DOCUMENTATION

Este documento contiene la lista de funciones utilizadas y una descripción de su funcionamiento en el programa realizado con Matlab que consiste en rotar un cubo mediante la técnica de Arcball.

```
function my_MouseMoveFcn(obj,event,hObject)

if xmouse > xlim(1) && xmouse < xlim(2) && ymouse > ylim(1) && ymouse < ylim(2)

    % Calculate m1
    m1 = Map2DPointsTo3D(xmouse,ymouse);
    % Get past info
    m0 = handles.m0;
    q0 = handles.q0;
    % Calculate delta quaternion
    dq = QuaternionFromTwoVectors(m0,m1);
    dq = dq/norm(dq);
    % Calculate actual quaternion
    qk = MultQuaternion(dq,q0);
    % Transform and publish differents attitudes
    UpdateAttitudes(qk, handles);
    % Redraw Cube
    handles.Cube = RedrawCube(qk,handles.Cube);
    % Save actual data
    handles.m0 = m1;
    handles.q0 = qk;
```

La función "`function my_MouseMoveFcn(obj,event,hObject)`" es llamada cuando se detecta un evento de drag en la interfaz del programa y comprueba si el ratón se encuentra dentro de los límites de la esfera.

Si se cumple dicha condición ejecutará los pasos del diagrama de Arcball que consisten en:

- Calcular la posición del ratón en el espacio tridimensional de la esfera convirtiendo un punto 2D de la pantalla del monitor a un punto 3D. Llama a la punción "`Map2DPointsTo3D`".

- El siguiente paso consiste en recuperar la información guardada en cada frame de la aplicación. Estos datos son:

    - La posición del ratón del frame anterior guardado en "`handles.m0`"
    - El cuaternión del frame anterior guardado en "`handles.q0`".

  Sí aún no ha habido un evento de drag "`handles.m0`" está inicializado a [0;0;0] y "`handles.q0`" está inicializado a [1;0;0;0].

- Se calcula el delta cuaternión que representa el incremento de rotación que ha habido. Para ello se utiliza la función "`QuaternionFromTwoVectors`" que convierte dos vectores en un cuaternión. Este cuaternión se normaliza para su posterior tratamiento.

- En este paso se calcula el cuaternión actual llamando a la función "`MultQuaternion`" que multiplica el delta cuaternión por el cuaternión del frame anterior para obtener como resultado el cuaternión de rotación actual.

- Actualizamos todas las variables de los paneles de parametrización y de la matriz de rotación. Se llama a la función "`UpdateAttitudes`" que se encarga de calcular todos los parámetros y setearlos por pantalla.

- Una vez hecho los cálculos y obtenido el nuevo cuaternión se vuelve a dibujar el cubo con la nueva rotación llamando a la función "`RedrawCube`" que se encarga de rotar la matriz de puntos del cubo y dibujarlo con la nueva orientación.

- Por último se guardan los datos de la posición del ratón en la figura tridimensional y el cuaternión actual.

Si la posición del ratón se encuentra fuera de los límites de la esfera entonces no se ejecuta el diagrama anterior.

UpdateAttitudes(q, handles)

Cuando esta función es llamada calcula todos los parámetros de los paneles de la interfaz y los setea para que el usuario pueda visualizar los datos actuales del cubo rotado.

- Inputs:
    - q: cuaternión
    - handles: puntero a los datos de la interfaz GUIDE

- El primer paso es calcular una matriz de rotación en función del cuaternión que recibimos como parámetro con la función "`RotationMatrix`". Esta matriz de rotación nos servirá para calcular los demás parámetros.

- Set Quaternion: Setear los componentes del cuaternión imprimido con los valores del cuaternión recibido.

- Set Euler principal Angles and axis: Llamamos a la función "`rotMat2Eaa`" para extraer de la matriz de rotación un ángulo y un vector director unitario con los que seteamos los valores imprimidos por pantalla.

- Set Euler Angles: Llamamos a la función "`rotM2eAngles`" para extraer los ángulos de euler yaw, pitch y roll y setearlos por pantalla.

- Set Rotation Vector: Con el vector unitario y el ángulo extraídos de la matriz de rotación hacemos el producto entre estos componentes y obtenemos el vector de rotación. Seteamos sus componentes por pantalla.

- Por último imprimimos los 9 componentes de la matriz de rotación.

handles.Cube = RedrawCube(qk,handles.Cube)

La función "`RedrawCube`" recibe como parámetros el cuaternión actual y el objeto cubo. Llama a la función "`RotationMatrix`" con la que se crea una matriz de rotación a partir del cuaternión. Se hace el producto entre la matriz de rotación y la matriz de puntos originales del cubo y se devuelve el resultado siendo esta la nueva orientación de nuestro cubo.

- Inputs:
    - qk: cuaternión actual
    - handles.Cube: objeto cubo
- Outputs:
    - handles.Cube: objeto cubo con la nueva orientación

m = Map2DPointsTo3D(x,y)

Esta función permite convertir un punto 2D con coordenadas en el viewport en coordenadas 3D en la superficie de una figura tridimensional con la finalidad de poder rotarla. Para una transición suave se ha utilizado una superficie hiperbólica que al intersectar con la superficie de la esfera se consigue un círculo con una altura constante. La siguiente ecuación muestra la obtención de la tercera coordenada a través de las dos primeras teniendo en cuenta si se encuentra dentro del círculo:

$$\boldsymbol{m}(x,y) = \begin{cases} \left(x,\, y,\, +\sqrt{r^2 - x^2 - y^2}\right)^{\mathsf{T}} & x^2 + y^2 < \frac{1}{2}r^2 \\ \dfrac{r\left(x,\, y,\, \frac{r^2}{2\sqrt{x^2+y^2}}\right)^{\mathsf{T}}}{\left\|\left(x,\, y,\, \frac{r^2}{2\sqrt{x^2+y^2}}\right)\right\|} & x^2 + y^2 \geq \frac{1}{2}r^2 \end{cases}$$

- Inputs:
  - x: posición x del ratón
  - y: posición y del ratón
- Outputs:
  - m: posición del ratón en la esfera que rodea al cubo [x;y;z]

q = QuaternionFromTwoVectors(u,v)

Quaternion From Two Vectors nos sirve para extraer el cuaternión que nos permite transformar el primer vector en el segundo vector.

La ecuación utilizada se ha extraído de la siguiente página web:

"Lol Engine, a community project. Beautiful maths simplication: http://lolengine.net/blog/2013/09/18/beautiful-maths-quaternion-from-vectors "

- Inputs:
  - u: vector
  - v: vector
- Outputs:
  - q: cuaternión

w = MultQuaternion(q,p)

Esta función será utilizada para calcular el cuaternión actual mediante el producto entre el delta cuaternión y el cuaternión del frame anterior.

$$\mathring{q}_k = \delta\mathring{q}_k\mathring{q}_{k-1}$$

La función recibe como parámetros dos cuaterniones, el cuaternión “q” y el cuaternión “p”. Previamente a los cálculos, se hará uso de 4 variables auxiliares:

- “q0” y “p0” se utilizarán para almacenar la parte real de los cuaterniones.
- “qv” y “pv” se utilizarán para guardar la parte imaginaria.

Para el producto de cuaterniones se ha utilizado la siguiente fórmula:

$$qp = \begin{pmatrix} q_0\, p_0 \; - q^T p \\ q_0\, p + p_0\, q + qXp \end{pmatrix}$$

- Inputs:
  - q: primer cuaternión
  - p: segundo cuaternión
- Outputs:
  - w: resultado de la multiplicación de los 2 cuaterniones

R = RotationMatrix(q)

Esta función sirve para crear una matriz de rotación a través de un cuaternión.
La función recibe como parámetro un cuaternión denominado “q”.

En caso de que el cuaternión recibido sea un cuaternión identidad la matriz de rotación formada será la matriz identidad.

En caso contrario se calculará la matriz de rotación mediante la siguiente fórmula:

$$R(q) = (q_0^2 \; - q^T q)I_3 \; + 2qq^T \; + 2q_0\,[q]_x$$

- Inputs:
  - q: cuaternión
- Outputs:
  - R: Matriz de rotación

q = RotationMatrix2Quaternion(R)

Esta función extrae de una matriz de rotación el cuaternión codificado en su interior. Para ello calculamos los componentes del cuaternión a través de las siguientes ecuaciones:

$$\begin{aligned} 4q_0^2 &= 1 + Tr(R) \\ 4q_1^2 &= 1 - Tr(R) + 2R_{11} \\ 4q_2^2 &= 1 - Tr(R) + 2R_{22} \\ 4q_3^2 &= 1 - Tr(R) + 2R_{33} \end{aligned}$$

Pero los resultados obtenidos no son precisos porque se pierde la información de su signo al hacer la raíz cuadrada. Por ello nos ayudamos de otra serie de ecuaciones para resolver esta incógnita:

Case1 : In case that $q_0^2 > q_i^2$ for $i = 1, 2, 3$, which is achieved if trace $(\mathbf{R}) \geq 0$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \sqrt{1 + r_{11} + r_{22} + r_{33}} \\ \frac{r_{32} - r_{23}}{\sqrt{1+r_{11}+r_{22}+r_{33}}} \\ \frac{r_{13} - r_{31}}{\sqrt{1+r_{11}+r_{22}+r_{33}}} \\ \frac{r_{21} - r_{12}}{\sqrt{1+r_{11}+r_{22}+r_{33}}} \end{bmatrix} \tag{8}$$

Case2 : In case that $q_1^2 > q_i^2$ for $i = 0, 2, 3$ which is achieved if trace $(\mathbf{R}) < 0$ and $r_{11} = \max(r_{11}, r_{22}, r_{33})$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \frac{r_{32} - r_{23}}{\sqrt{1+r_{11}-r_{22}-r_{33}}} \\ \sqrt{1 + r_{11} - r_{22} - r_{33}} \\ \frac{r_{21} + r_{12}}{\sqrt{1+r_{11}-r_{22}-r_{33}}} \\ \frac{r_{13} + r_{31}}{\sqrt{1+r_{11}-r_{22}-r_{33}}} \end{bmatrix} \tag{9}$$

Case3 : In case that $q_2^2 > q_i^2$ for $i = 0, 1, 3$ which is achieved if trace $(\mathbf{R}) < 0$ and $r_{22} = \max(r_{11}, r_{22}, r_{33})$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \frac{r_{13} - r_{31}}{\sqrt{1-r_{11}+r_{22}+r_{33}}} \\ \frac{r_{21} + r_{12}}{\sqrt{1-r_{11}+r_{22}+r_{33}}} \\ \sqrt{1 - r_{11} + r_{22} + r_{33}} \\ \frac{r_{32} + r_{23}}{\sqrt{1-r_{11}+r_{22}+r_{33}}} \end{bmatrix} \tag{10}$$

Case4 : In case that $q_3^2 > q_i^2$ for $i = 0, 1, 2$ which is achieved if trace $(\mathbf{R}) < 0$ and $r_{33} = \max(r_{11}, r_{22}, r_{33})$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \frac{r_{21} - r_{12}}{\sqrt{1-r_{11}-r_{22}+r_{33}}} \\ \frac{r_{13} + r_{31}}{\sqrt{1-r_{11}-r_{22}+r_{33}}} \\ \frac{r_{32} + r_{23}}{\sqrt{1-r_{11}-r_{22}+r_{33}}} \\ \sqrt{1 - r_{11} - r_{22} + r_{33}} \end{bmatrix} \tag{11}$$

Con esta serie de ecuaciones y condiciones se consigue extraer un cuaternión de una matriz de rotación. Ha sido utilizada en los botones “Update” de los ejes principales de Euler y en “Update” de los ángulos de Euler para enviarlo como parámetro a las funciones “`UpdateAttitudes`” y “`RedrawCube`”.

- Inputs:
  - R: Matriz de rotación
- Outputs:
  - q: cuaternión

R = Eaa2rotMat(a,u)

La función recibe 2 parámetros "a" un ángulo de rotación y "u" un vector, que son las variables que tenemos en el ángulo de Euler del guide.
Esta función empieza normalizando el vector "u" y posteriormente realizamos la siguiente ecuación, con ella obtendremos la matriz de rotación y posteriormente la retornamos.

$$R = I_3\ cos(a) + (1 - cos(a))(uu')) + [u]_x\ sin(a)$$

- Inputs:
  - a: angle of rotation
  - u: axis of rotation
- Outputs:
  - R: generated rotation matrix

R = eAngles2rotM(yaw, pitch, roll)

Utilizando los ángulos de Euler construimos una matriz para cada eje, después se multiplican las 3 matrices para obtener la matriz de rotación, se retorna la matriz.

$$Myaw = \begin{bmatrix} cos(yaw) & sin(yaw) & 0 \\ -sin(yaw) & cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Mroll = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(roll) & -sin(roll) \\ 0 & sin(roll) & cos(roll) \end{bmatrix}$$

$$Mpitch = \begin{bmatrix} cos(pitch) & 0 & -sin(pitch) \\ 0 & 1 & 0 \\ sin(pitch) & 0 & cos(pitch) \end{bmatrix} \qquad R = Myaw' * Mpitch' * Mroll'$$

- Inputs:
    - yaw: ángulo de rotación en el eje Z
    - pitch:ángulo de rotación en el eje Y
    - roll: ángulo de rotación en el eje X
- Outputs:
    - R: Matriz de rotación

[yaw, pitch, roll] = rotM2eAngles(R)

Esta función extrae de la matriz de rotación los ángulos de Euler. La matriz de rotación en función de los ángulos de euler tiene la siguiente forma:

$$\mathbf{R} = \mathbf{R}(\psi, \theta, \phi) = \begin{pmatrix} c_\theta c_\psi & c_\psi s_\theta s_\phi - c_\phi s_\psi & c_\psi c_\phi s_\theta + s_\psi s_\phi \\ c_\theta s_\psi & s_\psi s_\theta s_\phi + c_\phi c_\psi & c_\phi s_\psi s_\theta - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{pmatrix}$$

Conociendo el valor de cada posición podemos extraer los tres ángulos, cada uno siguiendo su respectiva fórmula:

$$\theta = arcsin(-r_{31}) \qquad \phi = arctan2\left(\frac{r_{32}}{cos\theta}, \frac{r_{33}}{cos\theta}\right) \qquad \psi = arctan2\left(\frac{r_{21}}{cos\theta}, \frac{r_{11}}{cos\theta}\right)$$

- Inputs:
    - R: Matriz de rotación
- Outputs:
    - yaw: angle of rotation around the z axis
    - pitch: angle of rotation around the y axis
    - roll: angle of rotation around the x axis

[a,u] = rotMat2Eaa(R)

Esta función nos permitirá obtener los ejes y el ángulo de Euler utilizando la matriz de rotación que que recibe como parámetro.
Primero encontramos el ángulo a partir de la traza de la matriz:

$$a = arcos(\frac{Tr(R) - 1}{2})$$

Para encontrar los componentes del vector unitario conseguimos [u]x a partir del ángulo y la matriz de rotación:

$$[u]_x = \frac{R - R^t}{2sin(a)}$$

Y con esto cogemos los 3 componentes de [u]x para hacer el vector unitario:

$$\begin{aligned} u_1 &= [u]_{x32} \\ u_2 &= [u]_{x13} \\ u_3 &= [u]_{x21} \end{aligned}$$

Una vez obtenidos los 3 componentes del vector unitario, los retornamos junto con el ángulo de rotación.

- Inputs:
  - R: Matriz de rotación
- Outputs:
  - a: ángulo de rotación
  - u: vector unitario

ResetCube_Callback(hObject, eventdata, handles)

Cuando se presiona el botón reset el cubo vuelve a su posición original y toda la parametrización toma los valores correspondientes a la matriz de rotación cero.

Se setea la posición del ratón a 0 handles.m0=[0;0;0] y se setea el cuaternión al cuaternión identidad handles.q0=[1;0;0;0]. Se llama la función "`UpdateAttitudes`" para setear los parámetros y la función "`RedrawCube`" para volver a dibujar el cubo en su posición inicial.

UpdateQuaternion_Callback(hObject, eventdata, handles)

Coge los valores introducidos por el usuario en el panel de “Quaternion”, lo convierte en un cuaternión unitario y setea toda la parametrización en función de este nuevo cuaternión. Por último vuelve a dibujar el cubo en función a la nueva orientación.

UpdateAngleAxis_Callback(hObject, eventdata, handles)

Coge los valores introducidos por el usuario en el panel de “Euler principal Angle and Axis”, transforma el vector en un vector unitario, construye una matriz de rotación mediante el vector y el ángulo con la función “`Eaa2rotMat`” y extrae el cuaternión de la matriz utilizando la función “`RotationMatrix2Quaternion`”. Normaliza el cuaternión y setea toda la parametrización en función de este nuevo cuaternión. Por último vuelve a dibujar el cubo en función a la nueva orientación.

UpdateEulerAngles_Callback (hObject, eventdata, handles)

Coge los valores introducidos por el usuario en el panel de “Euler Angles”, con los tres ángulos de Euler construye una matriz de rotación utilizando la función “`eAngles2rotM`” y extrae el cuaternión de la matriz utilizando la función “`RotationMatrix2Quaternion`”. Normaliza el cuaternión y setea toda la parametrización en función de este nuevo cuaternión. Por último vuelve a dibujar el cubo en función a la nueva orientación.

UpdateRotationVector_Callback(hObject, eventdata, handles)

Coge los valores introducidos por el usuario en el panel de “Rotation Vector”, con el vector de rotación extrae el ángulo y el vector unitario y construye el cuaternión. Normaliza el cuaternión y setea toda la parametrización en función de este nuevo cuaternión. Por último vuelve a dibujar el cubo en función a la nueva orientación.

Diagrama Arcball