MATH BASE

----------------------------------------------------------------------

**Список констант:**

PI = 3.14159265358979323846

TWO\_PI = PI \* 2.0

HALF\_PI = PI / 2.0

INV\_PI = 1.0 / PI

E = 2.71828182845904523536

SQRT\_TWO = 1.41421356237309504880

SQRT\_THREE = 1.73205080756887729352

SQRT\_HALF = 0.70710678118654752440

FLOAT\_INFINITY = 1e30

FLOAT\_EPSILON = 1.192092896e-07

FLOAT\_EPSILON **это** **минимальное число, для которого выполняется условие** 1.0f + FLOAT\_EPSILON != 1.0f.

**Базовые типы данных:**

mbool\_t – **тип данных определённый для математических логических операций. Имеет два возвращаемых значения**: mtrue (1) **и** mfalse (0). **Другие значения должны интерпретироваться** - 0 **как** mfalse, **и не** 0 **как** mtrue.

**Список базовых математических функций:**

**Функция** isqrt1f **возвращает обратный квадратный корень числа** x**. Если** x == 0.0 **то возвращаемое значение будет очень большим.**

float isqrt1f( float x );

**Функция** sqrt1f **возвращает квадратный корень числа** x**.**

float sqrt1f( float x );

**Функция** sin1f **возвращает синус угла** a**. Угол** a **задаётся в радианах. Возвращаемое значение будет в пределах** [-1, +1].

float sin1f( float a );

**Функция** cos1f **возвращает косинус угла** a**. Угол** a **задаётся в радианах. Возвращаемое значение будет в пределах** [-1, +1].

float cos1f( float a );

**Функция** sincosf **возвращает синус и косинус угла** a**. Угол** a **задаётся в радианах. Синус будет записан по указателю** s**, а косинус будет записан по указателю** с**.** s **и** c **не должны быть NULL.**

void sincosf( float a, float\* s, float\* c );

**Функция** tan1f **возвращает тангенс угла** a. **Угол** a **задаётся в радианах.**

float tan1f( float a );

**Функция** asin1f **возвращает арксинус синуса** a. a **должно быть в пределах** [-1, +1]. **Возвращаемое значение будет в пределах** [-PI/2, +PI/2].

float asin1f( float a );

**Функция** acos1f **возвращает арккосинус косинуса** a. a **должно быть в пределах** [-1, +1]. **Возвращаемое значение будет в пределах** [0, PI].

float acos1f( float a );

**Функция** atan1f **возвращает арктангенс тангенса** a. **Возвращаемое значение будет в пределах** [-PI/2, +PI/2].

float atan1f( float a );

**Функция** atan2f **возвращает арктангенс тангенса по координатам** x **и** y **в плоскости**. **Возвращаемое значение будет в пределах** [-PI/2, +PI/2].

float atan2f( float y, float x );

**Функция** pow2f **возвращает** x **возведённую в степень** y.

float pow2f( float x, float y );

**Функция** exp1f **возвращает** **экспоненту числа** f. **Экспонента это число** E **возведённое в степень** f.

float exp1f( float f );

**Функция** log1f **возвращает натуральный логарифм** от f.

float log1f( float f );

**Функция** pow2i **возвращает** x **возведённую в степень** y.

int pow2i( int x, int y );

**Функция** min2f **и** min2i **возвращает минимальное число из двух чисел** a **и** b.

float min2f( float a, float b );

int min2i( int a, int b );

**Функция** max2f **и** max2i **возвращает максимальное число из двух чисел** a **и** b.

float max2f( float a, float b );

int max2i( int a, int b );

**Функция** abs1i **и** abs1f **возвращает абсолютное значение числа** x.

int abs1i( int x );

float abs1f( float f );

**Функция** floor1f **возвращает целое число, которое меньше или равно числу** f. (**округление к меньшему целому**).

float floor1f( float f );

**Функция** ceil1f **возвращает целое число, которое больше или равно числу** f. (**округление к большему целому**).

float ceil1f( float f );

**Функция** round1f **возвращает целое число, округленное по математическим законам**.

float round1f( float f );

**Функция** trunc1f **отбрасывает дробную часть числа** f **и возвращает целое значение**.

float trunc1f( float f );

**Функция** frac1f **возвращает** f - floor1f( f ).

float frac1f( float f );

**Функция** clamp3i **и** clamp3f **возвращает число** min **если** val **меньше чем** min. **Возвращает число max, если** val **больше чем** max. **Иначе функция возвращает** val.

int clamp3i( int min, int max, int val );

float clamp3f( float min, float max, float val );

**Функция** lerpi **и** lerpf **возвращает линейную между двумя числами** a **и** b **с коэффициентом** scale.

int lerpi( int a, int b, float scale );

float lerpf( float a, float b, float scale );

VECTORS

----------------------------------------------------------------------

**Основные типы данных:**

vec2\_t – **двухмерный вектор (8 байт)**

vec3\_t – **трёхмерный вектор (12 байт)**

vec4\_t – **четырёхмерный вектор (16 байт)**

**Список функций:**

**Установить значение вектора** v.

void Vec2Set( vec2\_t\* v, float x, float y );

void Vec3Set( vec3\_t\* v, float x, float y, float z );

void Vec4Set( vec4\_t\* v, float x, float y, float z, float w );

**Скопировать вектор** v **в** out.

void Vec2Cpy( vec2\_t\* out, const vec2\_t\* v );

void Vec3Cpy( vec3\_t\* out, const vec3\_t\* v );

void Vec4Cpy( vec4\_t\* out, const vec4\_t\* v );

**Установить вектор в** 0.

void Vec2Zero( vec2\_t\* v );

void Vec3Zero( vec3\_t\* v );

void Vec4Zero( vec4\_t\* v );

**Сделать негативным каждое значение вектора** v.

void Vec2Neg( vec2\_t\* v );

void Vec3Neg( vec3\_t\* v );

void Vec4Neg( vec4\_t\* v );

**Сделать обратным каждое значение вектора** v **(обратное значение для n – это 1/n).**

void Vec2Inv( vec2\_t\* v );

void Vec3Inv( vec3\_t\* v );

void Vec4Inv( vec4\_t\* v );

**Отмасштабировать вектор** v. **Каждое значение вектора** v **перемножить на соответствующие значения вектора** s **и записать результат в** v.

void Vec2Scale( vec2\_t\* v, const vec2\_t\* s );

void Vec3Scale( vec3\_t\* v, const vec3\_t\* s );

void Vec4Scale( vec4\_t\* v, const vec4\_t\* s );

**Отмасштабировать вектор** v. **Каждое значение вектора** v **перемножить** f **и записать результат в** v.

void Vec2Scale1f( vec2\_t\* v, float f );

void Vec3Scale1f( vec3\_t\* v, float f );

void Vec4Scale1f( vec4\_t\* v, float f );

**Умножить вектор** v **на матрицу** m. **Результат записать в** out.

void Vec2MulMat2( vec2\_t\*out, const vec2\_t\*v, const mat2\_t\*m );

void Vec3MulMat3( vec3\_t\*out, const vec3\_t\*v, const mat3\_t\*m );

void Vec4MulMat4( vec4\_t\*out, const vec4\_t\*v, const mat4\_t\*m );

**Сложить два вектора** a **и** b, **результат записать в** out.

void Vec2Add( vec2\_t\* out, const vec2\_t\* a, const vec2\_t\* b );

void Vec3Add( vec3\_t\* out, const vec3\_t\* a, const vec3\_t\* b );

void Vec4Add( vec4\_t\* out, const vec4\_t\* a, const vec4\_t\* b );

**Вычесть вектор** b **из вектора** a, **результат записать в** out.

void Vec2Sub( vec2\_t\* out, const vec2\_t\* a, const vec2\_t\* b );

void Vec3Sub( vec3\_t\* out, const vec3\_t\* a, const vec3\_t\* b );

void Vec4Sub( vec4\_t\* out, const vec4\_t\* a, const vec4\_t\* b );

**Сравнить два вектора** a **и** b. **Если оба вектора равны, то возвращаемое значение будет** mtrue, **если не равны, то возвращаемое значение будет** mfalse.

mbool\_t Vec2Cmp( const vec2\_t\* a, const vec2\_t\* b );

mbool\_t Vec3Cmp( const vec3\_t\* a, const vec3\_t\* b );

mbool\_t Vec4Cmp( const vec4\_t\* a, const vec4\_t\* b );

**Сравнить два вектора** a **и** b **с использованием епсилона** eps. **Если значения обеих векторов лежат в пределах** eps**, то возвращаемое значение будет** mtrue, **иначе возвращаемое значение будет** mfalse.

mbool\_t Vec2CmpEps( const vec2\_t\* a, const vec2\_t\* b, float eps );

mbool\_t Vec3CmpEps( const vec3\_t\* a, const vec3\_t\* b, float eps );

mbool\_t Vec4CmpEps( const vec4\_t\* a, const vec4\_t\* b, float eps );

**Вернуть расстояние от вектора** a **вектора** b.

float Vec2Len( const vec2\_t\* a, const vec2\_t\* b );

float Vec3Len( const vec3\_t\* a, const vec3\_t\* b );

float Vec4Len( const vec4\_t\* a, const vec4\_t\* b );

**Вернуть расстояние от вектора** a **вектора** b **в квадрате**.

float Vec2SqrLen( const vec2\_t\* a, const vec2\_t\* b );

float Vec3SqrLen( const vec3\_t\* a, const vec3\_t\* b );

float Vec4SqrLen( const vec4\_t\* a, const vec4\_t\* b );

**Нормализовать вектор** v **и вернуть размер вектора** v. **Нормализованный вектор записывается в** v.

float Vec2Norm( vec2\_t\* v );

float Vec3Norm( vec3\_t\* v );

float Vec4Norm( vec4\_t\* v );

**Вернуть скалярное произведение векторов** a **и** b.

float Vec2Dot( const vec2\_t\* a, const vec2\_t\* b );

float Vec3Dot( const vec3\_t\* a, const vec3\_t\* b );

float Vec4Dot( const vec4\_t\* a, const vec4\_t\* b );

**Выполнить векторное произведение векторов** a **и** b, **результат записать в** out.

void Vec3Cross( vec3\_t\*out, const vec3\_t\*a, const vec3\_t\*b );

**Вернуть косинус угла между двумя векторами** a **и** b.

float Vec2Cos( const vec2\_t\* a, const vec2\_t\* b );

float Vec3Cos( const vec3\_t\* a, const vec3\_t\* b );

**Вернуть угол в радианах между двумя векторами** a **и** b.

float Vec2Angle( const vec2\_t\* a, const vec2\_t\* b );

float Vec3Angle( const vec3\_t\* a, const vec3\_t\* b );

**«Зажать» значения вектора** v **между минимальными** min **и максимальными** max **значениями. Результат записать в** v.

void Vec2Clamp( vec2\_t\*v,const vec2\_t\*min,const vec2\_t\*max );

void Vec3Clamp( vec3\_t\*v,const vec3\_t\*min,const vec3\_t\*max );

void Vec4Clamp( vec4\_t\*v,const vec4\_t\*min,const vec4\_t\*max );

**Выполнить линейную интерполяцию между двумя векторами** a **и** b **с коэффициентом** scale. **Результат записать в** out.

void Vec2Lerp( vec2\_t\*out, const vec2\_t\*a, const vec2\_t\*b, float s );

void Vec3Lerp( vec3\_t\*out, const vec3\_t\*a, const vec3\_t\*b, float s );

void Vec4Lerp( vec4\_t\*out, const vec4\_t\*a, const vec4\_t\*b, float s );

**Записать в строку** out **значения вектора** v **с количеством знаков после запятой** prec.

void Vec2ToStr( char\* out, const vec2\_t\* v, int prec );

void Vec3ToStr( char\* out, const vec3\_t\* v, int prec );

void Vec4ToStr( char\* out, const vec4\_t\* v, int prec );

**Преобразование из одного типа вектора в другой.**

void Vec2ToVec3( vec3\_t\* out, const vec2\_t\* v );

void Vec2ToVec4( vec4\_t\* out, const vec2\_t\* v );

void Vec3ToVec2( vec2\_t\* out, const vec3\_t\* v );

void Vec3ToVec4( vec4\_t\* out, const vec3\_t\* v );

void Vec4ToVec2( vec2\_t\* out, const vec4\_t\* v );

void Vec4ToVec3( vec3\_t\* out, const vec4\_t\* v );

MATRIX

----------------------------------------------------------------------

**Основные типы данных:**

mat2\_t – **матрица 2 на 2 (16 байт)**

mat3\_t – **матрица 3 на 3 (36 байт)**

mat4\_t – **матрица 4 на 4 (64 байт)**

**Список функций:**

**Установить значение матрицы** m **из значений векторов.**

void Mat2Set( mat2\_t\* m, const vec2\_t\* a, const vec2\_t\* b );

void Mat3Set( mat3\_t\* m, const vec3\_t\* a, const vec3\_t\* b,   
 const vec3\_t\* c );

void Mat4Set( mat4\_t\* m, const vec4\_t\* a, const vec4\_t\* b,  
 const vec4\_t\* c, const vec4\_t\* d );

**Установить значение матрицы** m.

void Mat2Set4f( mat2\_t\* m, float ax, float ay,   
 float bx, float by );

void Mat3Set9f( mat3\_t\* m, float ax, float ay, float az,  
 float bx, float by, float bz,  
 float cx, float cy, float cz );

void Mat4Set16f( mat4\_t\*m,float ax,float ay,float az,float aw,  
 float bx, float by, float bz, float bw,  
 float cx, float cy, float cz, float cw,  
 float dx, float dy, float dz, float dw );

**Установить значение матрицы** m **из значений массива** src.

void Mat2Set4fv( mat2\_t\* m, const float\* src );

void Mat3Set9fv( mat3\_t\* m, const float\* src );

void Mat4Set16fv( mat4\_t\* m, const float\* src );

**Установить нулевую матрицу** m**.**

void Mat2Zero( mat2\_t\* m );

void Mat3Zero( mat3\_t\* m );

void Mat4Zero( mat4\_t\* m );

**Установить единичную матрицу** m**.**

void Mat2Ident( mat2\_t\* m );

void Mat3Ident( mat3\_t\* m );

void Mat4Ident( mat4\_t\* m );

**Сделать все числа матрицы** m **негативными (инвертировать по знаку).**

void Mat2Neg( mat2\_t\* m );

void Mat3Neg( mat3\_t\* m );

void Mat4Neg( mat4\_t\* m );

**Получить обратную матрицу. Функция высчитывает обратную матрицу для матрицы** m**, результат записывает в** m. **Функция возвращает** mtrue **если определитель матрицы не был равен** 0 **(а значит обратная матрица была посчитана), иначе функция возвращает** mfalse **если определитель был равен нулю.**

mbool\_t Mat2Inv( mat2\_t\* m );

mbool\_t Mat3Inv( mat3\_t\* m );

mbool\_t Mat4Inv( mat4\_t\* m );

**Умножить каждое значение матрицы** m **на** s. **Результат записать в** m.

void Mat2Scale( mat2\_t\* m, float s );

void Mat3Scale( mat3\_t\* m, float s );

void Mat4Scale( mat4\_t\* m, float s );

**Умножить матрицу** a **на вектор** b. **Результат записать в** out.

void Mat2MulVec2( vec2\_t\*out, const mat2\_t\*m, const vec2\_t\*v );

void Mat3MulVec3( vec3\_t\*out, const mat3\_t\*m, const vec3\_t\*v );

void Mat4MulVec4( vec4\_t\*out, const mat4\_t\*m, const vec4\_t\*v );

**Умножить матрицу** a **на вектор** b. **Результат записать в** out.

void Mat4MulVec3( vec3\_t\*out, const mat4\_t\*m, const vec3\_t\*v );

**Умножить матрицу** a **на матрицу** b. **Результат записать в** out.

void Mat2Mul( mat2\_t\* out, const mat2\_t\* a, const mat2\_t\* b );

void Mat3Mul( mat3\_t\* out, const mat3\_t\* a, const mat3\_t\* b );

void Mat4Mul( mat4\_t\* out, const mat4\_t\* a, const mat4\_t\* b );

**Сложить две матрицы** a **и** b. **Результат записать в** out.

void Mat2Add( mat2\_t\* out, const mat2\_t\* a, const mat2\_t\* b );

void Mat3Add( mat3\_t\* out, const mat3\_t\* a, const mat3\_t\* b );

void Mat4Add( mat4\_t\* out, const mat4\_t\* a, const mat4\_t\* b );

**Вычесть из матрицы** b **матрицу** a. **Результат записать в** out.

void Mat2Sub( mat2\_t\* out, const mat2\_t\* a, const mat2\_t\* b );

void Mat3Sub( mat3\_t\* out, const mat3\_t\* a, const mat3\_t\* b );

void Mat4Sub( mat4\_t\* out, const mat4\_t\* a, const mat4\_t\* b );

**Сравнить две матрицы** a **и** b. **Если обе матрицы равны, то возвращаемое значение будет** mtrue, **если не равны, то возвращаемое значение будет** mfalse.

mbool\_t Mat2Cmp( const mat2\_t\* a, const mat2\_t\* b );

mbool\_t Mat3Cmp( const mat3\_t\* a, const mat3\_t\* b );

mbool\_t Mat4Cmp( const mat4\_t\* a, const mat4\_t\* b );

**Сравнить две матрицы** a **и** b **с использованием епсилона** eps. **Если значения обеих матриц лежат в пределах** eps**, то возвращаемое значение будет** mtrue, **иначе возвращаемое значение будет** mfalse.

mbool\_t Mat2CmpEps( const mat2\_t\* a, const mat2\_t\* b, float eps );

mbool\_t Mat3CmpEps( const mat3\_t\* a, const mat3\_t\* b, float eps );

mbool\_t Mat4CmpEps( const mat4\_t\* a, const mat4\_t\* b, float eps );

**Вернуть** mtrue **если матрица** m **является диагональной (используется** eps**). Иначе вернуть** mfalse.

mbool\_t Mat2IsDiag( const mat2\_t\* m );

mbool\_t Mat3IsDiag( const mat3\_t\* m );

mbool\_t Mat4IsDiag( const mat4\_t\* m );

**Вернуть** mtrue **если матрица** m **является единичной (используется** eps**). Иначе вернуть** mfalse.

mbool\_t Mat2IsIdent( const mat2\_t\* m );

mbool\_t Mat3IsIdent( const mat3\_t\* m );

mbool\_t Mat4IsIdent( const mat4\_t\* m );

**Вернуть определитель матрицы** m.

float Mat2Det( const mat2\_t\* m );

float Mat3Det( const mat3\_t\* m );

float Mat4Det( const mat4\_t\* m );

**Транспонировать матрицу** m **и записать результат в** m**.**

void Mat2Transp( mat2\_t\* m );

void Mat3Transp( mat3\_t\* m );

void Mat4Transp( mat4\_t\* m );

**Записать в строку** out **значения матрицы** m **с количеством знаков после запятой** prec.

void Mat2ToStr( char\* out, const mat2\_t\* m, int prec );

void Mat3ToStr( char\* out, const mat3\_t\* m, int prec );

void Mat4ToStr( char\* out, const mat4\_t\* m, int prec );

**Преобразование из одного типа матрицы в другой.**

void Mat2ToMat3( mat3\_t\* out, const mat2\_t\* m );

void Mat2ToMat4( mat4\_t\* out, const mat2\_t\* m );

void Mat3ToMat2( mat2\_t\* out, const mat3\_t\* m );

void Mat3ToMat4( mat4\_t\* out, const mat3\_t\* m );

void Mat4ToMat2( mat2\_t\* out, const mat4\_t\* m );

void Mat4ToMat3( mat3\_t\* out, const mat4\_t\* m );