CHUYỂN ĐỘNG QUAY TRONG KHÔNG GIAN VỚI ỨNG DỤNG TRONG LẬP TRÌNH VIDEO GAME

ROTATION IN SPACE WITH APPLICATIONS IN VIDEO GAMES

Nguyễn Mai Quyên[1], Chu Bình Minh[2], Hà Bình Minh[3]

[1]Khoa Toán kinh tế, Trường Đại học Kinh tế quốc dân
[2]Khoa Khoa học cơ bản, Trường Đại học Kinh tế - Kỹ thuật Công nghiệp
[3]Khoa Hệ thống thông tin quản lý, Trường Đại học Ngân hàng Thành phố Hồ Chí Minh

Đến Tòa soạn ngày 19/03/2020, chấp nhận đăng ngày 09/04/2020

Tóm tắt: Chuyển động quay là một trong những chuyển động phức tạp trong các video game. Bài báo sẽ giải thích việc sử dụng ma trận để tạo nên chuyển động quay, từ khái niệm toán học cơ bản cho đến ví dụ cụ thể.

Từ khóa: Phép quay, video games.

Abstract: Rotation is one of the most sophisticated movements in video games. In this paper we will explain from abstract concepts to concrete example that how matrix theory is used in rotations.

Keywords: Rotation, video games.

1. ĐẶT VẤN ĐỀ

Trò chơi điện tử (game) ngày càng đa dạng, phổ biến, và đem lại nhiều ích lợi cũng như trải nghiệm cho người chơi. Nhiều trò chơi được thiết kế với mục đích giáo dục, giúp cho trẻ em học ngôn ngữ lập trình, học phương pháp tư duy, học cách giải quyết vấn đề, chẳng hạn như Scrach (do MIT Media Lab phát triển), Minecraft (do Mojang phát triển), Roblox (do Roblox Corporationc phát triển),…

Trong lịch sử phát triển trò chơi điện tử, đáng chú ý là sự phát triển của các video game vào những năm cuối những năm 1990 [1]. Các video game mô phỏng những hình ảnh 3 chiều, mang lại cho người chơi những trải nghiệm gần với thực tế. Dưới góc nhìn của những chuyên gia lập trình game, phía sau những hình ảnh 3 chiều trong game là sự tổng hợp của rất nhiều những kỹ thuật tiên tiến về đồ họa máy tính, toán học, vật lý học, công nghệ mô phỏng, kỹ thuật lập trình,… (xem [1], [2]).

Hình 1. Một hình ảnh 3 chiều trong game Minecraft (thiết kế bởi Hà Tuệ Minh teky_00042)

Một trong những công cụ toán học được sử dụng rộng rãi trong việc lập trình và phát triển các video game là lý thuyết ma trận (xem [1, 2, 5]). Việc hiểu rõ các công thức toán học, đặc biệt về ma trận, là rất cần thiết đối với những chuyên gia lập trình game. Những công cụ phát triển video game đòi hỏi lập trình viên phải có hiểu biết sâu sắc về ma trận, như OpenGL (do Khronos Group phát triển), WebGL (do Khronos WebGL Working Group phát triển),

DirectX (do Microsoft phát triển),...

Các đối tượng trong video game (như không gian, bản đồ, vật thể, nhân vật, vũ khí…) đều là những đối tượng ảo mô phỏng lại thế giới 3 chiều trong thực tế (xem [1]). Những đối tượng này trong game cũng có sự chuyển động, biến đổi, giống hệt như trong thực tế. Đằng sau việc mô phỏng sự chuyển động của các đối tượng trong video game đó là một quá trình tính toán của máy tính, đặc biệt là việc sử dụng ma trận trong quá trình tính toán này. Bài báo này sẽ giải thích cụ thể về việc sử dụng ma trận để tạo một trong những chuyển động phức tạp trong video game, đó là chuyển động quay. Bài báo sẽ trình bày từ những khái niệm toán học cơ bản cho đến ví dụ cụ thể.

Cấu trúc của bài báo được trình bày như sau: Phần 2 sẽ giới thiệu về ma trận trực giao và vai trò của nó trong video game. Phần 3 sẽ mô tả các công thức toán học của chuyển động quay trong không gian. Phần 4 sẽ mô tả một ứng dụng cụ thể. Cuối cùng là kết luận sẽ được đưa ra trong Phần 5.

2. MA TRẬN TRỰC GIAO

2.1. Ma trận trực giao 3×3 và tính chất

Ma trận được sử dụng trong rất nhiều chuyển động trong các video game, như chuyển động cơ bản (trước sau, trên dưới, trái phải, quay), chuyển động phức hợp (là kết hợp là chuyển động cơ bản). Trong đó, chuyển động quay là một trong những chuyển động khó vì cần đến nhiều tính toán phức tạp để mô phỏng loại chuyển động này. Ma trận để tính toán những chuyển động quay này là ma trận trực giao, được định nghĩa như sau:

Định nghĩa. Ma trận A cỡ 3×3 được gọi là *ma trận trực giao* (hay còn gọi là *ma trận quay*) nếu $A^T = A^{-1}$.

Tính chất của ma trận trực giao:

- Định thức của A bằng 1 hoặc −1.
- Các vectơ cột của ma trận A là cơ sở trực chuẩn của $\mathbb{R}^3$.
- Ma trận A có một giá trị riêng bằng 1. Vector riêng của A ứng với giá trị riêng bằng 1 chính là trục quay của ma trận A.

2.2. Vai trò của ma trận trực giao trong video game

Việc sử dụng ma trận trong các video game, đặc biệt trong những chuyển động phức tạp như chuyển động quay, có những ưu điểm sau (xem [1]).

Ưu điểm:

- *Tính góc xoay dễ dàng.* Lý do là các phần tử của ma trận có mối liên hệ với các góc xoay thông qua các hàm lượng giác. Đây là ưu điểm mà không biểu diễn nào khác có thể thực hiện được.
- *Định dạng đồ họa API* (Application Programming Interface) là định dạng để giao tiếp với phần cứng chuyên về đồ hoạ. Định dạng này sử dụng ma trận để định hướng, tính toán.
- *Tính góc giữa các đối tượng dễ dàng.* Lý do là các thông tin về góc xoay được cho trong ma trận, nên khi ta biết góc giữa đối tượng A so với đối tượng B và góc giữa đối tượng B so với đối tượng C thì ta có thể dễ dàng xác định được góc của A so với C.
- *Tính ma trận nghịch đảo dễ dàng.* Điều này có thể thực hiện dễ dàng do ma trận quay là trận trực giao nên ma trận nghịch đảo là ma trận chuyển vị.

Tuy nhiên, một ma trận trực giao 3×3 cần đến 9 phần tử để lưu trữ, mặc dù theo lý thuyết chỉ cần sử dụng 3 tham số là đủ để biểu diễn một ma trận trực giao. Cách biểu diễn này của ma trận có một số hạn chế như sau (xem [1]).

Hạn chế:

- *Về bộ nhớ.* Chẳng hạn, ta xét một đoạn phim hoạt hình có một nhân vật trong khung hình. Mỗi hoạt động của nhân vật này là sự kết hợp của 20 phần cơ thể. Trong một khung hình, mỗi phần cơ thể được xác định sự chuyển động

bằng một hướng và được biểu diễn bằng một ma trận quay. Để cho nhân vật chuyển động thì ta cần có tối thiểu 15 khung hình trong 1 giây. Tức là, ta cần lưu trữ 300 ma trận trực giao cho mỗi giây. Mặc dù vậy, vấn đề này không quá nghiêm trọng do kỹ thuật lưu trữ dữ liệu ngày càng được nâng cấp về mặt dung lượng và tốc độ.

- *Không dễ hình dung.* Biểu diễn phép quay bằng ma trận không dễ hình dung đối với những lập trình viên mới vào nghề. Về mặt tự nhiên, con người luôn có thiên hướng định hướng theo các góc quay hơn là theo các ma trận.

- *Nhiều phương trình toán học biểu diễn ma trận.* Một ma trận quay gồm có 9 phần tử nhưng chỉ phụ thuộc vào 3 tham số, nên có nhiều cách biểu diễn ma trận khác nhau, có thể gây ra đôi chút lúng túng cho những lập trình viên chưa có kinh nghiệm.

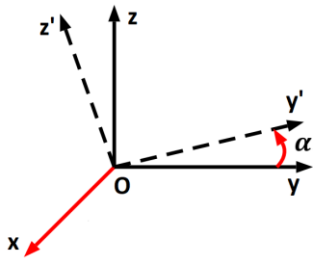
3. PHÉP QUAY TRONG KHÔNG GIAN

3.1. Các phép quay cơ bản và góc Euler

Ba phép quay cơ bản quanh các trục tọa độ Ox, Oy, Oz cùng với ba góc Euler tương ứng được định nghĩa như sau (xem [3, 5]):

- *Phép quay quanh trục Ox:* Nếu ta giữ nguyên trục Ox và quay mặt phẳng Oyz quanh trục Ox một góc α, ta thu được hệ trục tọa độ 3 chiều mới Oxy'z' như trong hình 2. Góc α được gọi là *góc roll*. Ma trận tương ứng với phép quay này là:

$$R_x(\alpha)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha\\ 0 & \sin\alpha & \cos\alpha\end{bmatrix} \quad (1)$$



Hình 2. Phép quay quanh trục Ox một góc α: Giữ nguyên trục Ox và quay mặt phẳng Oyz quanh trục Ox một góc α

- ***Phép quay quanh trục Oy:*** Tương tự như góc α, nếu ta giữ nguyên trục Oy và quay mặt phẳng Oxz quanh trục Oy một góc β, ta thu được hệ trục tọa độ 3 chiều mới Ox'yz'. Góc β được gọi là *góc pitch*. Ma trận tương ứng với phép quay này là:

$$R_y(\alpha)=\begin{bmatrix}\cos\beta & 0 & \sin\beta\\ 0 & 1 & 0\\ -\sin\beta & 0 & \cos\beta\end{bmatrix} \quad (2)$$

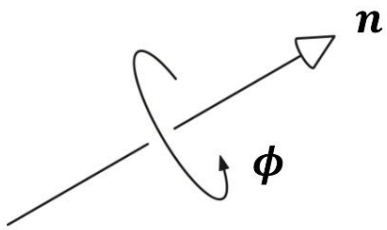
- ***Phép quay quanh trục Oz:*** Nếu ta giữ nguyên trục Oz và quay mặt phẳng Oxy quanh trục Oz một góc γ, ta thu được hệ trục tọa độ 3 chiều mới Ox'y'z. Góc γ được gọi là *góc yaw*. Ma trận tương ứng với phép quay này là:

$$R_z(\gamma)=\begin{bmatrix}\cos\gamma & -\sin\gamma & 0\\ \sin\gamma & \cos\gamma & 0\\ 0 & 0 & 1\end{bmatrix} \quad (3)$$

Để tham khảo chi tiết về các góc Euler, độc giả có thể tìm hiểu tài liệu [3, 5].

3.2. Phép quay quanh một trục bất kỳ

Ta ký hiệu $R(n,\phi)$ là phép quay quanh vector n một góc ϕ, theo như minh họa trong hình 3. Ở đây, n là vector đơn vị và góc ϕ biến thiên từ $-\pi$ đến π.



Hình 3. Phép quay quanh trục n một góc ϕ

Các phép quay cơ bản trong các công thức (1), (2), và (3) là những trường hợp đặc biệt của phép quay $R(n,\phi)$, theo như dưới đây.

Tính chất của phép quay $R(n,\phi)$:

- Giả sử *i, j, k* là các vector đơn vị tương ứng với mỗi trục tọa độ Ox, Oy, Oz. Khi đó, các

phép quay cơ bản trong các công thức (1), (2), và (3) tương ứng như sau:

$R_x(\alpha) = R(i, \alpha)$

$R_y(\beta) = R(j, \beta)$

$R_z(\gamma) = R(k, \gamma)$

- Ngoài ra, $R(n, \phi) = R(-n, -\phi)$.
- Phép quay ngược với $R(n, \phi)$ là $R(-n, \phi)$ hoặc $R(n, -\phi)$.

3.3. Xác định ma trận trực giao từ trục quay và góc quay

Bài toán 1. Giả sử vector đơn vị n có tọa độ trong không gian là $n=(n_x, n_y, n_z)$. Ta cần xác định ma trận trực giao A của phép quay $R(n, \phi)$ như miêu tả trong mục 3.2.

Ma trận A trong Bài toán 1 được cho bởi công thức sau [3].

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \quad (4)$$

trong đó:

$A_{11} = n_x n_x (1-\cos\phi) + \cos\phi$

$A_{21} = n_x n_y (1-\cos\phi) + n_z \sin\phi$

$A_{31} = n_x n_z (1-\cos\phi) - n_y \sin\phi$

$A_{12} = n_x n_y (1-\cos\phi) - n_z \sin\phi$

$A_{22} = n_y n_y (1-\cos\phi) + \cos\phi$

$A_{32} = n_y n_z (1-\cos\phi) + n_x \sin\phi$

$A_{13} = n_x n_z (1-\cos\phi) + n_y \sin\phi$

$A_{23} = n_y n_z (1-\cos\phi) - n_y \sin\phi$

$A_{33} = n_z n_z (1-\cos\phi) + \cos\phi$

Hàm số MATLAB **RotaAxis** sau đây cho phép ta tính toán ma trận quay khi biết vector n và góc ϕ.

```
function A = RotaAxis(n,phi)
  % Returns rotation matrix A from n and phi.
  % USAGE: A = RotaAxis(n,phi)
  A = [cos(phi)          -n(3)*sin(phi)       n(2)*sin(phi);
       n(3)*sin(phi)      cos(phi)       -n(1)*sin(phi);
      -n(2)*sin(phi)      n(1)*sin(phi)       cos(phi)];
  A = (1-cos(phi))*n'*n+A;
```

3.4. Xác định trục quay và góc quay từ ma trận trực giao

Bài toán 2. Giả sử ma trận trực giao A của phép quay $R(n, \phi)$ như sau:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

Ta cần xác định các thành phần của vector đơn vị $n=(n_x, n_y, n_z)$ và góc ϕ qua các phần tử của A.

Góc quay ϕ được xác định bởi công thức [3]

$$\cos\phi A = \frac{1}{2}\left(A_{11} + A_{22} + A_{33} - 1\right) \quad (5)$$

và vectơ đơn vị $n=(n_x, n_y, n_z)$ được xác định bởi

$$n_x = \frac{A_{32} - A_{23}}{2\sin\phi}, n_y = \frac{A_{13} - A_{31}}{2\sin\phi}, n_z = \frac{A_{21} - A_{12}}{2\sin\phi}$$

Hàm số MATLAB **RotaAxisInverse** sau đây trả về vectơ n và góc quay ϕ khi biết ma trận A.

```
function [n,phi] = RotaAxisInverse(A)
% Returns the vector n and angle phi from A
% USAGE: [n,phi] = RotaAxisInverse(A)
phi=acos(0.5*((trace(A)-1));
  n(1)=0.5*(A(3,2)-A(2,3))/sin(phi);
  n(2)=0.5*(A(1,3)-A(3,1))/sin(phi);
  n(3)=0.5*(A(2,1)-A(1,2))/sin(phi);
```

3.5. Mối quan hệ giữa phép quay $R(n, \phi)$ và các phép quay cơ bản

Nếu ta thực hiện phép quay theo thứ tự z-y-x như trong [4], ta có $R(n, \phi) = R_x(\alpha)R_y(\beta)R_z(\gamma)$. So sánh giữa hai ma trận A và $R(n, \phi)$, quan hệ giữa (n, ϕ) và các góc Euler được cho bởi các phương trình sau:

$$\sin\beta = A_{13} = n_x n_z(1-\cos\phi) + n_y \sin\phi$$

$$\tan\alpha = \frac{-A_{23}}{A_{33}} = \frac{-n_y n_z(1-\cos\phi) + n_x \sin\phi}{n_z n_z(1-\cos\phi) + \cos\phi}$$

$$\tan\gamma = \frac{-A_{12}}{A_{11}} = \frac{-n_x n_y(1-\cos\phi) + n_z \sin\phi}{n_x n_x(1-\cos\phi) + \cos\phi}$$

4. ỨNG DỤNG PHÉP QUAY TRONG VIDEO GAME

Bài toán áp dụng. Cho trước vector đơn vị n và góc quay ϕ, ta cần tìm ảnh hình khối lập phương trên qua phép quay $R(n,\phi)$).

Lời giải bài toán.

Ta xét bài toán trên cho khối lập phương C với 8 đỉnh có tọa độ tương ứng là 8 vector cột trong ma trận cỡ 3×8 sau:

$$C = \begin{bmatrix} 4 & 5 & 4 & 5 & 4 & 5 & 4 & 5 \\ 1 & 1 & 2 & 2 & 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Giả sử C_R là ảnh của khối lập phương C khi thực hiện phép quay với n=(2/3, 2/3,1/3) và $\phi = -\pi$.

Ta sẽ giải bài toán trên theo 2 cách. Cách thứ nhất, khối lập phương C_R sẽ được tìm trực tiếp thông qua phép quay $R(n,\phi)$. Cách thứ hai, ta sẽ tìm C_R một cách gián tiếp thông qua các phép quay cơ bản bằng việc tính toán các góc Euler.

Phương pháp 1 (tính trực tiếp)

Bước 1. Tính ma trận quay của $R(n,\phi)$. Theo công thức (3.4), ta có:

$$A = \begin{bmatrix} -0.1111 & 0.8889 & 0.4444 \\ 0.8889 & -0.1111 & 0.4444 \\ 0.4444 & 0.4444 & -0.7778 \end{bmatrix}$$

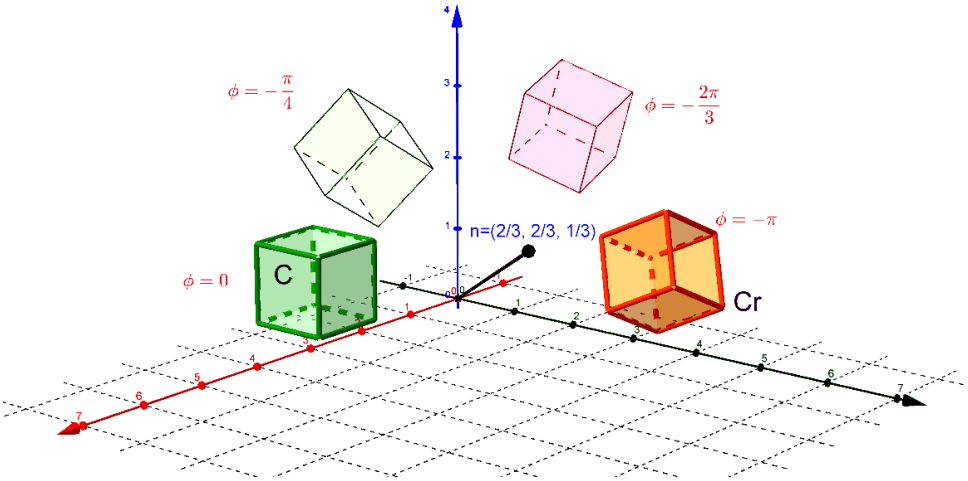
Bước 2. Tính ma trận tọa độ của C_R là tích của A và C:

$$C_R = AC$$

Quá trình tính toán được thực hiện bởi hàm RotaCube trong MATLAB như sau.

```
function Cr = RotaCube(C,n,phi)
% Returns the image's coordinate of cube C
% when rotating C around vector n at an angle phi.
% USAGE: Cr = RotaCube(C,n,phi)
A=RotaAxis(n,phi);
Cr=A*C;
```

Dữ liệu của ma trận C_R được cho trong hàng cuối của bảng 1. Hình 4 minh họa quá trình quay khối lập phương C theo vector n với góc ϕ lần lượt nhận các giá trị $\phi \in \left\{0, -\frac{\pi}{4}, -\frac{2\pi}{3}, -\pi\right\}$.



Hình 4. Các khối lập phương thu được trong Phương pháp 1 khi quay quanh trục n một góc ϕ, với góc ϕ lần lượt nhận các giá trị $\phi \in \left\{0, -\frac{\pi}{4}, -\frac{2\pi}{3}, -\pi\right\}$

Phương pháp 2 (thông qua các phép quay cơ bản)

Từ vectơ n và góc quay ϕ, ta sẽ tính được ma trận xoay A. Theo [4], từ A, ta sẽ tìm được các góc α, β, γ theo thứ tự quay z-y-x bằng hàm euler_angle(A). Sau đó, ảnh của C nhận được sau 3 phép quay cơ bản $R_z(\gamma)$, $R_y(\beta)$, và $R_x(\alpha)$. Quá trình tính toán được thực hiện theo các bước sau.

Bước 1. Tính ma trận quay A của $R(n, \phi)$. Ta làm tương tự như Bước 1 trong Phương pháp 1 bằng cách sử dụng hàm RotaAxis1(n,phi).

Bước 2. Tính các góc Euler. Các góc Euler nhận được từ ma trận A bằng cách sử dụng hàm euler_angle(A) trong [4]. Cụ thể:

$$(\alpha_1, \beta_1, \gamma_1) = (-2.6224,\ 0.4606,\ -1.6952)$$

$$(\alpha_2, \beta_2, \gamma_2) = (0.5191,\ 2,6810,\ 1.4464)$$

Từ đây, ta sẽ thực hiện tính toán với trường hợp $(\alpha_1, \beta_1, \gamma_1)$, trường hợp $(\alpha_2, \beta_2, \gamma_2)$ được tính toán tương tự.

Bước 3. Thực hiện phép quay khối lập phương C bằng phép quay $R_z(\gamma_1)$, $\gamma_1 = -1.6952$, với ma trận quay

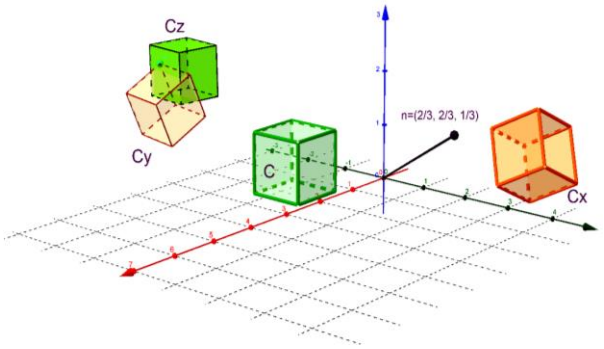
$$R_z(\gamma_1) = \begin{bmatrix} -0.1240 & 0.9923 & 0 \\ -0.9923 & -0.1240 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ta thu được khối lập phương với tọa độ các đỉnh là các vector cột của ma trận $C_z = R_z C$ được cho trong Bảng 1.

Bước 4. Thực hiện phép quay khối lập phương C_z bằng phép quay $R_y(\beta_1)$, $\beta_1 = 0.4606$, với ma trận quay

$$R_y(\beta_1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.8682 & 0.4961 \\ 0 & -0.4961 & -0.8682 \end{bmatrix}$$

ta thu được khối lập phương với tọa độ các đỉnh là các vector cột của ma trận $C_y = R_y C_z$ được cho trong bảng 1.



Hình 5. Các khối lập phương thu được trong phương pháp 2 khi quay C bởi các phép quay cơ bản theo thứ tự z-y-x. Khối lập phương C_x trùng với khối lập phương C_R trong phương pháp 1

Bước 5. Thực hiện phép quay khối lập phương C_y bằng phép quay $R_x(\alpha_1)$, $\alpha_1 = -2.6224$, với ma trận quay

$$R_x(\alpha_1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.8682 & 0.4961 \\ 0 & -0.4961 & -0.86821 \end{bmatrix}$$

ta thu được khối lập phương với tọa độ các đỉnh là các vector cột của ma trận $C_x = R_x C_y$ được cho trong bảng 1.

Các bước 3, bước 4 và bước 5 được tính toán bởi hàm RotaCubeEuler trong MATLAB như sau.

```
function [Cz Cy Cx] =
RotaCubeEuler(C,alpha,beta,gamma)
% Returns the coordinates of the image of C cube
% when rotating C using basic rotations.
% USAGE: [Cz Cy Cx] =
% RotaCubeEuler(C,alpha,beta,gamma)
Rz=[cos(gamma) -sin(gamma) 0;
        sin(gamma) cos(gamma) 0;
        0 0 1];
Rx=[1 0 0;
        0 cos(alpha) -sin(alpha);
        0 sin(alpha) cos(alpha)];
Ry=[cos(beta) 0 sin(beta);
        0 1 0;
       -sin(beta) 0 cos(beta)];
```

```
Cz=Rz*C;
Cy=Ry*Cz;
Cx=Rx*Cy;
```

Nhận xét. Các bước tính toán trong Phương pháp 2 được minh họa trong hình 5. Ta thấy rằng thực hiện bằng hai cách đều thu được cùng một ảnh của khối lập phương *C*, tức là:

$$C_x \equiv C_R$$

Khẳng định này được thể hiện rõ ràng trong các hình 4, hình 5 và dữ liệu trong bảng 1.

Bảng 1. Tọa độ khối lập phương C ban đầu và tọa độ các đỉnh khi quay lập phương theo thứ tự z-y-x. Hàng cuối là khối lập phương C_R thu được từ phương pháp 1, trùng với khối lập phương C_x thu được từ phương pháp 2

Khối lập phương	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6	Đỉnh 7	Đỉnh 8
Khối lập phương ban đầu C	4.0000	5.0000	4.0000	5.0000	4.0000	5.0000	4.0000	5.0000
	1.0000	1.0000	2.0000	2.0000	1.0000	1.0000	2.0000	2.0000
	1.0000	1.0000	1.0000	1.0000	2.0000	2.0000	2.0000	2.0000
$C_z = R_zC$ thu được bằng cách quay C theo $R_z(-1.6952)$	0.4961	0.3721	1.4884	1.3644	0.4961	0.3721	1.4884	1.3644
	−4.0931	−5.0854	−4.2172	−5.2095	−4.0931	−5.0854	−4.2172	−5.2095
	1.0000	1.0000	1.0000	1.0000	2.0000	2.0000	2.0000	2.0000
$C_y = R_yC_z$ thu được bằng cách quay C_z theo $R_y(0.4604)$	0.8889	0.7778	1.7778	1.6667	1.3333	1.2222	2.2222	2.1111
	−4.0931	−5.0854	−4.2172	−5.2095	−4.0931	−5.0854	−4.2172	−5.2095
	0.6753	0.7304	0.2343	0.2894	1.5711	1.6262	1.1301	1.1852
$C_x = R_xC_y$ thu được bằng cách quay C_y theo $R_x(-2.6224)$	0.8889	0.7778	1.7778	1.6667	1.3333	1.2222	2.2222	2.1111
	3.8889	4.7778	3.7778	4.6667	4.3333	5.2222	4.2222	5.1111
	1.4444	1.8889	1.8889	2.3333	0.6667	1.1111	1.1111	1.5556
C_R (khối lập phương thu được theo phương pháp 1)	0.8889	0.7778	1.7778	1.6667	1.3333	1.2222	2.2222	2.1111
	3.8889	4.7778	3.7778	4.6667	4.3333	5.2222	4.2222	5.1111
	1.4444	1.8889	1.8889	2.3333	0.6667	1.1111	1.1111	1.5556

5. KẾT LUẬN

Chúng tôi đã xây dựng các hàm MATLAB để tính toán các công thức của phép quay quanh một trục bất kỳ. Dựa vào các hàm này, chúng tôi đã xây dựng một ví dụ minh họa để mô phỏng một chuyển động quay trong video game theo 2 phương pháp khác nhau và cho kết quả phù hợp. Những kỹ thuật mô phỏng tiên tiến hơn, như kỹ thuật quaternion, sẽ là những vấn đề được thảo luận trong các nghiên cứu tiếp theo của chúng tôi.

TÀI LIỆU THAM KHẢO

[1] Fletcher Dunn, Ian Parberry, "3D Math Primer for Graphics and Game Development", CRC Press, (2011).

[2] Nataša Lončarić, Damira Keček, Marko Kraljić, “Matrices in Computer Graphics”, Technical Journal 12, 2, 120-123, (2018).

[3] Simon, L.A., *“Rotations, Quaternions, and Double Groups”,* Dover Publications, New York, (2005).

[4] Chu Bình Minh, Hà Bình Minh, Nguyễn Mai Quyên, “Góc Euler trong điều khiển cánh tay robot với 6 bậc tự do”, Tạp chí Khoa học và Công nghệ, đã nhận đăng (2020).

[5] Kuipers, B.J., *“Quaternions and* Rotation Sequences*”*, Princeton University Press, New Jersey, (1999).

Thông tin liên hệ: **Nguyễn Mai Quyên**

Điện thoại: 0914026515 - Email: nguyen-mai-quyen@neu.edu.vn
Khoa Toán kinh tế, Trường Đại học Kinh tế quốc dân.

Chu Bình Minh

Điện thoại: 0912207854 - Email: cbminh@uneti.edu.vn
Khoa Khoa học cơ bản, Trường Đại học Kinh tế - Kỹ thuật Công nghiệp.

Hà Bình Minh

Điện thoại: 0976788196 - Email: minhhb@buh.edu.vn
Khoa Hệ thống thông tin quản lý, Trường Đại học Ngân hàng
Thành phố Hồ Chí Minh.