Introduction to shaders using GLSL ES and WebCGF

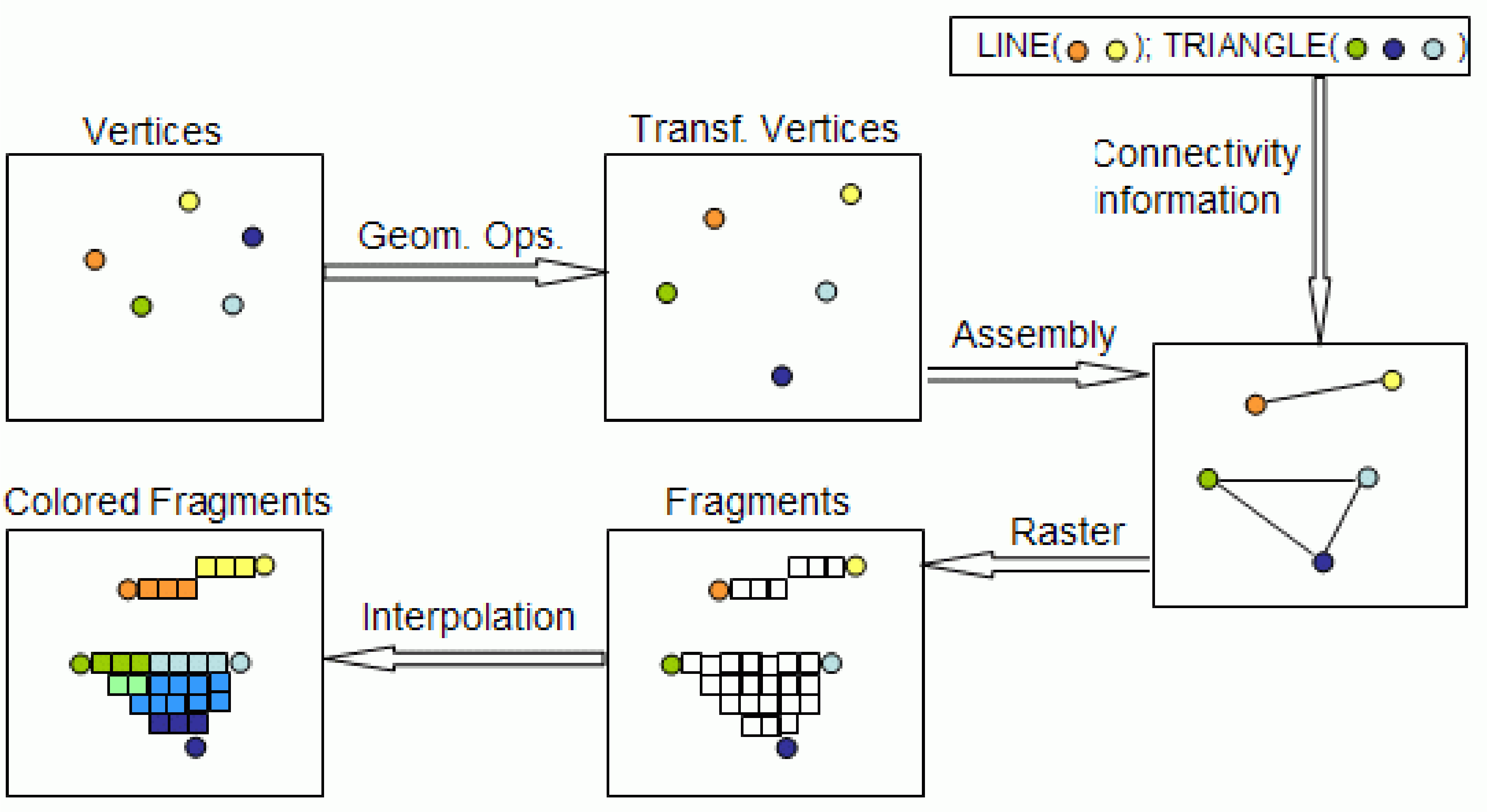
Rui Rodrigues
rui.rodrigues@fe.up.pt

v1.1, 11/2018

Outline

- Graphics pipeline
- Shader types
- Common shading languages
- GLSL details
    - Data types
    - Special variable declarations
    - Swizzling
- Passing values
    - From App to Shaders
    - From Vertex Shader to Fragment Shader
- Working with textures

Graphics pipeline: visual representation



OpenGL pipeline visual representation [GLSL12Tut11]

Graphics pipeline: simplified block diagram

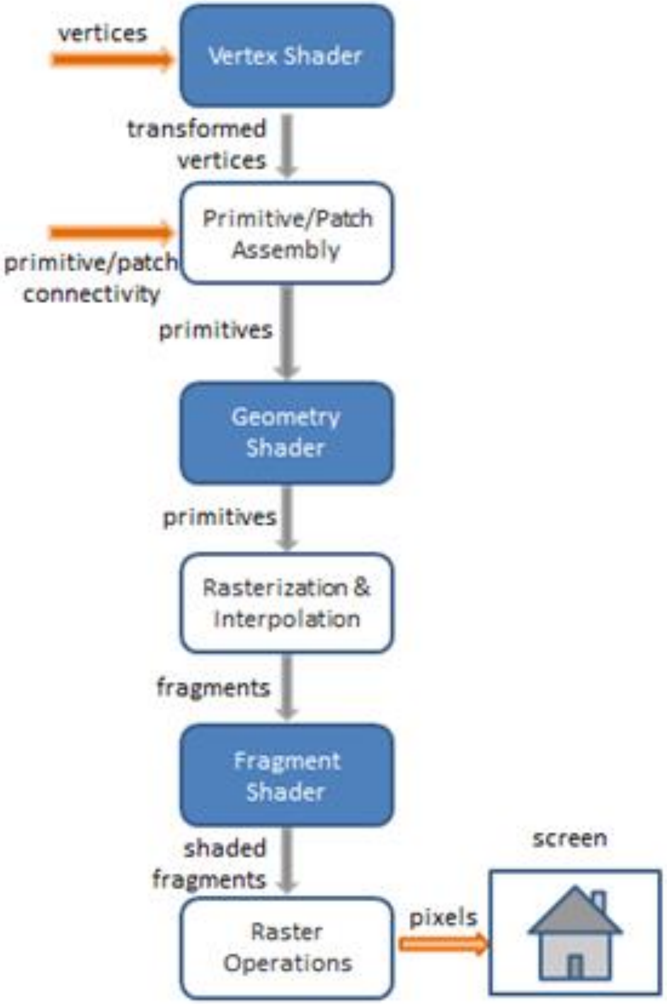
- Inputs

  (vertices, triangles, textures, matrices, etc.)

- **Vertex shading**
- Primitive assembly, culling and clipping
- **Geometry shading** (optional)
- Projection and rasterization
- **Fragment shading**

  (may output to multiple render targets)

- Depth, Stencyl and Alpha-blend (raster) operations
- Output to screen



OpenGL simplified pipeline
(Adapted from [GLSLTut11])

Shaders

- Small programs that replace the fixed functionality of some stages
  - Vertex shaders (VS)
    - Manipulate and define per-vertex properties (coordinates, color, normals)
  - Geometry shaders (GS) (less used)
    - Manipulate and define per-primitive properties (connectivity)
    - May generate new primitives
  - Fragment shaders (FS)
    - Manipulate and define per-fragment (pixel or sample) properties - typically color and transparency
  - Other (e.g. tesselation shaders)

Common shading languages

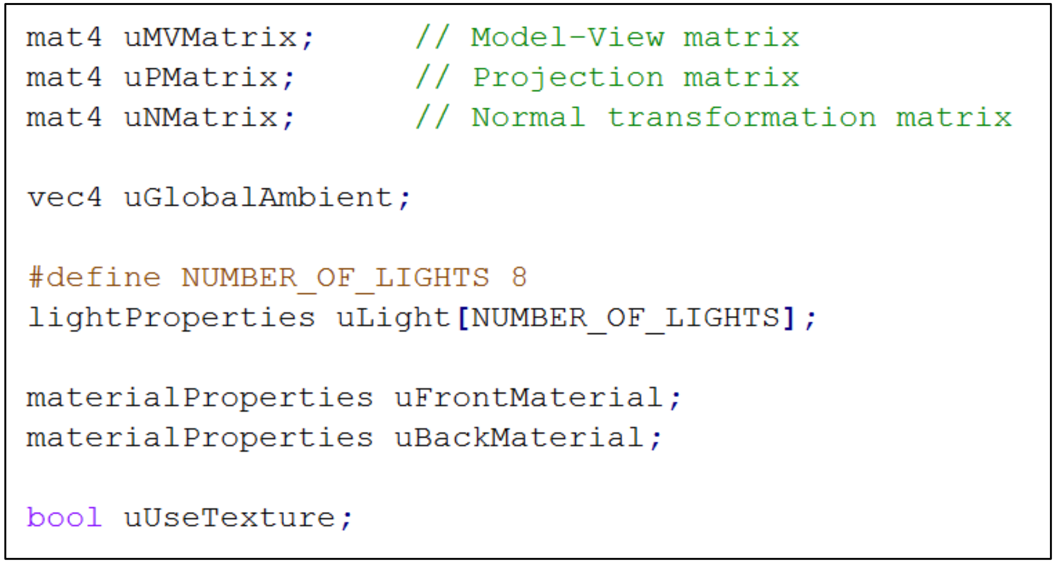
- OpenGL’s GLSL
  - And GLSL ES for mobile/web – our focus
- Microsoft’s HLSL
- Nvidia’s CG
- Other (earlier)
  - RenderMan
  - OpenGL ISL

GLSL

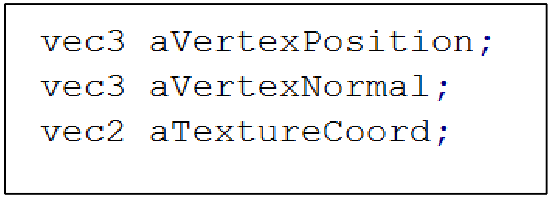
- C-like language
- Shaders can be loaded as text strings and are compiled in runtime
    - Meaning they can also be changed in runtime
- Values/variables can be passed from application to shaders
- Values can be output from the vertex shader and interpolated to the fragment shader
    - (e.g. Vertex's color interpolated over fragment)

Usage in WebCGF

- The default vertex shader receives all the necessary variables for implementing the local illumination model (lights, materials, projection and transformation matrices, etc.)
- Also, for each vertex, it receives its position, normal and texture coordinates

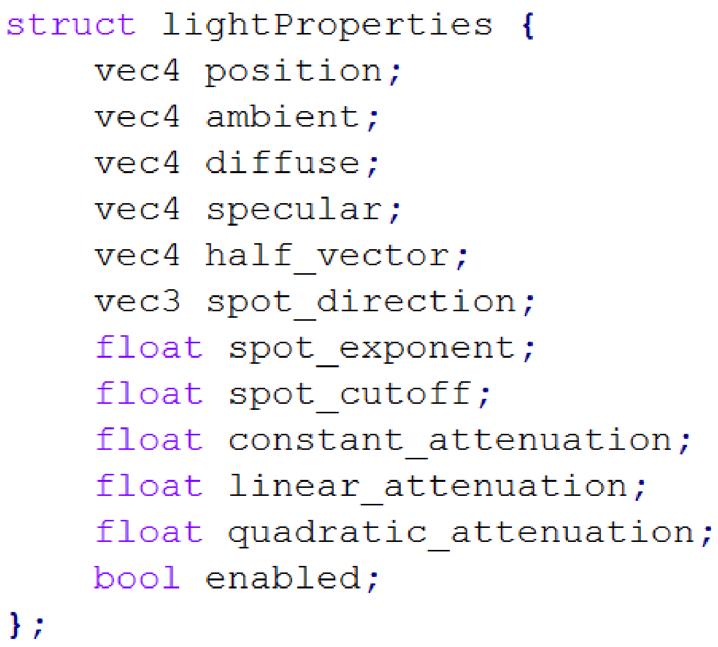


```
mat4 uMVMatrix;     // Model-View matrix
mat4 uPMatrix;      // Projection matrix
mat4 uNMatrix;      // Normal transformation matrix

vec4 uGlobalAmbient;

#define NUMBER_OF_LIGHTS 8
lightProperties uLight[NUMBER_OF_LIGHTS];

materialProperties uFrontMaterial;
materialProperties uBackMaterial;

bool uUseTexture;
```

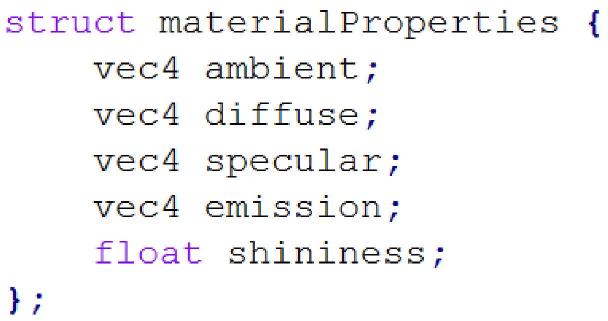


```
vec3 aVertexPosition;
vec3 aVertexNormal;
vec2 aTextureCoord;
```

Light and material properties



```
struct lightProperties {
    vec4 position;
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    vec4 half_vector;
    vec3 spot_direction;
    float spot_exponent;
    float spot_cutoff;
    float constant_attenuation;
    float linear_attenuation;
    float quadratic_attenuation;
    bool enabled;
};
```



```
struct materialProperties {
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    vec4 emission;
    float shininess;
};
```

First example (1/4): vertex shader

(Vertex shaders will be surrounded by dotted lines)

```
void main()
{
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
}
```

- Basic implementation of vertex transformation
- Applied to each vertex (while shader active)
- Outputs vertex’s position in eye space by multiplying...
    - vertex coordinates (e.g. from an object’s vertex buffer) in homogeneous form
    - scene’s model-view matrix (affected by transformations)
    - projection matrix

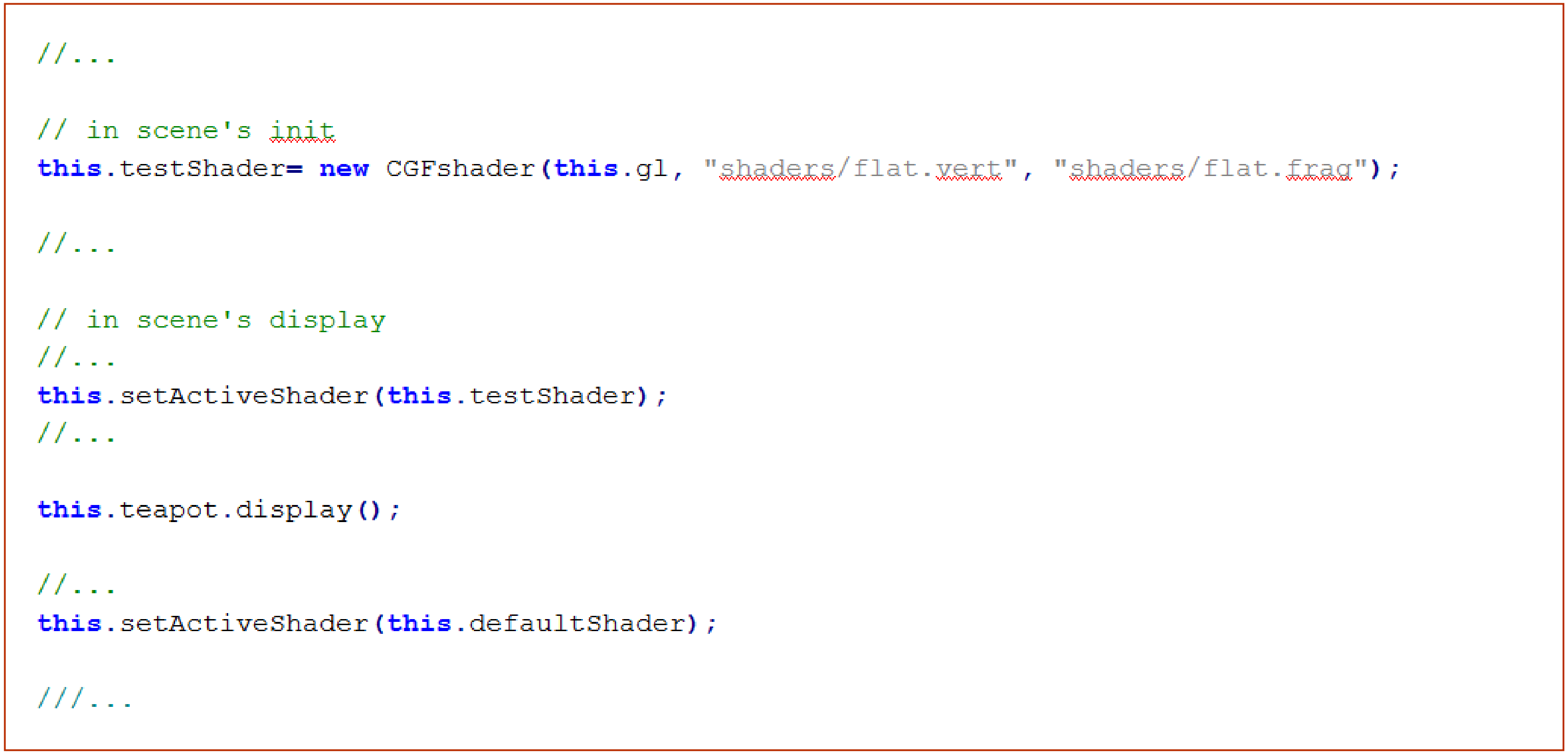
First example (2/4): fragment shader (FS)

(Fragment shaders will be surrounded by dashed lines)

```
void main()
{
        gl_FragColor = vec4(0.0,0.0,0.5, 1.0) * uLight[0].diffuse;
}
```

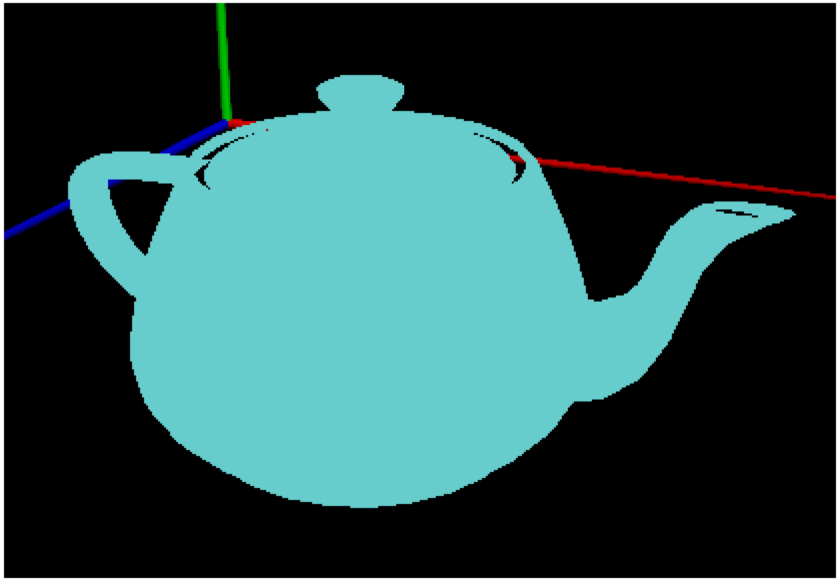
- A simple shader that sets the current fragment’s color based on the diffuse component of a light source

First example (3/4):
in the main code (scene)



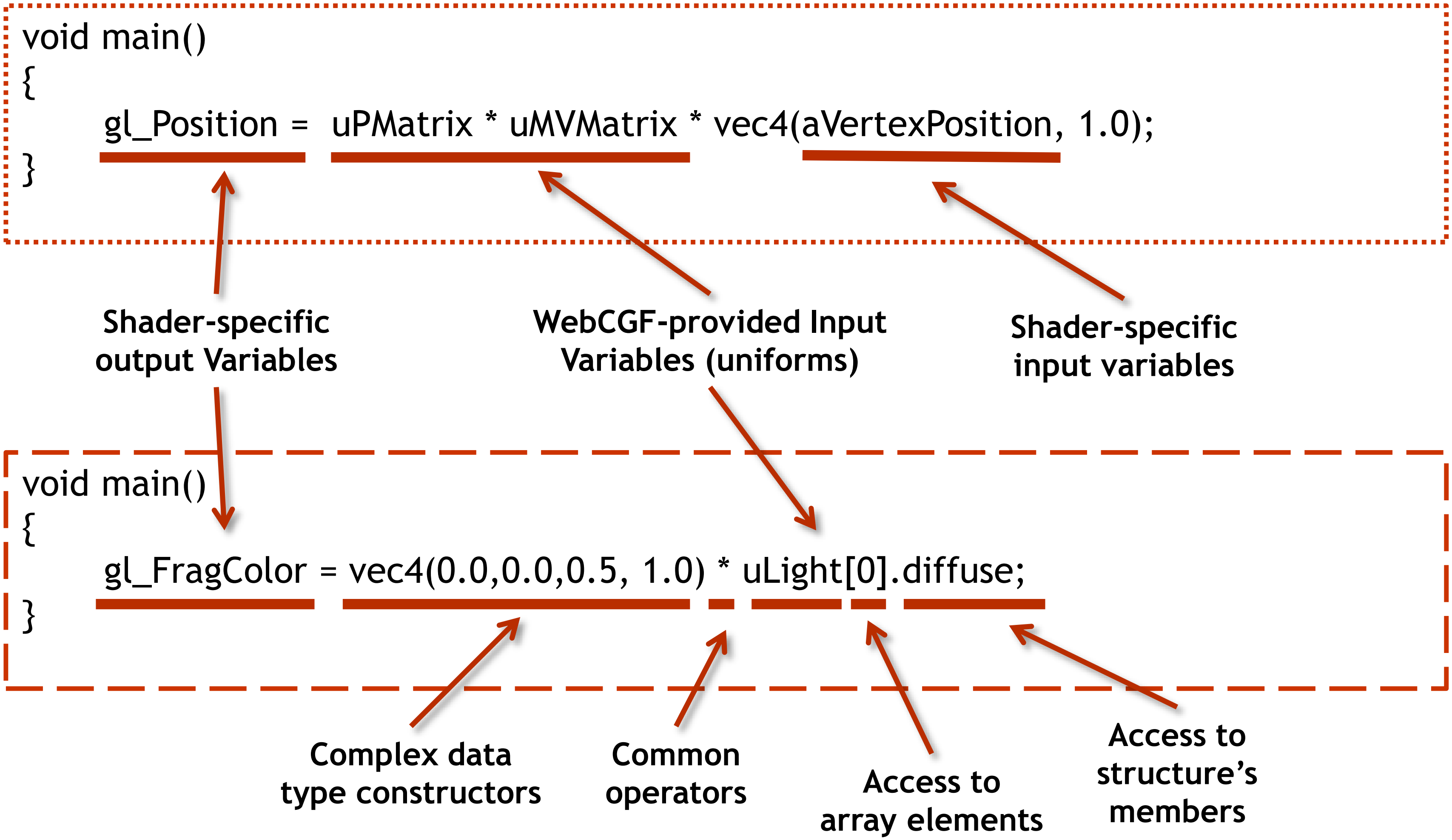
```
//...

// in scene's init
this.testShader= new CGFshader(this.gl, "shaders/flat.vert", "shaders/flat.frag");

//...

// in scene's display
//...
this.setActiveShader(this.testShader);
//...

this.teapot.display();

//...
this.setActiveShader(this.defaultShader);

///...
```

First example (4/4): sample output



- Notice that this gives a solid colored surface, as we set every fragment to the same color
- IMPORTANT: When shaders are active, the usual lighting and shading are disabled.

Some elements to notice



```
void main()
{
      gl_Position =  uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
}
```

Shader-specific output Variables

WebCGF-provided Input Variables (uniforms)

Shader-specific input variables

```
void main()
{
      gl_FragColor = vec4(0.0,0.0,0.5, 1.0) * uLight[0].diffuse;
}
```

Complex data type constructors

Common operators

Access to array elements

Access to structure’s members

What can be used in shaders?

- WebCGF-provided information and data structures such as
  - vertex, normal and color information
  - transformation matrices,
  - light sources and parameters,
  - material parameters, etc.
- Parameters in any of the supported data types
  - passed from the application to the shaders, and between shaders
- A series of built-in functions, including
  - trigonometry and other geometry-related functions,
  - matrix and vector calculus,
  - texture sampling and noise generation
- Multiple textures
  - can be used not only for color modulation, but also for passing information structured as arrays
- User-defined functions and structures, arrays

Data types

- float, vec2, vec3, vec4
  - Individual float values, and vectors of 2, 3 or 4 float components
- int, ivec2, ivec3, ivec4
  - Individual integer values, and vectors of 2, 3 or 4 integer components
- bool, bvec2, bvec3, bvec4
  - Individual boolean values, and vectors of 2, 3 or 4 boolean components
- mat2, mat3, mat4
  - Square matrices of dimensions 2x2, 3x3, or 4x4
- void
  - Used for functions with no return value
- sampler1D, sampler2D, sampler3D
  - Used to sample points on a texture map of 1, 2 or 3 dimensions
- Other samplers

Swizzling

- Accessing one or more vector components in any order

```
myColor.rgb = vec3(1.0,0.0,0.0);
myPos.xz = vec2(10.0,5.0);
myTexCoord.st = myPos.zx;
myVec4 = vec4(myPos.xyz,1.0);
```

- Three possible sets (cannot be mixed)

  xyzw (for coordinates)

  rgba (for colors)

  stpq (for texture coordinates)

Global variable declarations

- uniform
  - input to Vertex and Fragment shader from application (RO)
- attribute
  - input per-vertex to Vertex shader from application (RO)
- varying
  - output from Vertex shader (RW), and interpolated to serve as per-fragment input to Fragment shader (RO)
- const
  - compile-time constant (READ-ONLY)

Function parameter declaration

- In (default)
    - value initialized on entry, not copied on return
- out
    - copied out on return, but not initialized
- inout
    - value initialized on entry, and copied out on return
- const
    - constant function input

Vertex shader input attributes (RO)

- Coming from WebCGF
  - vec3 aVertexPosition
  - vec3 aVertexNormal
  - vec2 aTextureCoord
  - ...

Vertex shader output variables

- Special (RW)
  - vec4 gl_Position
    - must be written by VS, it is the vertex position in eye space
  - Other

Fragment shader inputs

- Varying Inputs (RO)
  - vec4 gl_FragColor
  - vec4 gl_FragCoord
  - vec2 gl_PointCoord
  - bool gl_FrontFacing
  - …

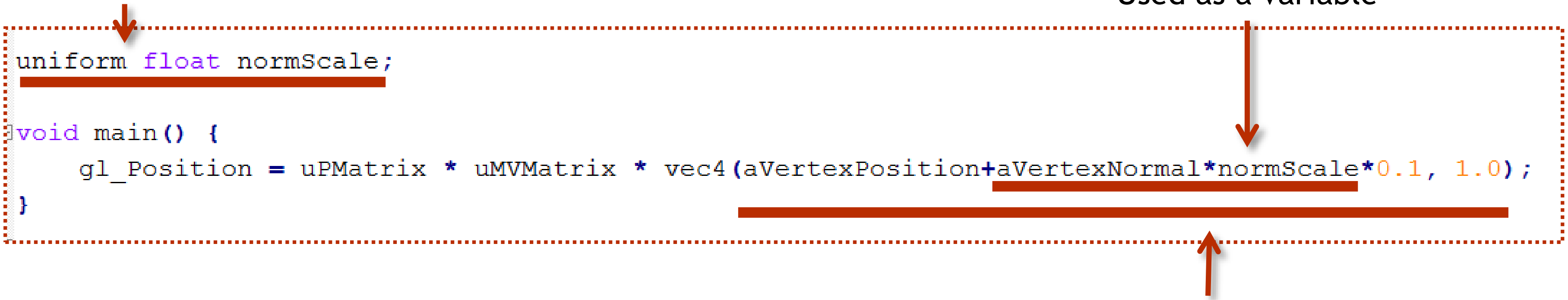
Fragment shader output variables

- Special (RW)
  - vec4 gl_FragColor;
  - vec4 gl_FragData[];
  - float gl_FragDepth;

Passing values: from app to shaders (1/3)

Uniform declaration

Used as a variable

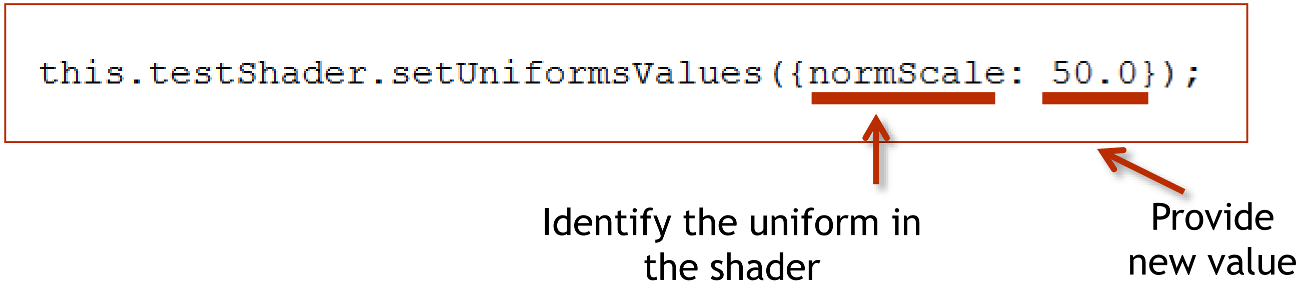


```
uniform float normScale;

void main() {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition+aVertexNormal*normScale*0.1, 1.0);
}
```

Notice building a vec4
using a vec3 plus a
fourth component

- This shader displaces a vertex by adding a vector that has the direction of the vertex’s normal, and a scale controlled by a parameter, *normScale*

Passing values: from app to shaders (2/3)



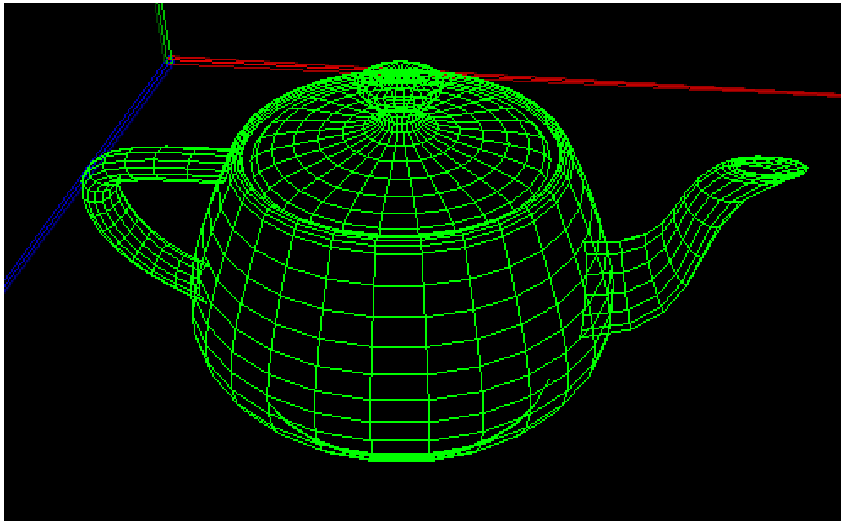
```
this.testShader.setUniformsValues({normScale: 50.0});
```

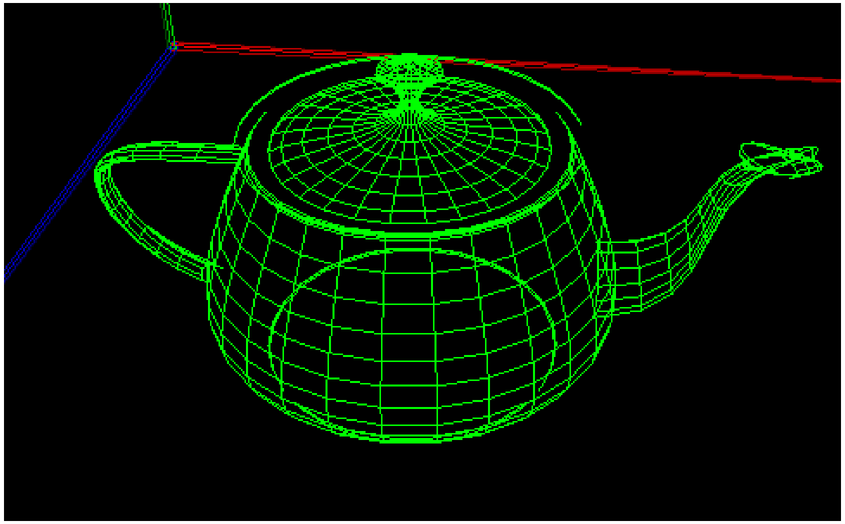
Identify the uniform in the shader

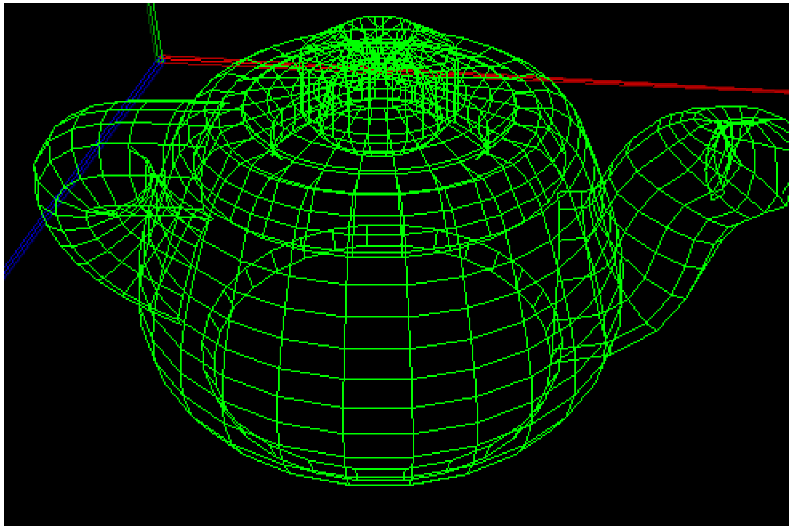
Provide new value

- The parameter value can be controlled in the application

Passing values: from app to shaders (3/3)

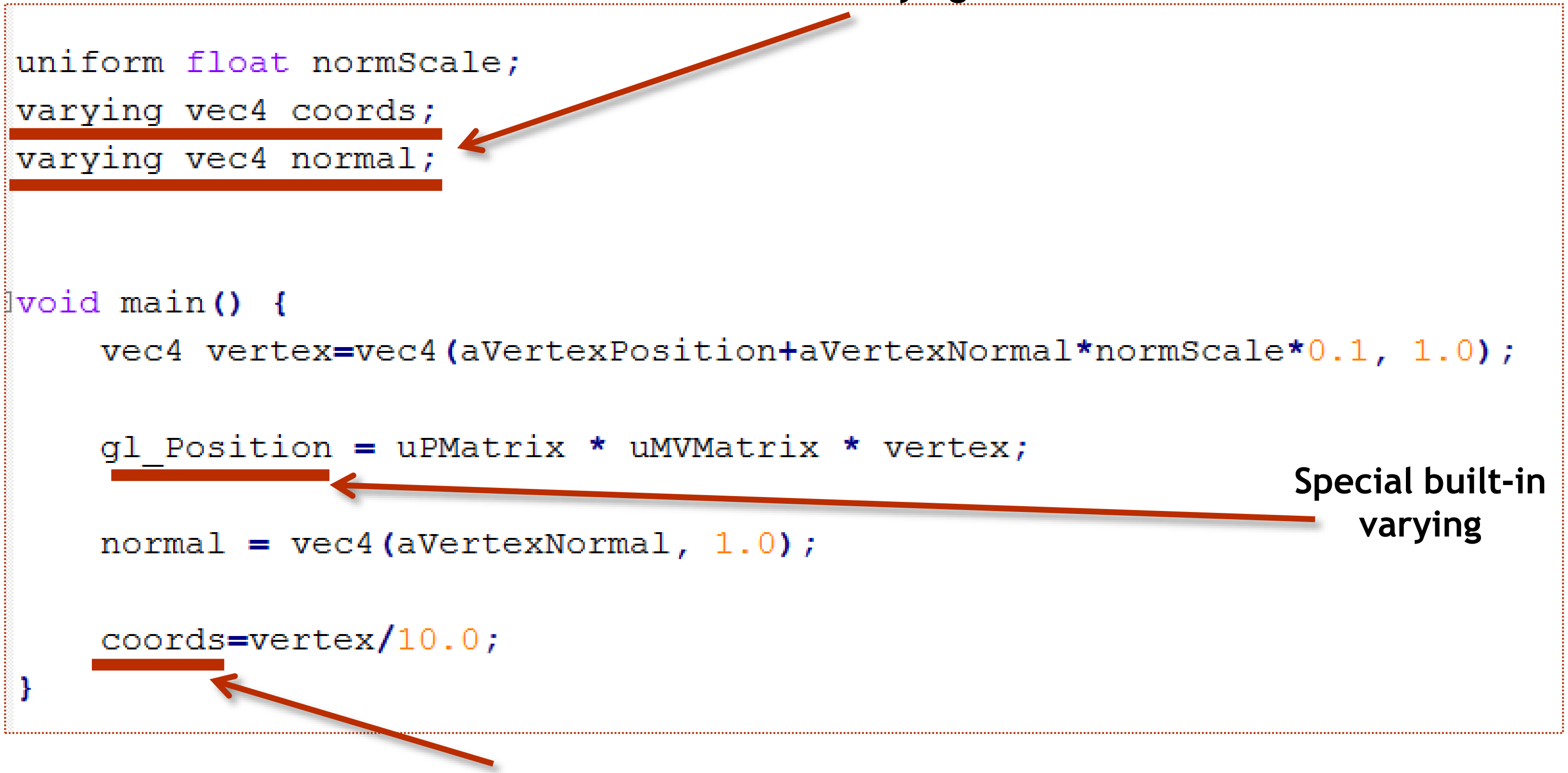






Passing values: from VS to FS (1/3)

Declaration of user-defined varying's

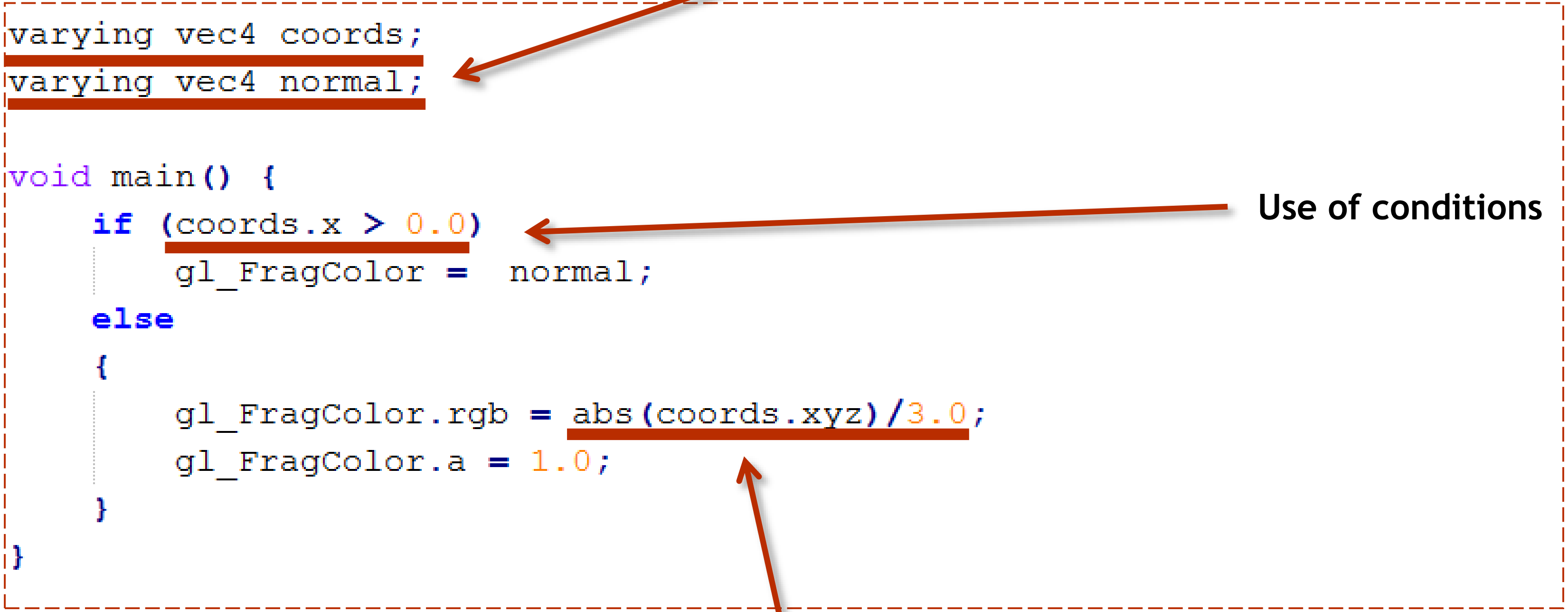


```
uniform float normScale;
varying vec4 coords;
varying vec4 normal;


void main() {
    vec4 vertex=vec4(aVertexPosition+aVertexNormal*normScale*0.1, 1.0);

    gl_Position = uPMatrix * uMVMatrix * vertex;

    normal = vec4(aVertexNormal, 1.0);

    coords=vertex/10.0;
}
```

Special built-in varying

Usage of user-defined varying

Passing values: from VS to FS (2/3)



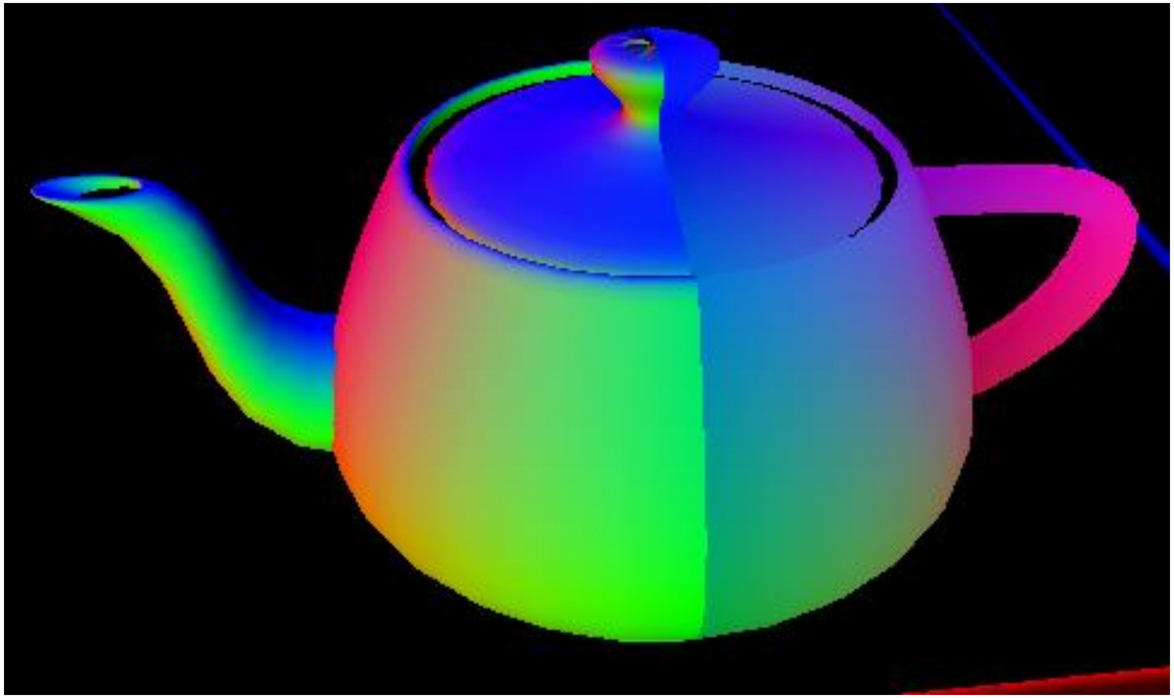
```
varying vec4 coords;
varying vec4 normal;

void main() {
    if (coords.x > 0.0)
        gl_FragColor =  normal;
    else
    {
        gl_FragColor.rgb = abs(coords.xyz)/3.0;
        gl_FragColor.a = 1.0;
    }
}
```

Declaration of user-defined varying

Use of conditions

Built-in functions and swizzling

Passing values: from VS to FS (3/3)

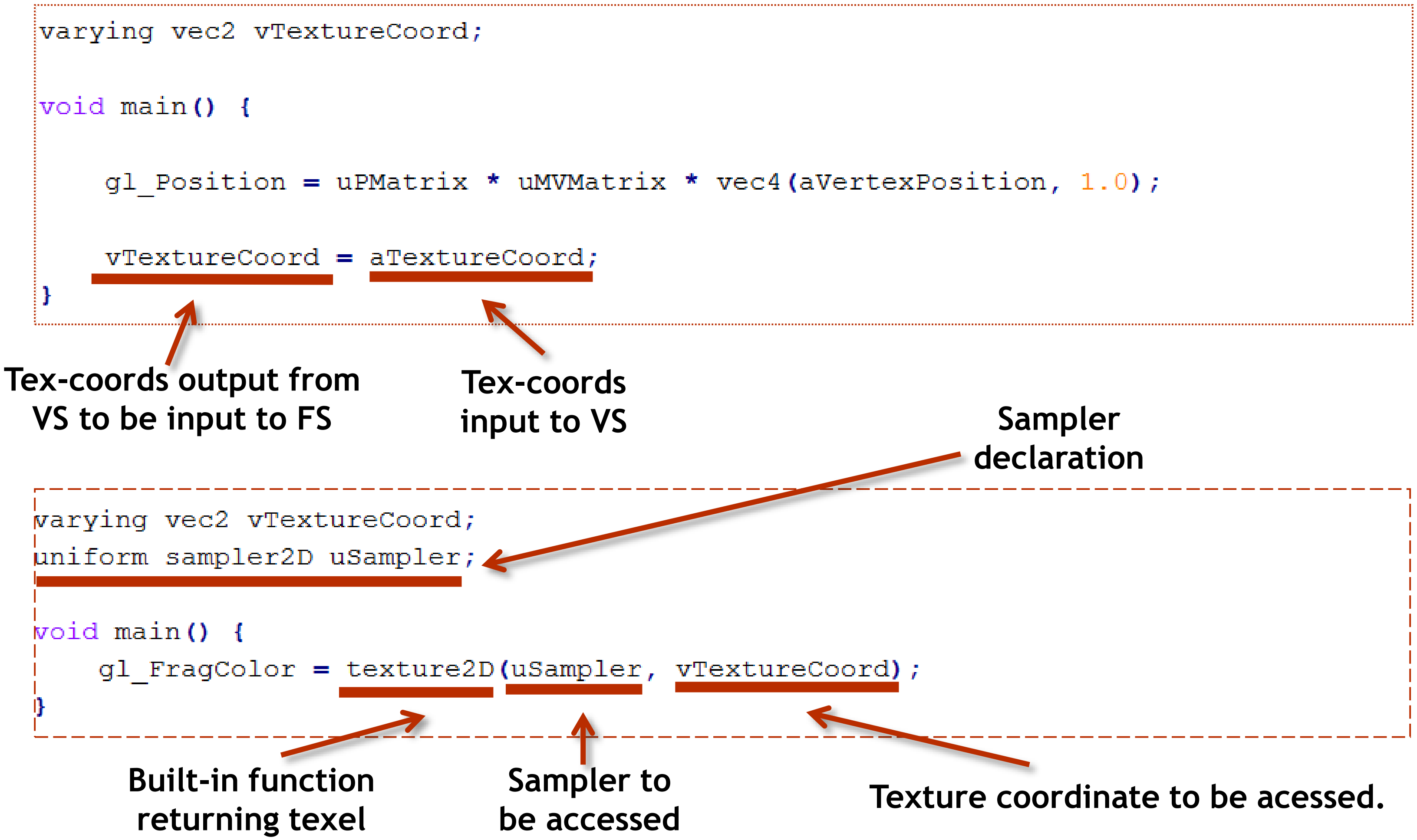


- The left half has color varying depending on the surface orientation (as it is based on the normals)
- The right half has color varying depending on their vertical and horizontal position

Working with textures (1/7)

- Textures are referenced in shaders as uniforms of type *int*, in which the uniform's value defines the texture unit to be used
  - A uniform sampler2D assigned with the value 0 gets linked to GL_TEXTURE0
  - WebCGF does this by default setting the value of a uniform called *uSampler* to 0
- For using a single texture, you only need to bind a texture as usual, and use *uSampler* in the shader code.

Working with textures (2/7)



```
varying vec2 vTextureCoord;

void main() {

    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);

    vTextureCoord = aTextureCoord;
}
```

Tex-coords output from VS to be input to FS

Tex-coords input to VS

Sampler declaration

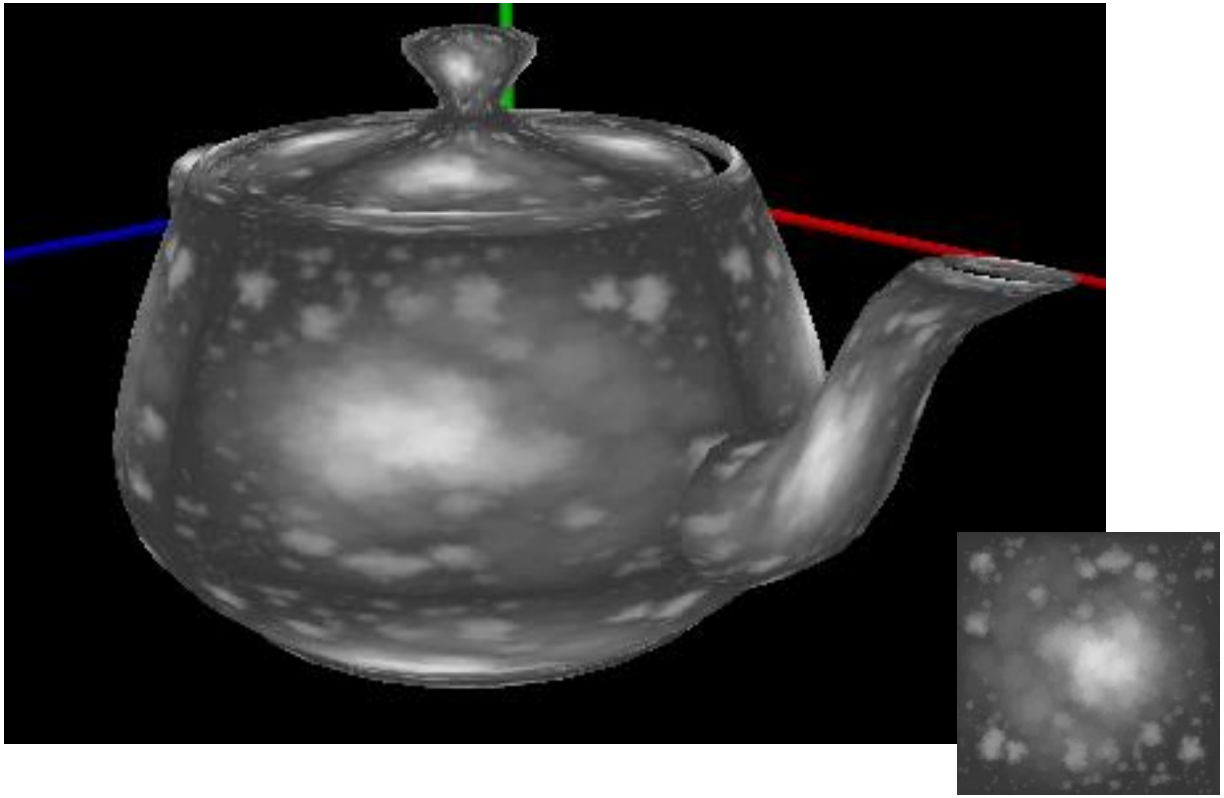
```
varying vec2 vTextureCoord;
uniform sampler2D uSampler;

void main() {
    gl_FragColor = texture2D(uSampler, vTextureCoord);
}
```

Built-in function returning texel

Sampler to be accessed

Texture coordinate to be acessed.

Working with textures (3/7)



Working with textures (4/7)

- Steps to work with a texture
  - Create uniform of type “sampler” in the shader(s)
  - In the app, set the uniform value to a texture unit number
  - bind a texture to the corresponding texture unit
- Do this for the number of textures needed by your shader
  - Remember *uSampler* is already provided and assigned to texture unit 0

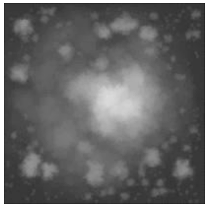
Working with textures (5/7)

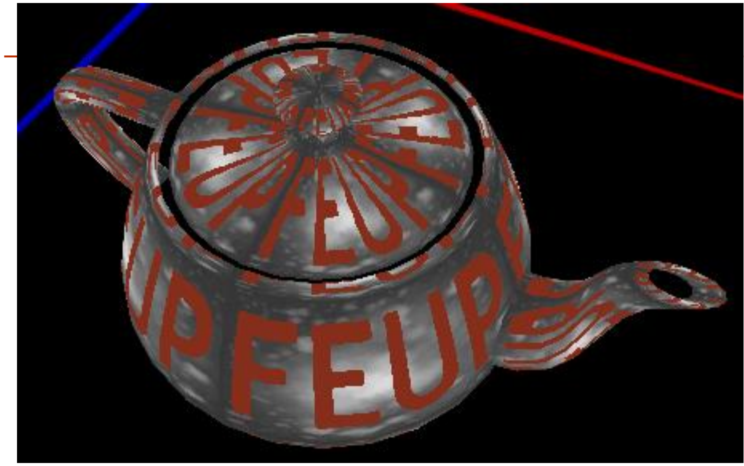
```
varying vec2 vTextureCoord;

uniform sampler2D uSampler;
uniform sampler2D uSampler2;

void main() {
    vec4 color = texture2D(uSampler, vTextureCoord);
    vec4 filter = texture2D(uSampler2, vec2(0.0,0.1)+vTextureCoord);

    if (filter.b > 0.5)
        color=vec4(0.52, 0.18, 0.11, 1.0);

    gl_FragColor = color;
}
```

Another sampler declaration (order not important here)

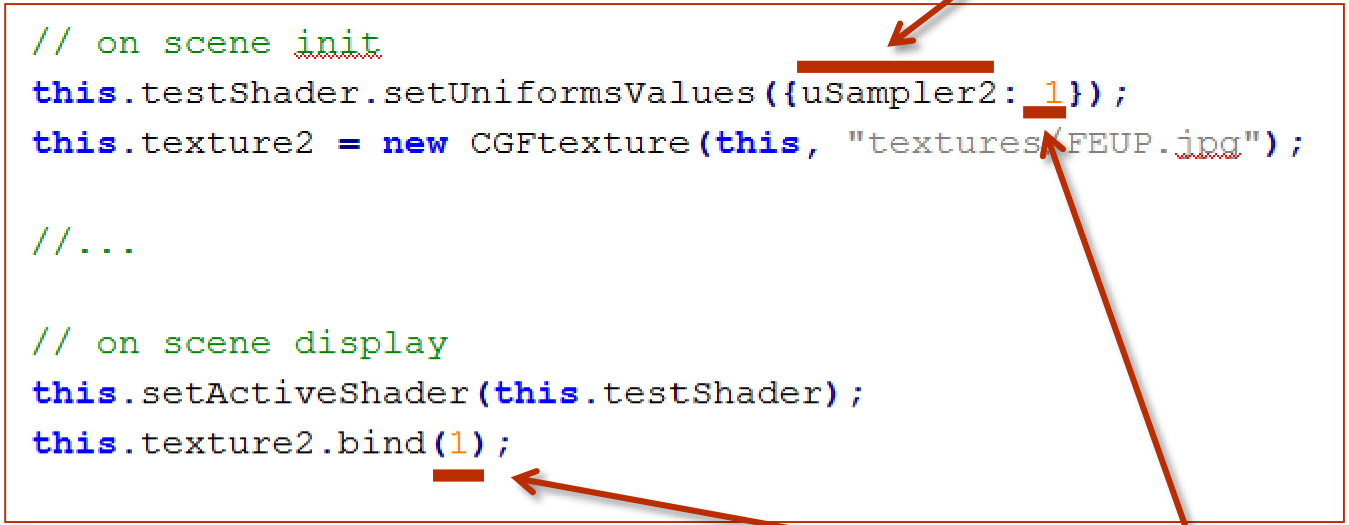
Texture coordinate to be acessed. Notice coordinates can be manipulated

Texture information being used as a filter





Working with textures (6/7)

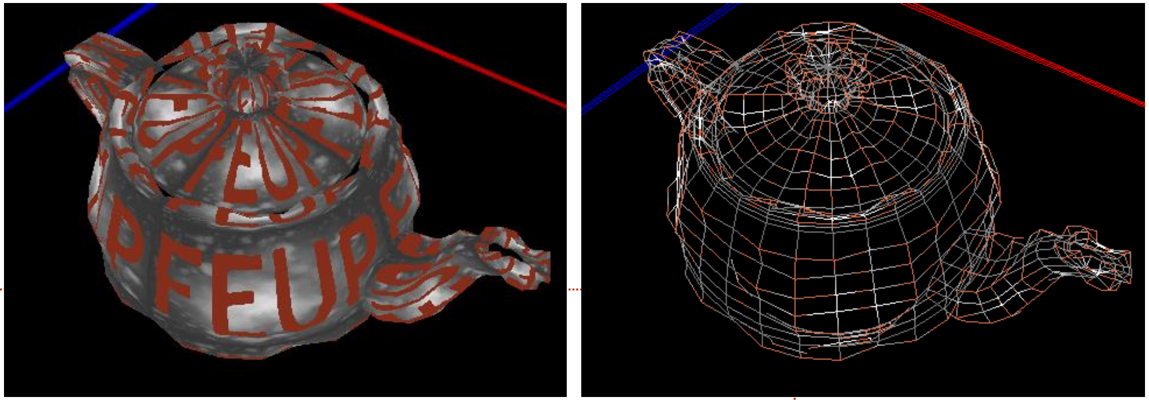


```
// on scene init
this.testShader.setUniformsValues({uSampler2: 1});
this.texture2 = new CGFtexture(this, "textures/FEUP.jpg");

//...

// on scene display
this.setActiveShader(this.testShader);
this.texture2.bind(1);
```

Sampler name
Used on shaders

Reference to
Texture unit

Working with textures (7/7)



Samplers can also be used in vertex shader

```
varying vec2 vTextureCoord;
uniform sampler2D uSampler2;

uniform float normScale;

void main() {
    vec3 offset=vec3(0.0,0.0,0.0);

    // pass texture coordinates from VS to FS
    vTextureCoord = aTextureCoord;

    // change vertex offset based on texture information
    if (texture2D(uSampler2, vec2(0.0,0.1)+vTextureCoord).b > 0.5)
        offset=aVertexNormal*normScale*0.1;

    // set the position of the current vertex
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition+offset, 1.0);
}
```

Sampler being used as a filter to change geometry

References

[GLSL12Tut11] GLSL 1.2 Tutorial, António Ramires Fernandes, http://www.lighthouse3d.com/tutorials/glsl-tutorial/ , Lighthouse3D tutorials (accessed October 2012)

[GLSLCTut11] GLSL Core Tutorial, António Ramires Fernandes, http://www.lighthouse3d.com/tutorials/glsl-core-tutorial/, Lighthouse3D tutorials (accessed October 2012)

[GLSLRC05] GLSL Reference Card, Michael E. Weiblen, http://mew.cx/glsl_quickref.pdf (accessed October 2012)

[GLSLSpec12] GLSL Specification, Khronos Group, http://www.opengl.org/documentation/glsl/ (accessed October 2012)