**Aproximarea poligonală a contururilor obiectelor**

Oprisiu Ionut Daniel

Grupa 30233

**Rationale and task management:**

**1. Obiectivul Proiectului și Overview**

Acest proiect vizează dezvoltarea unei soluții pentru aproximarea poligonală a contururilor obiectelor din imagini color (BMP, 24 biți/pixel). Metoda se desfășoară în două etape principale:

1. **Extragerea contururilor:** Se obțin contururile obiectelor prin aplicarea tehnicilor de preprocesare și edge detection (folosind, de exemplu, algoritmul Canny și funcțiile de detectare a contururilor din OpenCV).
2. **Aproximarea poligonală:** Se simplifică contururile extrase prin utilizarea algoritmului Ramer-Douglas-Peucker (prin funcția approxPolyDP din OpenCV), reducând numărul de puncte care descriu forma, păstrând totodată integritatea geometriei obiectului.

Această soluție reduce complexitatea datelor vizuale, facilitând aplicații practice precum vectorizarea imaginilor și recunoașterea obiectelor.

**2. Context și Motivație**

În cadrul materiei de procesare a imaginilor, proiectul abordează:

* **Vectorizarea grafică:** Transformarea imaginilor raster în reprezentări vectoriale, utile pentru editare, analiză și stocare eficientă.
* **Recunoașterea obiectelor:** Simplificarea contururilor pentru extragerea caracteristicilor esențiale, facilitând clasificarea și analiza formelor.
* **Aplicații industriale și de cercetare:** Reducerea complexității datelor geometrice contribuie la îmbunătățirea algoritmilor de analiză vizuală și la eficientizarea proceselor de prelucrare a imaginilor.

Implementarea acestui proiect va integra cunoștințele dobândite în laborator privind preprocesarea, extragerea și analiza contururilor.

**3. Planul de Implementare (Project Plan)**

Proiectul va fi structurat în următoarele etape:

**Etapa 1: Analiza și Preprocesarea Imaginilor**

* **Cercetare Preliminară:** Studierea tehnicilor de preprocesare, extragere a contururilor și a algoritmului Ramer-Douglas-Peucker.
* **Colectarea Datelor:** Selectarea unor imagini color BMP (24 biți/pixel) reprezentative pentru testare.
* **Preprocesare:** 
  + Convertirea imaginilor la grayscale (dacă este necesar) pentru a simplifica analiza.
  + Aplicarea unui filtru Gaussian pentru reducerea zgomotului, facilitând astfel extragerea contururilor clare.

**Etapa 2: Extragerea și Aproximarea Contururilor**

* **Extragerea Contururilor:** 
  + Aplicarea unui algoritm de edge detection (de exemplu, Canny) pentru evidențierea tranzițiilor semnificative în intensitatea pixelilor.
  + Utilizarea funcției findContours din OpenCV pentru a obține contururile obiectelor**.**
* **Aproximarea Poligonală:** 
  + Aplicarea algoritmului Ramer-Douglas-Peucker prin funcția approxPolyDP pentru a reduce numărul de puncte ale fiecărui contur.
  + Experimentarea cu parametrul de toleranță pentru a obține un echilibru optim între precizie și simplificare**.**

**Etapa 3: Evaluare și Feedback**

* **Testare Inițială:** Evaluarea rezultatelor prin compararea contururilor aproximative cu cele originale și analiza performanței în aplicații precum recunoașterea obiectelor.
* **Consultare:** Discuții pentru validarea metodei și identificarea posibilelor îmbunătățiri.

**4. Resurse și Tehnologii Utilizate**

* **C++:** Limbajul de implementare, integrând conceptele de procesare a imaginilor.
* **OpenCV:** Pentru preprocesarea imaginilor, extragerea contururilor și aplicarea algoritmului de aproximare (prin funcțiile findContours și approxPolyDP).
* **CMake:** Pentru gestionarea build-ului proiectului și facilitarea compilării.
* **Gestionarea Codului:** Utilizarea Git și GitHub pentru versionare și documentarea progresului.

**5. Provocări și Soluții**

* **Alegerea Parametrilor Optimi:** Stabilirea valorilor ideale pentru filtrul Gaussian și toleranța algoritmului Ramer-Douglas-Peucker printr-o abordare iterativă, folosind diverse seturi de imagini.
* **Optimizarea Performanței:** Asigurarea unui timp de execuție rezonabil prin optimizarea buclelor și a funcțiilor de procesare.
* **Calibrarea Rezultatelor:** Evaluarea subiectivă și obiectivă a aproximărilor pentru a se asigura păstrarea formei obiectelor în mod fidel.

**6. Timeline și Etapele de Dezvoltare**

* **Săptămânile 2–4 (Phase 1):** Definirea obiectivelor, redactarea documentului de proiect, cercetare și planificare.
* **Săptămânile 5–10 (Phase 2):** Implementarea preprocesării, extragerea contururilor și aplicarea algoritmului de aproximare.
* **Săptămânile 11–12 (Phase 3):** Testarea pe diverse scenarii și documentarea rezultatelor.
* **Săptămânile 13–14 (Phase 4):** Îmbunătățiri și optimizări bazate pe feedback-ul obținut în timpul testărilor.

**7. Concluzie**

Proiectul va demonstra integrarea teoriei și practicii în procesarea imaginilor, prin dezvoltarea unei soluții care extrage și simplifică contururile obiectelor. Documentarea detaliată și abordarea iterativă vor asigura o soluție robustă și eficientă, utilă în aplicații practice precum vectorizarea imaginilor și recunoașterea obiectelor.

**Implementation:**

**1. Arhitectura Sistemului**

**Structura Proiectului**

Proiectul este organizat în două componente principale:

* **Fișierul Header** (lab11.h):
* Definește interfața sistemului
* Conține structurile de date esențiale
* Declară toate funcțiile disponibile
* Stabilește constantele globale necesare
* **Fișierul de Implementare** (lab11.cpp):
* Oferă implementările concrete ale funcționalităților
* Conține logica de procesare
* Gestionează optimizările și validările

**2. Procesarea Imaginilor**

**2.1 Filtrare Gaussiană**

Sistemul implementează o filtrare Gaussiană sofisticată:

* **Generarea Kernel-ului**:
* Calculează automat parametrul sigma
* Generează coeficienții optimi
* Asigură normalizarea corectă
* **Aplicarea Filtrelor**:
* Procesează imaginea pe două direcții
* Gestionează eficient marginile
* Păstrează calitatea imaginii

**2.2 Detecția Marginilor**

**Calculul Gradientului**

Sistemul oferă multiple metode de calcul al gradientului:

* **Operatori Disponibili**:
* Sobel (precizie ridicată)
* Prewitt (sensibil la zgomot)
* Roberts (rapid, pentru imagini simple)
* **Procesare Gradient**:
* Calculul magnitudinii
* Determinarea direcției
* Cuantizarea în categorii

**Rafinarea Marginilor**

Implementarea include tehnici avansate de rafinare:

* **Supresia Non-Maximă**:
* Elimină marginile false
* Păstrează continuitatea
* Asigură precizie sub-pixel
* **Pragare Adaptivă**:
* Calculează praguri automate
* Adaptează-se la conținutul imaginii
* Optimizează detectarea
* **Hysteresis**:
* Conectează marginile relevante
* Elimină artefacte
* Asigură robustețe

**3. Procesarea Contururilor**

**3.1 Extragerea Contururilor**

Sistemul extrage contururile eficient:

* **Detecție Preciză**:
* Identifică contururi complete
* Păstrează topologia
* Gestionează complexitatea

**3.2 Aproximarea Poligonală**

Implementarea oferă metode sofisticate de aproximare:

* **Algoritmul Ramer-Douglas-Peucker**:
* Simplifică contururile
* Păstrează forma esențială
* Optimizează numărul de puncte
* **Rafinare Adaptivă**:
* Ajustează precizia
* Adaptează-se la complexitate
* Asigură reprezentare fidelă

**4. Configurare și Parametri**

**4.1 Parametri de Filtrare**

* **Kernel Gaussian**:
* Dimensiune optimă
* Sigma adaptiv
* Normalizare automată

**4.2 Parametri de Detecție**

* **Operator Gradient**:
* Alegere flexibilă
* Configurare precisă
* Adaptare la necesități
* **Pragare**:
* Valori adaptive
* Rata de hysteresis
* Sensibilitate ajustabilă

**5. Caracteristici Avansate**

**5.1 Optimizări**

* **Eficiență**:
* Procesare paralelă
* Gestionare memorie optimă
* Algoritmi rapizi
* **Calitate**:
* Precizie ridicată
* Robustețe la zgomot
* Rezultate consistente

**5.2 Flexibilitate**

* **Configurare**:
* Parametri ajustabili
* Multiple opțiuni
* Adaptare la necesități
* **Extensibilitate**:
* Ușor de extins
* Modular design
* Reutilizare cod

**6. Utilizare și Aplicații**

**6.1 Pipeline de Procesare**

1. **Preprocesare**:

* Filtrare zgomot
* Îmbunătățire contrast
* Normalizare

1. **Detecție**:

* Calcul gradient
* Rafinare margini
* Extragere contururi

1. **Post-procesare**:

* Aproximare poligonală
* Vizualizare
* Analiză rezultate

**6.2 Aplicații Practice**

* **Analiză Imagini**:
* Detecție obiecte
* Măsurare dimensiuni
* Recunoaștere forme
* **Procesare Industrială**:
* Control calitate
* Măsurare automată
* Inspecție vizuală

**7. Avantaje și Caracteristici Unice**

**7.1 Avantaje**

* **Precizie**:
* Rezultate exacte
* Robustețe
* Consistență
* **Eficiență**:
* Viteză procesare
* Utilizare resurse optimă
* Scalabilitate

Testing:

**Imaginea pe care am decis sa testez algoritmii:**

A bicycle with a basket in front of a door

AI-generated content may be incorrect.

**1. Test 1 – Compararea Algoritmului Canny**

**1.1 Obiectiv**

Validarea acurateții și evaluarea performanței implementării proprii a algoritmului Canny față de implementarea optimizată din OpenCV.

**1.2 Metrici urmărite**

* **Timp de execuție**
* **Diferența medie** dintre imaginile rezultate

**1.3 Rezultate**

* Implementare proprie: **344 ms**
* OpenCV: **46 ms**
* Diferență medie a rezultatelor: **52,4301**

**1.4 Analiză**

* Implementarea proprie este ≈ 7,5× mai lentă decât OpenCV, diferență așteptată din cauza optimizărilor avansate ale bibliotecii standard.
* Diferența medie de 52,4301 confirmă că rezultatele sunt **comparabile** calitativ, deși obținute mai lent.

**1.5 Concluzie**

Algoritmul nostru Canny este **corect** din punct de vedere funcțional, dar necesită **optimizări de performanță** pentru aplicații sensibile la timp de procesare.

**2. Test 2 – Compararea Algoritmului Ramer‑Douglas‑Peucker (RDP)**

**2.1 Obiectiv**

Evaluarea eficienței și a preciziei implementării proprii a algoritmului RDP la diferite valori **epsilon** (0,5 %, 2 %, 5 %) în raport cu OpenCV.

**2.2 Metrici urmărite**

* **Timp de execuție**
* **Număr de puncte** păstrate
* **Diferența** de puncte între cele două implementări

**2.3 Rezultate detaliate**

|  |  |  |  |
| --- | --- | --- | --- |
| **Epsilon** | **Implementare proprie** | **OpenCV** | **Diferență** |
| **0,5 %** | 28 ms · 2183 puncte | 3 ms · 2086 puncte | 97 puncte |
| **2 %** | 13 ms · 747 puncte | 3 ms · 606 puncte | 141 puncte |
| **5 %** | 9 ms · 470 puncte | 3 ms · 255 puncte | 215 puncte |

**2.4 Analiză**

* Timpul de execuție scade pe măsură ce **epsilon** crește, în timp ce OpenCV rămâne constant la ~3 ms.
* Implementarea noastră **păstrează mai multe puncte** (simplificare mai conservatoare), diferența crescând odată cu epsilon.

**2.5 Concluzie**

Algoritmul RDP propriu oferă **control mai fin** asupra nivelului de simplificare, însă rămâne mai lent și mai conservator decât OpenCV.

**3. Evaluare Generală**

**3.1 Succesuri**

* Ambele implementări produc rezultate **corecte** și **coerente** cu OpenCV.
* Algoritmul RDP propriu permite **ajustarea granulară** a simplificării conturului.

**3.2 Limitări**

* Performanță semnificativ mai slabă decât OpenCV (de la 3× la 7,5× mai lent).
* Simplificare mai puțin agresivă la valori mari ale epsilon‑ului, ceea ce duce la păstrarea unui număr mai mare de puncte.

**3.3 Recomandări pentru îmbunătățire**

* **Optimizarea accesului la memorie** și utilizarea de structuri de date cache‑friendly.
* **Paralelizarea** secțiunilor critice de calcul (GPU, multithreading, SIMD).
* Ajustarea algoritmului RDP pentru o simplificare **mai agresivă** la epsilon ridicat.
* Implementarea unui mecanism de **cache** pentru rezultate intermediare.

**3.4 Concluzie Finală**

Testele confirmă **validitatea** implementărilor proprii. Deși performanța este sub cea a OpenCV, rezultatele sunt suficiente pentru aplicații practice și oferă o bază solidă pentru optimizări ulterioare.

Improvements:

Select two test cases that initially failed and explain how you resolved the issues. Describe the challenging process and any modifications or additions you made to the previous solution. Clearly justify your design decisions throughout this process.