

SAN ANTONIO

SIGGRAPH

2002

SAN ANTONIO  
**SIGGRAPH**  
**2002**

# **Hardware Shading on the ATI RADEON™ 9700**

**Jason L. Mitchell**  
**ATI Research**  
**JasonM@ati.com**

# Outline



- **RADEON™ 9700 Shader Models**
  - 2.0 vertex and pixel shaders in DirectX® 9
  - ATI\_fragment\_program OpenGL Extension
- **Compulsories**
  - Shiny bumpy
  - Homomorphic BRDF
  - Procedural wood
- **Freestyle**
  - Per-Pixel Hatching
  - High Dynamic Range Rendering
    - HDR Environment/Light Maps
    - HDR Scene Post-processing
    - Local versus distant reflections / refractions
  - Motion Blur
  - Image Space Operations for NPR
  - Two-tone layered car paint model



# RADEON™ 9700

## Vertex Shaders



- **Exposed as DirectX® 9 2.0 Vertex Shaders and via ARB\_vertex\_program**
- **Improves upon previous models**
  - **Control flow**
  - **Longer programs**
  - **More constant storage**



# Vertex Shader Control Flow

SAN ANTONIO  
**SIGGRAPH**  
#2002#

- **Jumps, loops**
- **Useful in game space for solving the “permutation problem”**
  - Number/type of lights
  - Env mapping on/off
  - Bump mapping on/off
  - Skinning on/off
- **Constant based for this generation**



# RADEON™ 9700

## Pixel Shaders



- **DirectX® 9 2.0 pixel shaders**
- **ATI\_fragment\_program**
- **Floating point pixels**
- **64 ALU instructions**
- **32 texture instructions**
- **4 levels of dependent read**



# 2.0 Pixel Shader Instruction Set



- **ALU Instructions**
  - **ADD, MOV, MUL, MAD, DP3, DP4, FRAC, RCP, RSQ, EXP, LOG and CMP**
- **Texture Instructions**
  - **texld, texldp, texldb, texkill**



# Multiple render targets

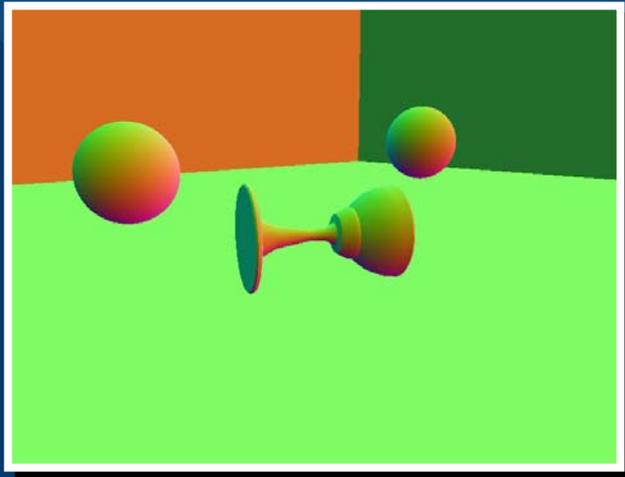
- **Output up to four colors from pixel shader**
- **Useful for intermediate results in multipass algorithms**
- **Can use as G-buffer [Saito and Takahashi 1990]**
  - **Used to optimize NPR outlining example**



# Pixel Pipeline Output



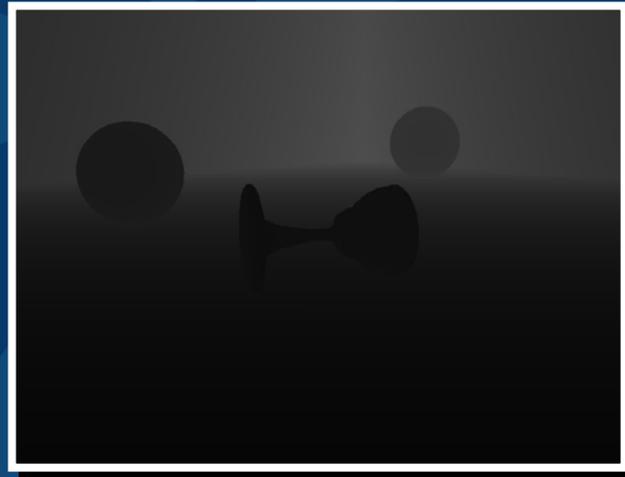
**Target 1**



**World Space Normal**



**Target 2**

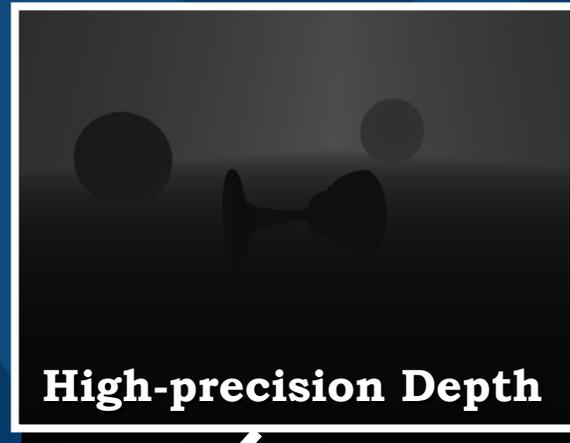
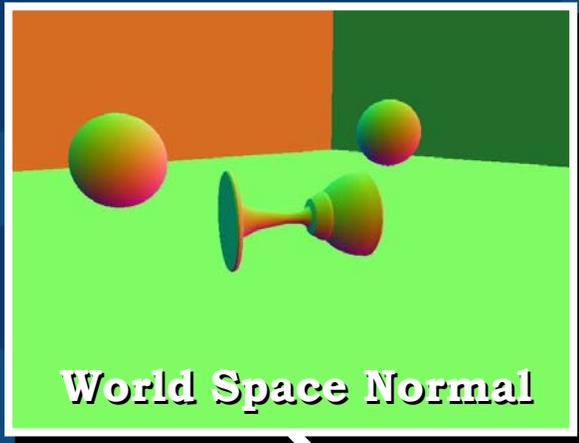


**High-precision Depth**

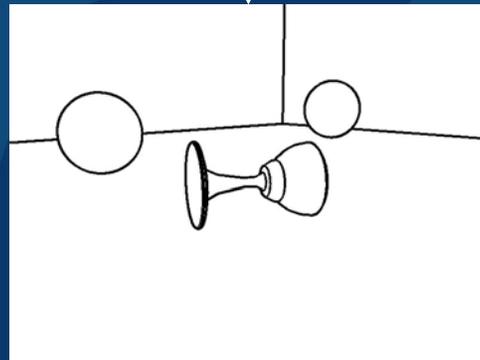


# Texture 1

# Texture 2



Pixel  
Shader



**High Precision  
depth gives better  
edges than low-  
precision depth  
used previously**



# Advanced Surface Types



- **IEEE 32-bit surfaces**
  - 1-, 2- and 4-channel versions
- **16-bit float s10e5 surfaces**
  - 1-, 2- and 4-channel versions
- **16-bit fixed point surfaces**
  - 1-, 2- and 4-channel versions
- **sRGB**
  - 2.2 gamma



# Compulsory Shaders



- **Shiny Bumpy**
  - ps.1.4
- **Homomorphic BRDF**
  - ATI\_fragment\_shader
- **Procedural Wood**
  - ps.2.0



# Shiny Bumpy

SAN ANTONIO  
**SIGGRAPH**  
#2002#

- **Shown in Treasure Chest demo on RADEON™ 8500**
- **Uses ps.1.4**
- **Transforms fetched normal to world space and performs reflection operation**
- **Samples cube map with reflected vector**

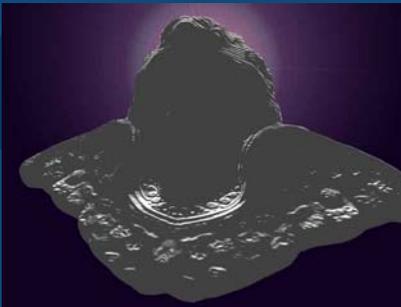


# More than just shiny and bumpy with ps.1.4



- **Easy to include a per-pixel Fresnel term and sample a diffuse cube map**
- **Runs in one pass on ps.1.4**

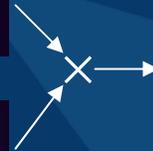




×



×



+



# Homomorphic BRDFs

SAN ANTONIO  
SIGGRAPH  
#2002#

- **Work is in the parametrization**
- **Reconstruction is one of:**
  - $(((\text{diffuse} * \text{tex0}) * \text{scale1}) * \text{tex1} * \text{scale2}) * \text{tex2}$
  - $(((\text{diffuse} * \text{tex0}) * \text{scale1}) * \text{tex1} * \text{scale2}) * \text{tex2} + \text{tex3}$
  - $(((\text{diffuse} * \text{tex0}) * \text{scale1}) * \text{tex1} * \text{scale2}) * \text{tex2}) * \text{tex3}$



Hardware Shading on ATI RADEON™ 9700



# Homomorphic BRDFs



```
glBeginFragmentShaderATI();
glSampleMapATI (GL_REG_0_ATI, GL_TEXTURE0_ARB, GL_SWIZZLE_STR_ATI); // Sample maps
glSampleMapATI (GL_REG_1_ATI, GL_TEXTURE1_ARB, GL_SWIZZLE_STR_ATI);
glSampleMapATI (GL_REG_2_ATI, GL_TEXTURE2_ARB, GL_SWIZZLE_STR_ATI);
if (param == PARAM_OHI_H) // only sample the specular map if necessary
{
    if (sg_tex3Type == GL_TEXTURE_CUBE_MAP_ARB) {
        glSampleMapATI (GL_REG_3_ATI, GL_TEXTURE3_ARB, GL_SWIZZLE_STR_ATI); }
    else {
        glSampleMapATI (GL_REG_3_ATI, GL_TEXTURE3_ARB, GL_SWIZZLE_STQ_ATI); }
}
// r0 = diffuse * tex0 * scale1
glColorFragmentOp2ATI (GL_MUL_ATI, GL_REG_0_ATI, GL_NONE, scale1,
                       GL_PRIMARY_COLOR_EXT, GL_NONE, GL_NONE,
                       GL_REG_0_ATI, GL_NONE, GL_NONE);

// r0 = (diffuse * tex0 * scale1) * tex1 * scale2
glColorFragmentOp2ATI (GL_MUL_ATI, GL_REG_0_ATI, GL_NONE, scale2,
                       GL_REG_1_ATI, GL_NONE, GL_NONE,
                       GL_REG_0_ATI, GL_NONE, GL_NONE);

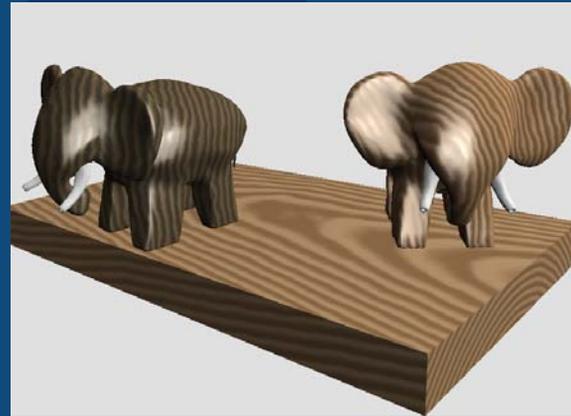
if (param == PARAM_OHI_H) // do a MAD if specular map is used
{
    // r0 = ((diffuse * tex0 * scale1) * tex1 * scale2) * tex2 + tex3
    glColorFragmentOp3ATI (GL_MAD_ATI, GL_REG_0_ATI, GL_NONE, GL_NONE,
                           GL_REG_0_ATI, GL_NONE, GL_NONE,
                           GL_REG_2_ATI, GL_NONE, GL_NONE,
                           GL_REG_3_ATI, GL_NONE, GL_NONE);
} else {
    // r0 = ((diffuse * tex0 * scale1) * tex1 * scale2) * tex2
    glColorFragmentOp2ATI (GL_MUL_ATI, GL_REG_0_ATI, GL_NONE, GL_NONE,
                           GL_REG_0_ATI, GL_NONE, GL_NONE,
                           GL_REG_2_ATI, GL_NONE, GL_NONE);
}

glEndFragmentShaderATI();
```



# Two versions of wood

- Hand coded ps.2.0 assembly
- RenderMan translated to ps.2.0 assembly



# Procedural Wood

- **Based on example in *Advanced RenderMan***
- **Uses volume texture for noise and 1D texture for smooth pulse train**
- **My version has 8 intuitive parameters**
  - Light Wood Color
  - Dark Wood
  - ring frequency
  - noise amplitude
  - trunk wobble frequency
  - trunk wobble amplitude
  - specular exponent scale
  - specular exponent bias

Non-Real-Time Version

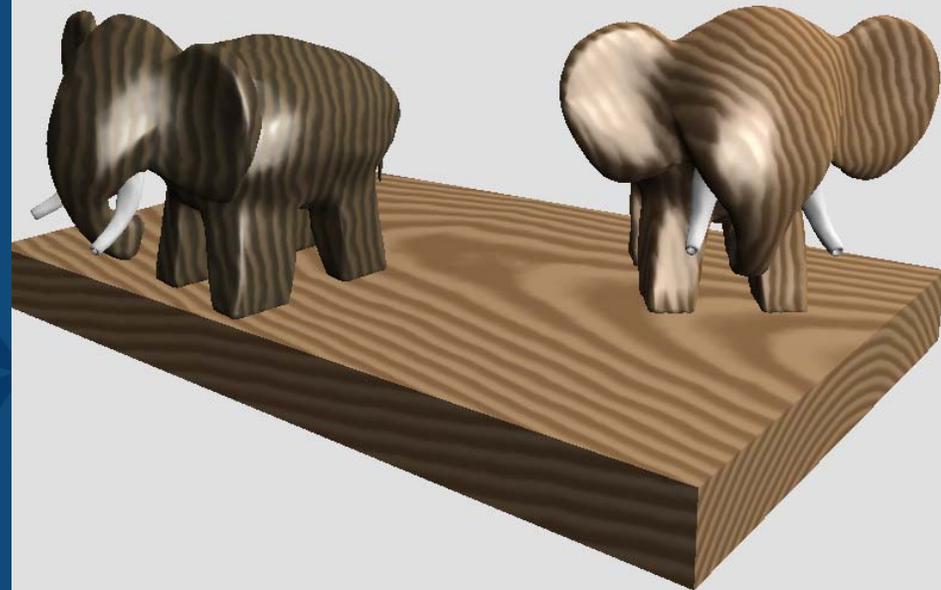


# Procedural Wood

SAN ANTONIO  
SIGGRAPH  
#2002#

- 40-instruction 2.0 pixel shader
- Samples noise map 6 times
- Phong shading

Real-Time Version



# Step-by-step Approach

- **Shader Space ( $P_{shade}$ )**
- **Distance from trunk axis ( $z$ )**
- **Add noise to  $P_{shade}$**
- **Add noise as function of  $z$  to wobble**
- **Run through pulse train**



# Wood Vertex Shader



```
dcl_position v0
dcl_normal   v3

def c40, 0.0f, 0.0f, 0.0f, 0.0f // All zeroes
m4x4 oPos, v0, c[0]           // Transform position to clip space

m4x4 r0, v0, c[17]           // Transformed Pshade (using texture matrix 0)
mov oT0, r0
m4x4 oT1, v0, c[21]         // Transformed Pshade (using texture matrix 1)
m4x4 oT2, v0, c[25]         // Transformed Pshade (using texture matrix 2)

mov r1, c40
mul r1.x, r0.z, c29.x       // {freq*Pshade.z, 0, 0, 0}
mov oT3, r1                 // {freq*Pshade.z, 0, 0, 0} for 1D trunkWobble noise in x
mov r1, c40
mad r1.x, r0.z, c29.x, c29.y // {freq*Pshade.z + 0.5, 0, 0, 0}
mov oT4, r1                 // {Pshade.z+0.5, 0, 0, 0} for 1D trunkWobble noise in y

m4x4 oT6, v0, c[4]         // Transform position to eye space
m4x4 oT7, v3, c[8]        // Transform normal to eye space
```



# Pshade

- For this app, *Pshade* is just world space
- The infinite virtual log runs along the *z* axis
- I make a few different transformed versions of *Pshade* in the vertex shader in order to turn scalar noise into color noise, as I'll show later



# Distance from z axis

SAN ANTONIO  
SIGGRAPH  
#2002#

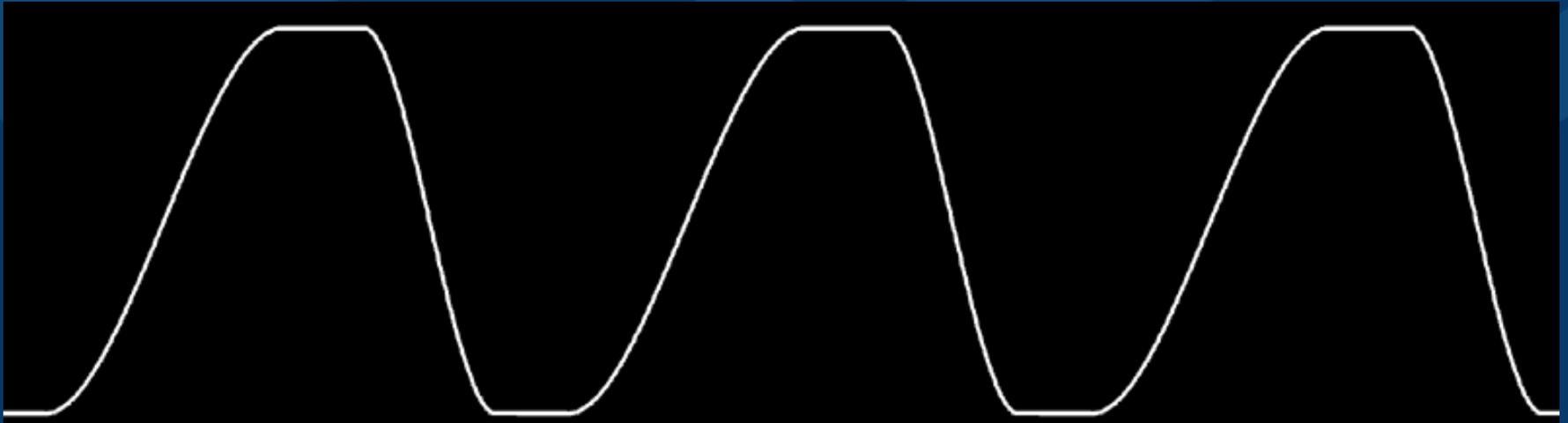
- $\text{sqrt} (P_{\text{shade}} \cdot x^2 + P_{\text{shade}} \cdot y^2) * \text{freq}$
- Pass this in to pulse train



# Pulse Train

SAN ANTONIO  
SIGGRAPH  
#2002#

- Tuned to mimic the way colors mix in real wood
- One pulse stored in 1D texture which repeats



# Concentric Rings



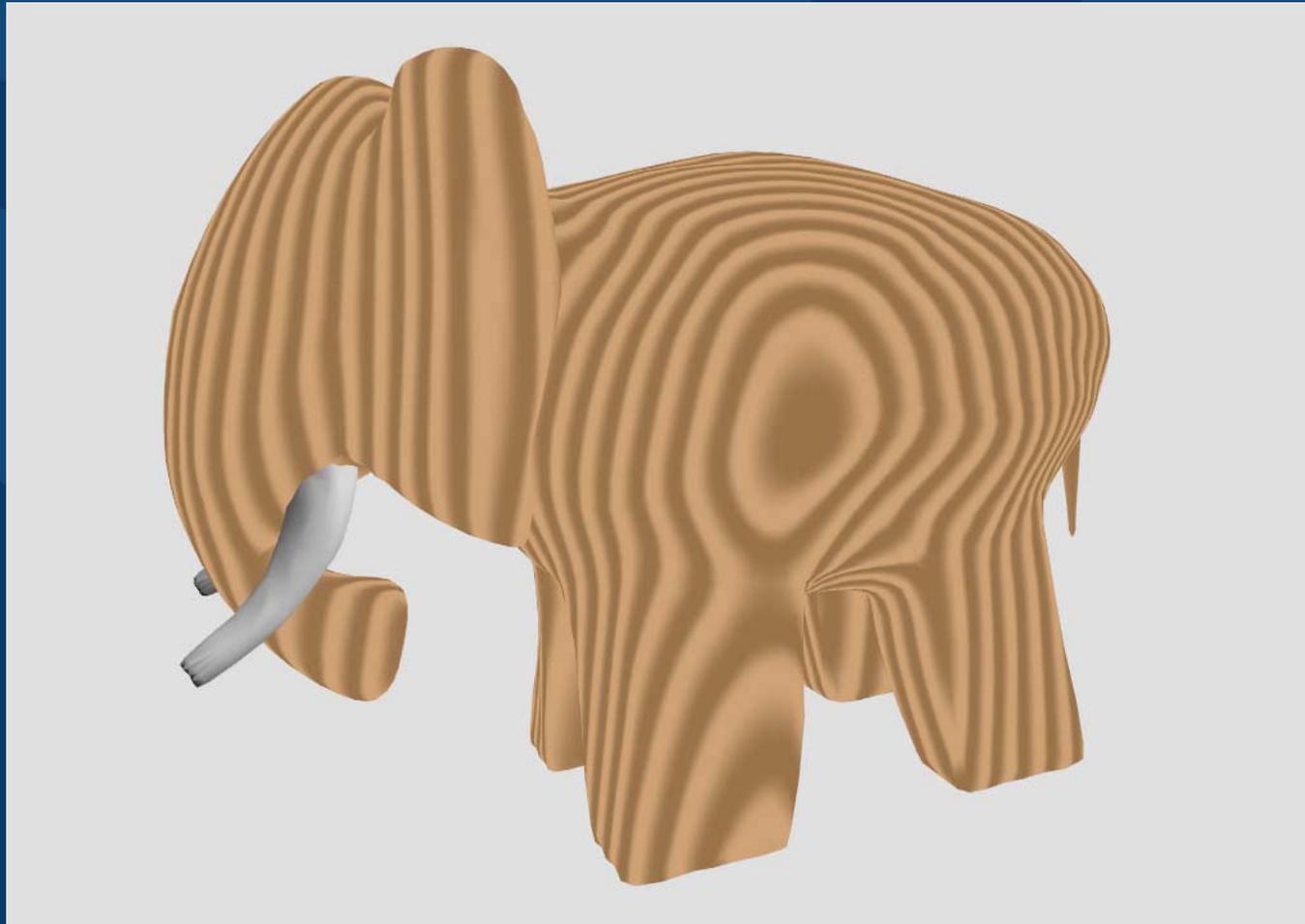
```
ps.2.0
def c0, 2.0f, -1.0f, 0.5f, 0.5f // scale, bias, half, X
def c1, 1.0f, 1.0f, 0.1f, 0.0f // X, X, 0.1, zero
// c2: xyz == Light Wood Color, w == ringFreq
// c3: xyz == Dark Wood Color

dcl t0.xyzw // xyz == Pshade (shader-space position), w == X
dcl_2d s1 // 1D smooth step function
dp2add r0, t0, t0, c1.w // x2 + y2 + 0
rsq r0, r0.x // 1/sqrt(x2 + y2)
rcp r0, r0.x // sqrt(x2 + y2)
mul r0, r0, c2.w // sqrt(x2 + y2) * freq
texld r0, r0, s1 // Sample from 1D pulse train texture
mov r1, c3
lrp r2, r0.x, c2, r1 // Blend between light and dark wood colors
mov oC0, r2
```



# Concentric Rings

SAN ANTONIO  
SIGGRAPH  
#2002#



Hardware Shading on ATI RADEON™ 9700



# Noisy Rings

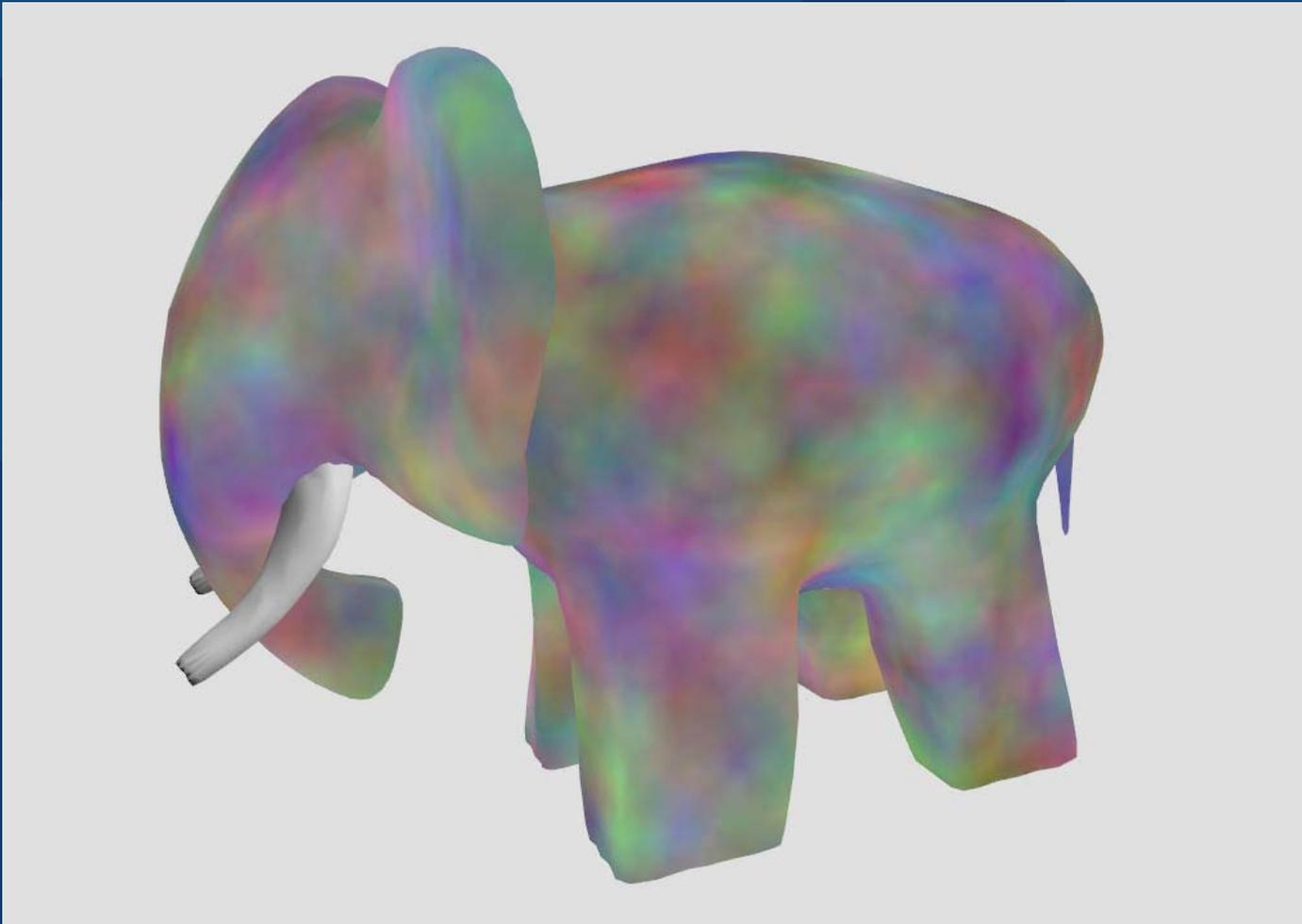
```
ps.2.0
def c0, 2.0f, -1.0f, 0.5f, 0.5f // scale, bias, half, X
def c1, 1.0f, 1.0f, 0.1f, 0.0f // X, X, 0.1, zero
// c2: xyz == Light Wood Color, w == ringFreq
// c3: xyz == Dark Wood Color, w == noise amplitude
// c4: xyz == L_eye, w == trunkWobbleAmplitude
dcl t0.xyzw // xyz == Pshade (shader-space position), w == X
dcl t1.xyzw // xyz == Perturbed Pshade, w == X
dcl t2.xyzw // xyz == Perturbed Pshade, w == X
dcl_volume s0 // Luminance-only Volume noise
dcl_2d s1 // 1D smooth step function
texld r3, t0, s0 // Sample dX from scalar noise at Pshade
texld r4, t1, s0 // Sample dY from scalar noise at perturbed Pshade
texld r5, t2, s0 // Sample dZ from scalar noise at perturbed Pshade
mov r3.y, r4.x // Put dY in y
mov r3.z, r5.x // Put dZ in z
mad r3, r3, c0.x, c0.y // Put noise in -1..+1 range
mad r7, c3.w, r3, t0 // Scale by amplitude and add to Pshade to warp the domain
dp2add r0, r7, r7, c1.w // x2 + y2 + 0
rsq r0, r0.x // 1/sqrt(x2 + y2)
rcp r0, r0.x // sqrt(x2 + y2)
mul r0, r0, c2.w // sqrt(x2 + y2) * freq
texld r0, r0, s1 // Sample from 1D pulse train texture
mov r1, c3
lrp r2, r0.x, c2, r1 // Blend between light and dark wood colors
mov oC0, r2
```

New code



# Colored Volume Noise

SAN ANTONIO  
SIGGRAPH  
#2002#

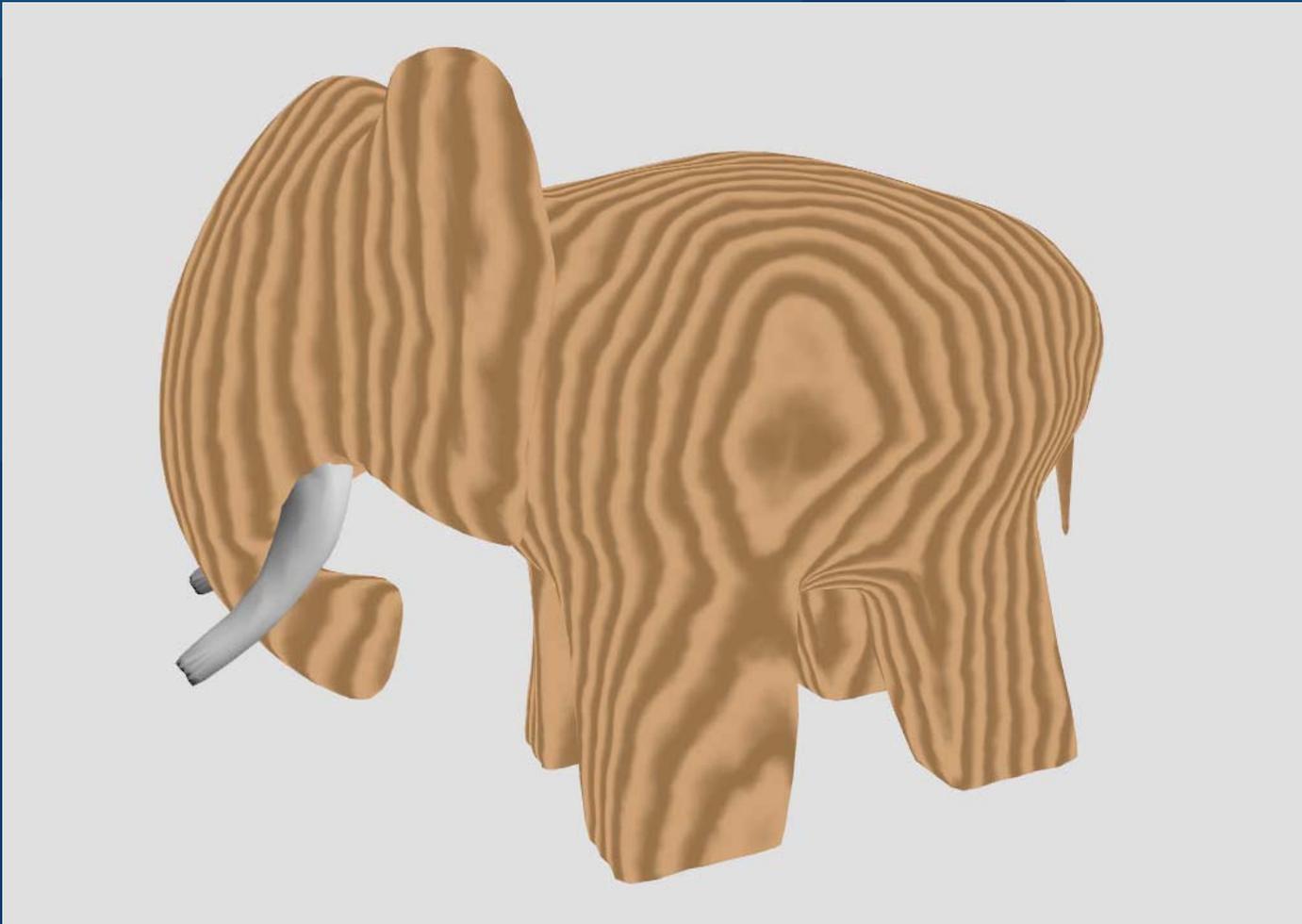


Hardware Shading on ATI RADEON™ 9700



# Noisy Rings

SAN ANTONIO  
SIGGRAPH  
#2002#

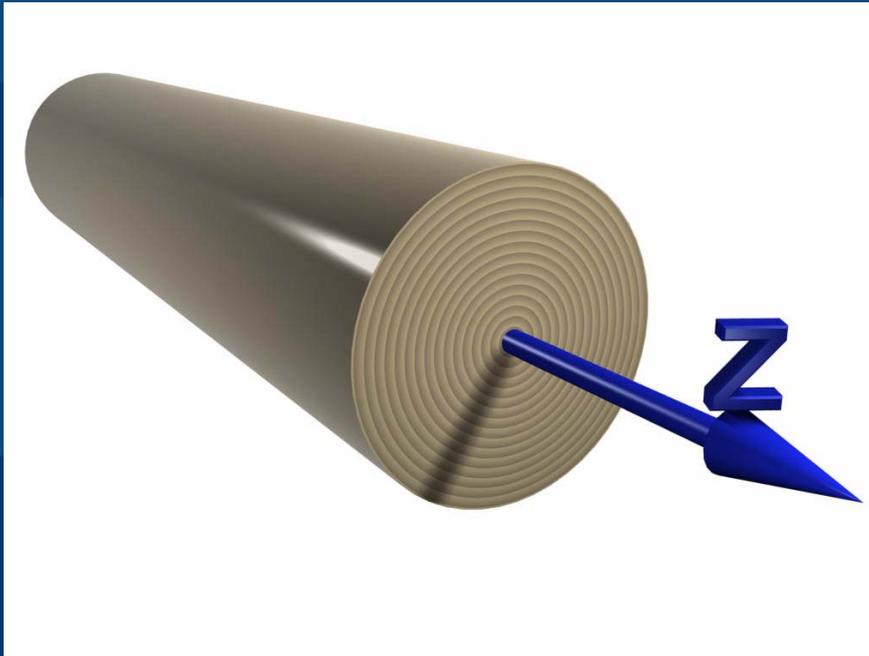


Hardware Shading on ATI RADEON™ 9700

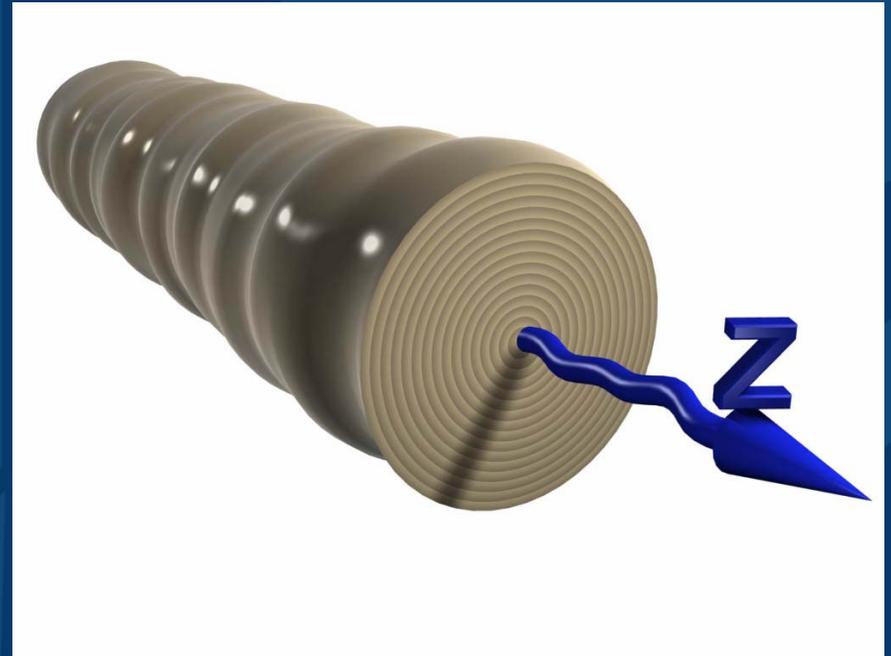


# Trunk Wobble

SAN ANTONIO  
SIGGRAPH  
#2002#



Without Wobble



With Wobble



# Noise and wobble



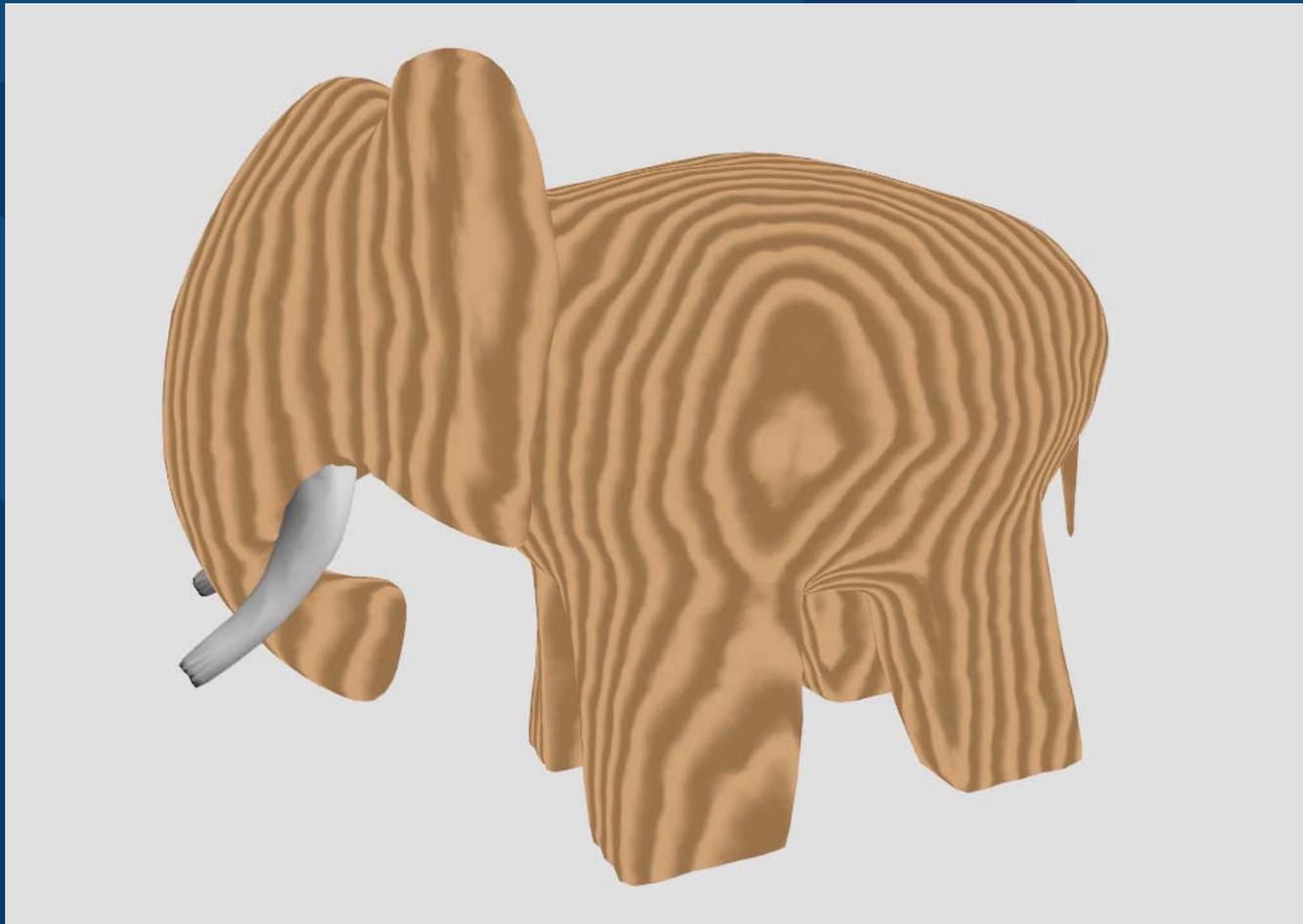
**New code**

```
def c0, 2.0f, -1.0f, 0.5f, 0.5f // scale, bias, half, X
def c1, 1.0f, 1.0f, 0.1f, 0.0f // X, X, 0.1, zero
// c2: xyz == Light Wood Color, w == ringFreq
// c3: xyz == Dark Wood Color, w == noise amplitude
// c4: xyz == L_eye, w == trunkWobbleAmplitude
dcl t0.xyzw // xyz == Pshade (shader-space position), w == X
dcl t1.xyzw // xyz == Perturbed Pshade, w == X
dcl t2.xyzw // xyz == Perturbed Pshade, w == X
dcl t3.xyzw // xyz == {Pshade.z, 0, 0}, w == X
dcl t4.xyzw // xyz == {Pshade.z + 0.5, 0, 0}, w == X
dcl_volume s0 // Luminance-only Volume noise
dcl_2d s1 // 1D smooth step function (blend factor in x, spec exp in y, ...)
texld r3, t0, s0 // Sample dX from scalar volume noise texture at P_shade
texld r4, t1, s0 // Sample dY from scalar volume noise texture at perturbed P_shade
texld r5, t2, s0 // Sample dZ from scalar volume noise texture at perturbed P_shade
texld r6, t3, s0 // Sample trunkWobble.x from scalar noise at {P_shade, 0, 0}
texld r7, t4, s0 // Sample trunkWobble.y from scalar noise at {P_shade + 0.5, 0, 0}
mov r3.y, r4.x // Put dY in y
mov r3.z, r5.x // Put dZ in z
mov r6.y, r7.x // Move to get {trunkWobble.x, trunkWobble.y, 0}
mad r6, r6, c0.x, c0.y // Put {trunkWobble.x, trunkWobble.y, 0} in -1..+1 range
mad r3, r3, c0.x, c0.y // Put noise in -1..+1 range
mad r7, c3.w, r3, t0 // Scale noise by amplitude and add to P_shade to warp the domain
mad r7, c4.w, r6, r7 // Scale {trunkWobble.x, trunkWobble.y, 0} by amplitude and add in
dp2add r0, r7, r7, c1.w // x^2 + y^2 + 0
rsq r0, r0.x // 1/sqrt(x^2 + y^2)
rcp r0, r0.x // sqrt(x^2 + y^2)
mul r0, r0, c2.w // sqrt(x^2 + y^2) * freq
texld r0, r0, s1 // Sample from 1D pulse train texture
mov r1, c3
lrp r2, r0.x, c2, r1 // Blend between light and dark wood colors
mov oC0, r2
```



# Noise and Wobble

SAN ANTONIO  
SIGGRAPH  
#2002#



Hardware Shading on ATI RADEON™ 9700



```

ps.2.0
def c0, 2.0f, -1.0f, 0.5f, 0.5f // scale, bias, half, X
def c1, 1.0f, 1.0f, 0.1f, 0.0f // X, X, 0.1, zero
dcl t0.xyzw // xyz == Pshade (shader-space position), w == X
dcl t1.xyzw // xyz == Perturbed Pshade, w == X
dcl t2.xyzw // xyz == Perturbed Pshade, w == X
dcl t3.xyzw // xyz == {Pshade.z, 0, 0}, w == X
dcl t4.xyzw // xyz == {Pshade.z + 0.5, 0, 0}, w == X
dcl t6.xyzw // xyz == P_eye, w == X
dcl t7.xyzw // xyz == N_eye, w == X
dcl_volume s0 // Luminance-only Volume noise
dcl_2d s1 // 1D smooth step function (blend factor in x, specular exponent in y,
    ...)
texld r3, t0, s0 // Sample dX from scalar volume noise texture at P_shade
texld r4, t1, s0 // Sample dY from scalar volume noise texture at perturbed P_shade
texld r5, t2, s0 // Sample dZ from scalar volume noise texture at perturbed P_shade
texld r6, t3, s0 // Sample trunkWobble.x from scalar volume noise at {P_shade.z, 0, 0}
texld r7, t4, s0 // Sample trunkWobble.y from scalar volume noise at {P_shade.z + 0.5, 0, 0}
mov r3.y, r4.x // Put dY in y
mov r3.z, r5.x // Put dZ in z
mov r6.y, r7.x // Move to get {trunkWobble.x, trunkWobble.y, 0}
mad r6, r6, c0.x, c0.y // Put {trunkWobble.x, trunkWobble.y, 0} in -1..+1 range
mad r3, r3, c0.x, c0.y // Put noise in -1..+1 range
mad r7, c3.w, r3, t0 // Scale noise by amplitude and add to P_shade to warp the domain
mad r7, c4.w, r6, r7 // Scale {trunkWobble.x, trunkWobble.y, 0} by amplitude and add in
dp2add r0, r7, r7, c1.w // x^2 + y^2 + 0
rsq r0, r0.x // 1/sqrt(x^2 + y^2)
rcp r0, r0.x // sqrt(x^2 + y^2)
mul r0, r0, c2.w // sqrt(x^2 + y^2) * freq
texld r0, r0, s1 // Sample from 1D pulse train texture
mov r1, c3
lrp r2, r0.x, c2, r1 // Blend between light and dark wood colors
sub r4, c4, t6 // Compute normalized vector from vertex to light in eye space (L_eye)
dp3 r5.w, r4, r4 //
rsq r5.w, r5.w //
mul r4, r4, r5.w // L_eye
dp3 r6.w, t7, t7 // Normalize the interpolated normal
rsq r6.w, r6.w //
mul r5, t7, r6.w // N_eye
dp3 r3.w, t6, t6 // Compute normalized vector from the eye to the vertex
rsq r3.w, r3.w //
mul r3, -t6, r3.w // V_eye
add r6, r3, r5 // Compute Eye-Space HalfAngle (L_eye+V_eye)/|L_eye+V_eye|
dp3 r6.w, r6, r6 //
rsq r6.w, r6.w //
mul r6, r6, r6.w // H_eye
dp3_sat r6, r5, r6 // N.H
mad r0.z, r0.z, c5.z, c5.w // scale and bias wood ring pulse to specular exponent range
pow r6, r6.x, r0.z // (N.H)^k
dp3 r5, r4, r5 // Non-clamped N.L
mad_sat r5, r5, c0.z, c0.z // "Half-Lambert" trick for more pleasing diffuse term
mul r6, r6, r0.y // Gloss the highlight with the ramp texture
mad r2, r5, r2, r6 // N.L * procedural albedo + specular
mov oc0, r2

```



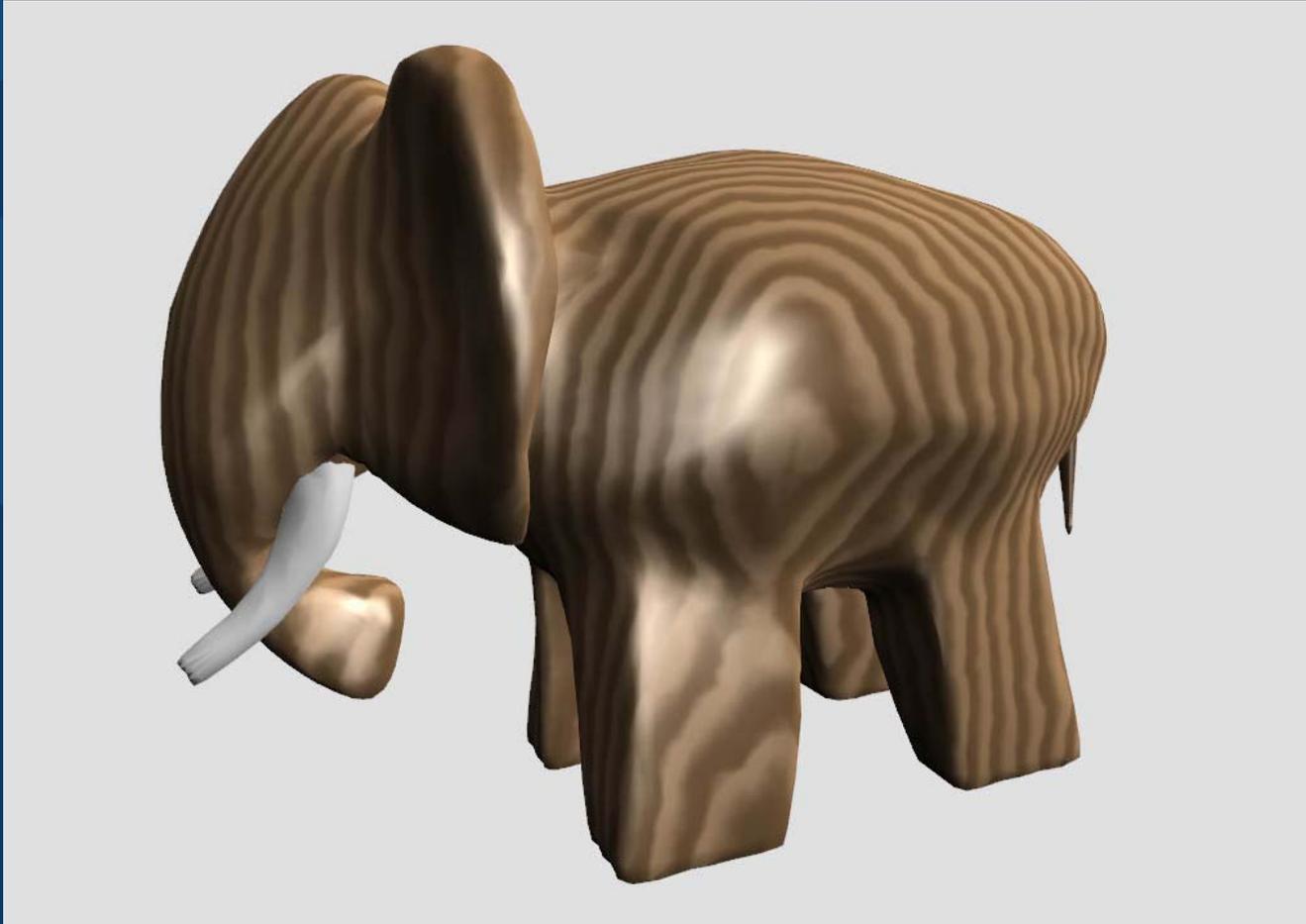
**New code**

# Full Shader



# With Phong Shading

SAN ANTONIO  
SIGGRAPH  
#2002#

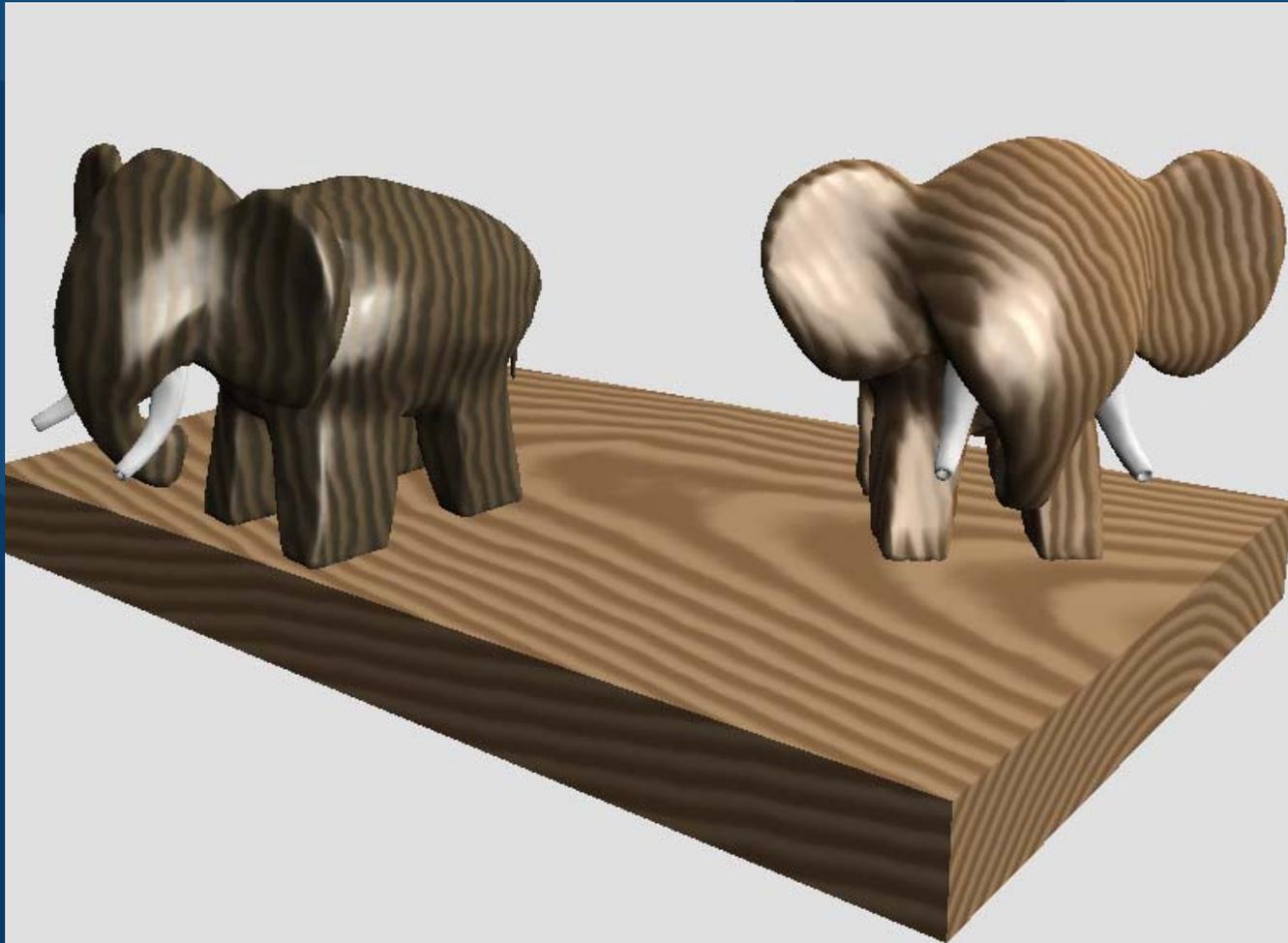


Hardware Shading on ATI RADEON™ 9700



# Final Scene

SAN ANTONIO  
SIGGRAPH  
#2002#

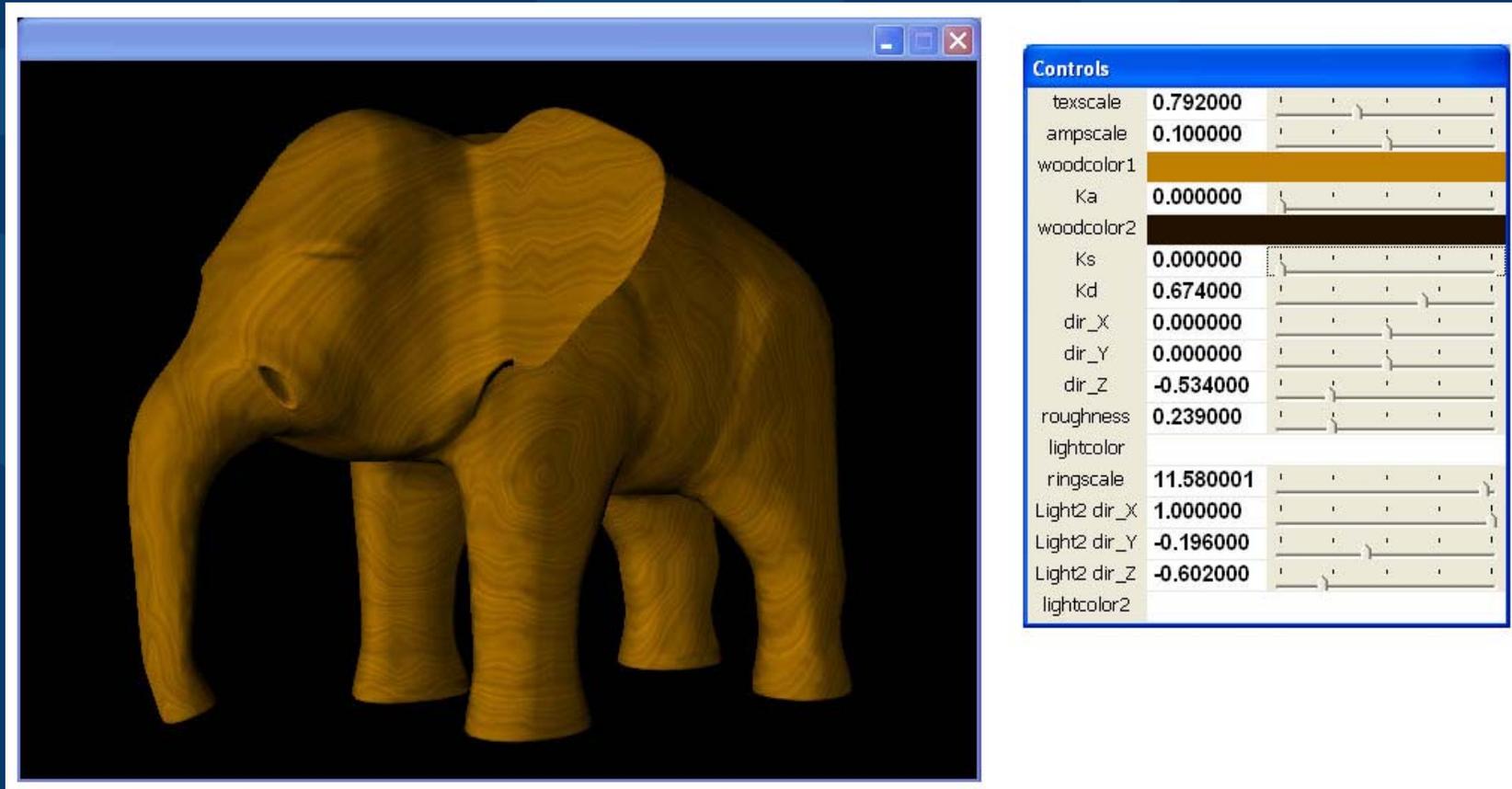


Hardware Shading on ATI RADEON™ 9700



# RenderMonkey™ Compiler

SAN ANTONIO  
**SIGGRAPH**  
#2002#



Controls	
texscale	0.792000
ampscale	0.100000
woodcolor1	
Ka	0.000000
woodcolor2	
Ks	0.000000
Kd	0.674000
dir_X	0.000000
dir_Y	0.000000
dir_Z	-0.534000
roughness	0.239000
lightcolor	
ringscale	11.580001
Light2 dir_X	1.000000
Light2 dir_Y	-0.196000
Light2 dir_Z	-0.602000
lightcolor2	



# Freestyle Shaders



- **Shaders in Chapter 3 of bound notes**
  - **Per-Pixel Hatching**
  - **Refer to bound notes for others**
    - **Per-pixel specular exponent**
    - **Skin**
- **Shaders in Supplement - Chapter 3.1**
  - **High Dynamic Range Rendering**
    - **HDR Environment/Light Maps**
    - **HDR Scene Post-processing**
    - **Local versus distant reflections / refractions**
  - **Motion Blur**
    - **Flying Balls and Plucked Strings**
  - **Image Space Operations for NPR**
  - **Two-tone layered car paint model**



# Real-Time Hatching

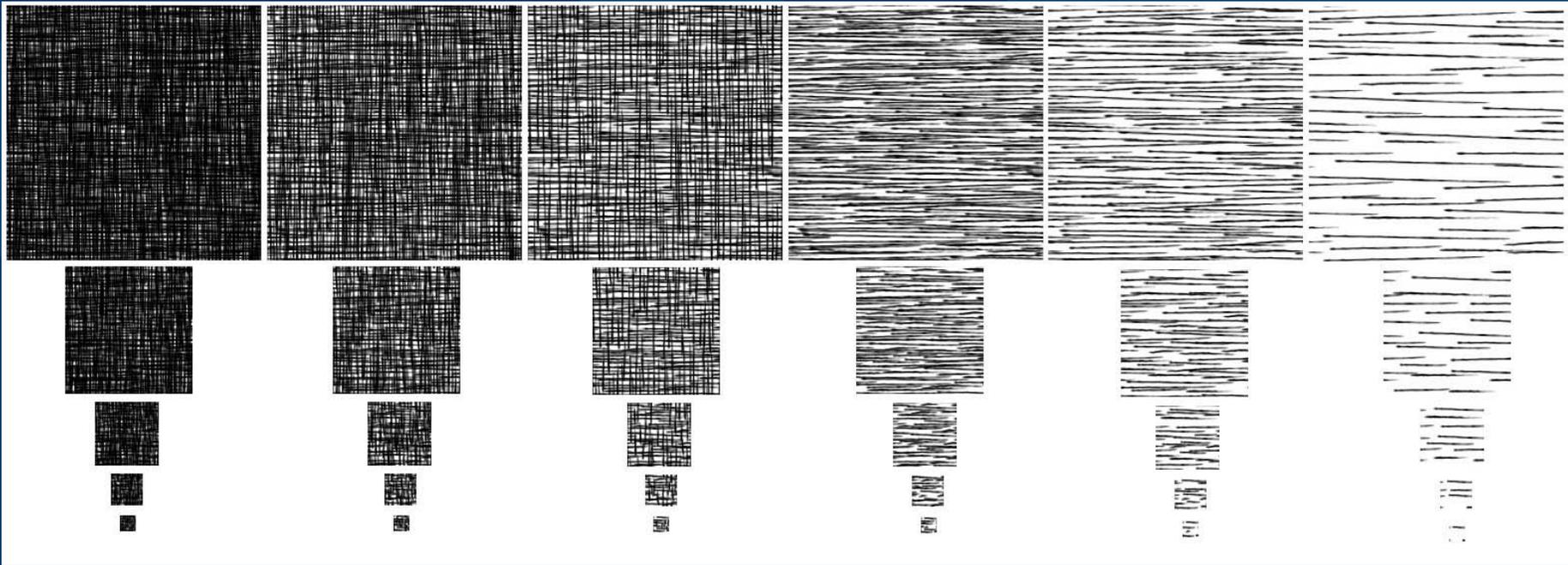


- **Shown at SIGGRAPH 2001 by Praun et al**
- **Tonal Art Maps (TAMs) contain hatching patterns of varying density in different channels**
- **Compute linear combination of TAM channels based on N·L**



# Tonal Art Map

- Weighted sum of these channels determines final tone



# Hatched Shadowed Scene

SAN ANTONIO  
**SIGGRAPH**  
#2002#



Hardware Shading on ATI RADEON™ 9700



# Basic Hatching



ps.1.4

```
texld r0, t0 ; sample the 1st three channels of the TAM
texld r1, t0 ; sample the 2nd three channels of the TAM
texcrd r2.rgb, t1.xyz ; get the 123 TAM weights and place in r2
texcrd r3.rgb, t2.xyz ; get the 456 TAM weights and place in r3
dp3_sat r0, 1-r0, r2 ; dot 123 TAM values with 123 TAM weights
dp3_sat r1, 1-r1, r3 ; dot 456 TAM values with 456 TAM weights
add_sat r0, r0, r1 ; add reg 0 and reg1
mov_sat r0, 1-r0 ; complement and saturate
```



# Hatching Enhancements



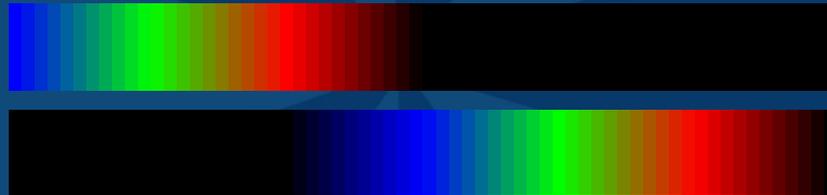
- **Per-pixel determination of TAM weights based on N·L and distance attenuation of light**
- **Hatch tinting as function of a base color map**



# Per-Pixel TAM Weighting



1. Compute  $N \cdot L$
2. Compute Distance Attenuation
3. Modulate attenuation and  $N \cdot L$  with base map intensity
4. Do dependent read from 1D textures to convert above intensity to TAM weights:

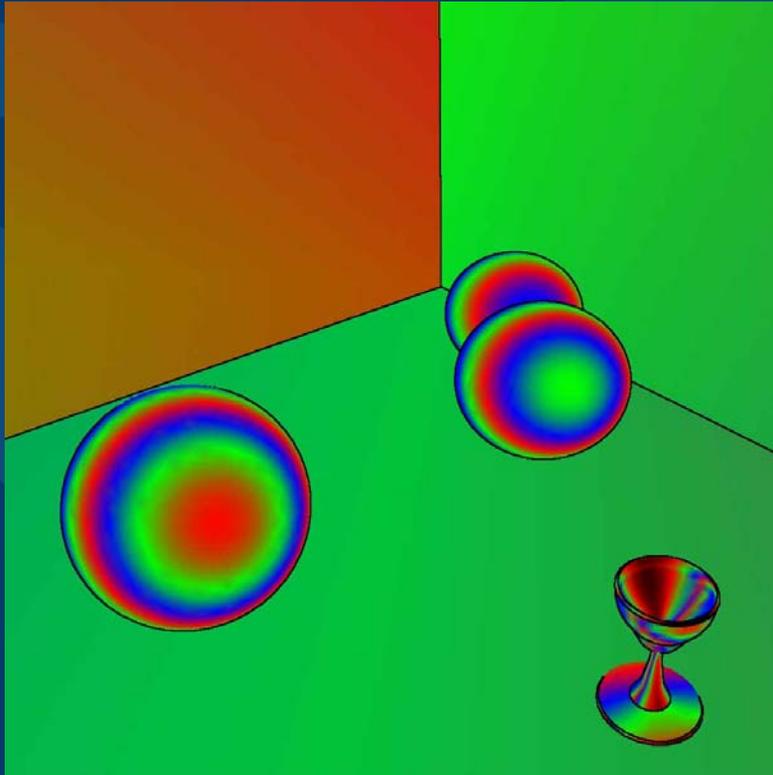


5. Fetch TAMs
6. Compute Linear Combination of TAM channels
7. Tint according to base map color

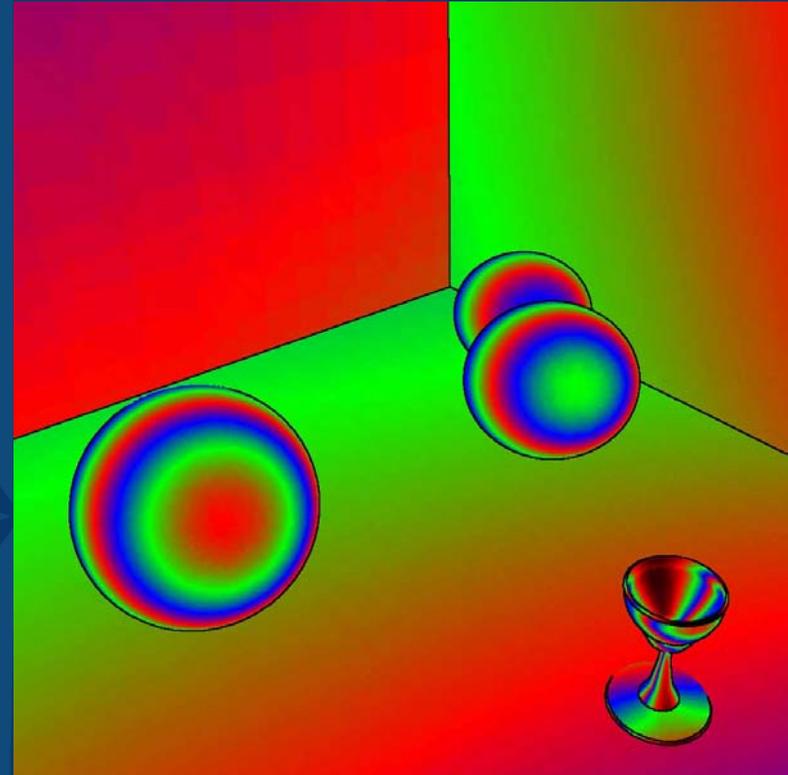


# Per-Pixel TAM Weights

SAN ANTONIO  
**SIGGRAPH**  
#2002#



**Per Vertex TAM Weights**

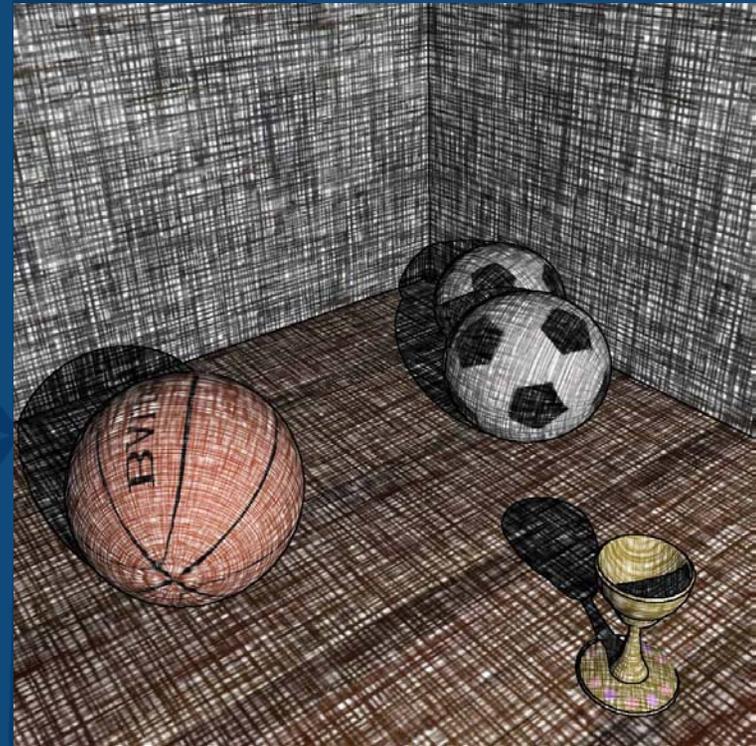
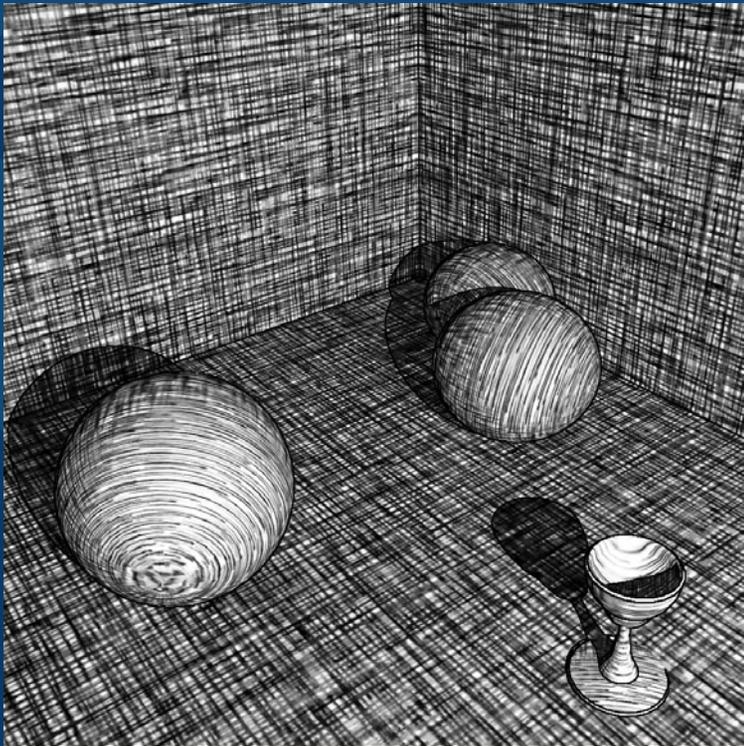


**Per Pixel TAM Weights**



# Hatched Images with Shadows

SAN ANTONIO  
**SIGGRAPH**  
#2002#



# Enhanced Hatching



ps.1.4

def c0, 1.00f, 1.00f, 1.00f, 1.00f

def c1, 0.30f, 0.59f, 0.11f, 0.00f ; RGB to luminance conversion weights

texcrd r1.rgb, t2 ; N·L

texld r4, t3 ; Intensity map looked up from light space position

texld r5, t0 ; Base Texture

mul\_x2 r4, r4.r, r1.r ; N·L \* attenuation

add r4, r4, c2 ; += ambient

dp3 r3, r5, c1 ; Intensity of base map

mul r5, r4, r5 ; Modulate base map by light

mul r4, r4, r3 ; Modulate light by base map intensity

phase

texld r0, t1 ; sample the first three channels of the TAM

texld r1, t1 ; sample the second three channels of the TAM

texld r2, r4 ; Get weights for 123

texld r3, r4 ; Get weights for 456

dp3\_sat r0, 1-r0, r2 ; dot the reg0 (TAM values) with reg2 (TAM weights)

dp3\_sat r1, 1-r1, r3 ; dot the reg1 (TAM values) with reg3 (TAM weights)

add\_sat r0, r0, r1 ; add reg0 and reg1

mul r0.rgb, 1-r5, r0 ; Color hatches with base texture

mov\_sat r0, 1-r0 ; complement and saturate



# High Dynamic Range Rendering

SAN ANTONIO  
SIGGRAPH  
#2002#



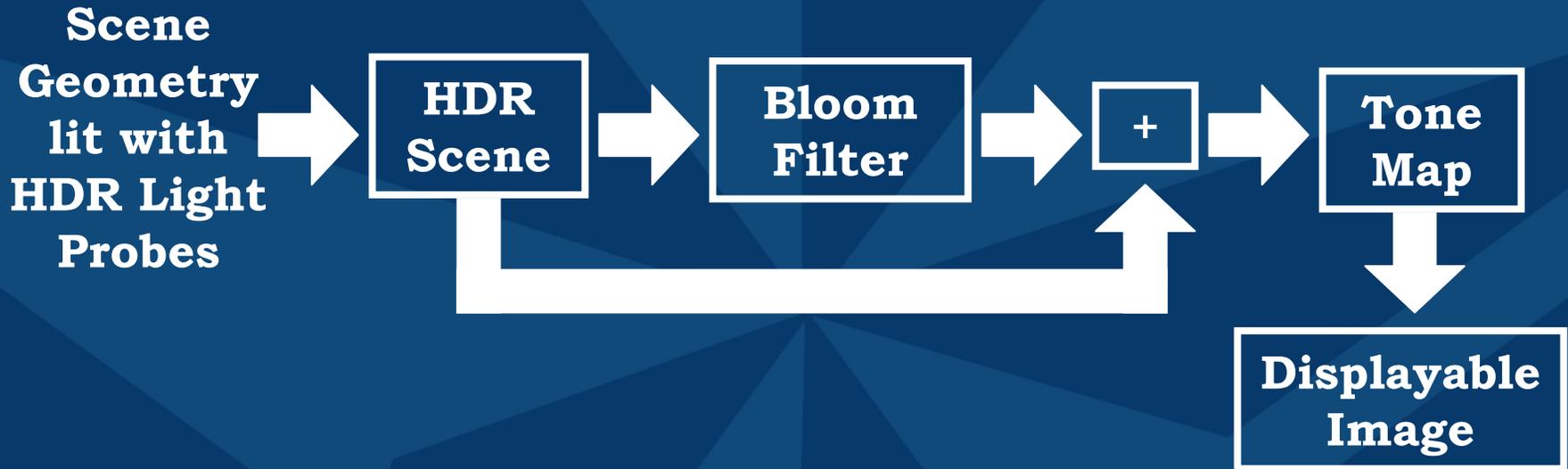
 Radeon 9700

Hardware Shading on ATI RADEON™ 9700



# HDR Rendering Process

SAN ANTONIO  
SIGGRAPH  
#2002#



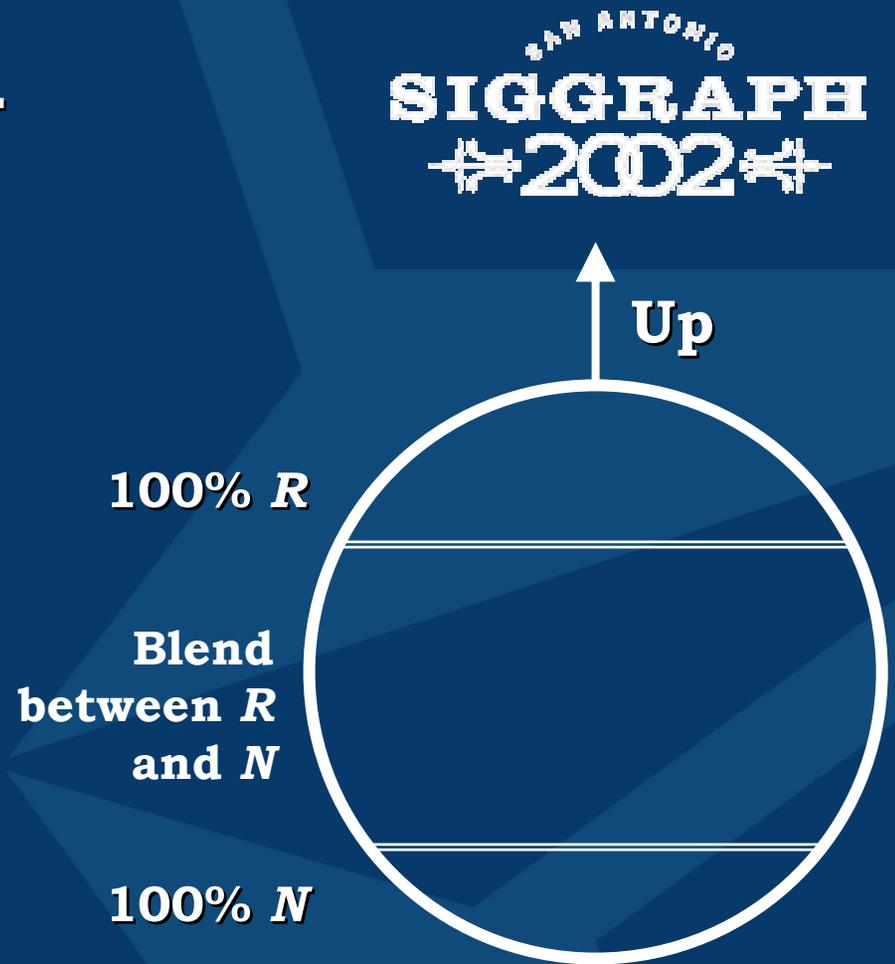
# Rendering the Scene

- **Render reflected scene into HDR planar reflection map for table top**
- **HDR light probe for distant environment**
- **HDR environment maps for local reflections from balls on pedestals**
- **Postprocess to get glows**
- **Tone map to displayable image**



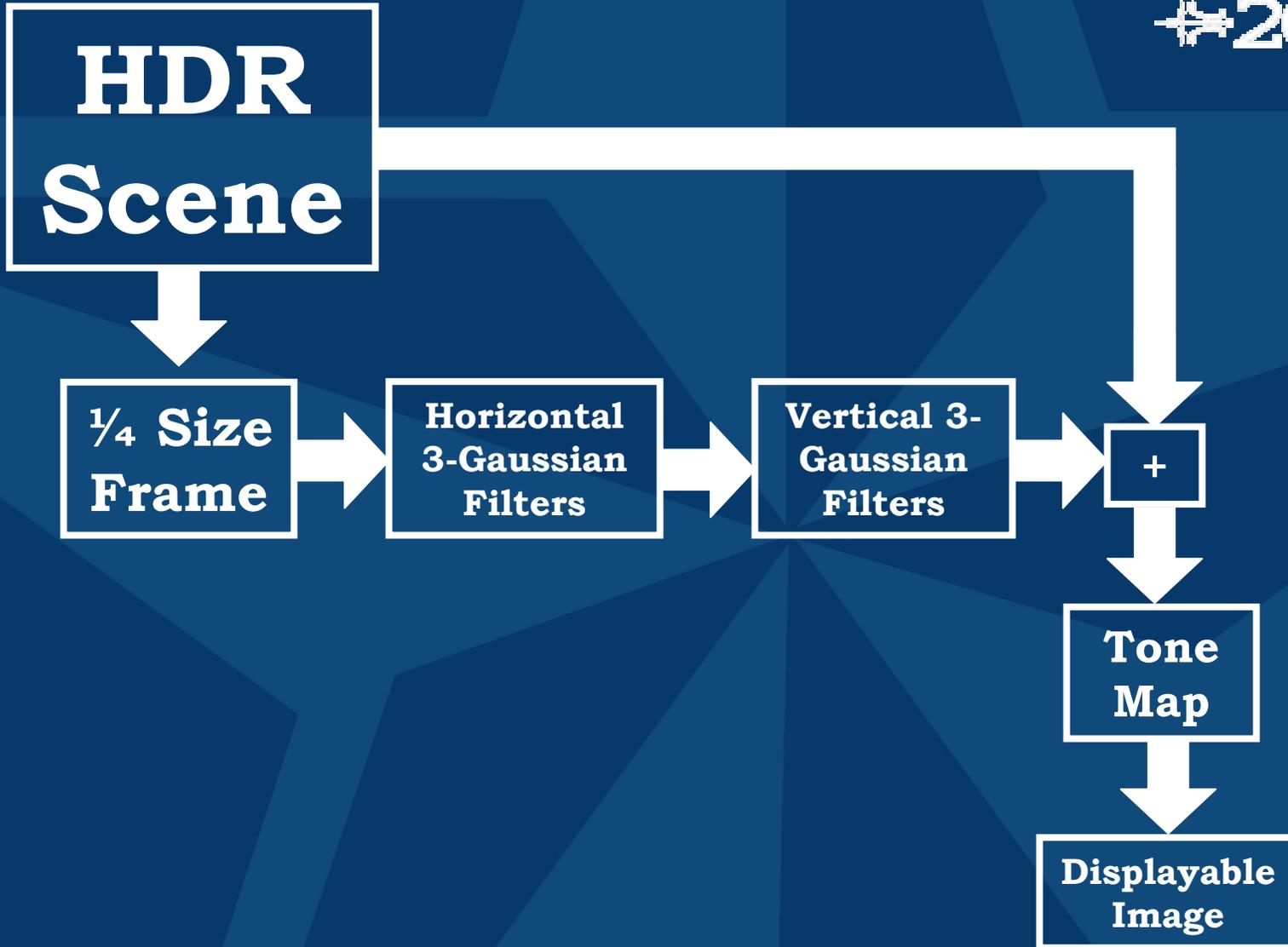
# Local Reflection

- Distant HDR Light probe is always sampled with reflection vector in pixel shader
- Local environment map is sampled with a blend of the surface normal ( $N$ ) and the reflection vector ( $R$ )



# Frame Postprocessing

SAN ANTONIO  
**SIGGRAPH**  
#2002#



# Tone Mapping

SAN ANTONIO  
**SIGGRAPH**  
#2002#



Hardware Shading on ATI RADEON™ 9700



# Motion Blur in Animusic *Pipe Dream* Demo

SAN ANTONIO  
**SIGGRAPH**  
#2002#

- **Real-time version of Animusic *Pipe Dream* animation from SIGGRAPH 2001 Electronic Theater**



Image from Real-Time  
Animusic *Pipe Dream* demo



# Motion Blur Distorts Shape and Shading



- **Shape**

- **Stretching objects similar to [Wloka & Zeleznik 96]**
  - **Flying balls**
  - **Plucked strings**

- **Shading**

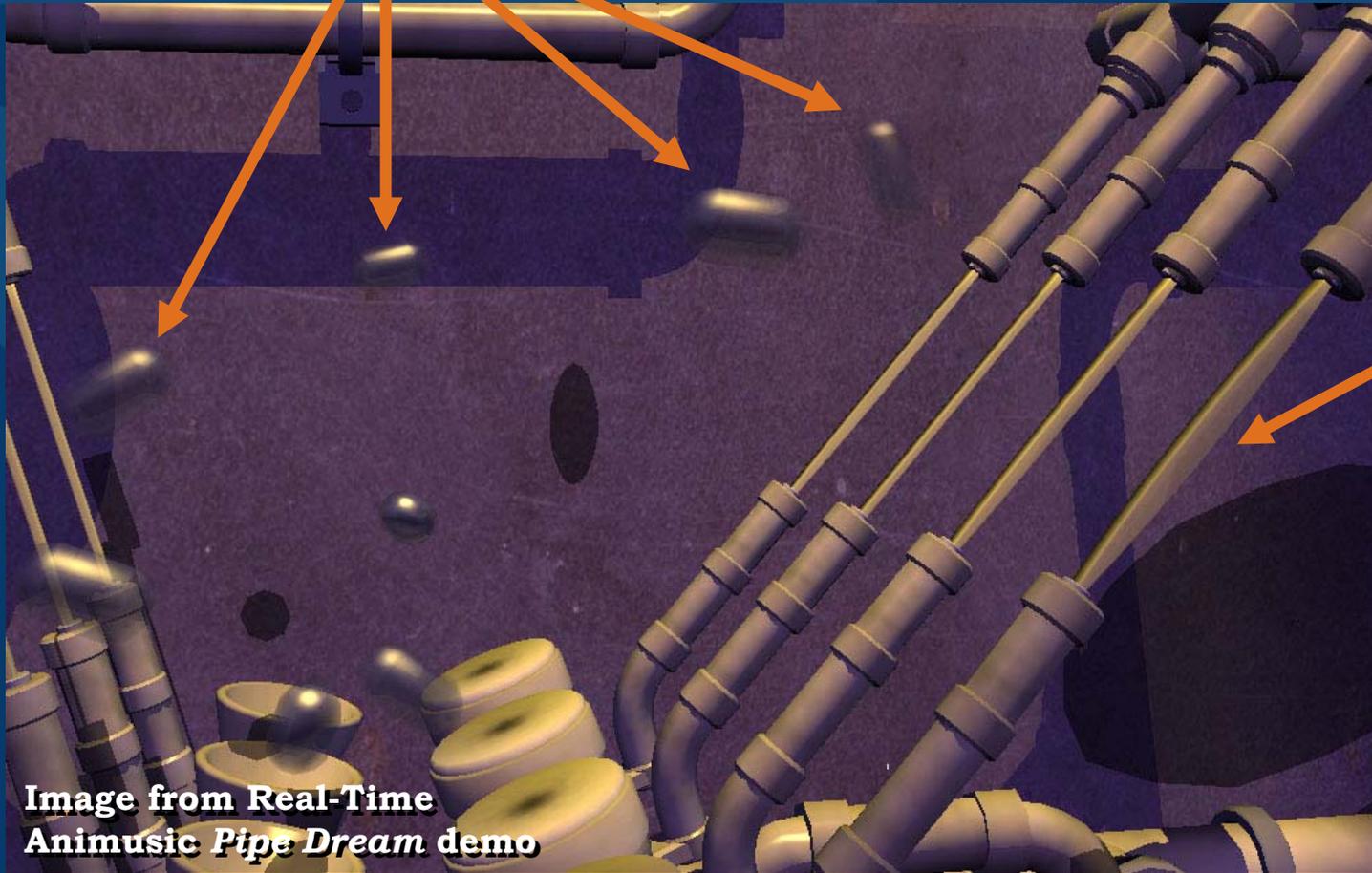
- **Lower specular exponent and intensity as function of velocity**
- **Apply per-pixel lod bias as a function of velocity**
- **Alpha set to represent contribution to the scene**



# Motion Blur

SAN ANTONIO  
**SIGGRAPH**  
#2002#

## Moving Balls



**Plucked  
String**

Image from Real-Time  
Animusic *Pipe Dream* demo

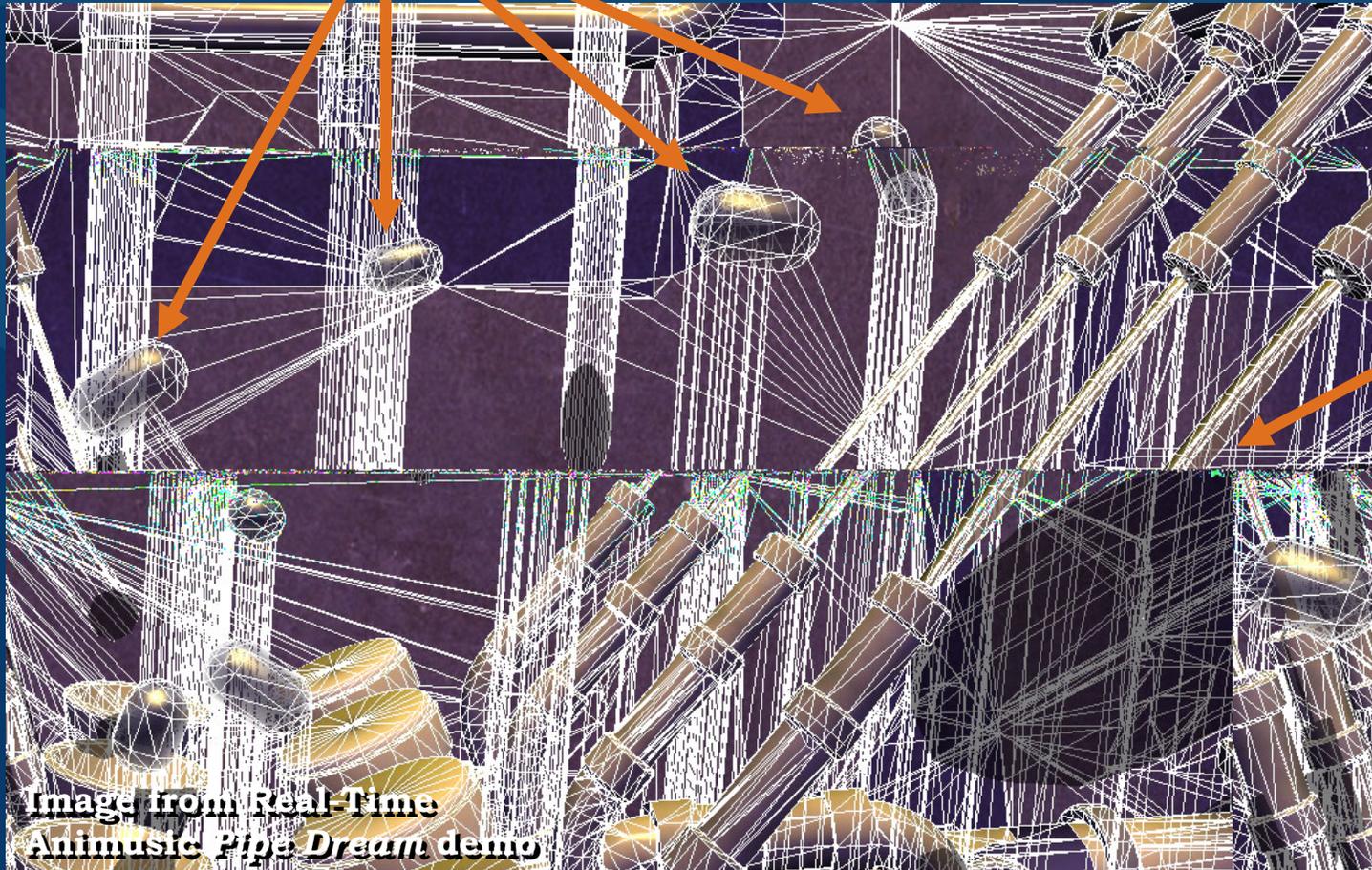


Hardware Shading on ATI RADEON™ 9700

# Motion Blur

SAN ANTONIO  
**SIGGRAPH**  
#2002#

## Moving Balls



**Plucked  
String**

Hardware Shading on ATI RADEON™ 9700



# Distorting Shape of Balls

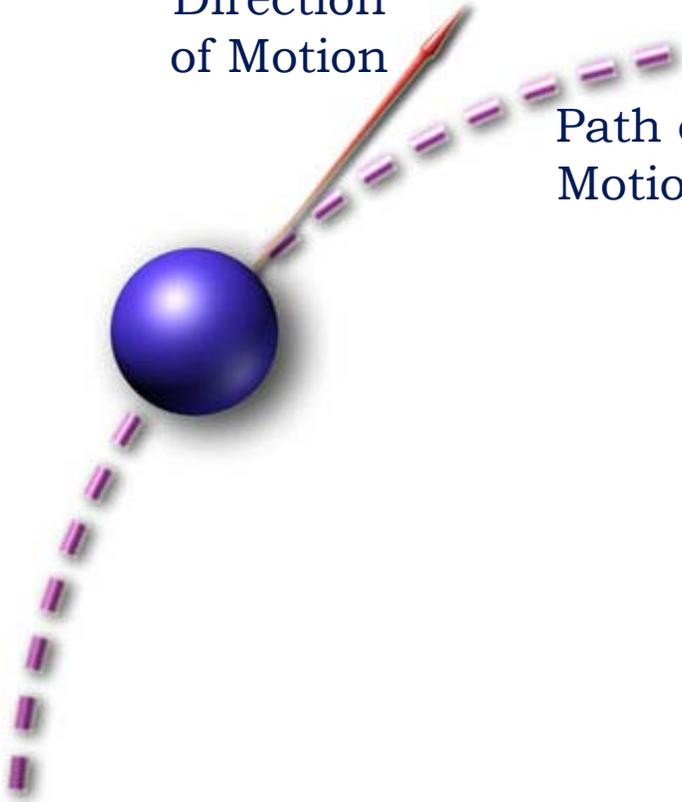
- **Instantaneous velocity (distance ball moved since previous time step) is input to vertex shader as is the motion vector  $M$**
- **$M - (M \cdot \text{Eye})$  is motion perpendicular to the eye**
- **Vertex shader computes Blur factor**
- **$1/(1+\text{distance traveled}/\text{ball diameter})$**



# Capsule Distortion

Direction  
of Motion

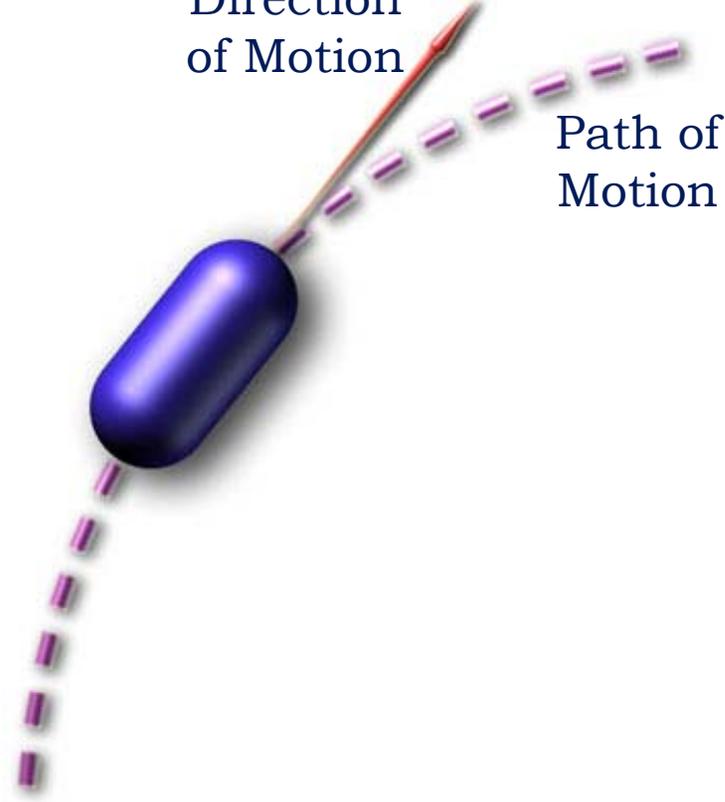
Path of  
Motion



**No Motion Blur**

Direction  
of Motion

Path of  
Motion



**With Motion Blur**



# Distorting Shading of Balls

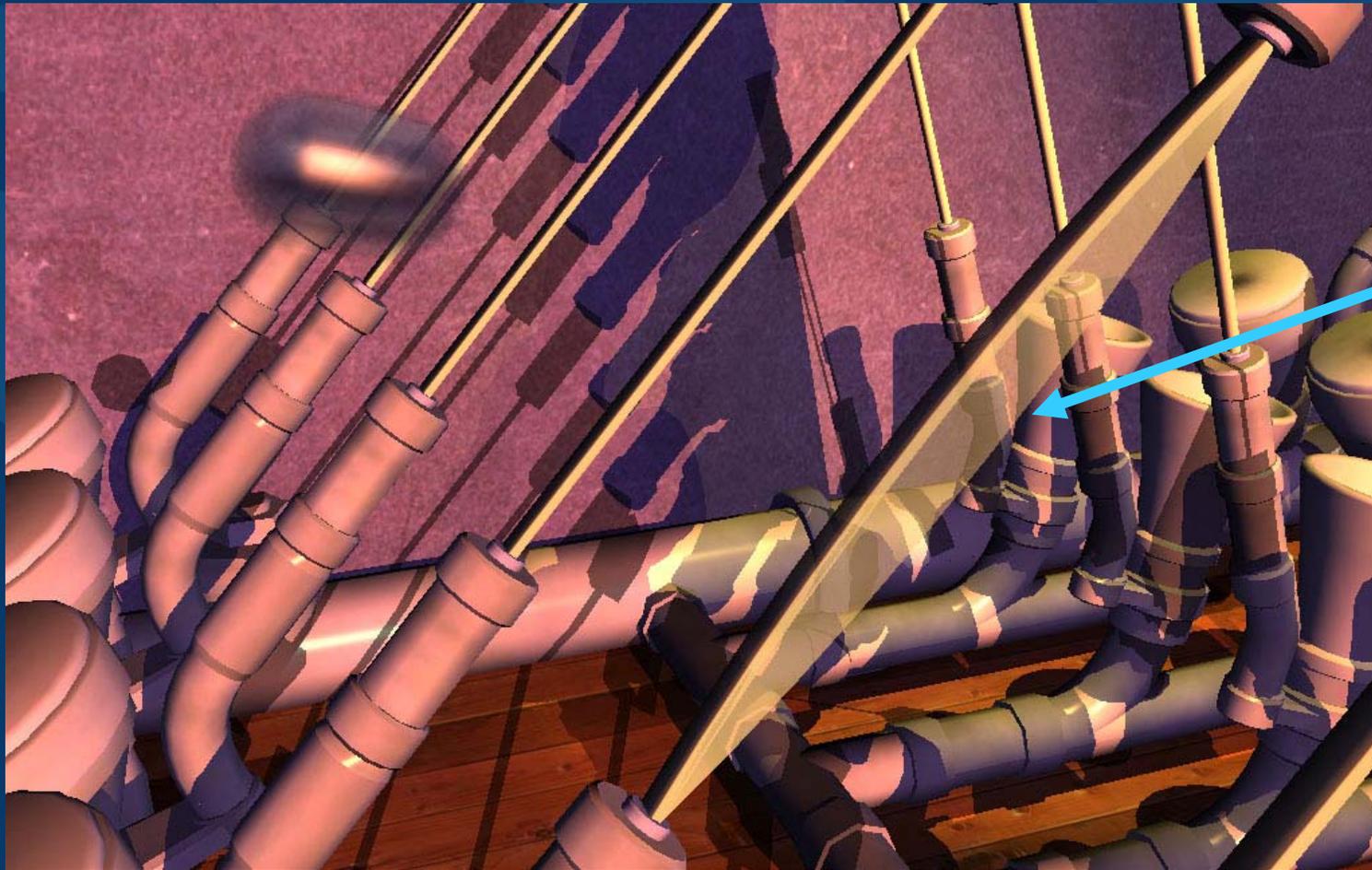


- **Blur factor interpolated across polygons**
  - **Pixel shader does LOD biased texture load from environment map as function of blurriness factor**
  - **Specular exponent ( $k$ ) gets smaller as ball goes faster, broadening the highlight**
  - **Scalar gloss term also get smaller**
  - **Serves to distribute the energy over pixels as specular highlight “smears”**
  - **Alpha of pixel says how much the ball was “there” at a given pixel**



# Motion Blur on Strings

SAN ANTONIO  
**SIGGRAPH**  
#2002#



**Plucked  
String**



Hardware Shading on ATI RADEON™ 9700

# Motion Blur on Strings



- **Two instances of string geometry are drawn when in motion**
- **One instance bends back and forth over time and retains original thickness**
- **Other instance is stretched apart to span full current amplitude of motion and is blended on top**
- **Alpha of second string is function of amplitude**
  - **More motion equals lower alpha**



# Image Space Outlining for NPR

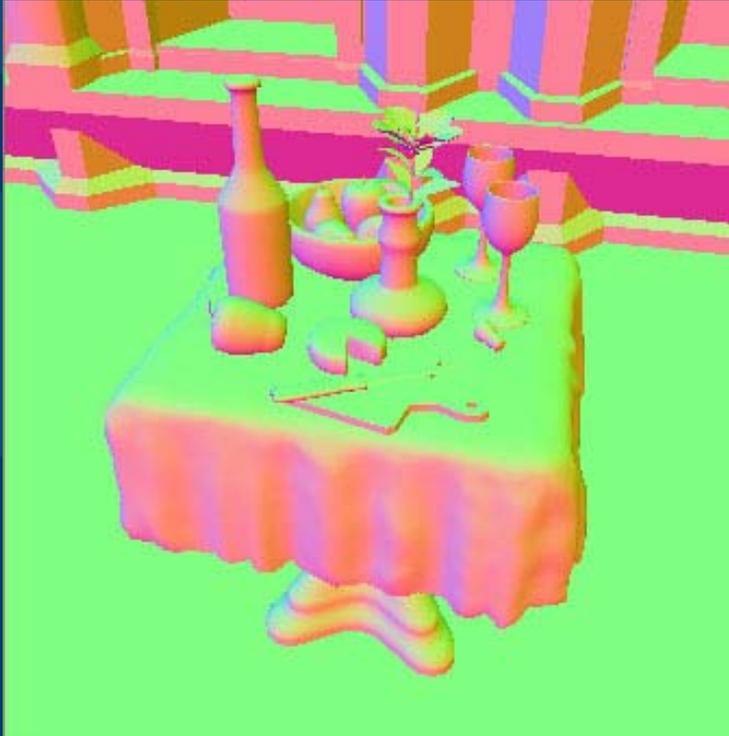


- **Render alternate representation of scene into texture map**
  - **With the RADEON 9700, we're able to render into up to four targets simultaneously, effectively implementing Saito and Takahashi's G-buffer**
- **Run filter over image to detect edges**
  - **Implemented using pixel shading hardware**



# Normal and Depth

SAN ANTONIO  
**SIGGRAPH**  
#2002#



**World Space Normal**



**Eye Space Depth**



# Outlines

SAN ANTONIO  
**SIGGRAPH**  
#2002#



**Normal Edges**

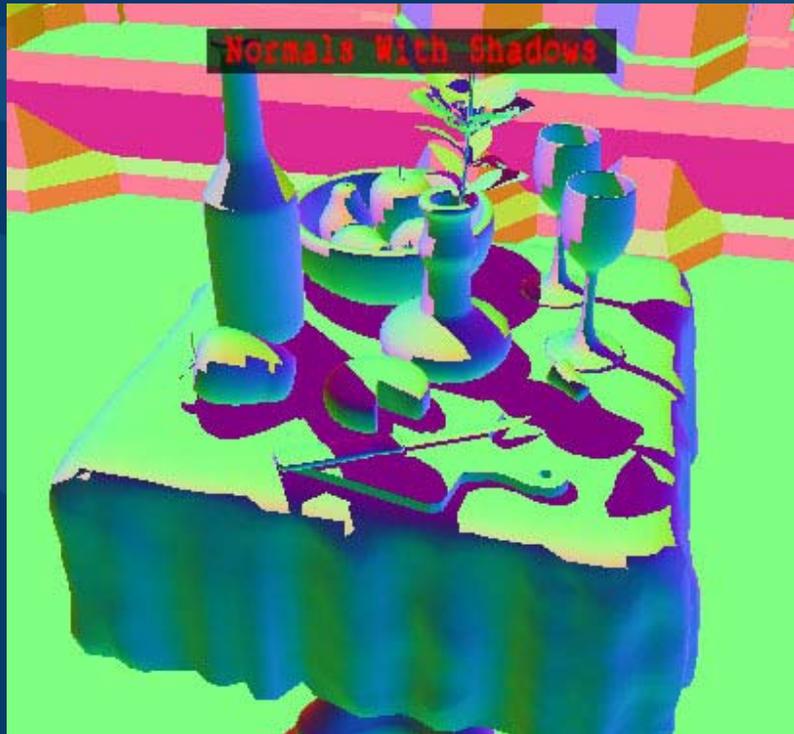


**Depth Edges**



# Normal and Depth Negated in Shadow

SAN ANTONIO  
**SIGGRAPH**  
#2002#



**World Space Normal  
Negated in Shadow**



**Eye Space Depth  
Negated in Shadow**



# Normal and Depth Outlines



**Edges from Normals**



**Edges from Depth**



# Object and Shadow Outlines

SAN ANTONIO  
**SIGGRAPH**  
#2002#



**Outlines from selectively  
negated normals and depths**

Hardware Shading on ATI RADEON™ 9700



# Texture Region IDs

SAN ANTONIO  
SIGGRAPH  
#2002#



Hardware Shading on ATI RADEON™ 9700



# Edges at Texture Region Boundaries

SAN ANTONIO  
**SIGGRAPH**  
#2002#

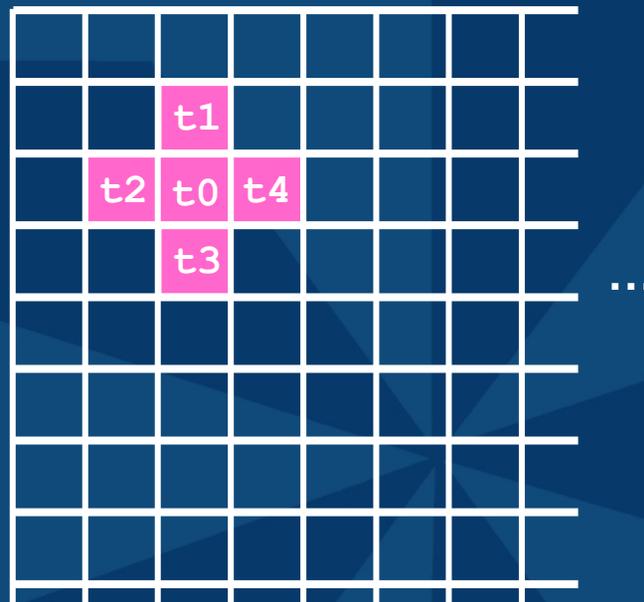


Hardware Shading on ATI RADEON™ 9700



# Edge Filter

SAN ANTONIO  
**SIGGRAPH**  
#2002#



**5-tap Filter**



# Edge Filter Code



```
ps.2.0
def c0, 0.0f, 0.80f, 0, 0 // normal thresholds
def c3, 0, .5, 1, 2
def c8, 0.0f, 0.0f, -0.01f, 0.0f // Depth thresholds
def c9, 0.0f, -0.25f, 0.25f, 1.0f
def c12, 0.0f, 0.01f, 0.0f, 0.0f // TexID Thresholds
dcl_2d s0
dcl_2d s1
dcl t0
dcl t1
dcl t2
dcl t3
dcl t4

// Sample the map five times
texld r0, t0, s0 // Center Tap
texld r1, t1, s0 // Down/Right
texld r2, t2, s0 // Down/Left
texld r3, t3, s0 // Up/Left
texld r4, t4, s0 // Up/Right

//-----
// NORMALS
//-----
mad r0.xyz, r0, c3.w, -c3.z
mad r1.xyz, r1, c3.w, -c3.z
mad r2.xyz, r2, c3.w, -c3.z
mad r3.xyz, r3, c3.w, -c3.z
mad r4.xyz, r4, c3.w, -c3.z

// Take dot products with center (Signed result -1 to 1)
dp3 r5.r, r0, r1
dp3 r5.g, r0, r2
dp3 r5.b, r0, r3
dp3 r5.a, r0, r4

// Subtract threshold
sub r5, r5, c0.g

// Make black/white based on threshold
cmp r5, r5, c1.g, c1.r

// detect any 1's
dp4_sat r11, r5, c3.z
mad_sat r11, r11, c1.b, c1.w // Scale and bias result

//-----
// z
//-----
add r10.r, r0.a, -r1.a // Take four deltas
add r10.g, r0.a, -r2.a
add r10.b, r0.a, -r3.a
add r10.a, r0.a, -r4.a

cmp r10, r10, r10, -r10 // Take absolute value
add r10, r10, c8.b // Subtract threshold
cmp r10, r10, c1.r, c1.g // Make black/white
dp4_sat r10, r10, c3.z // Sum up detected pixels
mad_sat r10, r10, c1.b, c1.w // Scale and bias result
mul_r11, r11, r10 // Combine with previous

//-----
// TexIDs
//-----
// Sample the map five times
texld r0, t0, s1 // Center Tap
texld r1, t1, s1 // Down/Right
texld r2, t2, s1 // Down/Left
texld r3, t3, s1 // Up/Left
texld r4, t4, s1 // Up/Right

// Get differences in color
sub r1.rgb, r0, r1
sub r2.rgb, r0, r2
sub r3.rgb, r0, r3
sub r4.rgb, r0, r4

// Calculate magnitude of color differences
dp3 r1.r, r1, c3.z
dp3 r1.g, r2, c3.z
dp3 r1.b, r3, c3.z
dp3 r1.a, r4, c3.z

cmp r1, r1, r1, -r1 // Take absolute values
sub r1, r1, c12.g // Subtract threshold
cmp r1, r1, c1.r, c1.g // Make black/white
dp4_sat r10, r1, c3.z // Total up edges
mad_sat r10, r10, c1.b, c1.w // Scale and bias result
mul_r11, r10, r11 // Combine with previous

// Output
mov oC0, r11
```



# Morphology

SAN ANTONIO  
SIGGRAPH  
#2002#



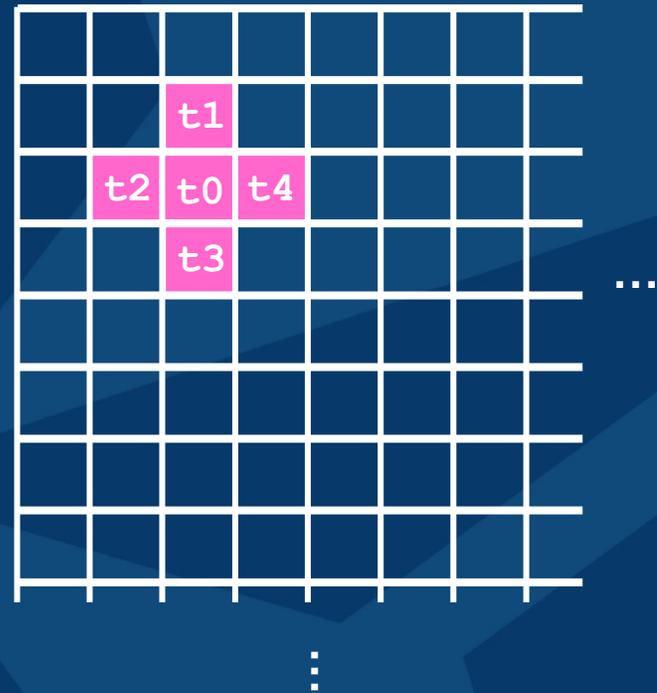
# Dilation Shader



```
ps.2.0
def c0, 0, .5, 1, 2
def c1, 0.4f, -1, 5.0f, 0
dcl_2d s0
dcl t0
dcl t1
dcl t2
dcl t3
dcl t4

// Sample the map five times
texld r0, t0, s0 // Center Tap
texld r1, t1, s0 // Up
texld r2, t2, s0 // Left
texld r3, t3, s0 // Down
texld r4, t4, s0 // Right

// Sum the samples
add r0, r0, r1
add r1, r2, r3
add r0, r0, r1
add r0, r0, r4
mad_sat r0, r0.r, c1.r, c1.g // Threshold
mov oC0, r0
```



# Outlining Sketch



- **More detail on this in *Real-Time Image-Space Outlining for Non-Photorealistic Rendering* sketch on Thursday in the Rendering session at 3:30 - 5:30 pm in River Room 001**



# Two-tone Car Paint

SAN ANTONIO  
SIGGRAPH  
#2002#

- **Normal Decompression**
- **Sparkle from microflake**
- **Base color**
- **Clear coat**
- **Rough Specular**



# Two-tone Car Paint

SAN ANTONIO  
SIGGRAPH  
#2002#



Hardware Shading on ATI RADEON™ 9700



# Normal Decompression



- Sample from two-channel 16-16 normal map
- Derive  $z$  from  $+\text{sqrt}(1 - x^2 - y^2)$



# $N_s$ and $N_{ss}$

- **Two normal maps on car**
  - Normal from appearance preserving simplification process,  $N$
  - High frequency normalized vector noise,  $N_n$
- **Compute  $N_s$  and  $N_{ss}$  from  $N$  and  $N_n$** 
  - $N_s = (aN_n + bN) / |aN_n + bN|$ , where  $a \ll b$
  - $N_{ss} = (cN_n + dN) / |cN_n + dN|$ , where  $c = d$



# Base Color and Flake

- Base color and flake effect are derived from  $N_s$  and  $N_{ss}$  using the following polynomial

$$c_0(N_s \cdot V) + c_1(N_s \cdot V)^2 + c_2(N_s \cdot V)^4 + c_3(N_{ss} \cdot V)^{16}$$

Base Color

Flake



# Layers of Car Paint

SAN ANTONIO  
**SIGGRAPH**  
#2002#

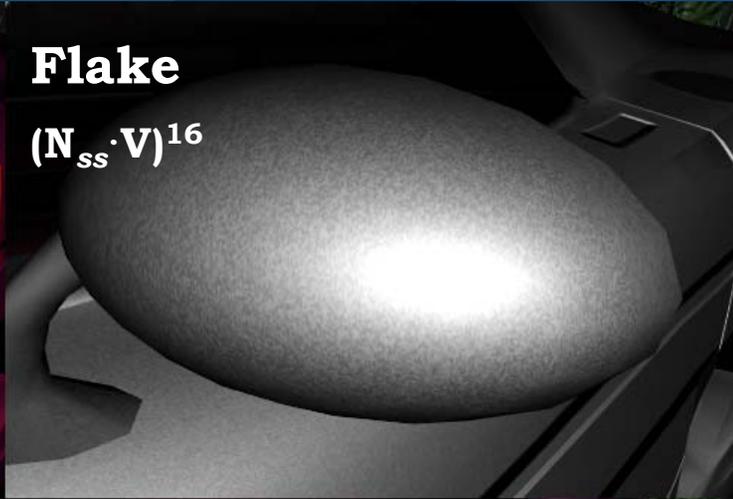
## Base Color

$$c0(Ns \cdot V) + c1(Ns \cdot V)^2 + c2(Ns \cdot V)^4$$



## Flake

$$(N_{ss} \cdot V)^{16}$$



## HDR Clear Coat



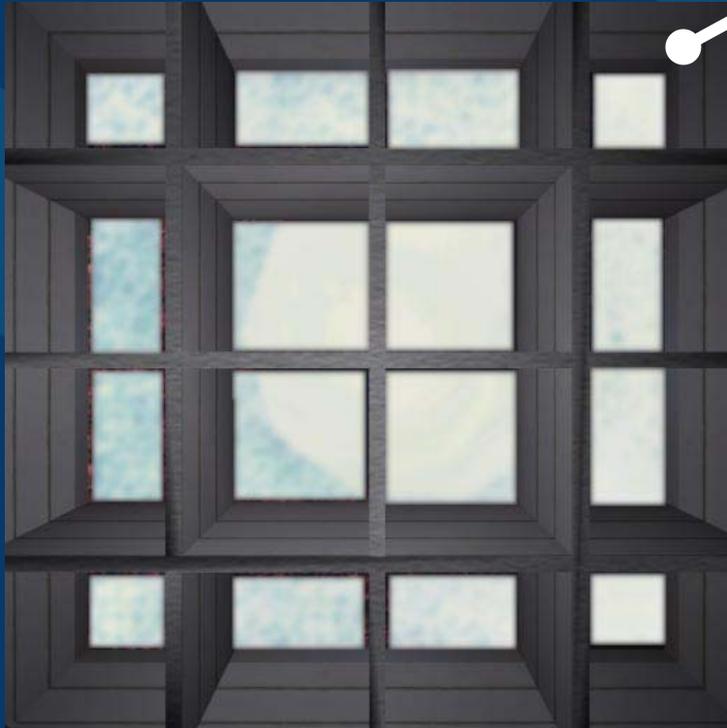
## Final Color



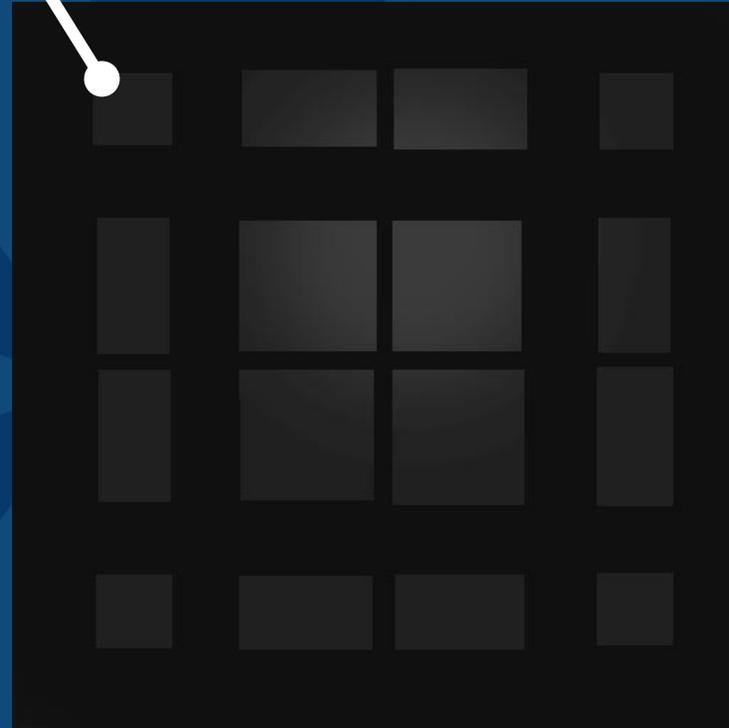
# RGBScale HDR Environment Map

SAN ANTONIO  
**SIGGRAPH**  
#2002#

Ceiling of car showroom



**Top Cube Map  
Face RGB**



**Top Face Scale in  
Alpha Channel**



# RGBScale HDR Environment Map



- Alpha channel contains  $1/16$  of the true HDR scale of the pixel value
- RGB contains normalized color of the pixel
- Pixel shader reconstructs HDR value from  $scale * 8 * color$  to get half of the true HDR value
- Obvious quantization issues, but reasonable for some applications
- Similar to Ward's RGBE "Real Pixels" but simpler to reconstruct in the pixel shader



# Image Space Glows



- **Render scene into multisample AA back buffer**
- **Also render HDR clear coat and other emissive objects into small texture map**
- **Run separable Gaussian blur over this small texture similar to RNL demo shown earlier**
- **Composite this with rendered scene**
- **Gives glows off of any bright areas in the scene, including reflections off of the car body**



# Rough Specular

SAN ANTONIO  
SIGGRAPH  
#2002#



Hardware Shading on ATI RADEON™ 9700



# Rough Specular



- **Use texldb for all accesses to cubic environment map**
- **For rough specular, the bias is high, causing a blurring effect**
- **Also convert color fetched from environment map to luminance in rough trim areas**



# Summary



- **RADEON™ 9700 Shader Models**
  - 2.0 vertex and pixel shaders in DirectX® 9
  - ATI\_fragment\_program OpenGL Extension
- **Compulsories**
  - Homomorphic BRDF
  - Shiny bumpy bouncy fun thing
  - Procedural wood
- **Freestyle**
  - Per-Pixel Hatching
  - High Dynamic Range Rendering
    - HDR Environment/Light Maps
    - HDR Scene Post-processing
    - Local versus distant reflections / refractions
  - Motion Blur
  - Image Space Operations for NPR
  - Two-tone layered car paint model



# More Information



- Notes online at [www.ati.com/developer](http://www.ati.com/developer)
- Shader Tool set, RenderMonkey™, will be discussed in Tech Talks
  - Tuesday @ 10 am to noon in Hall D
  - Thursday @ 1pm to 3pm in Hall D
- Image-space Outlining sketch Thursday in the Rendering session at 3:30 - 5:30 pm in River Room 001
- Fur Sketch Thursday in the Hardware Rendering session at 10:30 am - 12:15 pm in Room 217BCD
- Come by the ATI Booth

