InfGraf- MiniProyecto

Daniel Pérez López

Fernando Morell Fernández

Índice

1. Introducción

Este proyecto tiene como objetivo desarrollar un juego interactivo estilo Arkanoid utilizando el algoritmo de detección de color conocido como Camshift (Continuously Adaptive Mean Shift). Este algoritmo, ampliamente utilizado en aplicaciones de visión por ordenador, permite rastrear objetos en movimiento basándose en las características de color en tiempo real.

El juego, inspirado en el clásico Arkanoid, desafiará al jugador a controlar una barra que rebota una pelota para romper bloques en pantalla. A diferencia de la interacción tradicional con teclado o ratón, el control de la barra se logrará mediante el seguimiento del movimiento de un objeto de color específico, lo que introduce un componente innovador e inmersivo. Este enfoque aprovecha el potencial de la visión por ordenador para transformar objetos cotidianos en controles físicos, haciendo que la experiencia sea más intuitiva y dinámica.

El desarrollo de este proyecto combina principios de programación gráfica, diseño de juegos y procesamiento de imágenes, integrando conceptos como la segmentación de color, el seguimiento de objetos y la implementación de físicas básicas. El uso del algoritmo Camshift no solo permite el rastreo eficiente de un objeto con base en su histograma de color, sino que también ajusta dinámicamente la región de interés, garantizando un seguimiento preciso incluso ante cambios de tamaño o posición del objeto.

En este documento se detallarán las etapas del desarrollo, desde la implementación del algoritmo hasta la integración del sistema de control en el juego, ofreciendo una solución interactiva que une tecnología y entretenimiento.

2. Juego

El juego es una versión moderna del clásico Arkanoid, donde el objetivo principal es destruir todos los ladrillos en pantalla utilizando una bola que rebota en las paredes y en una plataforma controlada por el jugador. Cada nivel o fase introduce nuevos retos con patrones de ladrillos más complicados y distribuciones estratégicas. El jugador debe evitar que la bola caiga fuera del área de juego, preservando sus vidas limitadas mientras acumula puntos al destruir ladrillos.

Ladrillos

Los ladrillos son los elementos principales a destruir en el juego. Están organizados en una matriz rectangular y se generan dinámicamente según la fase que se vaya a jugar. Cada ladrillo tiene un color de entre los siguientes: negro, rojo, gris, azul cian, amarillo, marrón, verde y rosa. Los ladrillos se representan mediante la clase Rectangulo, la cual gestiona sus propiedades y estado (válido o inválido) para determinar si deben ser renderizados o no.

Fases

El juego está dividido en fases, que definen la disposición inicial de los ladrillos en la pantalla. Cada fase se codifica como una cadena de caracteres donde cada letra representa un color o un espacio vacío. Por ejemplo:

- 'N' para ladrillos negros
- 'R' para ladrillos rojos
- ' ' para áreas vacías

Estas cadenas se utilizan para generar visualmente los niveles mediante la función generarBloques, que convierte las cadenas en estructuras gráficas en la pantalla. Con cada nueva fase, aumenta el desafío con patrones más complejos y bloques colocados estratégicamente.

Vidas

El jugador cuenta con un número limitado de vidas, generalmente inicializadas en 3. Cada vez que la bola atraviesa la parte inferior de la pantalla sin golpear la plataforma, el jugador pierde una vida. El juego finaliza si el jugador pierde todas las vidas antes de completar la puntuación máxima.

Puntuación

La puntuación refleja el progreso del jugador y se incrementa al destruir ladrillos. Cada ladrillo aporta un punto al destruirse, y la puntuación total necesaria para ganar se calcula a partir de la cantidad de ladrillos disponibles en todas las fases. Si el jugador alcanza la puntuación máxima (PUNT_MAX), el juego despliega una pantalla de felicitación indicando que el nivel ha sido completado.

3. Detección del color

En el desarrollo del juego basado en el algoritmo Camshift, varios pasos fundamentales son necesarios para lograr el rastreo de un objeto en tiempo real utilizando las características de color. A continuación, se describen en detalle los procesos involucrados:

Selección del Color

La selección del color es el primer paso para definir el objeto que se rastreará. A través de interacciones con el ratón, el usuario puede marcar un área de interés (ROI) sobre el objeto deseado en el marco de video. Este proceso inicializa el algoritmo Camshift, configurando la región que se utilizará para calcular las propiedades del histograma de color y otros parámetros relevantes para el seguimiento.

El evento de clic izquierdo activa la selección, y el área se define al arrastrar el ratón desde un punto inicial hasta el punto final. Una vez que se libera el botón del ratón, la región seleccionada se establece como el ROI y se activa el rastreo del objeto.

Formato HSV

El espacio de color HSV (Tono, Saturación y Valor) es ideal para tareas de procesamiento de imágenes porque separa la información de color (Hue) de la luminancia (Value). Esto permite un mejor control sobre las variaciones de iluminación, comúnmente presentes en videos en tiempo real.

Para convertir el marco de video al formato HSV, se utiliza la función cvtColor de OpenCV, que convierte cada píxel del espacio de color RGB al espacio HSV. Una vez convertido, el canal de Tono (Hue) se extrae para enfocarse exclusivamente en las características de color del objeto.

Creación del Histograma

El histograma es una representación estadística que muestra la distribución de los valores de color dentro del ROI seleccionado. En este proyecto:

- Se extrae el canal de Tono de la región seleccionada.
- Se utiliza una máscara para ignorar píxeles no deseados.
- Con la función calcHist, se genera el histograma basado en los valores de Tono.

El histograma se normaliza usando normalize para asegurar que los valores estén en un rango adecuado para ser procesados por Camshift, mejorando la precisión en el rastreo. También se genera una representación gráfica del histograma para depurar y visualizar la distribución del color.

Creación de Máscara

La máscara es una imagen binaria que ayuda a filtrar los píxeles del marco de video que no cumplen con ciertos criterios de color y saturación. Se genera usando la función inRange, la cual aplica límites inferiores y superiores definidos por los valores de saturación (smin) y brillo (vmin, vmax). Esto elimina ruidos y áreas irrelevantes del video, permitiendo al algoritmo enfocarse en el objeto rastreado.

Selección de TrackBox

Una vez calculado el histograma, se realiza la proyección hacia atrás (back projection) del histograma sobre el marco de video. Esto genera un mapa que resalta las áreas del video que coinciden con los valores de color del objeto rastreado.

El algoritmo Camshift utiliza este mapa para actualizar dinámicamente la ventana de seguimiento (trackWindow). Inicialmente, esta ventana corresponde a la región seleccionada, pero se ajusta automáticamente en tamaño y posición durante el rastreo para reflejar cambios en el objeto, como su movimiento o variación en dimensiones.

Creación de RotatedRect

El resultado del algoritmo Camshift es un objeto cv::RotatedRect, que define una caja envolvente ajustada al objeto rastreado. Este rectángulo puede rotar para adaptarse a la orientación del objeto y se utiliza en este proyecto para actualizar la posición de la barra controladora del juego.

El RotatedRect incluye información clave como:

- El centro del objeto rastreado.
- El tamaño y orientación del rectángulo.

Esta característica asegura un seguimiento más preciso, especialmente cuando el objeto cambia de posición o forma en el video.

4. Colisión

El comportamiento y las interacciones de la pelota con distintos objetos en el entorno se gestionan cuidadosamente en el código. A continuación, se explican los aspectos clave:

Velocidad de la Pelota

La velocidad de la pelota está representada como un vector bidimensional (Point2d) que indica su magnitud y dirección en el espacio 2D. La pelota se mueve según su velocidad, actualizándose en cada iteración del bucle principal. Los cambios en su velocidad ocurren principalmente en respuesta a colisiones, ya sea con los ladrillos, los bordes de la pantalla o el jugador. La velocidad se ajusta al invertir la componente correspondiente del vector en función de la superficie de impacto.

Colisión con Ladrillo

La interacción entre la pelota y los ladrillos implica varios tipos de colisiones:

1. Colisión con el Lado del Ladrillo
   La pelota rebota en los lados del ladrillo (superior, inferior, izquierdo o derecho).
   - Si el impacto ocurre en el lado superior o inferior, la componente vertical de la velocidad se invierte (velocidad.y = -velocidad.y).
   - Si el impacto es en los lados izquierdo o derecho, se invierte la componente horizontal (velocidad.x = -velocidad.x).
   - Ambos escenarios anteriores son una optimización del caso general que se estudiará más adelante. Puesto que los lados son siempre completamente verticales u horizontales.
   - El ladrillo afectado se marca como destruido, y la pelota continúa su trayectoria.
2. Colisión con la Esquina del Ladrillo
   Si la pelota impacta cerca de una esquina del ladrillo, se calcula una normal promedio basada en los vectores de las dos aristas adyacentes, en este caso como sabemos que los lados siempre son verticales u horizontales podemos asignar siempre el mismo valor para la misma esquina. Es decir, la inferior derecha tendrá el vector normal (1,1) mientras que la superior izquierda tendrá el (-1, -1).
   - La nueva dirección de la pelota se obtiene reflejando su velocidad respecto a esta normal utilizando la fórmula:

$$\vec{v}_{reflejada} = \vec{v} - 2(\vec{v} \cdot \vec{n})\vec{n}$$

3. Donde v es la velocidad actual y n la normal calculada.
   - Este cálculo asegura un rebote realista en la esquina.
4. Colisión con Ladrillos Adyacentes
   En casos donde la pelota colisiona con una esquina, se verifica si hay ladrillos adyacentes en las posiciones inmediatas.

- Si hay un ladrillo adyacente válido en la dirección de la velocidad, la colisión se maneja como un impacto en un lado en lugar de una esquina.
- Esto evita comportamientos inexactos donde la pelota podría interactuar con múltiples ladrillos simultáneamente. Además, aporta realismo ya que aunque choque con la esquina de un ladrillo, si está contiguo con otro el rebote se produce realmente en una superficie plana.

Colisión con el RotatedRect (Jugador)

El jugador está representado como un RotatedRect, que puede tener orientación arbitraria en el espacio. Las colisiones con este objeto se manejan con mayor complejidad debido a la posible rotación:

1. Colisión con una Esquina del RotatedRect
   - Si la pelota impacta en una esquina, se calcula la normal como promedio de las normales de las dos aristas que comparten la esquina.
   - La velocidad de la pelota se refleja con respecto a esta normal, simulando un rebote realista.
2. Colisión con los Lados del RotatedRect
   - Se determina el lado más cercano a la bola de los cuatro que forman el RotatedRect.
   - Una vez identificado, se calcula una normal perpendicular a este lado, y la velocidad de la pelota se refleja en relación con esta normal.
3. Verificación de Dirección
   En todos los casos, antes de aplicar un cambio de velocidad, se verifica que la pelota realmente se esté moviendo hacia el RotatedRect. Esto se logra calculando el producto vectorial entre la velocidad de la pelota y la normal correspondiente:
   - Si el producto vectorial es positivo, la pelota ya está moviéndose hacia fuera del RotatedRect y no se considera una colisión, evitando así que la pelota quede encerrada en el rectángulo.

Este enfoque asegura que las colisiones con el jugador sean consistentes, independientemente de la orientación del rectángulo.

Iteraciones internas

Por último, comentar que para mejorar la precisión a la hora de detectar las colisiones para cada *frame* que se dibuja se realizan 10 iteraciones del bucle principal del juego. Es decir, si el usuario ve la bola ir del punto A al punto B de un frame a otro, realmente la bola se ha movido 10 veces y se han comprobado las colisiones con los ladrillos y el jugador en cada movimiento de la bola.

5. Conclusión y vías futuras

El juego presentado ya ofrece una experiencia divertida y funcional basada en un clásico de los videojuegos. Sin embargo, existen múltiples oportunidades de mejora que podrían llevarse a cabo.

Actualmente, la pelota se mueve de manera uniforme siguiendo la dirección de su velocidad inicial y rebota en las paredes y ladrillos. Para agregar más realismo al comportamiento de la pelota, se podría incorporar un sistema de inercia. Para ello, sería conveniente suavizar las transiciones aplicando una interpolación lineal, de este modo sabríamos la velocidad del RotatedRect con precisión.

La incorporación de la interpolación lineal entre iteraciones también sería beneficiosa a la hora de detectar las colisiones de una manera aún más exacta.