tangent space and re-normalize
67          normal = normalize(n*tangent_space[0]);
68
69          // diffuse and specular results from lighting
70          Color diffuse  = Color(0,0,0,0);
71          Color specular = Color(0,0,0,0);
72
73          // iterate over scene lights
74          Light_iterator light;
75          foreach (light)
76          {
77              Vector3 vhalf = normalize(light.direction - direction);
78              Vector4 illum = illumination(
79              light.dot_nl, dot(normal, vhalf), specular_shininess);
80              diffuse  += illum.y * light.contribution;
81              specular += illum.z * light.contribution * specular_color;
82          }
83
```



mental mill® & MetaSL™

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



mental mill® & MetaSL™

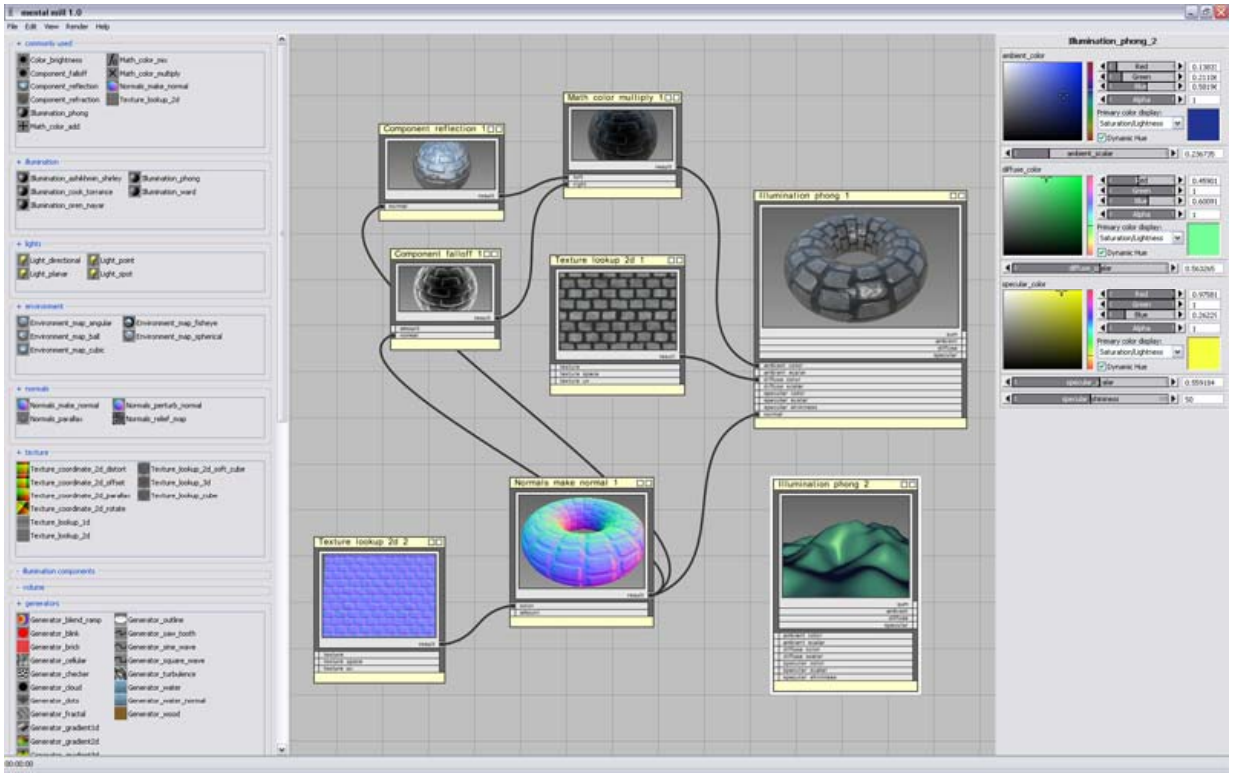
Abstraction: mental mill

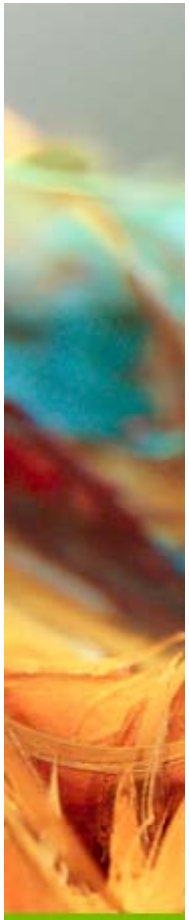
- Visual shader creation/programming environment
  - Provides the ability to create shader graphs rather than specify low-level programming details that encourage artists to experiment without:
    - specifying implementation details
    - depending on programmers

mental mill® & MetaSL™

Abstraction: mental mill

- Visual shader creation/programming environment



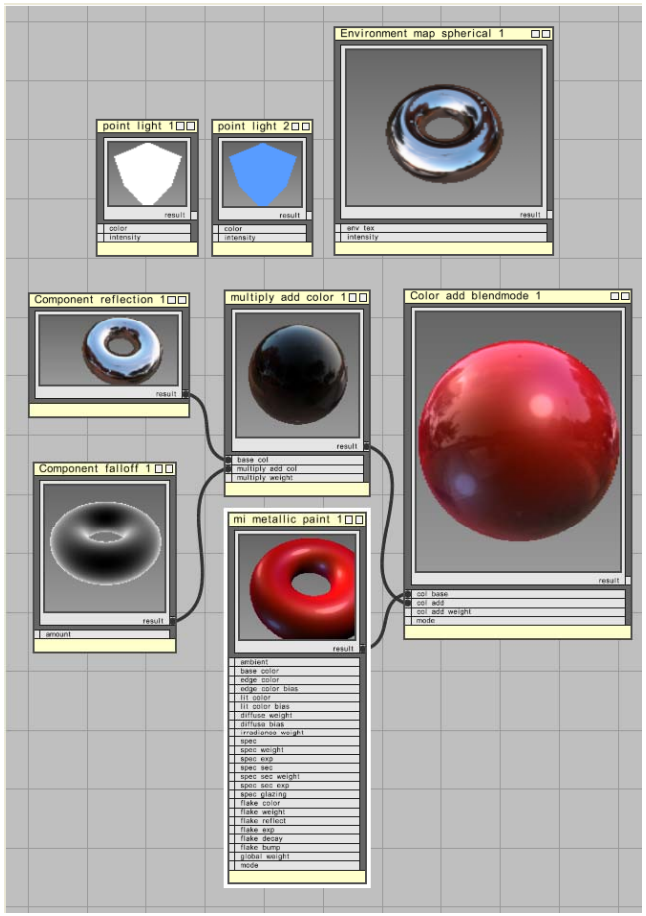


mental mill® & MetaSL™

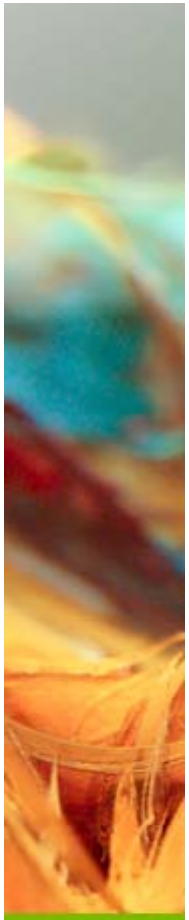
Modularity: mental mill

- 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

mental mill® & MetaSL™



Shader graph of atomic MetaSL nodes



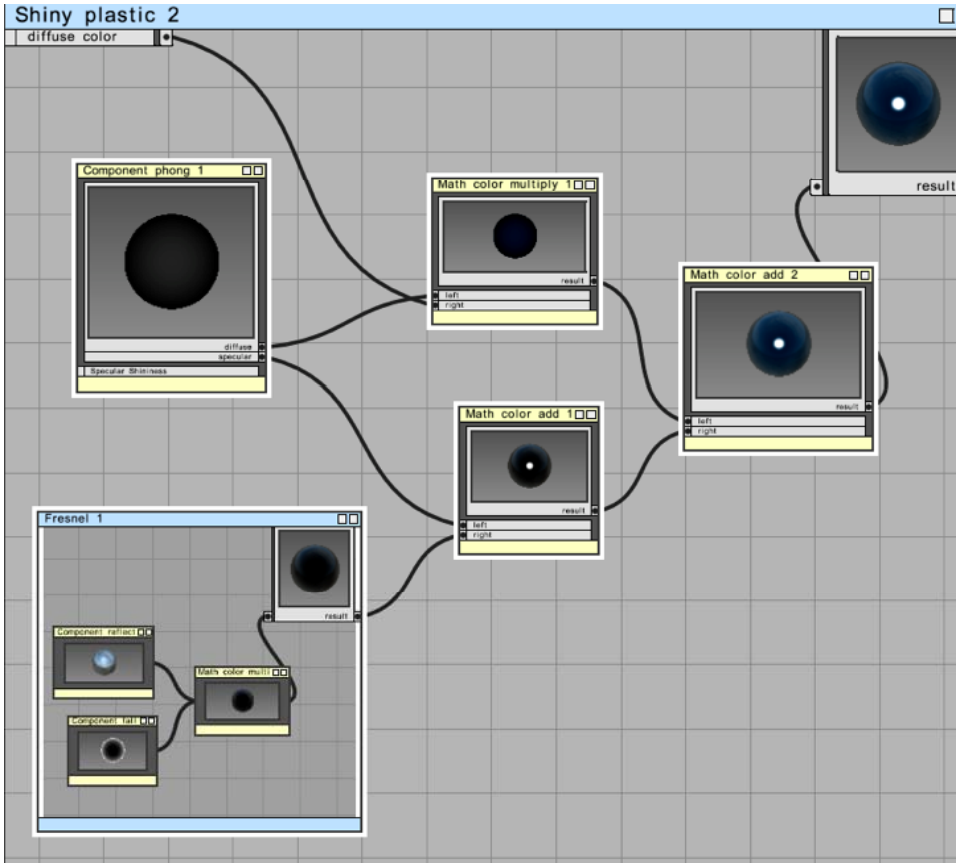
mental mill® & MetaSL™

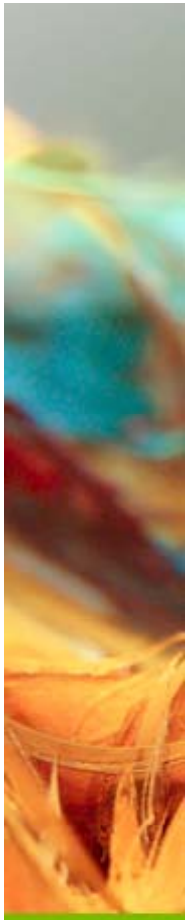
Encapsulation: mental mill

- 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

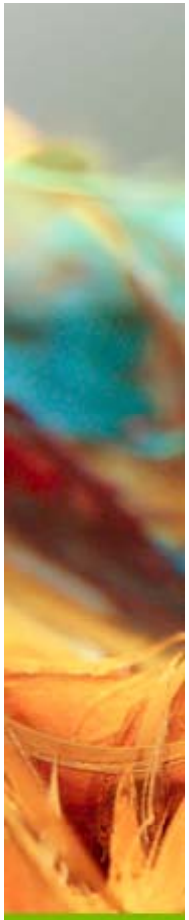




mental mill® & MetaSL™

mental mill

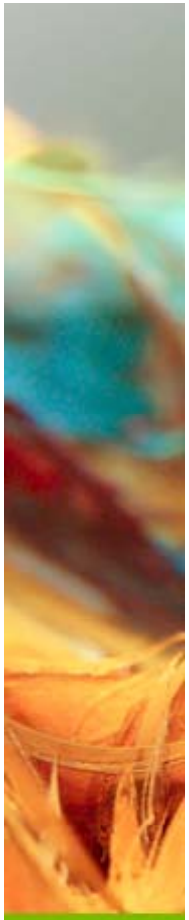
- Platform independent shader development library that provides tools to:
- parse MetaSL into an Abstract Syntax Tree (AST)
- write back-end translators/compilers
- create a graphical user interface



mental mill® & MetaSL™

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



mental mill® & MetaSL™

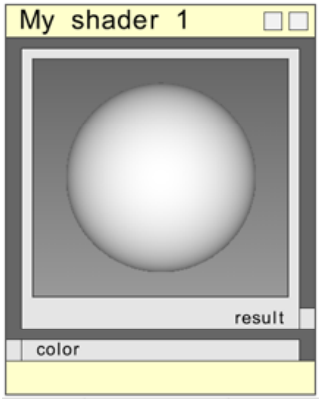
MetaSL

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

mental mill® & MetaSL™

MetaSL

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

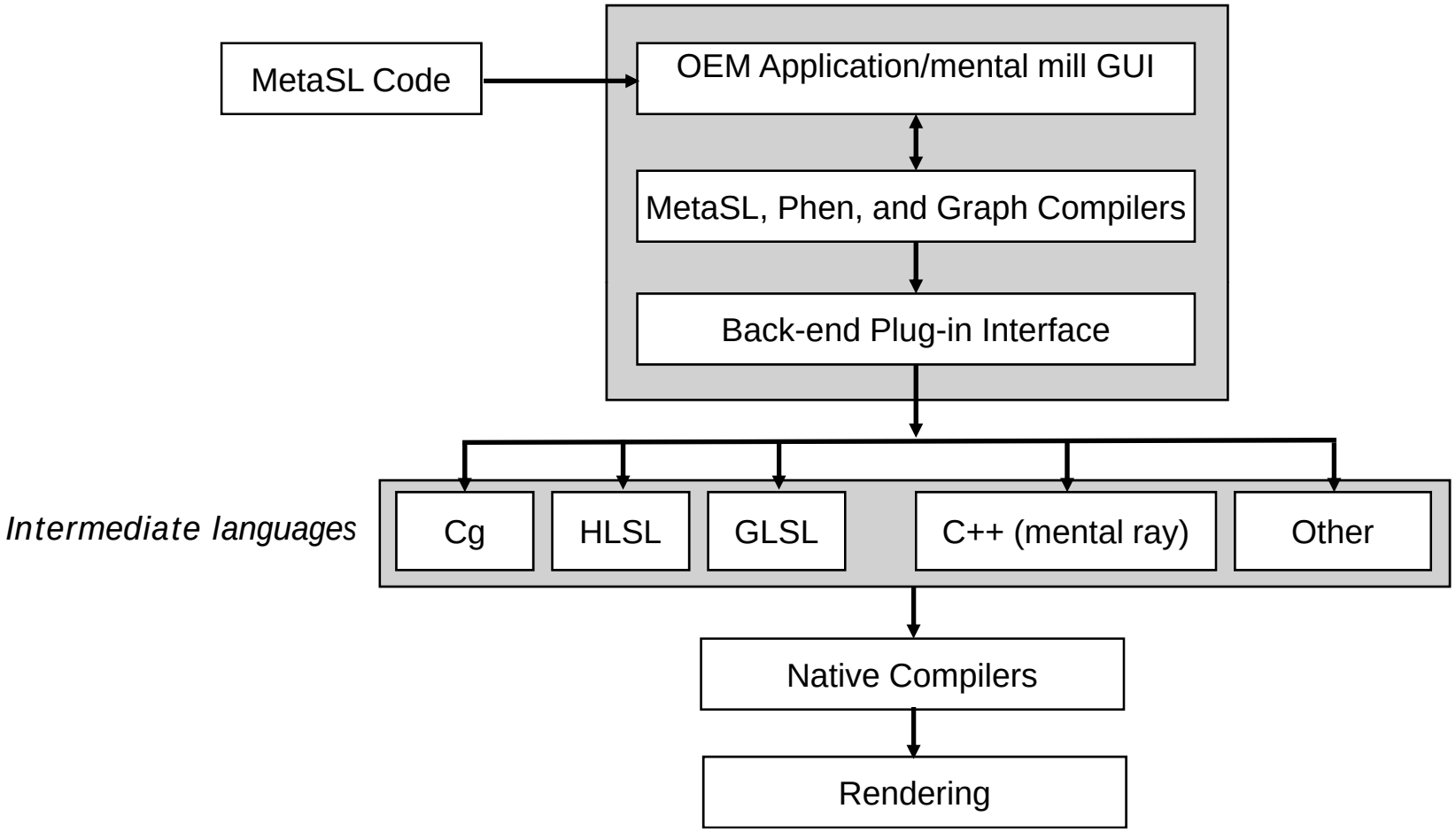



```
shader My_shader
{
  input:
    Color color(1,1,1,1);

  output:
    Color result;

    void main()
    {
      Vector3 n = transform_normal(
        "internal", "camera", normal);
      result = color * saturate(n.z);
    }
};
```

mental mill® & MetaSL™

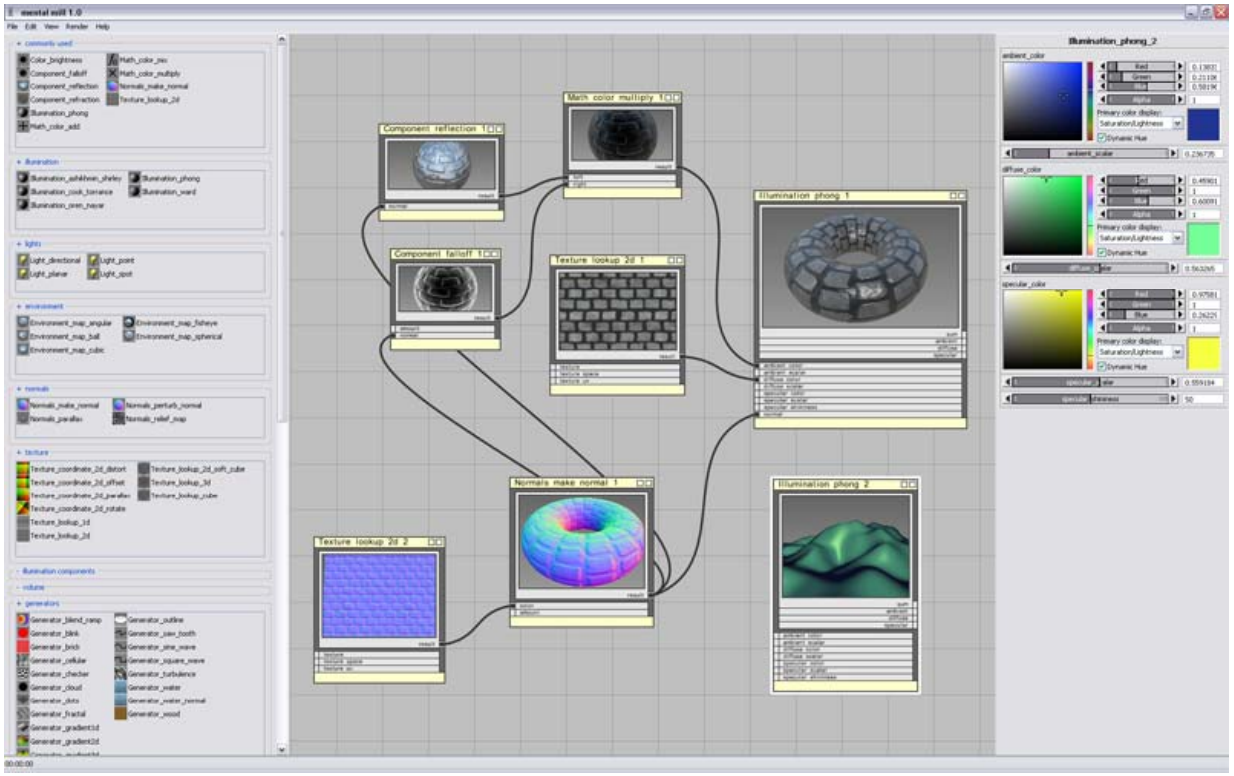


mental mill data flow overview

mental mill® & MetaSL™

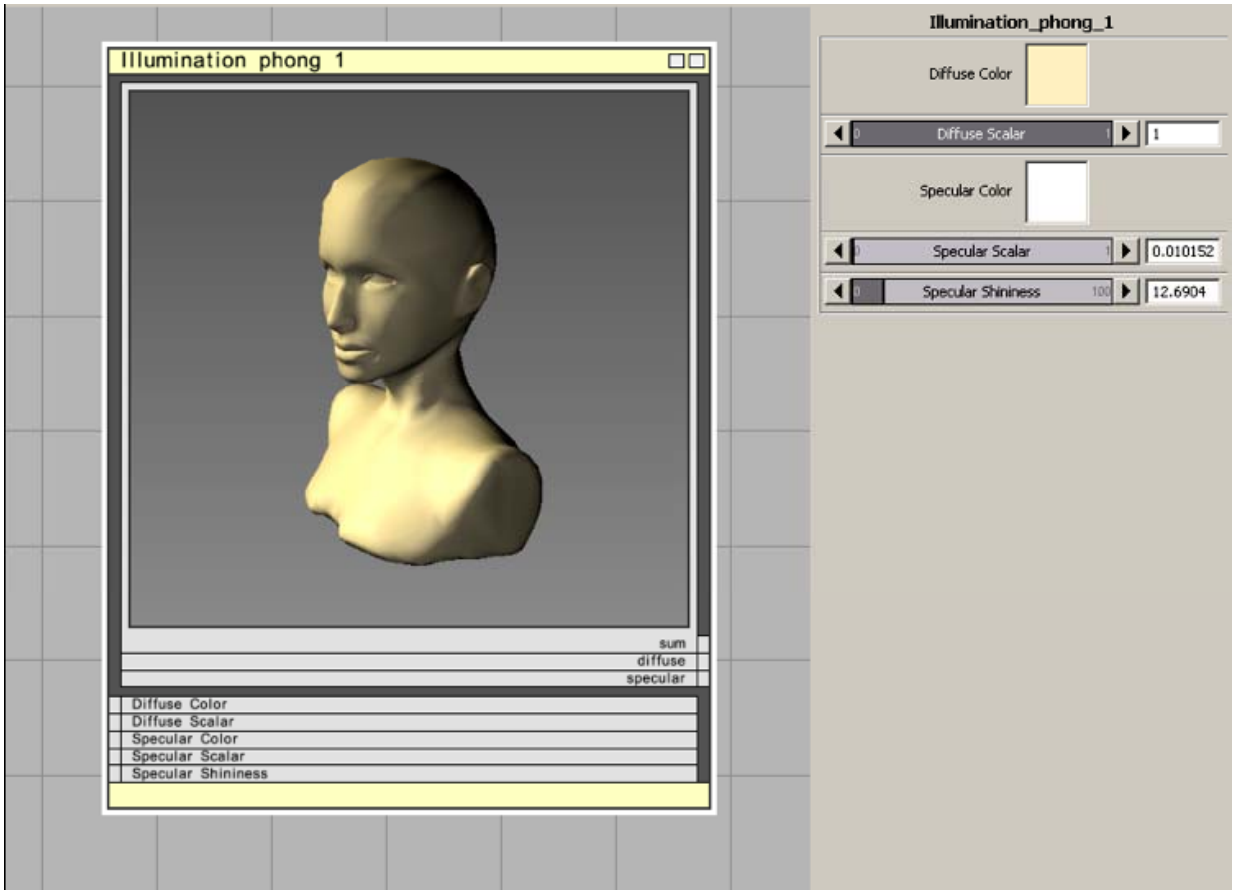
mental mill

- Visual shader creation/programming environment



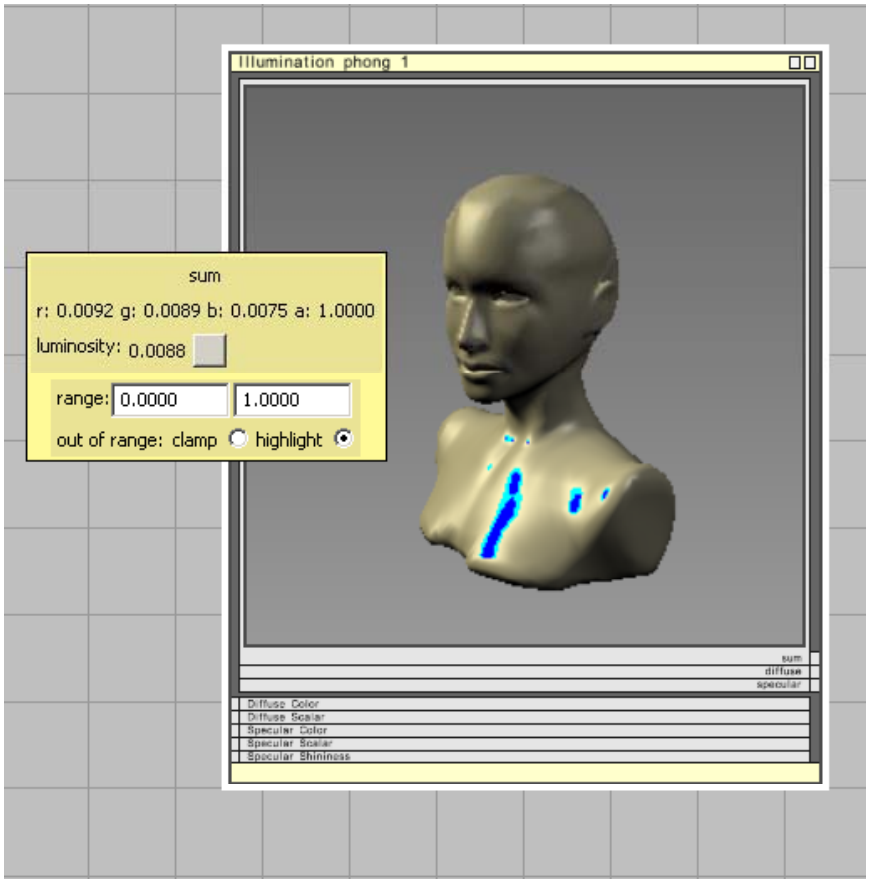
mental mill® & MetaSL™

Phong shader in MetaSL render



mental mill® & MetaSL™

Debugger ‘get info’



mental images®

mental mill® & MetaSL™

Phong shader

MetaSL code

Illumination phong 1

Diffuse Color
Diffuse Scalar
Specular Color
Specular Scalar
Specular Shininess
normal

```
/*****************************************************************************
 * Copyright 1986-2008 by mental images GmbH, Fasanenstr. 81, D-10623 Berlin,
 * Germany. All rights reserved.
 *****************************************************************************/
#import <mi_msl_shared.msl>

shader Illumination_phong
{
  input:
  Color  diffuse_color = Color(1,1,1,1)
  {
      display_name("Diffuse Color");
      hard_range(Color(0,0,0,0),Color(1,1,1,1));
      description( "The color used to tint the Diffuse Reflectance Color." );
  };
  Scalar diffuse_scalar = 0.7
  {
      display_name("Diffuse Scalar");
      hard_range(Scalar(0),Scalar(1));
      description( "The intensity of the light diffusely reflected from "
      "this surface." );
  };
  Color  specular_color = Color(1,1,1,1)
  {
      display_name("Specular Color");
      hard_range(Color(0,0,0,0),Color(1,1,1,1));
      description( "The color used to tint the Specular Reflectance "
      "Color." );
  };
  Scalar specular_scalar = 0.06
  {
      display_name("Specular Scalar");
      hard_range(Scalar(0),Scalar(1));
      description( "The intensity of the specular light reflections from "
      "this surface." );
  };

  Scalar specular_shininess = 15.0
  {
      display_name("Specular Shininess");
      soft_range(Scalar(0),Scalar(100));
      description( "The width of specular highlights across the "
      "surface." );
  };

  output:
  Color sum;
  Color diffuse;
  Color specular;

  void main()
  {
      diffuse  = Color(0,0,0,0);
      specular = Color(0,0,0,0);

      Color Rd = diffuse_color * diffuse_scalar;
      Color Rs = specular_color * specular_scalar;

      // normalize components
      Color temp = Rd + Rs;
      Scalar maxc = max(max(temp.r,temp.g),temp.b);
      if (maxc>1.0) {
      Rd /= maxc;
      Rs /= maxc;
      }

      Vector3 vdir = direction;
```

sum
diffuse
specular

diffuse
diffuse_color
specular_shininess
specular
diffuse_scalar
specular_scalar
sum
specular_color
direction
normal

surfaces with glossy properties, such as plastics and tiles.

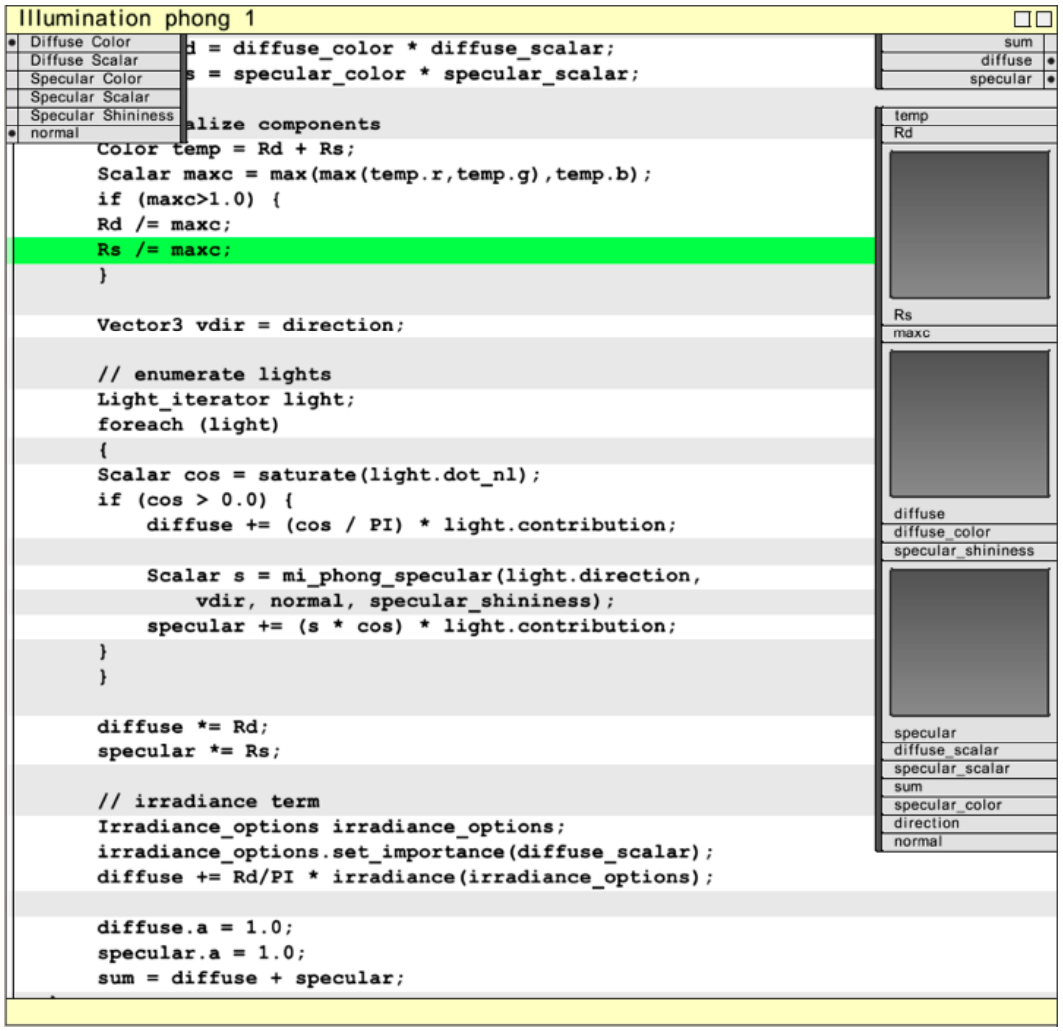
NVIDIA

mental mill® & MetaSL™

Phong shader

debug mode

evaluate RS

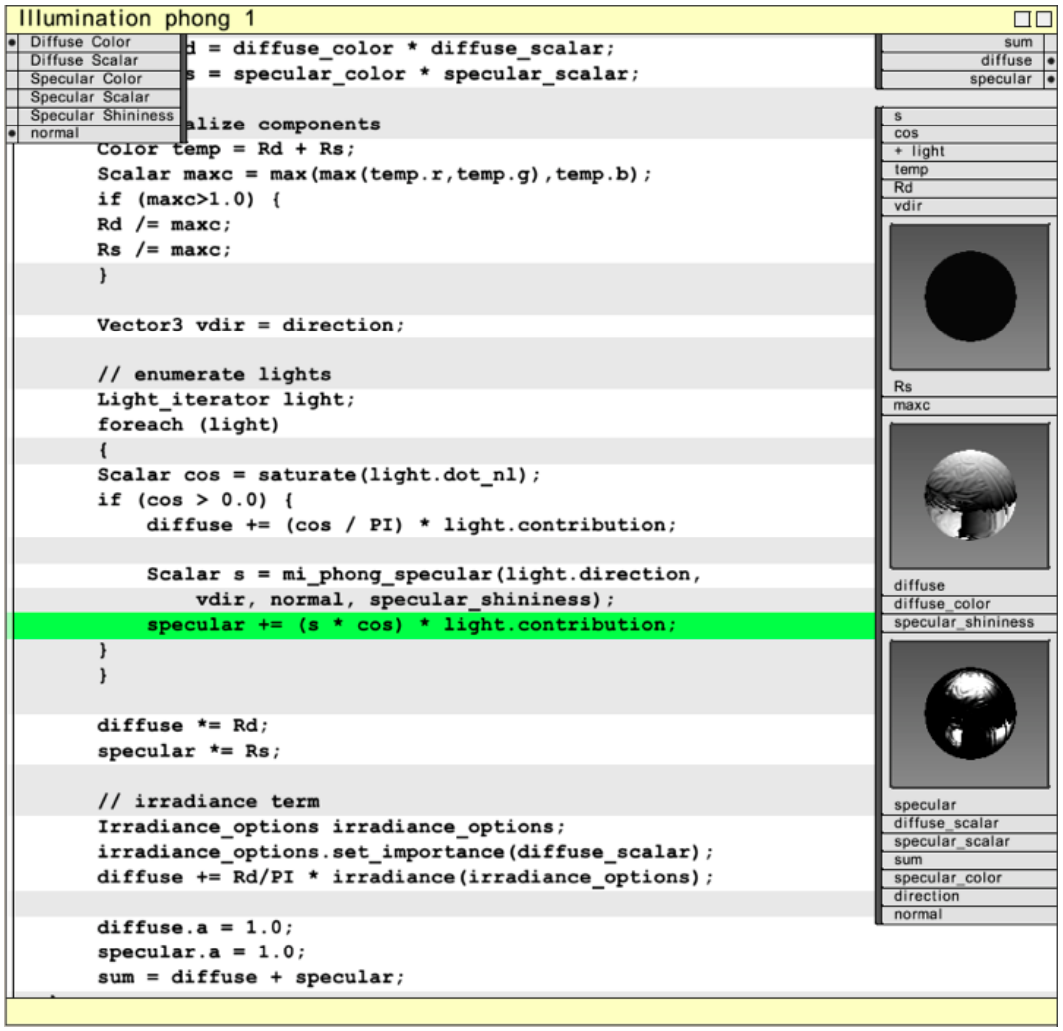


mental mill® & MetaSL™

Phong shader

debug mode

evaluate specular



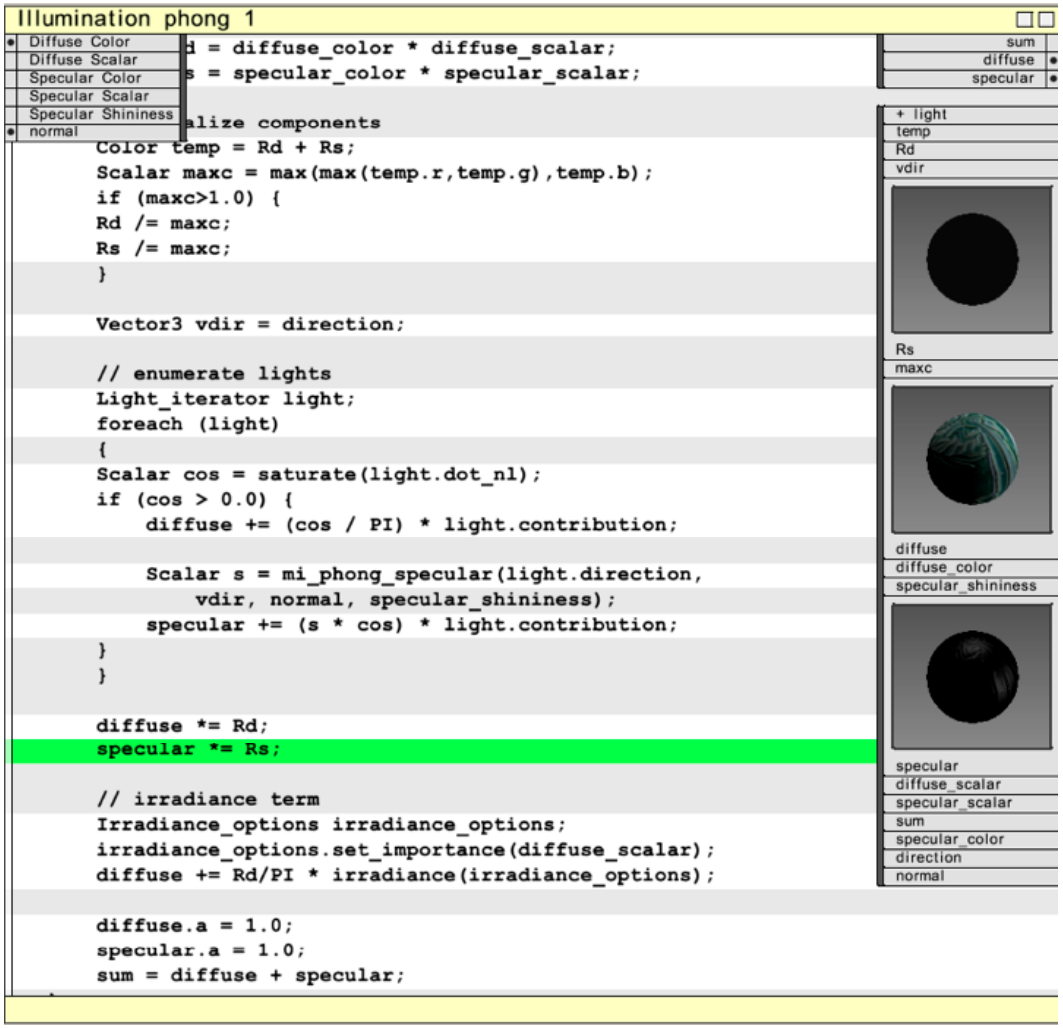
mental images®

mental mill® & MetaSL™

Phong shader

debug mode

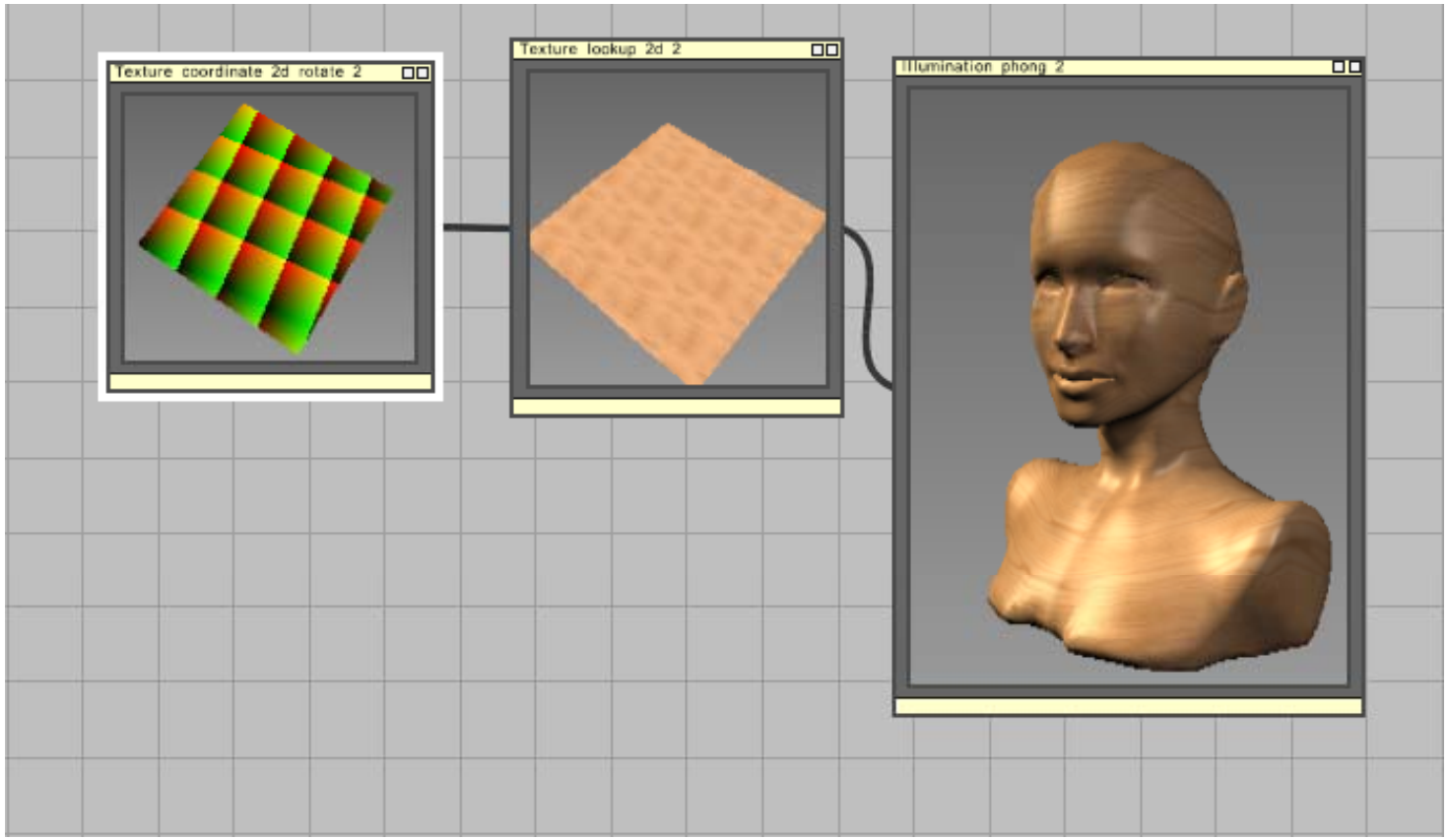
evaluate specular*RS



NVIDIA

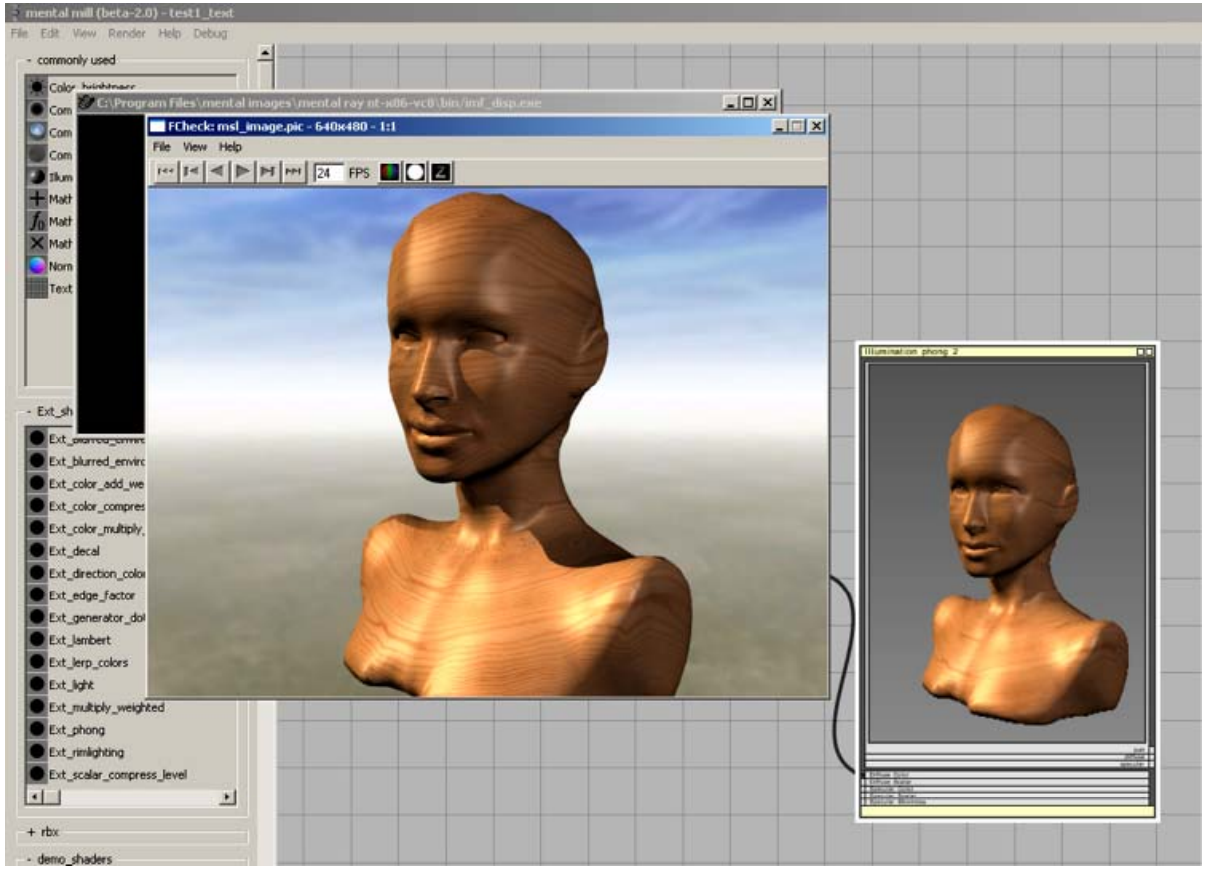
mental mill® & MetaSL™

Add a texture to the Phong shader



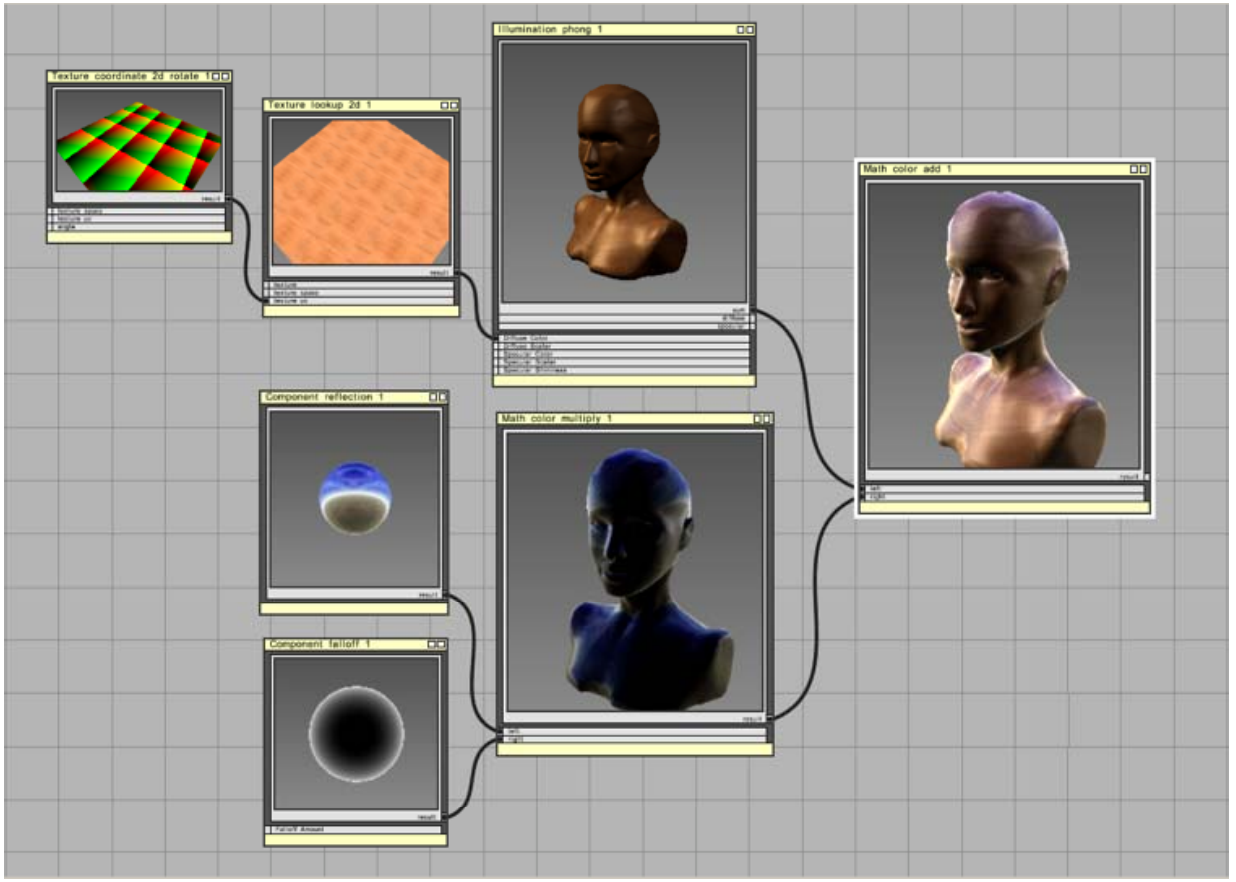
mental mill® & MetaSL™

mental ray® preview renderer plug-in



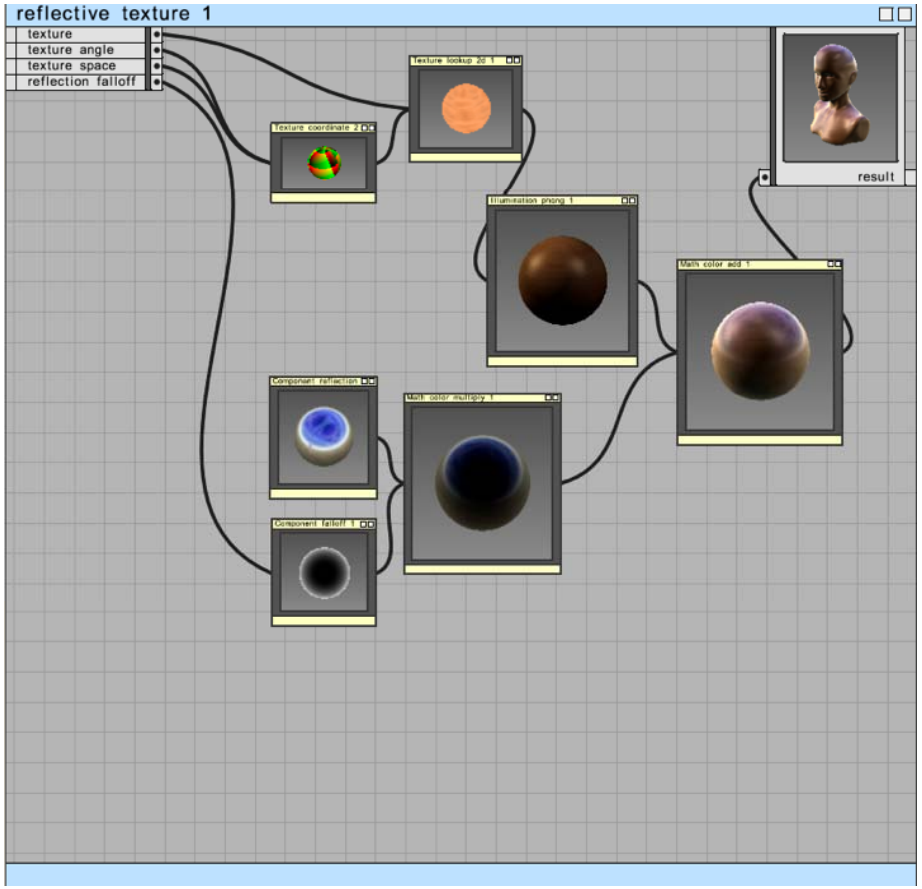
mental mill® & MetaSL™

Add a reflection to the Phong shader



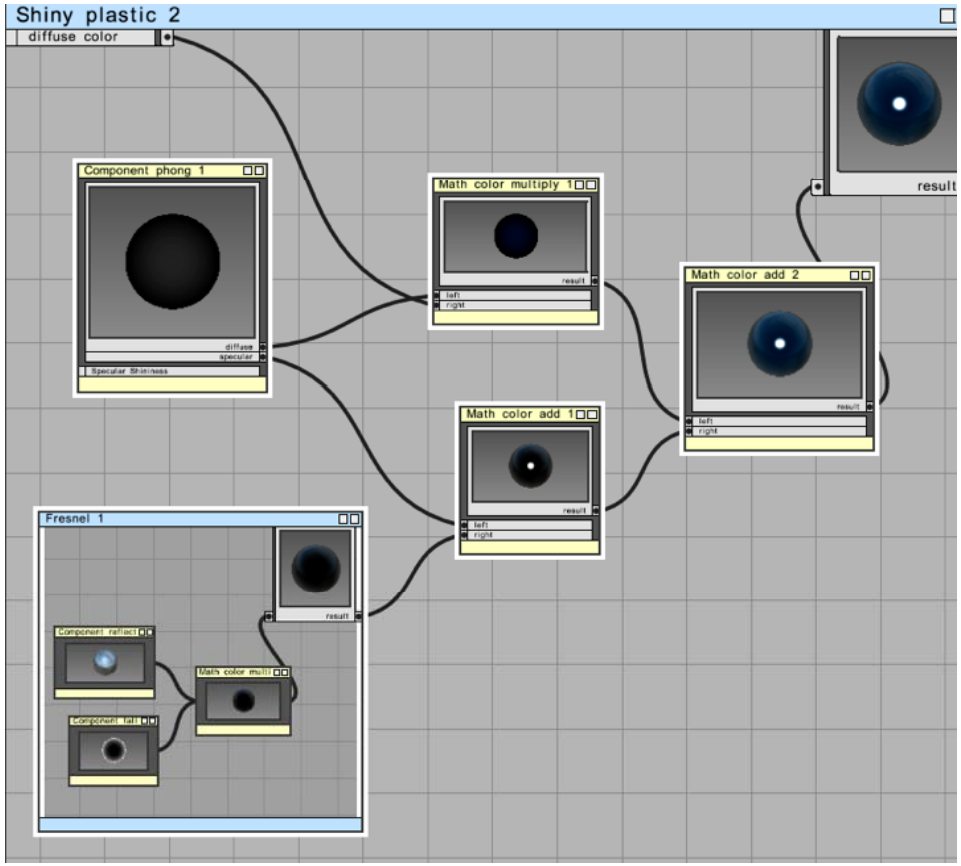
mental mill® & MetaSL™

Create a Phenomenon



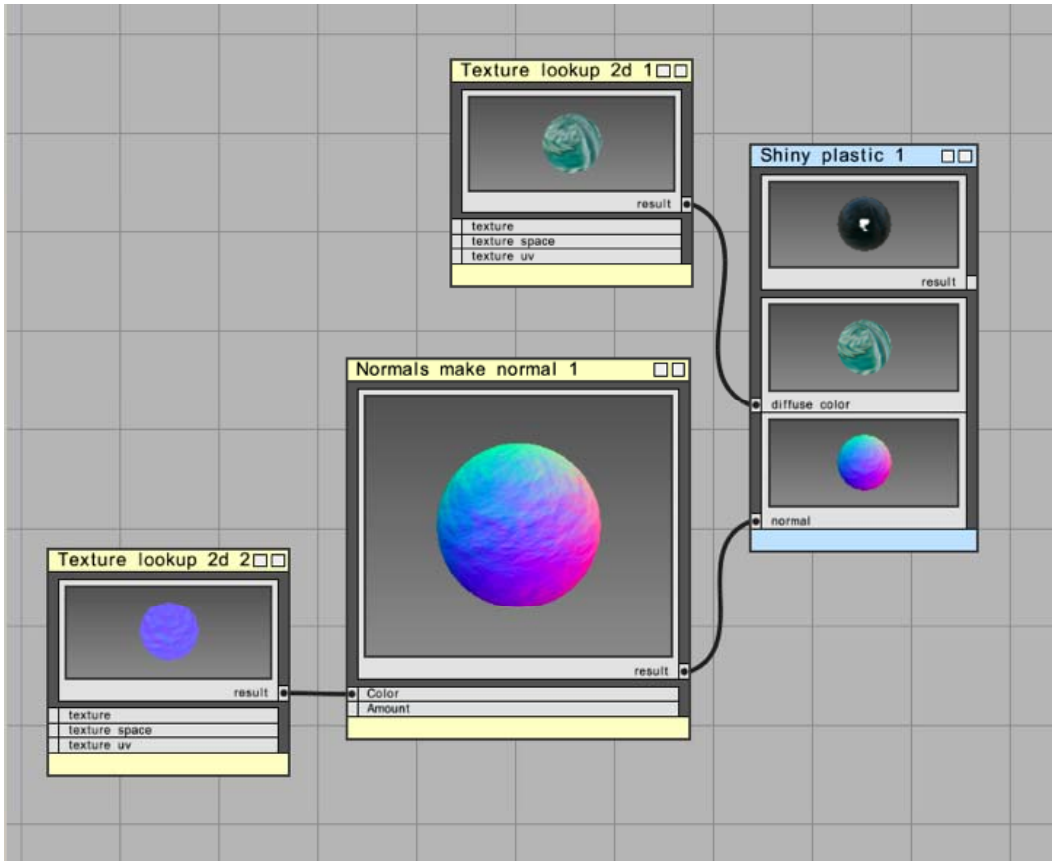
mental mill® & MetaSL™

Shiny Plastic Phenomenon



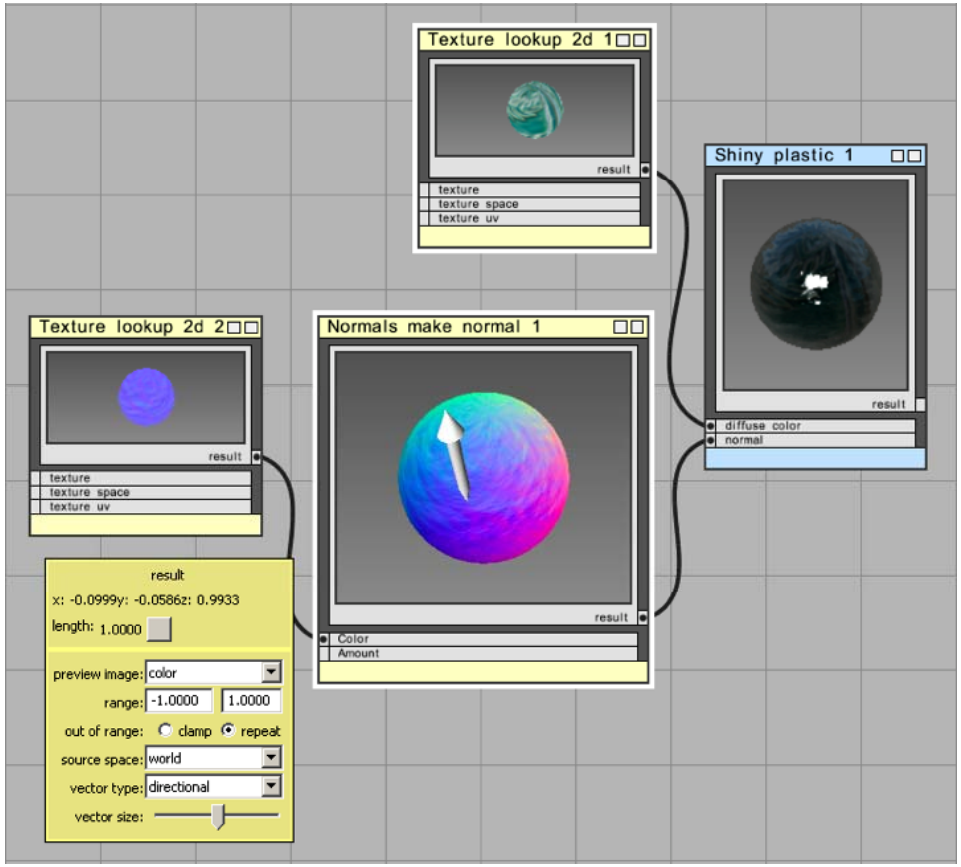
mental mill® & MetaSL™

Shiny Plastic Phenomenon in shader graph



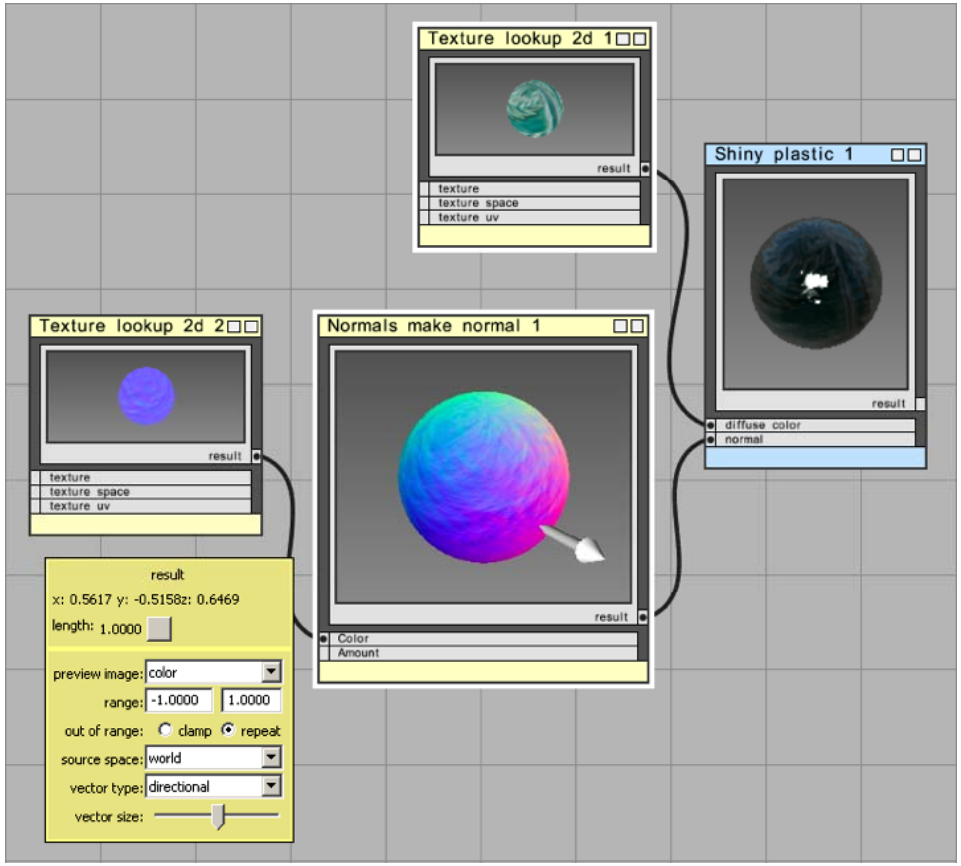
mental mill® & MetaSL™

Debugger ‘get info’



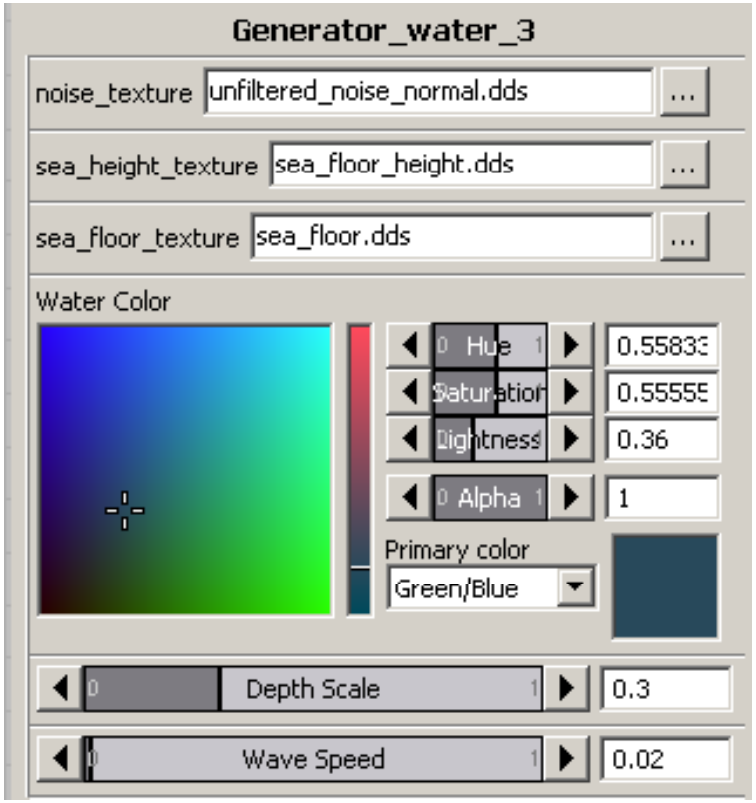
mental mill® & MetaSL™

Debugger ‘get info’



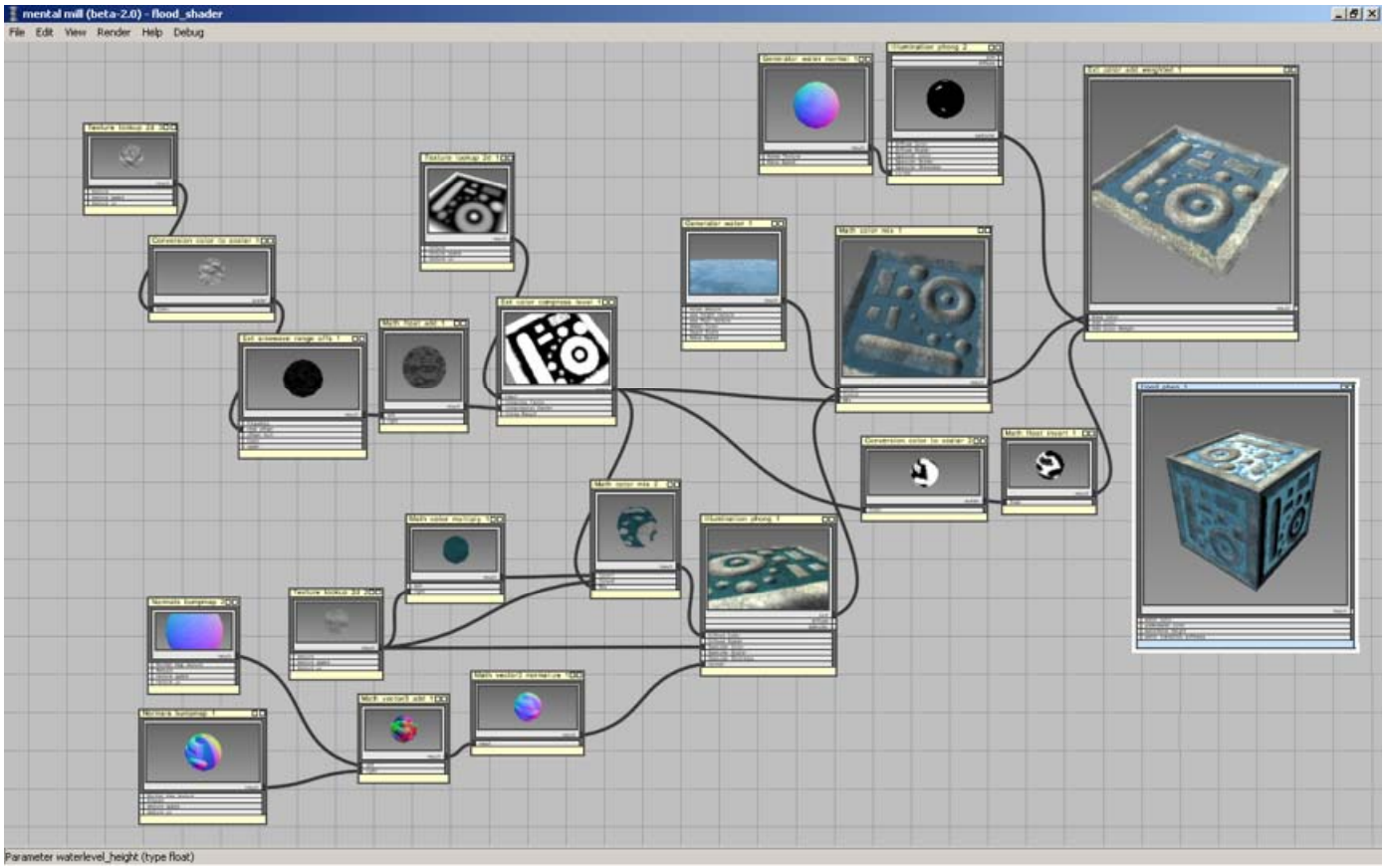
mental mill® & MetaSL™

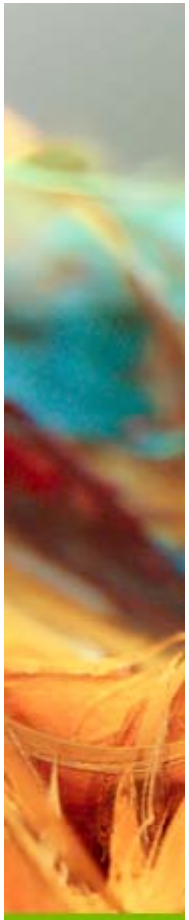
Generator_water shader



mental mill® & MetaSL™

Flood shader - using Generator_water



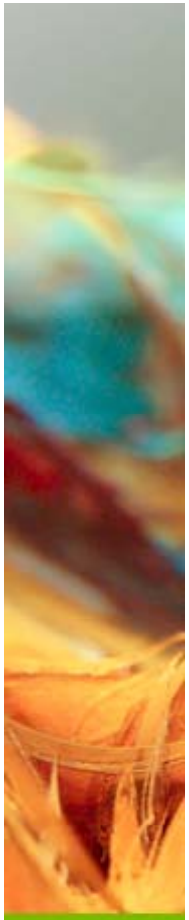


mental mill® & MetaSL™

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® & MetaSL™

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

mental mill® & MetaSL™

demonstration of mental mill workflow