GPU Shading and Rendering: OpenGL Shading Language

Marc Olano

Computer Science and Electrical Engineering
University of Maryland, Baltimore County

SIGGRAPH 2005

Outline

OpenGL

- Derived from SGI GL in 1992
- Cross-platform, Cross-vendor
- Architecture Review Board
  - Apple, ATI, Dell, IBM, Intel, NVIDIA, SGI, Sun, 3Dlabs
- Core, Extensions

OpenGL

- Derived from SGI GL in 1992
- Cross-platform, Cross-vendor
- Architecture Review Board
  - Apple, ATI, Dell, IBM, Intel, NVIDIA, SGI, Sun, 3Dlabs
- Core, Extensions

OpenGL

- Derived from SGI GL in 1992
- Cross-platform, Cross-vendor
- Architecture Review Board
  - Apple, ATI, Dell, IBM, Intel, NVIDIA, SGI, Sun, 3Dlabs
- Core, Extensions
  - Vendor extensions, EXT, ARB

OpenGL

- Derived from SGI GL in 1992
- Cross-platform, Cross-vendor
- Architecture Review Board
  - Apple, ATI, Dell, IBM, Intel, NVIDIA, SGI, Sun, 3Dlabs
- Core, Extensions
  - Vendor extensions, EXT, ARB

OpenGL

- Derived from SGI GL in 1992
- Cross-platform, Cross-vendor
- Architecture Review Board
  - Apple, ATI, Dell, IBM, Intel, NVIDIA, SGI, Sun, 3Dlabs
- Core, Extensions
  - Vendor extensions, EXT, ARB

OpenGL

- Derived from SGI GL in 1992
- Cross-platform, Cross-vendor
- Architecture Review Board
  - Apple, ATI, Dell, IBM, Intel, NVIDIA, SGI, Sun, 3Dlabs
- Core, Extensions
  - Vendor extensions, EXT, ARB

OpenGL Shading Language

- Originally proposed by 3DLabs
- ARB extension in OpenGL 1.5
- Core feature in OpenGL 2.0

OpenGL Shading Language

- Originally proposed by 3DLabs
- ARB extension in OpenGL 1.5
- Core feature in OpenGL 2.0

OpenGL Shading Language

- Originally proposed by 3DLabs
- ARB extension in OpenGL 1.5
- Core feature in OpenGL 2.0

Similarities to HLSL/Cg

- High-level language
- Inspired by C & RenderMan
- Float & vector types & operations
- RenderMan-inspired shading & math functions

Similarities to HLSL/Cg

- High-level language
- Inspired by C & RenderMan
- Float & vector types & operations
- RenderMan-inspired shading & math functions

Similarities to HLSL/Cg

- High-level language
- Inspired by C & RenderMan
- Float & vector types & operations
- RenderMan-inspired shading & math functions

Similarities to HLSL/Cg

- High-level language
- Inspired by C & RenderMan
- Float & vector types & operations
- RenderMan-inspired shading & math functions

Interesting features

- Compiler built-into driver
  - Enables vendor optimizations
- Virtualization
  - Compiler handles long shaders
- Direct access to built-in state
  - OpenGL state
  - vertex & fragment input & output
- New arbitrary parameters
  - vertex input
  - vertex/fragment communication

Interesting features

- Compiler built-into driver
  - Enables vendor optimizations
- Virtualization
  - Compiler handles long shaders
- Direct access to built-in state
  - OpenGL state
  - vertex & fragment input & output
- New arbitrary parameters
  - vertex input
  - vertex/fragment communication

Interesting features

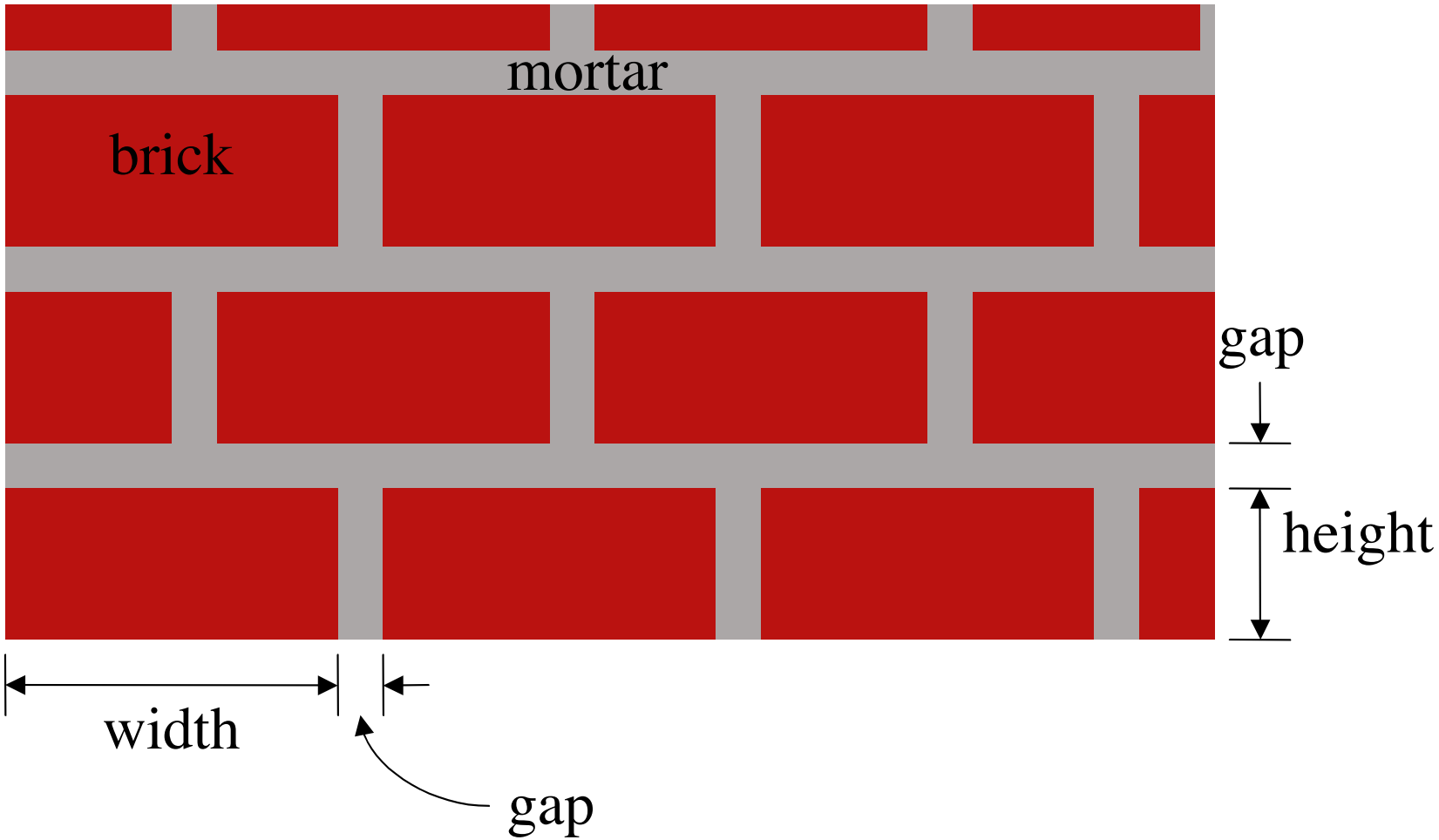
- Compiler built-into driver
  - Enables vendor optimizations
- Virtualization
  - Compiler handles long shaders
- Direct access to built-in state
  - OpenGL state
  - vertex & fragment input & output
- New arbitrary parameters
  - vertex input
  - vertex/fragment communication

Interesting features

- Compiler built-into driver
  - Enables vendor optimizations
- Virtualization
  - Compiler handles long shaders
- Direct access to built-in state
  - OpenGL state
  - vertex & fragment input & output
- New arbitrary parameters
  - vertex input
  - vertex/fragment communication

Outline

Simple Brick



OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
- Clip/Projection space: parallel; view down -z

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z
  - gl_ModelViewProjectionMatrix * gl_Vertex

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z
  - gl_ModelViewProjectionMatrix * gl_Vertex
  - Eye at vec4(0,0,1,0)

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z
  - gl_ModelViewProjectionMatrix * gl_Vertex
  - Eye at vec4(0,0,1,0)

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z
  - gl_ModelViewProjectionMatrix * gl_Vertex
  - Eye at vec4(0,0,1,0)

OpenGL spaces

vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);

- Model space: object vertices
- View space: common space for lighting
  - gl_ModelViewMatrix * gl_Vertex
  - Nominally E=vec4(0,0,0,1); view down -z
  - But not required!
- Clip/Projection space: parallel; view down -z
  - gl_ModelViewProjectionMatrix * gl_Vertex
  - Eye at vec4(0,0,1,0)

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

Homogeneous Subtraction

vec3 I = P.xyz*E.w - E.xyz*P.w;

- Common to see P.xyz/P.w - E.xyz/E.w
- Same as (P.xyz-E.xyz)/(P.w*E.w)
- Homogeneous result vector: vec4(P.xyz-E.xyz,P.w*E.w)
  - Works even if P.w or E.w = 0!
- Only care about direction
  - So ignore w vector scale factor
  - Also safe to ignore if normalizing

Homogeneous Subtraction

vec3 I = P.xyz*E.w - E.xyz*P.w;

- Common to see P.xyz/P.w - E.xyz/E.w
- Same as (P.xyz-E.xyz)/(P.w*E.w)
- Homogeneous result vector: vec4(P.xyz-E.xyz,P.w*E.w)
  - Works even if P.w or E.w = 0!
- Only care about direction
  - So ignore w vector scale factor
  - Also safe to ignore if normalizing

Homogeneous Subtraction

vec3 I = P.xyz*E.w - E.xyz*P.w;

- Common to see P.xyz/P.w - E.xyz/E.w
- Same as (P.xyz-E.xyz)/(P.w*E.w)
- Homogeneous result vector: vec4(P.xyz-E.xyz,P.w*E.w)
  - Works even if P.w or E.w = 0!
- Only care about direction
  - So ignore w vector scale factor
  - Also safe to ignore if normalizing

Homogeneous Subtraction

vec3 I = P.xyz*E.w - E.xyz*P.w;

- Common to see P.xyz/P.w - E.xyz/E.w
- Same as (P.xyz-E.xyz)/(P.w*E.w)
- Homogeneous result vector: vec4(P.xyz-E.xyz,P.w*E.w)
  - Works even if P.w or E.w = 0!
- Only care about direction
  - So ignore w vector scale factor
  - Also safe to ignore if normalizing

Homogeneous Subtraction

vec3 I = P.xyz*E.w - E.xyz*P.w;

- Common to see P.xyz/P.w - E.xyz/E.w
- Same as (P.xyz-E.xyz)/(P.w*E.w)
- Homogeneous result vector: vec4(P.xyz-E.xyz,P.w*E.w)
  - Works even if P.w or E.w = 0!
- Only care about direction
  - So ignore w vector scale factor
  - Also safe to ignore if normalizing

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader

- Transform & Light

```
void main(void) {
    vec4 P = gl_ModelViewMatrix * gl_Vertex;
    vec4 E = gl_ProjectionMatrixInverse * vec4(0,0,1,0);
    vec3 I = P.xyz*E.w - E.xyz*P.w;
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 Nf = normalize(faceforward(N,I,N));
    /* [accumulate light contribution] */
    gl_TexCoord[0] = gl_TextureMatrix[0] *
        gl_MultiTexCoord0;
    gl_Position = ftransform();
}
```

OpenGL Vertex Shader (II)

- Accumulate light contribution

```
gl_FrontColor = vec4(0,0,0,1);
for (int i=0; i < gl_MaxLights; i++) {
    vec3 L = normalize(
        gl_LightSource[i].position.xyz*P.w -
        P.xyz*gl_LightSource[i].position.w);
    gl_FrontColor.xyz +=
        gl_LightSource[i].ambient +
        gl_LightSource[i].diffuse*
        max(dot(Nf,L),0.);
}
```

OpenGL Vertex Shader (II)

- Accumulate light contribution

```
gl_FrontColor = vec4(0,0,0,1);
for (int i=0; i < gl_MaxLights; i++) {
    vec3 L = normalize(
        gl_LightSource[i].position.xyz*P.w -
        P.xyz*gl_LightSource[i].position.w);
    gl_FrontColor.xyz +=
        gl_LightSource[i].ambient +
        gl_LightSource[i].diffuse*
        max(dot(Nf,L),0.);
}
```

OpenGL Vertex Shader (II)

- Accumulate light contribution

```
gl_FrontColor = vec4(0,0,0,1);
for (int i=0; i < gl_MaxLights; i++) {
    vec3 L = normalize(
        gl_LightSource[i].position.xyz*P.w -
        P.xyz*gl_LightSource[i].position.w);
    gl_FrontColor.xyz +=
        gl_LightSource[i].ambient +
        gl_LightSource[i].diffuse*
        max(dot(Nf,L),0.);
}
```

OpenGL Vertex Shader (II)

- Accumulate light contribution

```
gl_FrontColor = vec4(0,0,0,1);
for (int i=0; i < gl_MaxLights; i++) {
    vec3 L = normalize(
        gl_LightSource[i].position.xyz*P.w -
        P.xyz*gl_LightSource[i].position.w);
    gl_FrontColor.xyz +=
        gl_LightSource[i].ambient +
        gl_LightSource[i].diffuse*
        max(dot(Nf,L),0.);
}
```

OpenGL Vertex Shader (II)

- Accumulate light contribution

```
gl_FrontColor = vec4(0,0,0,1);
for (int i=0; i < gl_MaxLights; i++) {
    vec3 L = normalize(
        gl_LightSource[i].position.xyz*P.w -
        P.xyz*gl_LightSource[i].position.w);
    gl_FrontColor.xyz +=
        gl_LightSource[i].ambient +
        gl_LightSource[i].diffuse*
        max(dot(Nf,L),0.);
}
```

OpenGL Vertex Shader (II)

- Accumulate light contribution

```
gl_FrontColor = vec4(0,0,0,1);
for (int i=0; i < gl_MaxLights; i++) {
    vec3 L = normalize(
        gl_LightSource[i].position.xyz*P.w -
        P.xyz*gl_LightSource[i].position.w);
    gl_FrontColor.xyz +=
        gl_LightSource[i].ambient +
        gl_LightSource[i].diffuse*
        max(dot(Nf,L),0.);
}
```

OpenGL Fragment Shader

- Compute per-fragment brick color

```
uniform float width, height, gap;
uniform vec3 brick, mortar;

void main(void) {
    vec3 bc;
    float s = gl_TexCoord[0].s;
    float t = gl_TexCoord[0].t;
    /* [compute brick color] */
    gl_FragColor = bc * gl_Color;
}
```

OpenGL Fragment Shader

- Compute per-fragment brick color

```
uniform float width, height, gap;
uniform vec3 brick, mortar;

void main(void) {
    vec3 bc;
    float s = gl_TexCoord[0].s;
    float t = gl_TexCoord[0].t;
    /* [compute brick color] */
    gl_FragColor = bc * gl_Color;
}
```

OpenGL Fragment Shader

- Compute per-fragment brick color

```
uniform float width, height, gap;
uniform vec3 brick, mortar;

void main(void) {
    vec3 bc;
    float s = gl_TexCoord[0].s;
    float t = gl_TexCoord[0].t;
    /* [compute brick color] */
    gl_FragColor = bc * gl_Color;
}
```

OpenGL Fragment Shader

- Compute per-fragment brick color

```
uniform float width, height, gap;
uniform vec3 brick, mortar;

void main(void) {
    vec3 bc;
    float s = gl_TexCoord[0].s;
    float t = gl_TexCoord[0].t;
    /* [compute brick color] */
    gl_FragColor = bc * gl_Color;
}
```

OpenGL Fragment Shader

- Compute per-fragment brick color

```
uniform float width, height, gap;
uniform vec3 brick, mortar;

void main(void) {
    vec3 bc;
    float s = gl_TexCoord[0].s;
    float t = gl_TexCoord[0].t;
    /* [compute brick color] */
    gl_FragColor = bc * gl_Color;
}
```

OpenGL Fragment Shader

- Compute per-fragment brick color

```
uniform float width, height, gap;
uniform vec3 brick, mortar;

void main(void) {
    vec3 bc;
    float s = gl_TexCoord[0].s;
    float t = gl_TexCoord[0].t;
    /* [compute brick color] */
    gl_FragColor = bc * gl_Color;
}
```

OpenGL Fragment Shader

- Compute per-fragment brick color

```
uniform float width, height, gap;
uniform vec3 brick, mortar;

void main(void) {
    vec3 bc;
    float s = gl_TexCoord[0].s;
    float t = gl_TexCoord[0].t;
    /* [compute brick color] */
    gl_FragColor = bc * gl_Color;
}
```

OpenGL Fragment Shader (II)

- Compute brick color
- Where am I in my brick (Brick Coordinates)

```
float bs, bt;
/* [compute brick coordinates] */
if (bs < width && bt < height)
    bc = brick;
else
    bc = mortar;
```

OpenGL Fragment Shader (II)

- Compute brick color
- Where am I in my brick (Brick Coordinates)


```
float bs, bt;
/* [compute brick coordinates] */
if (bs < width && bt < height)
    bc = brick;
else
    bc = mortar;
```

OpenGL Fragment Shader (II)

- Compute brick color
- Where am I in my brick (Brick Coordinates)


```
float bs, bt;
/* [compute brick coordinates] */
if (bs < width && bt < height)
    bc = brick;
else
    bc = mortar;
```

OpenGL Fragment Shader (II)

- Compute brick color
- Where am I in my brick (Brick Coordinates)


```
float bs, bt;
/* [compute brick coordinates] */
if (bs < width && bt < height)
    bc = brick;
else
    bc = mortar;
```

OpenGL Fragment Shader (II)

- Compute brick color
- Where am I in my brick (Brick Coordinates)


```
float bs, bt;
/* [compute brick coordinates] */
if (bs < width && bt < height)
    bc = brick;
else
    bc = mortar;
```

OpenGL Fragment Shader (III)

- Compute brick coordinates

```
bt = mod(t, height+gap);
bs = s;
if (mod((t-bt)/(height+gap), 2.) < 1.5)
    bs += (width+gap)/2.;
bs = mod(bs, width+gap);
```

OpenGL Fragment Shader (III)

- Compute brick coordinates

```
bt = mod(t, height+gap);
bs = s;
if (mod((t-bt)/(height+gap), 2.) < 1.5)
    bs += (width+gap)/2.;
bs = mod(bs, width+gap);
```

OpenGL Fragment Shader (III)

- Compute brick coordinates

```
bt = mod(t, height+gap);
bs = s;
if (mod((t-bt)/(height+gap), 2.) < 1.5)
    bs += (width+gap)/2.;
bs = mod(bs, width+gap);
```

OpenGL Fragment Shader (III)

- Compute brick coordinates

```
bt = mod(t, height+gap);
bs = s;
if (mod((t-bt)/(height+gap), 2.) < 1.5)
    bs += (width+gap)/2.;
bs = mod(bs, width+gap);
```

OpenGL Fragment Shader (III)

- Compute brick coordinates

```
bt = mod(t, height+gap);
bs = s;
if (mod((t-bt)/(height+gap), 2.) < 1.5)
      bs += (width+gap)/2.;
bs = mod(bs, width+gap);
```

3D Noise

- Recall

$$hash(x) = x^2 \bmod M$$

$$\begin{aligned} noise(\vec{p}) = flerp(z, &zconst(\vec{p}^x, \vec{p}^y + hash(Z_0)) \\ &+ zgrad(\vec{p}^x, \vec{p}^y\, hash(Z_0)) * z, \\ &zconst(\vec{p}^x, \vec{p}^y\, hash(Z_1)) \\ &+ zgrad(\vec{p}^x, \vec{p}^y + hash(Z_1)) * (z-1)) \end{aligned}$$

3D Noise

- Recall

$$hash(x) = x^2 \bmod M$$

$$\begin{aligned} noise(\vec{p}) = flerp(z, & zconst(\vec{p}^x, \vec{p}^y + hash(Z_0)) \\ & + zgrad(\vec{p}^x, \vec{p}^y\, hash(Z_0)) * z, \\ & zconst(\vec{p}^x, \vec{p}^y\, hash(Z_1)) \\ & + zgrad(\vec{p}^x, \vec{p}^y + hash(Z_1)) * (z-1)) \end{aligned}$$

3D Noise

- Recall

$$hash(x) = x^2 \bmod M$$

$$\begin{aligned} noise(\vec{p}) = flerp(z, &zconst(\vec{p}^x, \vec{p}^y + hash(Z_0)) \\ &+ zgrad(\vec{p}^x, \vec{p}^y\, hash(Z_0)) * z, \\ &zconst(\vec{p}^x, \vec{p}^y\, hash(Z_1)) \\ &+ zgrad(\vec{p}^x, \vec{p}^y + hash(Z_1)) * (z-1)) \end{aligned}$$

3D Noise Fragment Shader (I)

```
varying vec3 Nin;
const float modulus = 61;
uniform sampler2D ntex;

void main() {
    /* noise computation */
}
```

3D Noise Fragment Shader (I)

```
varying vec3 Nin;
const float modulus = 61;
uniform sampler2D ntex;

void main() {
    /* noise computation */
}
```

3D Noise Fragment Shader (I)

```
varying vec3 Nin;
const float modulus = 61;
uniform sampler2D ntex;

void main() {
    /* noise computation */
}
```

3D Noise Fragment Shader (I)

```
varying vec3 Nin;
const float modulus = 61;
uniform sampler2D ntex;

void main() {
    /* noise computation */
}
```

3D Noise Fragment Shader (II)

```
float fracArg = fract(modulus*Nin.z);
float intArg = floor(modulus*Nin.z);

vec2 hash = mod(intArg,modulus);
hash.y = hash.y+1;
hash = mod(hash*hash,modulus);
hash = hash/modulus;
```

3D Noise Fragment Shader (II)

```
float fracArg = fract(modulus*Nin.z);
float intArg = floor(modulus*Nin.z);

vec2 hash = mod(intArg,modulus);
hash.y = hash.y+1;
hash = mod(hash*hash,modulus);
hash = hash/modulus;
```

3D Noise Fragment Shader (III)

```
vec2 g0, g1;
g0 = texture2D(ntex, Nin.xy+vec2(0,hash.x)).xy*2-1;
g1 = texture2D(ntex, Nin.xy+vec2(0,hash.y)).xy*2-1;
float noise = mix(g0.r+g0.a*fracArg,
                  g1.r+g1.a*(fracArg-1),
                  smoothstep(0,1,fracArg));
```

3D Noise Fragment Shader (III)

```
vec2 g0, g1;
g0 = texture2D(ntex, Nin.xy+vec2(0,hash.x)).xy*2-1;
g1 = texture2D(ntex, Nin.xy+vec2(0,hash.y)).xy*2-1;
float noise = mix(g0.r+g0.a*fracArg,
                  g1.r+g1.a*(fracArg-1),
                  smoothstep(0,1,fracArg));
```

Outline

Resources


Kessenich, J., Baldwin, D., and Rost, R. (2004).
The OpenGL Shading Language.
3Dlabs, Inc. Ltd., version 1.10 edition.


Rost, R. J. (2004).
OpenGL(R) Shading Language.
Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.