Luke Monaghan library documentation

This documentation will explain all functionality of all class's and functions within this library.

Index

Maths.h
Generic Maths functions to be used within the Library.

PI
Defines PI for use within the library.

RAD2DEG
Defines Radian to Degree for use with angle converting.

DEG2RAD
Defines Degree to Radian for use with angle converting.

```
float radToDegree(float rad);
```
Takes in a radian angle and returns a degree.

```
float degreeToRad(float degree);
```
Takes in a degree angle and returns a radian.

```
float Recipf(float x);
```
Returns the reciprocal of the given float,

```
float Minf(float x, float y);
```
Returns the lower value of the two given floats.

```
float Maxf(float x, float y);
```
Returns the Greater value of the two given floats.

```
int PowerUp(int value);
```
Returns the closest power of two above the current value.

```
int PowerDown(int value);
```
Returns the closest power of two above the current value.

```
int PowerClosest(int value);
```
Returns the closest power of two.

```
float Lerp(float f_1, float f_2, float t);
```
Returns the Lineal Interpolation between float f_1 and float f_2 with translation of t where 't' is between 0.0f and 1.0f.

Vector2.h

Class of Vector2

This class uses an union to store the values. This allows for use of both x/y and u/v within the same memory position. Setting x will set u, same with y and v. This is used simply for coding behaviours, Very helpful when wanting to save UV coordinates in a Vector2.

Constructors

```
Vector2();
```
Default constructor. Sets both x/y to 0.0f.

```
Vector2(float a, float b);
```
Overloaded constructor. Sets x/u to a, y/v to b.

Member Functions

```
float Magnitude();
```
Returns the magnitude of the current Vector2.

```
float MagnitudeSquared();
```
Returns the magnitude squared of the current Vector2.

```
float Dot(Vector2 vect2);
```
Returns a float containing the value of the dot product between self and vect2.

```
bool IsUnit();
```
Returns a bool stating if the value is a Unit Vector, this is done by normalising.

```
Vector2 GetUnit();
```
Returns the unit vector of this vector to Vector2.

```
Vector2 NormalizeReturn();
```
Returns a Vector2 containing the normalised vector.

```
void Normalize();
```
Normalizes the current Vector2.

```
Vector2 GetPerpendicular();
```
Return the perpendicualar Vector2 of the current Vector2.

```
void RotateRad(float angle);
```
Rotates the current Vector2 by the Radian of angle.

```
void RotateDegree(float angle);
```
Rotates the current Vector2 by the Degree of angle.

```
void RadianToVect(float angle);
```
Converts the given Radian of angle to a Vector2.

```
void degreeToVect(float angle);
```
Converts the given Degree of angle to a Vector2.

```
void Project(float angle, float distance);
```
Projects the current Vector2 with a float angle and float distance.

```
Vector2 Lerp(Vector2 vect1,Vector2 vect2,float t);
```
Returns a Vector2 containing the Linear Interpolation of the Vector2 vect1, Vector2 vect2 with the distance of float t where t is between 0.0f and 1.0f.

```
void Zero();
```
Sets the values of Vector2 to 0.0f.

```
void One();
```
Sets the values of Vector2 to 1.0f.

```
float Sum();
```
Gets the sum of the values of Vector2.

```
float Min();
```
Gets the lesser value of the values of Vector2.

```
float Max();
```
Gets the greater value of the values of Vector2.

Vector2 Overloads

`bool Vector2 == Vector2`
: Returns a bool checking if left Vector2 is equal to right Vector2.

`bool  Vector2 != Vector2`
: Returns a bool checking if left Vector2 is not equal to right Vector2.

`Vector2 = float scalar`
: Sets Vector2's values to float scalar.

`-Vector2`
: Returns the negative values of Vector2.

`Vector2++`
: Adds 1.0f to both values inside Vector2.

`Vector2 + Vector2`
: Returns the values of left Vector2 plus right Vector2.

`Vector2 += Vector2`
: Returns the values of left Vector2 plus the right Vector2 to the left Vector2.

`Vector2 + float scalar`
: Returns the values of left Vector2 plus float scalar.

`Vector2 += float scalar`
: Returns the values of Vector2 plus scalar toVector2.

`Vector2--`
: Subtracts 1.0f from both values inside Vector2.

`Vector2 - Vector2`
: Returns the values of left Vector2 minus right Vector2.

`Vector2 -= Vector2`
: Returns the values of left Vector2 subtracted by right Vector2 to the left Vector2.

`Vector2 -float scalar`
: Returns the values of left Vector2 minus float scalar.

`Vector2 -= float scalar`
: Returns the values of left Vector2 minus scalar toVector2.

`Vector2 / Vector2`
: Returns the values of left Vector2 divide right Vector2.

`Vector2 /= Vector2`
: Returns the values of left Vector2 divided by right Vector2 to the left Vector2.

`Vector2 / float scalar`
: Returns the values of left Vector2 divide float scalar.

`Vector2 /= float scalar`
: Returns the values of left Vector2 divide scalar toVector2.

`Vector2 * Vector2`
: Returns the values of left Vector2 multiplied right Vector2.

`Vector2 *= Vector2`
: Returns the values of left Vector2 multiplied by right Vector2 to the left Vector2.

`Vector2 * float scalar`
: Returns the values of left Vector2 multiplied by the float scalar.

`Vector2 *= float scalar`
: Returns the values of left Vector2 multiplied by the scalar toVector2.

Vector3.h

Class of Vector3

Constructors

```
Vector3();
```
Default constructor. Sets both x/y/z to 0.0f.
```
Vector3(float a, float b,float c);
```
Overloaded constructor. Sets x to a, y to b and z.

Member Functions

```
float Magnitude();
```
Returns the magnitude of the current Vector3.
```
float MagnitudeSquared();
```
Returns the magnitude squared of the current Vector3.

```
float Dot(Vector3 vect);
```
Returns a float containing the value of the dot product between self and vect.

```
bool IsUnit();
```
Returns a bool stating if the value is a Unit Vector, this is done by normalising.
```
Vector3 GetUnit();
```
Returns the unit vector of this vector to Vector3.

```
Vector3 NormalizeReturn();
```
Returns a Vector3 containing the normalised vector.
```
void Normalize();
```
Normalizes the current Vector3.

```
Vector3 Cross(Vector3 vect);
```
Returns a Vector3 containing the Cross Product of Current Vector3 and Vector3 vect.
```
Vector3 Cross(Vector3 vect, Vector3 vect2);
```
Returns a Vector3 containing the Cross Product of Vector3 vect and Vector3 vect2.
```
Vector3 Cross2(Vector3 vect,Vector3 vect2);
```
Returns a Vector3 containing the Cross Product of Vector3 vect and Vector2 vect2.

```
void RotateXRad(float angle);
```
Rotates the current Vector3 by the Radian of angle on X axis.
```
void RotateXDeg(float angle);
```

Rotates the current Vector3 by the Degree of angle on X axis.
```
void RotateYRad(float angle);
```
Rotates the current Vector3 by the Radian of angle of Y axis.
```
void RotateYDeg(float angle);
```

Rotates the current Vector3 by the Degree of angle of Y axis.
```
void RotateZRad(float angle);
```
Rotates the current Vector3 by the Radian of angle of Z axis.
```
void RotateZDeg(float angle);
```
Rotates the current Vector3 by the Degree of angle of Z axis.

```
void Project(Vector3 angles, float distance);
```
Projects the current Vector3 with Vector3 angles and a float distance.
```
void ProjectOrtho(Vector3 angles, float distance);
```
Projects the current Vector3 with Vector3 angles and a float distance with Orthographic Projection.

```
Vector2 Lerp(Vector2 vect1,Vector2 vect2,float t);
```
Returns a Vector2 containing the Linear Interpolation of the Vector2 vect1, Vector2 vect2 with the distance of float t where t is between 0.0f and 1.0f.

```
void Zero();
```
Sets the values of Vector3 to 0.0f.
```
void One();
```
Sets the values of Vector3 to 1.0f.
```
float Sum();
```
Gets the sum of the values of Vector3.
```
float Min();
```
Gets the lesser value of the values of Vector3.
```
float Max();
```
Gets the greater value of the values of Vector3.

Vector3 Overloads

`bool Vector3 == Vector3`
: Returns a bool checking if left Vector3 is equal to right Vector3.

`bool  Vector3 != Vector3`
: Returns a bool checking if left Vector3 is not equal to right Vector3.

`Vector3 = float scalar`
: Sets Vector3's values to float scalar.

`- Vector3`
: Returns the negative values of Vector3.

`Vector3++`
: Adds 1.0f to both values inside Vector3.

`Vector3 + Vector3`
: Returns the values of left Vector3 plus right Vector3.

`Vector3 += Vector3`
: Returns the values of left Vector3 plus the right Vector3 to the left Vector3.

`Vector3 + float scalar`
: Returns the values of left Vector3 plus float scalar.

`Vector3 += float scalar`
: Returns the values of Vector3 plus scalar toVector3.

`Vector3--`
: Subtracts 1.0f from both values inside Vector3.

`Vector3 - Vector3`
: Returns the values of left Vector3 minus right Vector3.

`Vector3 -= Vector3`
: Returns the values of left Vector3 subtracted by right Vector3 to the left Vector3.

`Vector3 -float scalar`
: Returns the values of left Vector3 minus float scalar.

`Vector3 -= float scalar`
: Returns the values of left Vector3 minus scalar toVector3.

`Vector3 / Vector3`
: Returns the values of left Vector3 divide right Vector3.

`Vector3 /= Vector3`
: Returns the values of left Vector3 divided by right Vector3 to the left Vector3.

`Vector3 / float scalar`
: Returns the values of left Vector3 divide float scalar.

`Vector3 /= float scalar`
: Returns the values of left Vector3 divide scalar toVector3.

`Vector3 * Vector3`
: Returns the values of left Vector3 multiplied right Vector3.

`Vector3 *= Vector3`
: Returns the values of left Vector3 multiplied by right Vector3 to the left Vector3.

`Vector3 * float scalar`
: Returns the values of left Vector3 multiplied by the float scalar.

`Vector3 *= float scalar`
: Returns the values of left Vector3 multiplied by the scalar toVector3.

Vector4.h

Class of Vector4
This class uses an union to store the values. This allows for use of both w/r, x/g, y/b and z/a within the same memory position. Setting “w” will set “r”, same with “x”/”g”,”y”/”b” and “z”/”a” .

Constructors

`Vector4();`
Default constructor. Sets both w/x/y/z to 0.0f.

`Vector4(float a, float b,float c, float d);`
Overloaded constructor. Sets w to a, x to b, y to c and z to d.

`Vector4(char* hex);`
Default constructor. Creates the RGBA values from the 8 Character hex string.

Member Functions

`void Hex(char* hex);`
Creates the RGBA values from the 8 Character hex string.

`float Magnitude();`
Returns the magnitude of the current Vector4.

`float MagnitudeSquared();`
Returns the magnitude squared of the current Vector4.

`Vector4 NormalizeReturn();`
Returns a Vector2 containing the normalised Vector4.

`void Normalize();`
Normalizes the current Vector4.

`void Zero();`
Sets the values of Vector4 to 0.0f.

`void One();`
Sets the values of Vector4 to 1.0f.

`float Sum();`
Gets the sum of the values of Vector4.

`float Min();`
Gets the lesser value of the values of Vector4.

`float Max();`
Gets the greater value of the values of Vector4.

Vector4 Overloads

`bool Vector4 == Vector4`
: Returns a bool checking if left Vector4 is equal to right Vector4.

`bool  Vector4 != Vector4`
: Returns a bool checking if left Vector4 is not equal to right Vector4.

`Vector4 = float scalar`
: Sets Vector4's Values to float scalar.

`-Vector4`
: Returns the negative values of Vector4.

`Vector4++`
: Adds 1.0f to both values inside Vector4.

`Vector4 + Vector4`
: Returns the values of left Vector4 plus right Vector4.

`Vector4 += Vector4`
: Returns the values of left Vector4 plus the right Vector4 to the left Vector4.

`Vector4 + float scalar`
: Returns the values of left Vector4 plus float scalar.

`Vector4 += float scalar`
: Returns the values of Vector4 plus scalar toVector4.

`Vector4--`
: Subtracts 1.0f from both values inside Vector4.

`Vector4 - Vector4`
: Returns the values of left Vector4 minus right Vector4.

`Vector4 -= Vector4`
: Returns the values of left Vector4 subtracted by right Vector4 to the left Vector4.

`Vector4 -float scalar`
: Returns the values of left Vector4 minus float scalar.

`Vector4 -= float scalar`
: Returns the values of left Vector4 minus scalar toVector4.

`Vector4 / Vector4`
: Returns the values of left Vector4 divide right Vector4.

`Vector4 /= Vector4`
: Returns the values of left Vector4 divided by right Vector4 to the left Vector4.

`Vector4 / float scalar`
: Returns the values of left Vector4 divide float scalar.

`Vector4 /= float scalar`
: Returns the values of left Vector4 divide scalar toVector4.

`Vector4 * Vector4`
: Returns the values of left Vector4 multiplied right Vector4.

`Vector4 *= Vector4`
: Returns the values of left Vector4 multiplied by right Vector4 to the left Vector4.

`Vector4 * float scalar`
: Returns the values of left Vector4 multiplied by the float scalar.

`Vector4 *= float scalar`
: Returns the values of left Vector4 multiplied by the scalar toVector4.

Matrix3.h

Union Values

```
float m[3][3];
```

3x3 Matrix Array of m, useful for looping etc.

```
float m00,m01,m02,
      m10,m11,m12,
      m20,m21,m22;
```

Individual floats for each float in Matrix3. Useful for exact index's/passing.

Constructors

```
Matrix3();
```

Creates the Matrix3 setting to an Identity Matrix.

```
Matrix3(float fm00,float fm01,float fm02,
        float fm10,float fm11,float fm12,
        float fm20,float fm21,float fm22);
```

Creates the Matrix3 setting each index to the corresponding above floats.

Member Functions

```
float Determinant();
```

Returns the Determinant of the Matrix3.

```
bool Inverse();
```

Returns a bool stating if the inverse on Matrix3 was successful.

```
bool InverseGet(Matrix3 &Mat3);
```

Returns a bool stating if the inverse on Matrix3 was successful Does not change values.

```
void RotateX(float angle);
```

Creates a Rotation Matrix3 on the X axis with the given float angle.

```
void RotateY(float angle);
```

Creates a Rotation Matrix3 on the Y axis with the given float angle.

```
void RotateZ(float angle);
```

Creates a Rotation Matrix3 on the Z axis with the given float angle.

```
void RotateXYZ(Vector3 v3Angles);
```

Creates a Rotation Matrix3 on the XYZ axis with the given Vector3 v3Angles.

```
void RotateYXZ(Vector3 v3Angles);
```

Creates a Rotation Matrix3 on the YXZ axis with the givenVector3 v3Angles.

```
void RotateZXY(Vector3 v3Angles);
```

Creates a Rotation Matrix3 on the ZXY axis with the given Vector3 v3Angles.

```
void RotateZYX(Vector3 v3Angles);
```

Creates a Rotation Matrix3 on the ZYX axis with the given Vector3 v3Angles.

```
void Ortho();
```

Creates a Matrix3 Orthographic Projection.

```
void OrthoNormalize();
```

Normalize this Matrix3 of Orthographic Projection.

```
void Scale(Vector3 vect);
```

Create a Scale Matrix3 using the Vector3 vect.

```
void Scale(float scalar);
```

Create a Scale Matrix3 using the float scalar.

```
void Identity();
```

Set this Matrix3 as an Identity Matrix.

```
void Zero();
```

Set all of this Matrix3's Values to 0.0f.

Matrix3 Overloads

`bool Matrix3 ==  Matrix3`
: Returns a bool checking if left Matrix3 is equal to right Matrix3.

`bool  Matrix3 != Matrix3`
: Returns a bool checking if left Matrix3 is not equal to right Matrix3.

`bool Matrix3 = float scalar`
: Sets Matrix3's values to float scalar.

`Matrix3 =  Matrix3`
: Sets Matrix3's values right Matrix3's values.

`-Matrix3`
: Returns the negative values of Matrix3.

`Matrix3++`
: Adds 1.0f to both values inside Matrix3.

`Matrix3 + Matrix3`
: Returns the values of left Matrix3 plus right Matrix3.

`Matrix3 += Matrix3`
: Returns the values of left Matrix3 plus the right Matrix3 to the left Matrix3.

`Matrix3--`
: Subtracts 1.0f from both values inside Matrix3.

`Matrix3 - Matrix3`
: Returns the values of left Matrix3 minus right Matrix3.

`Matrix3 -= Matrix3`
: Returns the values of left Matrix3 subtracted by right Matrix3 to the left Matrix3.

`Matrix3 * Matrix3`
: Returns the values of left Matrix3 multiplied right Matrix3.

`Matrix3 *= Matrix3`
: Returns the values of left Matrix3 multiplied by right Matrix3 to the left Matrix3.

`Matrix3 * float scalar`
: Returns the values of left Matrix3 multiplied by the float scalar.

`Matrix3 *= float scalar`
: Returns the values of left Matrix3 multiplied by the scalar toMatrix3.

Matrix4.h

Union Values

```
float m[4][4];
```

4x4 Matrix Array of m, useful for looping etc.

```
float m00,m01,m02,m03,
      m10,m11,m12,m13,
      m20,m21,m22,m23,
      m30,m31,m32,m33;
```

Individual floats for each float in Matrix4. Useful for exact index's/passing.

Constructors

```
Matrix4();
```

Creates the Matrix4 setting to an Identity Matrix.

```
Matrix4(float fm00,float fm01,float fm02,float fm03,
        float fm10,float fm11,float fm12,float fm13,
        float fm20,float fm21,float fm22,float fm23,
        float fm30,float fm31,float fm32,float fm33);
```

Creates the Matrix4 setting each index to the corresponding above floats.

Member Functions

```
void RotateX(float angle);
```

Creates a Rotation Matrix4 on the X axis with the given float angle.

```
void RotateY(float angle);
```

Creates a Rotation Matrix4 on the Y axis with the given float angle.

```
void RotateZ(float angle);
```

Creates a Rotation Matrix4 on the Z axis with the given float angle.

```
void RotateXYZ(Vector3 v3Angles);
```

Creates a Rotation Matrix4 on the XYZ axis with the given Vector3 v3Angles.

```
void RotateYXZ(Vector3 v3Angles);
```

Creates a Rotation Matrix4 on the YXZ axis with the givenVector3 v3Angles.

```
void RotateZXY(Vector3 v3Angles);
```

Creates a Rotation Matrix4 on the ZXY axis with the given Vector3 v3Angles.

```
void RotateZYX(Vector3 v3Angles);
```

Creates a Rotation Matrix4 on the ZYX axis with the given Vector3 v3Angles.

```
void Ortho(float fLeft,float fRight,float fTop,float fBottom,float fNear,float fFar)
```

Creates a Matrix4 Orthographic Projection.

```
void OrthoNormalize();
```

Normalize this Matrix4 of Orthographic Projection.

```
void Scale(Vector3 vect);
```

Create a Scale Matrix4 using the Vector3 vect.

```
void Scale(float scalar);
```

Create a Scale Matrix4 using the float scalar.

```
void Identity();
```

Set this Matrix4 as an Identity Matrix.

```
void Zero();
```

Set all of this Matrix4's Values to 0.0f.

Matrix4 Overloads

`bool Matrix4 == Matrix4`
: Returns a bool checking if left Matrix4 is equal to right Matrix4.

`bool  Matrix4 != Matrix4`
: Returns a bool checking if left Matrix4 is not equal to right Matrix4.

`bool Matrix4 = float scalar`
: Sets Matrix4's values to float scalar.

`Matrix4 =  Matrix4`
: Sets Matrix4's values right Matrix4's values.

`-Matrix4`
: Returns the negative values of Matrix4.

`Matrix4++`
: Adds 1.0f to both values inside Matrix4.

`Matrix4 + Matrix4`
: Returns the values of left Matrix4 plus right Matrix4.

`Matrix4 += Matrix4`
: Returns the values of left Matrix4 plus the right Matrix4 to the left Matrix4.

`Matrix4--`
: Subtracts 1.0f from both values inside Matrix4.

`Matrix4 - Matrix4`
: Returns the values of left Matrix4 minus right Matrix4.

`Matrix4 -= Matrix4`
: Returns the values of left Matrix4 subtracted by right Matrix4 to the left Matrix4.

`Matrix4 * Matrix4`
: Returns the values of left Matrix4 multiplied right Matrix4.

`Matrix4 *= Matrix4`
: Returns the values of left Matrix4 multiplied by right Matrix4 to the left Matrix4.

`Matrix4 * float scalar`
: Returns the values of left Matrix4 multiplied by the float scalar.

`Matrix4 *= float scalar`
: Returns the values of left Matrix4 multiplied by the scalar toMatrix4.