Documento de diseño, parcial de Introducción a la Computación Gráfica

Briam Daniel Agudelo Mogollón

Septiembre 2019

1 Introducción

El presente documento pretende explicar las ideas de diseño utilizadas para la solución del problema que se propuso en el parcial de este curso, se explicará como se solucionó cada uno de los problemas que se presentaron al momento de idear la solución, esto dividiéndolos en dos grupos:

- **Gráfico:** Que elementos de dibujo provistos por OpenGL se utilizaron para modelar gráficamente cada elemento del mundo.
- **Software:** Que conceptos matemáticos (físicos) y estrategias de modelado de Computación Gráfica se utilizaron para poder representar el comportamiento esperado del brazo robótico.

Luego de esto se presentará un manual de uso con imágenes ilustrativas, explicando la secuencia de comandos necesaria para poder interactuar correctamente con el brazo. Las imágenes servirán como guía para poder interpretar los cambios (colores) de ciertos elementos en el mundo, esto para tener una mejor experiencia al momento manipular el brazo.
Por último se resaltarán algunas conclusiones resultantes del desarrollo del programa.

2 Diseño

2.1 Gráfico

En el mundo se pueden visualizar los siguientes elementos:

- **Suelo:** El suelo esta representado mediante un prisma decagonal centrado en el origen y orientado hacia arriba, usando la primitiva *gluDisk()*.
- **Uniones entre los brazos(articulaciones):** Las articulaciones están representadas mediante esferas sólidas, usando la primitiva *gluSphere()*.

- **Cuerpo del brazo:** El cuerpo del brazo está representado mediante segmentos de líneas, que unen cada una de las articulaciones mencionadas anteriormente, usando la primitiva *glBegin(GL_LINES)*.

- **Una bola:** La bola con la que interactuará el brazo, la cual se encuentra inmóvil y en una posición inicial especifica. Esta bola está representada mediante una esfera construida a partir de segmentos de linea(no sólida), usando la primitiva *gluSphere()*.

- **Base:** Representación de los ejes coordenados del espacio *(X, Y y Z)* con los colores: rojo, verde y azul respectivamente.

2.2 Software

A continuación se presentan algunos problemas que supuso el desarrollo la de solución:

2.2.1 ¿Como modelar la jerarquía en el brazo robótico?

Para que cualquier rotación en alguna de las articulaciones afecte a las demás articulaciones que dependen de ella, la jerarquía de componentes se implementa de tal forma que:

- Cada nodo dentro del árbol de jerarquía tiene únicamente un hijo.

- Hay tantos niveles en el árbol de jerarquía como cantidad de vectores o partes movibles del brazo(especificadas en el archivo de texto plano).

- Cada nodo posee una articulación (esfera sólida) y conectada a ella un segmento de brazo(segmento de línea).

2.2.2 ¿Que acciones realizadas sobre algún nodo del árbol de jerarquía se propagarán a los nodos dependientes?

Las secciones del brazo se pueden representar mediante un *vector* en R^3. Por lo tanto:

- Cuando se especifica la rotación θ, en alguno de los ejes *(X, Y y Z)* y sobre alguna sección del brazo(nodo), los nodos dependientes de este deberán realizar la misma rotación θ (en el mismo eje) sobre el *vector* que representa cada sección del brazo(nodo dependiente).

- Cuando se realiza la rotación anterior sobre cada *vector* se debe *trasladar* a todos los nodos dependientes (de forma recurrente) donde termina el vector trasladado del nodo actual. **Esto al final produce el mismo efecto que si se rotara cada sección del brazo con la primitiva *glRotatef()***, pero el no haberlo hecho de esta manera tiene una *justificación*, la cual se explica a continuación.

La razón de no utilizar la primitiva de rotación de OpenGL es para mantener la orientación de los ejes de rotación y que sean los mismos para todos los nodos del árbol de jerarquía. Esto al final tendrá un valor importante para la solución del siguiente problema.

Nota: El algoritmo utilizado para realizar la rotación sobre los vectores hace uso de las matrices de rotación vistas en clase.

2.2.3 ¿Como conocer la posición actual del extremo del brazo robótico y validar si se encuentra los suficientemente cerca a la bola para poder agarrarla?

El hecho de no alterar o deformar (rotar o escalar) el "espacio" de dibujado en cada uno de los nodos, permite conocer la posición actual del extremo del brazo en cualquier instante. Esto mediante una *suma ponderada de vectores*.

Ahora conociendo la posición actual del extremo del brazo y la posición actual de la pelota, podemos verificar si el brazo se encuentra lo suficientemente cerca para poder agarrar la pelota y calcular la distancia entre los dos, esto mediante la siguiente ecuación:

$$d = \sqrt{(E_x - B_x)^2 + (E_y - B_y)^2 + (E_z - B_z)^2} \tag{1}$$

Dónde E es el vector que representa la posición del extremo del brazo y B el vector que representa la posición de la bola. De esta forma si $d \leq r$(donde r es el radio de la bola) entonces es posible agarrar la bola.

2.2.4 Simulación de caída libre, rebote y cámara:

El movimiento de caída libre se puede separar en dos momentos, los cuales se alternan mientras la bola siga rebotando:
Sean y_i (posición en y inicial) y g(gravedad) elementos conocidos.

- **Bajada(antes de rebotar):** Para este caso la posición en el instante t esta dada por la siguiente fórmula:

$$y_t = y_i - \frac{1}{2} \cdot g \cdot t^2 \tag{2}$$

  El criterio de finalización de este caso se da cuando y_t sea próximo a cero. Luego de esto $v_i = g \cdot t \cdot 0.8$ y se procede con el siguiente caso

- **Subida(después de rebotar):** Para este caso la posición en el instante t esta dada por la siguiente fórmula:

$$y_t = v_i \cdot t - \frac{1}{2} \cdot g \cdot t^2 \tag{3}$$

El criterio de finalización de este caso se da cuando v_t sea próximo a cero, teniendo en cuenta la ecuación:

$$v_t = v_i - g \cdot t \tag{4}$$

Luego de esto $y_i = y_t$ y se procede con el caso anterior.

En relación con el modelado de la cámara se utilizó el implementado por el profesor.

3 Manual de uso

1. Compilar el programa con el comando: *g++ -std=c++17 main.cxx -lm -lGL -lGLU -lglut -o ejecutable*

2. Ejecutar el programa con el comando: *./ejecutable archivo.txt*
(donde "archivo.txt" especifica la topología del brazo)

3. Se iniciará el programa mostrando los elementos del mundo(ver figura siguiente).

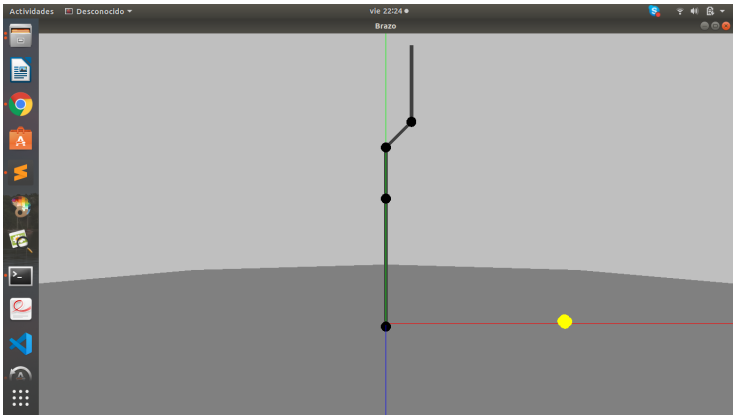

Figure 1: Imagen paso 3.

4. Aquí se puede seleccionar cualquier articulación mediante cualquier **número(1...n)**, con lo cual la articulación seleccionada cambiará de color a blanco(ver figura 2).

5. Luego de esto se puede seleccionar el eje de rotación *(X, Y y Z)*, con lo cual el color de la articulación ahora cambiará al color del eje de rotación seleccionado *(X = Rojo, Y = Verde y Z = Azul)*. **Para volver al principio y poder seleccionar el número de articulación nuevamente se debe presionar la tecla "q"**(ver figura 3).

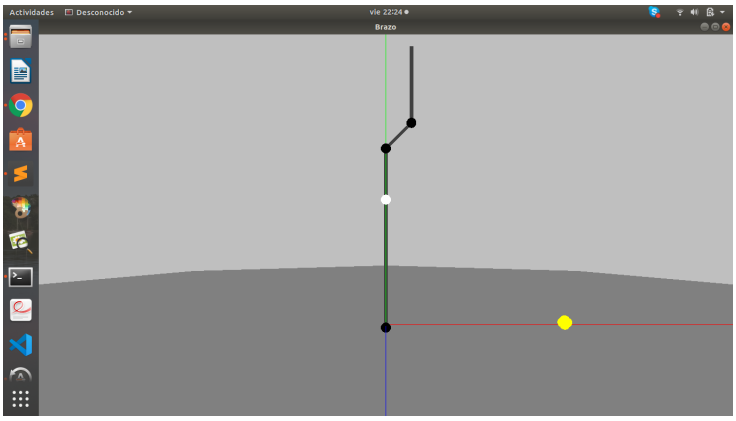

Figure 2: Imagen paso 4.

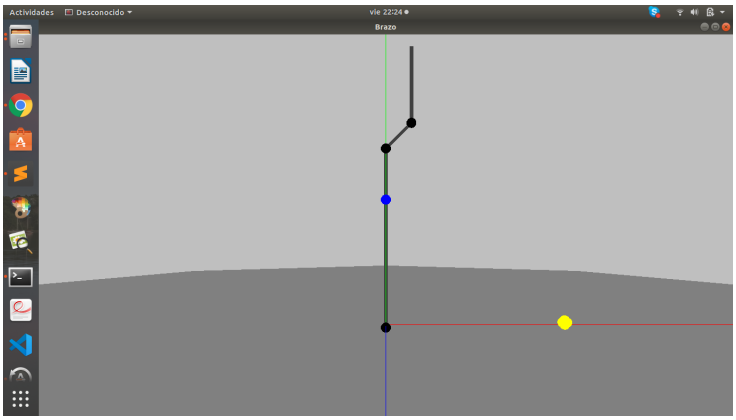

Figure 3: Imagen paso 5.

6. Luego de esto se podrá proceder con la rotación con las teclas "+" o "-"(ver figura 4).

7. Cuando el extremo del brazo este tocando la bola(o lo suficientemente cerca para agarrarla), la bola cambiará de color a blanco(ver figura 5).

8. Ahora presionando la tecla "p" se agarra la bola,con lo cual vuelve su color original(ver figura 6).

9. Después de esto se puede presionar de nuevo la tecla "p" para soltar la bola(ver figura 7).

10. Con esto es posible realizar de nuevo los pasos descritos anteriormente.

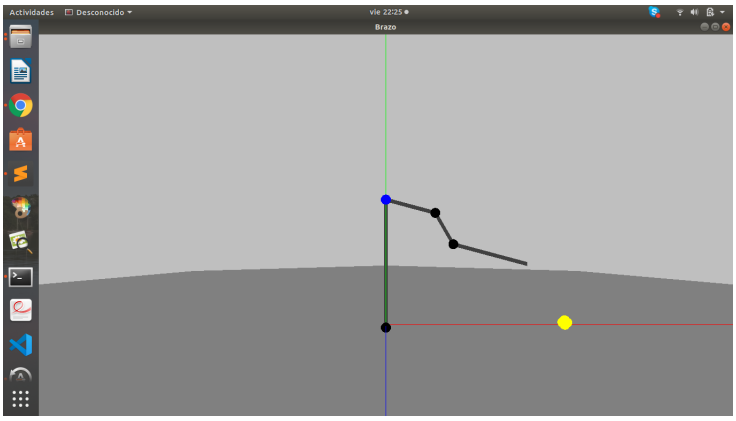

Figure 4: Imagen paso 6.

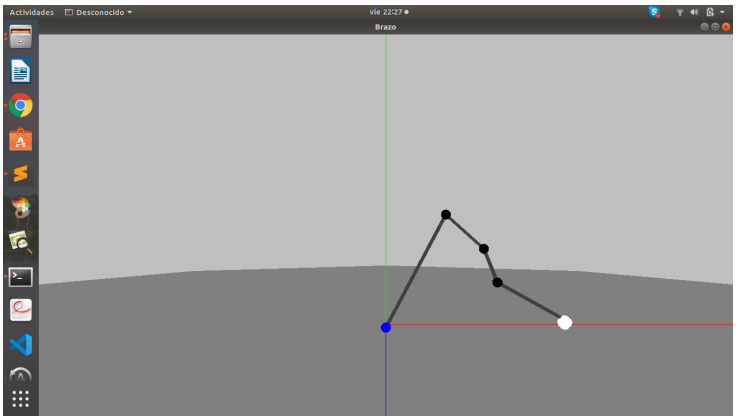

Figure 5: Imagen paso 7.

4 Conclusiones de desarrollo

- La utilización de árboles de jerarquía para la representación de un modelo puede facilitar algunos aspectos como también complicar otros, cuando a diseño de la solución nos referimos.
- El conocimiento básico de conceptos matemáticos (físicos) (e.g. suma de vectores, rotación respecto a los ejes, distancia entre puntos, etc...) puede llegar a ser imprescindible o muy importante para el desarrollo de modelos del mundo real.

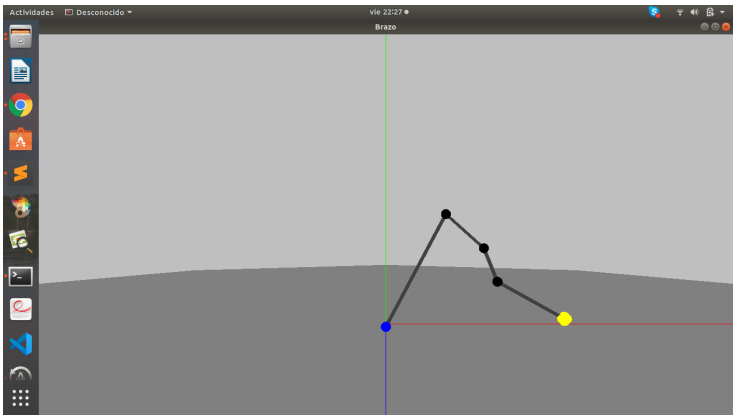

Figure 6: Imagen paso 8.

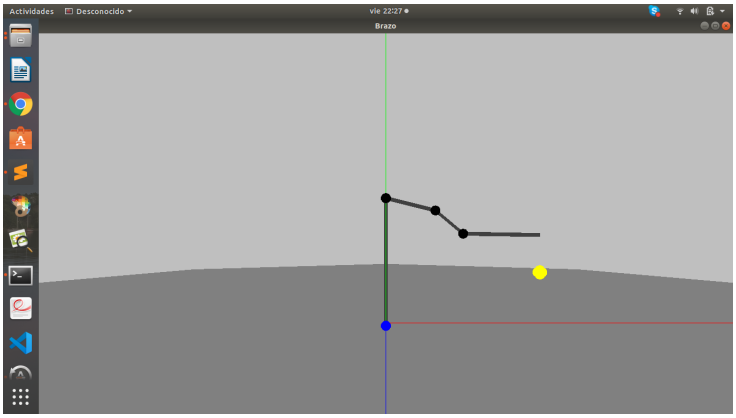

Figure 7: Imagen paso 9.