Zyron Melancon

12/16/16

Math for Games Assessment

1. **Requirements Documentation**
   1. **Description of Problem**

**Name:** Create a math library

**Problem Statement:** Create a basic math library to fully understanding the formulae needed to manipulate the Vectors and Matrices used within all games.

**Problem Specification:** The following methods implemented, where V is a Vector, M is a Matrix, and f is a float:

* V = V + V
* V = V – V
* V = V \* f
* M = M \* M
* f = V.dot(V)
* V = V.cross(V) (where V is a Vector3 or Vector4)
* f = V.magnitude()
* V.normalise()
* M.setRotateX(f) and M.setRotateY(f)
* M.setRotateZ(f) (for Matrix3 and Matrix4)
  1. **Input Information**

The user inputs floats, Vectors, and/or Matrices, depending on the function being used

* 1. **Output Information**

Method of output is chosen by the user. In the example provided, output is shown on the console window.

* 1. **User Interface**

I/O through console window

1. **System Architecture**
   1. **Vector 2:**

**Variables:**

**float m\_x**

**float m\_y**

**Functions:**

**float dot(Vector2 other)**

**Description:** Calculates and returns the dot product of the current Vector2, and a user-input Vector2

**Visibility:** Public

**float magnitude()**

**Description:** Calculate and returns the magnitude of the current Vector2

**Visibility:** Public

**void normalize()**

**Description:** Normalizes the current Vector2

**Visibility:** Public

**void print()**

**Description:** Prints the Vector2 to the console in x y format

**Visibility:** Public

**Vector2 operator + (Vector2 other)**

**Description:** Returns a Vector2 containing the sums of the current and input Vector2 m\_x and m\_y variables

**Visibility:** Public

**Vector2 operator - (Vector2 other)**

**Description:** Returns a Vector2 containing the difference of the current Vector2 by the input Vector2 m\_x and m\_y variables

**Visibility:** Public

**Vector2 operator \* (float m)**

**Description:** Returns a Vector2 containing the product of the current Vector2 m\_x and m\_y variables and a scalar value (m)

**Visibility:** Public

**Vector2 operator / (float d)**

**Description:** Returns a Vector2 containing the quotient of the current Vector2 m\_x and m\_y variables and a scalar value (d)

**Visibility:** Public

* 1. **Vector 3**

**Variables:**

**float m\_x**

**float m\_y**

**float m\_z**

**Functions:**

**float dot(Vector3 other)**

**Description:** Calculates and returns the dot product of the current Vector3, and a user-input Vector3

**Visibility:** Public

**float magnitude()**

**Description:** Calculate and returns the magnitude of the current Vector3

**Visibility:** Public

**void normalize()**

**Description:** Normalizes the current Vector3

**Visibility:** Public

**void print()**

**Description:** Prints the current Vector3 to the console in x y z format

**Visibility:** Public

**Vector3 operator + (Vector3 other)**

**Description:** Returns a Vector3 containing the sums of the current and input Vector3 m\_x, m\_y, and m\_z variables

**Visibility:** Public

**Vector3 operator - (Vector3 other)**

**Description:** Returns a Vector3 containing the difference of the current Vector3 by the input Vector2 m\_x, m\_y, and m\_z variables

**Visibility:** Public

**Vector3 operator \* (float m)**

**Description:** Returns a Vector3 containing product of the current Vector3 m\_x, m\_y, and m\_z variables and a scalar value (m)

**Visibility:** Public

**Vector3 operator / (float d)**

**Description:** Returns a Vector3 containing the quotient of the current Vector3 m\_x, m\_y, and m\_z variables and a scalar value (d)

**Visibility:** Public

* 1. **Vector 4**

**Variables:**

**float m\_x**

**float m\_y**

**float m\_z**

**float m\_w**

**Functions:**

**float dot(Vector4 other)**

**Description:** Calculates and returns the dot product of the current Vector4, and a user-input Vector4

**Visibility:** Public

**float magnitude()**

**Description:** Calculate and returns the magnitude of the current Vector4

**Visibility:** Public

**void normalize()**

**Description:** Normalizes the current Vector4

**Visibility:** Public

**void print()**

**Description:** Prints the current Vector4 to the console in x y z w format

**Visibility:** Public

**Vector4 operator + (Vector4 other)**

**Description:** Returns a Vector4 containing the sums of the current and input Vector4 m\_x, m\_y, m\_z, and m\_w variables

**Visibility:** Public

**Vector4 operator - (Vector4 other)**

**Description:** Returns a Vector4 containing the difference of the current Vector4 by the input Vector2 m\_x, m\_y, m\_z, and m\_w variables

**Visibility:** Public

**Vector4 operator \* (float m)**

**Description:** Returns a Vector4 containing product of the current Vector3 m\_x, m\_y, m\_z, and m\_w variables and a scalar value (m)

**Visibility:** Public

**Vector4 operator / (float d)**

**Description:** Returns a Vector3 containing the quotient of the current Vector3 m\_x, m\_y, m\_z, and m\_w variables and a scalar value (d)

**Visibility:** Public

* 1. **Matrix2**

**Variables:**

**Vector2 m\_a**

**Vector2 m\_b**

**Functions:**

**void print()**

**Description:** Prints the matrix into the console as a 2 x 2 grid

**Visibility:** Public

**Matrix2 operator \* (Matrix2 other)**

**Description:** Returns a Matrix2 containing the product of the current Matrix2 and an input Matrix2

**Visibility:** Public

* 1. **Matrix3**

**Variables:**

**Vector3 m\_a**

**Vector3 m\_b**

**Vector3 m\_c**

**Functions:**

**void print()**

**Description:** Prints the matrix into the console as a 2 x 2 grid

**Visibility:** Public

**Matrix3 operator \* (Matrix3 other)**

**Description:** Returns a Matrix3 containing the product of the current Matrix3 and an input Matrix2

**Visibility:** Public

**void setRotateX(float r)**

**Description:** Returns a Matrix3 containing a rotation matrix of the X axis. The amount of rotation is determined by a float input in radians (r)

**Visibility:** Public

**void setRotateY(float r)**

**Description:** Returns a Matrix3 containing a rotation matrix of the Y axis. The amount of rotation is determined by a float input in radians (r)

**Visibility:** Public

**void setRotateZ(float r)**

**Description:** Returns a Matrix3 containing a rotation matrix of the Z axis. The amount of rotation is determined by a float input in radians (r)

**Visibility:** Public

* 1. **Matrix4**

**Variables:**

**m\_a**

**m\_b**

**m\_c**

**m\_d**

**Functions:**

**void print()**

**Description:** Prints the matrix into the console as a 3 x 3 grid

**Visibility:** Public

**Matrix4 operator \* (Matrix4 other)**

**Description:** Returns a Matrix4 containing the product of the current Matrix3 and an input Matrix4

**Visibility:** Public

**void setRotateX(float r)**

**Description:** Returns a Matrix4 containing a rotation matrix of the X axis. The amount of rotation is determined by a float input in radians (r)

**Visibility:** Public

**void setRotateY(float r)**

**Description:** Returns a Matrix4 containing a rotation matrix of the Y axis. The amount of rotation is determined by a float input in radians (r)

**Visibility:** Public

**void setRotateZ(float r)**

**Description:** Returns a Matrix4 containing a rotation matrix of the Z axis. The amount of rotation is determined by a float input in radians (r)

**Visibility:** Public

**.CPP File:**

#include "Math.h"

Vector2::Vector2()

{

m\_x = 0;

m\_y = 0;

}

Vector2::Vector2(float x, float y)

{

m\_x = x;

m\_y = y;

}

float Vector2::dot(Vector2 other)

{

return m\_x \* other.m\_x + m\_y \* other.m\_y;

}

float Vector2::magnitude()

{

return sqrt((m\_x\*m\_x) + (m\_y\*m\_y));

}

void Vector2::normalise()

{

if (m\_x > m\_y)

{

m\_y = m\_y / m\_x;

m\_x = 1;

}

else

{

m\_x = m\_x / m\_y;

m\_y = 1;

}

}

Vector2 Vector2::operator+(Vector2 other)

{

return Vector2(m\_x + other.m\_x, m\_y + other.m\_y);

}

Vector2 Vector2::operator-(Vector2 other)

{

return Vector2(m\_x - other.m\_x, m\_y - other.m\_y);

}

Vector2 Vector2::operator\*(float m)

{

return Vector2(m\_x \* m, m\_y \* m);

}

Vector2 Vector2::operator/(float d)

{

return Vector2(m\_x / d, m\_y / d);

}

Vector3::Vector3()

{

m\_x = 0;

m\_y = 0;

m\_z = 0;

}

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

{

m\_x = x;

m\_y = y;

m\_z = z;

}

float Vector3::dot(Vector3 other)

{

return m\_x \* other.m\_x + m\_y \* other.m\_y + m\_z \* other.m\_z;

}

Vector3 Vector3::cross(Vector3 other)

{

return Vector3(m\_y \* other.m\_z - m\_z \* other.m\_y,

m\_x \* other.m\_z - m\_z \* other.m\_x,

m\_x \* other.m\_y - m\_y \* other.m\_x);

}

float Vector3::magnitude()

{

return sqrt(m\_x\*m\_x + m\_y\*m\_y + m\_z\*m\_z);

}

void Vector3::normalise()

{

if (m\_x >= m\_y && m\_x >= m\_z)

{

m\_z = m\_z / m\_x;

m\_y = m\_y / m\_x;

m\_x = 1;

}

else if (m\_y >= m\_x && m\_y >= m\_z)

{

m\_z = m\_z / m\_y;

m\_x = m\_x / m\_y;

m\_y = 1;

}

else if (m\_z >= m\_x && m\_z >= m\_y)

{

m\_y = m\_y / m\_z;

m\_x = m\_x / m\_z;

m\_z = 1;

}

}

Vector3 Vector3::operator+(Vector3 other)

{

return Vector3(m\_x + other.m\_x, m\_y + other.m\_y, m\_z + other.m\_z);

}

Vector3 Vector3::operator-(Vector3 other)

{

return Vector3(m\_x - other.m\_x, m\_y - other.m\_y, m\_z - other.m\_z);

}

Vector3 Vector3::operator\*(float m)

{

return Vector3(m\_x \* m, m\_y \* m, m\_z \* m);

}

Vector3 Vector3::operator/(float d)

{

return Vector3(m\_x / d, m\_y / d, m\_z / d);

}

Vector4::Vector4()

{

m\_x = 0;

m\_y = 0;

m\_z = 0;

m\_w = 0;

}

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

{

m\_x = x;

m\_y = y;

m\_z = z;

m\_w = w;

}

float Vector4::dot(Vector4 other)

{

return m\_x\*other.m\_x + m\_y\*other.m\_y + m\_z\*other.m\_z + m\_w\*other.m\_w;

}

Vector4 Vector4::cross(Vector4 other)

{

return Vector4();

}

float Vector4::magnitude()

{

return sqrt(m\_x\*m\_x + m\_y\*m\_y + m\_z\*m\_z + m\_w\*m\_w);

}

void Vector4::normalise()

{

if (m\_x >= m\_y && m\_x >= m\_z && m\_x >= m\_w)

{

m\_w = m\_w / m\_x;

m\_z = m\_z / m\_x;

m\_y = m\_y / m\_x;

m\_x = 1;

}

else if (m\_y >= m\_x && m\_y >= m\_z && m\_x >= m\_w)

{

m\_w = m\_w / m\_y;

m\_z = m\_z / m\_y;

m\_x = m\_x / m\_y;

m\_y = 1;

}

else if (m\_z >= m\_x && m\_z >= m\_y && m\_x >= m\_w)

{

m\_w = m\_w / m\_z;

m\_y = m\_y / m\_z;

m\_x = m\_x / m\_z;

m\_z = 1;

}

else if (m\_w >= m\_x && m\_w >= m\_y && m\_w >= m\_z)

{

m\_z = m\_z / m\_w;

m\_y = m\_y / m\_w;

m\_x = m\_x / m\_w;

m\_w = 1;

}

}

Vector4 Vector4::operator+(Vector4 other)

{

return Vector4(m\_x + other.m\_x, m\_y + other.m\_y, m\_z + other.m\_z, m\_w + other.m\_w);

}

Vector4 Vector4::operator-(Vector4 other)

{

return Vector4(m\_x - other.m\_x, m\_y - other.m\_y, m\_z - other.m\_z, m\_w - other.m\_w);

}

Vector4 Vector4::operator\*(float m)

{

return Vector4(m\_x \* m, m\_y \* m, m\_z \* m, m\_w \* m);

}

Vector4 Vector4::operator/(float d)

{

return Vector4(m\_x / d, m\_y / d, m\_z / d, m\_w / d);

}

Matrix2::Matrix2(Vector2 a, Vector2 b)

{

m\_a = a;

m\_b = b;

}

Matrix2::Matrix2(float a, float b, float c, float d)

{

m\_a = Vector2(a, b);

m\_b = Vector2(c, d);

}

Matrix2 Matrix2::operator\*(Matrix2 other)

{

return Matrix2(m\_a.m\_x \* other.m\_a.m\_x + m\_a.m\_y \* other.m\_b.m\_x,

m\_a.m\_x \* other.m\_b.m\_y + m\_a.m\_y \* other.m\_b.m\_y,

m\_b.m\_x \* other.m\_a.m\_x + m\_b.m\_y \* other.m\_b.m\_x,

m\_b.m\_x \* other.m\_a.m\_y + m\_b.m\_y \* other.m\_b.m\_y);

}

Matrix3::Matrix3(Vector3 a, Vector3 b, Vector3 c)

{

m\_a = a;

m\_b = b;

m\_c = c;

}

Matrix3::Matrix3(float a, float b, float c, float d, float e, float f, float g, float h, float i)

{

m\_a = { a, b, c };

m\_b = { d, e, f };

m\_c = { g, h, i };

}

Matrix3 Matrix3::operator\*(Matrix3 o)

{

Vector3 newA = { m\_a.m\_x \* o.m\_a.m\_x + m\_a.m\_y \* o.m\_b.m\_x + m\_a.m\_z \* o.m\_c.m\_x,

m\_a.m\_x \* o.m\_a.m\_y + m\_a.m\_y \* o.m\_b.m\_y + m\_a.m\_z \* o.m\_c.m\_y,

m\_a.m\_x \* o.m\_a.m\_z + m\_a.m\_y \* o.m\_b.m\_z + m\_a.m\_z \* o.m\_c.m\_z };

Vector3 newB = { m\_b.m\_x \* o.m\_a.m\_x + m\_b.m\_y \* o.m\_b.m\_x + m\_b.m\_z \* o.m\_c.m\_x,

m\_b.m\_x \* o.m\_a.m\_y + m\_b.m\_y \* o.m\_b.m\_y + m\_b.m\_z \* o.m\_c.m\_y,

m\_b.m\_x \* o.m\_a.m\_z + m\_b.m\_y \* o.m\_b.m\_z + m\_b.m\_z \* o.m\_c.m\_z };

Vector3 newC = { m\_c.m\_x \* o.m\_a.m\_x + m\_c.m\_y \* o.m\_b.m\_x + m\_c.m\_z \* o.m\_c.m\_x,

m\_c.m\_x \* o.m\_a.m\_y + m\_c.m\_y \* o.m\_b.m\_y + m\_c.m\_z \* o.m\_c.m\_y,

m\_c.m\_x \* o.m\_a.m\_z + m\_c.m\_y \* o.m\_b.m\_z + m\_c.m\_z \* o.m\_c.m\_z };

return Matrix3(newA, newB, newC);

}

void Matrix3::setRotatex(float r)

{

m\_b.m\_y = cos(r);

m\_b.m\_z = -sin(r);

m\_c.m\_y = sin(r);

m\_c.m\_z = cos(r);

}

void Matrix3::setRotateY(float r)

{

m\_a.m\_x = cos(r);

m\_a.m\_z = sin(r);

m\_c.m\_x = -sin(r);

m\_c.m\_z = cos(r);

}

void Matrix3::setRotateZ(float r)

{

m\_a.m\_x = cos(r);

m\_a.m\_y = -sin(r);

m\_b.m\_x = sin(r);

m\_b.m\_y = cos(r);

}

Matrix4::Matrix4(Vector4 a, Vector4 b, Vector4 c, Vector4 d)

{

m\_a = a;

m\_b = b;

m\_c = c;

m\_d = d;

}

Matrix4::Matrix4(float a, float b, float c, float d, float e, float f, float g, float h, float i, float j, float k, float l, float m, float n, float o, float p)

{

m\_a = { a, b, c, d };

m\_b = { e, f, g, h };

m\_c = { i, j, k, l };

m\_d = { m, n, o, p };

}

Matrix4 Matrix4::operator\*(Matrix4 o)

{

Vector4 newA = { m\_a.m\_x \* o.m\_a.m\_x + m\_a.m\_y \* o.m\_b.m\_x + m\_a.m\_z \* o.m\_c.m\_x + m\_a.m\_w \* o.m\_d.m\_x,

m\_a.m\_x \* o.m\_a.m\_y + m\_a.m\_y \* o.m\_b.m\_y + m\_a.m\_z \* o.m\_c.m\_y + m\_a.m\_w \* o.m\_d.m\_y,

m\_a.m\_x \* o.m\_a.m\_z + m\_a.m\_y \* o.m\_b.m\_z + m\_a.m\_z \* o.m\_c.m\_z + m\_a.m\_w \* o.m\_d.m\_z,

m\_a.m\_x \* o.m\_a.m\_w + m\_a.m\_y \* o.m\_b.m\_w + m\_a.m\_z \* o.m\_c.m\_w + m\_a.m\_w \* o.m\_d.m\_w };

Vector4 newB = { m\_b.m\_x \* o.m\_a.m\_x + m\_b.m\_y \* o.m\_b.m\_x + m\_b.m\_z \* o.m\_c.m\_x + m\_b.m\_w \* o.m\_d.m\_x,

m\_b.m\_x \* o.m\_a.m\_y + m\_b.m\_y \* o.m\_b.m\_y + m\_b.m\_z \* o.m\_c.m\_y + m\_b.m\_w \* o.m\_d.m\_y,

m\_b.m\_x \* o.m\_a.m\_z + m\_b.m\_y \* o.m\_b.m\_z + m\_b.m\_z \* o.m\_c.m\_z + m\_b.m\_w \* o.m\_d.m\_z,

m\_b.m\_x \* o.m\_a.m\_w + m\_b.m\_y \* o.m\_b.m\_w + m\_b.m\_z \* o.m\_c.m\_w + m\_b.m\_w \* o.m\_d.m\_w };

Vector4 newC = { m\_c.m\_x \* o.m\_a.m\_x + m\_c.m\_y \* o.m\_b.m\_x + m\_c.m\_z \* o.m\_c.m\_x + m\_c.m\_w \* o.m\_d.m\_x,

m\_c.m\_x \* o.m\_a.m\_y + m\_c.m\_y \* o.m\_b.m\_y + m\_c.m\_z \* o.m\_c.m\_y + m\_c.m\_w \* o.m\_d.m\_y,

m\_c.m\_x \* o.m\_a.m\_z + m\_c.m\_y \* o.m\_b.m\_z + m\_c.m\_z \* o.m\_c.m\_z + m\_c.m\_w \* o.m\_d.m\_z,

m\_c.m\_x \* o.m\_a.m\_w + m\_c.m\_y \* o.m\_b.m\_w + m\_c.m\_z \* o.m\_c.m\_w + m\_c.m\_w \* o.m\_d.m\_w };

Vector4 newD = { m\_d.m\_x \* o.m\_a.m\_x + m\_d.m\_y \* o.m\_b.m\_x + m\_d.m\_z \* o.m\_c.m\_x + m\_d.m\_w \* o.m\_d.m\_x,

m\_d.m\_x \* o.m\_a.m\_y + m\_d.m\_y \* o.m\_b.m\_y + m\_d.m\_z \* o.m\_c.m\_y + m\_d.m\_w \* o.m\_d.m\_y,

m\_d.m\_x \* o.m\_a.m\_z + m\_d.m\_y \* o.m\_b.m\_z + m\_d.m\_z \* o.m\_c.m\_z + m\_d.m\_w \* o.m\_d.m\_z,

m\_d.m\_x \* o.m\_a.m\_w + m\_d.m\_y \* o.m\_b.m\_w + m\_d.m\_z \* o.m\_c.m\_w + m\_d.m\_w \* o.m\_d.m\_w };

return Matrix4(newA, newB, newC, newD);

}

void Matrix4::setRotateX(float r)

{

m\_b.m\_y = cos(r);

m\_b.m\_z = -sin(r);

m\_c.m\_y = sin(r);

m\_c.m\_z = cos(r);

}

void Matrix4::setRotateY(float r)

{

m\_a.m\_x = cos(r);

m\_a.m\_z = sin(r);

m\_c.m\_x = -sin(r);

m\_c.m\_z = cos(r);

}

void Matrix4::setRotateZ(float r)

{

m\_a.m\_x = cos(r);

m\_a.m\_y = -sin(r);

m\_b.m\_x = sin(r);

m\_b.m\_y = cos(r);

}

**Read Me**

All source code is found on GitHub at

<https://github.com/ZyronMelancon/MathLibrary>