SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM MAROSVÁSÁRHELYI KAR
SZOFTVERFEJLESZTÉS SZAK

Sub-Pixel éldetektálás
Canny algoritmus felhasználásával

2019

Csalóka Csongor

Tartalom

Bevezető

Az éldetektálás a digitális képeken a számítógépes látás egyik legrégebbi problémája és egy nagyon aktív kutatási területet fed le. Az élek a leggyakrabban használt lokális képi sajátosságok, jelentőségük abban rejlik, hogy az élek alkotják a képen látható objektumok kontúrjait, ezért az élkiemelés az első lépés az objektumok behatárolása és meghatározása felé.

Az élek esetén két összefüggő, de mégis világosan elkülönülő fogalomról kell beszélni, a képi élek nem mindig esnek egybe a fizikai élekkel, amelyek a 3D-s tárgyak élei, fizikai határai. A képi élek hirtelen intenzitás-változások (intensity discontinuities), a fizikai élek ezzel ellentétben hirtelen felület-változások (surface discontinuities). Példaként: az árnyékok élei nem esnek egybe a felület-változásokkal, habár fizikai élekből erednek, azok vetületei.

Az éllokalizálás egy vagy több utófeldolgozási lépést tartalmaz, amellyel eltünteti a zajos és az úgynevezett „fantom” éleket. Ezzel vékony, folytonos kontúrokat biztosít és olyan bináris élképeket eredményez, amely jelzi, hogy egy adott képelem élpont-e.

Az éldetektálás az első lépés a legtöbb számítógépes-látás algoritmusban. Tehát egy képen ott található él, ahol hirtelen intenzitásváltozás történik: a felület normalitása változik, a megvilágítás változik, a fényvisszaverő tulajdonság változik. A témával kapcsolatosan számtalan szakirodalom van, és valószínűleg több algoritmus található éldetektálásra, mint más képfeldolgozási problémára.

Az egyik legismertebb módszer a Canny által javasolt módszer; három minőségi mutatót határozott meg egy éldetektálónak (jó detektálást, az él képpontok jó lokalizációját, valamint a szűrők egyedi reakcióját), és megtalált egy optimális szűrőt, hogy egy lépésjel zajként modellezett határt észleljen. Canny javaslata, hogy közelítsék az ideális szűrőt a kép konvolúciójához a Gaussi függvény első deriváltjával, majd ezután elnyomni a pixeleket a nem-maximális gradiensmodulussal a gradiens irányába.

A Canny éldetektáló bővítése érdekében az idő múlásával számost fejlesztés történt. Ezek közül a Devernay által javasolt utófeldolgozást alkalmaztam a projekt elkészítésekor, hogy megkapjam a sub-pixel pontosságot. Tehát az élpont pozícióját a gradiens norma értékei interpolációjának maximumaként becsüljük meg. E módszer előnye, hogy kiváló pontosságot eredményez alacsony számítási költséggel.

A projektben a Sub-Pixel éldetektáló algoritmus futási sebességét szeretném összehasonlítani egy, már CPU-ra megírt kód és az általam elkészített GPU-val gyorsított kód között. Az algoritmus jól párhuzamosítható feladat, így a gyorsulás szembeötlő lehet CPU és GPU között.

Canny algoritmus

A Canny algoritmus a leggyakrabban alkalmazott és a mai napig a leghatékonyabb éldetektor. Ez az algoritmus kezdetben egy Gauss simítást hajt végre a képen, majd közelíti az iránymenti deriváltakat, az ezt követő lépésben ezekből a deriváltakból kiszámítja a gradiensvektor nagyságát, az így kialakult kép nagy intenzitású nyeregein lesz az él. Ezt az algoritmust John F. Canny talált ki, 1986-ban.

A Canny éldetektáló algoritmus az alább kritériumok listáját követi, amelyek javítják az éldetektálás módszereit:

- Jó detekció: a lehetséges legtöbb élet találja meg.
- Jó lokalizáció: a megtalált él pontjai az él közepén helyezkedjenek el.
- Kevés hamis találat: egy élet csak egyszer jelezzen, és a megtalált élek között ne legyenek zaj által okozott élek.

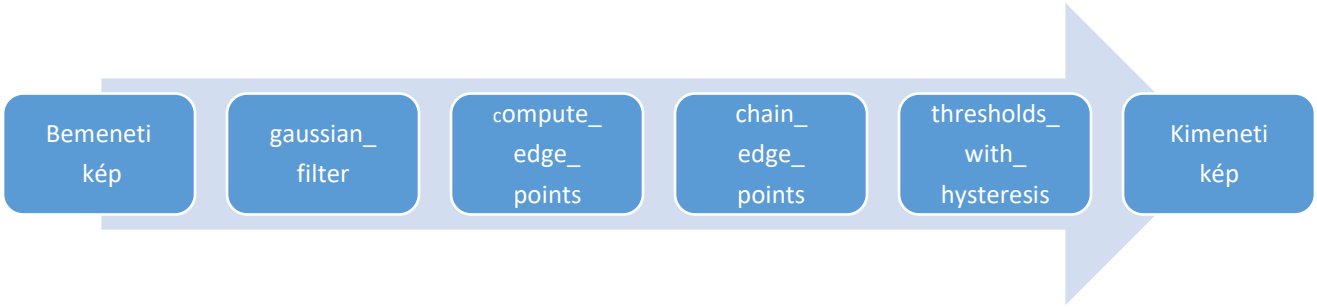
A fenti kritériumokat matematikailag egy hibafüggvényben összefoglalva meghatározhatjuk az optimális szűrőt, amely nagyon közel áll a Gauss függvény első deriváltjához:

$$G'(x) = -\frac{x}{\sqrt{2\pi}\sigma^3}\exp(-\frac{x^2}{2\sigma^2})$$

Ennek segítségével megkeressük az x és y szerinti deriváltakat, majd kiszámoljuk a gradiens erősségét és irányát. Ezt követően megkeressük az élek középpontját, és a többi pontot elhagyjuk, ezzel megvékonyítva a kapott éleket, majd kizárjuk a hamis találatokat hiszterézis küszöböléssel.

Tehát a Canny éldetektáló algoritmus négy különböző lépésre bontható:

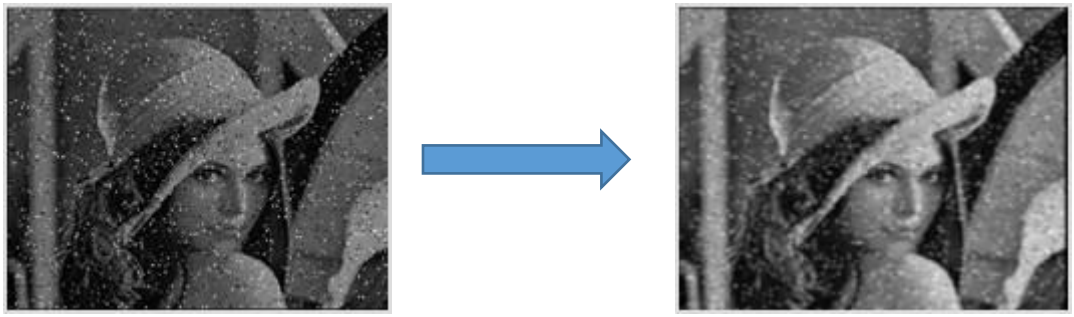
1. Gauss-szűrő felhasználása a kép simításához, a zaj eltávolításához
2. A kép intenzitásának megtalálása
3. Nem-maximumok elhagyása
4. Hiszterézis küszöbölés



Ábra 1. Canny algoritmus fő függvényei

Gauss-szűrő

A Gauss-szűrőt képek elsimításánál alkalmazzák, ilyenkor a képeket Gauss függvénnyel konvolválják, vagyis a kép minden pixeléhez kiszámítják a konvolúció értékét. A kiinduló pont intenzitását lecserélik a korábban már kiszámolt minden pont intenzitásának Gauss függvénnyel vett szorzatára, ahol a Gauss függvény bemenete az adott pont koordinátáinak és a kiinduló pont koordinátáinak a különbsége. Természetesen az egész folyamat legalább két dimenzióra van értelmezve, ami azzal egyenlő, hogy előbb elvégezzük x, majd y irányban.



$$g(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

A Gauss szűrő σ paraméterének hatása:

- σ növekedésével több pixel kerül az átlagolásba
- σ növekedésével a kép jobban el lesz mosva
- σ növekedésével a zaj jobban el lesz távolítva

A kép intenzitásának megtalálása

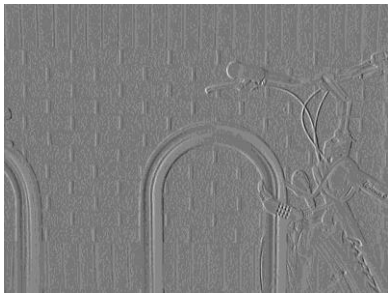
Simítás után a következő lépés a kép intenzitásának megtalálása. A gradiens számítás után körvonalazódik a lehetséges élek erőssége és iránya. A Sobel operátor egyszerű, de hatékony gradiens közelítést tesz lehetővé, az iránymenti deriváltakat közelíti véges differenciával, melyekből a gradiens vektor normája és szöge számítható ki. Az iránymenti deriváltak számítási algoritmusa irányonként egy 3x3-as mátrixszal való konvolúcióból áll. A végeredmény ekvivalens egy közelítő Gauss simítás utáni véges differencia számítással. A konvolúciók szeparálhatók két darab, három hosszú vektorral való konvolúcióra, melynek köszönhetően az algoritmus igen gyors lesz. A módszer hátránya, hogy nem invariáns a forgatásra. Ebből a szempontból jobb

megoldást jelenhet például a Scharr operátor, mely a Gauss simítás pontosabb közelítésével ér el jobb forgatás invarianciát.

Ábra 2. Szürke kép

Ábra 3. Normalizált gradiens nagysága

Ábra 4. Normalizált X-gradiens



Ábra 5. Normalizált Y-gradiens

Nem-maximumok elhagyása

A nem-maximumok elhagyásának gondolata onnan ered, hogy élek ott vannak, ahol a gradiensnek lokális maximuma van, azonban különböző éleknél eltérő lehet az az árnyalatváltozás, amelynek az él megjelenése köszönhető, így szükségünk lesz egy módszerre, ami ezen segít. A módszer lényege az, hogy minden pixelre közelítjük a kiszámolt gradiens irányát: 4 irány-vízszintes, függőleges és a két diagonális valamelyikét. Ezután az ez iránymenti két szomszédjával összehasonlítjuk a gradiens erősségeket, majd, ha nem volt maximum, akkor nem az lesz az él közepe, tehát elvetjük; ha az volt a maximum, akkor megtartjuk. Ennek a módszernek a másik megközelítése, amikor nem a kiszámolt irányt közelítjük, hanem a képet folytonosnak tekintve, interpoláció segítségével állapítjuk meg, hogy milyen értékekkel kell az összehasonlítást végezni.

Ábra 6. 1-Eredeti 2-Gradiens magnitúdó 3-Nem-max. gradiens elnyomása

Hiszterézis küszöbölés

Az optimális küszöbérték megtalálása problémát okozott. Ha túl nagy a küszöb, rengeteg élet elveszthetünk, vagy szakadásokat okozhatunk, ha túl kicsi értéket választunk küszöbnek, akkor pedig sok olyan élet is találhatunk, amely eredetileg nem is volt él. Ezt a problémát próbálja orvosolni a hiszterézis küszöbölés úgy, hogy két küszöböt is megadunk: egy nagyobb és egy kisebb értéket. Jelölje ezeket rendre H és L. Minden pont mely esetén a gradiens erősség kisebb, mint L, azt elvetjük, és minden pont melyre a gradiens erősség nagyobb, mint H, azt megtartjuk, ezeket nevezzük el erős élpontoknak. Azon pontokat, amelyek a kettő közé esnek, nevezzük gyenge élpontoknak. Végeredményben azokat a pontokat tartjuk meg, amelyek vagy erős élpontok voltak, vagy olyan gyenge élpontok, hogy valamilyen úton (szomszédos gyenge élpontok sorozatán) el tudunk jutni egy erős élpontba (ez természetesen nem jelentheti azt, hogy vastagodott egy él, mert ezt már a nem-maximumok elhagyása után alkalmaztuk).

Eredeti kép

Magas küszöb
(erős élek)

Alacsony küszöb
(gyenge élek)

Hiszterézis küszöb

Devernay sub-pixel korrekció

A Canny algoritmus megtalálja az élpixeleket, más szóval az élpontokat kinyerjük a pixel rács pontosságáig. Néhány alkalmazásnál szükség lehet arra, hogy a perempontok helyzete finom pontossággal legyen meghatározva. E célból a Deverney egy nagyon elegáns kiegészítőt javasol a Canny algoritmushoz. A kiegészítő sub-pixel pontosságú élpontokat állít elő, az élpont pozíciója a gradiens norma interpolációjának maximális értékeként finomítva van. A korrekció ugyanolyan egyszerű, mint a gradiens norma kvadratikus interpolációjának kiszámítása három szomszédos pozíció között, a gradiens irányában.

A Devernay korrekció szépsége, hogy egy kiváló sub-pixel pontosság érhető el egyszerű és gyors számítással, amely a Canny operátor által már megkapott értékeket tartalmazza.

Élpont lánc

Minden élpont a többitől függetlenül számítódik ki, és az ugyanazon szegmenshez tartozókat láncolni kell, hogy a kép kontúrjai megfelelő görbéket képezzenek. Mivel a Devernay metódus sub-pixel pontosságot biztosít a pontok számára, láncoláshoz egy egyszerű eljárást lehet használni: az élpontot a legközelebbi másik élponthoz csatlakoztatjuk, nem messzebb, mint egy bizonyos tűrés.

Az algoritmus implementálása CPU-n

Algoritmus 1. *Canny/ Devernay éldetektor*

```
gaussian_filter                    /* Algoritmus 2 */
compute_gradient                   /* Algoritmus 3 */
compute_edge_points                /* Algoritmus 4 */
chain_edge_points                  /* Algoritmus 5 */
thresholds_with_hysteresis         /* Algoritmus 6 */
```

A processzora való implementált kódot online github-on találtam, amelyet néhány módosítás után futtathatóvá tettem. Ez a projekt a fent leírt algoritmusok segítségével valósítja meg a sub-pixel éldetektort. A projektben segítséget nyújtott az OpenCV (Open Source Computer Vision Library), mely az egyik leginkább felkapott, nyílt forráskódú képfeldolgozó könyvtár, ezáltal lényegesen leegyszerűsödnek a képfeldolgozás folyamatai.

Az 1. algoritmusban Canny/ Devernay éldetektor általános folyamatát öt lépésben valósítja meg: Gauss szűrőt alkalmaz a képen, kiszámolja a kép gradienst, kiszámolja a sub-pixel élpontokat, láncolja az éleket, és alkalmazza a Canny küszöbértéket hiszterézissel. A módszer eredménye a láncolt élpontok listája lesz, amelyek sub-pixel pontosságúak lesznek.

A 2. algoritmus gaussian_filter szűrőt alkalmaz a képre, a Gaussi kernel sigma paraméter felhasználásával. A Gauss-szűrőt a képek élsimításához alkalmazzuk, ilyenkor a képet Gauss függvénnyel konvoluáljuk. Tehát a kiinduló pont intenzitását lecserélik a korábban már kiszámolt minden pont intenzitásának gauss függvénnyel vett szorzatára, ahol a gauss függvény bemenete az adott pont koordinátáinak és a kiinduló pont koordinátáinak a különbsége.

A kép gradiens kiszámítását a 3. algoritmus tartalmazza, megadva a kép méretét és modulusát, itt történik a kép gradiens összetevőinek a kiszámítása központosított különbségekkel.

A 4. algoritmus a compute_edge_points, úgymond a teljes algoritmus legfőbb pontja, itt kerülnek kiszámításra a sub-pixel élpontok. Ha a kép gradiens modulusa egy adott pixelnél nagyobb, mint a jobbra és balra eső gradiens modulusa, akkor vízszintes élpontnak nevezzük. A Devernay korrekció jól működik abban az esetben, ha csak függőleges vagy csak vízszintes irányba interpolálják. Ahhoz, hogy egy pixel élpont lehessen, vízszintesen vagy függőlegesen lokális maximumnak kell lennie. A gradiens orientációjától függően, ha a gradiens X komponense nagyobb, mint az Y komponens, az azt jelenti, hogy a gradiens nagyjából vízszintes. Emellett ennek a gradiensnek vízszintes maximum szélső pontnak kell lennie.

A láncolási eljárást az 5. algoritmus tartalmazza, a listában lévő minden élpont kiértékelésre kerül. Az élpontok a pixelekhez kapcsolódnak, amely a gradiens modulus helyi maximuma, függőlegesen vagy vízszintesen. A szomszédok halmaza az 5x5-ös körzetben lévő pixelekhez kapcsolódó élpontok, vagyis azok a pixelek, amelyek maximális koordináta különbsége nem nagyobb, mint 2. Az algoritmus egy új lánc létrehozása előtt ellenőrzi, hogy a cél már láncolt-e és ebben az esetben, hogy az jobb vagy sem, mint a javasolt.

Az 6. algoritmus hiszterézissel meghatározza a küszöbértéket, itt minden élponthoz egy flag-et definiálunk, amely jelzi, hogy a pixel volt-e validálva vagy sem; ezek kezdetben mind FALSE-ra vannak állítva. Ezután végig iterálunk az élpontokon. Ahol a kép gradiens modulusa nagyobb vagy egyenlő a magas küszöbértékkel, ott validdá nyilvánítjuk. Ha minden görbén végig iteráltunk, akkor azokat a pontokat, amelyek nem érvényesek, eltávolítjuk a kimeneti listából.

Implementálás GPU-n és eredmények

A CPU és GPU számítási kapacitása közötti különbséget úgy tudom mérni, hogy ugyanazon feladatot adom ki mindkettőnek, és számolom, hogy melyik mennyi idő alatt végzi el az adott számítást.

Az eltelt időt a CPU-n a high_resolution_clock segítségével mérem, mégpedig a függvények lefuttatása előtt meghatározom a pontos időt (t1), valamit ezt teszem a függvények lefuttatása után (t2) is. A függvények lefuttatása előtt és után feljegyzet időpontok különbsége (t2-t1) adja a függvények által felvett időt.

GPU-n pedig a CUDA Eventeket használom, mert ezáltal pontosabb időt tudok mérni, mintha CPU-n mérném. A mérés hasonlóan müködik, mint a CPU-n.

A hardver, amelyen a projektet futtatom:

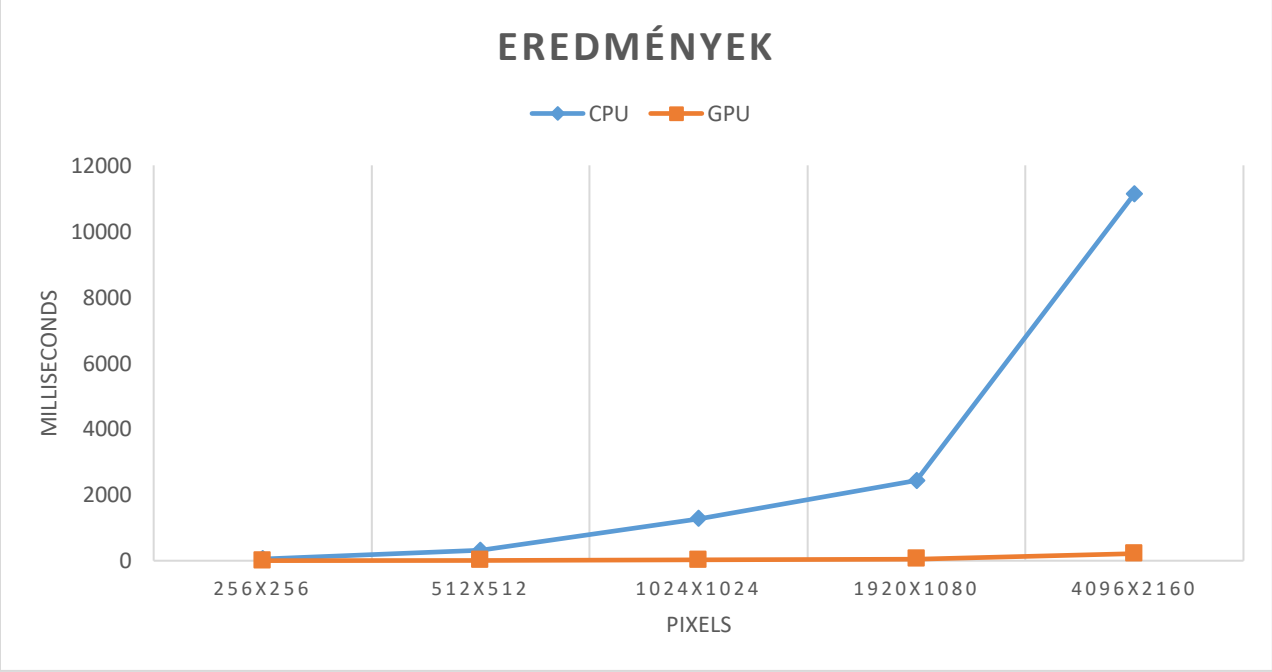
- CPU - Intel® Core™ i7-4720HQ (2.6- 3.6 GHz, 4 mag, 8 szál)
- GPU – Nvidia GeForce GTX 1070 (1920 CUDA mag, 1506/1683 MHz órajel, 8192 MB GDDR5, 256 GB/s memória sávszélesség)

A CPU és a GPU közti gyorsulást úgy mérem meg, hogy azonos képet adok mindkét programnak. Különböző méretű képek kerülnek feldolgozásra 256x256, 512x512, 1024x1024, 1920x1080 (FHD), 4096x2160 (4K). Minden mérés végső eredménye egy átlag lesz, amelyet úgy állapítok meg, hogy az azonos képet egymás után ötször futtatom le. Az eredményt a következő táblázatban rögzítettem.

	Sigma = 1.5, LowThreshold=5, HighThreshold= 15										
	256x256		512x512		1024x1024		1920x1080(FHD)		4096x2160(4K)		
	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	
IMAGE	52.837	1.705	311.278	7.280	1284.29	25.314	2442.26	48.173	11192.1	208.47	milliseconds
	51.092	2.100	308.309	7.259	1273.89	25.187	2452.72	48.914	11170.4	207.39	
	50.581	1.763	310.001	7.255	1265.24	25.207	2451.17	48.626	11096.6	207.99	
	50.844	2.090	307.255	7.281	1261.05	25.013	2437.43	50.005	11072.3	242.51	
	50.956	1.708	310.877	7.229	1270.15	25.083	2394.53	48.182	11153.6	208.69	
Average	51.262	1.873	309.544	7.261	1270.92	25.161	2435.62	48.78	11137	215.01	ms
Speedup		27.3		42.6		50.5		50.2		51.7	times

A táblázatban látható eredmények alapján megállapítható, hogy a gyorsulás szemmel látható eredményt ért el. A GPU nagyságrendekkel gyorsabban futott le bármilyen felbontás esetén, tehát a felgyorsítás sikeres volt.

- 256x256px a gyorsulás 27.3x.
- 512x512px a gyorsulás 42.6x.
- 1024x1024px a gyorsulás 50.5x.
- 1920x1080px a gyorsulás 50.2x.
- 4096x2160px a gyorsulás 51.7x.



Összegzés és továbbfejlesztési lehetőségek

Összességében, a fent említett kutatás egy éldetektor sub-pixel pontossággal történő megvalósítása, e módszer a jól ismert, klasszikus Canny és Devernay algoritmusokon alapul. A CPU-hoz viszonyított gyorsulást vizsgáltam, és megállapítható, hogy sikeresen, szemmel jól látható gyorsulást sikerült elérni a fejlesztés során. A képfeldolgozás egy jól párhuzamosítható feladat; éppen ez a GPU erőssége, ha ugyanazt a munkát többször is el kell végezni. A GPU által akár 51-szeres gyorsulást is elértem a CPU-hoz képest. Tehát, kijelenthetjük azt, hogy a GPU-n történő programozás egyre inkább tért fog hódítani, hiszen nagy gyorsulásokat lehet elérni általa.

A fenti kutatás során akadt olyan CPU-s algoritmus, amit nem sikerült párhuzamosítanom, ezért a további fejlesztésekkel szeretnék az elkövetkezőkben foglalkozni. A további bővítések által a most elért gyorsulásnál is nagyobb várható. Emellett, az esetlegesen felmerülő hibák orvoslását is a továbbfejlesztési lehetőségek közé sorolnám.

Bibliográfia

1. Rafael Grompone von Giol, Gregory Randall – A Sub-Pixel Edge Detector: an Implementation of the Canny/ Devernay Algorithm, 2017.
2. Peter Rocket – The Accuracy of Sub-Pixel Localisation in the Canny Edge Detector.
3. Edge detection - https://en.wikipedia.org/wiki/Edge_detection
4. Canny edge detector - https://en.wikipedia.org/wiki/Canny_edge_detector
5. Mark Harris - How to Implement Performance Metrics in CUDA C/C++ - https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc/
6. CPU Sub-Pixel edge detector from GitHub - https://github.com/fcqing/sub-pixel-edge-detect
7. Éldetektálás – Kató Zoltán - http://www.inf.u-szeged.hu/~kato/teaching/DigitalisKepfeldolgozasTG/06-EdgeDetection.pdf
8. NVIDIA Developer Blog - https://devblogs.nvidia.com/