

[読者になる](#)

LIGHT11

2018-08-02

【数学】ゲーム開発で覚えておくべきQuaternion（クオータニオン/四元数）の性質

Math

3D空間で回転を計算する時によく使われるQuaternion（四元数）について簡単にまとめます。
ゲーム開発で必要な知識のみ。深入りはしません。

- [Quaternionとその表現方法](#)
- [基礎知識](#)
- [Quaternionの積](#)
- [回転に使う](#)
- [実装してみる](#)
- [結果](#)
- [関連](#)
- [参考](#)

Quaternionとその表現方法

Quaternionとは複素数のように、虚数部分を持つ数です。

ただし複素数と違い、虚数単位を3つ持ります。

$$q = xi + yj + zk + w$$

x, y, z, w は実数、 i, j, k は虚数単位です。

上記の式は各実数を便宜的に4次元ベクトルの要素とみなして次のようにも表現されます。

$$q = (x, y, z, w)$$

さらに便宜的に虚数部の (x, y, z) を3次元ベクトル \vec{v} とみなすことで次のようにも表現されます。

$$q = (\vec{v}, w)$$

まとめると以下のようになります。

表現の形が違うだけですべてQuaternionを表しています。

$$q = xi + yj + zk + w = (x, y, z, w) = (\vec{v}, w)$$

基礎知識

Quaternionで回転の計算をするのに必要な基礎知識がいくつかあります。

まず、Quaternionをベクトルとみなしたとき、
ベクトルと同様にノルム（長さ）が求められます。

$$|q| = \sqrt{x^2 + y^2 + z^2 + w^2}$$

次に、虚数部に-1を掛けたものを共役Quaternion q^* として定義します。

$$q^* = -xi - yj - zk + w$$

Quaternionと共役Quaternionとの間には、
積がノルムの二乗になるという関係があります。

$$qq^* = |q|^2$$

次に逆Quaternion q^{-1} を定義します。

逆Quaternionは元のQuaternionとの積が1となるQuaternionです。

$$qq^{-1} = 1$$

この式はここまで学習した知識を使って次のように変形できます。

$$q^{-1} = \frac{1}{q} = \frac{q^*}{|q|^2}$$

したがって、 q のノルムが1(単位Quaternion)ならば、
逆Quaternionと共役Quaternionは等しくなります。

$$q^{-1} = q^*$$

Quaternionの積

Quaternion同士の積は次のように計算できます。

$$q_1 q_2 = (\vec{v}_1 \times \vec{v}_2 + w_2 \vec{v}_1 + w_1 \vec{v}_2, w_1 w_2 - \vec{v}_1 \cdot \vec{v}_2)$$

これを頑張って解くと次のようになります。

$$\begin{aligned} q_1 q_2 &= (x_1 w_2 + w_1 x_2 - z_1 y_2 + y_1 z_2, \\ &\quad y_1 w_2 + z_1 x_2 + w_1 y_2 - x_1 z_2, \\ &\quad z_1 w_2 - y_1 x_2 + x_1 y_2 + w_1 z_2, \\ &\quad w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \end{aligned}$$

回転に使う

単位ベクトル $\vec{u} = (u_1, u_2, u_3)$ を軸として θ だけ回転させると、まずは下記のようなQuaternionを定義します。

$$q = \left(\vec{u} \sin\left(\frac{\theta}{2}\right), \cos\left(\frac{\theta}{2}\right) \right)$$

三次元のベクトル $p = (p_1, p_2, p_3)$ を上記のQuaternionで回転させたベクトル p' を求めるには、

$$p' = qpq^*$$

を求めます。

ただしQuaternionと次元数を合わせるために、計算上は $p = (p_1, p_2, p_3, 1)$ とします。

ここで \vec{u} は単位ベクトルなので、 $|q| = 1$ となります。 $(\sin^2\theta + \cos^2\theta = 1)$

よって $q^{-1} = q^*$ なので、上の式は次のように変換できます。

$$p' = qpq^{-1}$$

実装してみる

さてここまで内容をUnityで実装してみます。

```
using UnityEngine;

namespace Sample
{
    [System.Serializable]
    public struct Quaternion
    {
        public float x;
        public float y;
        public float z;
        public float w;

        /// <summary>
        /// 共役Quaternion
        /// </summary>
        private Quaternion Conjugated 共役
        {
            get { return new Quaternion(-x, -y, -z, w); }
        }

        public static Quaternion operator*(Quaternion a, Quaternion b)
        {
            // Quaternion同士の積の計算
            return new Quaternion(
左側                a.x * b.w + a.w * b.x - a.z * b.y + a.y * b.z, - x
左側                a.y * b.w + a.z * b.x + a.w * b.y - a.x * b.z,
左側                a.z * b.w - a.y * b.x + a.x * b.y + a.w * b.z,
左側                a.w * b.w - a.x * b.x - a.y * b.y - a.z * b.z - w(実部)
            );
        }

        public static Vector3 operator *(Quaternion a, Vector3 b)
        {
            // ベクトルをQuaternionに変換
            var bQuaternion = new Quaternion(b.x, b.y, b.z, 1);
            // q * p * q^-1 でベクトルを回転
            var pos = a * bQuaternion * a.Conjugated;
            return new Vector3(pos.x, pos.y, pos.z);
        }

        /// <summary>
        /// 回転角度と回転軸からQuaternionを作成する
        /// </summary>
    }
}
```

```

public static Quaternion AngleAxis(float angle, Vector3 axis)
{
    var rad = angle * Mathf.Deg2Rad;
    var halfRad = rad * 0.5f;
    var sin = Mathf.Sin(halfRad);
    var cos = Mathf.Cos(halfRad);
    axis.Normalize();

    return new Quaternion(axis.x * sin, axis.y * sin, axis.z * sin, cos);
}

public Quaternion(float x, float y, float z, float w)
{
    this.x = x;
    this.y = y;
    this.z = z;
    this.w = w;
}
}
}

```

とりあえずQuaternion同士の積とVectorの回転をできるようにしました。
パフォーマンスとかは気にしてません。

これを使うスクリプトを書いてみます。

```

using UnityEngine;

public class QuaternionSample : MonoBehaviour
{
    private Sample.Quaternion _rotate;

    private void Awake()
    {
        transform.position = Vector3.up;
        _rotate = Sample.Quaternion.AngleAxis(1, Vector3.forward) * Sample.Quaternion.AngleAxis(1, Vector3.up);
    }

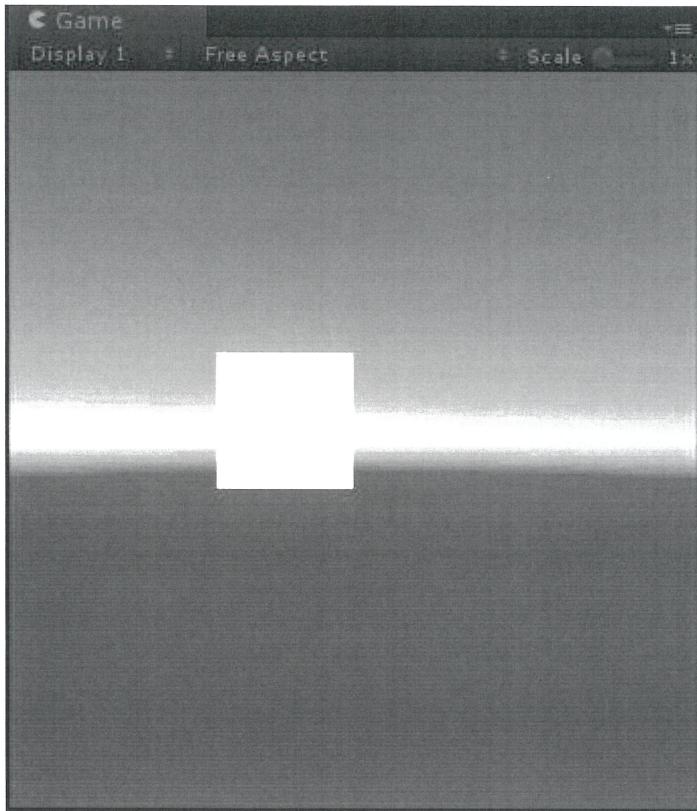
    private void Update()
    {
        transform.position = _rotate * transform.position;
    }
}

```

QuaternionをAwake()であらかじめ合成して、Update()でベクトルを回転させています。

結果

前節のスクリプトを適当なGameObjectにアタッチして再生してみます。



うまく動いていそうです。

関連



LIGHT11
id:halya_11



Hatena Blog

【数学】ゲーム開発で覚えておくべき複素数の性質

ゲーム開発に必要な複素数の知識をまとめてみました。あくまでゲーム開発に必要な部分のみ。深入りはしません。



2018-07-24 00:13

light11.hatenadiary.com

参考

Quaternionによる3次元の回転変換 - Qiita

コンピュータグラフィックスにおいて、図形を変換するには、ベクトルやマトリックス（行列）の演算が多用されます。その中でも、Quaternion(= 4元数 = 虚数単位が3つある複素数)を用いて...

ternionによる3次元の回

tirananabe



qiita.com

<http://www.mss.co.jp/technology/report/pdf/18-07.pdf>

Haruma:K (id:halya_11) 3年前



0

0

ツイート

6

シェア

読者になる

193

コメントを書く

« [【Unity】【エディタ拡張】非
MonoBehaviou...](#)

[【Unity】現在値と目的値を補間してアニ
メ...»](#)

プロフィール



Haruki Yano / Haruma-K(@harumak_11)
Unityエンジニア。

記事の質は気分によって大きく上下いたします。
できるだけ正確な情報を心がけてますが間違い・知識不足など気づいたら教えてくださいませ。

※当ブログの内容は所属団体とは関係ありません

読者になる

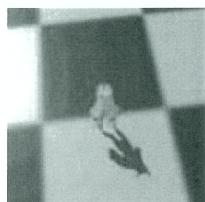
193

[このブログについて](#)

検索

記事を検索

注目記事



2019-10-24

[【Unity】Cinemachineの全機能を解説！ありとあらゆるカメラワークを高クオリティに作る方法総まとめ](#)