Platform-independent shader development with mental mill®: the making of Dead Rising 2

Laura Scholl, mental images<sup>®</sup> Izmeth Siddeek, Blue Castle Games



## Platform-independent shader development with mental mill<sup>®</sup>: the making of Dead Rising 2



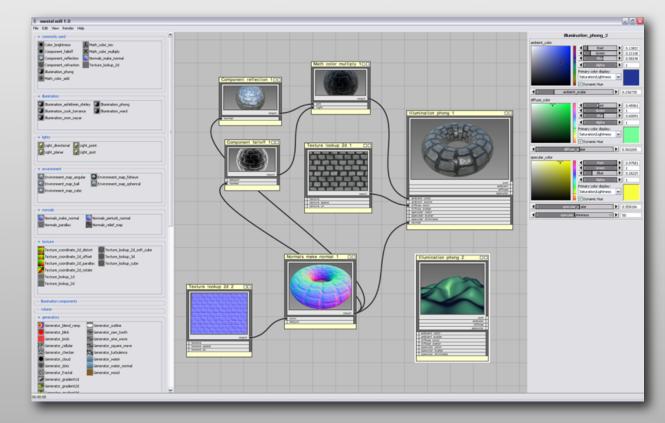


## mental mill: integrated visual development environment (IVDE)

Artists and shader writers can

- develop
- test
- manage

shaders written in MetaSL™





### MetaSL: high-level, platform-independent shading language

## Artists and shader writers can develop for

- every platform
- different contexts
- evolving rendering algorithms
- rapidly advancing GPUs

future-proof shader assets

47	
48	void main()
49	1
50	// Sample the height and center on 0.0 (scaled down too)
51	<pre>Scalar h = tex2D(height_tex, texture_coordinate[1].xy).x;</pre>
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 torward 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	<pre>// get the normal from the normal map</pre>
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	<pre>normal = normalize(n*tangent_space[0]);</pre>
68	
69	<pre>// diffuse and specular results from lighting</pre>
70	Color diffuse = Color(0,0,0,0);
71	Color specular = Color(0,0,0,0);
72	
73	<pre>// iterate over scene lights</pre>
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	<pre>specular += illum.z * light.contribution * specular_color;</pre>
82	)
83	

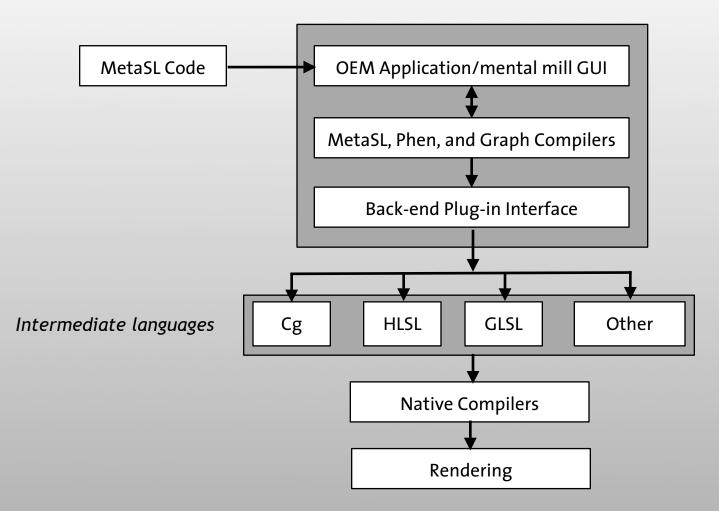


## mental mill is more than just another shader graph editor

As GPU's and rendering algorithms become more powerful it becomes increasingly more difficult and more costly to:

- create and maintain shader assets
- share shader assets between different media/content

Artists can build shader graphs that allow them to experiment and develop the look of their game assets independent of the shader programmers, without being limited by the implementation details.

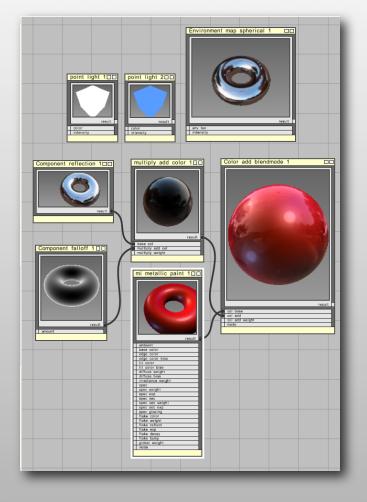


#### mental mill data flow overview

2009 NVIDIA Corporation



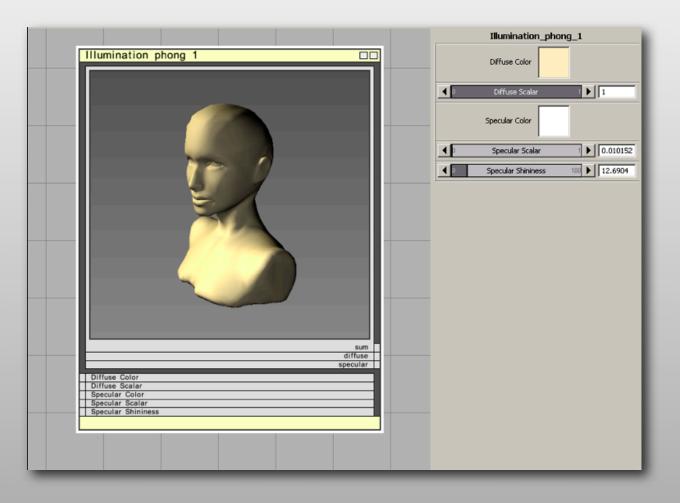
# Build shader graphs from the included library of Metanodes™



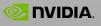
2009 NVIDIA Corporatio



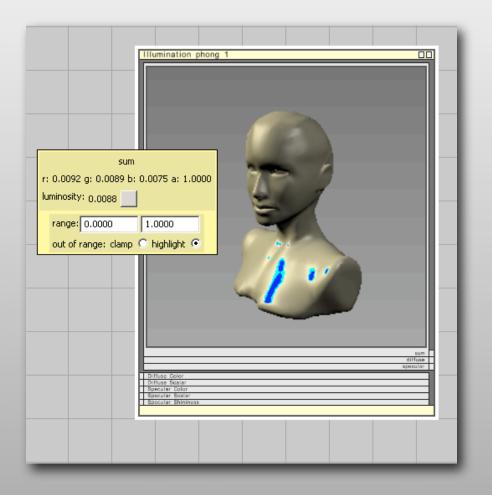
## Phong shader in MetaSL render



2009 NVIDIA Corporatio



## Debugger 'get info'



2009 NVIDIA Corporatio



Phong shader debug mode evaluate RS

iffuse Color d = diffuse_color * diffuse_scalar;	s. diffu
pecular Color s = specular_color * specular_scalar;	specul
necular Scalar	
ormal alize components	temp
Color temp = Rd + Rs;	Rd
Scalar maxc = max(max(temp.r,temp.g),temp.b);	
if (maxc>1.0) {	
Rd $/=$ maxc;	
Rs $/=$ maxc;	
T	
Vector3 vdir = direction;	Rs
vectors var = direction;	maxc
<pre>// enumerate lights</pre>	
// enumerate lights Light_iterator light;	
foreach (light)	
{	
<pre>Scalar cos = saturate(light.dot_nl);</pre>	
if (cos > 0.0) {	diffuse
diffuse += (cos / PI) * light.contribution;	diffuse_color
	specular_shinine
<pre>Scalar s = mi_phong_specular(light.direction,</pre>	
<pre>vdir, normal, specular_shininess);</pre>	
<pre>specular += (s * cos) * light.contribution;</pre>	
}	
}	
diffuse *= Rd;	specular
<pre>specular *= Rs;</pre>	diffuse_scalar
	specular_scalar sum
// irradiance term	specular_color
Irradiance_options irradiance_options;	direction
<pre>irradiance_options.set_importance(diffuse_scalar);</pre>	normal
diffuse += Rd/PI * irradiance(irradiance_options);	
diffuse.a = 1.0;	
<pre>specular.a = 1.0;</pre>	
<pre>sum = diffuse + specular;</pre>	

2009 NVIDIA Corporatio



Phong shader debug mode evaluate specular

Illumination phong 1	
Diffuse Color d = diffuse_color * diffuse_scalar;	sum
Diffuse Scalar	diffuse •
Specular Color s = specular_color * specular_scalar; Specular Scalar	specular •
Specular Statian Snecular Shininess	s
Specular Shininess normal alize components	cos
Color temp = Rd + Rs;	+ light
<pre>Scalar maxc = max(max(temp.r,temp.g),temp.b);</pre>	temp
if (maxc>1.0) {	Rd
Rd $/=$ maxc;	vdir
Rs /= maxc;	
}	
Vector3 vdir = direction;	
// enumerate lights	
Light iterator light;	Rs
	maxc
foreach (light)	
1	
Scalar cos = saturate(light.dot_nl);	Carolina 14
if $(\cos > 0.0)$ {	
diffuse += (cos / PI) * light.contribution;	
Scalar s = mi phong specular(light.direction,	
	diffuse
<pre>vdir, normal, specular_shininess);</pre>	diffuse_color
<pre>specular += (s * cos) * light.contribution;</pre>	specular_shininess
}	
}	
diffuse *= Rd;	
<pre>specular *= Rs;</pre>	
// irradiance term	
Irradiance options irradiance options;	specular diffuse scalar
	specular_scalar
<pre>irradiance_options.set_importance(diffuse_scalar);</pre>	sum
diffuse += Rd/PI * irradiance(irradiance_options);	specular_color
	direction normal
diffuse.a = 1.0;	nonnar
<pre>specular.a = 1.0;</pre>	
<pre>sum = diffuse + specular;</pre>	
oum - diride + shecuidi'	

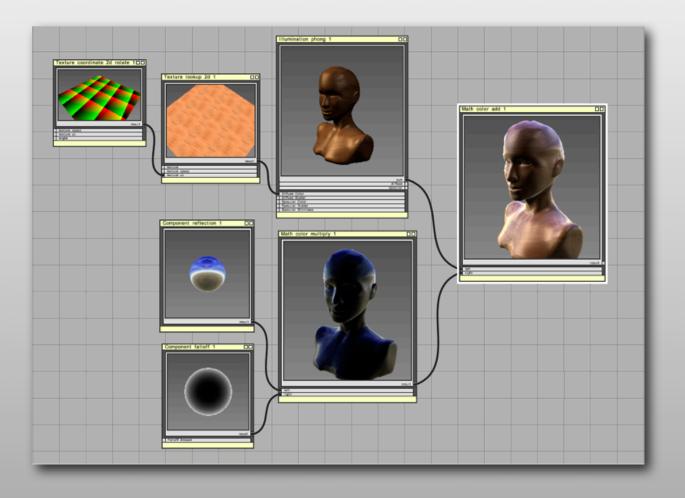


Phong shader debug mode specular\*RS

Illumination phong 1	
• Diffuse Color d = diffuse_color * diffuse_scalar;	sum
Diffuse Scalar	diffuse •
Specular Color s = specular_color * specular_scalar;	specular •
Specular Scalar	+ light
Specular Shininess • normal alize components	temp
Color temp = Rd + Rs;	Rd
<pre>Scalar maxc = max(max(temp.r,temp.g),temp.b);</pre>	vdir
if (maxc>1.0) {	
Rd /= maxc;	
Rs /= maxc;	
}	
Vector3 vdir = direction;	
rectors full - diffection,	Rs
// enumerate lights	maxc
// enumerate lights	
Light_iterator light;	
foreach (light)	(10 Carlos
f	
<pre>Scalar cos = saturate(light.dot_nl);</pre>	
if $(\cos > 0.0)$ {	
diffuse += (cos / PI) * light.contribution;	
	diffuse
Scalar s = mi phong specular(light.direction,	diffuse_color
vdir, normal, specular shininess);	specular_shininess
<pre>specular += (s * cos) * light.contribution;</pre>	
}	
1	
diffuse *= Rd;	
<pre>specular *= Rs;</pre>	specular
	diffuse_scalar
// irradiance term	specular_scalar
Irradiance_options irradiance_options;	sum
<pre>irradiance_options.set_importance(diffuse_scalar);</pre>	specular_color direction
diffuse += Rd/PI * irradiance(irradiance options);	normal
	-
diffuse.a = 1.0;	
specular.a = $1.0;$	
<pre>specular:a = 1.0; sum = diffuse + specular;</pre>	
Jum - Gillube + Specular,	



## Add a texture and a reflection to the Phong node



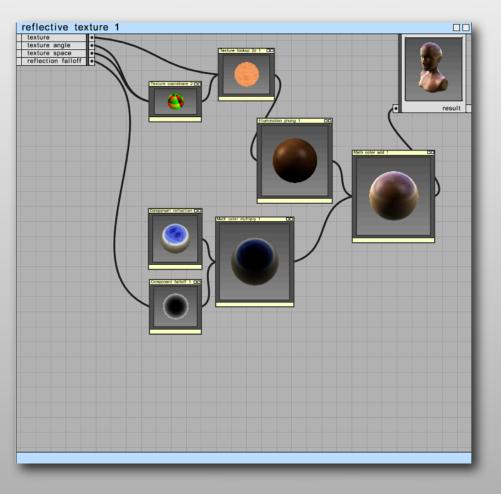
2009 NVIDIA Corporatio



## Create a Phenomenon from the Phong shader graph

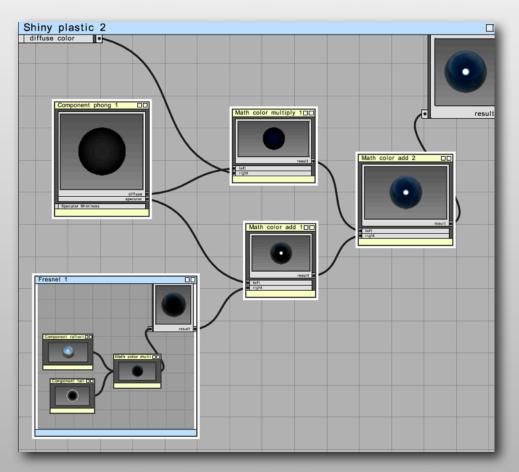
Encapsulate a shader graph to:

- simplify attributes
- protect a developed look
- reuse in creating shader graphs





## Reuse a Phenomenon inside another Phenomenon



2009 NVIDIA Corporation.



## mental ray<sup>®</sup> preview renderer plug-in



2009 NVIDIA Corporatio



## Platform-independent shader development with mental mill<sup>®</sup>: the making of Dead Rising 2



2009 NVIDIA Corporation



## What version of mental mill do I need?

Artist Edition: bundled with Autodesk 3ds Max®

- \* Metanode library to build shader graphs and Phenomena
- ★ 3ds Max viewport and mental ray rendering





## What version of mental mill do I need?

Standard Edition: available from mental images

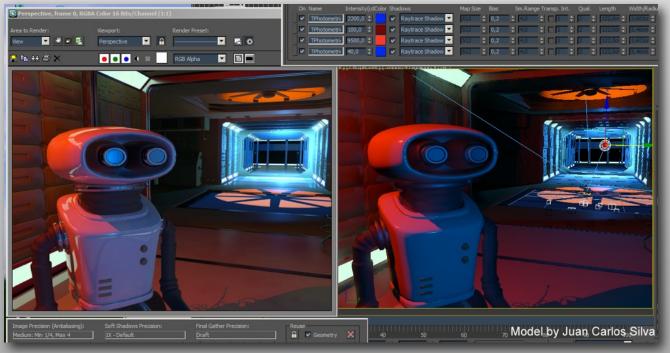
- ★ Everything in Artist Edition
- Integrated shader editing and visual debugging tools
- Customizable back-end formats for CgFX, HLSL, and GLSL
- \* Back-end plug-ins for Maya, Softimage, CATIA, and FX Composer
- mental ray preview plug-in



## What version of mental mill do I need?

Integrator Edition: available from mental images

- \* Component mental mill API libraries for integration into:
  - Design and DCC applications
  - shader pipelines





How to learn more about mental mill

Online:

- Website: http://www.mentalimages.com
- \* Forum: http://forum.mentalimages.com

During GDC:

★ Expo Suite 656, West Hall



