



A simple and easy-to-use library to enjoy videogames programming

[raylib Discord server][github.com/raysan5/raylib][raylib.h]

raylib

v4.5 quick reference card

module: raymath

```
// Utils math
float Clamp(float value, float min, float max);
float Lerp(float start, float end, float amount);
float Normalize(float value, float start, float end);
float Remap(float value, float inputStart, float inputEnd, float outputStart, float outputEnd); // Remap input value within input range to output range
float Wrap(float value, float min, float max);
int FloatEquals(float x, float y);

// Vector2 math
Vector2 Vector2Zero(void);
Vector2 Vector2One(void);
Vector2 Vector2Add(Vector2 v1, Vector2 v2);
Vector2 Vector2AddValue(Vector2 v, float add);
Vector2 Vector2Subtract(Vector2 v1, Vector2 v2);
Vector2 Vector2SubtractValue(Vector2 v, float sub);
float Vector2Length(Vector2 v);
float Vector2LengthSqr(Vector2 v);
float Vector2DotProduct(Vector2 v1, Vector2 v2);
float Vector2Distance(Vector2 v1, Vector2 v2);
float Vector2DistanceSqr(Vector2 v1, Vector2 v2);
float Vector2Angle(Vector2 v1, Vector2 v2);
Vector2 Vector2Scale(Vector2 v, float scale);
Vector2 Vector2Multiply(Vector2 v1, Vector2 v2);
Vector2 Vector2Negate(Vector2 v);
Vector2 Vector2Divide(Vector2 v1, Vector2 v2);
Vector2 Vector2Normalize(Vector2 v);
Vector2 Vector2Transform(Vector2 v, Matrix mat);
Vector2 Vector2Lerp(Vector2 v1, Vector2 v2, float amount);
Vector2 Vector2Reflect(Vector2 v, Vector2 normal);
Vector2 Vector2Rotate(Vector2 v, float angle);
Vector2 Vector2MoveTowards(Vector2 v, Vector2 target, float maxDistance);
Vector2 Vector2Invert(Vector2 v);
Vector2 Vector2Clamp(Vector2 v, Vector2 min, Vector2 max);
Vector2 Vector2ClampValue(Vector2 v, float min, float max);
int Vector2Equals(Vector2 p, Vector2 q);

// Vector3 math
Vector3 Vector3Zero(void);
Vector3 Vector3One(void);
Vector3 Vector3Add(Vector3 v1, Vector3 v2);
Vector3 Vector3AddValue(Vector3 v, float add);
Vector3 Vector3Subtract(Vector3 v1, Vector3 v2);
Vector3 Vector3SubtractValue(Vector3 v, float sub);
Vector3 Vector3Scale(Vector3 v, float scalar);
Vector3 Vector3Multiply(Vector3 v1, Vector3 v2);
Vector3 Vector3CrossProduct(Vector3 v1, Vector3 v2);
Vector3 Vector3Perpendicular(Vector3 v);
float Vector3Length(const Vector3 v);
float Vector3LengthSqr(const Vector3 v);
float Vector3DotProduct(Vector3 v1, Vector3 v2);
float Vector3Distance(Vector3 v1, Vector3 v2);
float Vector3DistanceSqr(Vector3 v1, Vector3 v2);
float Vector3Angle(Vector3 v1, Vector3 v2);
Vector3 Vector3Negate(Vector3 v);
Vector3 Vector3Divide(Vector3 v1, Vector3 v2);
Vector3 Vector3Normalize(Vector3 v);
void Vector3OrthoNormalize(Vector3 *v1, Vector3 *v2);
Vector3 Vector3Transform(Vector3 v, Matrix mat);
Vector3 Vector3RotateByQuaternion(Vector3 v, Quaternion q);
Vector3 Vector3RotateByAxisAngle(Vector3 v, Vector3 axis, float angle);
Vector3 Vector3Lerp(Vector3 v1, Vector3 v2, float amount);
Vector3 Vector3Reflect(Vector3 v, Vector3 normal);
Vector3 Vector3Min(Vector3 v1, Vector3 v2);
Vector3 Vector3Max(Vector3 v1, Vector3 v2);
Vector3 Vector3Barycenter(Vector3 p, Vector3 a, Vector3 b, Vector3 c);
Vector3 Vector3Unproject(Vector3 source, Matrix projection, Matrix view);
float3 Vector3ToFloatV(Vector3 v);
Vector3 Vector3Invert(Vector3 v);
Vector3 Vector3Clamp(Vector3 v, Vector3 min, Vector3 max);
Vector3 Vector3ClampValue(Vector3 v, float min, float max);
int Vector3Equals(Vector3 p, Vector3 q);
Vector3 Vector3Refract(Vector3 v, Vector3 n, float r);

// Matrix math
float MatrixDeterminant(Matrix mat);
float MatrixTrace(Matrix mat);
Matrix MatrixTranspose(Matrix mat);
Matrix MatrixInvert(Matrix mat);
Matrix MatrixIdentity(void);
Matrix MatrixAdd(Matrix left, Matrix right);
Matrix MatrixSubtract(Matrix left, Matrix right);
Matrix MatrixMultiply(Matrix left, Matrix right);
Matrix MatrixTranslate(float x, float y, float z);
Matrix MatrixRotate(Vector3 axis, float angle);
Matrix MatrixRotateX(float angle);
Matrix MatrixRotateY(float angle);
Matrix MatrixRotateZ(float angle);
Matrix MatrixRotateXYZ(Vector3 angle);
Matrix MatrixRotateZYX(Vector3 angle);
Matrix MatrixScale(float x, float y, float z);
Matrix MatrixFrustum(double left, double right, double bottom, double top, double near, double far); // Get perspective projection matrix
Matrix MatrixPerspective(double fovy, double aspect, double near, double far); // Get perspective projection matrix NOTE: Fovy angle must be provided in radians
Matrix MatrixOrtho(double left, double right, double bottom, double top, double near, double far); // Get orthographic projection matrix
Matrix MatrixLookAt(Vector3 eye, Vector3 target, Vector3 up);
float16 MatrixToFloatV(Matrix mat);

// Quaternion math
Quaternion QuaternionAdd(Quaternion q1, Quaternion q2);
Quaternion QuaternionAddValue(Quaternion q, float add);
Quaternion QuaternionSubtract(Quaternion q1, Quaternion q2);
Quaternion QuaternionSubtractValue(Quaternion q, float sub);
Quaternion QuaternionIdentity(void);
float QuaternionLength(Quaternion q);
Quaternion QuaternionNormalize(Quaternion q);
Quaternion QuaternionInvert(Quaternion q);
Quaternion QuaternionMultiply(Quaternion q1, Quaternion q2);
Quaternion QuaternionScale(Quaternion q, float mul);
Quaternion QuaternionDivide(Quaternion q1, Quaternion q2);
Quaternion QuaternionLerp(Quaternion q1, Quaternion q2, float amount);
Quaternion QuaternionNlerp(Quaternion q1, Quaternion q2, float amount);
Quaternion QuaternionSlerp(Quaternion q1, Quaternion q2, float amount);
Quaternion QuaternionFromVector3ToVector3(Vector3 from, Vector3 to);
Quaternion QuaternionFromMatrix(Matrix mat);
Matrix QuaternionToMatrix(Quaternion q);
Quaternion QuaternionFromAxisAngle(Vector3 axis, float angle);
void QuaternionToAxisAngle(Quaternion q, Vector3 *outAxis, float *outAngle);
Quaternion QuaternionFromEuler(float pitch, float yaw, float roll);
Vector3 QuaternionToEuler(Quaternion q);
Quaternion QuaternionTransform(Quaternion q, Matrix mat);
int QuaternionEquals(Quaternion p, Quaternion q);

// Vector with components value 0.0f
// Vector with components value 1.0f
// Add two vectors (v1 + v2)
// Add vector and float value
// Subtract two vectors (v1 - v2)
// Subtract vector by float value
// Calculate vector length
// Calculate vector square length
// Calculate two vectors dot product
// Calculate distance between two vectors
// Calculate square distance between two vectors
// Calculate angle from two vectors
// Scale vector (multiply by value)
// Multiply vector by vector
// Negate vector
// Divide vector by vector
// Normalize provided vector
// Transforms a Vector2 by a given Matrix
// Calculate linear interpolation between two vectors
// Calculate reflected vector to normal
// Rotate vector by angle
// Move Vector towards target
// Invert the given vector
// Clamp the components of the vector between min and max values specified by the given vectors
// Clamp the magnitude of the vector between two min and max values
// Check whether two given vectors are almost equal

// Vector with components value 0.0f
// Vector with components value 1.0f
// Add two vectors
// Add vector and float value
// Subtract two vectors
// Subtract vector by float value
// Multiply vector by scalar
// Multiply vector by vector
// Calculate two vectors cross product
// Calculate one vector perpendicular vector
// Calculate vector length
// Calculate vector square length
// Calculate two vectors dot product
// Calculate distance between two vectors
// Calculate square distance between two vectors
// Calculate angle between two vectors
// Negate provided vector (invert direction)
// Divide vector by vector
// Normalize provided vector
// Orthonormalize provided vectors Makes vectors normalized and orthogonal to each other Gram-Schmidt
// Transforms a Vector3 by a given Matrix
// Transform a vector by quaternion rotation
// Rotates a vector around an axis
// Calculate linear interpolation between two vectors
// Calculate reflected vector to normal
// Get min value for each pair of components
// Get max value for each pair of components
// Compute barycenter coordinates (u, v, w) for point p with respect to triangle (a, b, c) NOTE: As
// Projects a Vector3 from screen space into object space NOTE: We are avoiding calling other rayma
// Get Vector3 as float array
// Invert the given vector
// Clamp the components of the vector between min and max values specified by the given vectors
// Clamp the magnitude of the vector between two values
// Check whether two given vectors are almost equal
// Compute the direction of a refracted ray where v specifies the normalized direction of the incom

// Compute matrix determinant
// Get the trace of the matrix (sum of the values along the diagonal)
// Transposes provided matrix
// Invert provided matrix
// Get identity matrix
// Add two matrices
// Subtract two matrices (left - right)
// Get two matrix multiplication NOTE: When multiplying matrices... the order matters!
// Get translation matrix
// Create rotation matrix from axis and angle NOTE: Angle should be provided in radians
// Get x-rotation matrix NOTE: Angle must be provided in radians
// Get y-rotation matrix NOTE: Angle must be provided in radians
// Get z-rotation matrix NOTE: Angle must be provided in radians
// Get xyz-rotation matrix NOTE: Angle must be provided in radians
// Get zyx-rotation matrix NOTE: Angle must be provided in radians
// Get scaling matrix

// Add two quaternions
// Add quaternion and float value
// Subtract two quaternions
// Subtract quaternion and float value
// Get identity quaternion
// Computes the length of a quaternion
// Normalize provided quaternion
// Invert provided quaternion
// Calculate two quaternion multiplication
// Scale quaternion by float value
// Divide two quaternions
// Calculate linear interpolation between two quaternions
// Calculate slerp-optimized interpolation between two quaternions
// Calculates spherical linear interpolation between two quaternions
// Calculate quaternion based on the rotation from one vector to another
// Get a quaternion for a given rotation matrix
// Get a matrix for a given quaternion
// Get rotation quaternion for an angle and axis NOTE: Angle must be provided in radians
// Get the rotation angle and axis for a given quaternion
// Get the quaternion equivalent to Euler angles NOTE: Rotation order is ZYX
// Get the Euler angles equivalent to quaternion (roll, pitch, yaw) NOTE: Angles are returned in a
// Transform a quaternion given a transformation matrix
// Check whether two given quaternions are almost equal
```

Other cheatsheets

- [raylib cheatsheet](#)