Introduction to GLSL

(OpenGL Shading Language)

Advanced Graphics Programming

What is GLSL

GLSL: **G**raphics **L**ibrary **S**hading **L**anguage

- Syntax similar to C/C++
- Language used to write shaders
  - Vertex, fragment, geometry, tessellation, compute
- First available in OpenGL 2.0 (2004)
- Alternatives
  - Nvidia Cg
  - Microsoft HLSL

Without GLSL

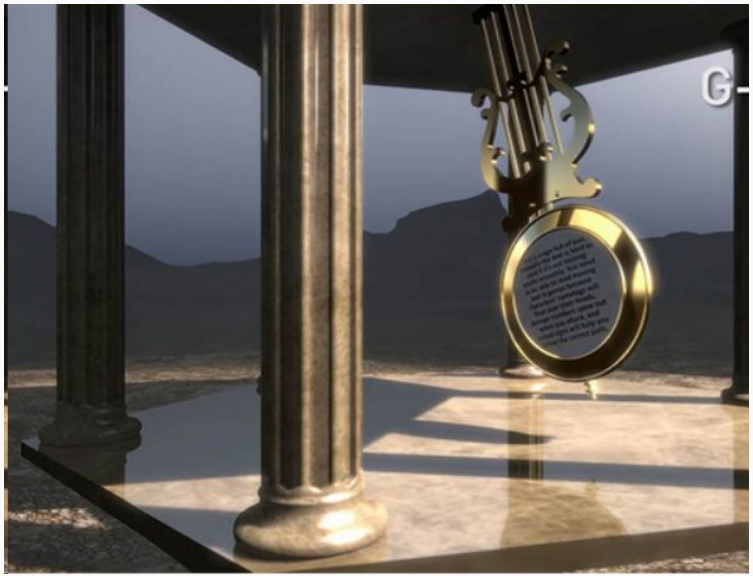
OpenGL 25 years ago

- No programmable pipeline
- Graphics cards had a very limited set of operations
- Vertex transform / fragment shading was hardcoded into GPUs

With GLSL

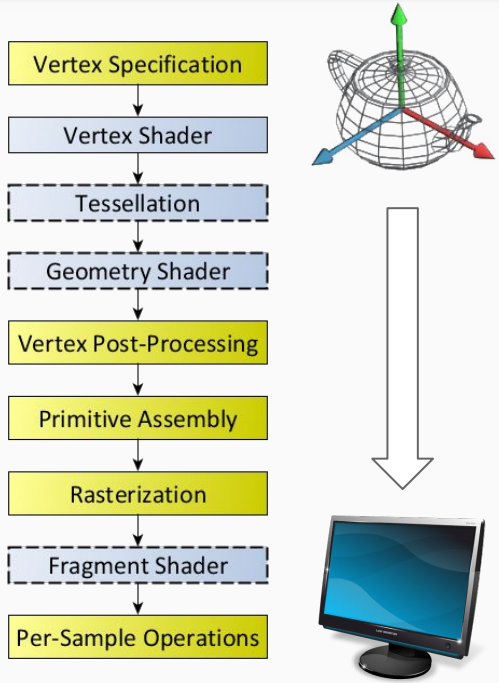
Modern OpenGL

- More parts of the GPU are programmable (but not all)
  - Vertex processor
  - Fragment processor
- The flexibility of programmable shaders allows the creation of astonishing visual effects



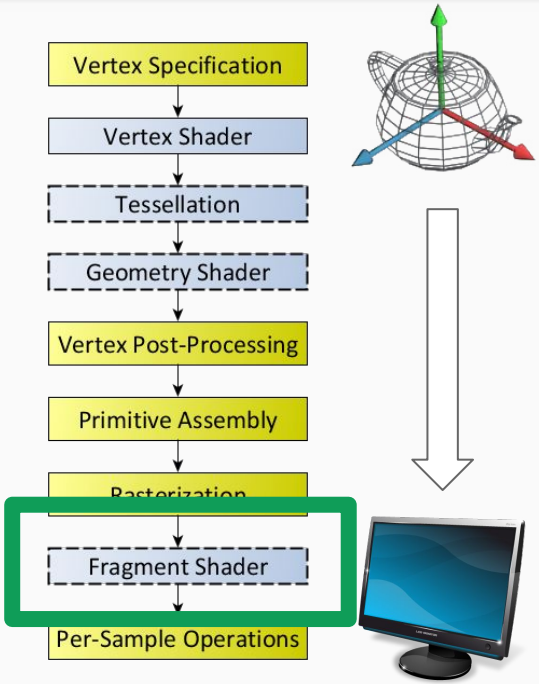
What is a shader program?

- A small program that controls parts of the graphics pipeline
- Consists of at least 2 separate parts
  - Vertex shader (controls vertex transforms)
  - Fragment shader (controls fragment shading)
- Can also contain
  - Geometry shader (controls additional geometry generation)
  - Tessellation shader (controls primitive tessellation)



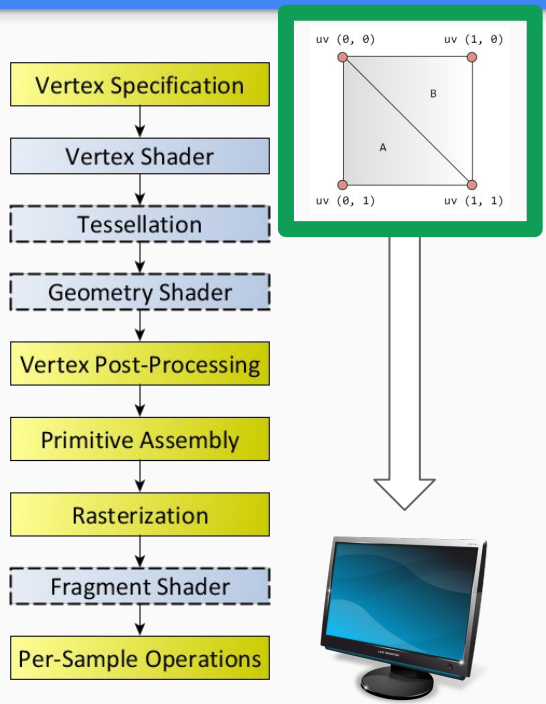
What is a shader program?

- In this part of the subject, we are going to practice a little bit with **fragment shaders**



What is a shader program?

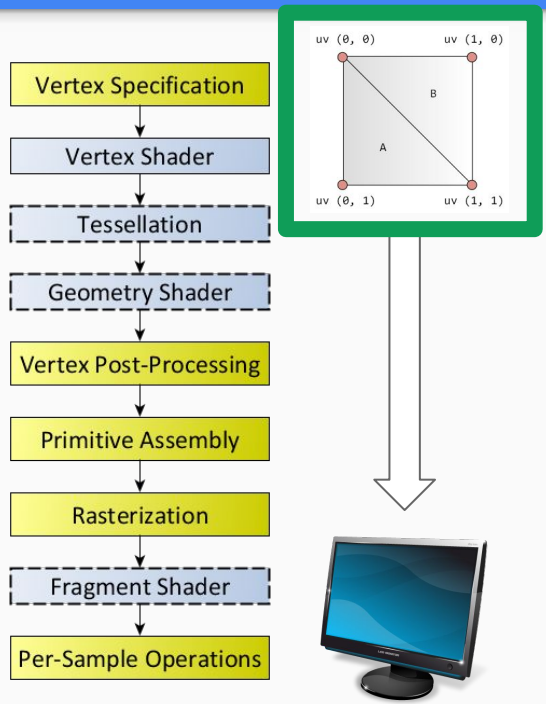
- In this part of the subject, we are going to practice a little bit with **fragment shaders**
- We will not worry about the model: just **a quad (two triangles)** will be rendered



What is a shader program?

- In this part of the subject, we are going to practice a little bit with **fragment shaders**
- We will not worry about the model: just **a quad (two triangles)** will be rendered
- Shadertoy!

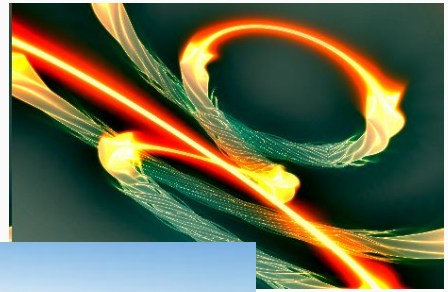
Shadertoy



Take a look...

Even big procedural worlds can be created within the fragment processor.

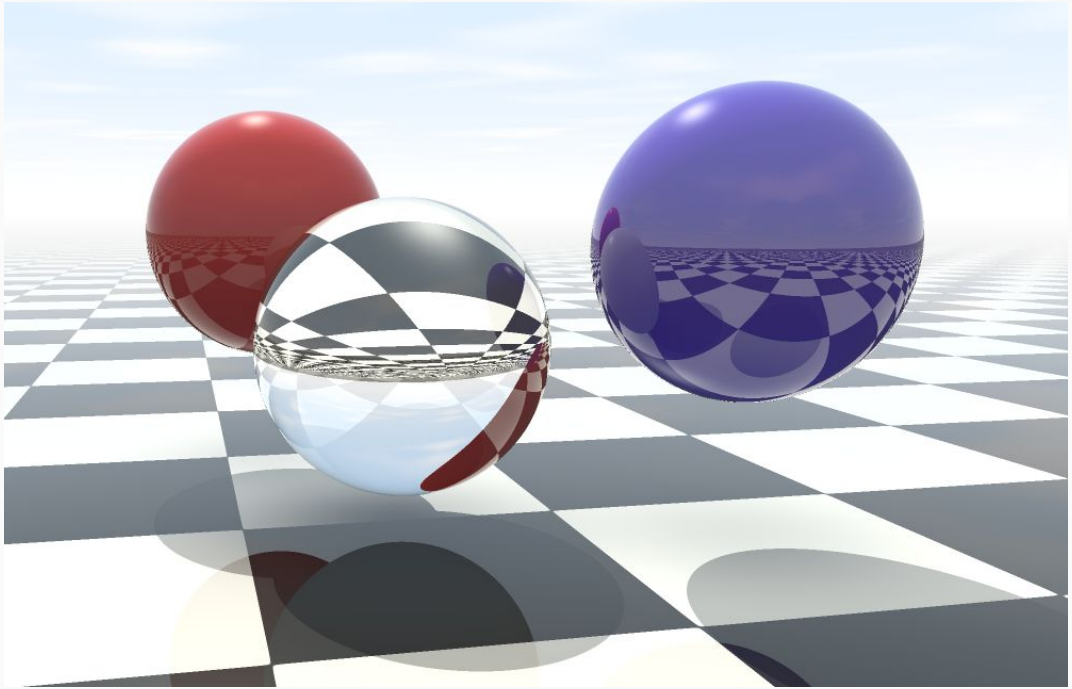
- GPU intensive
- Difficult maths :_D



Some examples:

- https://www.shadertoy.com/
- https://www.shadertoy.com/results?query=landscape
- https://www.shadertoy.com/results?query=reflections

Our goal



GLSL basics

Data types

```
// booleans
bool
bvec2
bvec3
bvec4
```

```
// integers
int
ivec2
ivec3
ivec4
```

```
// u. integers
uint
uvec2
uvec3
uvec4
```

```
// floats
float
vec2
vec3
vec4
```

```
// double
double
dvec2
dvec3
dvec4
```

```
// matrices
mat2
mat3
mat4
```

```
// matrices (doubles)
dmat2
dmat3
dmat4
```

```
// some variable definitions
bool  b;
int   i;
float v;
vec2 texCoord;
vec3 rgbColor;
vec4 position;
mat3 rotationMatrix;
mat4 transformMatrix;
```

Data types

```
// GLSL type          // OpenGL texture type
sampler1D             GL_TEXTURE_1D
sampler2D             GL_TEXTURE_2D
sampler3D             GL_TEXTURE_3D
samplerCube           GL_TEXTURE_CUBE_MAP
sampler2DRect         GL_TEXTURE_RECTANGLE
sampler1DArray        GL_TEXTURE_1D_ARRAY
sampler2DArray        GL_TEXTURE_2D_ARRAY
samplerCubeArray      GL_TEXTURE_CUBE_MAP_ARRAY
samplerBuffer         GL_TEXTURE_BUFFER
sampler2DMS           GL_TEXTURE_2D_MULTISAMPLE
sampler2DMSArray      GL_TEXTURE_2D_MULTISAMPLE_ARRAY
```

- These are sampler for standard **color** textures
  - Color values will have values from 0 to 1.
- The prefixes i (*isampler*), and u (*usampler*) can be used for textures storing integers.
- Used for uniform inputs.

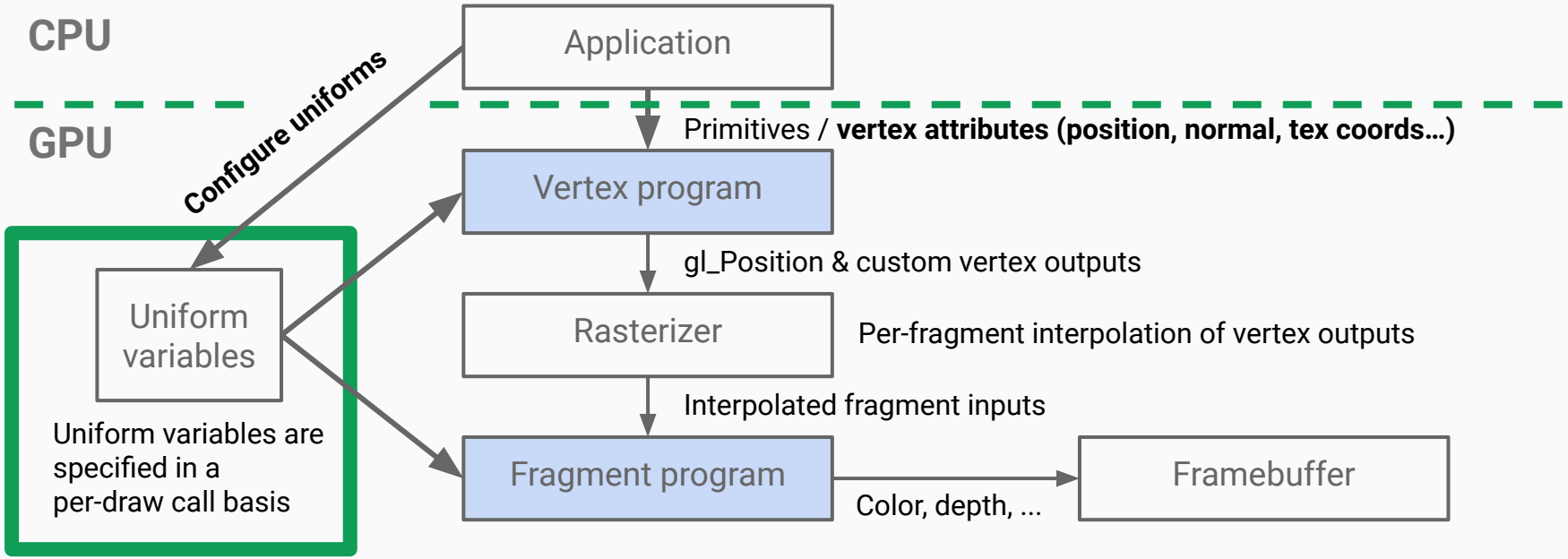
Data types

Reference

For more information about GLSL data types, visit the wiki reference page at **khronos.org**:

https://www.khronos.org/opengl/wiki/Data_Type_(GLSL)

Uniform variables (any shader inputs)



Uniform variables in Shadertoy

```
uniform vec3  iResolution;
uniform float iTime;
uniform vec4  iMouse;
// Many others...
```

Shadertoy documentation:
https://www.shadertoy.com/howto

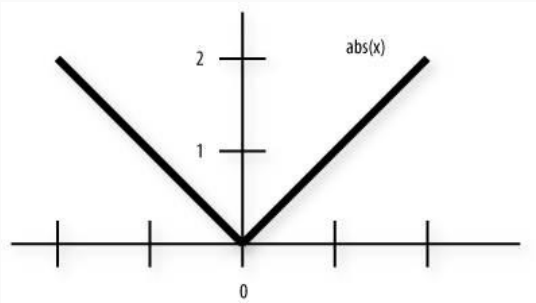
Built-in functions

GLSL provides fairly big set of functions to develop shaders

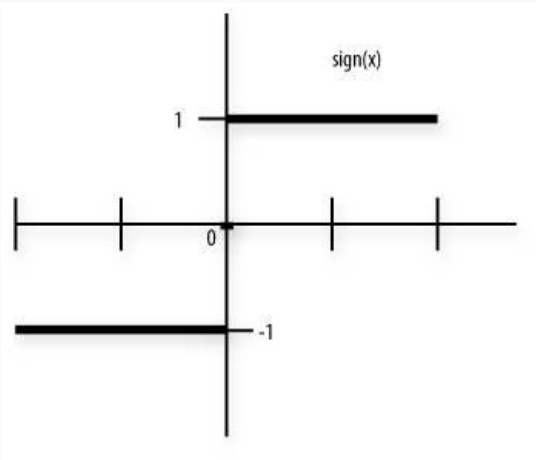
- Mathematics
  - GPU-accelerated
  - Efficient
- Texture lookups

Built-in functions: Abs / Sign

```
// Returns x if x >= 0; otherwise, it returns -x.
float abs (float x)
vec2 abs (vec2 x)
vec3 abs (vec3 x)
vec4 abs (vec4 x)
```



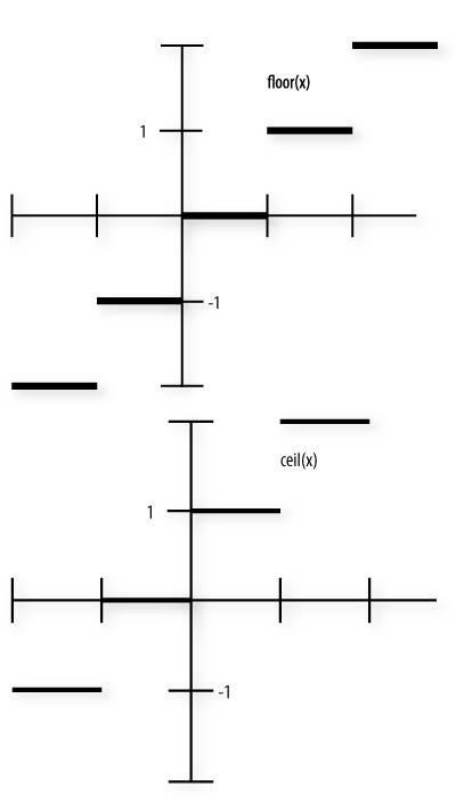

```
// Returns 1.0 if x > 0, 0.0 if x = 0, or -1.0 if x < 0.
float sign (float x)
vec2 sign (vec2 x)
vec3 sign (vec3 x)
vec4 sign (vec4 x)
```



Built-in functions: Floor / Ceil

```
// Returns a value equal to the nearest integer that is
// less than or equal to x.
float floor (float x)
vec2 floor (vec2 x)
vec3 floor (vec3 x)
vec4 floor (vec4 x)
```

```
// Returns a value equal to the nearest integer that is
// greater than or equal to x.
float ceil (float x)
vec2 ceil (vec2 x)
vec3 ceil (vec3 x)
vec4 ceil (vec4 x)
```

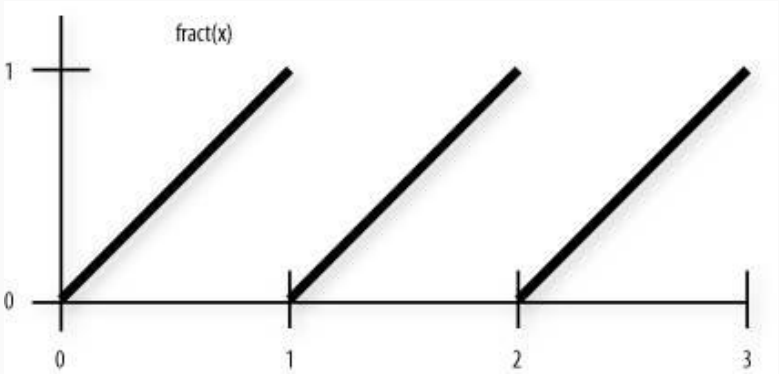


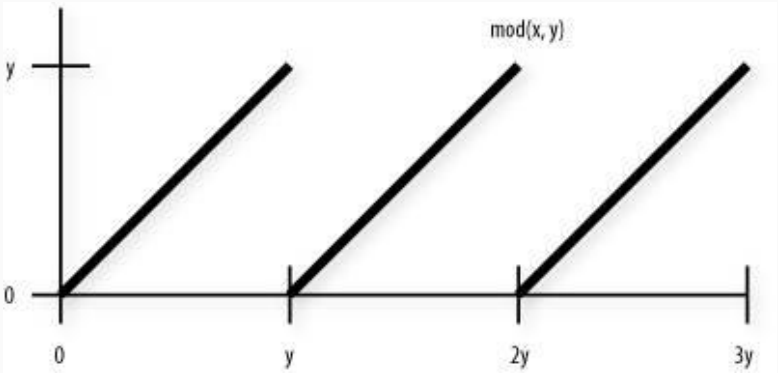
Built-in functions: Fract / mod

```
// Returns x - floor (x).
float fract (float x)
vec2 fract (vec2 x)
vec3 fract (vec3 x)
vec4 fract (vec4 x)

// Modulus. Returns x - y * floor (x/y) for each
// component in x using the floating-point value y.
float mod (float x, float y)
vec2 mod (vec2 x, float y)
vec3 mod (vec3 x, float y)
vec4 mod (vec4 x, float y)

// Modulus. Returns x - y * floor (x/y) for each
// component in x using the corresponding component of y.
vec2 mod (vec2 x, vec2 y)
vec3 mod (vec3 x, vec3 y)
vec4 mod (vec4 x, vec4 y)
```



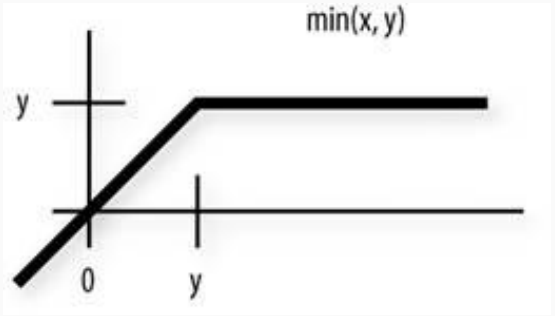


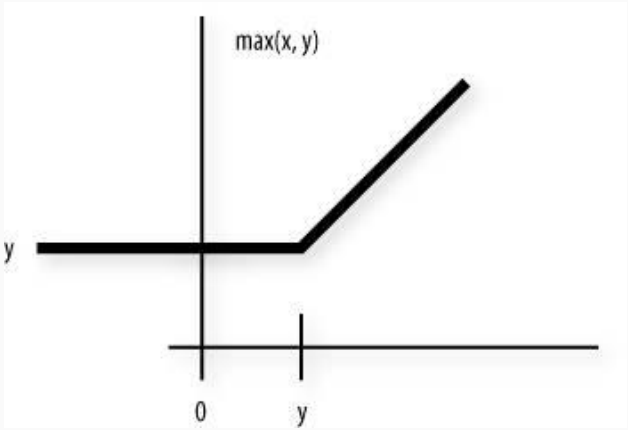
Built-in functions: Min / Max

```
// Returns minimum of each component of x compared with
the floating-point value y.
vec2 min (vec2 x, float y)
vec3 min (vec3 x, float y)
vec4 min (vec4 x, float y)

// Returns y if x < y; otherwise, it returns x.
float max (float x, float y)
vec2 max (vec2 x, vec2 y)
vec3 max (vec3 x, vec3 y)
vec4 max (vec4 x, vec4 y)

// Returns maximum of each component of x compared
// with the floating-point value y.
vec2 max (vec2 x, float y)
vec3 max (vec3 x, float y)
vec4 max (vec4 x, float y) // and more...
```

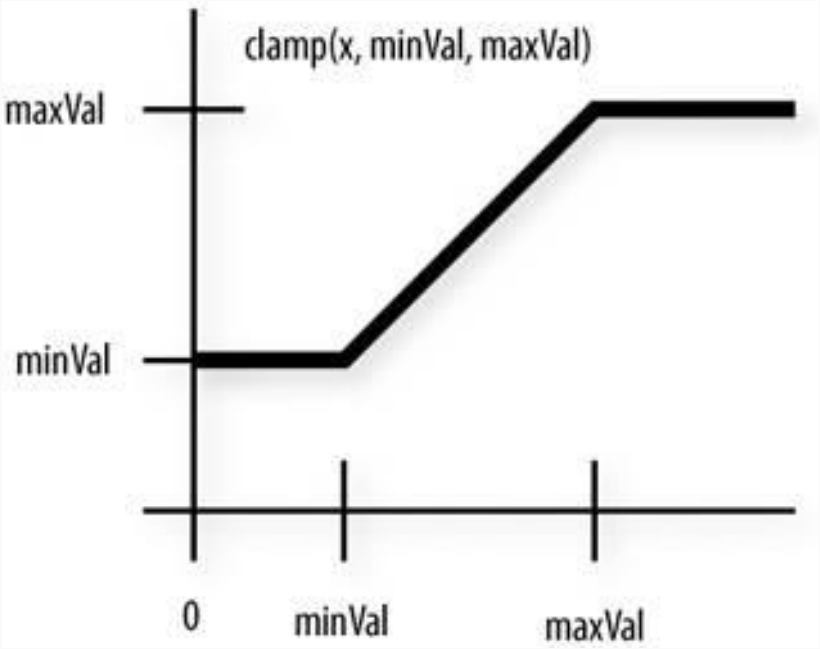




Built-in functions: Clamp

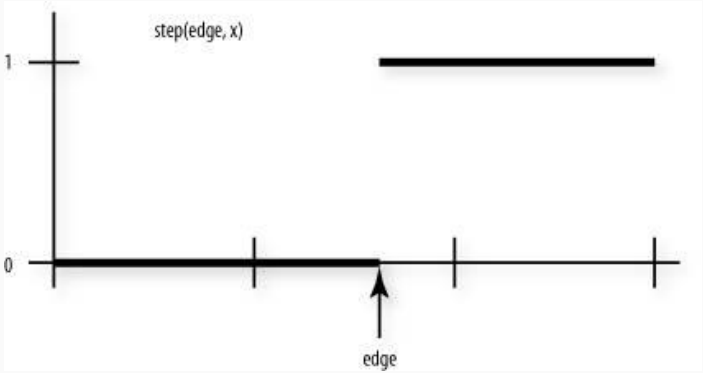
```
// Returns min (max (x, minVal), maxVal) for each
// component in x using the floating-point values minVal
// and maxVal. Results are undefined if minVal > maxVal.
float clamp (float x, float minVal, float maxVal)
vec2 clamp (vec2 x, float minVal, float maxVal)
vec3 clamp (vec3 x, float minVal, float maxVal)
vec4 clamp (vec4 x, float minVal, float maxVal)
```

```
// Returns the component-wise result of min (max (x,
// minVal), maxVal). Results are undefined if minVal >
// maxVal.
vec2 clamp (vec2 x, vec2 minVal, vec2 maxVal)
vec3 clamp (vec3 x, vec3 minVal, vec3 maxVal)
vec4 clamp (vec4 x, vec4 minVal, vec4 maxVal)
```

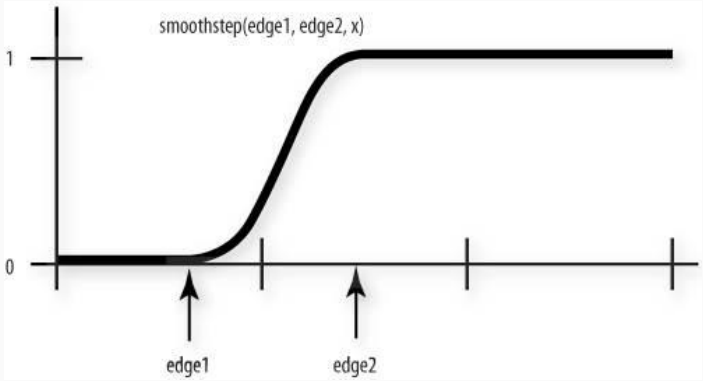


Built-in functions: Step / Smoothstep

```
// Returns 0 if x < edge; otherwise, it returns 1.0.
float step (float edge, float x)
vec2 step (vec2 edge, vec2 x)
vec3 step (vec3 edge, vec3 x)
vec4 step (vec4 edge, vec4 x)
```



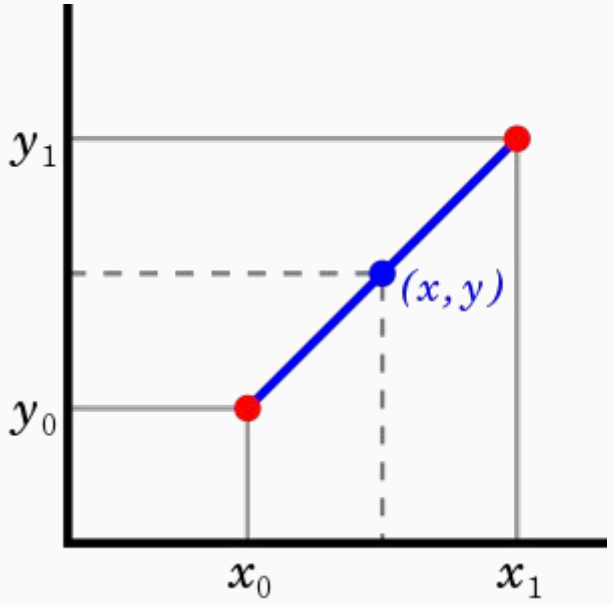

```
// Returns 0 if x <= edge0 and 1.0 if x >= edge1 and
// performs smooth Hermite interpolation between 0 and 1
// when edge0 < x < edge1. Results are undefined if
// edge0 >= edge1.
float smoothstep (float edge0, float edge1, float x)
vec2 smoothstep (vec2 edge0, vec2 edge1, vec2 x)
vec3 smoothstep (vec3 edge0, vec3 edge1, vec3 x)
vec4 smoothstep (vec4 edge0, vec4 edge1, vec4 x)
```



Built-in functions: Mix

```
// Returns x * (1.0 - a) + y * a, i.e., the linear blend
// of x and y using the floating-point value a. The value
// for a is not restricted to the range [0,1].
float mix (float x, float y, float a)
vec2 mix (vec2 x, vec2 y, float a)
vec3 mix (vec3 x, vec3 y, float a)
vec4 mix (vec4 x, vec4 y, float a)

// Returns the component-wise result of x * (1.0 - a) + y
// * a, i.e., the linear blend of vectors x and y using
// the vector a. The value for a is not restricted to the
// range [0,1].
vec2 mix (vec2 x, vec2 y, vec2 a)
vec3 mix (vec3 x, vec3 y, vec3 a)
vec4 mix (vec4 x, vec4 y, vec4 a)
```



Built-in functions: Many others

- Trigonometry
  - Sin, cos, tan, asin, acos, atan… **// In radians!**
- Exponential
  - Pow, exp2, log2, sqrt, inversesqrt...
- Geometric
  - Length, distance, dot, cross, normalize, reflect, refract…
- Texture lookup
  - Before: texture1D, texture2D, texture3D, textureCube
  - Now: texture (overloads for all texture types)
- Texture info
  - textureSize

OpenGL Shading Language

Reference

For more information about GLSL, visit the wiki reference page at **khronos.org**:

https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language