Informática Gráfica

Grados en Ingenierías Informática, de Computadores y del Software

Curso 13-14. Práctica 2.0

Carácter: obligatorio.

Fecha de entrega: domingo 24 de noviembre (partes obligatoria y opcional). La defensa de las prácticas se llevará a cabo el lunes 25 en la sesión de laboratorio.

Objetivo: colisiones, rebotes, simulación de movimiento.

Descripción: se trata de implementar el movimiento lineal de una *pelota* en una habitación cerrada en la que se encuentran varios *obstáculos* inmóviles. Aunque tienes libertad para elegir cuántos obstáculos introduces, cuál es su tamaño y cuál es su forma, al menos debes incluir un obstáculo de cada una de estas dos categorías:

1. Triángulos
2. Círculos

Esta clasificación se basa en la forma en que resuelven la cuestión "¿la pelota me golpea?". Concretamente, implementarás los algoritmos específicos para triángulos y círculos vistos en clase. No se admitirá implementar en su lugar el algoritmo de Cyrus-Beck.

Las pelotas, por su parte, son círculos, que se desplazan con velocidad lineal constante por la habitación. Entre sus atributos se incluyen el polígono regular que se utiliza para representarla gráficamente, su radio, la posición de su centro, y el vector que expresa su movimiento. Puedes entender que el módulo de este vector corresponde a su velocidad.

Al moverse por la habitación, la pelota puede colisionar con las paredes o con los obstáculos, en cuyo caso rebotará por reflexión. Por último, la generación de las pelotas tendrá lugar en una determinada posición de la habitación que debe estar suficientemente apartada de las paredes y de los obstáculos, para garantizar que pueden generarse sin problemas. El vector de movimiento de cada pelota se inicializa de forma aleatoria.

Detalles de la implementación: Estructura de la información

Debes estructurar tu código usando entre otras las siguientes clases. Otorga a cada clase el comportamiento que le corresponde.

1. `PV2D` guarda las coordenadas de un punto o vector en dos dimensiones.
2. `Obstaculo` es una interfaz que ofrece un método para resolver el problema de la intersección con una pelota. Devolverá un booleano para indicar si hay colisión y, en caso afirmativo, también el t_{HIT} y la normal implicada. La implementación de este método se realizará en las clases que extiendan esta interfaz.
3. `Triangulo` y `Circulo` extienden `Obstaculo` y resuelven la colisión contra pelota de forma específica.
4. *Pelota* guarda su radio, la posición de su centro, su vector de movimiento y el círculo que usamos para dibujarla.

Detalles de la implementación: Diseño algorítmico

1. Puedes suponer que las dimensiones de la ventana donde se presenta la habitación no pueden modificarse. Así no debes preocuparte del callback `resize`.
2. La animación se consigue ejecutando repetidas veces una función que denominaremos `step`, y que se encarga de avanzar la animación un paso. Esta función determina primero si se producirá alguna colisión usando el vector de movimiento de la pelota, para a continuación mover la pelota adecuadamente. Tu implementación debe permitir visualizar la animación de dos formas:
   - De forma *controlada*, esto es, la animación avanza paso a paso como respuesta a eventos del teclado. Un evento supone ejecutar la función `step` una sola vez. Este modo de visualización te permitirá depurar tu código con comodidad.
   - De forma *automática*; esto es, la animación se consigue programando un reloj que ejecute en cada ciclo la función `step`. En este modo de visualización la pelota parece moverse de forma continua. Para registrar el callback asociado a un reloj usa el comando `glutTimerFunc`.
3. Con respecto a las colisiones que sufrirá la pelota:
   a. Ten en cuenta los posibles errores de redondeo que puedan ocasionarse al implementar los algoritmos de intersección. Puede ser buena solución usar en ciertas ocasiones un valor muy pequeño en lugar del 0 exacto. Usar el tipo `double` en lugar de `float` también puede resultar beneficioso.
   b. Las paredes de la habitación pueden conseguirse colocando convenientemente cuatro triángulos en la escena.
   c. Al impactar la pelota resulta interesante suponer que su movimiento (el que le corresponde avanzar en el paso actual) se agota justo al colisionar; es decir, que la pelota se detiene al impactar y renuncia a avanzar tras el rebote lo que "le quede por gastar" de su vector de movimiento. De esta forma, la pelota empezaría a moverse en el siguiente paso.
   d. El sentido de la pelota tras el choque se calcula mediante reflexión. Su tamaño debe coincidir con el del antiguo sentido, para así garantizar que la velocidad de la pelota permanece constante.
   e. Pinta las normales de cada obstáculo para facilitar la depuración de los algoritmos de intersección.
4. La generación de números aleatorios se consigue con los métodos `randomize()` y `random(int num)` de la clase `Math`.

Detalles de la implementación: Etapas de desarrollo

Lo que sigue son las etapas que puedes seguir para desarrollar la implementación cómodamente. Como es habitual resulta conveniente ir archivándolas tras probar que funcionan adecuadamente.

- Una pelota moviéndose con libertad (sin obstáculos).
- Una pelota rebotando en habitación cerrada.
- Una pelota rebotando en habitación cerrada con triángulos.
- Una pelota rebotando en habitación cerrada con círculos.
- Una pelota rebotando en habitación cerrada con ambos tipos de obstáculos.

Parte Opcional

[+] En cuanto a la gestión de colisiones, tu práctica debe admitir dos modos de ejecución. En el más sencillo se permitirá la penetración de la pelota, ya que supondremos que la pelota es simplemente una partícula sin masa (su centro). En este modo, los algoritmos de intersección son exactamente los vistos en clase. En el más complicado, evitarás la penetración recubriendo cada obstáculo con una corteza cuyo grosor sea el radio de la pelota. Para el caso de los triángulos elige entre estas dos opciones:

1. La corteza es un polígono convexo que recubre el triángulo, y que resuelve la colisión con la pelota mediante el algoritmo de Cyrus-Beck. Usa arcos para redondear la corteza a su paso por los vértices del obstáculo original.
2. La corteza es un objeto compuesto formado por tres triángulos y tres círculos. Si T es el triángulo original, usa un nuevo triángulo para recubrir cada arista de T, y un círculo para envolver cada vértice de T.