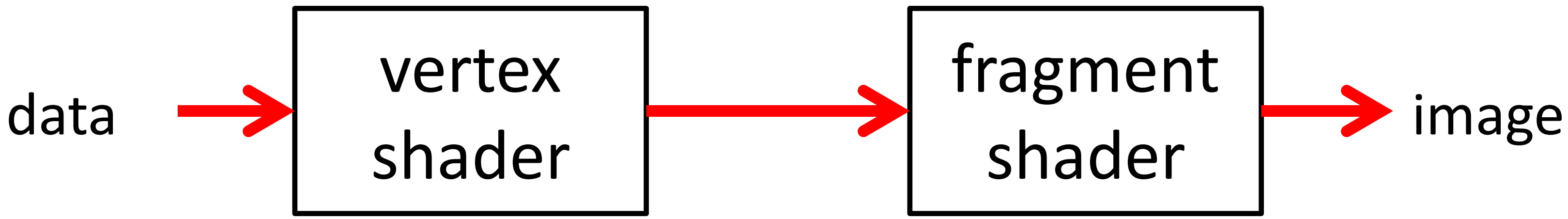
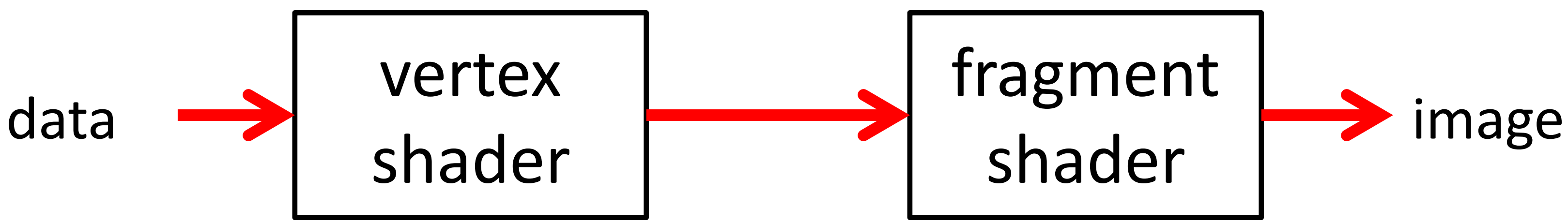
Assignment 4

Shading

GLSL Basics



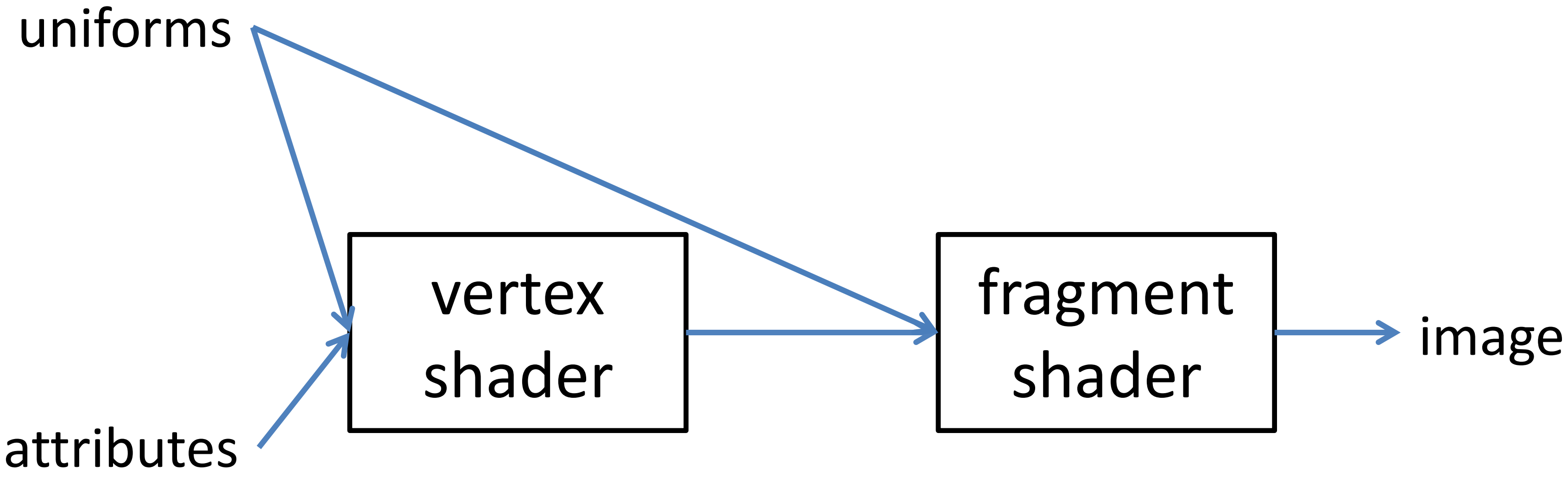
GLSL Basics



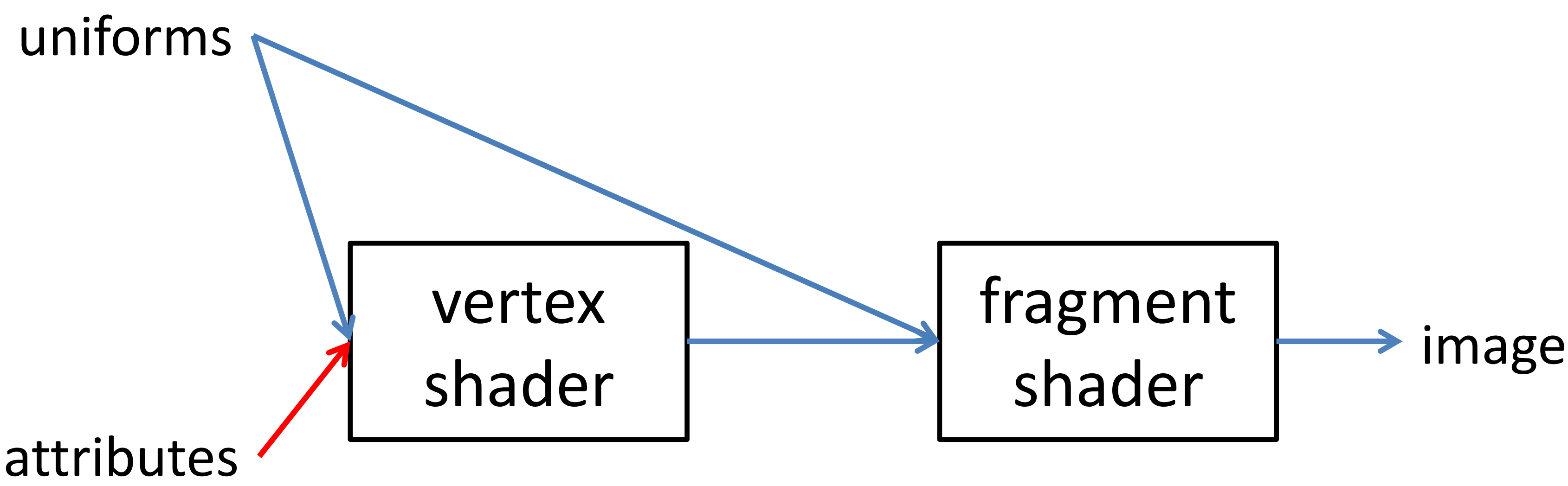
1x per vertex

1x per pixel inside of a triangle

GLSL Basics



GLSL Basics



Attributes:

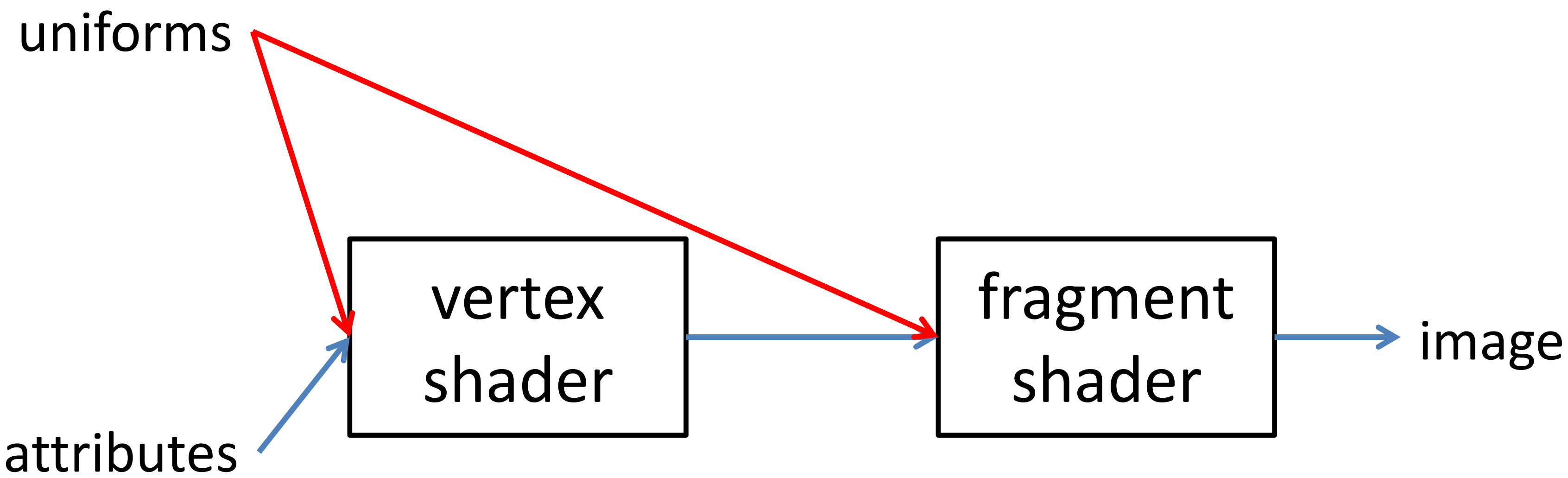
= type ***in*** in vertexshader

Input for each vertex

Read only access in vertexshader

Examples: normals, colors, vertexpositions etc.

GLSL Basics



uniforms:

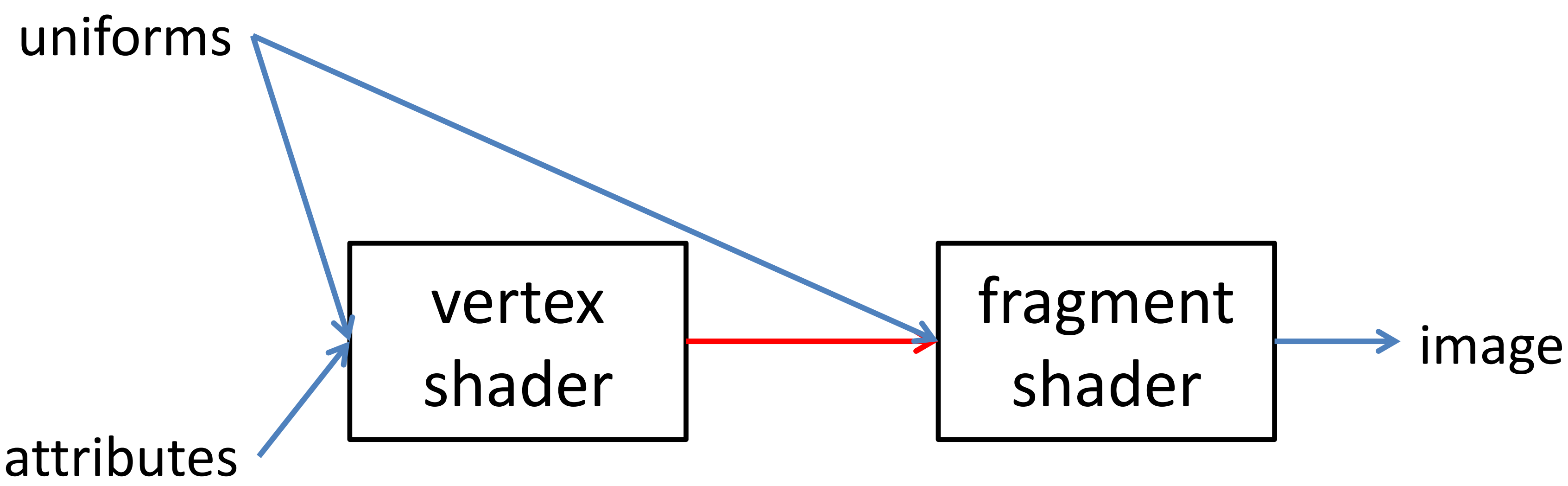
= type ***uniform*** in vertex- and fragmentshader

Are identical for all vertices and fragments

Read only access in vertex- **and** fragmentshader

Examples: properties of lightsources, transformation-matrices etc.

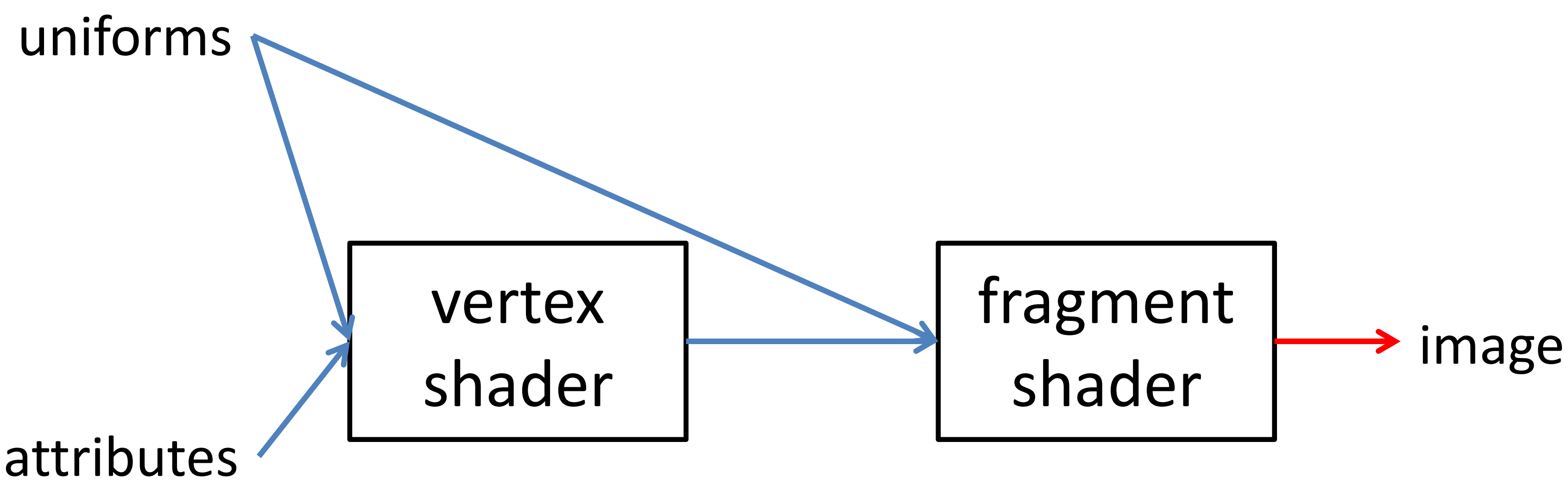
GLSL Basics



Output variables of the vertexshader are <u>automatically</u> interpolated perspective correctly and passed to the fragmentshader as input variables

= type ***out*** in vertexshader / type ***in*** in fragmentshader

GLSL Basics



= type ***out*** in fragmentshader

Color of the pixel in the framebuffer (=image)

<u>Z-buffering is performed automatically</u>

GLSL

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

GLSL

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

GLSL

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

GLSL Datatypes

vec2, vec3, vec4:

vectors of float elements

Direct access to x,y,z,w-components possible by using point-operator „.“

GLSL Datatypes

vec2, vec3, vec4:

vectors of float elements

Direct access to x,y,z,w-components possible by using point-operator „.“

Example:

```
vec4 myVec = vec4(1,2,3,4);
myVec.x         -> 1
myVec.xy        -> (1,2)
myVec.yx        -> (2,1)
myVec.zzz       -> (3,3,3)
myVec.wwxy      -> (4,4,1,2)
```

GLSL Datatypes

mat3, mat4:

Float-matrices of size 3x3 or 4x4

Access on specific element possible with double brackets „[ ][ ]“
Access on specific row possible with single bracket „[ ]“

GLSL Datatypes

mat3, mat4:

Float-matrices of size 3x3 or 4x4

Access on specific element possible with double brackets „[ ][ ]“
Access on specific row possible with single bracket „[ ]“

Example:

```
mat3 myMat= (1, 2, 3,
             4, 5, 6,
             7, 8, 9);
myMat[2][2]        -> 9
myMat[2]           -> (7,8,9)
```

GLSL Datatypes

sampler2D:

Identifier for 2D-texture

Used to fetch data from texture

…more on this later…

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

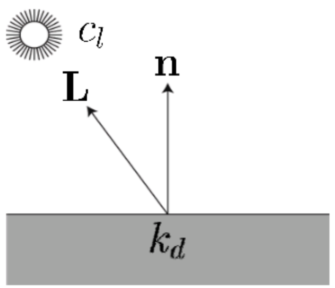
fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```



Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

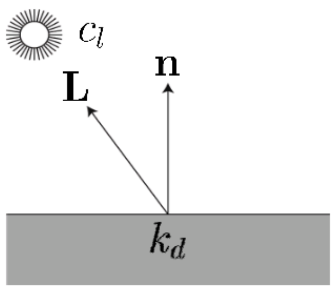
fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```



Important:
L and n have to be in the same coordinate system!

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

pass texture coordinate on to fragment shader

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

predefined output variable gl_Position
The vertexshader **must** always assign a value to this!

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

$$\mathbf{p}' = \mathbf{D}\mathbf{P}\mathbf{C}^{-1}\mathbf{M}\mathbf{p}$$

Object space

World space

Camera space

Canonic view volume

Image space

Example

diffuse shading with texture for directional light

vertex shader

```
uniform mat4 projection;
uniform mat4 modelview;
uniform vec4 lightDirection;

in vec3 normal;
in vec4 position;
in vec2 texcoord;

out float ndotl;
out vec2 frag_texcoord;

void main()
{

    ndotl =
        max(dot(modelview * vec4(normal,0), lightDirection),0);

    frag_texcoord = texcoord;

    gl_Position = projection * modelview * position;
}
```

fragment shader

```
uniform sampler2D myTexture;

in float ndotl;
in vec2 frag_texcoord;

out vec4 frag_shaded;

void main()
{
    frag_shaded = ndotl * texture(myTexture, frag_texcoord);
}
```

texture(***sampler2D***, ***vec2***)
= predefined function for texture-fetches.

argument 1: texture identifier
argument 2: (u,v) - texture coordinates
(normalized to range [0,1])

Return value is of type ***vec4***

Passing Variables from Host to GLSL

glUniform* binds uniform data to names in shader

Each datatype has his own **glUniform***-function:

glUniform(1|2|3|4)(f|i):

(1|2|3|4): dimension of type
(f|i): type (float or int)

Passing Variables from Host to GLSL

Example:

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Example:

Returns identifier for a specific uniform variable in a specific shader

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Example:

Identifier is required to let glUniform know to which variable a value should be bound

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Example:

Type of univorm variable in shader
Here its **vec4**

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Example:

Data that should be passed. List of arguments depends on type.
Here, for vec4, we need 4 values

```
int id = gl.glGetUniformLocation(activeShader.programId(), "lightDirection");
gl.glUniform4f(id, 0, 0, 1, 0);
```

Passing Variables from Host to GLSL

Arrays are passed with *glUniform*****v*** (more on this later...)

Matrices are passed with *glUniform****Matrix**********v***

For more detail refer to:
http://www.opengl.org/sdk/docs/man/xhtml/glUniform.xml

Assignments

Preparation:

- Modify jrtr (refer to assignment description):
    - SceneManager must be able to handle several lightsources
    - Shapes should have a reference to a material
    - Materials should have a reference to the shaders they use

Assignments

1. Create a diffuse shader for multiple point lights:

Assignments

1. Create a diffuse shader for multiple point lights:

   - Point lights have a radiance $\boldsymbol{c_l}$ and a position $\boldsymbol{p}$.
   - Objects have a diffuse reflection coefficient $\boldsymbol{k_d}$

Assignments

1. Create a diffuse shader for multiple point lights:

    - Point lights have a radiance $\boldsymbol{c_l}$ and a position $\boldsymbol{p}$.
    - Objects have a diffuse reflection coefficient $\boldsymbol{k_d}$

Uniforms!

Assignments

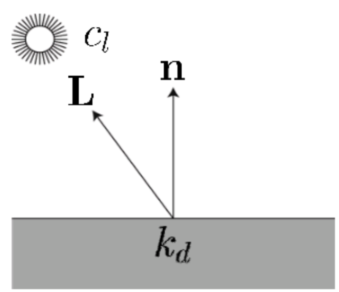
1. Create a diffuse shader for multiple point lights:

    - Point lights have a radiance $\boldsymbol{c_l}$ and a position $\boldsymbol{p}$.
    - Objects have a diffuse reflection coefficient $\boldsymbol{k_d}$

    Uniforms!

    - **Diffuse shading of an object with several point lights is:**

$$\sum_i c_{l_i} k_d (n \cdot L_i)$$



Assignments

1. Create a diffuse shader for multiple point lights:

- Uniforms of the point lights can be passed as array
- **But**: Size of arrays must be known to GLSL ***at compile time!***

- Therefore its okay if the maximum number of point sizes is fixed (as long as its >1 ☺)

```
#define MAX_LIGHTS 8

uniform vec3 light_color[MAX_LIGHTS];
```

Assignments

1. Create a diffuse shader for multiple point lights:

    - To pass arrays we need to use **glUniform*v**

Assignments

1. Create a diffuse shader for multiple point lights:

- To pass arrays we need to use **glUniform*v**

Example:

```
int numOfElements = 3;
float[] dataArray = {1,2,3,4,5,5};
int offset = 0;

gl.glUniform2fv(id, numOfElements, dataArray, offset);
```

Then the uniform-array in the shader has the values

{ vec2(1,2), vec2(3,4), vec2(5,5) }

Assignments

2. Per-pixel Phong shading for several point lights:

- Objects have an additional specular reflection coefficient $\boldsymbol{k_s}$ and a Phong-exponent $\boldsymbol{p}$.

Assignments

2. Per-pixel Phong shading for several point lights:

- Objects have an additional specular reflection coefficient $\boldsymbol{k_s}$ and a Phong-exponent $\boldsymbol{p}$.

  => more Uniforms!

Assignments

2. Per-pixel Phong shading for several point lights:

- Objects have an additional specular reflection coefficient $\boldsymbol{k_s}$ and a Phong-exponent $\boldsymbol{p}$.

  => more Uniforms!

- **We need to calculate:**

$$\sum_i c_{l_i} \left(k_d (n \cdot L_i) + k_s (R \cdot e)^p \right)$$

Assignments

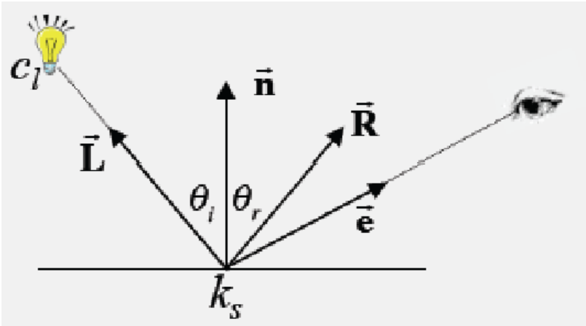
2. Per-pixel Phong shading for several point lights:

- Objects have an additional specular reflection coefficient $\boldsymbol{k_s}$ and a Phong-exponent $\boldsymbol{p}$.

  => more Uniforms!

- **We need to calculate:**

$$\sum_i c_{l_i}\left(k_d(n \cdot L_i) + k_s(R \cdot e)^p\right)$$



Assignments

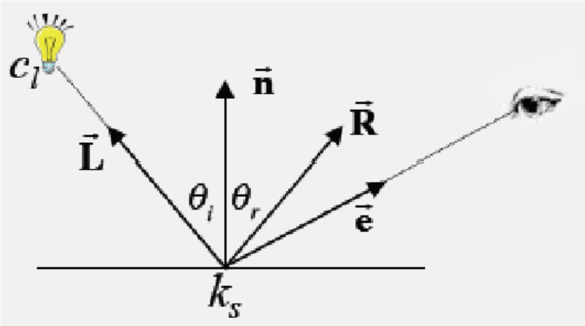
2. Per-pixel Phong shading for several point lights:

- Objects have an additional specular reflection coefficient $\boldsymbol{k_s}$ and a Phong-exponent $\boldsymbol{p}$.

  => more Uniforms!

- **We need to calculate:**

$$\sum_i c_{l_i} \left(k_d (n \cdot L_i) + k_s (R \cdot e)^p \right)$$



- **R** can be computed using the predefined function **reflect**
- For computing **e** its useful to pass the camera position as uniform variable, because $e = camera_{pos} - surface_{pos}$

Assignments

3. Texturing:

- Extend shader from exercise 2 to support textures!
    - Meaning $\boldsymbol{k_d}$ becomes the color that is fetched from the texture

Assignments

3. Texturing:

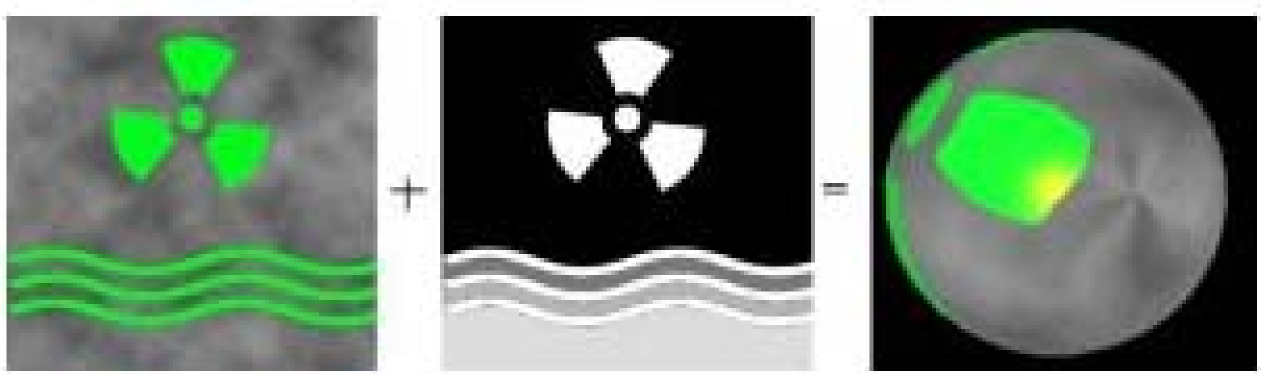
- Extend shader from exercise 2 to support textures!
  - Meaning $\boldsymbol{k_d}$ becomes the color that is fetched from the texture
- Further, extend the shader to support a **gloss map**

Assignments

3. Texturing:

- **Gloss map**:

  An additional texture whose brightness (sum of R,B and G) is used to control the specular coefficient



Assigmnent

4. Experiment with shaders:

Create your own shader, that can do whatever you want ☺

Assigmnent

4. Experiment with shaders:

You can use code from the internet if you want!

(but you probably need to modify it to work with jrtr...)

...but feel free to do your own stuff!

Shader Ideas

Toonshader

Shader Ideas

Procedural Brick Shader

Shader Ideas

Procedural Stripe Shader

Shader Ideas

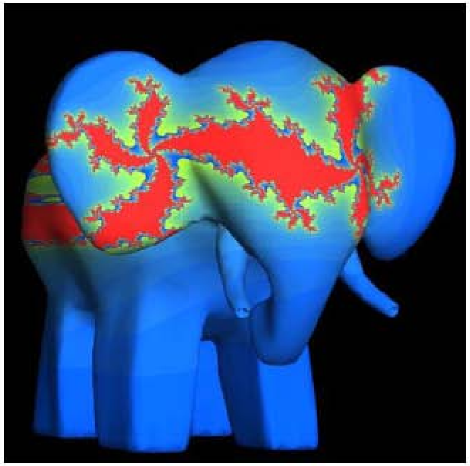
Procedural Noise Shader

(if you dare...)

Shader Ideas

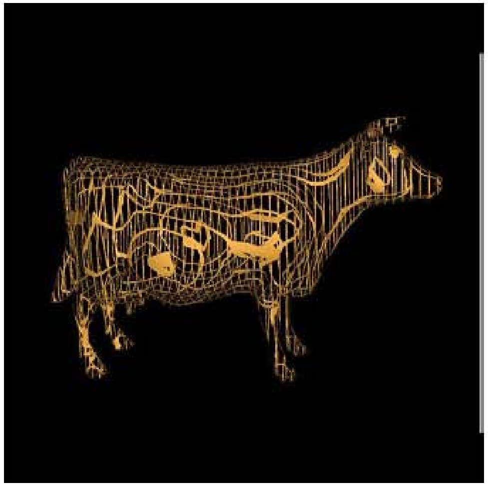
Glyph Bombing

Gooch Shading



Julia Set

Toy Ball



Discarding Fragments

Remarks

- The jrtr uses GLSL 1.5 .
  But most tutorials are still using older GLSL verisons...

- There is a GLSL syntax highlight add-on for eclipse:
  http://sourceforge.net/projects/glshaders/

  To make it work you need to modify the file associations of eclipse:

  Goto: *Window->Preferences->General->Editors->File Associations*
  and add **.frag* und **.vert* as file types and associate them with GLSL

Remarks

GLSL should be hardware independent in theory...
…but this is not always the case

=>Therefore program the stuff on the machine you use to demonstrate it!