raylib

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

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

v4.5 quick reference card

module: raymath

```
// Utils math
float Clamp(float value, float min, float max);                             // Function specifiers definition Defines and Macros Get float vector for Matrix Get float vector f
float Lerp(float start, float end, float amount);                           // Calculate linear interpolation between two floats
float Normalize(float value, float start, float end);                       // Normalize input value within input range
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);                              // Wrap input value from min to max
int FloatEquals(float x, float y);                                          // Check whether two given floats are almost equal

// Vector2 math
Vector2 Vector2Zero(void);                                                  // Vector with components value 0.0f
Vector2 Vector2One(void);                                                   // Vector with components value 1.0f
Vector2 Vector2Add(Vector2 v1, Vector2 v2);                                 // Add two vectors (v1 + v2)
Vector2 Vector2AddValue(Vector2 v, float add);                              // Add vector and float value
Vector2 Vector2Subtract(Vector2 v1, Vector2 v2);                            // Subtract two vectors (v1 - v2)
Vector2 Vector2SubtractValue(Vector2 v, float sub);                         // Subtract vector by float value
float Vector2Length(Vector2 v);                                             // Calculate vector length
float Vector2LengthSqr(Vector2 v);                                          // Calculate vector square length
float Vector2DotProduct(Vector2 v1, Vector2 v2);                            // Calculate two vectors dot product
float Vector2Distance(Vector2 v1, Vector2 v2);                              // Calculate distance between two vectors
float Vector2DistanceSqr(Vector2 v1, Vector2 v2);                           // Calculate square distance between two vectors
float Vector2Angle(Vector2 v1, Vector2 v2);                                 // Calculate angle from two vectors
Vector2 Vector2Scale(Vector2 v, float scale);                               // Scale vector (multiply by value)
Vector2 Vector2Multiply(Vector2 v1, Vector2 v2);                            // Multiply vector by vector
Vector2 Vector2Negate(Vector2 v);                                           // Negate vector
Vector2 Vector2Divide(Vector2 v1, Vector2 v2);                              // Divide vector by vector
Vector2 Vector2Normalize(Vector2 v);                                        // Normalize provided vector
Vector2 Vector2Transform(Vector2 v, Matrix mat);                            // Transforms a Vector2 by a given Matrix
Vector2 Vector2Lerp(Vector2 v1, Vector2 v2, float amount);                  // Calculate linear interpolation between two vectors
Vector2 Vector2Reflect(Vector2 v, Vector2 normal);                          // Calculate reflected vector to normal
Vector2 Vector2Rotate(Vector2 v, float angle);                              // Rotate vector by angle
Vector2 Vector2MoveTowards(Vector2 v, Vector2 target, float maxDistance);   // Move Vector towards target
Vector2 Vector2Invert(Vector2 v);                                           // Invert the given vector
Vector2 Vector2Clamp(Vector2 v, Vector2 min, Vector2 max);                  // Clamp the components of the vector between min and max values specified by the given vectors
Vector2 Vector2ClampValue(Vector2 v, float min, float max);                 // Clamp the magnitude of the vector between two min and max values
int Vector2Equals(Vector2 p, Vector2 q);                                    // Check whether two given vectors are almost equal

// Vector3 math
Vector3 Vector3Zero(void);                                                  // Vector with components value 0.0f
Vector3 Vector3One(void);                                                   // Vector with components value 1.0f
Vector3 Vector3Add(Vector3 v1, Vector3 v2);                                 // Add two vectors
Vector3 Vector3AddValue(Vector3 v, float add);                              // Add vector and float value
Vector3 Vector3Subtract(Vector3 v1, Vector3 v2);                            // Subtract two vectors
Vector3 Vector3SubtractValue(Vector3 v, float sub);                         // Subtract vector by float value
Vector3 Vector3Scale(Vector3 v, float scalar);                              // Multiply vector by scalar
Vector3 Vector3Multiply(Vector3 v1, Vector3 v2);                            // Multiply vector by vector
Vector3 Vector3CrossProduct(Vector3 v1, Vector3 v2);                        // Calculate two vectors cross product
Vector3 Vector3Perpendicular(Vector3 v);                                    // Calculate one vector perpendicular vector
float Vector3Length(const Vector3 v);                                       // Calculate vector length
float Vector3LengthSqr(const Vector3 v);                                    // Calculate vector square length
float Vector3DotProduct(Vector3 v1, Vector3 v2);                            // Calculate two vectors dot product
float Vector3Distance(Vector3 v1, Vector3 v2);                              // Calculate distance between two vectors
float Vector3DistanceSqr(Vector3 v1, Vector3 v2);                           // Calculate square distance between two vectors
float Vector3Angle(Vector3 v1, Vector3 v2);                                 // Calculate angle between two vectors
Vector3 Vector3Negate(Vector3 v);                                           // Negate provided vector (invert direction)
Vector3 Vector3Divide(Vector3 v1, Vector3 v2);                              // Divide vector by vector
Vector3 Vector3Normalize(Vector3 v);                                        // Normalize provided vector
void Vector3OrthoNormalize(Vector3 *v1, Vector3 *v2);                       // Orthonormalize provided vectors Makes vectors normalized and orthogonal to each other Gram-Schmi
Vector3 Vector3Transform(Vector3 v, Matrix mat);                            // Transforms a Vector3 by a given Matrix
Vector3 Vector3RotateByQuaternion(Vector3 v, Quaternion q);                 // Transform a vector by quaternion rotation
Vector3 Vector3RotateByAxisAngle(Vector3 v, Vector3 axis, float angle);     // Rotates a vector around an axis
```

```
Vector3 Vector3Lerp(Vector3 v1, Vector3 v2, float amount);                // Calculate linear interpolation between two vectors
Vector3 Vector3Reflect(Vector3 v, Vector3 normal);                        // Calculate reflected vector to normal
Vector3 Vector3Min(Vector3 v1, Vector3 v2);                               // Get min value for each pair of components
Vector3 Vector3Max(Vector3 v1, Vector3 v2);                               // Get max value for each pair of components
Vector3 Vector3Barycenter(Vector3 p, Vector3 a, Vector3 b, Vector3 c);    // Compute barycenter coordinates (u, v, w) for point p with respect to triangle (a, b, c) NOTE: As
Vector3 Vector3Unproject(Vector3 source, Matrix projection, Matrix view); // Projects a Vector3 from screen space into object space NOTE: We are avoiding calling other rayma
float3 Vector3ToFloatV(Vector3 v);                                        // Get Vector3 as float array
Vector3 Vector3Invert(Vector3 v);                                         // Invert the given vector
Vector3 Vector3Clamp(Vector3 v, Vector3 min, Vector3 max);                // Clamp the components of the vector between min and max values specified by the given vectors
Vector3 Vector3ClampValue(Vector3 v, float min, float max);               // Clamp the magnitude of the vector between two values
int Vector3Equals(Vector3 p, Vector3 q);                                  // Check whether two given vectors are almost equal
Vector3 Vector3Refract(Vector3 v, Vector3 n, float r);                    // Compute the direction of a refracted ray where v specifies the normalized direction of the incom

// Matrix math
float MatrixDeterminant(Matrix mat);                                      // Compute matrix determinant
float MatrixTrace(Matrix mat);                                            // Get the trace of the matrix (sum of the values along the diagonal)
Matrix MatrixTranspose(Matrix mat);                                       // Transposes provided matrix
Matrix MatrixInvert(Matrix mat);                                          // Invert provided matrix
Matrix MatrixIdentity(void);                                              // Get identity matrix
Matrix MatrixAdd(Matrix left, Matrix right);                              // Add two matrices
Matrix MatrixSubtract(Matrix left, Matrix right);                         // Subtract two matrices (left - right)
Matrix MatrixMultiply(Matrix left, Matrix right);                         // Get two matrix multiplication NOTE: When multiplying matrices... the order matters!
Matrix MatrixTranslate(float x, float y, float z);                        // Get translation matrix
Matrix MatrixRotate(Vector3 axis, float angle);                           // Create rotation matrix from axis and angle NOTE: Angle should be provided in radians
Matrix MatrixRotateX(float angle);                                        // Get x-rotation matrix NOTE: Angle must be provided in radians
Matrix MatrixRotateY(float angle);                                        // Get y-rotation matrix NOTE: Angle must be provided in radians
Matrix MatrixRotateZ(float angle);                                        // Get z-rotation matrix NOTE: Angle must be provided in radians
Matrix MatrixRotateXYZ(Vector3 angle);                                    // Get xyz-rotation matrix NOTE: Angle must be provided in radians
Matrix MatrixRotateZYX(Vector3 angle);                                    // Get zyx-rotation matrix NOTE: Angle must be provided in radians
Matrix MatrixScale(float x, float y, float z);                            // Get scaling matrix
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);             // Get camera look-at matrix (view matrix)
float16 MatrixToFloatV(Matrix mat);                                       // Get float array of matrix data

// Quaternion math
Quaternion QuaternionAdd(Quaternion q1, Quaternion q2);                   // Add two quaternions
Quaternion QuaternionAddValue(Quaternion q, float add);                   // Add quaternion and float value
Quaternion QuaternionSubtract(Quaternion q1, Quaternion q2);              // Subtract two quaternions
Quaternion QuaternionSubtractValue(Quaternion q, float sub);              // Subtract quaternion and float value
Quaternion QuaternionIdentity(void);                                      // Get identity quaternion
float QuaternionLength(Quaternion q);                                     // Computes the length of a quaternion
Quaternion QuaternionNormalize(Quaternion q);                             // Normalize provided quaternion
Quaternion QuaternionInvert(Quaternion q);                                // Invert provided quaternion
Quaternion QuaternionMultiply(Quaternion q1, Quaternion q2);              // Calculate two quaternion multiplication
Quaternion QuaternionScale(Quaternion q, float mul);                      // Scale quaternion by float value
Quaternion QuaternionDivide(Quaternion q1, Quaternion q2);                // Divide two quaternions
Quaternion QuaternionLerp(Quaternion q1, Quaternion q2, float amount);    // Calculate linear interpolation between two quaternions
Quaternion QuaternionNlerp(Quaternion q1, Quaternion q2, float amount);   // Calculate slerp-optimized interpolation between two quaternions
Quaternion QuaternionSlerp(Quaternion q1, Quaternion q2, float amount);   // Calculates spherical linear interpolation between two quaternions
Quaternion QuaternionFromVector3ToVector3(Vector3 from, Vector3 to);      // Calculate quaternion based on the rotation from one vector to another
Quaternion QuaternionFromMatrix(Matrix mat);                              // Get a quaternion for a given rotation matrix
Matrix QuaternionToMatrix(Quaternion q);                                  // Get a matrix for a given quaternion
Quaternion QuaternionFromAxisAngle(Vector3 axis, float angle);            // Get rotation quaternion for an angle and axis NOTE: Angle must be provided in radians
void QuaternionToAxisAngle(Quaternion q, Vector3 *outAxis, float *outAngle); // Get the rotation angle and axis for a given quaternion
Quaternion QuaternionFromEuler(float pitch, float yaw, float roll);       // Get the quaternion equivalent to Euler angles NOTE: Rotation order is ZYX
Vector3 QuaternionToEuler(Quaternion q);                                  // Get the Euler angles equivalent to quaternion (roll, pitch, yaw) NOTE: Angles are returned in a
Quaternion QuaternionTransform(Quaternion q, Matrix mat);                 // Transform a quaternion given a transformation matrix
int QuaternionEquals(Quaternion p, Quaternion q);                         // Check whether two given quaternions are almost equal
```

Other cheatsheets

- raylib cheatsheet

raylib quick reference card - Copyright (c) 2013-2024 Ramon Santamaria (@raysan5)