INFORMÁTICA GRÁFICA Y VISUALIZACIÓN
CARLOS REQUENA, YESSIN MOHAMED
CRD00016 Y YMM00014

PROYECTO FINAL: ビリビリ(BIRI-BIRI) EL ROMPELATAS

—

Realizado en OpenGL

INTRODUCCIÓN

Juego que consiste en obtener la mejor puntuación posible derribando latas que aparecen por el escenario. ¡Pero cuidado! Las latas desaparecen y puedes perder la oportunidad de aumentar tu puntuación.

ÍNDICE

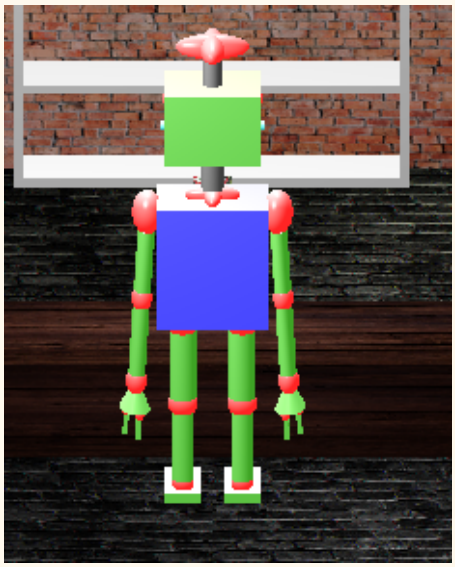
1.Funcionamiento general del juego

El juego consiste en derribar latas en el escenario que aparecen de forma aleatoria, la manera de poder interactuar con el juego es a través del click del ratón, con el que seleccionaremos la lata a derribar y esta será derribada por el robot ビリビリ. Durante la traslación de la pelota hacia la lata podemos cambiar nuestro objetivo.

Existen cuatro tipos de puntuaciones que se pueden obtener. Las primeras tres son derribando latas normales dado que dependiendo del color tienen una puntuación mayor o menor al ser destruidas. En este caso, el color rojo es el máximo con 150, el color azul proporciona 100 y el color verde proporciona 50. Por otro lado, tenemos una “Pelota especial”, la cuál proporciona 1000 puntos tras ser destruida, sin embargo, debemos golpearla 10 veces para poder derribarla, siendo que se moverá a otra posición por cada golpe, pudiendo bloquear o molestar al usuario. La “Pelota especial” supone una mecánica aparte dado que con el tiempo límite del juego que son 100 segundos, te interesa obtener las latas de mayor calidad y aún así hacerte con la “Pelota especial”.

El juego está pensado para alternar entre dos cámaras, de modo que puedas ver mejor las latas de la derecha y la izquierda dependiendo de tu ángulo.

ELEMENTOS SELECCIONABLES



Se puede seleccionar y rotar todos los elementos pintados de verde en esta imagen, todos tienen un solo movimiento menos el brazo izquierdo superior y la cabeza los cuales tienen movimientos en el eje x e y.

Dichos movimientos están limitados en base al desplazamiento del cursor por lo que es recomendable que al cambiar de sentido se vuelva a seleccionar, por ejemplo si se desplaza el brazo hacia arriba y después se quiere desplazar hacia abajo.

Se rota manteniendo el ratón pulsado con el clic izquierdo sobre el elemento y desplazando el ratón en el eje x o y.

Elementos seleccionables: **cabeza**(da igual si se pulsa en la base, en un ojo o en cualquier lugar de la cabeza),**brazos superiores, brazos inferiores, manos, dedos**(los 6 dedos), **piernas superiores, piernas inferiores, pies.**

Ya que no se ha implementado el escalado y la traslación mediante selección en el robot se ha creado un objeto exclusivo para ello. Se trata de una lata que aparece en medio del escenario y que se puede escalar moviendo el ratón en el ‘eje y’ o trasladarla moviendo el ratón en el ‘eje x’. Hay que mantener el clic izquierdo sobre la lata y después el desplazamiento del ratón.

ATAJOS DE TECLADO

Las teclas minúsculas representan un aumento o valor positivo siendo negativo la mayúscula

-Tecla q/Q (Aumentar ángulo del foco o disminuirlo)
-Tecla e/E (Aumentar exponente/reducir exponente)
-Tecla r/R (Aumentar/disminuir G y B de reflexión difusa)
-Tecla t/T (Aumentar/disminuir G y B de reflexión especular)
-Tecla y/Y (Apagar/encender foco)
-Tecla d (Desplazar foco derecha)
-Tecla a (Desplazar foco izquierda)
-Tecla w (Desplazar foco arriba)
-Tecla s (Desplazar foco abajo)
-Tecla i (Activar animación del robot)
-Tecla o (Resetear animación)
-Tecla +/- (Zoom in/out)
-Tecla c/C (Incrementar/decrementar la distancia del plano cercano)
-Tecla v/V (Mover cámara en z positivo/negativo)
-Tecla x (Cámara hombro derecho)
-Tecla z (Cámara hombro izquierdo)
-Tecla b/B (Movimiento panorámica derecha/izquierda)
-Tecla n/N (Desplazar cámara en x/-x)
-Tecla m/M (Desplazar cámara en y/-y)
-Tecla h/H (Cabeceo hacia arriba/abajo)
-Tecla 1 (Cambia el tipo de proyección de paralela a perspectiva)
-Tecla 2 (Cambia el tipo de proyección de perspectiva a paralela)
-Tecla 3 (Vista cenital)
-Tecla 4 (Vista de perfil)
-Tecla </> (Moverse a la derecha/izquierda en el menú)
-Tecla ENTER (Tecla enter para seleccionar opción de menú)
-Tecla RETROCESO (Volver al menú principal)
-Tecla f (Intercambiar entre sombreado de Gourand y plano)
-Tecla g/G (Rotar el robot en eje positivo/negativo)

Teclas para controlar la velocidad del juego para ajustar la experiencia de juego entre distintos equipos.
-Tecla 8/9 (Aumentar/disminuir velocidad de la pelota)
-Tecla j/J (Aumentar/disminuir velocidad de animación del robot)

2.Métodos y explicación de las clases y sus componentes

Clase igvEscena3D

Atributos:

-std::vector<GLfloat> color_grisOscuro, color_rojo, color_verdeAzul, color_azul, color_marron, color_naranja (estos atributos representan una paleta de colores definida para su uso a lo largo de práctica)

-Modelos* modelos (este atributo representa la clase modelos de la que obtenemos todas las primitivas de la práctica)

-bool modo_act (indica si se está en el modo selección)

-bool animacionPelota (booleano que representa si el robot está en animación y pinta la pelota en su mano en consecuencia)

-bool lanzarPelota (booleano que representa que la pelota está en trayectoria hacia el objeto destino)

-std::vector<hitbox*> hitboxes (vector que almacena todas las hitboxes de los objetos que se encuentran en escena, cada objeto hitbox está asociado a un objeto por sus coordenadas)

-hitbox hitboxDestino (objeto hitbox que define el objeto al que se dispara la pelota)

-juego juego (objeto de la clase juego que representa la partida y sus métodos correspondientes)

-float a (atributo que forma parte de la fórmula de interpolación utilizada en la trayectoria de la pelota)

-float movementSpeed (atributo que forma parte de la fórmula de interpolación utilizada en la trayectoria de la pelota y determina la velocidad de la trayectoria)

-float deltaTime (atributo que forma parte de la fórmula de interpolación y representa los frames en la misma)

-std::vector<hitbox*> hitboxesPendientes (vector que almacena las hitboxes pendientes de ser añadidas)

-std::vector<int> hitboxes_a_borrar (vector que almacena el índice de las hitboxes respecto al vector de hitboxes para ser eliminadas)

-int segundos1, segundos2, segundos3 (atributos que sirven como medida de control del tiempo de reaparición de las latas de modo que cada determinados segundos se llama una nueva lata)

-bool finPartida (booleano que determina el fin de la partida para terminar de ejecutar y procesar la información de latas, hitboxes y demás)

-bool iniciarPartida (booleano que activa el juego si está activado)

GESTIÓN DE LUCES

-float ang_foco (atributo que sirve para cambiar el ángulo de amplitud del foco. Se controla con el teclado)

-exp_foco (atributo que sirve para cambiar el exponente de atenuación angular. Se controla con el teclado)

-GB_dif (atributo que sirve para modificar los valores G y B de los posibles valores RGB que puede tener la reflexión difusa del foco. Se controla con el teclado)

-GB_esp (atributo que sirve para modificar los valores G y B de los posibles valores RGB que puede tener la reflexión especular del foco. Se controla con el teclado)

-float X, Y (atributos que sirve para mover el foco en la posición X e Y. Se controla con el teclado)

-bool foco_activo (atributo para activar/desactivar el foco. Se controla con el teclado)

-std::vector<GLfloat> colores (array que almacena los diferentes colores de los que se pintaran los objetos en la selección)

-int pos_r, pos_a, pos_v, pos_g (atributos que sirven para guardar la posición en la que empieza cada color en el vector de colores, por ejemplo pos_g es la posición donde empieza el color gris)

GESTIÓN DE GRADOS DE LIBERTAD DE ROBOT

-Se utilizan varios atributos para la gestión de los grados de libertad, estos están declarados en el código con un nombre intuitivo y con un comentario que describe qué parte del grafo representan

Métodos:

-std::vector<hitbox*>& **getHitboxes()** (método que se encarga de devolver el vector de hitboxes)

-int **buscarHitbox(hitbox &h)** (devuelve el índice de una hitbox respecto al vector de hitboxes)

-void **calculoTrayectoriaPelota(hitbox h1, hitbox h2)** (este método calcula la trayectoria de la pelota mediante dos posiciones, la inicial y la final, de modo que mediante interpolación se llega al destino)

-void **activarLanzamientoPelota()** (activa el lanzamiento de la pelota)

-void **desactivarLanzamientoPelota()** (desactiva el lanzamiento de la pelota)

-bool **getLanzandoPelota()** (Se comprueba que la pelota esté en movimiento)

-bool **detectarColisiones(const hitbox &h1, hitbox &h2)** (se comprueba que haya colisión entre dos hitboxes, esto se hace entre el eje X y Z)

-void **actualizarCoordenadaFinal(const igvPunto3D &inicial)** (se actualiza la coordenada final, es decir, la coordenada objetivo)

-void **sustituirLata(const int &i)** (se invoca a método de la clase juego para poder sustituir la lata y añade hitboxes a espera de ser borradas)

-void **determinarColorLata(const int &i, std::vector<float> &colorin)** (se determina el color de la lata en función de su puntuación)

-void **crearLata()** (se llama al método de la clase juego para inicializar una lata)

-void **asignarLatasIniciales()** (se llama al método de la clase juego para inicializar 10 latas iniciales)

-void **procesarColisiones(const hitbox &h1, hitbox h2)** (método que procesa una colisión tras impactar la pelota con un objeto de modo que haga lo necesario con la hitbox destino)

-void **interpolacionTrayectoria(hitbox &h1, hitbox &h2)** (método que se encarga del cálculo matemático de la trayectoria de la pelota)

-void **gestionarLatasEventos()** (método que gestiona la creación y sustitución de las latas)

-void **gestionarPelotaEspecialEventos()** (controla la creación y asignación de pelota especial)

-void **spawnPelotas()** (controla la aparición periódica de latas en la escena)

-void **gestionarLatasEventosVB()** (método que gestiona la creación y sustitución de las latas en el modo selección)

-void **gestionarPelotaEspecialEventosVB()** (controla la creación y asignación de pelota especial en el modo selección)

-void **pintar_robot()** (método para pintar el grafo de escena en modo visualización);

-void **pintar_robotVB()** (método para pintar el grafo de escena en modo selección);

-void **cambia_color(std::vector<GLfloat> color, std::vector<GLfloat>& destino, int& pos, int tam)** (método que sirve para insertar en el vector 'destino' el color que hay en el vector 'color', los atributos pos y tam sirven para gestionar a partir de donde se inserta y cuántos valores)

-void **reinicio_colores()** (método que sirve para reinicializar los atributos color_grisOscuro, color_rojo, color_verdeAzul, color_azul a su valor original)

-void **drawText(float x, float y, float z, const char* text)** (método para generar texto)

-void **gestionarTextos()** (método que muestra texto en pantalla de la puntuación)

- Get y Set genéricos de muchos atributos

Clase Juego

Clase que representa aquello más cercano a la lógica pura del juego, es decir, alejarse más de aquello relacionado con los objetos en sí y elementos 3D. Se gestiona también la puntuación de partidas y demás.

Atributos:

-int puntuacion (atributo que almacena la puntuación de la partida)

-int punt_maxima (puntuación máxima de las partidas de una sesión)

-clock_t t, aux (atributos de la clase ctime que gestionan la duración de la partida)

-std::vector<igvPunto3D> vectoresPos (vector que guarda todas las posiciones posibles de las latas en la escena)

-std::vector<igvPunto3D> vectoresPosPelotaEspecial (vector que guarda todas las posiciones posibles de la pelota especial en la escena)

-bool posicionesVectorOcupadas[49] (vector de booleanos que controla qué posiciones han sido ocupadas dado que se corresponde con "vectoresPos", siendo que la posición 0 se corresponde con el contenido de la posición 0 de "vectoresPos")

-int numMaxLatas (atributo que almacena la cantidad máxima de latas permitida en escena)

Métodos:

-void **setPosicionesOcupadas(int i, bool ocupada)** (mediante una i que representa el índice, se accede al vector de posicionesVectorOcupadas para modificar el estado de este, siendo que puede ponerlo en false o true, es decir, está ocupada o no)

-bool **estaLaPosicionOcupada(int i)** (se comprueba si una posición está ocupada en la escena mirando el estado del vector posicionesVectorOcupadas)

-void **inicializarLata(std::vector<hitbox*> &hitboxes, const int &i)** (se escoge aleatoriamente una posición del vector de posiciones, se asigna un valor de puntuación aleatorio y se asigna un tiempo de refresco de lata aleatorio. Todo estos valores asignados sirven para crear un objeto **hitbox** que es almacenado en el vector de punteros de hitboxes que proviene de **igvEscena3D** que se manda al método)

-void **posicionesObjetos(std::vector<igvPunto3D>& vector)** (método que asigna todas las posiciones posibles a un vector)

-void **posicionPelotaEspecial()** (método que asigna todas las posiciones posibles de la **Pelota Especial** a un vector que las almacena)

-igvPunto3D **nuevaPosicionPelotaEspecial()** (asigna una nueva posición a la Pelota Especial elegida al azar del vector **vectoresPosPelotaEspecial**)

Clase hitbox

Esta clase busca una asociación entre los objetos generados en escena a través de la posición de modo que se pueda calcular colisión en caso de haberla y almacenar color del modo selección entre otras gestiones.

Atributos:

-int valorObjeto (atributo que determina la puntuación de un objeto)

-std::vector<GLfloat> colorObjeto (vector que almacena el color gris del objeto para la selección)

-int tiempoRefresco (tiempo que tarda la lata en ser sustituida)

-clock_t t, aux (reloj que lleva la cuenta del tiempo de vida de la lata)

-igvPunto3D posicion, tamano (vectores que determinan la posición del objeto y su tamaño que sirve para calcular colisiones)

Métodos:

-void **actualizarCoordenadas(float x, float y, float z)** (actualiza las coordenadas de la hitbox)

-bool **pasadoDeTiempo()** (devuelve un valor booleano en función de si el tiempo que lleva de vida del objeto supera al atributo tiempoRefresco)

-std::vector<float> **getPosicionFloat()** (devuelve la posición de la hitbox en forma de vector de igvPunto3D)

-void **setPosicion(const std::vector<float> &v), (igvPunto3D p)** (diferentes métodos para asignar posición a la hitbox. Uno con un vector de flotantes y otro con un igvPunto3D)

Clase igvCamara

Métodos:

-void **zoomOut()** (método que incrementa el zoom de la cámara)

-void **zoom()** (método que incrementa el zoom de la cámara)

-void **maszP0(float a)** (aumenta la coordenada z de la posición de la cámara permitiendo desplazamiento de la misma)

-void **menoszP0(float a)** (disminuye la coordenada z de la posición de la cámara permitiendo desplazamiento de la misma)

-void **masyP0(float a)** (aumenta la coordenada y de la posición de la cámara permitiendo desplazamiento de la misma)

-void **menosyP0(float a)** (disminuye la coordenada y de la posición de la cámara permitiendo desplazamiento de la misma)

-void **masxP0(float a)** (aumenta la coordenada x de la posición de la cámara permitiendo desplazamiento de la misma)

-void **menosxP0(float a)** (disminuye la coordenada x de la posición de la cámara permitiendo desplazamiento de la misma)

-void **panoramica(float a)** (método que simula el movimiento panorámico de una cámara desplazando el punto de referencia de la cámara. Simula el movimiento de la cabeza hacia los lados de una persona)

-void **cabeceo(float a)** (método que simula el movimiento de cabeceo de una cámara moviendo el punto de referencia de la cámara en el eje y. Simula el movimiento de la cabeza hacia arriba y abajo, como si mirases un edificio alto y mirases hacia arriba)

Clase Cylinder

Clase para instanciar un cilindro formado por malla de triángulos, que funciona así :

Se indica el tamaño y demás a través del constructor

-**Cylinder(float baseRadius, float topRadius, float height, int sectorCount, int stackCount, bool smooth)**;

Se visualiza el cilindro con el método draw

-void **draw( )** const

Esta clase está sacada de internet: http://www.songho.ca/opengl/gl_cylinder.html

Clase Sphere

Clase para instanciar una esfera formado por malla de triángulos, que funciona así :

Se indica el tamaño y demás a través del constructor

-**Sphere(float radius, int sectorCount, int stackCount, bool smooth)**

Se visualiza la esfera con el método draw

-void **draw( )** const

Esta clase está sacada de internet: http://www.songho.ca/opengl/gl_sphere.html

Clase Cubo

Métodos:

-static void **visualizar()** (visualizar cubo formado por quads con 20 divisiones)

-static void **visualizarLatas()** (visualizar cubo formado por quads con 5 divisiones)

-static void **cara_abajo()** (visualizar solo quad de abajo)

Clase Colores

Clase que contiene el color de las diferentes partes del grafo de escena, por ejemplo el color de la cabeza del robot se declara en esta clase y se puede modificar o acceder a él a través de get y set .

Clase Modelos

Clase que contiene los diferentes objetos que se instancian en la escena

Atributos:

-Cylinder* cil

-Sphere* sph

-Cylinder* cono (cilindro en forma de cono)

-Colores* colores_robot

-igvTextura* text (textura de madera)

-igvTextura* text2 (textura de ladrillo)

-igvTextura* text3 (textura de suelo)

Métodos:

Primitivas básicas

-void **cilindro(GLfloat color_cilindro[])**

-void **esfera(GLfloat color_esfera[])**

-void **cubo(GLfloat color_cubo[])**

-void cono3D(GLfloat color_cono[])

Primitivas del robot (grafo de escena)

-void **cabeza(std::vector<GLfloat> color_rojo2, std::vector<GLfloat> color_Verde_Azul, std::vector<GLfloat> color_gris)** (método para visualizar la cabeza, los colores que se pasan por cabecera son para indicar el color de ojos, boca y demás, la base de la cabeza se pinta del color que viene determinado por el objeto colores)

Todo método que incluya el atributo 'lado' indica si es la parte del cuerpo izquierda o derecha, dependiendo del lado se pinta de un color u otro

-void **torso()** (método para visualizar el torso)

-void **piernas(int lado)** (método para visualizar la pierna superior(de ingles a rodilla))

-void **piernas_inf(int lado)** (método para visualizar de rodillas a pie(pie no incluido)

-void **pies(int lado)** (método para visualizar el pie)

-void **brazo()** (método para visualizar el hombro)

-void **brazo_superior(int lado)** (método para visualizar de hombro(no incluido) hasta el codo)

-void **brazo_inferior(int lado)** (método para visualizar desde codo hasta muñeca)

-void **mano(int lado)** (método para visualizar la muñeca y la palma de la mano)

-void **dedo(int num_dedo)** (método para visualizar el dedo, el atributo num_dedo indica qué dedo es para pintar cada uno de su color ;

-void **articulacionDedo()** (se visualiza la articulación del dedo)

-void **cuello()** (se visualiza el cuello)

Primitivas Puesto

-void **Estanteria()**

-void **Mostrador()** (Se le aplica la textura 'txt' de madera)

-void **Suelo()** (Se le aplica la textura 'txt3' de suelo)

-void **Pared()** (Se le aplica la textura 'text' de ladrillos)

-void latas(GLfloat color_lata[],bool textura = true) (Se le aplica la textura 'text' de coca cola si el bool es true)

-Colores* **Get_coloresRobot()** (método que devuelve el objeto colores, para que desde fuera se puedan cambiar los colores de las primitivas del robot)

Clase igvColor

Clase reutilizada de la práctica 4, contiene lo mismo que en la práctica.

Clase igvMaterial

Clase reutilizada de la práctica 4, contiene lo mismo que en la práctica

Clase igvTextura

Esta clase se basa en la de la práctica 4, pero a diferencia de esa ahora en el constructor sólo se carga la imagen

Atributos

-SDL_Surface* imagen (atributo que almacena la imagen)

Métodos

-void **pre_aplicar()** (Establece los parámetros de textura en opengl)

-void **aplicar(void)** (Establece la textura como la activa)

-void **setIdTextura** (unsigned int id) (modifica el id)

Clase igvFuenteLuz

Clase reutilizada de la práctica 4, con la única diferencia de que ahora a través del constructor se indica con un bool si la luz es direccional y no.

Clase igvInterfaz

Atributos:

-bool animar (indica si está activa la animación)

-int fin_primera_fase (atributo utilizado para la gestión de la animación)

-modoInterfaz modo (indica si visualizamos en modo selección o no. Igual que en prácticas)

-int cursorX, cursorY (pixel de la pantalla sobre el que está situado el ratón, por pulsar o arrastrar. Igual que en prácticas)

-int objeto_seleccionado (código del objeto seleccionado, -1 si no hay ninguno. Igual que en prácticas)

-bool boton_retenido (indica si el botón está pulsado (true) o se ha soltado (false). Igual que en prácticas)

-int velocidadAnimacion, velocidadAnimacionNegativa (atributos para parametrizar la velocidad de la animacion por temas de diferentes velocidades en distintos equipos)

Métodos

Se mantienen los mismos métodos que en la prácticas y se introducen estos:

-void **resetear_colores()** (método para resetear los colores originales del robot)

-void **pintar_seleccion()** (método para pintar el elemento seleccionado)

-void **resetear()** (Se reinicia a la posición original a aquellos elementos rotados del grafo de escena)

-void **cambiarEscenaEnMenu()** (en función de la opción seleccionada en el menú principal se activa una escena u otra, es básicamente el método que representa la interacción con la opción seleccionada)

-void **menuHandle(int value)** (método que genera el menú desplegable con el click derecho del ratón para el modo de juego principal, el menú solo funciona en el modo de "jugar")

3.Funcionalidades implementadas

SELECCIÓN

Se aplica una selección basada en buffer de color, para ello se sigue este procedimiento:

1. TENER UN VECTOR DE COLORES PARA DIFERENCIAR LAS DIFERENTES PARTES MÓVILES

Para rellenar el vector 'colores' en el constructor de la clase igvEscena3D lo que se hace es insertar los diferentes colores por orden, de modo que por ejemplo los primeros x colores representan la cabeza del robot. Esto se hace a través de un bucle el cual va generando 1 color nuevo(3 GLfloat) por cada vuelta.

2. ASIGNAR A LAS DIFERENTES PARTES MÓVILES LOS COLORES GENERADOS

Este proceso se hace en el método 'igvEscena3D::pintar_robotVB( )' para las partes seleccionables del grafo de escena.

Para ello se utiliza el método 'igvEscena3D::cambia_color( )' para asignarle a un atributo auxiliar el siguiente color no utilizado en el vector 'colores', tras esto se sustituye el color con el que se pinta la primitiva por el color que se ha almacenado en el atributo auxiliar. De modo que cada primitiva se pinte con un color diferente entre todos los que hay en el vector 'colores'.

Ese es el funcionamiento general, pero hay que tener en cuenta lo siguiente:

La cabeza está compuesta por varias partes (ojos, cuello, cabeza…) por lo que a cada parte de la cabeza se le asigna un color del vector 'colores', los demás objetos seleccionables solo están formados por un color.

3. SELECCIONAR EL OBJETO, DETECTARLO Y ACTUAR EN BASE A ELLO

En el método ‘igvInterfaz::set_glutMouseFunc’ se cambia de modo visualización a modo selección, se almacenan las posiciones en las que se ha pulsado(en los atributos CursorX y CursorY) y se informa a la clase escena de que estamos en el modo selección a través del atributo ‘escena->modo_act’ cuando se pulsa el clic izquierdo.

En el método ‘igvInterfaz::set_glutDisplayFunc()’ si estamos en el modo visualización se habilitan las luces y texturas y se visualiza la escena con los colores originales. Si estamos en el modo selección se realiza lo siguiente en orden:

- Se resetean las primitivas a su color original.
- Se desactivan las luces y texturas.
- Se visualiza la escena con los colores del vector ‘escena->colores’.
- Se lee el color del píxel correspondiente a la posición ‘CursorX’ y ‘CursorY’ y se compara con los colores almacenados en el vector ‘escena->colores’, de modo que nos quedamos con el índice del vector cuando el color coincide.
- Se pinta de color verde la primitiva a la que le corresponde el índice almacenado en el punto anterior.
- Se cambia el modo de selección a visualización, se le indica a escena con el atributo ‘escena->modo_act’, se refresca la ventana con el método ‘PostRedisplay()’ y se habilita la luz.

Tras este proceso tenemos almacenado el índice del color seleccionado, por consiguiente podemos detectar que primitiva se ha seleccionado, dependiendo de qué primitivas se ha seleccionado hacemos lo siguiente:

- Detectar la lata que se ha seleccionado en el juego y por tanto dirigir la pelota hacia ella(a partir del índice 66). Esto se hace en ‘igvInterfaz::Gestion_seleccion_lanzamiento’
- Detectar el desplazamiento del ratón cuando el clic está pulsado y realizar una rotación, escalado o traslación al objeto correspondiente. Esto se hace en ‘igvInterfaz::set_glutMotionFunc’

VISUALIZACIÓN

La visualización está compuesta por dos métodos, el primero es ‘igvEscena3D::Visualizar()’ el cual establece las luces y llama al segundo método ‘igvEscena3D::VisualizarVB’ en el cual se visualizan las primitivas como tal ya sea del color original en el modo visualización o de diferentes colores en el modo selección.

Hay que tener en cuenta que se diferencia del grafo de escena original (‘igvEscena3D::pintar_robot’) del que tienes colores para la selección (‘igvEscena3D::Pintar_robotVB()’)

LUCES

Existen 3 luces una direccional, una puntual y un foco, estas se instancian en la clase ‘igvEscena3D’ indicando sus parámetros en el constructor y se aplican con el método ‘igvFuenteLuz::aplicar()’.

TEXTURAS

En la clase ‘Modelos’ existen varios objetos de tipo ‘igvTexturas’ los cuales se han construido con una url y se les ha asignado una id de textura fija. Para aplicar esta imagen en alguna primitiva se llama desde el objeto de la clase al método ‘igvtextura::pre_aplicar( )’ y después a ‘igvtextura::aplicar( )’.

Hay que tener en cuenta que no cargamos la imagen de nuevo por cada vez que queremos aplicarla, sino que la cargamos en el constructor y la guardamos en una variable y que no se utiliza el método GenTextures para asignar id de textura, sino que se hace manualmente a través del método ‘setidtextura()’.

ANIMACIÓN

La animación se genera modificando los valores de las rotaciones de las primitivas en cada frame, esta puede simular el lanzamiento de la pelota o puede representar un lanzamiento

seguido de una patada dependiendo de si estamos en el modo juego (solo lanzamiento) o modo creativo/robot (ambos).

LÓGICA DEL JUEGO

Para entender la lógica detrás del juego, se deben tocar distintas fases o apartados por los que pasa el juego para mantener su estructura.

1. SELECCIÓN DE LOS OBJETOS

Los objetos, o mejor dicho latas, se seleccionan con el click izquierdo, lo cuál se hace en 'igvInterfaz::Gestion_seleccion_lanzamiento', de modo que aprovechamos la **SELECCIÓN** para escoger el objetivo dado que cada lata, tendrá un color asignado en su **HITBOX(clase que gestiona las colisiones y vincula objetos por su posición en la escena)**. En ese mismo método, se asigna la hitbox relacionada con el color seleccionado como 'igvEscena3D::hitboxDestino' de modo que esa lata está guardada para su posterior uso y cálculo.

2. LANZAMIENTO DE LA PELOTA

-Respecto al lanzamiento de la pelota, el robot(personaje jugable) realizará una animación trás seleccionar una lata donde se prepara y lanza una pelota, esto se hace en 'igvEscena3D::pintar_robot()' activando una pelota en el grafo de escena mientras se realiza la animación. 'igvInterfaz::set_glutIdleFunc()' es donde se controla la animación de modo que una vez se termine la animación, se comprueba que estemos en el modo de jugar a las latas mediante 'igvEscena3D::estaEnJuego' y se activa el booleano de 'igvEscena3D::lanzarPelota' para indicar que la pelota debe comenzar a moverse en una trayectoria que se calcula a continuación, se desactiva la animación con el 'igvInterfaz::animar' y se resetea la animación.

-Una vez hecha la animación, toca comenzar con el cálculo de la trayectoria de la pelota, el atributo de 'igvEscena3D::lanzarPelota' que se activó anteriormente tras la animación hace que en el 'igvEscena3D::visualizarVB()' se proceda con la llamada de **'igvEscena3D::calculoTrayectoriaPelota(hitbox h1, hitbox h2)'.**

-Este método **'igvEscena3D::calculoTrayectoriaPelota(hitbox h1, hitbox h2)'** recibe dos objetos **HITBOX,** el primero representa la posición de la pelota y el segundo objeto representa la igvEscena3D::hitboxDestino(hitbox de la lata seleccionada), a continuación, se llama a 'igvEscena3D::interpolacionTrayectoria(hitbox& h1, hitbox& h2)' en el que a rasgos grandes se

realiza una interpolación con una fórmula específica, cabe destacar que sobre estos cálculos, ‘igvEscena3D::posicionPelota’ y la hitbox de la pelota va actualizando su posición cada frame de modo que eventualmente llegará a la coordenada objetivo(hitbox destino).

Siguiendo con el método **‘igvEscena3D::calculoTrayectoriaPelota(hitbox h1, hitbox h2)’** toca comprobar colisiones y lo relacionado, de modo que se llama a **‘igvEscena3D::procesarColisiones(h1, h2)’** en cuyo método en primer lugar, se comparan los dos objetos **HITBOX**, para ello, se llama a la función ‘igvEscena3D::detectarColisiones(const hitbox& h1, hitbox& h2)’, en esta se comprueba colisión comparando el vector posición y tamaño(en el eje x, z) esto se hace para simplificar el proceso, dado que no es necesario llevarlo al plano 3D para poder detectar estas colisiones básicas. Siguiendo con ‘igvEscena3D::procesarColisiones(h1, h2)’, si se detectaron colisiones(en el método igvEscena3D::detectarColisiones(const hitbox& h1, hitbox& h2)), se comprueba si la hitbox con la que se colisiona se corresponde a la de la **“Pelota Especial”(pelota que actúa aparte de las demás hitboxes almacenadas en la clase)** y se suma la puntuación adecuada y se procesa. En caso de no ser la “Pelota Especial” se procede a buscar la hitbox correspondiente a la lata colisionada, para ello, se mira la hitbox de la lata en busca del índice que le corresponde en **el vector de hitboxes** y se suma su puntuación y se elimina del vector, además se desactiva ‘igvEscena3D::lanzarPelota’ y se activa ‘igvEscena3D::animacionPelota’ para poder lanzar de nuevo correctamente.

3. GENERACIÓN DE LATAS, REFRESCO Y GESTIÓN

En cuanto a las latas que componen la escena, estas comienzan siendo 10 en un principio, sin embargo, conforme pasa el tiempo, nuevas van apareciendo, y las antiguas se renuevan.

-Las primeras 10 latas generadas se generan mediante el método ‘igvEscena3D::asignarLatasIniciales()’ que llama al método ‘juego::inicializarLata()’ al cuál se le pasa como argumento el vector de hitboxes y 10(número de latas a generar), dicho método asigna un tiempo de refresco, un valor de puntuación y una posición aleatoria(son aleatorios dentro de unos rangos específicos dependiendo del valor).

-En el transcurso normal del juego, las latas que estén en escena deben ser sustituidas por otra pasado un tiempo determinado para forzar al jugador a pensar rápido, para cumplir con este propósito, tenemos el método **‘igvEscena3D::gestionarLatasEventos()’**. Este método gestiona las latas que se renuevan, en primer lugar, el método recorre el **vector de hitboxes** para comprobar si una lata se ha pasado de tiempo, esto es posible dado que cada lata al ser inicializada tiene un reloj interno que mide el tiempo que lleva en escena. Al encontrar una lata

que haya superado su tiempo límite se llama al método ‘igvEscena3D::sustituirLata(i)’ pasándole el índice de la hitbox que se ha pasado de tiempo, es entonces que añade dicho índice a un vector de borrado de hitboxes ‘igvEscena3D::hitboxes_a_borrar’ además de generar una nueva lata y almacenándola en un vector de hitboxes que servirá para añadirla al vector de hitboxes a posteriori. En caso de que la lata no se haya pasado de tiempo, se pinta en la escena con su color correspondiente al valor de esta.

Una vez pintadas las latas a pintar, en caso de que una o más latas hayan sido sustituidas, con el vector de hitboxes a borrar, se libera la posición que tenían ocupadas dichas latas para poder ser ocupadas por nuevas latas que se generen más adelante, las posiciones que se ocupan se guardan en ‘juego::posicionesVectorOcupadas[49]’ y se añade un booleano true or false en función de estar ocupadas, dicha asignación se hace cada vez que se añade una lata o se borra. En cuanto al vector en el que se añaden los punteros de las latas nuevas derivadas de una sustitución, se procede a añadirlas al vector de las hitboxes en escena ‘igvEscena3D::hitboxes’.

Cabe destacar que también se llama a gestionar colisiones en este método.

-Existe también **‘igvEscena3D::gestionarPelotaEspecialEventos()’**, método que en esencia, hace que la **“Pelota Especial”** se pinte.

-Existen métodos para gestionar el pintado de latas para la funcionalidad de la **SELECCIÓN.** Estos son ‘**igvEscena3D::gestionarLatasEventosVB()**’ y ‘**igvEscena3D::gestionarPelotaEspecialEventosVB()**’. Son básicamente pintar pero con los colores del vector ‘colores’ y en función a lo ya añadido en el método anterior.

-Por último, tenemos el método ‘igvEscena3D::spawnPelotas()’ cuyo objetivo es generar periódicamente latas nuevas para mantener un ritmo estable en el juego.

MENÚ DEL JUEGO

El menú del juego se representa con tres cubos escalados con texto encima, de modo que conforme te mueves entre las opciones con las flechas, un foco apunta a la opción seleccionada haciéndola la única visible.

El menú se implementa en torno a ciertas variables de control, las cuales controlan el estado del juego en general, siendo posibles el modo menú, modo juego y modo robot/creativo ‘igvEscena3D::menu, juegoL, robot’. Por otro lado, si la escena en ejecución es el menú

principal, tenemos una variable ‘igvEscena3D::estadoMenu’ que controla la opción seleccionada.

Dependiendo del estado del menú, el foco apuntará a una posición u otra, de modo que se ilumine siempre la opción seleccionada.

En cuanto al cambio de escenas, tenemos un método que ‘igvInterfaz::cambiarEscenaEnMenu()’ que cambia a la escena de la opción seleccionada, en este método se prepara el salto de escena deshabilitando luces entre escenas además de aplicar la cámara necesaria para dicha escena.

Para el funcionamiento como tal, dependiendo de las tres variables de control booleanas que expresan la escena actual se ejecuta una visualización u otra.

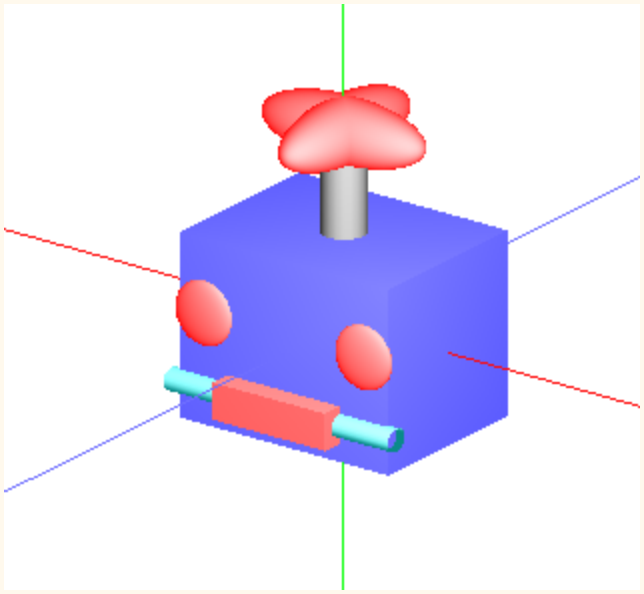
4.GRAFO DE ESCENA

Modelo del robot

El robot consta de muchos movimientos en todas las articulaciones principales, estas tienen ciertos movimientos que son imposibles para un humano, pero el robot puede llevarlas a cabo dadas sus “bolas rojas” que representan las articulaciones.
Tenemos un modelo bien modularizado con bastantes movimientos disponibles, siendo 21 movimientos posibles dentro de este.
Ciertos grados de libertad presentan limitaciones de modo que no realice posiciones imposibles para el robot como tal, véase, meter el brazo dentro del cuerpo al rotarlo.

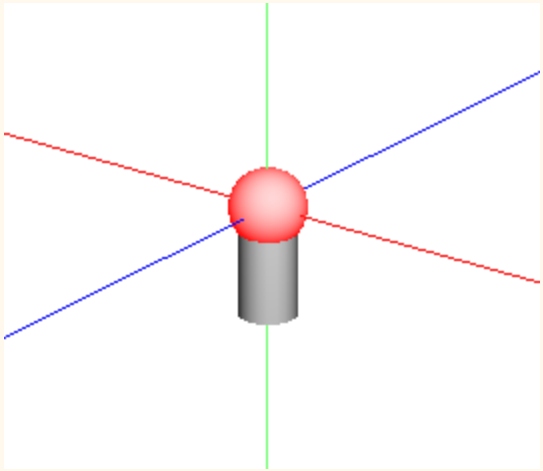
Primitivas modelo Robot “ビリビリ”

-Cabeza



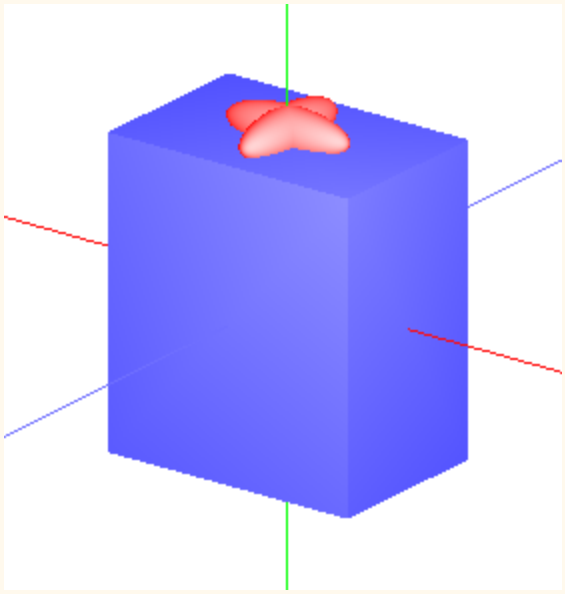
La cabeza consta de 8 primitivas, estas son conos para los ojos, un cubo para representar el volumen de la propia cabeza, una antena que es un cilindro, y dos esferas que hacen de hélice para la antena del robot.

-Cuello



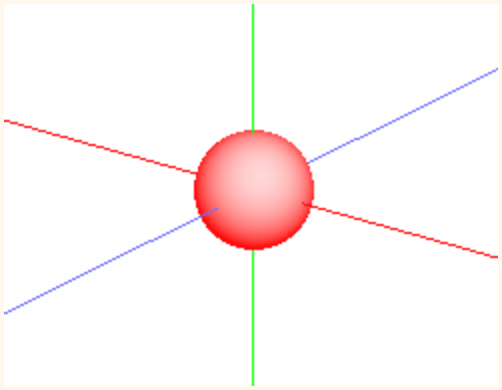
Esta consta de dos primitivas que forman el cuello.
Son un cilindro y una esfera.

-Torso



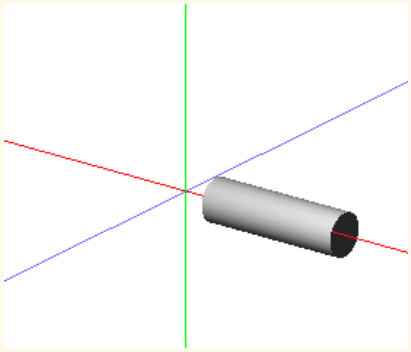
El torso consta de 3 primitivas, una de ellas es la base del cuerpo como tal, que es un cubo. Por otro lado, como adorno tenemos dos esferas deformadas a modo de ornamento para el cuello.

-Brazo



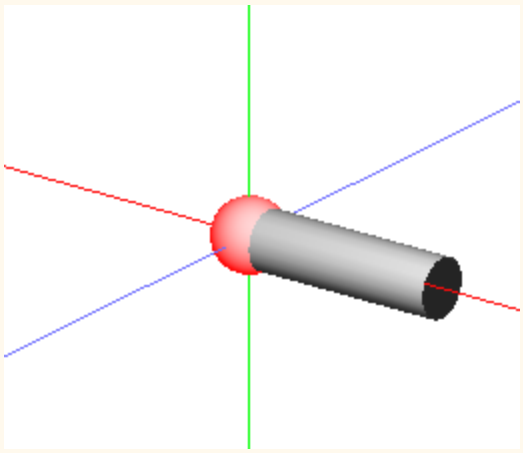
Esta primitiva representa la articulación que mueve todo el brazo en su completitud, este es una esfera.

-Brazo superior



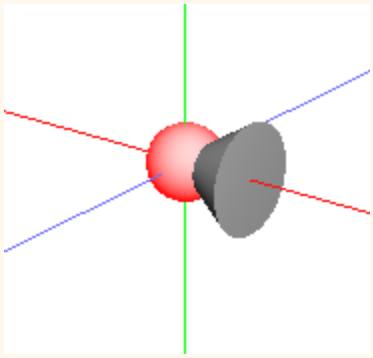
Simplemente un cilindro

-Brazo inferior



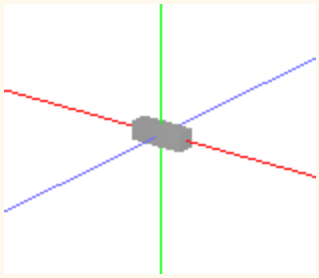
Este consta de dos primitivas, que son una esfera y un cilindro

-Mano



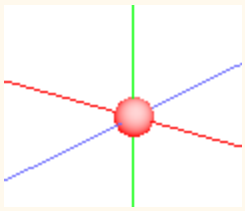
Esta primitiva consta de un cono a modo de palma de mano robótica y de una esfera como articulación.

-Dedo



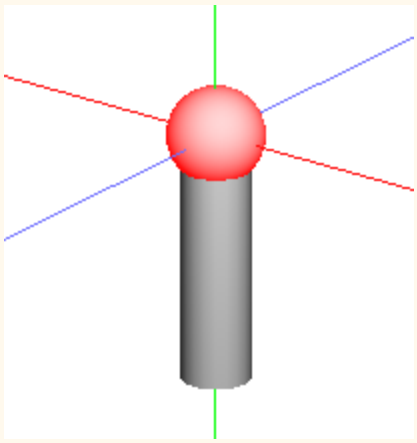
Simple cubo que es escalado para simular un dedo.

-Articulación de dedo



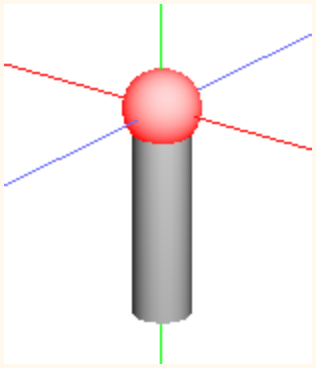
Simple esfera que representa la articulación de un dedo

-Pierna



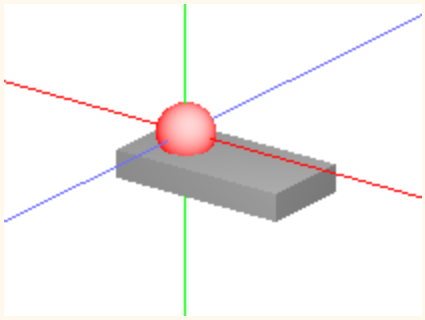
Esta primitiva representa de cadera a rodillas, consta de un cilindro y una esfera que hace de articulación.

-Pierna inferior



Es idéntico al modelo anterior, sin embargo, su escala es la diferencia, esta es menor para representar la parte inferior de la pierna que precede al pie.

-Pie



Esta primitiva representa el pie con su tobillo y su planta del pie, consta de una esfera a modo de articulación y de un cubo transformado para parecerse lo más posible a la planta del pie de un robot convencional.

Árbol de grafo de escenas del modelo

En el siguiente enlace se puede observar el modelo entero más cómodamente:
https://lucid.app/documents/view/70ac911c-351a-4a1f-ba1e-6f77466ffc6a
Aunque es más recomendable observar el modelo con el primer enlace es también posible observarse a través de google drive:
https://drive.google.com/file/d/1YdHZOjMoWnDHK_ncZFk05lAs8Gtbzz4y/view?usp=sharing