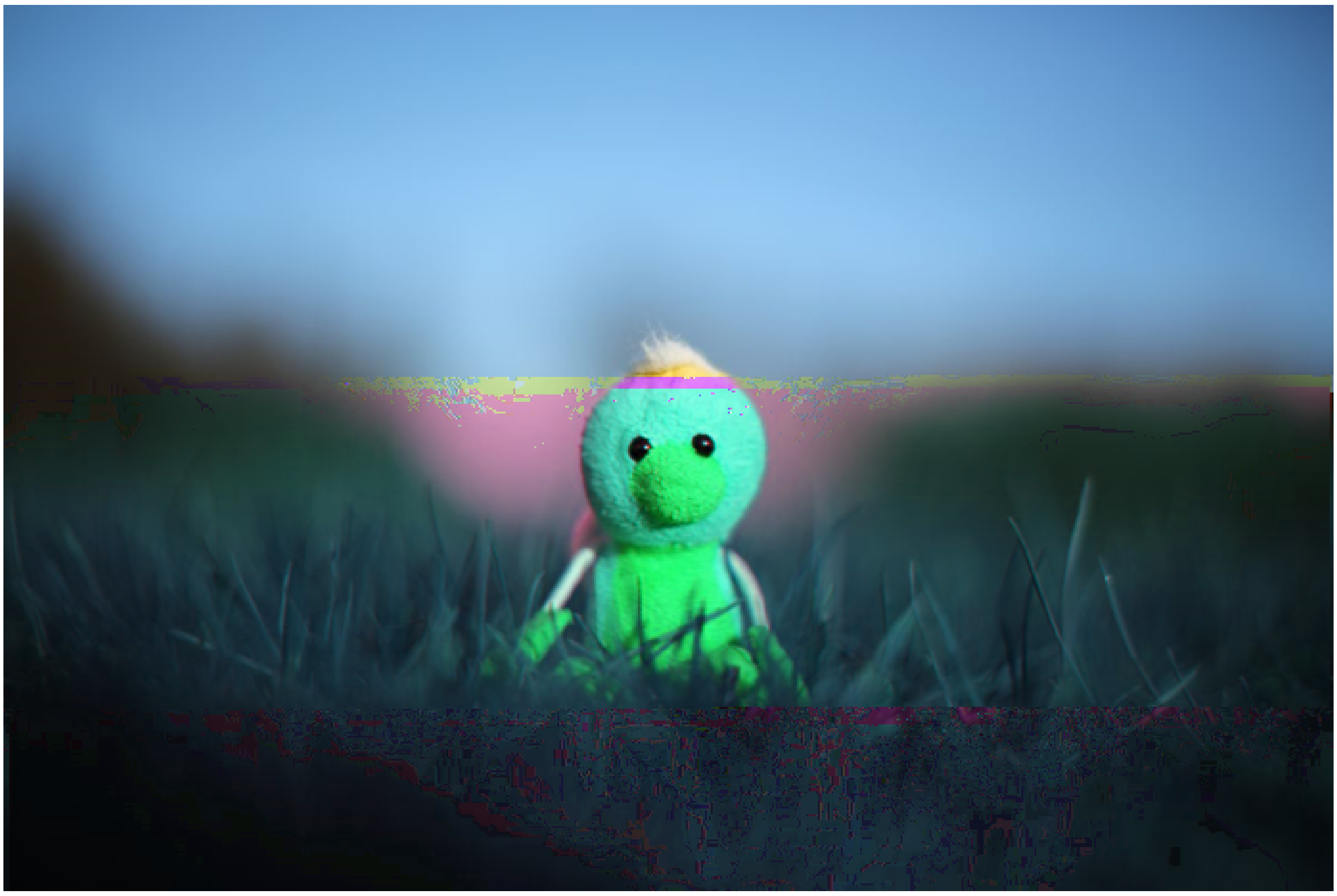
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•

•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•
•

-
-
-
-
-



<glm/glm.hpp>

```
#include <glm/glm.hpp>
```

```
// Include all GLM core / GLSL features
#include <glm/glm.hpp> // vec2, vec3, mat4, radians

// Include all GLM extensions
#include <glm/ext.hpp> // perspective, translate, rotate

glm::mat4 transform(glm::vec2 const& Orientation, glm::vec3 const& Translate,
glm::vec3 const& Up)
{
    glm::mat4 Proj = glm::perspective(glm::radians(45.f), 1.33f, 0.1f, 10.f);
    glm::mat4 ViewTranslate = glm::translate(glm::mat4(1.f), Translate);
    glm::mat4 ViewRotateX = glm::rotate(ViewTranslate, Orientation.y, Up);
    glm::mat4 View = glm::rotate(ViewRotateX, Orientation.x, Up);
    glm::mat4 Model = glm::mat4(1.0f);
    return Proj * View * Model;
}
```

Note: Including `<glm/glm.hpp>` and `<glm/ext.hpp>` is convenient but pull a lot of code which will significantly increase build time, particularly if these files are included in all source files. We may prefer to use the approaches describe in the two following sections to keep the project build fast.

```
#include <glm/vec2.hpp>               // vec2, bvec2, dvec2, ivec2 and uvec2
#include <glm/vec3.hpp>               // vec3, bvec3, dvec3, ivec3 and uvec3
#include <glm/vec4.hpp>               // vec4, bvec4, dvec4, ivec4 and uvec4
#include <glm/mat2x2.hpp>             // mat2, dmat2
#include <glm/mat2x3.hpp>             // mat2x3, dmat2x3
#include <glm/mat2x4.hpp>             // mat2x4, dmat2x4
#include <glm/mat3x2.hpp>             // mat3x2, dmat3x2
```

```
#include <glm/mat3x3.hpp>              // mat3, dmat3
#include <glm/mat3x4.hpp>              // mat3x4, dmat2
#include <glm/mat4x2.hpp>              // mat4x2, dmat4x2
#include <glm/mat4x3.hpp>              // mat4x3, dmat4x3
#include <glm/mat4x4.hpp>              // mat4, dmat4
#include <glm/common.hpp>              // all the GLSL common functions: abs, min,
mix, isnan, fma, etc.
#include <glm/exponential.hpp>         // all the GLSL exponential functions: pow,
log, exp2, sqrt, etc.
#include <glm/geometry.hpp>            // all the GLSL geometry functions: dot,
cross, reflect, etc.
#include <glm/integer.hpp>             // all the GLSL integer functions: findMSB,
bitfieldExtract, etc.
#include <glm/matrix.hpp>              // all the GLSL matrix functions: transpose,
inverse, etc.
#include <glm/packing.hpp>             // all the GLSL packing functions:
packUnorm4x8, unpackHalf2x16, etc.
#include <glm/trigonometric.hpp>       // all the GLSL trigonometric functions:
radians, cos, asin, etc.
#include <glm/vector_relational.hpp>  // all the GLSL vector relational functions:
equal, less, etc.
```

```
// Include GLM core features
#include <glm/vec2.hpp>           // vec2
#include <glm/vec3.hpp>           // vec3
#include <glm/mat4x4.hpp>         // mat4
#include <glm/trigonometric.hpp>  //radians

// Include GLM extension
#include <glm/ext/matrix_transform.hpp> // perspective, translate, rotate

glm::mat4 transform(glm::vec2 const& Orientation, glm::vec3 const& Translate,
glm::vec3 const& Up)
{
    glm::mat4 Proj = glm::perspective(glm::radians(45.f), 1.33f, 0.1f, 10.f);
    glm::mat4 ViewTranslate = glm::translate(glm::mat4(1.f), Translate);
    glm::mat4 ViewRotateX = glm::rotate(ViewTranslate, Orientation.y, Up);
    glm::mat4 View = glm::rotate(ViewRotateX, Orientation.x, Up);
    glm::mat4 Model = glm::mat4(1.0f);
    return Proj * View * Model;
}
```

```
// Include GLM vector extensions:
#include <glm/ext/vector_float2.hpp>                // vec2
#include <glm/ext/vector_float3.hpp>                // vec3
#include <glm/ext/vector_trigonometric.hpp>         // radians

// Include GLM matrix extensions:
#include <glm/ext/matrix_float4x4.hpp>              // mat4
#include <glm/ext/matrix_transform.hpp>             // perspective, translate,
rotate

glm::mat4 transform(glm::vec2 const& Orientation, glm::vec3 const& Translate,
glm::vec3 const& Up)
{
    glm::mat4 Proj = glm::perspective(glm::radians(45.f), 1.33f, 0.1f, 10.f);
    glm::mat4 ViewTranslate = glm::translate(glm::mat4(1.f), Translate);
    glm::mat4 ViewRotateX = glm::rotate(ViewTranslate, Orientation.y, Up);
    glm::mat4 View = glm::rotate(ViewRotateX, Orientation.x, Up);
    glm::mat4 Model = glm::mat4(1.0f);
    return Proj * View * Model;
}
```

`<GL/gl.h>` `<GL/glcorearb.h>`
`<GLES3/gl3.h>` `<GL/glu.h>` `<windows.h>`

GLM_FORCE_MESSAGES

```
#define GLM_FORCE_MESSAGES // Or defined when building (e.g. -DGLM_FORCE_SWIZZLE)
#include <glm/glm.hpp>
```

GLM_FORCE_MESSAGES

```
GLM: version 0.9.9.1
GLM: C++ 17 with extensions
GLM: Clang compiler detected
GLM: x86 64 bits with AVX instruction set build target
GLM: Linux platform detected
GLM: GLM_FORCE_SWIZZLE is undefined. swizzling functions or operators are
disabled.
GLM: GLM_FORCE_SIZE_T_LENGTH is undefined. .length() returns a glm::length_t, a
typedef of int following GLSL.
GLM: GLM_FORCE_UNRESTRICTED_GENTYPE is undefined. Follows strictly GLSL on valid
function genTypes.
GLM: GLM_FORCE_DEPTH_ZERO_TO_ONE is undefined. Using negative one to one depth
clip space.
GLM: GLM_FORCE_LEFT_HANDED is undefined. Using right handed coordinate system.
```

GLM_FORCE_PLATFORM_UNKNOWN

GLM_FORCE_COMPILER_UNKNOWN

GLM_FORCE_ARCH_UNKNOWN

GLM_FORCE_CSS_UNKNOWN

GLM_FORCE_CXX98

<glm/glm.hpp>

```
#define GLM_FORCE_CXX98
#include <glm/glm.hpp>
```

- GLM_FORCE_CXX11
- GLM_FORCE_CXX14
- GLM_FORCE_CXX14

```
#define GLM_FORCE_CXX11
#include <glm/glm.hpp>

// If the compiler doesn’t support C++11, compiler errors will happen.
```

GLM_FORCE_CXX17 GLM_FORCE_CXX14 GLM_FORCE_CXX14 GLM_FORCE_CXX11 GLM_FORCE_CXX11 GLM_FORCE_CXX98

vec4

GLM_FORCE_EXPLICIT_CTOR

```
#include <glm/glm.hpp>

void foo()
{
    glm::ivec4 a;
    ...

    glm::vec4 b(a); // Explicit conversion, OK
    glm::vec4 c = a; // Implicit conversion, OK
    ...
}
```

GLM_FORCE_EXPLICIT_CTOR

```
#define GLM_FORCE_EXPLICIT_CTOR
#include <glm/glm.hpp>

void foo()
{
    glm::ivec4 a;
    {
        glm::vec4 b(a); // Explicit conversion, OK
        glm::vec4 c = a; // Implicit conversion, ERROR
        ...
}
```

GLM_FORCE_INLINE

<glm/glm.hpp>

```
#define GLM_FORCE_INLINE
#include <glm/glm.hpp>
```

std::size_t

std::alignment_of std::align

std::aligned_storage

alignas

```
#define GLM_FORCE_DEFAULT_ALIGNED_GENTYPES
#include <glm/glm.hpp>

struct MyStruct
{
    glm::vec4 a;
    float b;
    glm::vec3 c;
};

void foo()
{
    printf("MyStruct requires memory padding: %d bytes\n", sizeof(MyStruct));
}

>>> MyStruct requires memory padding: 48 bytes
```

```
#include <glm/glm.hpp>

struct MyStruct
{
    glm::vec4 a;
    float b;
    glm::vec3 c;
};

void foo()
{
    printf("MyStruct is tightly packed: %d bytes\n", sizeof(MyStruct));
}

>>> MyStruct is tightly packed: 32 bytes
```

Note: GLM SIMD optimizations require the use of aligned types

`/arch:AVX`

`GLM_FORCE_SSE2` `GLM_FORCE_SSE3` `GLM_FORCE_SSSE3` `GLM_FORCE_SSE41` `GLM_FORCE_SSE42` `GLM_FORCE_AVX` `GLM_FORCE_AVX2` `GLM_FORCE_AVX512`

`GLM_FORCE_PURE`

`constexpr`

```
#define GLM_FORCE_PURE
#include <glm/glm.hpp>

static_assert(glm::vec4::length() == 4, "Using GLM C++ 14 constexpr support for
compile time tests");
```

```
// GLM code will be compiled using pure C++ code without any intrinsics
```

```
#define GLM_FORCE_SIMD_AVX2
#include <glm/glm.hpp>

// If the compiler doesn’t support AVX2 instrinsics, compiler errors will happen.
```

```
precision mediump int;
precision highp float;
```

`glm.hpp`

```
#define GLM_FORCE_PRECISION_MEDIUMP_INT
#define GLM_FORCE_PRECISION_HIGHP_FLOAT
#include <glm/glm.hpp>
```

`glm::vec\*` `glm::mat\*`

- `GLM_FORCE_PRECISION_LOWP_FLOAT`
- `GLM_FORCE_PRECISION_MEDIUMP_FLOAT`
- `GLM_FORCE_PRECISION_HIGHP_FLOAT`

`glm::dvec\*` `glm::dmat\*`

- `GLM_FORCE_PRECISION_LOWP_DOUBLE`
- `GLM_FORCE_PRECISION_MEDIUMP_DOUBLE`
- `GLM_FORCE_PRECISION_HIGHP_DOUBLE`

`glm::ivec\*`

- `GLM_FORCE_PRECISION_LOWP_INT`
- `GLM_FORCE_PRECISION_MEDIUMP_INT`
- `GLM_FORCE_PRECISION_HIGHP_INT`

`glm::uvec\*`

- GLM_FORCE_PRECISION_LOWP_UINT
- GLM_FORCE_PRECISION_MEDIUMP_UINT
- GLM_FORCE_PRECISION_HIGHP_UINT

--m4-single-only GLM_FORCE_SINGLE_ONLY

variable.x variable.xzy variable.zxyy

xyzw

rgba stpq

```
vec4 A;
vec2 B;

B.yx = A.wy;
B = A.xx;
vec3 C = A.bgr;
vec3 D = B.rsz; // Invalid, won't compile
```

GLM_FORCE_SWIZZLE

Note: Enabling swizzle expressions will massively increase the size of your binaries and the time it takes to compile them!

2.13.1. Swizzle functions for standard C++ 98

```
#define GLM_FORCE_SWIZZLE // Or defined when building (e.g. -DGLM_FORCE_SWIZZLE)
#include <glm/glm.hpp>

void foo()
{
    glm::vec4 const ColorRGBA = glm::vec4(1.0f, 0.5f, 0.0f, 1.0f);
    glm::vec3 const ColorBGR = ColorRGBA.bgr();

    glm::vec3 const PositionA = glm::vec3(1.0f, 0.5f, 0.0f);
    glm::vec3 const PositionB = PositionXYZ.xyz() * 2.0f;

    glm::vec2 const TexcoordST = glm::vec2(1.0f, 0.5f);
    glm::vec4 const TexcoordSTPQ = TexcoordST.stst();
}
```

copy *can't*

```
#define GLM_FORCE_SWIZZLE
#include <glm/glm.hpp>

void foo()
{
  glm::vec3 const A = glm::vec3(1.0f, 0.5f, 0.0f);

  // No compiler error, but A is not modified.
  // An anonymous copy is being modified (and then discarded).
  A.bgr() = glm::vec3(2.0f, 1.5f, 1.0f); // A is not modified!
}
```

2.13.2. Swizzle operations for C++ 98 with language extensions

non-standard language extension `struct` `union`

`GLM_FORCE_SWIZZLE` `#define`

```
#define GLM_FORCE_SWIZZLE
#include <glm/glm.hpp>

// Only guaranteed to work with Visual C++!
// Some compilers that support Microsoft extensions may compile this.
void foo()
{
  glm::vec4 ColorRGBA = glm::vec4(1.0f, 0.5f, 0.0f, 1.0f);

  // l-value:
  glm::vec4 ColorBGRA = ColorRGBA.bgra;

  // r-value:
  ColorRGBA.bgra = ColorRGBA;

  // Both l-value and r-value
  ColorRGBA.bgra = ColorRGBA.rgba;
}
```

implicitly convert

can't be directly used

`operator()`

```
#define GLM_FORCE_SWIZZLE
#include <glm/glm.hpp>

using namespace glm;

void foo()
{
  vec4 Color = vec4(1.0f, 0.5f, 0.0f, 1.0f);

  // Generates compiler errors. Color.rgba is not a vector type.
  vec4 ClampedA = clamp(Color.rgba, 0.f, 1.f); // ERROR

  // Explicit conversion through a constructor
  vec4 ClampedB = clamp(vec4(Color.rgba), 0.f, 1.f); // OK

  // Explicit conversion through operator()
  vec4 ClampedC = clamp(Color.rgba(), 0.f, 1.f); // OK
}
```

Note: The implementation has a caveat: Swizzle operator types must be different on both size of the equal operator or the operation will fail. There is no known fix for this issue to date

`GLM_FORCE_XYZW_ONLY`

`GLM_FORCE_LEFT_HANDED`

`GLM_FORCE_DEPTH_ZERO_TO_ONE`

```cpp
#include <glm/glm.hpp>

void foo(vec4 const& v)
{
    int Length = v.length();
    ...
}
```

`int` `size_t`
`GLM_FORCE_SIZE_T_LENGTH` `length()`
`size_t`

`glm::length_t` `length()`
`GLM_FORCE_SIZE_T_LENGTH`

```cpp
#define GLM_FORCE_SIZE_T_LENGTH
#include <glm/glm.hpp>

void foo(vec4 const& v)
{
    glm::length_t Length = v.length();
    ...
}
```

`GLM_FORCE_UNRESTRICTED_GENTYPE`

```cpp
#include <glm/glm.hpp>

float average(float const A, float const B)
{
    return glm::mix(A, B, 0.5f); // By default glm::mix only supports floating-point types
}
```

```
#define GLM_FORCE_UNRESTRICTED_GENTYPE
#include <glm/glm.hpp>

int average(int const A, int const B)
{
    return glm::mix(A, B, 0.5f); // integers are ok thanks to
GLM_FORCE_UNRESTRICTED_GENTYPE
}
```

3.1.1. GLM_EXT_scalar_int_sized

<glm/ext/scalar_int_sized.hpp>

3.1.2. GLM_EXT_scalar_uint_sized

```
#include <glm/ext/scalar_common.hpp>

glm::uint64 pack(glm::uint32 A, glm::uint16 B, glm::uint8 C, glm::uint8 D)
{
	glm::uint64 ShiftA = 0;
	glm::uint64 ShiftB = sizeof(glm::uint32) * 8;
	glm::uint64 ShiftC = (sizeof(glm::uint32) + sizeof(glm::uint16)) * 8;
	glm::uint64 ShiftD = (sizeof(glm::uint32) + sizeof(glm::uint16) +
sizeof(glm::uint8)) * 8;
	return (glm::uint64(A) << ShiftA) | (glm::uint64(B) << ShiftB) |
(glm::uint64(C) << ShiftC) | (glm::uint64(D) << ShiftD);
}
```

<glm/ext/scalar_uint_sized.hpp>

3.2.1. GLM_EXT_scalar_common

min max

fmin fmax NaN

```
#include <glm/ext/scalar_common.hpp>

float positiveMax(float const a, float const b)
{
    return glm::fmax(a, b, 0.0f);
}
```

<glm/ext/scalar_common.hpp>

3.2.2. GLM_EXT_scalar_relational

equal notEqual

```
#include <glm/ext/scalar_relational.hpp>

bool epsilonEqual(float const a, float const b)
{
    float const CustomEpsilon = 0.0001f;
    return glm::equal(a, b, CustomEpsilon);
}
```

<glm/ext/scalar_relational.hpp>

3.2.3. GLM_EXT_scalar_constants

epsilon pi

```
#include <glm/ext/scalar_constants.hpp>

float circumference(float const Diameter)
{
    return glm::pi<float>() * Diameter;
}
```

```
#include <glm/common.hpp> // abs
#include <glm/ext/scalar_constants.hpp> // epsilon

bool equalULP1(float const a, float const b)
{
    return glm::abs(a - b) <= glm::epsilon<float>();
}
```

<glm/ext/scalar_constants.hpp>

3.3.1. GLM_EXT_vector_float1

vec1

<glm/ext/vector_float1.hpp>

3.3.2. GLM_EXT_vector_float2

vec2

<glm/ext/vector_float2.hpp>

3.3.3. GLM_EXT_vector_float3

`vec3`

`<glm/ext/vector_float3.hpp>`

3.3.4. GLM_EXT_vector_float4

`vec4`

`<glm/ext/vector_float4.hpp>`

3.3.5. GLM_EXT_vector_double1

`dvec1`

`<glm/ext/vector_double1.hpp>`

3.3.6. GLM_EXT_vector_double2

`dvec2`

`<glm/ext/vector_double2.hpp>`

3.3.7. GLM_EXT_vector_double3

`dvec3`

`<glm/ext/vector_double3.hpp>`

3.3.8. GLM_EXT_vector_double4

`dvec4`

`<glm/ext/vector_double4.hpp>`

3.3.9. GLM_EXT_vector_int1

`ivec1`

`<glm/ext/vector_int1.hpp>`

3.3.10. GLM_EXT_vector_int2

`ivec2`

`<glm/ext/vector_int2.hpp>`

3.3.11. GLM_EXT_vector_int3

`ivec3`

`<glm/ext/vector_int3.hpp>`

3.3.12. GLM_EXT_vector_int4

`ivec4`

`<glm/ext/vector_int4.hpp>`

3.3.13. GLM_EXT_vector_int1

`uvec1`

`<glm/ext/vector_uint1.hpp>`

3.3.14. GLM_EXT_vector_uint2

`uvec2`

`<glm/ext/vector_uint2.hpp>`

3.3.15. GLM_EXT_vector_uint3

`uvec3`

`<glm/ext/vector_uint3.hpp>`

3.3.16. GLM_EXT_vector_uint4

`uvec4`

`<glm/ext/vector_uint4.hpp>`

3.3.17. GLM_EXT_vector_bool1

`bvec1`

`<glm/ext/vector_bool1.hpp>`

3.3.18. GLM_EXT_vector_bool2

`bvec2`

`<glm/ext/vector_bool2.hpp>`

3.3.19. GLM_EXT_vector_bool3

`bvec3`

`<glm/ext/vector_bool3.hpp>`

3.3.20. GLM_EXT_vector_bool4

`bvec4`

`<glm/ext/vector_bool4.hpp>`

3.4.1. GLM_EXT_vector_float1_precision

`lowp_vec1` `mediump_vec1` `highp_vec1`

`<glm/ext/vector_float1_precision.hpp>`

3.4.2. GLM_EXT_vector_float2_precision

`lowp_vec2` `mediump_vec2` `highp_vec2`

`<glm/ext/vector_float2_precision.hpp>`

3.4.3. GLM_EXT_vector_float3_precision

`lowp_vec3` `mediump_vec3` `highp_vec3`

`<glm/ext/vector_float3_precision.hpp>`

3.4.4. GLM_EXT_vector_float4_precision

`lowp_vec4` `mediump_vec4` `highp_vec4`

`<glm/ext/vector_float4_precision.hpp>`

3.4.5. GLM_EXT_vector_double1_precision

`lowp_dvec1` `mediump_dvec1` `highp_dvec1`

`<glm/ext/vector_double1_precision.hpp>`

3.4.6. GLM_EXT_vector_double2_precision

`lowp_dvec2` `mediump_dvec2` `highp_dvec2`

`<glm/ext/vector_double2_precision.hpp>`

3.4.7. GLM_EXT_vector_double3_precision

`lowp_dvec3` `mediump_dvec3` `highp_dvec3`

`<glm/ext/vector_double3_precision.hpp>`

3.4.8. GLM_EXT_vector_double4_precision

`lowp_dvec4` `mediump_dvec4` `highp_dvec4`

`<glm/ext/vector_double4_precision.hpp>`

3.5.1. GLM_EXT_vector_common

`min` `max`

`fmin` `fmax` `NaN`

```
#include <glm/ext/vector_float2.hpp> // vec2
#include <glm/ext/vector_common.hpp> // fmax

float positiveMax(float const a, float const b)
{
    return glm::fmax(a, b, 0.0f);
}
```

`<glm/ext/vector_common.hpp>`

3.5.2. GLM_EXT_vector_relational

`equal` `notEqual`

```
#include <glm/ext/vector_float2.hpp> // vec2
#include <glm/ext/vector_relational.hpp> // equal, all

bool epsilonEqual(glm::vec2 const& A, glm::vec2 const& B)
{
    float const CustomEpsilon = 0.0001f;
    return glm::all(glm::equal(A, B, CustomEpsilon));
}
```

`<glm/ext/vector_relational.hpp>`

3.6.1. GLM_EXT_matrix_float2x2

mat2x2

<glm/ext/matrix_float2x2.hpp>

3.6.2. GLM_EXT_matrix_float2x3

mat2x3

<glm/ext/matrix_float2x3.hpp>

3.6.3. GLM_EXT_matrix_float2x4

mat2x4

<glm/ext/matrix_float2x4.hpp>

3.6.4. GLM_EXT_matrix_float3x2

mat3x2

<glm/ext/matrix_float3x2.hpp>

3.6.5. GLM_EXT_matrix_float3x3

mat3x3

<glm/ext/matrix_float3x3.hpp>

3.6.6. GLM_EXT_matrix_float3x4

mat3x4

<glm/ext/matrix_float3x4.hpp>

3.6.7. GLM_EXT_matrix_float4x2

mat4x2

<glm/ext/matrix_float4x2.hpp>

3.6.8. GLM_EXT_matrix_float4x3

mat4x3

<glm/ext/matrix_float4x3.hpp>

3.6.9. GLM_EXT_matrix_float4x4

mat4x4

<glm/ext/matrix_float4x4.hpp>

3.6.10. GLM_EXT_matrix_double2x2

`dmat2x2`

`<glm/ext/matrix_double2x2.hpp>`

3.6.11. GLM_EXT_matrix_double2x3

`dmat2x3`

`<glm/ext/matrix_double2x3.hpp>`

3.6.12. GLM_EXT_matrix_double2x4

`dmat2x4`

`<glm/ext/matrix_double2x4.hpp>`

3.6.13. GLM_EXT_matrix_double3x2

`dmat3x2`

`<glm/ext/matrix_double3x2.hpp>`

3.6.14. GLM_EXT_matrix_double3x3

`dmat3x3`

`<glm/ext/matrix_double3x3.hpp>`

3.6.15. GLM_EXT_matrix_double3x4

`dmat3x4`

`<glm/ext/matrix_double3x4.hpp>`

3.6.16. GLM_EXT_matrix_double4x2

`dmat4x2`

`<glm/ext/matrix_double4x2.hpp>`

3.6.17. GLM_EXT_matrix_double4x3

`dmat4x3`

`<glm/ext/matrix_double4x3.hpp>`

3.6.18. GLM_EXT_matrix_double4x4

`dmat4x4`

`<glm/ext/matrix_double4x4.hpp>`

3.7.1. GLM_EXT_matrix_float2x2_precision

`lowp_mat2x2` `mediump_mat2x2` `highp_mat2x2`

`<glm/ext/matrix_float2x2_precision.hpp>`

3.7.2. GLM_EXT_matrix_float2x3_precision

`lowp_mat2x3` `mediump_mat2x3` `highp_mat2x3`

`<glm/ext/matrix_float2x3_precision.hpp>`

3.7.3. GLM_EXT_matrix_float2x4_precision

`lowp_mat2x4` `mediump_mat2x4` `highp_mat2x4`

`<glm/ext/matrix_float2x4_precision.hpp>`

3.7.4. GLM_EXT_matrix_float3x2_precision

`lowp_mat3x2` `mediump_mat3x2` `highp_mat3x2`

`<glm/ext/matrix_float3x2_precision.hpp>`

3.7.5. GLM_EXT_matrix_float3x3_precision

`lowp_mat3x3` `mediump_mat3x3` `highp_mat3x3`

`<glm/ext/matrix_float3x3_precision.hpp>`

3.7.6. GLM_EXT_matrix_float3x4_precision

`lowp_mat3x4` `mediump_mat3x4` `highp_mat3x4`

`<glm/ext/matrix_float3x4_precision.hpp>`

3.7.7. GLM_EXT_matrix_float4x2_precision

`lowp_mat4x2` `mediump_mat4x2` `highp_mat4x2`

`<glm/ext/matrix_float4x2_precision.hpp>`

3.7.8. GLM_EXT_matrix_float4x3_precision

`lowp_mat4x3`	`mediump_mat4x3`	`highp_mat4x3`

`<glm/ext/matrix_float4x3_precision.hpp>`

3.7.9. GLM_EXT_matrix_float4x4_precision

`lowp_mat4x4`	`mediump_mat4x4`	`highp_mat4x4`

`<glm/ext/matrix_float4x4_precision.hpp>`

3.7.10. GLM_EXT_matrix_double2x2_precision

`lowp_dmat2x2`	`mediump_dmat2x2`	`highp_dmat2x2`

`<glm/ext/matrix_double2x2_precision.hpp>`

3.7.11. GLM_EXT_matrix_double2x3_precision

`lowp_dmat2x3`	`mediump_dmat2x3`	`highp_dmat2x3`

`<glm/ext/matrix_double2x3_precision.hpp>`

3.7.12. GLM_EXT_matrix_double2x4_precision

`lowp_dmat2x4`	`mediump_dmat2x4`	`highp_dmat2x4`

`<glm/ext/matrix_double2x4_precision.hpp>`

3.7.13. GLM_EXT_matrix_double3x2_precision

`lowp_dmat3x2`	`mediump_dmat3x2`	`highp_dmat3x2`

`<glm/ext/matrix_double3x2_precision.hpp>`

3.7.14. GLM_EXT_matrix_double3x3_precision

`lowp_dmat3x3`	`mediump_dmat3x3`	`highp_dmat3x3`

`<glm/ext/matrix_double3x3_precision.hpp>`

3.7.15. GLM_EXT_matrix_double3x4_precision

lowp_dmat3x4 mediump_dmat3x4 highp_dmat3x4

<glm/ext/matrix_double3x4_precision.hpp>

3.7.16. GLM_EXT_matrix_double4x2_precision

lowp_dmat4x2 mediump_dmat4x2 highp_dmat4x2

<glm/ext/matrix_double4x2_precision.hpp>

3.7.17. GLM_EXT_matrix_double4x3_precision

lowp_dmat4x3 mediump_dmat4x3 highp_dmat4x3

<glm/ext/matrix_double4x3_precision.hpp>

3.7.18. GLM_EXT_matrix_double4x4_precision

lowp_dmat4x4 mediump_dmat4x4 highp_dmat4x4

<glm/ext/matrix_double4x4_precision.hpp>

3.8.1. GLM_EXT_matrix_relational

equal notEqual

```
#include <glm/ext/vector_bool4.hpp> // bvec4
#include <glm/ext/matrix_float4x4.hpp> // mat4
#include <glm/ext/matrix_relational.hpp> // equal, all

bool epsilonEqual(glm::mat4 const& A, glm::mat4 const& B)
{
    float const CustomEpsilon = 0.0001f;
    glm::bvec4 const ColumnEqual = glm::equal(A, B, CustomEpsilon); // Evaluation
per column
    return glm::all(ColumnEqual);
}
```

<glm/ext/matrix_relational.hpp>

3.8.2. GLM_EXT_matrix_transform

translate rotate scale

```
#include <glm/ext/vector_float2.hpp> // vec2
#include <glm/ext/vector_float3.hpp> // vec3
#include <glm/ext/matrix_float4x4.hpp> // mat4x4
#include <glm/ext/matrix_transform.hpp> // translate, rotate, scale, identity

glm::mat4 computeModelViewMatrix(float Translate, glm::vec2 const & Rotate)
{
        glm::mat4 View = glm::translate(glm::identity(), glm::vec3(0.0f, 0.0f, -
Translate));
        View = glm::rotate(View, Rotate.y, glm::vec3(-1.0f, 0.0f, 0.0f));
        View = glm::rotate(View, Rotate.x, glm::vec3(0.0f, 1.0f, 0.0f));
        glm::mat4 Model = glm::scale(glm::identity(), glm::vec3(0.5f));
        return View * Model;
}
```

<glm/ext/matrix_transform.hpp>

3.8.3. GLM_EXT_matrix_clip_space

```
#include <glm/ext/matrix_float4x4.hpp> // mat4x4
#include <glm/ext/matrix_clip_space.hpp> // perspective
#include <glm/trigonometric.hpp> // radians

glm::mat4 computeProjection(float Width, float Height)
{
        return glm::perspective(glm::radians(45.0f), Width / Height, 0.1f, 100.f);
}
```

<glm/ext/matrix_clip_space.hpp>

3.8.4. GLM_EXT_matrix_projection

<glm/ext/matrix_projection.hpp>

3.9.1. GLM_EXT_quaternion_float

quat

<glm/ext/quaternion_float.hpp>

3.9.2. GLM_EXT_quaternion_double

`dquat`

`<glm/ext/quaternion_double.hpp>`

3.10.1. GLM_EXT_quaternion_float_precision

`lowp_quat` `mediump_quat` `highp_quat`

`<glm/ext/quaternion_float_precision.hpp>`

3.10.2. GLM_EXT_quaternion_double_precision

`lowp_dquat` `mediump_dquat` `highp_dquat`

`<glm/ext/quaternion_double_precision.hpp>`

3.11.1. GLM_EXT_quaternion_common

`slerp` `conjugate` `inverse`

`<glm/ext/quaternion_common.hpp>`

3.11.2. GLM_EXT_quaternion_geometric

`length` `normalize` `dot` `cross`

`<glm/ext/quaternion_geometric.hpp>`

3.11.3. GLM_EXT_quaternion_trigonometric

`angle` `axis`

`<glm/ext/quaternion_trigonometric.hpp>`

3.11.4. GLM_EXT_quaternion_exponential

`exp` `log` `pow` `sqrt`

`<glm/ext/quaternion_exponential.hpp>`

3.11.5. GLM_EXT_quaternion_relational

```
<glm/ext/quaternion_relational.hpp>
```

3.11.6. GLM_EXT_quaternion_transform

```
<glm/ext/quaternion_transform.hpp>
```

```cpp
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>

int foo()
{
    glm::vec4 Position = glm::vec4(glm:: vec3(0.0f), 1.0f);
    glm::mat4 Model = glm::translate(glm::mat4(1.0f), glm::vec3(1.0f));

    glm::vec4 Transformed = Model * Position;
    ...

    return 0;
}
```

`<glm/gtc/bitfield.hpp>`

`<glm/gtc/color_space.hpp>`

`<glm/gtc/constants.hpp>`

`<glm/gtc/epsilon.hpp>`

`<glm/gtc/integer.hpp>`

`<glm/gtc/matrix_access.hpp>`

`<glm/gtc/matrix_integer.hpp>`

`<glm/gtc/matrix_inverse.hpp>`

`lookAt`

`perspective` `ortho`

`lookAt`

`perspective` `ortho`

`<glm/gtc/matrix_transform.hpp>`

`glm/gtc/noise.hpp>`

`<glm/gtc/packing.hpp>`

`<glm/gtc/quaternion.hpp>`

`<glm/gtc/random.hpp>`

`<glm/gtc/reciprocal.hpp>`

`<glm/gtc/round.hpp>`

`<glm/gtc/type_aligned.hpp>`

`i8vec4`

`<glm/gtc/type\_precision.hpp>`

`float*` `mat4`

`glm::value_ptr`

`vec3` `mat4`

```
// GLM_GTC_type_ptr provides a safe solution:
#include <glm/glm.hpp>
#include <glm/gtc/type_ptr.hpp>

void foo()
{
    glm::vec4 v(0.0f);
    glm::mat4 m(1.0f);
    ...
    glVertex3fv(glm::value_ptr(v))
    glLoadMatrixfv(glm::value_ptr(m));
}

// Another solution, this one inspired by the STL:
#include <glm/glm.hpp>

void foo()
{
    glm::vec4 v(0.0f);
    glm::mat4 m(1.0f);
    ...
    glVertex3fv(&v[0]);
    glLoadMatrixfv(&m[0][0]);
}
```

`glVertex3fv`

`<glm/gtc/type_ptr.hpp>`

`<glm/gtc/ulp.hpp>`

`<glm/gtc/vec1.hpp>`

---

glRotate{f, d}:

```
glm::mat4 glm::rotate(glm::mat4 const& m, float angle, glm::vec3 const& axis);
glm::dmat4 glm::rotate(glm::dmat4 const& m, double angle, glm::dvec3 const& axis);
```

`GLM_GTC_matrix_transform`

glScale{f, d}:

```
glm::mat4 glm::scale(glm::mat4 const& m, glm::vec3 const& factors);
glm::dmat4 glm::scale(glm::dmat4 const& m, glm::dvec3 const& factors);
```

`GLM_GTC_matrix_transform`

glTranslate{f, d}:

```
glm::mat4 glm::translate(glm::mat4 const& m, glm::vec3 const& translation);
glm::dmat4 glm::translate(glm::dmat4 const& m, glm::dvec3 const& translation);
```

`GLM_GTC_matrix_transform`

glLoadIdentity:

```
glm::mat4(1.0) or glm::mat4();
glm::dmat4(1.0) or glm::dmat4();
```

`<glm/glm.hpp>`

glMultMatrix{f, d}:

```
glm::mat4() * glm::mat4();
glm::dmat4() * glm::dmat4();
```

`<glm/glm.hpp>`

glLoadTransposeMatrix{f, d}:

```
glm::transpose(glm::mat4());
glm::transpose(glm::dmat4());
```

`<glm/glm.hpp>`

glMultTransposeMatrix{f, d}:

```
glm::mat4() * glm::transpose(glm::mat4());
glm::dmat4() * glm::transpose(glm::dmat4());
```

`<glm/glm.hpp>`

glFrustum:

```
glm::mat4 glm::frustum(float left, float right, float bottom, float top, float
zNear, float zFar);
glm::dmat4 glm::frustum(double left, double right, double bottom, double top,
double zNear, double zFar);
```

`GLM_GTC_matrix_transform` `<glm/gtc/matrix_transform.hpp>`

glOrtho:

```
glm::mat4 glm::ortho(float left, float right, float bottom, float top, float
zNear, float zFar);
glm::dmat4 glm::ortho(double left, double right, double bottom, double top, double
zNear, double zFar);
```

`GLM_GTC_matrix_transform` `<glm/gtc/matrix_transform.hpp>`

gluLookAt:

```
glm::mat4 glm::lookAt(glm::vec3 const& eye, glm::vec3 const& center, glm::vec3
const& up);
glm::dmat4 glm::lookAt(glm::dvec3 const& eye, glm::dvec3 const& center, glm::dvec3
const& up);
```

`GLM_GTC_matrix_transform` `<glm/gtc/matrix_transform.hpp>`

gluOrtho2D:

```
glm::mat4 glm::ortho(float left, float right, float bottom, float top);
glm::dmat4 glm::ortho(double left, double right, double bottom, double top);
```

`GLM_GTC_matrix_transform`   `<glm/gtc/matrix_transform.hpp>`

gluPerspective:

```
glm::mat4 perspective(float fovy, float aspect, float zNear, float zFar);
glm::dmat4 perspective(double fovy, double aspect, double zNear, double zFar);
```

`GLM_GTC_matrix_transform`   `<glm/gtc/matrix_transform.hpp>`

gluPickMatrix:

```
glm::mat4 pickMatrix(glm::vec2 const& center, glm::vec2 const& delta, glm::ivec4
const& viewport);
glm::dmat4 pickMatrix(glm::dvec2 const& center, glm::dvec2 const& delta,
glm::ivec4 const& viewport);
```

`GLM_GTC_matrix_transform`   `<glm/gtc/matrix_transform.hpp>`

gluProject:

```
glm::vec3 project(glm::vec3 const& obj, glm::mat4 const& model, glm::mat4 const&
proj, glm::ivec4 const& viewport);
glm::dvec3 project(glm::dvec3 const& obj, glm::dmat4 const& model, glm::dmat4
const& proj, glm::ivec4 const& viewport);
```

`GLM_GTC_matrix_transform`   `<glm/gtc/matrix_transform.hpp>`

gluUnProject:

```
glm::vec3 unProject(glm::vec3 const& win, glm::mat4 const& model, glm::mat4 const&
proj, glm::ivec4 const& viewport);
glm::dvec3 unProject(glm::dvec3 const& win, glm::dmat4 const& model, glm::dmat4
const& proj, glm::ivec4 const& viewport);
```

`GLM_GTC_matrix_transform`   `<glm/gtc/matrix_transform.hpp>`

not_

```
// Using precision qualifier in GLSL:

ivec3 foo(in vec4 v)
{
    highp vec4 a = v;
    mediump vec4 b = a;
    lowp ivec3 c = ivec3(b);
    return c;
}

// Using precision qualifier in GLM:

#include <glm/glm.hpp>

ivec3 foo(const vec4 & v)
{
    highp_vec4 a = v;
    medium_vec4 b = a;
    lowp_ivec3 c = glm::ivec3(b);
    return c;
}
```

API documentation

`/W4`

`/Za`

`define NOMINMAX`

`GLM_FORCE_PURE`

```
#include <glm/glm.hpp> // vec3 normalize cross

glm::vec3 computeNormal(glm::vec3 const& a, glm::vec3 const& b, glm::vec3 const&
c)
{
    return glm::normalize(glm::cross(c - a, b - a));
}

// A much faster but less accurate alternative:
#include <glm/glm.hpp> // vec3 cross
#include <glm/gtx/fast_square_root.hpp> // fastNormalize

glm::vec3 computeNormal(glm::vec3 const& a, glm::vec3 const& b, glm::vec3 const&
c)
{
    return glm::fastNormalize(glm::cross(c - a, b - a));
}
```

```
#include <glm/glm.hpp> // vec3, vec4, ivec4, mat4
#include <glm/gtc/matrix_transform.hpp> // translate, rotate, scale, perspective
#include <glm/gtc/type_ptr.hpp> // value_ptr

void setUniformMVP(GLuint Location, glm::vec3 const& Translate, glm::vec3 const&
Rotate)
{
    glm::mat4 Projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.f);
    glm::mat4 ViewTranslate = glm::translate(
        glm::mat4(1.0f), Translate);
    glm::mat4 ViewRotateX = glm::rotate(
        ViewTranslate, Rotate.y, glm::vec3(-1.0f, 0.0f, 0.0f));
    glm::mat4 View = glm::rotate(ViewRotateX,
        Rotate.x, glm::vec3(0.0f, 1.0f, 0.0f));
    glm::mat4 Model = glm::scale(
        glm::mat4(1.0f), glm::vec3(0.5f));
    glm::mat4 MVP = Projection * View * Model;
    glUniformMatrix4fv(Location, 1, GL_FALSE, glm::value_ptr(MVP));
}
```

```
#include <glm/glm.hpp> // vec2
#include <glm/gtc/type_precision.hpp> // hvec2, i8vec2, i32vec2

std::size_t const VertexCount = 4;

// Float quad geometry
std::size_t const PositionSizeF32 = VertexCount * sizeof(glm::vec2);
glm::vec2 const PositionDataF32[VertexCount] =
{
    glm::vec2(-1.0f,-1.0f),
    glm::vec2( 1.0f,-1.0f),
    glm::vec2( 1.0f, 1.0f),
    glm::vec2(-1.0f, 1.0f)
};

// Half-float quad geometry
std::size_t const PositionSizeF16 = VertexCount * sizeof(glm::hvec2);
glm::hvec2 const PositionDataF16[VertexCount] =
{
    glm::hvec2(-1.0f, -1.0f),
    glm::hvec2( 1.0f, -1.0f),
    glm::hvec2( 1.0f, 1.0f),
    glm::hvec2(-1.0f, 1.0f)
};

// 8 bits signed integer quad geometry
std::size_t const PositionSizeI8 = VertexCount * sizeof(glm::i8vec2);
glm::i8vec2 const PositionDataI8[VertexCount] =
{
    glm::i8vec2(-1,-1),
    glm::i8vec2( 1,-1),
    glm::i8vec2( 1, 1),
    glm::i8vec2(-1, 1)
};

// 32 bits signed integer quad geometry
std::size_t const PositionSizeI32 = VertexCount * sizeof(glm::i32vec2);
glm::i32vec2 const PositionDataI32[VertexCount] =
{
    glm::i32vec2(-1,-1),
    glm::i32vec2( 1,-1),
    glm::i32vec2( 1, 1),
    glm::i32vec2(-1, 1)
};
```

```
#include <glm/glm.hpp> // vec3 normalize reflect dot pow
#include <glm/gtc/random.hpp> // ballRand
```

```
// vecRand3, generate a random and equiprobable normalized vec3
glm::vec3 lighting(intersection const& Intersection, material const& Material,
light const& Light, glm::vec3 const& View)
{
    glm::vec3 Color = glm::vec3(0.0f);
    glm::vec3 LightVertor = glm::normalize(
        Light.position() - Intersection.globalPosition() +
        glm::ballRand(0.0f, Light.inaccuracy());

    if(!shadow(Intersection.globalPosition(), Light.position(), LightVertor))
    {
        float Diffuse = glm::dot(Intersection.normal(), LightVector);
        if(Diffuse &lt;= 0.0f)
            return Color;

        if(Material.isDiffuse())
            Color += Light.color() * Material.diffuse() * Diffuse;

        if(Material.isSpecular())
        {
            glm::vec3 Reflect = glm::reflect(-LightVector, Intersection.normal());
            float Dot = glm::dot(Reflect, View);
            float Base = Dot &gt; 0.0f ? Dot : 0.0f;
            float Specular = glm::pow(Base, Material.exponent());
            Color += Material.specular() \* Specular;
        }
    }

    return Color;
}
```

`GLM_FORCE_MESSAGES`

```
#define GLM_FORCE_MESSAGES
#include <glm/glm.hpp>
```

```
GLM: 0.9.9.1
GLM: C++ 17 with extensions
GLM: GCC compiler detected"
GLM: x86 64 bits with AVX instruction set build target
GLM: Linux platform detected
GLM: GLM_FORCE_SWIZZLE is undefined. swizzling functions or operators are
disabled.
GLM: GLM_FORCE_SIZE_T_LENGTH is undefined. .length() returns a glm::length_t, a
typedef of int following GLSL.
GLM: GLM_FORCE_UNRESTRICTED_GENTYPE is undefined. Follows strictly GLSL on valid
function genTypes.
GLM: GLM_FORCE_DEPTH_ZERO_TO_ONE is undefined. Using negative one to one depth
clip space.
GLM: GLM_FORCE_LEFT_HANDED is undefined. Using right handed coordinate system.
```

Step 1: Setup our GLM Fork

```
>>> git clone <our-repository-fork-git-url>
```

`https://github.com/<our-username>/<repository-name>.git`

Step 2: Synchronizing our fork

`upstream`

```
>>> git remote add upstream https://github.com/processing/processing.git
```

```
>>> git fetch upstream
```

```
>>> git merge upstream/master
```

```
>>> git push origin master
```

Step 3: Modifying our GLM Fork

```
>>> git checkout master
```

bugifx

```
git branch bugfix
```

```
>>> git commit -m "Resolve the issue that caused problem with a specific fix #432"
```

```
>>> git push origin bugfix
```

-
-
- glm/test
- YRL QWHV IERU% SQP($¿!O Y• @Q!U• À 0 À p € 0 ` ° XX9` W H H ¿!O P€B qU• À P 0 @ p

Indentation

Spacing

```
if(blah)
{
    // yes like this
}
else
{
    // something besides
}
```

```
if(blah)
    // yes like this
else
    // something besides
```

```
if (blah)    // No
if( blah )   // No
if ( blah )  // No
if(blah)     // Yes
```

```
someFunction(apple,bear,cat);     // No
someFunction(apple, bear, cat);   // Yes
```

`+, -, *, /, %, >>, <<, |, &, ^, ||, &&`

```
vec4 v = a + b;
```

Blank lines

Comments

Cases

```
#define GLM_MY_DEFINE 76

class myClass
{};

myClass const MyClass;

namespace glm{ // glm namespace is for public code
namespace detail // glm::detail namespace is for implementation detail
{
    float myFunction(vec2 const& V)
    {
        return V.x + V.y;
    }

    float myFunction(vec2 const* const V)
    {
        return V->x + V->y;
    }
}//namespace detail
}//namespace glm
```

-
- *GLU 1.3 specification*

-

Leo's Fortune

OpenGL 4.0 Shading Language Cookbook

-
-
-

Outerra

Falcor

Cinder

opencloth

LibreOffice

Are you using GLM in a project?

-
-
- *The OpenGL Samples Pack*
- *Learning Modern 3D Graphics programming*
- *Morten Nobel-Jørgensen's* *OpenGL renderer*
- *Swiftless' OpenGL tutorial*
- *Rastergrid*
- *OpenGL Tutorial*

- *OpenGL Programming on Wikibooks*
- *3D Game Engine Programming*
-
-
-
-
- ***Are you using GLM in a tutorial?***

- *cglm*
- *GlmSharp*
-
-
-
-
-
-
-
-

- *CML*
- *Eigen*
- *glhlib*
-

Christophe Riccio

- *webgl-noise*

- *Arthur Winters*
-

- *nightlight build system*
-
-
- *Grant James*
-
-