IG1-2007-FEB.pdf

Anónimo

Informática Gráfica I

2º Grado en Desarrollo de Videojuegos

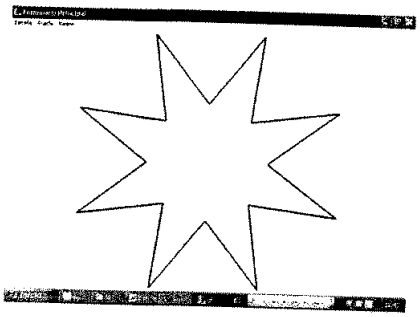
Facultad de Informática
Universidad Complutense de Madrid

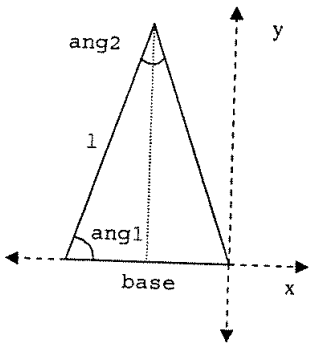
Informática Gráfica

Ingeniería en Informática.

Curso 2006-2007. Parcial de Febrero

1. Dibuja la estrella que resulta al sustituir cada uno de los lados de un polígono regular por el "pico" de un triángulo isósceles cuya base coincide con el lado que reemplaza. Observa que la base de cada uno de estos triángulos no debe pintarse. Por ejemplo, la figura de la izquierda se ha construido a partir de un polígono regular de ocho lados.





   a. **[1.25]** Implementa el método `void isosceles(GLdouble base, GLdouble angl, bool conBase)` para que dibuje un triángulo isósceles situado en el marco actual como muestra la figura de la derecha. La longitud de la base será `base` y cada uno de los ángulos iguales, `angl`. Antes de hacerlo debes averiguar la longitud de los otros lados (`l`) y el ángulo opuesto a la base (`ang2`). El parámetro `conBase` indica si la base del triángulo debe pintarse o no. Es obligatorio realizar el dibujo usando **un objeto de la clase `Lapiz`** que vimos en clase.

   b. **[2.5]** Implementa el método `void estrella(PV* centro, GLdouble r, int n, GLdouble angl)` para que dibuje una estrella a partir del polígono regular de `n` lados inscrito en una circunferencia de radio `r` centrada en `centro`. El parámetro `angl` sirve para indicar el ángulo que forma cada pico con el lado que sustituye. Es obligatorio realizar el dibujo usando el método `isosceles` y las **transformaciones afines** oportunas.

Además de los métodos habituales para puntos y vectores que se encuentran en la clase `PV`, sólo dispones de los siguientes métodos en la clase `Lapiz`:

   a. `Lapiz()` que construye un nuevo lápiz en el origen de coordenadas con dirección 0 (eje positivo de la equis).
   b. `void forward(GLdouble dist, bool pintando)` que hace avanzar al lápiz en línea recta la distancia `dist`. El lápiz deja rastro a su paso según indique el parámetro `pintando`.
   c. `void turn(GLdouble a)` que gira la dirección del lápiz el ángulo `a` hacia la izquierda. Recuerda que para girar a la derecha debes girar `-a`.

2. Queremos incluir un nuevo tipo de obstáculo en la práctica que simula el movimiento de una pelota. Se trata de la recta definida por la expresión `A + ta`, donde `A` es un punto de la recta y `a` es el vector correspondiente a su dirección. No hay restricciones para el parámetro real `t`, ya que consideramos que la recta es infinita. En adelante supón que ya hemos definido la clase `Recta` para gestionar estos obstáculos, y que en ella disponemos de los atributos privados `A` y `a`. Por su parte la pelota viene determinada por su radio `r`, la posición de su centro `c`, y el sentido de su movimiento `s`. Como es habitual entendemos que un paso de la simulación es capaz de avanzar todo el vector `s`, o una fracción de `s` si la pelota choca con un obstáculo antes.

   a. **[2.5]** Implementa en la clase `Recta` el método `bool golpeaPelotaARecta( PV* c, PV* s, GLdouble r, GLdouble& tHit, PV*& nHit)` para que decida si la pelota chocará en su próximo paso contra la recta que recibe el mensaje. En caso de colisión, `tHit` devolverá la fracción de `s` en que se produce el impacto ($0<$`tHit`≤ 1), y `nHit` la normal que usaremos para hacer rebotar la pelota. No se permitirá la penetración de la pelota, por lo que al moverla debes usar el primer punto de la circunferencia que golpearía la recta. Observa que la pelota puede golpear en ambos lados de la recta. Puedes suponer que la situación inicial de la pelota no interseca con la recta.

Suponiendo ahora que la recta es un obstáculo móvil y que su movimiento no es sencillo, pretendemos aproximar el momento en que la recta golpearía la pelota, que ahora se considera inmóvil. Para ello debes:

   b. **[1.25]** Implementar en la clase `Recta` el método `bool invadeRectaAPelota( PV* c, GLdouble r)` para que decida si la recta que recibe el mensaje está invadiendo la pelota.
   c. **[2.5]** Implementar en la clase `Recta` el método `bool golpeaRectaAPelota( PV* c, GLdouble r, GLdouble& tHit, GLdouble epsilon)` para que decida si la recta en su próximo movimiento golpeará la pelota. En caso de impacto, `tHit` devolverá el momento aproximado de la colisión. Para calcularlo debes usar el algoritmo de bisección para ir estrechando el intervalo `[tMin, tMax]` en el que se encuentra `tHit`, deteniendo el proceso cuando la longitud del intervalo sea menor que `epsilon`. Se supone que antes de comenzar a moverse, la recta no invade la pelota. También se supone que la recta no puede atravesar totalmente la pelota en un solo paso. Para realizar el movimiento de la recta dispones del método privado `void avanzaRecta( GLdouble t)` para avanzar la recta que recibe el mensaje hasta el instante `t` ($0\leq$`t`≤ 1, 1 indica que el paso de la recta se ha completado). Obviamente, la ejecución de este método oculta la forma en que evolucionan los parámetros que caracterizan la recta (`a` y `A`).

Cuentas con la clase `PV` para almacenar las coordenadas (`GLdouble`) de un punto o vector, y los siguientes métodos además del constructor habitual:

   a. `GLdouble getX(), GLdouble getY()` para acceder a las coordenadas, y `void setXY(GLdouble x1, GLdouble y1)` para fijarlas.
   b. `PV* clona()` que devuelve un clon del objeto que recibe el mensaje
   c. `GLdouble dot(PV* pv)` que calcula el producto escalar entre el objeto que recibe el mensaje y `pv`.
   d. `void escalar(GLdouble f)` que multiplica las coordenadas del objeto receptor por el factor `f`.
   e. `void suma(PV* v2)` que suma al objeto receptor el vector `v2`.