BSP

Resumen del Ejercicio:

El ejercicio consiste en implementar una función llamada bsp (Binary Space Partitioning) que determina si un punto dado está dentro de un triángulo definido por tres vértices. Además, se debe desarrollar una clase Point en formato Canónico Ortodoxo (es decir, siguiendo las reglas de diseño estándar de clases en C++). La clase Point utiliza una clase Fixed, previamente desarrollada, para representar coordenadas con precisión fija.

Objetivo del Ejercicio:

1. **Objetivo Principal:** Comprender y aplicar el algoritmo para determinar si un punto está dentro de un triángulo (técnica fundamental en gráficos y geometría computacionales).
2. **Uso Práctico:** Aplicar la clase Fixed en un contexto realista (manejo de coordenadas con precisión fija).
3. **Desarrollo de Buenas Prácticas:** Diseñar clases según la Canónica Ortodoxa para reforzar principios de encapsulación, robustez y mantenibilidad.

Materias de C++ 98 a Desarrollar:

1. **Clases en Canonical Form:**
   a. Uso de constructores, destructores, operadores de copia, etc.
   b. Const-correctness para atributos y métodos.
2. **Manejo de Precisión Fija:**
   a. Uso de la clase Fixed para representar coordenadas de puntos.
3. **Sobrecarga de Operadores:**
   a. Posible uso para operaciones con la clase Point o Fixed.
4. **Funciones Matemáticas:**
   a. Uso de roundf y posibles cálculos geométricos como determinantes para comprobar la posición del punto respecto al triángulo.
5. **Desarrollo Modular:**
   a. Separación adecuada de código en diferentes archivos (.h, .cpp, etc.).
6. **Testeo:**
   a. Creación de pruebas personalizadas para verificar el comportamiento del programa.

Enfoque para Afrontar el Ejercicio:

1. **Planificación:**
   a. Identificar claramente las dependencias: la clase Point usa Fixed, y la función bsp usa Point.
   b. Diseñar el prototipo de la función bsp y las operaciones necesarias.
2. **Implementación de la Clase Point:**
   a. Asegurarse de que sigue la Canonical Form.

   b. Usar la clase Fixed para las coordenadas x e y.
   c. Añadir constructores y métodos útiles para manejar puntos en operaciones geométricas.
3. **Lógica de la Función bsp:**
   a. Implementar el algoritmo geométrico para comprobar si un punto está dentro del triángulo:
      i. Usar áreas o productos cruzados para determinar la posición relativa del punto.
      ii. Asegurarse de manejar correctamente casos donde el punto esté en un borde o vértice.
   b. Mantener el código limpio y legible.
4. **Pruebas:**
   a. Crear diferentes casos de prueba:
      i. Punto dentro del triángulo.
      ii. Punto fuera del triángulo.
      iii. Punto en un borde o vértice.
   b. Asegurarse de cubrir casos límite y validar la función.
5. **Optimización y Revisión:**
   a. Verificar el cumplimiento del formato Canónico Ortodoxo.
   b. Revisar el código en busca de optimizaciones y posibles errores.

Conclusión:

Este ejercicio no solo refuerza fundamentos de geometría computacional, sino que también fomenta el uso de buenas prácticas de programación en C++. Al abordarlo, estarás desarrollando habilidades esenciales como diseño modular, manejo de precisión fija y algoritmos matemáticos. Además, te familiarizarás con el proceso de escribir código limpio y testeable, lo que es crucial para proyectos más grandes en el futuro.

Fundamentos Geométricos y Matemáticos

¿Por qué usar triángulos?

Los triángulos son la forma más simple de polígono y tienen propiedades únicas que los hacen ideales para la representación de áreas en la geometría computacional. Algunas de estas propiedades son:

- **Rigidez:** Un triángulo está completamente definido por sus tres vértices. No puede deformarse sin cambiar la longitud de sus lados.
- **Teselación:** Cualquier polígono puede ser descompuesto en triángulos. Esto hace que los triángulos sean una estructura de datos fundamental en muchos algoritmos de gráficos y geometría.
- **Facilidad de cálculo:** Las operaciones geométricas en triángulos suelen ser más sencillas que en polígonos de mayor número de lados.

Conceptos clave

- **Vector:** Un segmento de línea dirigido en el espacio. Un vector tiene una magnitud (longitud) y

una dirección.

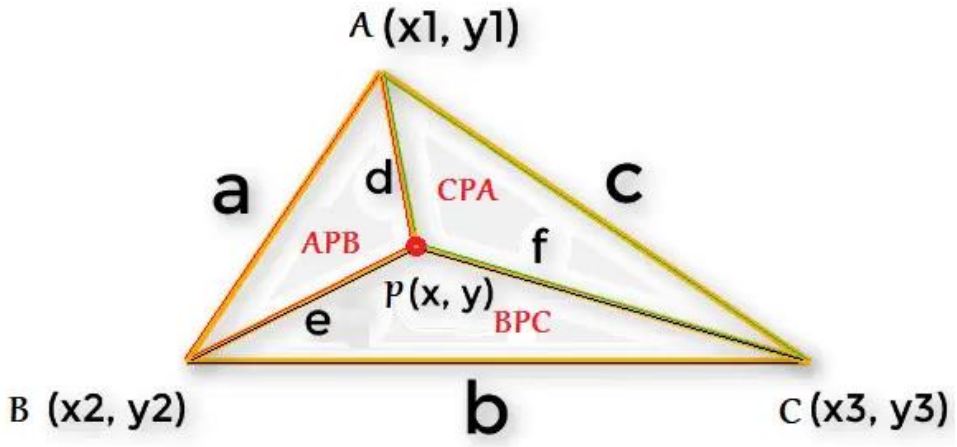
- **Producto cruz:** Una operación binaria en dos vectores en un espacio tridimensional. El resultado es un vector perpendicular al plano definido por los dos vectores originales. La magnitud del producto cruz es igual al área del paralelogramo definido por los dos vectores.
- **Orientación:** La orientación de un conjunto de puntos (por ejemplo, los vértices de un triángulo) indica si los puntos están ordenados en sentido horario o antihorario.
- **Área de un triángulo:** Se puede calcular utilizando la fórmula de Herón o, de manera más eficiente, utilizando el producto cruz.

Algoritmo básico

La idea principal es dividir el triángulo en tres subtriángulos, cada uno formado por el punto a evaluar y dos vértices del triángulo original. Si la suma de las áreas de estos subtriángulos es igual al área del triángulo original, entonces el punto está dentro del triángulo.

Pasos:

1. **Calcular el área del triángulo original:** Usando el producto cruz entre dos de los lados del triángulo.
2. **Calcular el área de cada subtriángulo:** Usando el producto cruz entre el punto a evaluar y los vértices del triángulo original.
3. **Sumar las áreas de los subtriángulos:** Si la suma es igual al área del triángulo original, el punto está dentro.



Fundamentos de Programación

Clase Point

- **Encapsulación:** Cada punto tiene sus coordenadas x e y encapsuladas dentro de la clase. Esto evita errores y facilita la gestión de los puntos.
- **Sobrecarga de operadores:** Podemos sobrecargar operadores como +, -, * para realizar operaciones vectoriales de manera intuitiva.
- **Const-correctness:** Garantiza que los métodos que no modifican el objeto sean marcados como `const`, evitando modificaciones accidentales.

Función bsp

- **Modularidad:** La función `bsp` se encarga exclusivamente de determinar si un punto está dentro de un triángulo, separando la lógica geométrica del resto del código.
- **Claridad:** Un buen nombre de función y comentarios explicativos hacen que el código sea más fácil de entender y mantener.

Lógica y Algoritmos

- **Condiciones de borde:** Es importante considerar los casos en los que el punto está exactamente sobre un borde o en un vértice del triángulo.
- **Orientación:** La orientación de los puntos puede utilizarse para determinar si un punto está dentro o fuera del triángulo.
- **Precisión:** Al trabajar con coordenadas, es fundamental tener en cuenta los errores de redondeo y utilizar una representación numérica adecuada (como la clase Fixed).

En resumen

Al combinar estos conceptos de geometría, álgebra lineal y programación orientada a objetos, podemos desarrollar una solución robusta y eficiente para determinar si un punto está dentro de un triángulo.

bsp.cpp - Fixed.cpp - Fixed.hpp - main.cpp - Point.cpp - Point.hpp

Comprender las funcionalidades principales

El código proporcionado implementa una prueba de punto en un triángulo, un problema fundamental en geometría computacional. Aprovecha el concepto de coordenadas baricéntricas para determinar si un punto dado se encuentra dentro de un triángulo.

Clases y funciones clave:

- **Fixed:**
  - Representa un número de punto fijo, que ofrece un control preciso sobre valores fraccionarios.
  - Proporciona operadores sobrecargados para operaciones aritméticas y de comparación.
- **Point:**
  - Representa un punto en el espacio 2D, utilizando coordenadas de punto fijo para mayor precisión.
- **área:**

  - Calcula el área de un triángulo dados sus tres vértices.
  - Utiliza la fórmula del cordón para un cálculo eficiente del área.
- **bsp:**
  - Implementa la prueba de coordenadas baricéntricas.
  - Calcula las áreas de tres subtriángulos formados por el punto y los vértices del triángulo.
  - Si la suma de las áreas de los subtriángulos es igual al área del triángulo original, el punto está dentro.

Explicación paso a paso

1. **Clase Fixed:**
   a. **Propósito:** Proporciona un tipo de datos personalizado para aritmética de punto fijo.
   b. **Implementación:** utiliza un número entero para representar tanto la parte entera como la parte fraccionaria de un número, y la cantidad de bits fraccionarios está controlada por fractBits.
   c. **Operadores:** sobrecarga los operadores aritméticos, de comparación y de asignación para permitir operaciones naturales en objetos fijos.
2. **ClaseP oint:**
   a. **Propósito:** Representa un punto en el espacio 2D.
   b. **Implementación:** utiliza dos objetos fijos para almacenar las coordenadas x e y.
   c. **Métodos:** Proporciona captadores para las coordenadas x e y.
3. **Función Área:**
   a. **Propósito:** Calcula el área de un triángulo dados sus tres vértices.
   b. **Implementación:**
      i. Extrae las coordenadas x e y de los vértices.
      ii. Aplica la fórmula del cordón para calcular el área.
      iii. Devuelve el valor absoluto del resultado.
4. **Función bsp:**
   a. **Propósito:** Determina si un punto se encuentra dentro de un triángulo.
   b. **Implementación:**
      i. Calcula el área del triángulo original.
      ii. Calcula las áreas de tres subtriángulos formados por el punto y los vértices del triángulo.
      iii. Compara la suma de las áreas de los subtriángulos con el área del triángulo original.
      iv. Si son iguales el punto está dentro.

Cómo funciona la prueba de coordenadas baricéntricas:

- **Coordenadas baricéntricas:** representan un punto dentro de un triángulo como una combinación ponderada de los vértices del triángulo.

- **Cálculo del área:** al calcular las áreas de los subtriángulos, esencialmente estamos encontrando los pesos de las coordenadas baricéntricas.
- **Prueba interna:** si la suma de los pesos es 1, el punto se encuentra dentro del triángulo.

Estructura y organización del código:

- **Archivos de encabezado:** Fixed.hpp y Point.hpp definen las clases y sus interfaces.
- **Archivos de implementación:** contienen las implementaciones reales de los métodos y funciones de la clase.
- **Función principal:** demuestra el uso de la función bsp con varios casos de prueba.

Puntos clave:

- El código combina la programación orientada a objetos con conceptos matemáticos.
- El uso de la clase Fixed garantiza cálculos precisos con aritmética de punto fijo.
- La prueba de coordenadas baricéntricas proporciona una forma sólida y eficiente de determinar relaciones entre puntos en un triángulo.
- El código está bien estructurado y es fácil de entender.

En esencia, este código proporciona una base sólida para los cálculos geométricos y puede ampliarse para resolver problemas más complejos en geometría computacional.