TITAN FORCE ENGINE

Scripting API & Manual

Created by: Malik Allen

Math Namespace

Definitions:

Definitions of const variables used throughout the Math Library.

VERY_SMALL	1.0e-7f
PI	3.14159265358979323846f
DEGREES_TO_RADIANS	PI / 180.0f
RADIANS_TO_DEGREES	180.0f / PI

Vector 2

Vector2 is a structure representing a 2D vector point using float point values.

Member Variables:

float	x, y

Constructors:

Vector2 (float s = float(0.0))	Default Constructor that populates the x and y variables with the passed argument, if no argument, Vector2 is loaded with a default of zero.
Vector2 (float x_, float y_)	Constructor that takes x and y as arguments, and populates this Vector2's variables with the passed parameters.
Vector2(const Vector2& v)	A copy constructor. Copying the value of the Vector2 at the passed address and copying those value to this Vector2.

Operators:

Operator= (const Vector2& v)	Assignment operator.
Operator+ (const Vector2& v)	Addition Operator.
Operator- (const Vector2& v)	Subtraction Operator.
Operator+= (const Vector2& v)	Addition and then assignment onto self.
Operator-= (const Vector2& v)	Subtraction and then assignment onto self.
Operator- ()	Returns a negative of a Vector2.
Operator * (const float s)	Multiplication of a Vector2 by a scalar value.
Operator *= (const float s)	Multiplication of a Vector2 by a scalar value and then assignment onto self.
Operator / (const float s)	Division of a Vector2 by a scalar value.
Operator /= (const float s)	Division of a Vector2 by a scalar value and then assignment onto self.
Operator [] (int index)	Allows Vector2 to be written and read like an array.

Member Functionality:

void Load (float x_, float y_)	Utility used to populate x and y variables.
float magnitude ()	Returns the length of this Vector2.
Vector2 normalized ()	Returns this Vector2 with a magnitude of 1.
void Normalized ()	Gives this Vector2 a magnitude of 1, by dividing each component by its magnitude.
void Scale (const Vector2& v)	Multiplies this Vector2 and the passed Vector2 component-wise.

Static Vector Math Functionality:

float Dot (const Vector2& v1, const Vector2& v2)	Returns the dot product between Vector2 v1 and v2.
float Angle (const Vector2& a, const Vector2& b)	Returns the angle from Vector2 a to Vector2 b (angle is returned in radians).
Vector2 Lerp (const Vector2& start, const Vector2& finish, float t)	Preforms Linear Interpolation from Vector2 start to Vector2 finish at a rate of float t. (When t = 0, then the Vector2 = start. When t = 1, then the Vector2 = finish. When t= 0.5, then the Vector2 is the halfway point between the start and finish.)
Vector2 Reflect (const Vector2& init, const Vector2& normal)	Returns the reflection Vector2 init reflected by the Vector2 normal.

VECTOR 3

Vector3 is a structure representing a 3D vector point using floating point values.

Along with the listed functions other classes may manipulate and use the Vector3 structure.

Member Variables:

float	x, y, z

Constructors:

Vector3 (float s = float(0.0))	Default Constructor that populates the x, y, and z variables with the passed argument, if no argument, Vector3 loads all components with a value of zero.
Vector3 (float x_, float y_, float z_)	Constructor that takes x, y, and z as arguments, and populates this Vector3's variables with the passed parameters.
Vector3(const Vector3& v)	A copy constructor. Copying the value of the vector at the passed address and copying those value to this Vector3.

Operators:

Operator= (const Vector3& v)	Assignment operator.
Operator+ (const Vector3& v)	Addition Operator.
Operator- (const Vector3& v)	Subtraction Operator.
Operator+= (const Vector3& v)	Addition and then assignment onto self.
Operator-= (const Vector3& v)	Subtraction and then assignment onto self.
Operator- ()	Returns a negative of a Vector3.
Operator * (const float s)	Multiplication of a Vector3 by a scalar value.
Operator *= (const float s)	Multiplication of a Vector3 by a scalar value and then assignment onto self.
Operator / (const float s)	Division of a Vector3 by a scalar value.
Operator /= (const float s)	Division of a Vector3 by a scalar value and then assignment onto self.
Operator [] (int index)	Allows Vector3 to be written and read like an array.

Member Functionality:

void Load (float x_, float y_, float z_)	Utility used to populate x, y, and z variables.
float magnitude ()	Returns the length of this Vector3.
Vector3 normalized ()	Returns this Vector3 with a magnitude of 1.
void Normalize ()	Gives this Vector3 a magnitude of 1 by dividing its components by its original magnitude.
void Scale (const Vector3& v)	Multiplies this Vector3 and the passed Vector3 component-wise.

Static Vector Math Functionality:

Vector3 normalize (const Vector3& v)	Returns the passed Vector3 as a Vector3 with a magnitude of 1.
float Dot (const Vector3& v1, const Vector3& v2)	Returns the dot product between Vector3 v1 and v2.
float Distance (const Vector3& v1, const Vector3& v2)	Returns the distance between Vector3 v1 and Vector3 v2.
float Angle (const Vector3& a, const Vector3& v2)	Returns the angle from Vector3 a to Vector3 b, angle returned is in radians.
Vector3 Lerp (const Vector3& start, const Vector3& finish, float t)	Preforms linear interpolation from Vector3 start to Vector3 finish at the rate of float t. (When t = 0, then the Vector3 is at start. When t = 1, then the Vector3 is at the finish. When t = 0.5 then the Vector3 is at the halfway point between the start and finish.).
Vector3 Reflect (const Vector3& v, const Vector3& normal)	Returns the reflection Vector3 v reflected by the Vector3 normal.
Vector3 Project (const Vector3& u, const Vector3& v)	Returns the projection of Vector3 u onto Vector3 v.
Vector3 SLerp (const Vector3& start, const Vector3& finish, float t)	Preforms the spherical linear interpolation between Vector3 start and Vector3 finish at the rate of float t. (When t = 0, then the Vector3 is at start. When t = 1, then the Vector3 is at the finish. When t = 0.5 then the Vector3 is at the halfway point between the start and finish.).

VECTOR 4

Vector4 is a structure representing a 4D vector point. This structure inherits the properties of Vector3.

Member Variables:

float	x, y, z, w

Constructors:

Vector4 (float s = float(0.0))	Default Constructor that populates the x, y, z, and w variables with the passed argument, if no argument, Vector4 loads all components with a value of zero.
Vector4 (float x_, float y_, float z_, float w_)	Constructor that takes x, y, z, and w as arguments, and populates this Vector4's variables with the passed parameters.
Vector4(const Vector4& v)	A copy constructor. Copying the value of the vector at the passed address and copying those value to this Vector4.

Operators:

Operator= (const Vector4& v)	Assignment operator.
Operator+ (const Vector4& v)	Addition Operator.
Operator- (const Vector4& v)	Subtraction Operator.
Operator+= (const Vector4& v)	Addition and then assignment onto self.
Operator-= (const Vector4& v)	Subtraction and then assignment onto self.
Operator- ()	Returns a negative of a Vector4.
Operator * (const float s)	Multiplication of a Vector4 by a scalar value.
Operator *= (const float s)	Multiplication of a Vector4 by a scalar value and then assignment onto self.
Operator / (const float s)	Division of a Vector4 by a scalar value.
Operator /= (const float s)	Division of a Vecto4 by a scalar value and then assignment onto self.
Operator [] (int index)	Allows Vector4 to be written and read like an array.

Member Functionality:

void Load (float x_, float y_, float z_, float w_)	Utility used to populate x, y, z, and w variables.
float magnitude ()	Returns the length of this Vector4.
Vector4 normalized ()	Returns this Vector4 with a magnitude of 1.
void Normalize ()	Gives this Vector4 a magnitude of 1 by dividing its components by its original magnitude.
void Scale (const Vector4& v)	Multiplies this Vector4 and the passed Vector4 component-wise.

Static Vector Math Functionality:

float Dot (const Vector4& v1, const Vector4& v2)	Returns the dot product between Vector4 v1 and v2.
float Distance (const Vector4& v1, const Vector4& v2)	Returns the distance between Vector4 v1 and Vector4 v2.
Vector4 Lerp (const Vector4& start, const Vector4& finish, float t)	Preforms linear interpolation from Vector4 start to Vector4 finish at the rate of float t. (When t = 0, then the Vector4 is at start. When t = 1, then the Vector4 is at the finish. When t = 0.5 then the Vector4 is at the halfway point between the start and finish.).
Vector4 Project (const Vector4& u, const Vector4& v)	Returns the projection of Vector4 u onto Vector4 v.

Static Vector Math Class

This static class uses the Vector2, Vector3, and Vector4 to calculate the more complex vector operations.

Static Functionality:

float Dot (const Vector2& v1, const Vector2& v2)	Returns the dot product between Vector2 v1 and v2.
float Angle (const Vector2& a, const Vector2& b)	Returns the angle from Vector2 a to Vector2 b (angle is returned in radians).
Vector2 Lerp (const Vector2& start, const Vector2& finish, float t)	Preforms Linear Interpolation from Vector2 start to Vector2 finish at a rate of float t. (When t = 0, then the Vector2 = start. When t = 1, then the Vector2 = finish. When t= 0.5, then the Vector2 is the halfway point between the start and finish.)
Vector2 Reflect (const Vector2& init, const Vector2& normal)	Returns the reflection Vector2 init reflected by the Vector2 normal.
Vector3 normalize (const Vector3& v)	Returns the passed Vector3 as a Vector3 with a magnitude of 1.
float Dot (const Vector3& v1, const Vector3& v2)	Returns the dot product between Vector3 v1 and v2.
float Distance (const Vector3& v1, const Vector3& v2)	Returns the distance between Vector3 v1 and Vector3 v2.
float Angle (const Vector3& a, const Vector3& v2)	Returns the angle from Vector3 a to Vector3 b, angle returned is in radians.
Vector3 Lerp (const Vector3& start, const Vector3& finish, float t)	Preforms linear interpolation from Vector3 start to Vector3 finish at the rate of float t. (When t = 0, then the Vector3 is at start. When t = 1, then the Vector3 is at the finish. When t = 0.5 then the Vector3 is at the halfway point between the start and finish.).
Vector3 Reflect (const Vector3& v, const Vector3& normal)	Returns the reflection Vector3 v reflected by the Vector3 normal.
Vector3 Project (const Vector3& u, const Vector3& v)	Returns the projection of Vector3 u onto Vector3 v.
Vector3 SLerp (const Vector3& start, const Vector3& finish, float t)	Preforms the spherical linear interpolation between Vector3 start and Vector3 finish at the rate of float t. (When t = 0, then the Vector3 is at start. When t = 1, then the Vector3 is at the finish. When t = 0.5 then the Vector3 is at the halfway point between the start and finish.).
float Dot (const Vector4& v1, const Vector4& v2)	Returns the dot product between Vector4 v1 and v2.
float Distance (const Vector4& v1, const Vector4& v2)	Returns the distance between Vector4 v1 and Vector4 v2.
Vector4 Lerp (const Vector4& start, const Vector4& finish, float t)	Preforms linear interpolation from Vector4 start to Vector4 finish at the rate of float t. (When t = 0, then the Vector4 is at start. When t = 1, then the Vector4 is at the finish. When t = 0.5 then the Vector4 is at the halfway point between the start and finish.).
Vector4 Project (const Vector4& u, const Vector4& v)	Returns the projection of Vector4 u onto Vector4 v.

Matrix 3

Matrix3 is a class composed are an array of 9 floating point values. The values are used to represent a **column major** 3x3 Matrix.

All Matrix Linear Transformations are applied with the Matrix4 Class to account for *affine transformations*, then if needed you can extract a Matrix3 from a Matrix4 to work with the transformed matrix.

Member Variables:

float	m[16]

Constructors:

Matrix3 (float d = float(0.0))	A default constructor, when the passed parameter is zero, an identity matrix populates the Matrix3 by default, otherwise it will populate all components of the Matrix3 with the value of d.
Matrix3 (float xx, float xy, float xz, float yx, float yy, float yz, float zx, float zy, float zz)	Constructor used to populate each number in the Matrix3.
Matrix3 (const Matrix4& m)	A copy construct that extracts the inner 3x3 Matrix from a Matrix4.

Operators:

Operator [] (int index)	Allows Matrix3 to be read and written as an array.
Operator = (const Matrix3& m)	Assignment operator.
Operator * (const Matrix3& n)	Preforms multiplication between two Matrix3.
Operator *= (const Matrix3& m)	Preforms matrix multiplication and assignment onto self.
Vector3 Operator * (const Vector3& v)	Preforms multiplication between a 3D vector and a 3x3 Matrix and returns the resultant vector.
Operator = (const Matrix4& m)	Extracts the inner 3x3 Matrix from the 4x4 Matrix using the assignment operator.

Member Functions:

Load (float xx, float xy, float xz, float yx, float yy, float yz, float zx, float zy, float zz)	Utility used to populate a Matrix3
Load Identity ()	Utility used to load Matrix3 with a Identity Matrix.

Matrix 4

Matrix4 is a class composed are an array of 16 floating point values. The values are used to represent a **column major** 4x4 Matrix.

All Linear Transformations of Matrices are applied with the Matrix4 class. We are using the *right-hand rule*. Keep in mind that clock-wise rotation about an axis is negative, counter clock-wise rotation negative, our positive z-axis is pointing away from the screen and the negative towards it.

Member Variables:

float	m[16]

Constructors:

Matrix4 (float x0, float x1, float x2, float x3, float y0, float y1, float y2, float y3, float z0, float z1, float z2, float z3, float w0, float w1, float w2, float w3)	A constructor used to populate each element in the array of numbers representing a Matrix4.
Matrix4 (float d = 1.0f)	This constructor will load all elements in the Matrix4 array with the passing parameter. When no parameter or a parameter of 1 is given, an identity matrix is loaded.

Operators:

Operator = (const Matrix4& m)	Assignment operator.
Operator * (const Matrix4& m)	Preforms Matrix multiplication between two Matrix4
Operator *= (const Matrix4& m)	Preforms Matrix multiplication between two Matrix4, then assignment onto self.
Vector4 Operator * (const Vector4& v)	Preforms multiplication between a Vector4 and a Matrix4 returning the resultant Vector4.
Operator [] (int index)	Allows the components of this Matrix4 to read and written like an array.
Vector3 Operator * (const Vector3& v)	Preforms multiplication between a Vector3 and a Matrix4 returning the resultant Vector3.

Static Matrix Math Class

The uses of the Matrix4 class for Linear Transformations of Matrices. Remember we are using the *right-hand rule*.

Matrix4 Rotate (float degrees, float x, float y, float z)	Creates a rotation Matrix allowing rotation about any arbitrary axis, provided the value of the axis is greater than zero, and preferably 1.
Matrix4 Rotate (float degrees, const Vecotr3& axis)	Creates a rotation Matrix allowing rotation about any arbitrary axis, this time take the axis as a Vector3.
Matrix4 Scale (float x, float y, float z)	Creates a scaling matrix scaling the values of x, y, and z with the passed parameters respectively across each component.
Matrix4 Scale (const Vector3& scale)	Creates a scaling matrix scaling the values of x, y, and z with the components of the passed Vector3 respectively across each component.
Matrix4 Translate (float x, float y, float z)	Creates a Translation Matrix, that will translate in the direction of the passed components.
Matrix4 Translate (const Vecotr3& v)	Creates a Translation Matrix, that will translate in the direction of the passed Vector3.
Matrix4 Perspective (float fovy, float aspect, float zNear, float zFar)	Creates a Perspective Projection Matrix/ Viewing Frustum, float fovy is the degrees of view along y, the aspect is the screen's height divided by the screen's width and the zNear is the distance between the eye and the near Z plane and zFar is the distance between the eye and the far z plane.
Matrix4 Othographic(float xMin, float xMax, float yMin, float yMax, float zMin, float zMax)	Creates an Orthographic Projection Matrix/ View Frustum using the passed min and max values of each component that will be used to define the bounds of the Viewing Frustum.
Matrix4 ViewportNDC (int width, int height)	Creates the Normalized Device Coordinate Matrix, using the width and height passed as parameters. Keep in mind that this Matrix follows the *left-hand rule.* (Used for OpenGL).
Matrix4 Transpose (const Matrix4& m)	Creates a transpose of the passed Matrix4, returning the matrix with its columns and rows switched.
Matrix4 LookAt(float eyeX, float eyeY, float eyeZ, float atX, float atY, float atZ, float upX, float upY, float upZ)	Creates the eye, look at position and up direction of the camera.
Matrix4 LookAt(const Vector3& eye, const Vector3& at, const Vector3& up)	Creates the eye, look at position and up direction of the camera, taking all above as Vector3s.
Matrix4 Inverse (const Mtrix4& m)	Creates the inverse of a the passed Matrix4.

Euler Angles

The Euler struct is a method of representing orientation. Euler inherits its functionality from the Vector3 struct.

Member Variables:

float	x, y, z

Constructors:

Euler (float a = float(0.0))	Default Constructor that populates the x, y, and z variables with the passed argument, if no argument, Euler loads all components with a value of zero.
Euler (float pitch, float yaw, float roll)	Constructor that takes x, y, and z as arguments, and populates this Euler's variables with the passed parameters.
Euler (const Euler& e)	A copy constructor. Copying the value of the euler at the passed address and copying those value to this Euler.

Operators:

Operator= (const Euler& e)	An assignment operator.

Quaternion

The Quaternion struct is a method to represent rotations. Quaternions interpolate between angles much easier, and do not suffer from gimbal lock. The Quaternion struct inherits directly from the Vector4 struct.

Member Variables:

float	x, y, z, w

Constructors:

Quaternion (float s = float(0.0))	Default Constructor that populates the x, y, z, and w variables with the passed argument, if no argument, Quaternion loads all components with a value of zero.
Quaternion (float x, float y, float z, float w)	Constructor that takes x, y, z, and w as arguments, and populates this Quaternion's variables with the passed parameters.
Quaternion (const Vector3& axis, float degrees)	Constructor takes a unit Vector3 as an axis of rotation and a float as the degrees of rotation along passed axis.
Quaternion (const Quaternion& q)	A copy constructor.

Operators:

Operator=(const Quaternion& q)	An assignment operator.
Operator+(const Quaternion& q)	Addition operator.
Operator * (const Quaternion& q)	A Quaternion Multiplication operator.
Operator * (const float t)	Preforms multiplication between a Quaternion and a scalar.
Operator *= (const float t)	Preforms multiplication between a Quaternion and a scalar and assignment onto self.

Member Functionality:

Conjugated()	Returns the conjugate of this Quaternion.
Euler()	Returns the representation of this Quaternion as Euler angles.
Normalized()	Returns this Quaternion as a unit Quaternion.

Static Quaternion Math Class

The Quaternion Math Class preforms a variety of operations to a Quaternion.

Member Functions:

Quaternion Inverse(const Quaternion& q)	Calculates the normalized inverse of the passed Quaternion.
Quaternion Conjugate(const Quaternion &q)	Returns the conjugate of the passed Quaternion.
float Dot(const Quaternion& q1, const Quaternion& q2)	Calculates the dot product between two Quaternions.
Quaternion Normalize(const Quaternion& q)	Returns the unit Quaternion.
float Angle (const Quaternion& q1, const Quaternion& q2)	Calculates the angle starting from q1 to q2.
Vector3 Rotate (const Quaternion& q, const Vector3& v)	Calculates the rotation applied to a Vector3 by the passed Quaternion.
Quaternion Lerp (const Quaternion& start, const Quaternion& end, float t)	Preforms linear interpolation from Quaternion start to Quaternion end at the rate of float t. (When t = 0, then the Quaternion is at start. When t = 1, then the Quaternion is at the end. When t = 0.5 then the Quaternion is at the halfway point between the start and Quaternion.).
Quaternion SLerp(const Quaternion& start, const Quaternion end, float t)	Preforms the spherical linear interpolation between Quaternion start and Quaternion finish at the rate of float t. (When t = 0, then the Quaternion is at start. When t = 1, then the Quaternion is at the end. When t = 0.5 then the Quaternion is at the halfway point between the start and end.).

Static Rotation Conversion Class

This class is used to convert between the three different methods of representing orientation. Euler Angles, Matrices and Quaternions.

Matrix4 EulerToMatrix(const Euler& e)	Converts the representation of a Euler to a Matrix4.
Euler MatrixToEuler(const Matrix4& m)	Converts the representation of a Matrix4 to a Euler.
Matrix4 QuaternionToMatrix(const Quaternion& q)	Converts the representation of a Quaternion to a Matrix4.
Quaternion MatrixToQuaternion(const Matrix4& m)	Converts the representation of a Matrix4 to a Quaternion.
Euler QuaternionToEuler(const Quaternion& q)	Converts the representation of a Quaternion to a Euler.
Quaternion EulerToQuaternion(const Euler& q)	Converts the representation of a Euler to a Quaternion.