Renaissance: Next-Generation Real-Time Shading Language

Why another real-time shading language?

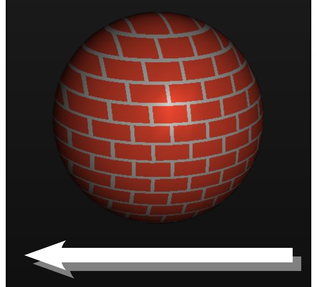
- **A better model.** Existing languages borrow heavily from C, which is a poor model for stream-oriented processing. Let's borrow instead from modern, pure functional languages.
- **Better signal-to-noise ratio.** We believe unification of the three stages of graphics processing (CPU, vertex processing, fragment processing) improves clarity at no cost in performance.
- **Shading components.** Existing languages don't facilitate combining shading concepts as in the fixed function pipeline. Renaissance effectively lets us glEnable(GL_SKELETAL_ANIMATION).
- **Programming is a human task.** We believe we can successfully apply concepts from Psychology of Programming and Human Computer Interaction to the design of a new programming language.

Goals

- Apply HCI principles to functional programming in the relatively new context of shader programs.
- Facilitate efficient and natural composition of shader components. You should be able to enable bone deformation, per-pixel or per-vertex lighting, and multiple lights and light types simply by enabling some flags, just as the fixed function pipeline does.
- Borrow, from metaprogramming shader languages such as Sh and Vertigo, the greatly simplified data transfer interface between the application and the shader.
- Raise the bar in shading language evolution by contributing higher-order functions, expression rewriting (normalize (normalize x) -> normalize x), automatic type inference, and staged computation.

```
# Uniforms.

uniform vec3 LightPosition
uniform vec3 BrickColor
uniform vec3 MortarColor
uniform vec2 BrickSize
uniform vec2 BrickPct

# Constants.

SpecularContribution = 0.3
DiffuseContribution = 1.0 - SpecularContribution

# Transform.

gl_Position = ftransform
ecPosition = (gl_ModelViewMatrix * gl_Vertex).xyz
tnorm = normalize (gl_NormalMatrix * gl_Normal)

# Lighting.

lightVec   = normalize (LightPosition - ecPosition)
reflectVec = reflect (-lightVec) tnorm
viewVec    = normalize (-ecPosition)

diffuse = max (dot lightVec viewVec) 0.0
spec = if (diffuse > 0.0) then s else 0.0
    where s = pow (max (dot reflectVec viewVec) 0.0) 16.0
LightIntensity = DiffuseContribution * diffuse +
                 SpecularContribution * specular

# Brick.

position = gl_Vertex.xy / BrickSize + (vec2 xoffset 0.0)
    where xoffset = if fract (position.y * 0.5) > 0.5 then 0.5 else 0.0
useBrick = step (fract position) BrickPct

color = mix MortarColor BrickColor amount
    where amount = useBrick.x * useBrick.y * LightIntensity
gl_FragColor = color ++ 1.0
```

Example Renaissance Brick Shader



```
uniform vec3 LightPosition;

const float SpecularContribution = 0.3;
const float DiffuseContribution = 1.0 - SpecularContribution;

varying float LightIntensity;
varying vec2  MCposition;

void main(void)
{
    vec3 ecPosition = vec3(gl_ModelViewMatrix * gl_Vertex);
    vec3 tnorm      = normalize(gl_NormalMatrix * gl_Normal);
    vec3 lightVec   = normalize(LightPosition - ecPosition);
    vec3 reflectVec = reflect(-lightVec, tnorm);
    vec3 viewVec    = normalize(-ecPosition);
    float diffuse   = max(dot(lightVec, tnorm), 0.0);
    float spec      = 0.0;

    if (diffuse > 0.0)
    {
        spec = max(dot(reflectVec, viewVec), 0.0);
        spec = pow(spec, 16.0);
    }

    LightIntensity = DiffuseContribution * diffuse +
                     SpecularContribution * spec;

    MCposition = gl_Vertex.xy;
    gl_Position = ftransform();
}
```

GLSL Brick Vertex Shader

```
uniform vec3 BrickColor, MortarColor;
uniform vec2 BrickSize;
uniform vec2 BrickPct;

varying vec2  MCposition;
varying float LightIntensity;

void main(void)
{
    vec3 color;
    vec2 position, useBrick;

    position = MCposition / BrickSize;

    if (fract(position.y * 0.5) > 0.5)
        position.x += 0.5;

    position = fract(position);

    useBrick = step(position, BrickPct);

    color  = mix(MortarColor, BrickColor, useBrick.x * useBrick.y);
    color *= LightIntensity;
    gl_FragColor = vec4(color, 1.0);
}
```

GLSL Brick Fragment Shader

Chad Austin & Dr. Dirk Reiners
Human Computer Interaction
Iowa State University

human computer interaction | HCI

VIRTUAL REALITY APPLICATIONS CENTER