**Luke Plaisance**

**12/5/17**

**Assessment : 2 Math Library**

1. **Requirements Documentation**

**1.1 Description of the Problem**

**Name :** Math Library

**Problem Statement :** Create a math library with various functions for Vectors and Matrices.

**Problem Specification :**  Have the library with the functions linked to the bootstrap project

**2. System Architecture**

**Vector2 Class:**

**float mX**

x value

**Float mY**

y value

**Vector2()**

**Prototype** : Vector2()

**Argument** : none

**Description** : makes an instance of the Vector2 class

**Precondition** : none

**Postcondition** : Constructs the Vector2 instance

**Protection Level** : Public

**Vector2(float x, float y)**

**Prototype** : Vector2(float x, float y)

**Argument** : float x, float y

**Description** : sets the X and Y value for the vector

**Precondition** : an instance of the Vector2 class

**Postcondition** : sets the X and Y value for the vector

**Protection Level** : Public

**Vector2 operator+(Vector2 & other)**

**Prototype** : Vector2 operator+(Vector2 & other)

**Argument** : Vector2 & other

**Description** : adds two vectors together

**Precondition** : an instance of the Vector2 class

**Postcondition** : returns the sum of two vectors

**Protection Level** : Public

**Vector2 operator-(Vector2 & other)**

**Prototype** : Vector2 operator-(Vector2 & other)

**Argument** : Vector2 & other

**Description** : subtracts two vectors

**Precondition** : an instance of the Vector2 class

**Postcondition** : returns the difference of two vectors

**Protection Level** : Public

**Vector2 operator\*(Vector2 & other)**

**Prototype** : Vector2 operator\*(float other)

**Argument** : float other

**Description** : multiplies two vectors

**Precondition** : an instance of the Vector2 class

**Postcondition** : returns the product of two vectors

**Protection Level** : Public

**bool operator==(Vector2 & other)**

**Prototype** : bool operator==(Vector2 & other)

**Argument** : Vector2 & other

**Description** : checks to see if two vectors are equal

**Precondition** : an instance of the Vector2 class

**Postcondition** : checks to see if two vectors are equal

**Protection Level** : Public

**float Dot(Vector2 & other)**

**Prototype** : float Dot(Vector2 & other)

**Argument** : Vector2 & other

**Description** : finds the dot product of the vector

**Precondition** : an instance of the Vector2 class

**Postcondition** : finds the dot product of the vector

**Protection Level** : Public

**Vector2 Normalize()**

**Prototype** : Vector2 Normalize()

**Argument** : none

**Description** : normalizes the vector after finding the magnitude

**Precondition** : an instance of the Vector2 class

**Postcondition** : normalizes the vector

**Protection Level** : Public

**float Magnitude()**

**Prototype** : float Magnitude()

**Argument** : none

**Description** : finds the magnitude of the vector

**Precondition** : an instance of the Vector2 class

**Postcondition** : finds the magnitude

**Protection Level** : Public

**friend ostream& operator<<(ostream& os, const Vector2 vector)**

**Prototype** : friend ostream& operator<<(ostream& os, const Vector2 vector)

**Argument** : ostream& os, const Vector2 vector

**Description** : outputs the X and Y value to the console

**Precondition** : an instance of the Vector2 class

**Postcondition** : outputs the X and Y value to the console

**Protection Level** : Public

**friend istream& operator<<(istream& is, const Vector2 vector)**

**Prototype** : friend istream& operator<<(istream& is, const Vector2 vector)

**Argument** : ostream& os, const Vector2 vector

**Description** : takes in the X and Y value via user

**Precondition** : an instance of the Vector2 class

**Postcondition** : takes in the X and Y value via user

**Protection Level** : Public

**Vector 3 Class :**

**float mX**

x value

**float mY**

y value

**float mZ**

z value

**Vector3()**

**Prototype** : Vector3()

**Argument** : none

**Description** : makes an instance of the Vector3 class

**Precondition** : none

**Postcondition** : Constructs the Vector3 instance

**Protection Level** : Public

**Vector3(float x, float y, float z)**

**Prototype** : Vector2(float x, float y, float z)

**Argument** : float x, float y, float z

**Description** : sets the X, Y, and Z value for the vector

**Precondition** : an instance of the Vector3 class

**Postcondition** : sets the X, Y, and Z value for the vector

**Protection Level** : Public

**Vector3 operator+(Vector3 & other)**

**Prototype** : Vector3 operator+(Vector3 & other)

**Argument** : Vector3 & other

**Description** : adds two vectors together

**Precondition** : an instance of the Vector3 class

**Postcondition** : returns the sum of two vectors

**Protection Level** : Public

**Vector3 operator-(Vector3 & other)**

**Prototype** : Vector3 operator-(Vector3 & other)

**Argument** : Vector3 & other

**Description** : subtracts two vectors

**Precondition** : an instance of the Vector3 class

**Postcondition** : returns the difference of two vectors

**Protection Level** : Public

**Vector3 operator\*(Vector3 & other)**

**Prototype** : Vector3 operator\*(float other)

**Argument** : float other

**Description** : multiplies two vectors

**Precondition** : an instance of the Vector3 class

**Postcondition** : returns the product of two vectors

**Protection Level** : Public

**bool operator==(Vector3 & other)**

**Prototype** : bool operator==(Vector3 & other)

**Argument** : Vector3 & other

**Description** : checks to see if two vectors are equal

**Precondition** : an instance of the Vector3 class

**Postcondition** : checks to see if two vectors are equal

**Protection Level** : Public

**float Dot(Vector3 & other)**

**Prototype** : float Dot(Vector3 & other)

**Argument** : Vector3 & other

**Description** : finds the dot product of the vector

**Precondition** : an instance of the Vector3 class

**Postcondition** : finds the dot product of the vector

**Protection Level** : Public

**Vector3 Normalize()**

**Prototype** : Vector3 Normalize()

**Argument** : none

**Description** : normalizes the vector after finding the magnitude

**Precondition** : an instance of the Vector3 class

**Postcondition** : normalizes the vector

**Protection Level** : Public

**Vector3 Cross(Vector3 & other)**

**Prototype** : Vector3 Cross(Vector3 & other);

**Argument** : Vector3 & other

**Description** : finds the cross product of the vector

**Precondition** : an instance of the Vector3 class

**Postcondition** : finds the cross product

**Protection Level** : Public

**float Magnitude()**

**Prototype** : float Magnitude()

**Argument** : none

**Description** : finds the magnitude of the vector

**Precondition** : an instance of the Vector3 class

**Postcondition** : finds the magnitude

**Protection Level** : Public

**friend ostream& operator<<(ostream& os, const Vector3 vector)**

**Prototype** : friend ostream& operator<<(ostream& os, const Vector3 vector)

**Argument** : ostream& os, const Vector3 vector

**Description** : outputs the X, Y, and Z value to the console

**Precondition** : an instance of the Vector3 class

**Postcondition** : outputs the X,Y, and Z value to the console

**Protection Level** : Public

**friend istream& operator<<(istream& is, const Vector3 vector)**

**Prototype** : friend istream& operator<<(istream& is, const Vector3 vector)

**Argument** : ostream& os, const Vector3 vector

**Description** : takes in the X,Y, and Z value via user

**Precondition** : an instance of the Vector3 class

**Postcondition** : takes in the X,Y, and Z value via user

**Protection Level** : Public

**Vector 4 Class :**

**float mX**

x value

**float mY**

y value

**float mZ**

z value

**float mW**

w value

**Vector4()**

**Prototype** : Vector4()

**Argument** : none

**Description** : makes an instance of the Vector4 class

**Precondition** : none

**Postcondition** : Constructs the Vector4 instance

**Protection Level** : Public

**Vector4(float x, float y, float z, float w)**

**Prototype** : Vector2(float x, float y, float z, float w)

**Argument** : float x, float y, float z, float w

**Description** : sets the X, Y ,Z, and W value for the vector

**Precondition** : an instance of the Vector3 class

**Postcondition** : sets the X, Y, Z, and W value for the vector

**Protection Level** : Public

**Vector4 operator+(Vector4 & other)**

**Prototype** : Vector4 operator+(Vector3 & other)

**Argument** : Vector4 & other

**Description** : adds two vectors together

**Precondition** : an instance of the Vector4 class

**Postcondition** : returns the sum of two vectors

**Protection Level** : Public

**Vector4 operator-(Vector3 & other)**

**Prototype** : Vector4 operator-(Vector4 & other)

**Argument** : Vector4 & other

**Description** : subtracts two vectors

**Precondition** : an instance of the Vector4 class

**Postcondition** : returns the difference of two vectors

**Protection Level** : Public

**Vector4 operator\*(Vector4 & other)**

**Prototype** : Vector4 operator\*(float other)

**Argument** : float other

**Description** : multiplies two vectors

**Precondition** : an instance of the Vector4 class

**Postcondition** : returns the product of two vectors

**Protection Level** : Public

**bool operator==(Vector4 & other)**

**Prototype** : bool operator==(Vector4 & other)

**Argument** : Vector4 & other

**Description** : checks to see if two vectors are equal

**Precondition** : an instance of the Vector4 class

**Postcondition** : checks to see if two vectors are equal

**Protection Level** : Public

**float Dot(Vector4 & other)**

**Prototype** : float Dot(Vector4 & other)

**Argument** : Vector4 & other

**Description** : finds the dot product of the vector

**Precondition** : an instance of the Vector4 class

**Postcondition** : finds the dot product of the vector

**Protection Level** : Public

**Vector4 Normalize()**

**Prototype** : Vector4 Normalize()

**Argument** : none

**Description** : normalizes the vector after finding the magnitude

**Precondition** : an instance of the Vector4 class

**Postcondition** : normalizes the vector

**Protection Level** : Public

**float Magnitude()**

**Prototype** : float Magnitude()

**Argument** : none

**Description** : finds the magnitude of the vector

**Precondition** : an instance of the Vector4 class

**Postcondition** : finds the magnitude

**Protection Level** : Public

**friend ostream& operator<<(ostream& os, const Vector3 vector)**

**Prototype** : friend ostream& operator<<(ostream& os, const Vector4 vector)

**Argument** : ostream& os, const Vector4 vector

**Description** : outputs the X, Y, Z, and W value to the console

**Precondition** : an instance of the Vector4 class

**Postcondition** : outputs the X,Y, Z, and W value to the console

**Protection Level** : Public

**friend istream& operator<<(istream& is, const Vector4 vector)**

**Prototype** : friend istream& operator<<(istream& is, const Vector4 vector)

**Argument** : ostream& os, const Vector4 vector

**Description** : takes in the X,Y, Z, and W value via user

**Precondition** : an instance of the Vector4 class

**Postcondition** : takes in the X,Y, Z, and W value via user

**Protection Level** : Public

**Matrix 2 Class :**

**float mMat[4]**

Sets the matrix

**Matrix2()**

**Prototype** : Matrix2()

**Argument** : none

**Description** : makes an instance of the Matrix2 class

**Precondition** : none

**Postcondition** : Constructs the Matrix instance

**Protection Level** : Public

**Matrix2(float mat[])**

**Prototype** : Matrix2(float mat[])

**Argument** : float mat[]

**Description** : loops through the matrix

**Precondition** : an instance of the Matrix2 class

**Postcondition** : loops through the matrix

**Protection Level** : Public

**Matrix2(float indexA, float indexB, float indexC, float indexD);**

**Prototype** : Matrix2(float indexA, float indexB, float indexC, float indexD)

**Argument** : float indexA, float indexB, float indexC, float indexD

**Description** : sets all the indexes in the matrix

**Precondition** : an instance of the Matrix2 class

**Postcondition** : sets all the indexes in the matrix

**Protection Level** : Public

**Matrix2 Rotate(float angle);**

**Prototype** : Matrix2 Rotate(float angle)

**Argument** : float angle

**Description** : rotates the matrix on a 2D plan

**Precondition** : an instance of the Matrix2 class

**Postcondition** : rotates the matrix on a 2D plan

**Protection Level** : Public

**Matrix2 operator+(Matrix2 &other);**

**Prototype** : Matrix2 operator+(Matrix2 &other)

**Argument** : Matrix2 &other

**Description** : adds two matrices

**Precondition** : an instance of the Matrix2 class

**Postcondition** : returns the sum of two matrices

**Protection Level** : Public

**Matrix2 operator-(Matrix2 &other);**

**Prototype** : Matrix2 operator-(Matrix2 &other)

**Argument** : Matrix2 &other

**Description** : subtracts two matrices

**Precondition** : an instance of the Matrix2 class

**Postcondition** : returns the difference of two matrices

**Protection Level** : Public

**Matrix2 operator\*(Matrix2 &other);**

**Prototype** : Matrix2 operator\*(Matrix2 &other)

**Argument** : Matrix2 &other

**Description** : multiplies two matrices

**Precondition** : an instance of the Matrix2 class

**Postcondition** : returns the product of two matrices

**Protection Level** : Public

**friend ostream& operator<<(ostream& os, const Matrix2 matrix);**

**Prototype** : friend ostream& operator<<(ostream& os, const Matrix2 vector)

**Argument** : ostream& os, const Matrix2 vector

**Description** : outputs the numbers in the matrix to the console

**Precondition** : an instance of the Matrix2 class

**Postcondition** : outputs the numbers in the matrix to the console

**Protection Level** : Public

**friend istream& operator >> (istream& is, Matrix2 matrix);**

**Prototype** : friend istream& operator<<(istream& is, const Matrix2 vector)

**Argument** : ostream& os, const Matrix2 vector

**Description** : takes in the indexes for the matrix via user

**Precondition** : an instance of the Matrix2 class

**Postcondition** : takes in indexes for the matrix via user

**Protection Level** : Public

**Matrix 3 Class :**

**float mMat[9]**

Sets the matrix

**Matrix3()**

**Prototype** : Matrix3()

**Argument** : none

**Description** : makes an instance of the Matrix3 class

**Precondition** : none

**Postcondition** : Constructs the Matrix instance

**Protection Level** : Public

**Matrix3(float mat[])**

**Prototype** : Matrix3(float mat[])

**Argument** : float mat[]

**Description** : loops through the matrix

**Precondition** : an instance of the Matrix3 class

**Postcondition** : loops through the matrix

**Protection Level** : Public

**Matrix3(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, f float indexG, float indexH, float indexI);**

**Prototype** : Matrix2(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI)

**Argument** : float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI

**Description** : sets all the indexes in the matrix

**Precondition** : an instance of the Matrix3 class

**Postcondition** : sets all the indexes in the matrix

**Protection Level** : Public

**Matrix3 RotateX(float angle);**

**Prototype** : Matrix3 RotateX(float angle)

**Argument** : float angle

**Description** : rotates the matrix on the X axis

**Precondition** : an instance of the Matrix3 class

**Postcondition** : rotates the matrix on the X axis

**Protection Level** : Public

**Matrix3 RotateY(float angle);**

**Prototype** : Matrix3 RotateY(float angle)

**Argument** : float angle

**Description** : rotates the matrix on the Y axis

**Precondition** : an instance of the Matrix3 class

**Postcondition** : rotates the matrix on the Y axis

**Protection Level** : Public

**Matrix3 RotateZ(float angle);**

**Prototype** : Matrix3 RotateZ(float angle)

**Argument** : float angle

**Description** : rotates the matrix on the Z axis

**Precondition** : an instance of the Matrix3 class

**Postcondition** : rotates the matrix on the Z axis

**Protection Level** : Public

**Matrix3 operator+(Matrix3 &other);**

**Prototype** : Matrix3 operator+(Matrix3 &other)

**Argument** : Matrix3 &other

**Description** : adds two matrices

**Precondition** : an instance of the Matrix3 class

**Postcondition** : returns the sum of two matrices

**Protection Level** : Public

**Matrix3 operator-(Matrix3 &other);**

**Prototype** : Matrix3 operator-(Matrix3 &other)

**Argument** : Matrix3 &other

**Description** : subtracts two matrices

**Precondition** : an instance of the Matrix3 class

**Postcondition** : returns the difference of two matrices

**Protection Level** : Public

**Matrix3 operator\*(Matrix3 &other);**

**Prototype** : Matrix3 operator\*(Matrix3 &other)

**Argument** : Matrix3 &other

**Description** : multiplies two matrices

**Precondition** : an instance of the Matrix3 class

**Postcondition** : returns the product of two matrices

**Protection Level** : Public

**friend ostream& operator<<(ostream& os, const Matrix3 matrix);**

**Prototype** : friend ostream& operator<<(ostream& os, const Matrix3 vector)

**Argument** : ostream& os, const Matrix3 vector

**Description** : outputs the numbers in the matrix to the console

**Precondition** : an instance of the Matrix3 class

**Postcondition** : outputs the numbers in the matrix to the console

**Protection Level** : Public

**friend istream& operator >> (istream& is, Matrix2 matrix);**

**Prototype** : friend istream& operator<<(istream& is, const Matrix3 vector)

**Argument** : ostream& os, const Matrix3 vector

**Description** : takes in the indexes for the matrix via user

**Precondition** : an instance of the Matrix3 class

**Postcondition** : takes in indexes for the matrix via user

**Protection Level** : Public

**Matrix 4 Class :**

**float mMat[16]**

Sets the matrix

**Matrix4()**

**Prototype** : Matrix4()

**Argument** : none

**Description** : makes an instance of the Matrix4 class

**Precondition** : none

**Postcondition** : Constructs the Matrix instance

**Protection Level** : Public

**Matrix4(float mat[])**

**Prototype** : Matrix4(float mat[])

**Argument** : float mat[]

**Description** : loops through the matrix

**Precondition** : an instance of the Matrix3 class

**Postcondition** : loops through the matrix

**Protection Level** : Public

**Matrix4(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, f float indexG, float indexH, float indexI, float indexJ,**

**float indexK, float indexL, float indexM, float indexN, float indexO, float indexP)**

**Prototype** : Matrix2(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI, float indexJ,

float indexK, float indexL, float indexM, float indexN, float indexO, float indexP)

**Argument** : float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI, float indexJ,

float indexK, float indexL, float indexM, float indexN, float indexO, float indexP

**Description** : sets all the indexes in the matrix

**Precondition** : an instance of the Matrix4 class

**Postcondition** : sets all the indexes in the matrix

**Protection Level** : Public

**Matrix4 RotateX(float angle);**

**Prototype** : Matrix4 RotateX(float angle)

**Argument** : float angle

**Description** : rotates the matrix on the X axis

**Precondition** : an instance of the Matrix4 class

**Postcondition** : rotates the matrix on the X axis

**Protection Level** : Public

**Matrix4 RotateY(float angle);**

**Prototype** : Matrix4 RotateY(float angle)

**Argument** : float angle

**Description** : rotates the matrix on the Y axis

**Precondition** : an instance of the Matrix4 class

**Postcondition** : rotates the matrix on the Y axis

**Protection Level** : Public

**Matrix4 RotateZ(float angle);**

**Prototype** : Matrix4 RotateZ(float angle)

**Argument** : float angle

**Description** : rotates the matrix on the Z axis

**Precondition** : an instance of the Matrix4 class

**Postcondition** : rotates the matrix on the Z axis

**Protection Level** : Public

**Matrix4 operator+(Matrix4 &other);**

**Prototype** : Matrix4 operator+(Matrix4 &other)

**Argument** : Matrix4 &other

**Description** : adds two matrices

**Precondition** : an instance of the Matrix4 class

**Postcondition** : returns the sum of two matrices

**Protection Level** : Public

**Matrix4 operator-(Matrix4 &other);**

**Prototype** : Matrix4 operator-(Matrix4 &other)

**Argument** : Matrix4 &other

**Description** : subtracts two matrices

**Precondition** : an instance of the Matrix4 class

**Postcondition** : returns the difference of two matrices

**Protection Level** : Public

**Matrix4 operator\*(Matrix4 &other);**

**Prototype** : Matrix4 operator\*(Matrix4 &other)

**Argument** : Matrix4 &other

**Description** : multiplies two matrices

**Precondition** : an instance of the Matrix4 class

**Postcondition** : returns the product of two matrices

**Protection Level** : Public

**friend ostream& operator<<(ostream& os, const Matrix4 matrix);**

**Prototype** : friend ostream& operator<<(ostream& os, const Matrix4 vector)

**Argument** : ostream& os, const Matrix4 vector

**Description** : outputs the numbers in the matrix to the console

**Precondition** : an instance of the Matrix4 class

**Postcondition** : outputs the numbers in the matrix to the console

**Protection Level** : Public

**friend istream& operator >> (istream& is, Matrix4 matrix);**

**Prototype** : friend istream& operator<<(istream& is, const Matrix4 vector)

**Argument** : ostream& os, const Matrix4 vector

**Description** : takes in the indexes for the matrix via user

**Precondition** : an instance of the Matrix4 class

**Postcondition** : takes in indexes for the matrix via user

**Protection Level** : Public

**Source Code :**

**Vector 2 :**

#include "Vector2.h"

#include <math.h>

Vector2::Vector2()

{

}

Vector2::Vector2(float x, float y)

{

mX = x;

mY = y;

}

Vector2 Vector2::operator+(Vector2 & other)//adds

{

Vector2 add;

add.mX = mX + other.mX;

add.mY = mY + other.mY;

return add;

}

Vector2 Vector2::operator-(Vector2 & other)//subtracts

{

Vector2 sub;

sub.mX = mX - other.mX;

sub.mY = mY - other.mY;

return sub;

}

Vector2 Vector2::operator\*(float other)//scales

{

Vector2 scale;

scale.mX = mX \* other;

scale.mY = mY \* other;

return scale;

}

bool Vector2::operator==(Vector2 & other) //compares

{

if (mX == other.mX && mY == other.mY)

return true;

}

float Vector2::Dot(Vector2 & other) //shows dot product

{

float dot;

dot = (mX \* other.mX) + (mY \* other.mY);

return dot;

}

float Vector2::Magnitude() //finds the magnitude

{

float Xsquare = mX\*mX;

float Ysquare = mY\*mY;

float sum = Xsquare + Ysquare;

float mag = sqrt(sum);

return mag;

}

Vector2 Vector2::Normalize() //normalizes the vectors

{

Vector2 norm;

norm.mX = mX / Magnitude();

norm.mY = mY / Magnitude();

return norm;

}

ostream& operator <<(ostream& os, const Vector2 vector)

{

os << "<" << vector.mX << "," << vector.mY << ">";

return os;

}

istream & operator >>(istream & is, Vector2 vector)

{

is >> vector.mX >> vector.mY;

return is;

}

**Vector 3 :**

#include "Vector3.h"

#include <math.h>

Vector3::Vector3()

{

}

Vector3::Vector3(float x, float y, float z)

{

mX = x;

mY = y;

mZ = z;

}

Vector3 Vector3::operator+(Vector3 & other)//adds

{

Vector3 add;

add.mX = mX + other.mX;

add.mY = mY + other.mY;

add.mZ = mZ + other.mZ;

return add;

}

Vector3 Vector3::operator-(Vector3 & other)//subtracts

{

Vector3 sub;

sub.mX = mX - other.mX;

sub.mY = mY - other.mY;

sub.mZ = mZ - other.mZ;

return sub;

}

Vector3 Vector3::operator\*(float other)//scales

{

Vector3 scale;

scale.mX = mX \* other;

scale.mY = mY \* other;

scale.mZ = mZ \* other;

return scale;

}

bool Vector3::operator==(Vector3 & other) //compares

{

if (mX == other.mX && mY == other.mY && mZ == other.mZ)

return true;

}

float Vector3::Dot(Vector3 & other) //shows dot product

{

float dot;

dot = (mX \* other.mX) + (mY \* other.mY) + (mZ \* other.mZ);

return dot;

}

Vector3 Vector3::Cross(Vector3 & other) //shows cross product

{

Vector3 X;

X.mX = (mY \* other.mZ) - (mZ \* other.mY);

X.mY = (mZ \* other.mX) - (mX \* other.mZ);

X.mZ = (mX \* other.mY) - (mY \* other.mX);

return X;

}

Vector3 Vector3::Normalize() //normalises the vectors

{

Vector3 norm;

norm.mX = mX / Magnitude();

norm.mY = mY / Magnitude();

norm.mZ = mZ / Magnitude();

return norm;

}

float Vector3::Magnitude() //finds the magnitude

{

float Xsquare = mX\*mX;

float Ysquare = mY\*mY;

float Zsquare = mZ\*mZ;

float sum = Xsquare + Ysquare + Zsquare;

float mag = sqrt(sum);

return mag;

}

ostream& operator <<(ostream& os, const Vector3 vector)

{

os << "<" << vector.mX << "," << vector.mY << "," << vector.mZ << ">";

return os;

}

istream & operator >> (istream & is, Vector3 vector)

{

is >> vector.mX >> vector.mY >> vector.mZ;

return is;

}

**Vector 4 :**

#include "Vector4.h"

#include <math.h>

Vector4::Vector4()

{

}

Vector4::Vector4(float x, float y, float z, float w)

{

mX = x;

mY = y;

mZ = z;

mW = w;

}

Vector4 Vector4::operator+(Vector4 & other)

{

Vector4 add;

add.mX = mX + other.mX;

add.mY = mY + other.mY;

add.mZ = mZ + other.mZ;

add.mW = mW + other.mW;

return add;

}

Vector4 Vector4::operator-(Vector4 & other)

{

Vector4 sub;

sub.mX = mX - other.mX;

sub.mY = mY - other.mY;

sub.mZ = mZ - other.mZ;

sub.mW = mW - other.mW;

return sub;

}

Vector4 Vector4::operator\*(float other)

{

Vector4 scale;

scale.mX = mX \* other;

scale.mY = mY \* other;

scale.mZ = mZ \* other;

scale.mW = mW \* other;

return scale;

}

bool Vector4::operator==(Vector4 & other)

{

if (mX == other.mX && mY == other.mY && mZ == other.mZ && mW == other.mW)

return true;

}

float Vector4::Dot(Vector4 & other)

{

float dot;

dot = (mX \* other.mX) + (mY \* other.mY) + (mZ \* other.mZ) + (mW \* other.mW);

return dot;

}

Vector4 Vector4::Normalize()

{

Vector4 norm;

norm.mX = mX / Magnitude();

norm.mY = mY / Magnitude();

norm.mZ = mZ / Magnitude();

norm.mW = mW / Magnitude();

return norm;

}

float Vector4::Magnitude()

{

float Xsquare = mX\*mX;

float Ysquare = mY\*mY;

float Zsquare = mZ\*mZ;

float Wsquare = mW\*mW;

float sum = Xsquare + Ysquare + Zsquare + Wsquare;

float mag = sqrt(sum);

return mag;

}

ostream & operator<<(ostream & os, const Vector4 vector)

{

os << "<" << vector.mX << "," << vector.mY << "," << vector.mZ << "," << vector.mW << ">";

return os;

}

istream & operator >> (istream & is, Vector4 vector)

{

is >> vector.mX >> vector.mY >> vector.mZ >> vector.mW;

return is;

}

**Matrix 2 :**

#include "Matrix2.h"

Matrix2::Matrix2()

{

}

Matrix2::Matrix2(float mat[])

{

for (int i = 0; i < 4; i++)

{

mMat[i] = mat[i];

}

}

Matrix2::Matrix2(float indexA, float indexB, float indexC, float indexD)

{

mMat[0] = indexA;

mMat[1] = indexB;

mMat[2] = indexC;

mMat[3] = indexD;

}

Matrix2 Matrix2::Rotate(float angle)

{

float rad = ((angle \* 3.1415926535897) / 180);

Matrix2 \*rotate = new Matrix2(cos(rad), sin(rad), -sin(rad), cos(rad));

return \*rotate \* \*this;

}

Matrix2 Matrix2::operator+(Matrix2 & other)

{

Matrix2 add;

for (int i = 0; i < 4; i++)

{

add.mMat[i] = mMat[i] + other.mMat[i];

}

return add;

}

Matrix2 Matrix2::operator-(Matrix2 & other)

{

Matrix2 sub;

for (int i = 0; i < 4; i++)

{

sub.mMat[i] = mMat[i] - other.mMat[i];

}

return sub;

}

Matrix2 Matrix2::operator\*(Matrix2 &other)

{

float \*newMat = new float[4];

newMat[0] = (mMat[0] \* other.mMat[0]) + (mMat[1] \* other.mMat[2]);

newMat[1] = (mMat[0] \* other.mMat[1]) + (mMat[1] \* other.mMat[3]);

newMat[2] = (mMat[2] \* other.mMat[0]) + (mMat[3] \* other.mMat[2]);

newMat[3] = (mMat[2] \* other.mMat[1]) + (mMat[3] \* other.mMat[3]);

return newMat;

}

ostream & operator<<(ostream & os, const Matrix2 matrix)

{

os << matrix.mMat[0] << " " << matrix.mMat[1] << endl;

os << matrix.mMat[2] << " " << matrix.mMat[3] << endl;

return os;

}

istream & operator >> (istream & is, Matrix2 matrix)

{

is >> matrix.mMat[0] >> matrix.mMat[1];

is >> matrix.mMat[2] >> matrix.mMat[3];

return is;

}

**Matrix 3 :**

#include "Matrix3.h"

Matrix3::Matrix3()

{

}

Matrix3::Matrix3(float mat[])

{

for (int i = 0; i < 9; i++)

{

mMat[i] = mat[i];

}

}

Matrix3::Matrix3(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI)

{

mMat[0] = indexA;

mMat[1] = indexB;

mMat[2] = indexC;

mMat[3] = indexD;

mMat[4] = indexE;

mMat[5] = indexF;

mMat[6] = indexG;

mMat[7] = indexH;

mMat[8] = indexI;

}

Matrix3 Matrix3::RotateX(float angle)

{

float rad = ((angle \* 3.1415926535897) / 180);

Matrix3 \*rotate = new Matrix3(1, 0, 0, 0, cos(rad), -sin(rad), 0, sin(rad), cos(rad));

return \*rotate \* \*this;

}

Matrix3 Matrix3::RotateY(float angle)

{

float rad = ((angle \* 3.1415926535897) / 180);

Matrix3 \*rotate = new Matrix3(cos(rad), 0, -sin(rad), 0, 1, 0, sin(rad), 0, cos(rad));

return \*rotate \* \*this;

}

Matrix3 Matrix3::RotateZ(float angle)

{

float rad = ((angle \* 3.1415926535897) / 180);

Matrix3 \*rotate = new Matrix3(cos(rad), -sin(rad), 0, sin(rad), cos(rad), 0, 0, 0, 1);

return \*rotate \* \*this;

}

Matrix3 Matrix3::operator+(Matrix3 & other)

{

Matrix3 add;

for (int i = 0; i < 9; i++)

{

add.mMat[i] = mMat[i] + other.mMat[i];

}

return add;

}

Matrix3 Matrix3::operator-(Matrix3 & other)

{

Matrix3 sub;

for (int i = 0; i < 4; i++)

{

sub.mMat[i] = mMat[i] - other.mMat[i];

}

return sub;

}

Matrix3 Matrix3::operator\*(Matrix3 & other)

{

float \*newMat = new float[9];

newMat[0] = (mMat[0] \* other.mMat[0]) + (mMat[1] \* other.mMat[3]) + mMat[2] \* other.mMat[6];

newMat[1] = (mMat[0] \* other.mMat[1]) + (mMat[1] \* other.mMat[4]) + mMat[2] \* other.mMat[7];

newMat[2] = (mMat[0] \* other.mMat[2]) + (mMat[1] \* other.mMat[5]) + mMat[2] \* other.mMat[8];

newMat[3] = (mMat[3] \* other.mMat[0]) + (mMat[4] \* other.mMat[3]) + mMat[5] \* other.mMat[6];

newMat[4] = (mMat[3] \* other.mMat[1]) + (mMat[4] \* other.mMat[4]) + mMat[5] \* other.mMat[7];

newMat[5] = (mMat[3] \* other.mMat[2]) + (mMat[4] \* other.mMat[5]) + mMat[5] \* other.mMat[8];

newMat[6] = (mMat[6] \* other.mMat[0]) + (mMat[7] \* other.mMat[3]) + mMat[8] \* other.mMat[6];

newMat[7] = (mMat[6] \* other.mMat[1]) + (mMat[7] \* other.mMat[4]) + mMat[8] \* other.mMat[7];

newMat[8] = (mMat[6] \* other.mMat[2]) + (mMat[7] \* other.mMat[5]) + mMat[8] \* other.mMat[8];

return newMat;

}

ostream & operator<<(ostream & os, const Matrix3 matrix)

{

os << matrix.mMat[0] << " " << matrix.mMat[1] << " " << matrix.mMat[2] << endl;

os << matrix.mMat[3] << " " << matrix.mMat[4] << " " << matrix.mMat[5] << endl;

os << matrix.mMat[6] << " " << matrix.mMat[7] << " " << matrix.mMat[8] << endl;

return os;

}

istream & operator >> (istream & is, Matrix3 matrix)

{

is >> matrix.mMat[0] >> matrix.mMat[1] >> matrix.mMat[2];

is >> matrix.mMat[3] >> matrix.mMat[4] >> matrix.mMat[5];

is >> matrix.mMat[6] >> matrix.mMat[7] >> matrix.mMat[8];

return is;

}

**Matrix 4 :**

#include "Matrix4.h"

Matrix4::Matrix4()

{

}

Matrix4::Matrix4(float mat[])

{

for (int i = 0; i < 16; i++)

{

mMat[i] = mat[i];

}

}

Matrix4::Matrix4(float indexA, float indexB, float indexC, float indexD,

float indexE, float indexF, float indexG, float indexH, float indexI, float indexJ,

float indexK, float indexL, float indexM, float indexN, float indexO, float indexP)

{

mMat[0] = indexA;

mMat[1] = indexB;

mMat[2] = indexC;

mMat[3] = indexD;

mMat[4] = indexE;

mMat[5] = indexF;

mMat[6] = indexG;

mMat[7] = indexH;

mMat[8] = indexI;

mMat[9] = indexJ;

mMat[10] = indexK;

mMat[11] = indexL;

mMat[12] = indexM;

mMat[13] = indexN;

mMat[14] = indexO;

mMat[15] = indexP;

}

Matrix4 Matrix4::RotateX(float angle)

{

float rad = ((angle \* 3.1415926535897) / 180);

Matrix4 \*rotate = new Matrix4(1, 0, 0, 0,

0, cos(rad), -sin(rad), 0,

0, sin(rad),cos(rad), 0,

0, 0, 0, 1);

return \*rotate \* \*this;

}

Matrix4 Matrix4::RotateY(float angle)

{

float rad = ((angle \* 3.1415926535897) / 180);

Matrix4 \*rotate = new Matrix4(cos(rad), 0, sin(rad), 0, 0, 1, 0, 0, -sin(rad), 0,

cos(rad), 0,0, 0, 0, 1);

return \*rotate \* \*this;

}

Matrix4 Matrix4::RotateZ(float angle)

{

float rad = ((angle \* 3.1415926535897) / 180);

Matrix4 \*rotate = new Matrix4(cos(rad), -sin(rad), 0, 0, sin(rad), cos(rad), 0, 0,

0, 0, 1, 0, 0, 0, 0, 1);

return \*rotate \* \*this;

}

Matrix4 Matrix4::operator+(Matrix4 & other)

{

Matrix4 add;

for (int i = 0; i < 16; i++)

{

add.mMat[i] = mMat[i] + other.mMat[i];

}

return add;

}

Matrix4 Matrix4::operator-(Matrix4 & other)

{

Matrix4 sub;

for (int i = 0; i < 16; i++)

{

sub.mMat[i] = mMat[i] - other.mMat[i];

}

return sub;

}

Matrix4 Matrix4::operator\*(Matrix4 & other)

{

float \*newMat = new float[16];

newMat[0] = (mMat[0] \* other.mMat[0]) + (mMat[1] \* other.mMat[4]) + (mMat[2] \* other.mMat[8]) + (mMat[3] \* other.mMat[12]);

newMat[1] = (mMat[0] \* other.mMat[1]) + (mMat[1] \* other.mMat[5]) + (mMat[2] \* other.mMat[9]) + (mMat[3] \* other.mMat[13]);

newMat[2] = (mMat[0] \* other.mMat[2]) + (mMat[1] \* other.mMat[6]) + (mMat[2] \* other.mMat[10]) + (mMat[3] \* other.mMat[14]);

newMat[3] = (mMat[0] \* other.mMat[3]) + (mMat[1] \* other.mMat[7]) + (mMat[2] \* other.mMat[11]) + (mMat[3] \* other.mMat[15]);

newMat[4] = (mMat[4] \* other.mMat[0]) + (mMat[5] \* other.mMat[4]) + (mMat[6] \* other.mMat[8]) + (mMat[7] \* other.mMat[12]);

newMat[5] = (mMat[4] \* other.mMat[1]) + (mMat[5] \* other.mMat[5]) + (mMat[6] \* other.mMat[9]) + (mMat[7] \* other.mMat[13]);

newMat[6] = (mMat[4] \* other.mMat[2]) + (mMat[5] \* other.mMat[6]) + (mMat[6] \* other.mMat[10]) + (mMat[7] \* other.mMat[14]);

newMat[7] = (mMat[4] \* other.mMat[3]) + (mMat[5] \* other.mMat[7]) + (mMat[6] \* other.mMat[11]) + (mMat[7] \* other.mMat[15]);

newMat[8] = (mMat[8] \* other.mMat[0]) + (mMat[9] \* other.mMat[4]) + (mMat[10] \* other.mMat[8]) + (mMat[11] \* other.mMat[12]);

newMat[9] = (mMat[8] \* other.mMat[1]) + (mMat[9] \* other.mMat[5]) + (mMat[10] \* other.mMat[9]) + (mMat[11] \* other.mMat[13]);

newMat[10] = (mMat[8] \* other.mMat[2]) + (mMat[9] \* other.mMat[6]) + (mMat[10] \* other.mMat[10]) + (mMat[11] \* other.mMat[14]);

newMat[11] = (mMat[8] \* other.mMat[3]) + (mMat[9] \* other.mMat[7]) + (mMat[10] \* other.mMat[11]) + (mMat[11] \* other.mMat[15]);

newMat[12] = (mMat[12] \* other.mMat[0]) + (mMat[13] \* other.mMat[4]) + (mMat[14] \* other.mMat[8]) + (mMat[15] \* other.mMat[12]);

newMat[13] = (mMat[12] \* other.mMat[1]) + (mMat[13] \* other.mMat[5]) + (mMat[14] \* other.mMat[9]) + (mMat[15] \* other.mMat[13]);

newMat[14] = (mMat[12] \* other.mMat[2]) + (mMat[13] \* other.mMat[6]) + (mMat[14] \* other.mMat[10]) + (mMat[15] \* other.mMat[14]);

newMat[15] = (mMat[12] \* other.mMat[3]) + (mMat[13] \* other.mMat[7]) + (mMat[14] \* other.mMat[11]) + (mMat[15] \* other.mMat[15]);

return newMat;

}

ostream & operator<<(ostream & os, const Matrix4 matrix)

{

os << matrix.mMat[0] << " " << matrix.mMat[1] << " " << matrix.mMat[2] << " " << matrix.mMat[3] << endl;

os << matrix.mMat[4] << " " << matrix.mMat[5] << " " << matrix.mMat[6] << " " << matrix.mMat[7] << endl;

os << matrix.mMat[8] << " " << matrix.mMat[9] << " " << matrix.mMat[10] << " " << matrix.mMat[11] << endl;

os << matrix.mMat[12] << " " << matrix.mMat[13] << " " << matrix.mMat[14] << " " << matrix.mMat[15] << endl;

return os;

}

istream & operator >> (istream & is, Matrix4 matrix)

{

is >> matrix.mMat[0] >> matrix.mMat[1] >> matrix.mMat[2] >> matrix.mMat[3];

is >> matrix.mMat[4] >> matrix.mMat[5] >> matrix.mMat[6] >> matrix.mMat[7];

is >> matrix.mMat[8] >> matrix.mMat[9] >> matrix.mMat[10] >> matrix.mMat[11];

is >> matrix.mMat[12] >> matrix.mMat[13] >> matrix.mMat[14] >> matrix.mMat[15];

return is;

}