SIGGRAPH2005

GPU Shading and Rendering

Shading Compilers

Avi Bleiweiss

ATI Research Silicon Valley

Overview

- GPU evolving in fast pace
- *Shader Model 3.0*
  - Extended resources
  - Dynamic flow control
- Multi GPU systems
  - PCI Express bus

Shading Compilers

- Evidently divisible
- High level challenges
  - Encapsulate description
  - Virtualize GPU processors
- Scheduling, optimization
  - Intimate hardware

Outline

- Hiding shading languages
- Multi GPU shader partition
- Remapping GPU processors
- Source level debugger

  - Ashli shading technology

Outline

- Hiding shading languages
- Multi GPU shader partition
- Remapping GPU processors
- Source level debugger

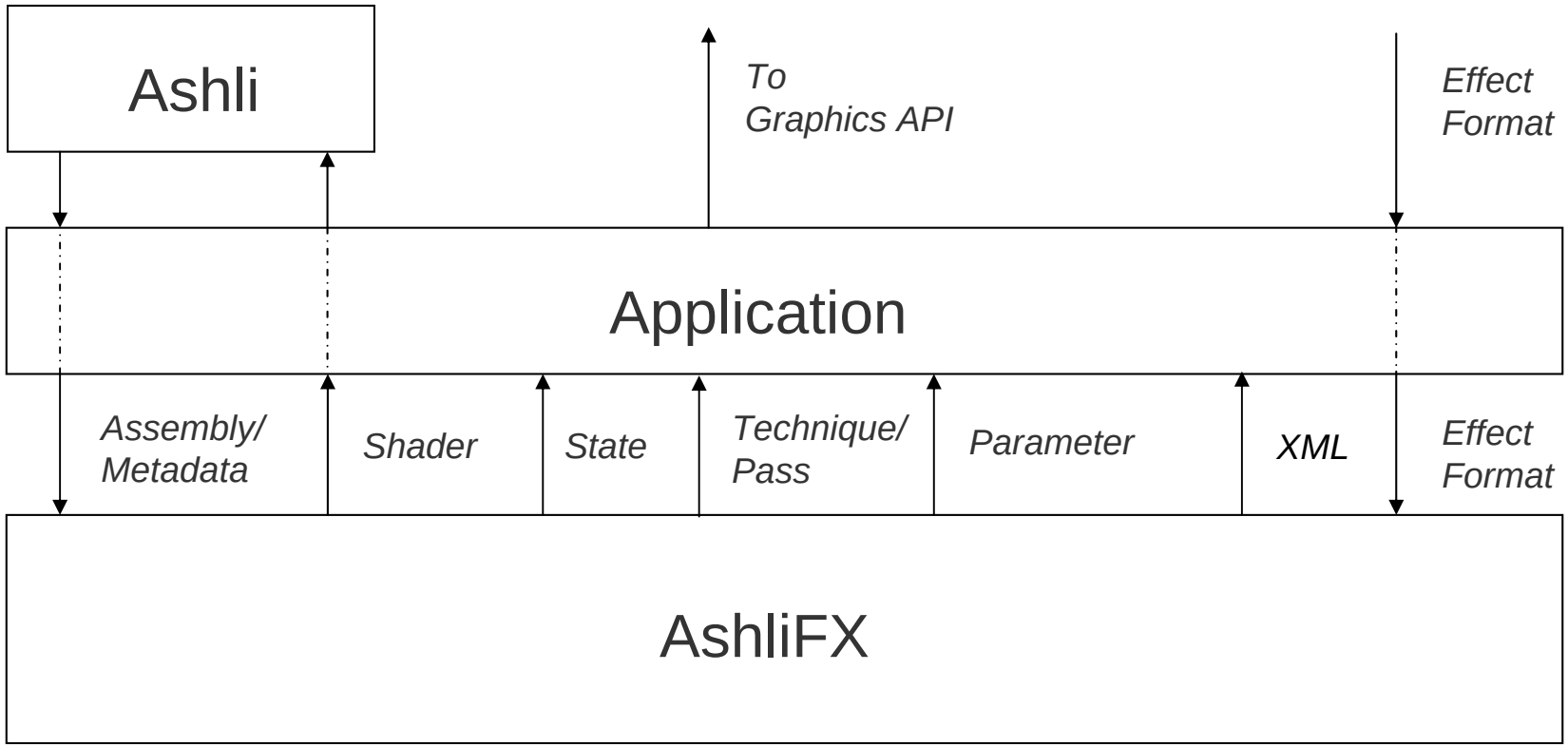
Representation

- Content management
  - Single representation
- Ties to graphics API
- Higher level abstraction
  - Microsoft *Effect* format

Effect Format

- Parameters and functions
- Techniques and passes
- Any shading language
- Ashli for Effect - *AshliFX*
  - Multi lingual Effect compiler

AshliFX


Ashli
To
Graphics API
Effect
Format
Application
Assembly/
Metadata
Shader
State
Technique/
Pass
Parameter
XML
Effect
Format
AshliFX

AshliFX (cnt'd)

- Implicit language binding
- Extracted shader filtering
  - Avoid semantics, annotations
- Metadata to Effect state
- Effect state observer
  - Graphics API neutral

Sample

SIGGRAPH2005

```
// cartoon.fx

#pragma language GLSL

varying vec3 eyeNormal;

void mainVS()
{
   gl_Position = ftransform();
   eyeNormal = gl_NormalMatrix * gl_Normal;
}

const vec3 diffuse : Diffuse = vec3( 1.0, 0.0, 0.0);
uniform vec4 scale : Scale = vec4(1., 2., 3., 4.);

void mainPS()
{
  vec3 normal;
  vec3 color;
  float illum;
  vec3 lightDirection=vec3(1.0,1.0,1.0);

  lightDirection = normalize(lightDirection);
  normal = normalize( eyeNormal );
  vec3 hlf = normalize( lightDirection + vec3(0.0,0.0,1.0));
  illum = clamp( dot(normal, lightDirection), 0.0, 1.0);
  color = clamp( diffuse*(illum+0.2) +
              pow( max(dot(hlf,normal),0.0), 16.0), 0.0, 1.0);
  color = scale.xyz* floor(color*4.0) *0.25;

  gl_FragColor = vec4( color, 1.0);
}

technique BumpReflect0 {
   pass p0 {
      VertexShader = compile vs_2_0 mainVS();

      ZEnable = true;
      ZWriteEnable = true;
      ZFunc =Always;
      CullMode = None;

      PixelShader = compile ps_2_0 mainPS();
   }
}
```

cartoon Effect - GLSL bound

Sample

SIGGRAPH2005

```
//  texturedphong.fx  (only shown pixel parameters and entry point)

float4 fvAmbient <
  string UIName = "fvAmbient";
  string UIWidget = "Color";
  bool UIVisible =  true;
> = float4( 0.37, 0.37, 0.37, 1.00 );
float4 fvDiffuse <...> = float4( 0.89, 0.89, 0.89, 1.00 );
float4 fvSpecular <...> = float4( 0.49, 0.49, 0.49, 1.00 );
float fSpecularPower <...> = float( 25.00 );

texture base_Tex <string ResourceName = "Textures\\Fieldstone.tga";>;

sampler2D baseMap = sampler_state {
  Texture = (base_Tex);
  AddressU = WRAP;
  AddressV = WRAP;
};

struct PS_INPUT  {
  float2 Texcoord          :  TEXCOORD0;
  float3 ViewDirection     :  TEXCOORD1;
  float3 LightDirection    :  TEXCOORD2;
  float3 Normal            :  TEXCOORD3;
};

float4  TexturedPhongPS( PS_INPUT Input ) : COLOR0
{
  float3 fvLightDirection = normalize( Input.LightDirection );
  float3 fvNormal         = normalize( Input.Normal );
  float  fNDotL           = dot( fvNormal, fvLightDirection );
  float3 fvReflection     = normalize( ( ( 2.0f * fvNormal ) * ( fNDotL ) ) -
                            fvLightDirection );
  float3 fvViewDirection  = normalize( Input.ViewDirection );
  float  fRDotV           = max( 0.0f, dot( fvReflection, fvViewDirection ) );
  float4 fvBaseColor      = tex2D( baseMap, Input.Texcoord );
  float4 fvTotalAmbient   = fvAmbient * fvBaseColor;
  float4 fvTotalDiffuse   = fvDiffuse * fNDotL * fvBaseColor;
  float4 fvTotalSpecular  = fvSpecular * pow( fRDotV, fSpecularPower );
  return( saturate( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular ) );
}

technique TexturedPhong {
  pass Pass0  {
    VertexShader = compile vs_2_0 TexturedPhongVS();
    PixelShader = compile ps_2_0 TexturedPhongPS();
  }
}
```

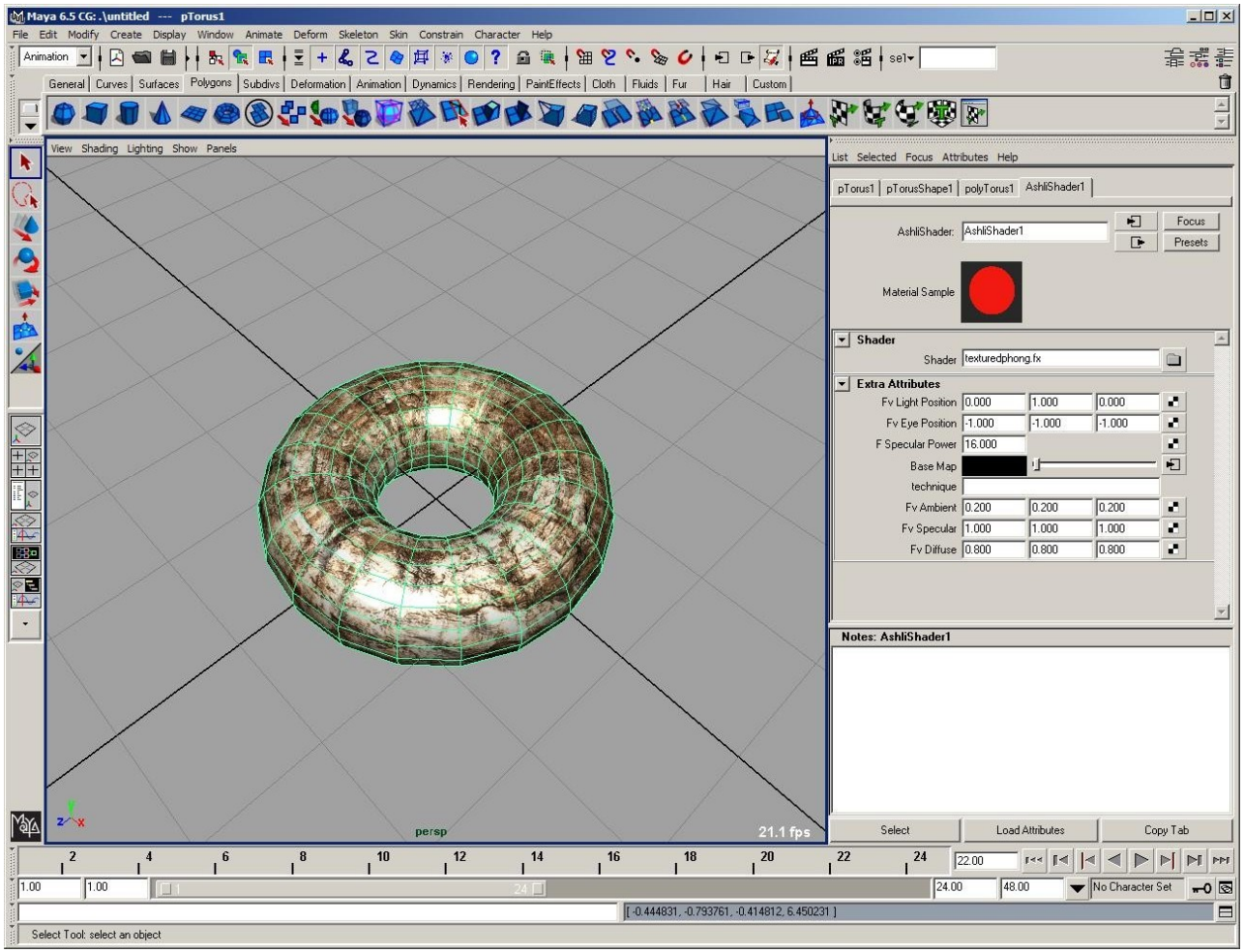
texturedphong Effect
HLSL bound

```
// Auto-generated GLSL file - do not edit!
// Pixel shader for program name: texturedphong

#include "GLSLOperator.h"

uniform vec3 fvEyePosition_0_0;
uniform vec4 fvAmbient_0_0;
uniform vec4 fvSpecular_0_0;
uniform vec4 fvDiffuse_0_0;
uniform float fSpecularPower_0_0;

uniform sampler2D baseMap_0_0;

void main()
{
  vec3 fvLightDirection;
  vec3 fvNormal;
  float fNDotL;
  vec3 fvReflection;
  vec3 fvViewDirection;
  float fRDotV;
  vec4 fvBaseColor;
  vec4 fvTotalAmbient;
  vec4 fvTotalDiffuse;
  vec4 fvTotalSpecular;

  fvLightDirection = Normalize(gl_TexCoord[2]);
  fvNormal = Normalize(gl_TexCoord[3]);
  fNDotL = DotProduct(fvNormal,fvLightDirection);
  fvReflection = Normalize((Subtract((Multiply(Multiply(2.,
                            fvNormal),fNDotL)),fvLightDirection)));
  fvViewDirection = Normalize(gl_TexCoord[1]);
  fRDotV = Max(0.,DotProduct(fvReflection,fvViewDirection));
  fvBaseColor = texture2D(baseMap_0_0,
                          gl_TexCoord[0]);
  fvTotalAmbient = Multiply(fvAmbient_0_0,
                            fvBaseColor);
  fvTotalDiffuse = Multiply(fvDiffuse_0_0,
                            Multiply(fNDotL,fvBaseColor));
  fvTotalSpecular = Multiply(fvSpecular_0_0,
                          Power(fRDotV,fSpecularPower_0_0));
  gl_FragData[0] = Clamp(Add(fvTotalAmbient,
                          Add(fvTotalDiffuse,fvTotalSpecular));
}
```

Ashli
HLSL->GLSL conversion

Maya Effect

SIGGRAPH2005



Streaming

- GPU incentive domains
- Auto generated description
  - Target platform seamless
- *AshliDI* -
  - Digital imaging streaming

AshliDI

- High performance
- Detail and quality
  - Extended dynamic range
- Stream based interface
  - Image tree, feedback

AshliDI (cnt'd)

SIGGRAPH2005

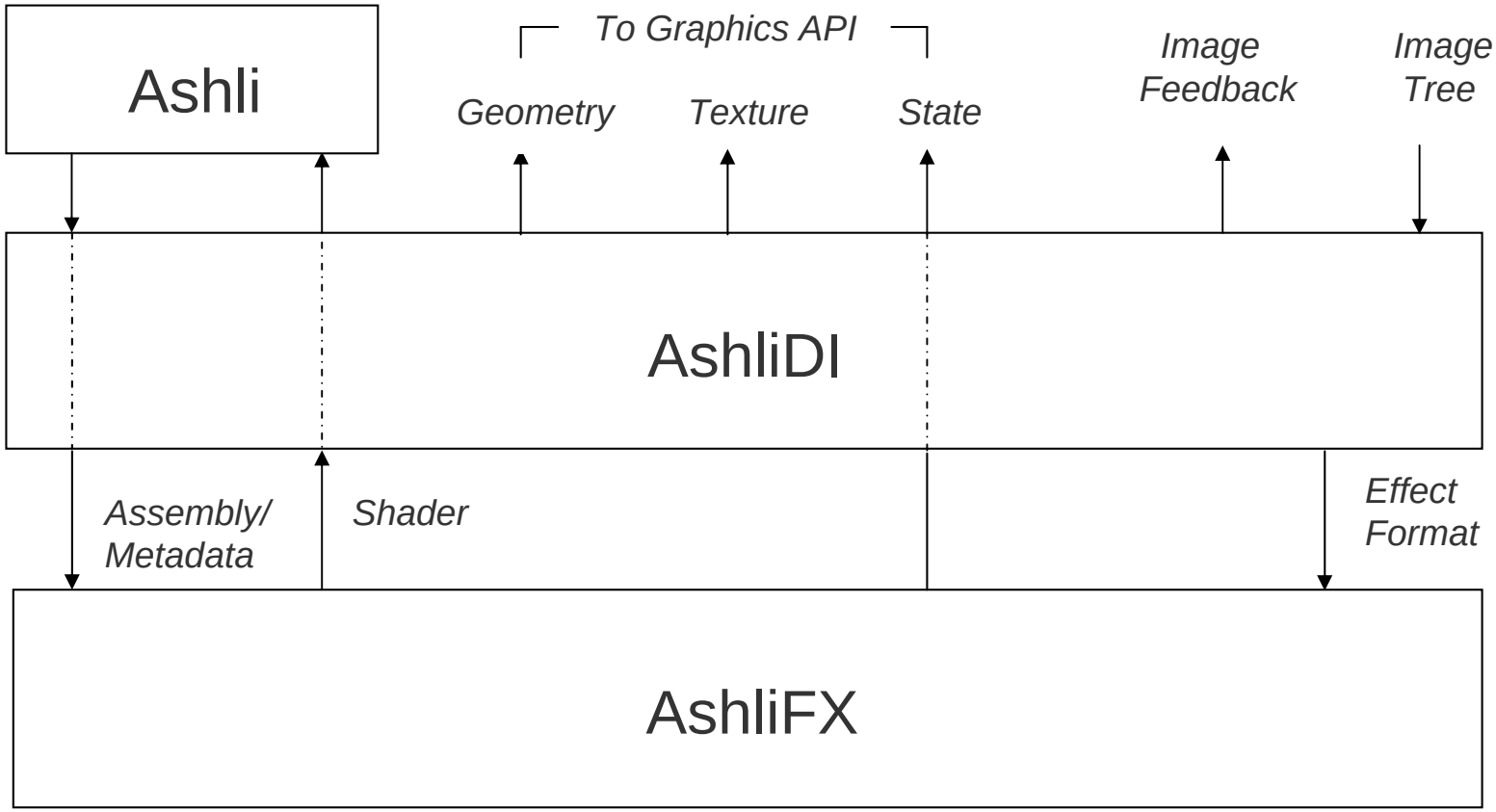
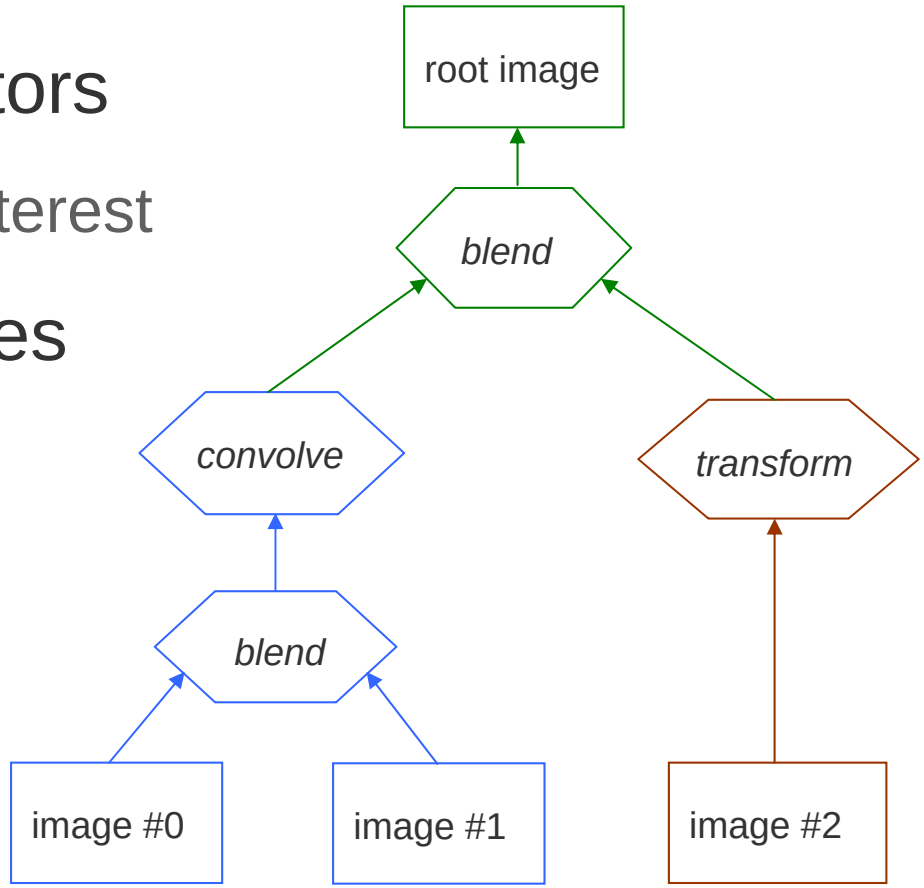


Image Tree

- Directed graph, no cycles
- Nodes - image operators
  - Attributes: region of interest
- Source images - leaves



Effect Mapping

- Tree to Effect transform
  - Constitutes rendering format
- Techniques - sub trees
  - Nodes onto passes, pipe state
- Seamless language binding

Sample

- Single node
  - Gaussian filter (5x5)
  - Region of interest

```
#define KERNEL_SIZE  5

float gaussianImpl( float2 f, float stddev;)
{
   float PI = 3.141592653589;
   return (1.0 / sqrt(2.0*PI*stddev) ) * exp( -(f.x*f.x + f.y*f.y) / (2.0*stddev) );
}

float4
gaussian (float2 pos : VPOS,
              uniform float3 cntrl,
              uniform sampler img) : COLOR0
{
   float4 col = float4(0,0,0,0);
   float weight = 0;

   float delta = 4.0 / KERNEL_SIZE;
   float2 filter = float2(-2, -2);
   float2 init = float2(pos.x - cntrl.x*( floor(KERNEL_SIZE/2),
                        pos.y - cntrl.x*( floor(KERNEL_SIZE/2));
   float2 pixel = init;

   float samples = KERNEL_SIZE*KERNEL_SIZE;
   float samp=0;
   float filter_cur = 0;
   while( samp < samples)  {
      filter_cur = gaussianImpl(filter, cntrl.z);
      weight += filter_cur;

      col += filter_cur * tex2D( img, pixel );
      pixel.x += cntrl.x;
      filter.x += delta;
      if(mod(samp, KERNEL_SIZE) == 0 ) {
         filter.x = -2;
         filter.y += delta;
         pixel.x = init.x;
         pixel.y += cntrl.y;
      }
      samp = samp + 1;
   }
   return (col / weight);
}
```

Rendering

- Predominant stream gather
- Intermediate image results
  - Previous render-to-texture
- 2D rendering API
  - Extensible to 3D for volumes
- Feedback storage stream

Outline

- Hiding shading languages
- Multi GPU shader partition
- Remapping GPU processors
- Source level debugger

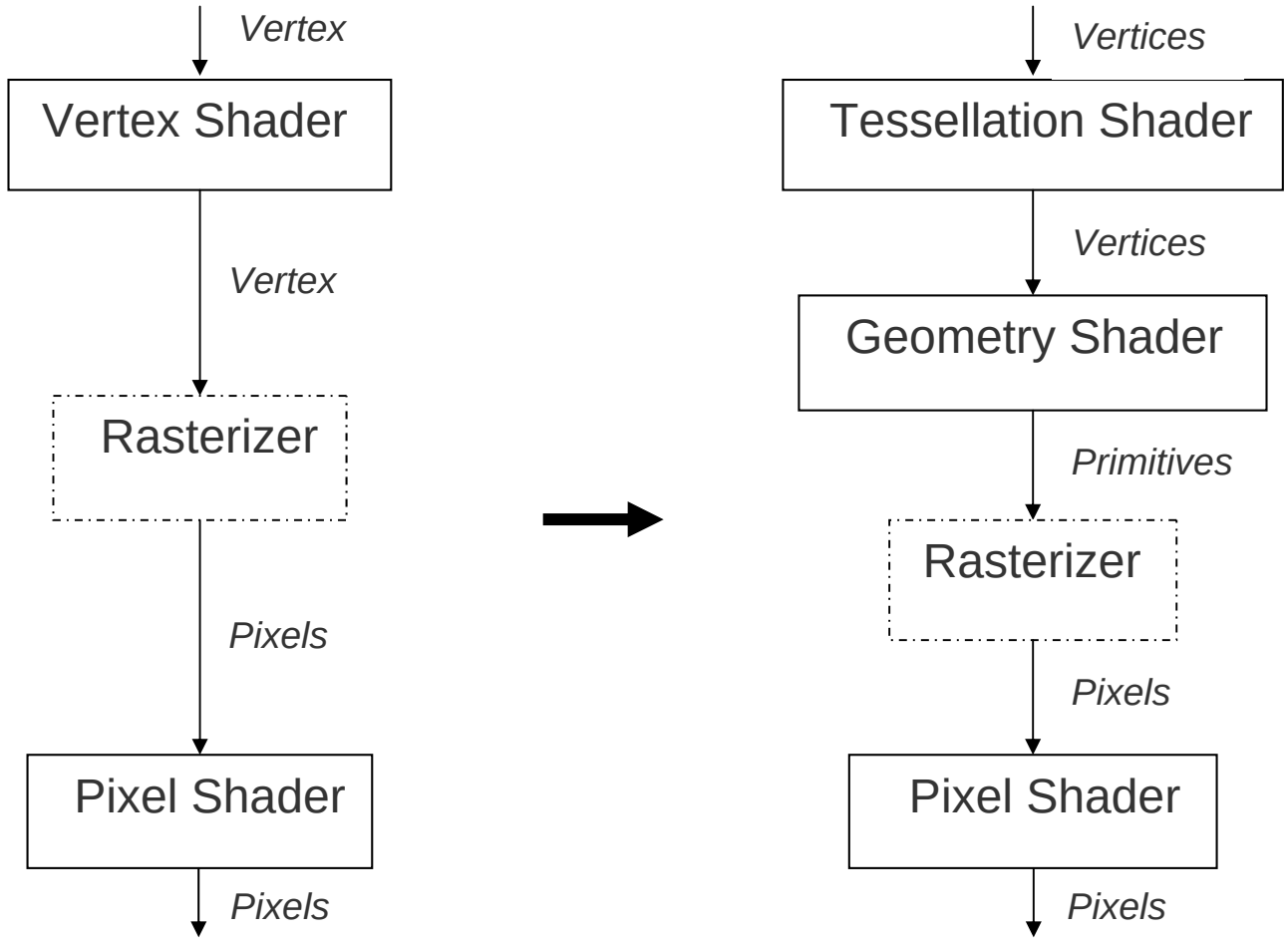
Multi GPU

- Multi GPU affordable
  - PCI Express reaching 4GB/sec
- Image, time based partition
  - Adaptive tiling more scalable
- Geometry, textures replicated
  - Vertex limited lower gains

Pipe Modality

- Vertex, pixel separate entities
  - Self resourced
- Concurrency higher on pixel
- Single amplification source
- Vertex turning into Geometry

Pipe Modality (cnt'd)



Vertex-Pixel Modality

Geometry-Pixel Modality

Geometry Modality

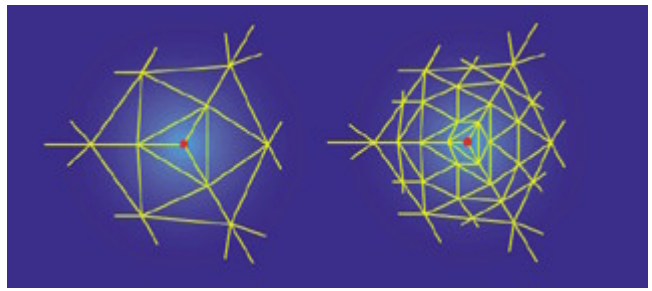
- Input, collection of vertices
  - Tessellation shader refines
- Geometry shader
  - Primitive input, output topology
- Multiplicity at top, mid pipe
- Inter shader communication

Load Balance

- Shared resources
  - Shader task scheduling
- Pipe dynamics
  - More degrees of interaction
- Walkthrough example
  - Motion blurred, displaced geometry

Displacement

- Triangular control mesh
  - Recursively refined



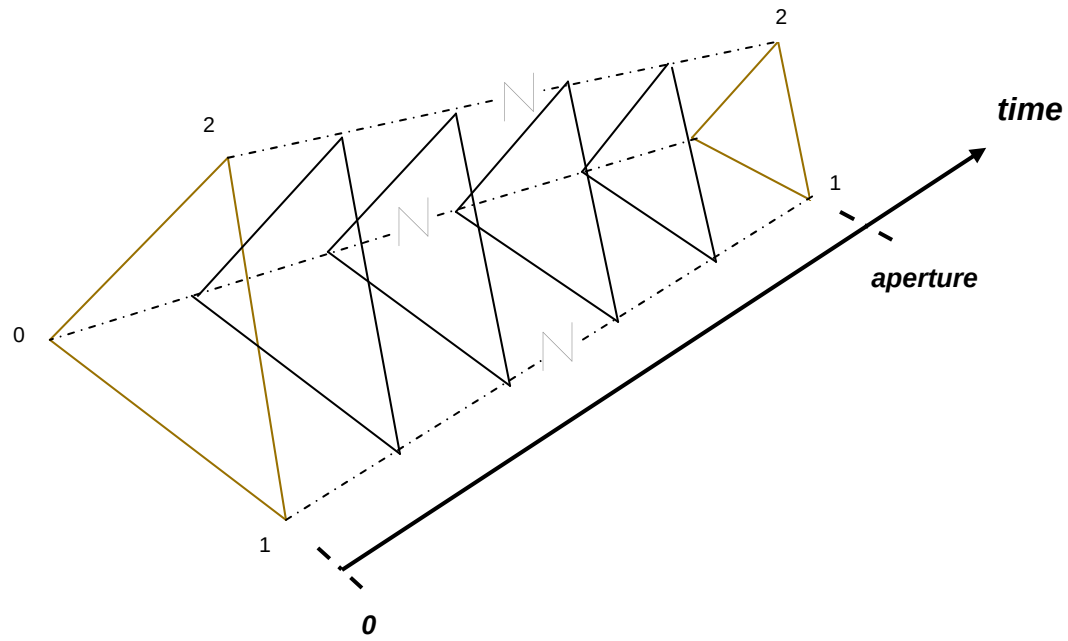
Courtesy Brian Sharp 2000

- Parametric space displacement
  - Neighbors for derivatives
  - Fine level of detail

Motion Blur

SIGGRAPH2005

- Motion blur on geometry shader
  - Aperture triangle pair
  - Samples interposed in hull



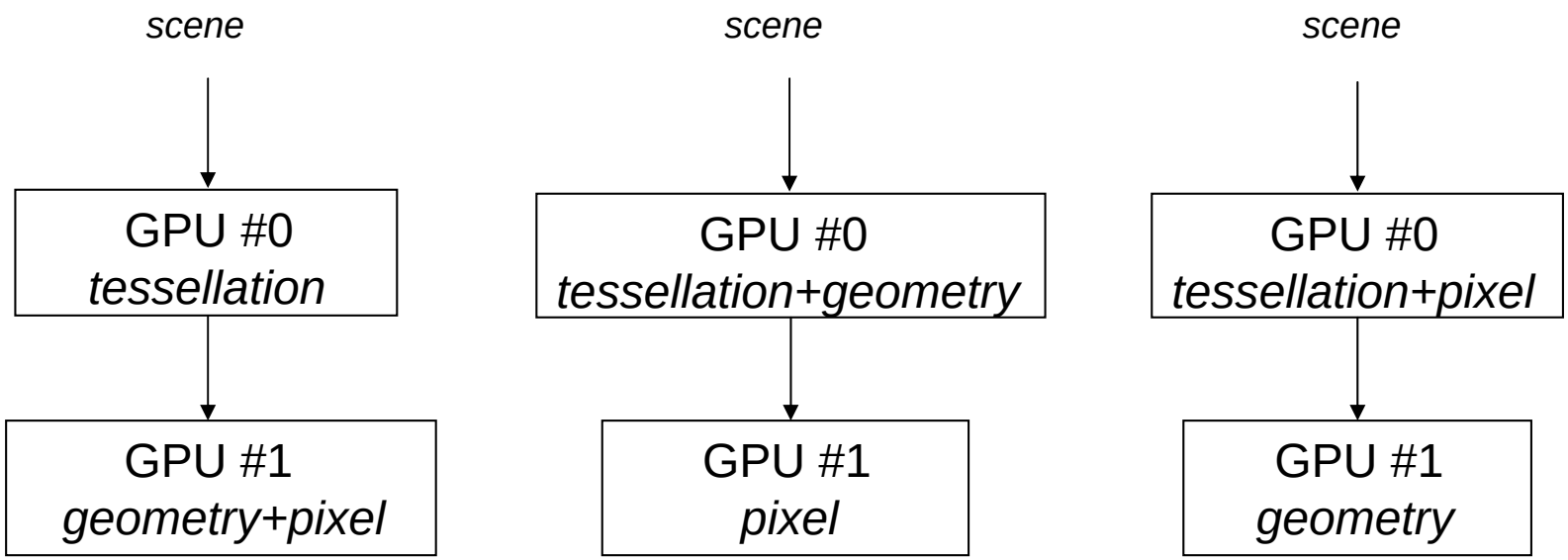
Distribution

- Geometry critical path
- Pipe behavior -
  - Shader level macro threads
  - Shader results exposed
- GPU fed in pipe manner
  - Single copy scene description

Partition

SIGGRAPH2005

- Two GPUs pairing options:



inter GPU copy	1	1	2

Programming

- Automate shader partition
  - Task based
- Compilers figure shader cost
  - Amplification factor, copy
- Evolve multi GPU API
  - Simple, no user intervention

Outline

- Hiding shading languages
- Multi GPU shader partition
- Remapping GPU processors
- Source level debugger

Shader Model 3.0

- Consistent instruction set
  - Vertex, pixel little adversity
- Vertex texture fetch
- Retargeting pixel onto vertex
  - Seemingly underutilized vertex
- Pixel exploits higher parallelism

Retargeting

- Vertex, pixel resource match
  - 4 vs. 16 samplers, respectively
- Automate processor remapping
- Ashli *vertex-to-pixel* conversion
  - Exploits multipass
- Analyze performance trade offs

Render to Vertex Buffer

- Vertex streams as textures
- Vertex format
  - Attributes of any type

```
struct VertexInput {
    float4 pos          : POSITION;
    float3 normal       : NORMAL;
    float color         : COLOR0;
    float2 tex0         : TEXCOORD0;
};
```

- *Packed* and *Unpacked*
  - Vertex storage format

Storage Format

- Contiguous vertex
  - Addressing: base, attribute stride
  - Per component fetch
- Padded vertex
  - Four component IEEE float
  - Single attribute pointer

Inputs

- Vertex and pixel inputs
  - 16 vs. 10, respectively
- Vertex buffer fetch
  - Pixel texture access
- Mapping criteria
  - vertex inputs + samplers <= 16

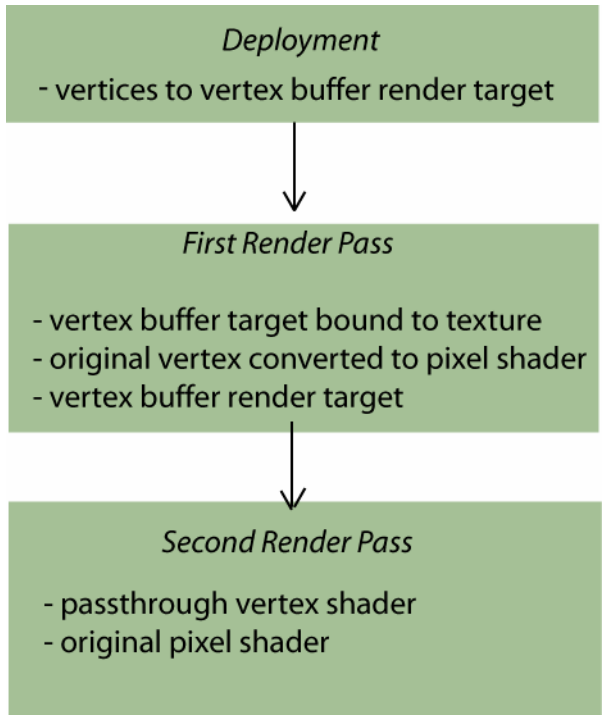
Outputs

- Vertex and pixel outputs
  - 12 vs. 4, respectively
- Ashli incorporates
  - Pixel *Virtual Color Outputs*
- Color outputs exceeding cap
  - Code segmentation

Vertex Output	Pixel Output
Position	C0
Color	C1
SecondaryColor	C2
TexCoord0	C3
TexCoord1	C4
TexCoord2	C5
TexCoord3	C6
TexCoord4	C7
TexCoord5	C8
TexCoord6	C9
TexCoord7	C10
Fog	C11

Passes

- Two pass rendering
- Vertex-to-pixel
  - Texture vertex fetch
  - Processing
  - Output mapping
- *Passthrough* vertex
  - Inverse output mapping



Sample

- HLSL vertex shader
  - Vertex texture
- Ashli emits detached shaders
  - Vertex-to-pixel, passthrough
- Metadata
  - Inverse output mapping

Sample (cnt'd)

SIGGRAPH2005

```
struct VertexInput {
   float4 pos          : POSITION;
   float2 tex          : TEXCOORD0;
};

struct VertexOutput {
   float4 pos          : POSITION;
   float  energy       : TEXCOORD1;
};

float4x4 mWorldViewProjection;
float3x3 mWorldInverseTranspose;
sampler shadow;

VertexOutput
main(VertexInput vi)
{
   VertexOutput vo;
   vo.energy = 1.0f - tex2Dlod(shadow, float4(vi.tex, 0, 1)).x;
   vo.pos = mul(vi.pos, mWorldViewProjection);
   return vo;
}
```

HLSL vertex shader

```
begin fragment
t 0 -1 _T0_
s 0 2D File shadow
s 1 2D File attr0
s 2 2D File attr1
c 3 -1 mWorldViewProjection_0_0[0] 0 0 0 0
c 2 -1 mWorldViewProjection_0_0[1] 0 0 0 0
c 1 -1 mWorldViewProjection_0_0[2] 0 0 0 0
c 0 -1 mWorldViewProjection_0_0[3] 0 0 0 0
o 0 -1 _C0_
o 1 -1 _C4_
end
begin vertex
a 0 -1 _Vertex_
a 9 -1 _TexCoord1_
o 0 -1 _Position_
t 1 -1 _T1_
end
```

metadata

```
ps_3_0

dcl_texcoord0 v2.rgba
dcl_2d s0
dcl_2d s1
dcl_2d s2

def c0, 0, 1, 0, 0
def c11, 0, 0, 0, 1

dp3 r0, t0, c10
mov r1, c0.r
frc r1.r, r0.r
add r1.g, r0.r, -r1.r
mul r1.g, r1.g, c10.a
------------------------
texld r0, r1, s2
texld r1, r2, s2

dp3 r2, t0, c8
mov r3, c0.r
frc r3.r, r2.r
add r3.g, r2.r, -r3.r
mul r3.g, r3.g, c8.a
------------------------
texld r1, r3, s1
texld r2, r4, s1
texld r3, r5, s1
texld r4, r6, s1

mov r2, c11
mov r2.r, r0.r
mov r2.g, r1.r
mov r0, c0.r
mov r0.rg, r2
mov r0.b, c0.r
mov r0.a, c0.g
texldl r0, r0, s0
mov r5, c0.r
mov r5.r, r1.r
mov r5.g, r2.r
mov r5.b, r3.r
mov r5.a, r4.r
mov r1, c0.r
dp4 r1.r, c4, r5
dp4 r1.g, c3, r5
dp4 r1.b, c2, r5
dp4 r1.a, c1, r5
add r0, c0.g, -r0.r

mov oC0, r1
mov oC1, r0
```

vertex-to-pixel

```
vs_3_0

dcl_position0 v0
dcl_texcoord1 v9

dcl_position o0
dcl_texcoord1 o4

mov o0, v0
mov o4, v9
```

passthrough vertex

Performance

- Two pass overhead
  - Deployment and recirculation
- Speedup observed
  - Larger mesh size
  - Higher compute to fetch ratio
- Operate on vertex collection

Outline

- Hiding shading languages
- Multi GPU shader partition
- Remapping GPU processors
- Source level debugger

Tools

- Debugging increasingly important
  - Long, complex shaders
- Microsoft Visual Studio .NET
  - High level and assembly
  - File/line # and pixel area
  - No direct hardware

Tools (cnt'd)

- Shadesmith
  - Fragment assembly, on hardware
  - Register watch, inline editing
  - Platform dependent
- Source high level - Ashli

Ashli

- Language orthogonal
- Inspecting and editing code
  - Less so for hardware savvy
- Runs on graphics hardware
  - Pixel/Fragment shader
- Visual validation

API

- File/line # break points
- Debugger exposure
  - Add/remove break point
  - Continue, single step
  - Query current break point

Sample

SIGGRAPH2005

```
float sqr(float x)
{
   return x*x;
}

float dist (float x,y)
{
   return (sqrt(sqr(x)+sqr(y)));
}

float Gaussian(float x,m,y)
{
   return (exp(-sqr((x)-(m))/sqr(y)));
}

surface spiderweb(
          float    cRadialWebs    = 7;
          float    cCrossWebs     = 10;
          float    rWebDiam       = .01;
          float    rWebCenterT    = 0, rWebCenterS   = 0)
{
   float TT=(t-rWebCenterT);
   float SS=(s-rWebCenterS);
   float rRadius = dist(SS,TT);
   float rAngle=atan(SS,TT);
   float rAngleStep=2*PI/cRadialWebs;
   float rDist;
   float rAccum=0;
   float cRadSeg;
   float cCrossSeg;

   /* Calculate which radial section its in */
   cRadSeg=floor(rAngle/rAngleStep);

   /* Color Radial Webs */
   rAccum+=Gaussian(rAngle,
                    cRadSeg*rAngleStep,
                    rWebDiam / rRadius);
   rAccum+=Gaussian(rAngle,
          mod((cRadSeg+1),cRadialWebs)*rAngleStep,
          rWebDiam / rRadius);
   rDist=(SS * cos((cRadSeg+.5)*rAngleStep) +
          TT * sin((cRadSeg+.5)*rAngleStep));

   /* Calculate which cross section its in */
   cCrossSeg=floor(rDist * cCrossWebs / 2);

   /* Color Tangential Webs */
   rAccum+=Gaussian(rDist,
          cCrossSeg/cCrossWebs * 2,
          rWebDiam / rRadius);
   rAccum+=Gaussian(rDist,
          (cCrossSeg+1)/cCrossWebs*2,
          rWebDiam / rRadius);

   /* clamp to [0,1] */
   rAccum=clamp(rAccum,0,1);

   Ci=Os * mix( color(0,0,0), Cs, rAccum);
   Oi=Os * rAccum;
}
```

RenderMan spiderweb shader

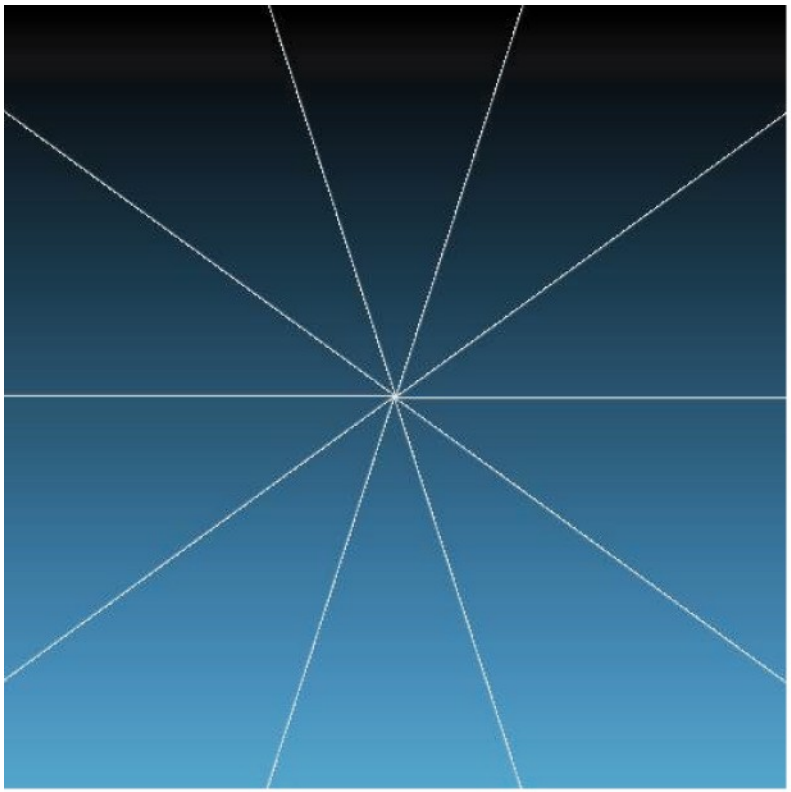
```
ps_3_0

// Instructions (Alu):        53

dcl_texcoord0 v2.rgba

def c0, c2,  series factors
def c3, -0.11573, 0.0519506, 1, 0

add r0.r, v2.g, -c1.a
abs r0.g, r0.r
add r0.b, v2.r, -c1.b
abs r0.a, r0.b
cmp r1.r, -r0.b, r0.b, c3.b
cmp r1.r, r0.b, r1.r, -c3.b
mul r1.g, r1.r, c0.a
cmp r0.a, -r0.a, c3.a, r1.g
rcp r1.g, r0.r
mul r1.g, r0.b, r1.g
abs r1.b, r1.g
cmp r1.g, -r0.r, r1.b, r1.g
abs r1.b, r1.g
add r1.b, c3.b, -r1.b
rcp r1.a, r1.g
cmp r1.a, r1.b, r1.g, r1.a
mul r2.r, r1.a, r1.a
mov r3, c0
mad r2.g, r2.r, r3.g, c3.g
mad r2.g, r2.r, r2.g, c3.r
mad r2.g, r2.r, r2.g, c2.a
mad r2.g, r2.r, r2.g, c2.b
mad r2.r, r2.r, r2.g, c2.g
mul r1.a, r2.r, r1.a
cmp r2.r, -r1.g, r1.g, c3.b
cmp r1.g, r1.g, r2.r, -c3.b
mad r1.g, r1.g, c0.a, -r1.a
cmp r1.g, r1.b, r1.a, r1.g
add r1.b, c2.r, -r1.g
mul r1.r, r1.r, r1.b
cmp r1.r, -r0.r, r1.r, r1.g
cmp r0.g, -r0.g, r0.a, r1.r
rcp r0.a, c1.g
mul r0.a, r0.a, c0.r
rcp r1.r, r0.a
mul r1.r, r0.g, r1.r
frc r1.g, r1.r
add r1.r, r1.r, -r1.g
mad r0.g, -r1.r, r0.a, r0.g
mul r0.g, r0.g, r0.g
mul r0.r, r0.r, r0.r
mad r0.r, r0.b, r0.b, r0.r
rsq r0.b, r0.r
mul r0.r, r0.r, r0.b
rcp r0.r, r0.r
mul r0.r, c1.r, r0.r
mul r0.r, r0.r, r0.r
rcp r0.r, r0.r
mul r0.r, r0.g, r0.r
mul r0.r, -r0.r, c0.b
exp r0.r, r0.r
mov r1, r0.r
mov oC0, r1
```

code @ first break point

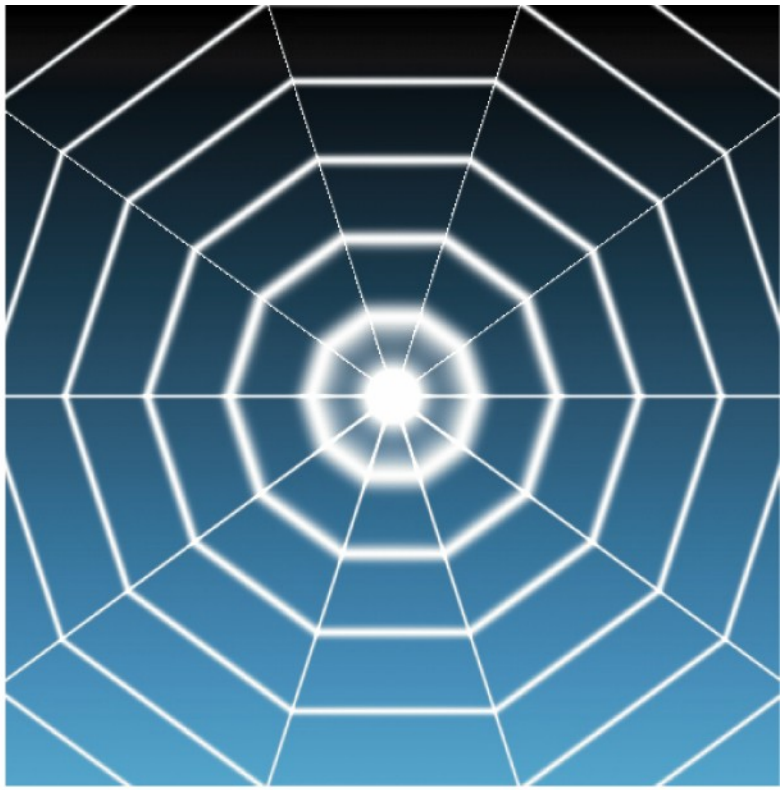
```
ps_3_0

// Instructions (Alu):        60

dcl_texcoord0 v2.rgba

def c0-c5, series factors
def c6, 0, 0, 0, 0

add r0.r, v2.r, -c1.b
add r0.g, v2.g, -c1.a
abs r0.b, r0.g
abs r0.a, r0.r
cmp r1.r, -r0.r, r0.r, c5.a
cmp r1.r, r0.r, r1.r, -c5.a
mul r1.g, r1.r, c4.r
cmp r0.a, -r0.a, c6.r, r1.g
rcp r1.g, r0.g
mul r1.g, r0.r, r1.g
abs r1.b, r1.g
cmp r1.g, -r0.g, r1.b, r1.g
abs r1.b, r1.g
add r1.b, c5.a, -r1.b
rcp r1.a, r1.g
cmp r1.a, r1.b, r1.g, r1.a
mul r2.r, r1.a, r1.a
mov r3, c0
mad r2.g, r2.r, r3.g, c5.b
mad r2.g, r2.r, r2.g, c5.g
mad r2.g, r2.r, r2.g, c5.r
mad r2.g, r2.r, r2.g, c4.a
mad r2.r, r2.r, r2.g, c4.b
mul r1.a, r2.r, r1.a
cmp r2.r, -r1.g, r1.g, c5.a
cmp r1.g, r1.g, r2.r, -c5.a
mad r1.g, r1.g, c4.r, -r1.a
cmp r1.g, r1.b, r1.a, r1.g
add r1.b, c4.g, -r1.g
mul r1.r, r1.r, r1.b
cmp r1.r, -r0.g, r1.r, r1.g
cmp r0.b, -r0.b, r0.a, r1.r
rcp r0.a, c1.g
mul r0.a, r0.a, c0.r
rcp r1.r, r0.a
mul r0.b, r0.b, r1.r
frc r1.r, r0.b
add r0.b, r0.b, -r1.r
add r0.b, r0.b, c3.a
mul r0.b, r0.b, r0.a
mul r0.a, r0.b, r0.b
mad r1.r, c3.b, r0.a, c3.g
mad r1.r, r1.r, r0.a, c3.r
mad r1.r, r1.r, r0.a, c2.a
mad r1.r, r1.r, r0.a, c2.b
mad r1.r, r1.r, r0.a, c5.a
mad r1.g, c2.g, r0.a, c2.r
mad r1.g, r1.g, r0.a, c0.a
mad r1.g, r1.g, r0.a, c0.b
mad r0.a, r1.g, r0.a, c5.a
mul r0.b, r0.a, r0.b
mul r0.g, r0.g, r0.b
mad r0.r, r0.r, r1.r, r0.g
mov r1, c1
mul r0.g, r1.r, c3.a
mul r0.r, r0.r, r0.g
frc r0.g, r0.r
add r0.r, r0.r, -r0.g
mov r1, r0.r
mov oC0, r1
```

code @ second break point

Visual Validation



rendered image @ first break point



rendered image of complete shader

Break Point

- Two pass
  - Output substitution
  - Replace lhs with color output
- Degenerated tree nodes
  - Break point to root
- Break point delimited program
  - Valid sub tree

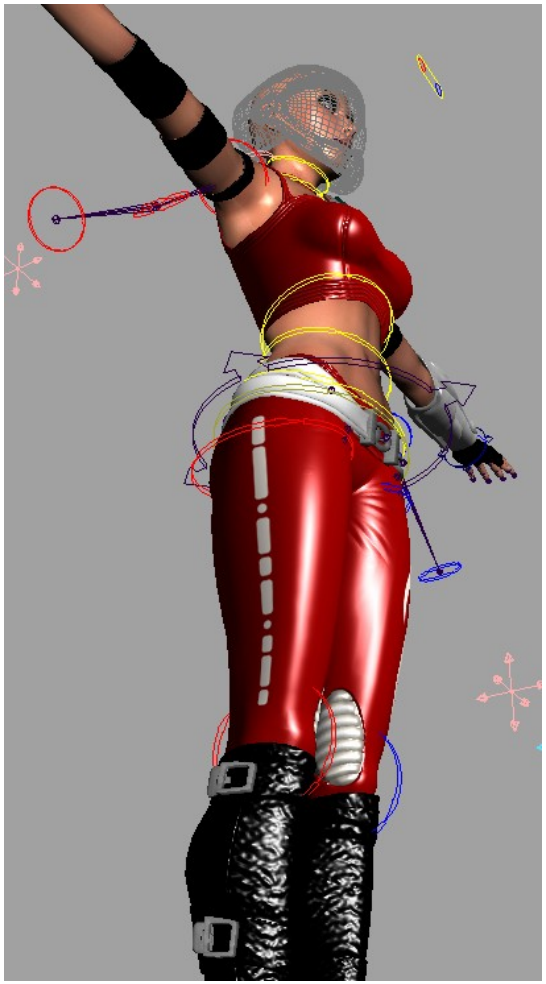
Flow Control

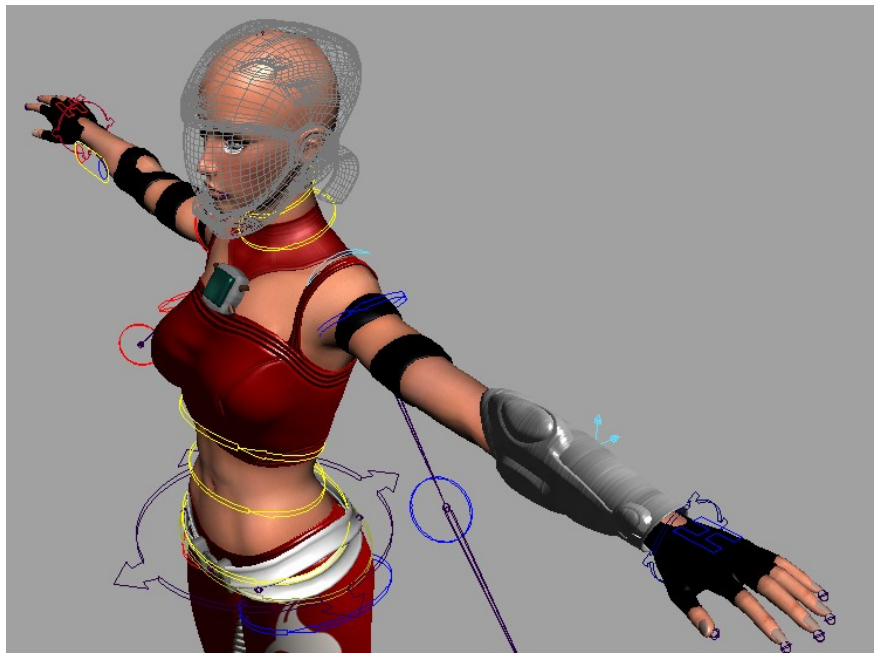
- Break point inside conditional
  - Replace lhs inside if (and else)
  - Replace lhs before conditional
- Break point in a loop
  - Conditional unrolling
  - Count set to a cap

Priorities

- Runs on hardware
- Tailored to audience
  - Content creator or
  - Hardware intimate
- Performance not critical

Demo





Summary

- Domain specific streaming
  - Hiding shading languages
- Multi GPU load distribution
  - Task based, proper API
- GPU processor virtualization
- Debugger, seriously taken

Info

- Ashli/AshliFX on multi systems
  - 32/64 Windows and Linux, Mac OS X
- Link, contact:
  - http://www.ati.com/developer/ashli.html
  - devrel@ati.com
- Acknowledgement:
  - Raja Koduri, Evan Hart, Joshua Barczak, Nikki Lukas