Mariojulio Zaldivar

December 11th, 2016

The VQM structure: The math behind the scenes and an implementation in C++

- **Purpose**

Understanding what the VQM set is, its operations and properties, and how it can be used in an actual computer graphics application.

- **Definitions**

Prior to define the VQM set, we need to define very quickly two things: Quaternions and VQS structure, so that we can have a context for discussing the VQM set.

Quaternions[1]: Defined as a number system that extends the complex numbers, usually represented in the form $x + yi + zj + sk$, where $a, b, c, d \in \mathbb{R}$ and $i, j, k \in \mathbb{I}$.

In terms of notations, a quaternion can have the notation of a 4D vector $\boldsymbol{q} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{s}] \in \mathbb{H}$, where s is a scalar value and x, y and z are the components of a 3D vector. Furthermore, that notation can be simplified as $\boldsymbol{q} = [\boldsymbol{v}, \boldsymbol{s}] \in \mathbb{H}$, where s is still the scalar component and v is a 3D vector.

Quaternions are widely used in computer graphics, due to their convenient way to represent rotations. The way we can achieve this is by the following equation: Let q be a quaternion, and $\boldsymbol{v} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ a 3D vector that represents a position in the virtual world, and that can be written as a pure quaternion $\boldsymbol{r} = [\boldsymbol{v}, \boldsymbol{0}]$**,** therefore a rotation quaternion R is $\boldsymbol{R_q(r) = qrq^{-1}} \in \mathbb{R}^3$.

VQS structure: Defined as a triplet $\boldsymbol{S} = [\boldsymbol{v}, \boldsymbol{q}, \boldsymbol{s}]$, where $\boldsymbol{v} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{0}] \in \mathbb{H}$ (pure quaternion), $\boldsymbol{q} = [\boldsymbol{v}, \boldsymbol{s}] \in \mathbb{H}$ (quaternion) and $\boldsymbol{s} \in \mathbb{R}$ (uniform scale). This structure is used in computer graphics as an efficient alternative to matrices transformation, since the amount of computations done in VQS are less than in matrix form. Consequently, VQS is used to compute the compute the translation, rotation and scale properties of a vector.

VQS transformation yields a pure quaternion and is noted as follows:

[1] More about Quaternions: https://en.wikipedia.org/wiki/Quaternion
Also, see the references for even more information

$\boldsymbol{S(r) = [v, q, s]r = q(sr)q^{-1} + v}$, where $\boldsymbol{r = [x, y, z, 0]} \in \mathbb{H}$

However, one particularity in VQS structure is that it only supports uniform scaling, which means that if the scale component is updated, the vector is scaled in in all the 3 axis x, y and z. As it can be seen, the whole vector is scaled by the s factor. (And this is where VQM takes place)

VQM set: It is defined as an extension of the VQS structure, in which the entire set has the same function as the VQS but the S component is replaced by a homogeneous Matrix, in order to support a non-uniform scale. Thus, the final form of the VQM set is as follows: Let T be a triplet such as $T = [v, q, M]$ where $\boldsymbol{v = [x, y, z, 0]} \in \mathbb{H}$ (pure quaternion), $\boldsymbol{q = [v, s]} \in \mathbb{H}$ (quaternion) and $\boldsymbol{M}$ is a 4x4 homogeneous quaternion matrix

- **VQM Algebraic Structure**

In this section we will introduce first of a few important operations and properties of quaternions (required to do the VQM math) and then, continue with the definitions of those of the VQM.

First of all, in order to understand the algebraic properties of a VQM set, we need to review how quaternions are multiplied by matrices. Thus, let **M** be a matrix such as $\boldsymbol{M = (m_i)} \in \boldsymbol{M_{4x4}}(\mathbb{R})$ where $\boldsymbol{m_i}$ ($\boldsymbol{i} \in \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}$) are the 4 column vectors of M and $\boldsymbol{q} \in \mathbb{H}$. Therefore:

Quaternion – Matrix Multiplication

$$\boldsymbol{q} \,.\, \boldsymbol{M} = (\boldsymbol{qm_i}) \text{ and } \boldsymbol{M} \,.\, \boldsymbol{q} = (\boldsymbol{m_i q})$$

or more explicitly:

$$\boldsymbol{q} \,.\, \boldsymbol{M} = \boldsymbol{q} \,.\, [\boldsymbol{m_1}, \boldsymbol{m_2}, \boldsymbol{m_3}, \boldsymbol{m_4}\,] = [\boldsymbol{qm_1}, \boldsymbol{qm_2}, \boldsymbol{qm_3}, \boldsymbol{qm_4}]$$

Important note here – Notice the "." operator introduced above. That does not mean dot product. That operator, in practice, means quaternion multiplication and the result of each multiplication, in the case of this Matrix Multiplication, will be a single column of $\boldsymbol{M}$.

Pure Quaternion – Matrix Multiplication

$\boldsymbol{e} \,.\, \boldsymbol{M} = (\boldsymbol{em_i}) = \boldsymbol{m_i} = \boldsymbol{M} = \boldsymbol{m_i} = (\boldsymbol{m_i e}) = \boldsymbol{M} \,.\, \boldsymbol{e}$, where $\boldsymbol{e} = [\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}]$ *(identity quaternion)*

Properties

Let q_1 and q_2 be quaternions, N and M homogeneous 4x4 matrix, r a vector in pure quaternion form

- $q_1 \,.\, (M \,.\, q_2) = (q_1 \,.\, M) \,.\, q_2$
- $q_1 \,.\, (q_2 \,.\, M) = (q_1 q_2) \,.\, M$
- $(M \,.\, q_1) \,.\, q_2 = M \,.\, (q_1 q_2)$

- $q(Mr)q^{-1} = (q\,.M\,.q^{-1})r$
- $(q_2\,.N\,.\,q_2^{-1})(q_1\,.M\,.\,q_1^{-1}) = q_2\,.\big(N(q_1\,.M\,.q_1^{-1})\big)\,.q_2^{-1}$, where q_1 and q_2 are quaternions and M and N are homogeneous quaternion matrices

Now, we *summarize the **VQM Algebraic operations and properties** as a list, since further information about how to demonstrate them is available in the original paper. So, n*ow that we know how to operate with quaternion-matrix multiplication and its properties, we can move on summarizing the VQM properties. For these properties, by following the notations described in the definitions section, then let:

- $M_1, M_2 \in M_{4x4}(\mathbb{R})$ (homogeneous quaternion matrices);
- $T = [v, q, M],\ T_1 = [v_1, q_1, M_1]\ ,\ T_2 = [v_2, q_2, M_2],\ T_3 = [v_3, q_3, M_3] \in \mathcal{T}$;
- $c \in \mathbb{R}$;

- Addition:

$$T_1 + T_2 = [v_1, q_1, M_1] + [v_2, q_2, M_2] = [v_1 + v_2,\ q_1 + q_2,\ M_1 + M_2]$$

- Scalar multiplication:

$$cT = c[v, q, M] = [cv, cq, cM]$$

- Multiplication on $\mathcal{T}$:

$$T_1T_2 = [v_1, q_1, M_1][v_2, q_2, M_2] = [q_2(M_2v_1)q_2^{-1} + v_2,\ q_2q_1,\ (q_1^{-1}.M_2\,.\,q_1)\ (q_1^{-1}.M_1\,.\,q_1)]$$

- Associativity:

$$[v_1, q_1, M_1]([\boldsymbol{v_2}, \boldsymbol{q_2}, \boldsymbol{M_2}][\boldsymbol{v_3}, \boldsymbol{q_3}, \boldsymbol{M_3}]) = ([\boldsymbol{v_1}, \boldsymbol{q_1}, \boldsymbol{M_1}][\boldsymbol{v_2}, \boldsymbol{q_2}, \boldsymbol{M_2}])[v_3, q_3, M_3]$$

Careful here, because if we had for instance $[v_1, 0, M_1]$ we cannot multiply this with anything on $\mathcal{T}$ since [0,0,0] does not exist in $\mathcal{T}^0$ which is not a vector space anymore. (Remember $q \in \mathbb{H} - \{0\}$).

- Invertibility

If M is invertible, then $q\,.M\,.q^{-1}$ is also invertible.

- Commutativity

This property is not defined for VQM because for quaternions multiplication is not either.

- **How to apply a transformation using VQM?**

Setting up a transformation VQM is very straightforward to understand since it is very similar to the VQS structure. Recall, a VQS transformation is define like this:

$\boldsymbol{S(r) = [v, q, s]r = q(sr)q^{-1} + v}$**,** where $\boldsymbol{r = [x, y, z, 0]} \in \mathbb{H}$

On the other hand, if we ever want to transform any pure quaternion (a vector in this form) $\boldsymbol{r = [x, y, z, 0]} \in \mathbb{H}$**,** we just multiply the VQM structure times the quaternion, just like we did with VQS or even with transformation matrices

Therefore, the VQM transformation will be defined like this:

$$\boxed{\boldsymbol{T(r) = [v, q, M]r = q(Mr)q^{-1} + v = (q \cdot M \cdot q^{-1})r + v}\text{, where } \boldsymbol{r = [x, y, z, 0]} \in \mathbb{H}} \quad \textbf{. . . (1)}$$

This operation is quite simple but, what if we had to transform a vector in a hierarchical model (A humanoid skeleton, for instance)? In this case, many operations are involved, such as multiplying a vector with vector, a quaternion with vector, matrix with vector, quaternion with quaternion and matrix with other matrix. We can organize the operations stated above in a similar way as we did with VQS, which is by concatenating the current VQM transform with that of its parent.

Recall the concatenation in VQS transform:

Let $\boldsymbol{S_1 = [v_1, q_1,\ s_1]}\ \ \boldsymbol{S_2 = [v_2, q_2,\ s_2]}$

$\boldsymbol{S_2\ S_1 =\ [v_2, q_2,\ s_2]\ [v_1, q_1,\ s_1] = [S_2 v_1,\ q_2 q_1,\ s_2 s_1]}$

So, for VQM we apply a similar form (Demonstration in the original paper):

Let $\boldsymbol{T_1 = [v_1, q_1,\ M_1]}\ \ \boldsymbol{T_2 = [v_2, q_2,\ M_2]}$

$$\boxed{\boldsymbol{T_2\ T_1 =\ [v_2, q_2,\ M_2]\ [v_1, q_1,\ M_1] = [T_2 v_1,\ q_2 q_1,\ M_s]}} \quad \textbf{. . . (2)}$$

Highlights:

- The vector part represents the translation contribution of the transform, the quaternion part represents the rotation contribution, and the matrix part represents the non-uniform scale contribution.
- The **order is important**. If you want to concatenate an object transform with that of its parent, $\boldsymbol{T_2}$ will be the parent's VQM transform and $\boldsymbol{T_1}$ will be the child's.
- The vector part of the resulting VQM is (in that order) a **VQM times a vector** operation.
- The vector to be transformed must have the form of a **pure quaternion** ($\boldsymbol{r = [x, y, z, 0]} \in \mathbb{H}$), this is convenient in order to be compatible with the homogeneous quaternion matrices $\boldsymbol{M_1}\ and\ \boldsymbol{M_2}$

- The structure of $\boldsymbol{M_1}$ or $\boldsymbol{M_2}$ is a homogeneous matrix, as follows:

$$M = \begin{bmatrix} \boldsymbol{m_{00}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{m_{11}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{m_{22}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

  Where: $\boldsymbol{m_{ij}}; \boldsymbol{i = j}$ are the scaling factors of each transformation.
- As stated in the first section of this paper, a vector, represented by a pure quaternion (position in the world) $\boldsymbol{r} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{0}]$ can be rotated to r' by $\boldsymbol{r'} = (\boldsymbol{q}\, \boldsymbol{r}\, \boldsymbol{q^{-1}})$
- The matrix $\boldsymbol{M_s}$ comes from rotating each matrix $\boldsymbol{M_2}$ and $\boldsymbol{M_1}$ by $\boldsymbol{q_2}$ and $\boldsymbol{q_1}$ respectively and multiplying those results, as in: $\boldsymbol{M_s} = (\boldsymbol{q_2} \cdot \boldsymbol{M_2} \cdot \boldsymbol{q_2^{-1}})(\boldsymbol{q_1} \cdot \boldsymbol{M_1} \cdot \boldsymbol{q_1^{-1}})$
- **Remember** that the "·" operator is not dot product.

- **Data structure for a VQM in C++**

Now, with all the mathematical foundation described in the previous sections of this paper, we can move forward to look for a data structure that can be added to the math library of our own game engine.

As for VQM, we can define it as a class or struct, however for the purpose of this paper, a class will be used.

Class design: For the design of the class, we will assume that the reader has already implemented data structures for Quaternion, Vector3 and Matrix4.

Class structure:

- Class type: VQM
- Private member attributes:
  - translator: this data will be of type Quaternion (for convenience in operations programming).
  - rotator: This data will be of type Quaternion.
  - scaler: This data will be of type Matrix4
- Private member functions:
  - concatenate(): Returns an object of type VQM, and receives two parameters of type VQM.
  - transformVector(): Returns a Vector3 and receives 4 parameters: the vector to be transformed (Vector3), a translator (Vector3), a rotator (Quaternion), and a scaler (Matrix4).
- Public member functions:

- A default constructor, to create a default VQM $\boldsymbol{T} = [\vec{\boldsymbol{0}}, \boldsymbol{I_q}, \boldsymbol{I_m}]$**;** where $\vec{\boldsymbol{0}}$ is a zero vector, $\boldsymbol{I_q}$ is an identity quaternion and $\boldsymbol{I_m}$ is an identity matrix.
- A second constructor to let the client build a VQM from some input: Receives 4 parameters: the vector to be transformed (Vector3), a translator (Vector3), a rotator (Quaternion), and a scaler (Matrix4).
- A destructor.
- Operator* overload 1: Returns a VQM type and receive a reference to a constant VQM
- Operator* overload 2: Returns a VQM type and receive a reference to a constant Vector3.
- Getters: Functions to get the values of the translator, rotator and scale properties.
- ToMatrix(): We need a mechanism to convert the VQM structure to a transformation matrix because current graphics cards do not support the VQM.

- **Implementation in C++**

The following listings are a proposal on how to implement a VQM structure in C++. The reader can use this proposal (as well as the code in the appendix section) in their own math libraries without any copyright restrictions, and use it and/or modify as required for their own game engines. (see next page). Take into account that the notation of the quaternions used in the listings are quite different from the paper. The paper uses $\boldsymbol{Q} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{S}]$ while the code uses $\boldsymbol{Q} = [\boldsymbol{S}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$

Listing 1: VQM.h

```
#pragma once
#include "Quaternion.h"
#include "Vector.h"
#include "Matrix.h"


class VQM
{
private:

	Quaternion translator; // pure quaternion
	Quaternion rotator;
	Matrix4 scaler;

	VQM concatenate(VQM vqm_1, VQM vqm_2);
	Vector3 transformVector(Vector3 v, Vector3 translator, Quaternion rotator, Matrix4
scaler);

public:
	VQM();
	VQM(Vector3 V, Quaternion Q, Matrix4 M) :
		translator(Quaternion(0.0f, V)), rotator(Q), scaler(M)
	{
		rotator.Normalize();
	}
	~VQM();

	Vector3 operator*(Vector3 const & rhs);
	VQM operator*(VQM const & rhs);

	Vector3 GetV() { return translator.Vector; }
	Quaternion GetQ() { return rotator; }
	Matrix4 GetM() { return scaler; }

	// We need a matrix, because graphics card does not support VQM structures
	Matrix4 ToMatrix();
};
```

Listing 2: VQM.cpp

```
#include "VQM.h"

VQM::VQM()
{
	Quaternion I; // identity quaternion
	Matrix4 M; // identity matrix
	this->translator = Quaternion(0.0f, Vector3(0.0f)); // zero vector in pure
	//quaternion form
	this->rotator = I;
	this->scaler = M;
}


VQM::~VQM()
{
}

VQM VQM::concatenate(VQM vqm_1, VQM vqm_2)
{
	//Using formula (2) from paper:
	//[
	//	vqm_2*vqm2_1.GetV(),
	//	vqm_2.GetQ * vqm_1.GetQ(),
	//	(vqm_2.GetQ*vqm_2.GetM()*vqm_2.GetQ()^-1) *	(vqm_1.GetQ * vqm_1.GetM()
	//	* vqm_1.GetQ()^-1)]
	// ]

	// compute the translation vector component
	Vector3 V = vqm_2 * vqm_1.GetV();

	// compute the quaternion component
	Quaternion Q = vqm_2.GetQ() * vqm_1.GetQ();

	// compute the matrix product
	Matrix4 ml = Quaternion::Qprod(vqm_2.GetQ(), Quaternion::Qprod(vqm_2.GetM(),
	vqm_2.GetQ() ^ -1));
	Matrix4 mr = Quaternion::Qprod(vqm_1.GetQ(), Quaternion::Qprod(vqm_1.GetM(),
	vqm_1.GetQ() ^ -1));
	Matrix4 M = ml * mr;

	return VQM(V, Q, M);
}

Vector3 VQM::operator*(Vector3 const & rhs)
{
	Vector3 v = this->translator.Vector;
	Quaternion q = this->rotator;
	Matrix4 m = this->scaler;
	return transformVector(rhs, v, q, m);
}

VQM VQM::operator*(VQM const & rhs)
{
	return concatenate(*this, rhs);
}
```

```
Vector3 VQM::transformVector(Vector3 v, Vector3 V, Quaternion Q, Matrix4 M)
{
	//Using formula (1) from paper: [Q*(M*r)*Q^-1] + V

	Quaternion r = Quaternion(0.0f, v);
	Quaternion rotM = M * r;
	Quaternion r_prime = Q * (rotM * (Q^-1));
	Quaternion pureV = Quaternion(0.0f, V);
	Vector3 result = (r_prime + pureV).Vector;
	return result;
}


Matrix4 VQM::ToMatrix()
{
	Matrix4 Ms;
	Ms[0][0] = this->scaler[0][0];
	Ms[1][1] = this->scaler[1][1];
	Ms[2][2] = this->scaler[2][2];

	Matrix4 Mv;
	Mv[3][0] = this->translator.Vector.x;
	Mv[3][1] = this->translator.Vector.y;
	Mv[3][2] = this->translator.Vector.z;

	Matrix4 Mq = this->rotator.ToMatrix();
	Mq = Matrix4.Transpose(Mq);
	Matrix4 M = Mv * Mq * Ms;

	return M;
}
```

- **References**

Michael Aristidou and Xin Li, “The VQM-Group and its Applications”, International Journal of Algebra, Vol. 2, 2008, no. 19, 905 – 918

Xin Li, “VQS Transformation with an Incremental Approach”

Xin Li, Cody Luitjens and Michael Beach, “Transformation and Interpolation with VQM Structure”

Rick Parent, “Computer Animation Algorithms and Techniques”, Morgan Kaufmann Publishers. Quaternions chapter. Page 450.

APPENDIX

- **Appendix 1**

In order to implement the listings above, the reader will need to implement a Quaternion class as well. Since Quaternion math is out of scope of this paper, see the references for more information. However, code for Quaternion class is provided, again, with no copyrights restrictions.

Download Quaternion Class Here

goo.gl/qiwL5E

- **Appendix 2**

Summary of operations with quaternions

- Addition:

$$\boldsymbol{q_{1+}\, q_2} = [\boldsymbol{S_1}, \boldsymbol{V_1}] + [\boldsymbol{S_2}, \boldsymbol{V_2}] = [\boldsymbol{S_1} + \boldsymbol{S_2}, \boldsymbol{V_1} + \boldsymbol{V_2}]$$

- Scalar Multiplication:

$$\boldsymbol{cq_2} = \boldsymbol{c}[\boldsymbol{S}, \boldsymbol{V}] = [\boldsymbol{cS}, \boldsymbol{cV}]$$

- Quaternion Multiplication:

$$\boldsymbol{q_1 q_2} = [\boldsymbol{S_1}, \boldsymbol{V_1}][\boldsymbol{S_2}, \boldsymbol{V_2}] = [\boldsymbol{S_1 S_2} - (\boldsymbol{V_1} \cdot \boldsymbol{V_2}), \boldsymbol{S_1 V_2} + \boldsymbol{S_2 V_1} + (\boldsymbol{V_1} \times \boldsymbol{V_2})]$$

- Dot Product of Quaternions:

$$\boldsymbol{q_1} \cdot \boldsymbol{q_2} = [\boldsymbol{S_1}, \boldsymbol{V_1}] \cdot [\boldsymbol{S_2}, \boldsymbol{V_2}] = \boldsymbol{S_1 S_2} + \boldsymbol{V_1} \cdot \boldsymbol{V_2}$$

- Quaternion Magnitude:

$$|\boldsymbol{q}| = |[\boldsymbol{S}, \boldsymbol{V}]| = \sqrt{\boldsymbol{S^2} + \boldsymbol{V^2}}$$

- Quaternion Conjugate:

$$\overline{\boldsymbol{q}} = \overline{[\boldsymbol{S}, \boldsymbol{V}]} = [\boldsymbol{S}, -\boldsymbol{V}]$$

- Quaternion Inverse

$$\boldsymbol{q^{-1}} = [\boldsymbol{S}, \boldsymbol{V}]^{-1} = \boldsymbol{q_1} \frac{\overline{\boldsymbol{q_2}}}{|\boldsymbol{q_2}|}$$

- Identity Quaternion

$$\boldsymbol{I_q} = \boldsymbol{q q^{-1}} = [-\mathbf{1}, \overline{\mathbf{0}}] = [-\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}]$$