ReadMe User

This contains the information for all the functions within the maths library.

/\*

\* class Vector2

\* This is a Vector2, which stores a 1d array of 2 values of type 'real' (defined in typeDef.h) within itself.

\* Within this Vector2, it stores functions which grants the ablity to be able to change values within the Vector2 itself.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Vector2();

\* This is the default constructor, it will create a default 1d array with 2 values of type 'real'.

\* This is created when a Vector2 is created with no parameters.

\*/

/\* Vector2(real newx, real newy);

\* This is a different Vector2 constructor, it will create a default 1d array with 2 values of type "real" with the parameters taken in.

\* This is created when a Vector2 is created with 2 parameters of type real.

@parameter 1 newx - a value of 'real' defined in typeDef.h

@parameter 2 newy - a value of 'real' defined in typeDef.h

\*/

/\* Vector2();

\* This is the default destructor, it will destroy a Vector2.

\* This is called when a the main program has reached the end of its scope.

\*/

/\* void Vector2::operator=(Vector4 &other);

\* This function overloads the = operator so that a Vector2 equal's a vector4.

\*

\* parameter 1 &other - a Vector4.

\* @returns void

\*/

/\* void Vector2::operator=(Vector3 &other);

\* This function overloads the = operator so that a Vector2 equal's a vector3.

\*

\* parameter 1 &other - a Vector3.

\* @returns void

\*/

/\* void Vector2::operator+=(const Vector2 &other);

\* This function overloads the += operator so that a Vector2's can add by itself followed by another Vector2.

\*

\* parameter 1 &other - a Vector2.

\* @returns Vector2

\*/

/\* Vector2 operator-(Vector2 &other);

\* This function overloads the - operator so that 2 Vector2's can be subtracted from each other.

\*

\* parameter 1 &other - a Vector2.

\* @returns Vector2

\*/

/\* void Vector2::operator-=(const Vector2 &other);

\* This function overloads the -= operator so that a Vector2's can subtract by itself followed by another Vector2.

\*

\* parameter 1 &other - a Vector2.

\* @returns Vector2

\*/

/\* void Vector2::operator\*=(const Vector2 &other);

\* This function overloads the \*= operator so that a Vector2's can multiply by itself followed by another Vector2.

\*

\* parameter 1 &other - a Vector2.

\* @returns Vector2

\*/

/\* Vector2 operator\*(real scalar);

\* This function overloads the \* operator so that a Vector2 can be multiplied by a real value.

\*

\* parameter 1 scalar - a 'real' value.

\* @returns Vector2

\*/

/\* friend Vector2 operator\*(real scalar, Vector2 &other);

\* This function overloads the \* operator so that a Vector2 can be equal to a 'real' value multiplied by a Vector2.

\*

\* parameter 1 scalar - a 'real' value.

\* parameter 2 &other - a Vector2.

\* @returns Vector2

\*/

/\*

\* class Vector3

\* This is a Vector3, which stores a 1d array of 3 values of type 'real' (defined in typeDef.h) within itself.

\* Within this Vector3, it stores functions which grants the ablity to be able to change values within the Vector3 itself.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Vector3();

\* This is the default constructor, it will create a default 1d array with 3 values of type 'real'.

\* This is created when a Vector3 is created with no parameters.

\*/

/\* Vector3(real newx, real newy, real newz);

\* This is a different Vector3 constructor, it will create a default 1d array with 3 values of type "real" with the parameters taken in.

\* This is created when a Vector3 is created with 3 parameters of type real.

@parameter 1 newx - a value of 'real' defined in typeDef.h

@parameter 2 newy - a value of 'real' defined in typeDef.h

@parameter 3 newz - a value of 'real' defined in typeDef.h

\*/

/\* Vector3 operator+(Vector3 &other);

\* This function overloads the + operator so that 2 Vector3's can be added together.

\*

\* parameter 1 &other - a Vector3.

\* @returns Vector3

\*/

/\* void Vector3::operator=(Vector4 &other);

\* This function overloads the = operator so that a Vector3 equal's a vector4.

\*

\* parameter 1 &other - a Vector4.

\* @returns void

\*/

/\* void Vector3::operator+=(const Vector3 &other);

\* This function overloads the += operator so that a Vector3's can add by itself followed by another Vector3.

\*

\* parameter 1 &other - a Vector3.

\* @returns Vector3

\*/

/\* Vector3 operator-(Vector3 &other);

\* This function overloads the - operator so that 2 Vector3's can be subtracted from each other.

\*

\* parameter 1 &other - a Vector3.

\* @returns Vector3

\*/

/\* void Vector3::operator-=(const Vector3 &other);

\* This function overloads the -= operator so that a Vector3's can subtract by itself followed by another Vector3.

\*

\* parameter 1 &other - a Vector3.

\* @returns Vector3

\*/

/\* void Vector3::operator\*=(const Vector3 &other);

\* This function overloads the \*= operator so that a Vector3's can multiply by itself followed by another Vector3.

\*

\* parameter 1 &other - a Vector3.

\* @returns Vector3

\*/

/\* Vector3 operator\*(real scalar);

\* This function overloads the \* operator so that a Vector3 can be multiplied by a real value.

\*

\* parameter 1 scalar - a 'real' value.

\* @returns Vector3

\*/

/\* friend Vector3 operator\*(real scalar, Vector3 &other);

\* This function overloads the \* operator so that a Vector3 can be equal to a 'real' value multiplied by a Vector3.

\*

\* parameter 1 scalar - a 'real' value.

\* parameter 2 &other - a Vector3.

\* @returns Vector3

\*/

/\* real dot(Vector3 &other);

\* This function returns a 'real' value that is based on the dot product of a inputted Vector3.

\*

\* parameter 1 &other - a Vector3.

\* @returns 'real'

\*/

/\* Vector3 cross(Vector3 &other);

\* This function returns a Vector3 that is based on the cross product of a Vector3.

\*

\* parameter 1 &other - a Vector3.

\* @returns Vector3

\*/

/\* real magnitude();

\* This function returns a 'real' value that is the magnitude of a Vector3.

\*

\* @returns 'real' (defined in typeDef.h)

\*/

/\* Vector3 normalised();

\* This function returns a Vector3 that has been normalised.

\*

\*

\* @returns Vector3

\*/

/\* void normalise();

\* This function normalises a Vector3.

\*

\*

\* @returns void

\*/

/\*

\* class Vector4

\* This is a Vector4, which stores a 1d array of 4 values of type 'real' (defined in typeDef.h) within itself.

\* Within this Vector4, it stores functions which grants the ablity to be able to change values within the Vector4 itself.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Vector4();

\* This is the default constructor, it will create a default 1d array with 4 values of tpye 'real'.

\* This is created when a Vector4 is created with no parameters.

\*/

/\* Vector4(real newx, real newy, real newz, real neww);

\* This is a different Vector4 constructor, it will create a default 1d array with 4 values of type "real" with the parameters taken in.

\* This is created when a Vector4 is created with 4 parameters of type real.

@parameter 1 newx - a value of 'real' defined in typeDef.h

@parameter 2 newy - a value of 'real' defined in typeDef.h

@parameter 3 newz - a value of 'real' defined in typeDef.h

@parameter 4 neww - a value of 'real' defined in typeDef.h

\*/

/\* Vector4();

\* This is the default destructor, it will destroy a Vector4.

\* This is called when a the main program has reached the end of its scope.

\*/

/\* Vector4 operator+(Vector4 &other);

\* This function overloads the + operator so that 2 Vector4's can be added together.

\*

\* parameter 1 &other - a Vector4.

\* @returns Vector4

\*/

/\* void Vector4::operator=(Vector2 &other);

\* This function overloads the = operator so that a Vector4 equal's a vector2.

\*

\* parameter 1 &other - a Vector2.

\* @returns void

\*/

/\* void Vector4::operator=(Vector3 &other);

\* This function overloads the = operator so that a Vector4 equal's a vector3.

\*

\* parameter 1 &other - a Vector3.

\* @returns void

\*/

/\* void Vector4::operator+=(const Vector4 &other);

\* This function overloads the += operator so that a Vector4's can add by itself followed by another Vector4.

\*

\* parameter 1 &other - a Vector4.

\* @returns Vector4

\*/

/\* Vector4 operator-(Vector4 &other);

\* This function overloads the - operator so that 2 Vector4's can be subtracted from each other.

\*

\* parameter 1 &other - a Vector4.

\* @returns Vector4

\*/

/\* void Vector4::operator-=(const Vector4 &other);

\* This function overloads the -= operator so that a Vector4's can subtract by itself followed by another Vector4.

\*

\* parameter 1 &other - a Vector4.

\* @returns Vector4

\*/

/\* void Vector4::operator\*=(const Vector4 &other);

\* This function overloads the \*= operator so that a Vector4's can multiply by itself followed by another Vector4.

\*

\* parameter 1 &other - a Vector4.

\* @returns Vector4

\*/

/\* Vector4 operator\*(real scalar);

\* This function overloads the \* operator so that a Vector4 can be multiplied by a real value.

\*

\* parameter 1 scalar - a 'real' value.

\* @returns Vector4

\*/

/\* friend Vector4 operator\*(real scalar, Vector4 &other);

\* This function overloads the \* operator so that a Vector4 can be equal to a 'real' value multiplied by a Vector4.

\*

\* parameter 1 scalar - a 'real' value.

\* parameter 2 &other - a Vector4.

\* @returns Vector4

\*/

/\* real dot(Vector4 &other);

\* This function returns a 'real' value that is based on the dot product of a inputted Vector4.

\*

\* parameter 1 &other - a Vector4.

\* @returns 'real'

\*/

/\* Vector4 cross(Vector4 &other);

\* This function returns a Vector4 that is based on the cross product of a Vector4.

\*

\* parameter 1 &other - a Vector4.

\* @returns Vector4

\*/

/\* real magnitude();

\* This function returns a 'real' value that is the magnitude of a Vector4.

\*

\* @returns 'real' (defined in typeDef.h)

\*/

/\* Vector4 normalised();

\* This function returns a Vector4 that has been normalised.

\*

\*

\* @returns Vector4

\*/

/\* void normalise();

\* This function normalises a Vector4.

\*

\*

\* @returns void

\*/

/\*

\* class Matrix2

\*

\* This is a matrix2, which stores a 2d array of 2 lots of vector2's together and is able to

\* store values of type "real" within itself. Within this matrix2, it stores functions

\* which grants the ablity to be able to change values within the matrix2 itself.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Matrix2();

\* This is the default constructor, it will create a default 2d array of type "real" with the idetity matrix of Matrix2.

\* This is created when a matrix2 is created with no parameters.

\*/

/\* Matrix2(real x1, real x2, real y1, real y2);

\* This is a different Matrix2 constructor, it will create a default 2d array of type "real" with the parameters taken in.

\* This is created when a matrix2 is created with 4 parameters of type real.

@parameter 1 x1 - a value of 'real' defined in typeDef.h

@parameter 2 x2 - a value of 'real' defined in typeDef.h

@parameter 3 y1 - a value of 'real' defined in typeDef.h

@parameter 4 y2 - a value of 'real' defined in typeDef.h

\*/

/\* Matrix2();

\* This is the default destructor, it will destroy a Matrix2.

\* This is called when a the main program has reached the end of its scope.

\*/

/\* Matrix2 operator\*(Matrix2 &mat1);

\* This function allows for the ablity to be able to multiply 2 Matrix2's together

\* in order to return a single Matrix2.

\*

\* @parameter 1 &mat1 - a Matrix2.

\*

\* @returns tempMatrix - a Matrix2 which is two Matrix2's multipled together.

\*/

/\* Vector2 Matrix2::operator\*(Vector2 &vec);

\* This function allows for the ablity to be able to multiply a Matrix2 by a Vector2

\* in order to return a single Vector2.

\*

\* @parameter 1 &vec - a Vector2.

\*

\* @returns tempVec - a Vector2 which is of a Vector2 and a Matrix2 multipled together.

\*/

/\* void Matrix2::setRotate(real rad);

\*

\* This function sets the rotation matrix of a Matrix2 if required based on the parameter taken in (which is a radiant).

\*

\* @parameter 1 - rad is of type real, which is defined in typeDef.h

\*

\* @returns void

\*/

/\* void Matrix2::identity();

\*

\* This function returns the identity matrix of a Matrix2.

\*

\* @returns void

\*/

/\* real Matrix2::determinant(real x1, real y1, real x2, real y2);

\* This is a function that will find the determinat of a matrix2 and return it as type 'real' as defined in typeDef.h.

\*

@parameter 1 x1 - a value of 'real' defined in typeDef.h

@parameter 2 x2 - a value of 'real' defined in typeDef.h

@parameter 3 y1 - a value of 'real' defined in typeDef.h

@parameter 4 y2 - a value of 'real' defined in typeDef.h

@ returns determinat - a value of type 'real'.

\*/

/\* void Matrix2::inverse();

\*

\* This function returns the inverse of a Matrix2.

\*

\* @returns void

\*/

/\*

\* class Matrix3

\*

\* This is a matrix3, which stores a 2d array of 3 lots of vector3's together and is able to

\* store values of type "real" within itself. Within this matrix3, it stores functions

\* which grants the ablity to be able to change values within the matrix3 itself.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Matrix3();

\* This is the default constructor, it will create a default 2d array of type "real" with the idetity matrix of Matrix3.

\* This is created when a matrix3 is created with no parameters.

\*/

/\* Matrix3(real x1, real x2, real x3, real y1, real y2, real y3, real z1, real z2, real z3);

\* This is a different Matrix3 constructor, it will create a default 3d array of type "real" with the parameters taken in.

\* This is created when a matrix2 is created with 9 parameters of type real.

@parameter 1 x1 - a value of 'real' defined in typeDef.h

@parameter 2 x2 - a value of 'real' defined in typeDef.h

@parameter 3 x3 - a value of 'real' defined in typeDef.h

@parameter 4 y1 - a value of 'real' defined in typeDef.h

@parameter 5 y2 - a value of 'real' defined in typeDef.h

@parameter 6 y3 - a value of 'real' defined in typeDef.h

@parameter 7 z1 - a value of 'real' defined in typeDef.h

@parameter 8 z2 - a value of 'real' defined in typeDef.h

@parameter 9 z9 - a value of 'real' defined in typeDef.h

\*/

/\* Matrix3();

\* This is the default destructor, it will destroy a Matrix3.

\* This is called when a the main program has reached the end of its scope.

\*/

/\* real Matrix3::determinant();

\* This is a function that will find the determinat of a matrix3 and return it as type 'real' as defined in typeDef.h.

\*

@ returns determinat - a value of type 'real'.

\*/

/\* void Matrix3::inverse();

\*

\* This function returns the inverse of a Matrix3.

\*

\* @returns void

\*/

/\* Matrix3 operator\*(Matrix3 &mat1);

\* This function allows for the ablity to be able to multiply 2 Matrix3's together

\* in order to return a single Matrix3.

\*

\* @parameter 1 &mat1 - a Matrix3.

\*

\* @returns tempMatrix - a Matrix3 which is two Matrix3's multipled together.

\*/

/\* Vector3 Matrix3::operator\*(Vector3 &vec);

\* This function allows for the ablity to be able to multiply a Matrix3 by a Vector3

\* in order to return a single Vector3.

\*

\* @parameter 1 &vec - a Vector3.

\*

\* @returns tempVec - a Vector3 which is of a Vector2 and a Matrix2 multipled together.

\*/

/\* Vector2 Matrix3::operator\*(Vector2 &vec);

\* This function allows for the ablity to be able to multiply a Matrix3 by a Vector2

\* in order to return a single Vector2.

\*

\* @parameter 1 &vec2 - a Vector2.

\*

\* @returns tempVec - a Vector2 which is of a Vector2 and a Matrix2 multipled together.

\*/

/\* void Matrix3::setRotateX(real x);

\*

\* This function sets the rotation matrix of a Matrix3 if required based on the parameter taken in (which is a radiant) and affects the x axis.

\*

\* @parameter 1 - x is of type real, which is defined in typeDef.h

\*

\* @returns void

\*/

/\* void Matrix3::setRotateY(real y);

\*

\* This function sets the rotation matrix of a Matrix3 if required based on the parameter taken in (which is a radiant) and affects the y axis.

\*

\* @parameter 1 - y is of type real, which is defined in typeDef.h

\*

\* @returns void

\*/

/\* void Matrix3::setRotateZ(real z);

\*

\* This function sets the rotation matrix of a Matrix3 if required based on the parameter taken in (which is a radiant) and affects the z axis.

\*

\* @parameter 1 - z is of type real, which is defined in typeDef.h

\*

\* @returns void

\*/

/\* void Matrix3::identity();

\*

\* This function returns the identity matrix of a Matrix3.

\*

\* @returns void

\*/

/\*

\* class Matrix4

\*

\* This is a matrix4, which stores a 2d array of 4 vector4's together and is able to

\* store values of type "real" within itself. Within this matrix4, it stores functions

\* which grants the ablity to be able to change values within the matrix4 itself.

\* @author - Zackary Direen, Academy of Interactive Entertainment, 2017

\*/

/\* Matrix4();

\* This is the default constructor, it will create a default 2d array of type "real" with the idetity matrix of Matrix4.

\* This is created when a matrix3 is created with no parameters.

\*/

/\* Matrix4(real x1, real x2, real x3, real x4, real y1, real y2, real y3, real y4, real z1. real z2, real z3, real z4, real w1, real w2, real w3, real w4);

\* This is a different Matrix3 constructor, it will create a default 3d array of type "real" with the parameters taken in.

\* This is created when a matrix2 is created with 9 parameters of type real.

@parameter 1 x1 - a value of 'real' defined in typeDef.h

@parameter 2 x2 - a value of 'real' defined in typeDef.h

@parameter 3 x3 - a value of 'real' defined in typeDef.h

@parameter 4 x4 - a value of 'real' defined in typeDef.h

@parameter 5 y1 - a value of 'real' defined in typeDef.h

@parameter 6 y2 - a value of 'real' defined in typeDef.h

@parameter 7 y3 - a value of 'real' defined in typeDef.h

@parameter 8 y4 - a value of 'real' defined in typeDef.h

@parameter 9 z1 - a value of 'real' defined in typeDef.h

@parameter 10 z2 - a value of 'real' defined in typeDef.h

@parameter 11 z3 - a value of 'real' defined in typeDef.h

@parameter 12 z4 - a value of 'real' defined in typeDef.h

@parameter 13 w1 - a value of 'real' defined in typeDef.h

@parameter 14 w2 - a value of 'real' defined in typeDef.h

@parameter 15 w3 - a value of 'real' defined in typeDef.h

@parameter 16 w4 - a value of 'real' defined in typeDef.h

\*/

/\* Matrix4();

\* This is the default destructor, it will destroy a Matrix4.

\* This is called when a the main program has reached the end of its scope.

\*/

/\* real Matrix4::determinant();

\* This is a function that will find the determinat of a matrix4 and return it as type 'real' as defined in typeDef.h.

\*

@ returns determinat - a value of type 'real'.

\*/

/\* void Matrix4::inverse();

\*

\* This function returns the inverse of a Matrix4.

\*

\* @returns void

\*/

/\* Matrix4 operator\*(Matrix4 &mat1);

\* This function allows for the ablity to be able to multiply 2 Matrix4's together

\* in order to return a single Matrix4.

\*

\* @parameter 1 &mat1 - a Matrix4.

\*

\* @returns tempMatrix - a Matrix4 which is two Matrix4's multipled together.

\*/

/\* Vector4 Matrix4::operator\*(Vector4 &vec);

\* This function allows for the ablity to be able to multiply a Matrix4 by a Vector4

\* in order to return a single Vector4.

\*

\* @parameter 1 &vec - a Vector4.

\*

\* @returns tempVec - a Vector4 which is of a Vector4 and a Matrix4 multipled together.

\*/

/\* void Matrix4::setRotateX(real x);

\*

\* This function sets the rotation matrix of a Matrix4 if required based on the parameter taken in (which is a radiant) and affects the x axis.

\*

\* @parameter 1 - x is of type real, which is defined in typeDef.h

\*

\* @returns void

\*/

/\* void Matrix4::setRotateY(real y);

\*

\* This function sets the rotation matrix of a Matrix4 if required based on the parameter taken in (which is a radiant) and affects the y axis.

\*

\* @parameter 1 - y is of type real, which is defined in typeDef.h

\*

\* @returns void

\*/

/\* void Matrix4::setRotateZ(real z);

\*

\* This function sets the rotation matrix of a Matrix4 if required based on the parameter taken in (which is a radiant) and affects the z axis.

\*

\* @parameter 1 - z is of type real, which is defined in typeDef.h

\*

\* @returns void

\*/

/\* void Matrix4::identity();

\*

\* This function returns the identity matrix of a Matrix4.

\*

\* @returns void

\*/

/\* operator Vector2();

\*

\* This function allows a Vector3 to be used as a Vector2.

\*

\* @returns Vector4(x,y)

\*/

/\* operator Vector4();

\*

\* This function allows a Vector3 to be used as a Vector4.

\*

\* @returns Vector4(x,y,z,0)

\*/

/\* operator Vector3();

\*

\* This function allows a Vector4 to be used as a Vector3.

\*

\* @returns Vector3(x,y,z)

\*/

/\* operator Vector2();

\*

\* This function allows a Vector4 to be used as a Vector2.

\*

\* @returns Vector2(x,y)

\*/

/\* operator Vector3();

\*

\* This function allows a Vector2 to be used as a Vector3.

\*

\* @returns Vector3(x,y,0)

\*/

/\* operator Vector3();

\*

\* This function allows a Vector2 to be used as a Vector4.

\*

\* @returns Vector4(x,y,0,0)

\*/