Poisson Image Editing

Prosjektoppgave i IMT3881 Vitenskapelig programmering

Våren 2020

1 Metoden i generelle trekk

En rekke problemer i bildebehandling kan løses med en teknikk som kalles «Poisson Image Editing» [1]. Metoden går i korthet ut på at man representerer bildet man ønsker å komme frem til som en funksjon $u : \Omega \to C$, der $\Omega \subset \mathbb{R}^2$ er det rektangulære området hvor bildet er definert, og C er fargerommet, vanligvis $C = [0, 1]$ for gråtonebilder og $C = [0, 1]^3$ for fargebilder. Bildet $u(x, y)$ fremkommer som en løsning av Poisson-ligningen

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \equiv \nabla^2 u = h,$$

der randverdier på $\partial\Omega$ og funksjonen $h : \Omega \to \mathbb{R}^{\dim(C)}$ spesifiseres avhengig av hvilket problem som skal løses.

En måte å løse Poisson-ligningen på er å iterere seg frem til løsningen vha. såkalt gradientnedstigning («gradient descent»). I praksis gjøres dette ved å innføre en kunstig tidsparameter og la løsningen utvikle seg mot konvergens:

$$\frac{\partial u}{\partial t} = \nabla^2 u - h. \tag{1}$$

Når man velger denne fremgangsmåten, må man også velge en initialverdi for bildet, $u(x, y, 0) = u_0(x, y)$. Randverdiene er vanligvis av Dirichlet- eller Neumann-typen.

To diskrete numeriske skjemaer for (1) kan finnes ved henholdsvis eksplisitt og implisitt tidsintegrasjon og sentrerte differanser for de spatielle deriverte:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \frac{1}{\Delta x^2}(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 4u_{i,j}^{n}) - h_{i,j}, \tag{2}$$

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \frac{1}{\Delta x^2}(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1} - 4u_{i,j}^{n+1}) - h_{i,j}. \tag{3}$$

Det eksplisitte skjemaet (2) er noenlunde rett frem å implementere, mens det implisitte (3) er noe mer krevende. Sistnevnte løses enklest ved å skrive det på matriseform og bruke rutiner for glisne matriser i implementasjonen. Diricletbetingelser på randen implementeres ved direkte innsetting, mens Neumannbetingelser kan implementeres på forskjellige måter, jf. Arbeidskrav 6.

2 Anvendelser

Her følger en kort beskrivelse av noen av anvendelsene av «Poisson Image Editing».

2.1 Glatting

Hvis vi tar utgangspunkt i et originalbilde $u_0(x, y)$, kan man implementere glatting («blurring») av bildet ved å iterere ligning (1) med $h = 0$ i hele Ω. Bildet vil da bli stadig glattere (mer uskarpt) med tiden t. Som randverdier kan man enten bruke Diriclet med $u(x, y, t) = u_0(x, y)$ på $\partial\Omega$ (gir litt skarpere kant enn strengt tatt nødvendig), eller, bedre, Neumann med $\partial u/\partial n = 0$ på $\partial\Omega$ (symmetri).

2.2 Inpainting

Hvis vi enten mangler noe informasjon i et bilde $u_0(x, y)$, eller ønsker å fjerne noe fra det (støy, tekst som er lagt oppå el.l.), kan vi gjøre dette ved å fylle inn («inpaint») informasjon i gjeldende område basert på informasjonen rundt området. Hvis $\Omega_i \subset \Omega$ er området som skal fylles inn, kan dette gjøres ved å sette $h = 0$ i Ω_i og løse ligning (1) i Ω_i med Dirichlet-betingelsen $u(x, y, t) = u_0(x, y)$ på $\partial\Omega_i$.

For å implementere dette er det hensiktsmessig å innføre en maske i form av en boolsk array som er sann for alle pixler som er innenfor Ω_i og usann for alle som er utenfor. En slik array kan da benyttes som index for et «view»[1] av bildet, og gjøre operasjoner bare på pixlene innenfor eller utenfor. F.eks. kan alle pixlene utenfor Ω_i settes lik verdiene fra originalbildet ved å skrive `u[~omega_i] = u_0[~omega_i]`, der `omega_i` er den boolske arrayen (masken).

Dersom masken berører kanten av bildet, trenger vi en randbetingelse på $\partial\Omega$ også. Dette kan da gjøres på samme måte som for glatting (se over).

2.3 Kontrastforsterkning

Jo større den lokale kontrasten i et bilde u er, desto større er gradienten til bildet, ∇u. For å finne en mer kontrastert utgave av originalbildet u_0, kan vi altså forsøke finne et bilde som har samme gradient som u_0, men forsterket med en konstant $k > 1$. Dette kan gjøres ved å sette $h = k\nabla^2 u_0$ inn i (1)

[1] `https://docs.scipy.org/doc/numpy/user/basics.indexing.html`

og løse for u. Hensiktsmessige randverdier er enten Dirichlet, $u(x,y,t) = u_0(x,y)$ eller, bedre, Neumann, $\partial u/\partial n = k\partial u_0/\partial n$ på $\partial\Omega$. Merk at iterering av (1) med $k > 1$ fort kan føre til løsninger med $u > 1$ eller $u < 0$, altså utenfor det tilgjengelige fargeområdet. Det må derfor innføres som en føring at $u \in [0,1]$. Dette kan i praksis gjøres i koden ved å klippe verdiene for u til intervallet i slutten av hver iterasjon.

En mer avansert form for kontrastforsterkning kan vi lage som beskrevet i [2] ved å innføre en ikkelineær funksjon av gradienten, f.eks. $g = f(\nabla u_0)$, og så la $h = \nabla \cdot g = \nabla \cdot (f(\nabla u_0))$ i ligning (1). Randverdier og føringer blir som beskrevet over.

2.4 Demosaicing

Bildesensoren i et digitalkamera er egentlig monokrom, og kan bare måle mengden lys som faller inn på den i hver pixel. For å kunne lage fargebilder, legger man en mosaikk av fargefiltere over den, slik at i hver pixel måles i praksis kun én av fargekanalene, f.eks. R, G og B. Og ut av sensoren kommer det altså en gråtonemosaikk.

En slik gråtonemosaikk kan man simulere i Python ved å ta utgangspunkt i et fargebilde `u` representert ved en $M \times N \times 3$ `numpy array`. Gråtonemosaikken kan lages som følger:

```
mosaic = np.zeros(u.shape[:2])           # Alloker plass
mosaic[ ::2,  ::2] = u[ ::2,  ::2, 0]   # R-kanal
mosaic[1::2,  ::2] = u[1::2,  ::2, 1]   # G-kanal
mosaic[ ::2, 1::2] = u[ ::2, 1::2, 1]   # G-kanal
mosaic[1::2, 1::2] = u[1::2, 1::2, 2]   # B-kanal
```

Oppgaven til en demosaicing-algoritme er å rekonstruere et fargebilde ut av en slik gråtonemosaikk. Én måte å gjøre dette på, er å først flytte informasjonen som finnes i mosaikken over i de rette kanalene i et fargebilde, for deretter å «inpaint-e» den manglende informasjonen vha. inpaintingsmetoden beskrevet over. Det må da altså lages en Ω_i for hver kanal som definerer pixlene som skal fylles inn.

2.5 Sømløs kloning

Noen ganger ønsker man av ymse grunner å kunne flytte en del av et bilde inn i et annet bilde på en slik måte at det ikke blir synlige overganger der objektet er limt inn. Dette kalles gjerne sømløs kloning. Hvis vi kaller originalbildet det skal klones inn i for u_0, og bildet som det skal klones inn fra for u_1, kan dette formuleres som et Poisson-problem: Finn u slik at $u = u_0$ for $x \notin \Omega_i$ og $\nabla^2 u = \nabla^2 u_1$ i Ω_i. Dette kan altså gjøres ved å sette $h = \nabla^2 u_1$ og løse (1) i Ω_i med Diriclet-betingelsen $u = u_0$ på $\partial\Omega_i$. Også her må man huske å klippe slik at $u \in [0,1]$.

For implementasjonen i Pyton bør man også merke seg at Ω_i ikke nødvendigvis behøver å befinne seg på samme sted i u_0 og u_1. Dette kan løses ved bruk av «views» på `numpy array`-ene.

2.6 Konvertering av fargebilder til gråtone

Den vanligste måten å konvertere fargebilder til gråtone er ved å ta et (veiet) gjennomsnitt av R, G og B-kanalene. Når fargebilder konverteres til gråtonebilder på denne måten, kan det lett skje at noe av informasjonen i bildet forsvinner. Særlig gjelder dette der det er detaljinformasjon (teksturer, kanter etc.) mellom elementer som i hovedsak skiller seg i fargetone og/eller metning, men som i hovedsak har samme lyshet.

En litt mer sofistikert teknikk går ut på å søke å konstruere et gråtonebilde som har så lik lokal variasjon som det originale fargebildet som mulig. Dette kan gjøres ved å konstruere en ny gradient g gitt med lengde $||\nabla u_0||/\sqrt{3}$ (hvorfor $\sqrt{3}$?) og retning som $\nabla(u_{0R} + u_{0G} + u_{0B})$, og så la $h = \nabla \cdot g$ og løse ligning (1) for u i hele Ω. Her vil det være flere muligheter for spesifikasjon av randbetingelsene. En naturlig initialverdi vil være gjennomsnittet av de tre fargekanalene i originalbildet.

En enda mer sofisikert teknikk for definisjon av h er ved bruk den såkalte strukturtensoren til fargebildet, se [3].

2.7 Rekonstruksjon og visualisering av HDR-bilder

Scener i dagslys har ofte en ekstremt høy dynamikk, det vil si forskjell mellom mørkeste og lyseste punkt i scenen som skal avbildes. Ofte er denne dynamikken så høy at selv en god bildesensor ikke klarer å registrere hele dynamikken, og vi får bilder som samtidig er både over- og undereksponert, altså bilder der noen områder er helt svarte, og andre er helt hvite. En teknikk for å unngå dette for statiske scener, er å ta flere bilder (med kameraet på stativ) med ulik eksponeringstid, og så sette dem sammen til et HDR-bilde («High Dynamic Range») i ettertid. Dette gjøres ved at man estimerer kamera-respons-kurven samtidig som man estimerer den faktiske lysheten i scenen som et stort, sammensatt minste-kvadraters problem, se [4].

Når man så har fått et slik HDR-bilde støter man på et nytt problem, nemlig at de fleste enheter for visning av bilder (skjermer, projektorer, skrivere etc.) heller ikke har stor nok dynamikk til å vise dem frem. Så dynamikken i bildet må altså komprimeres igjen før det kan vises og lagres i vanlige 8-bits-formater. Dette kan gjøres på akkurat motsatt måte av kontrastforsterkningen vi innførte i andre del av avsnitt 2.3 med et passende valg av gradient-komprimeringsfunksjon, se [2]

2.8 Anonymisering av bilder med ansikter

Noen ganger trenger man å anonymisere personene i et bilde før det vises frem offentlig. En måte å gjøre dette på, er å gjøre ansiktene uskarpe mens resten av bildet beholdes skarpt. Hvis man har en maske Ω_i som beskriver områdene i bildene som inneholder ansikter, kan dette gjøres ved å løse ligning (1) med $h = 0$ i Ω_i og Dirichlet-betingelsen $u = u_0$ på $\partial\Omega_i$. Utfordringen er å finne masken. Her kan kanskje OpenCV[2] være til hjelp.

2.9 Kantbevarende glatting

Noen ganger ønsker man å glatte et bilde uten å gjøre kantene i bildet uskarpe. Dette kan gjøres ved å innføre en posisjonsavhengig diffusjonsparameter $D : \Omega \to [0, 1]$, der D = 0 gir null diffusjon/glatting og $D = 1$ gir full glatting. Diffusjonsparameteren kan f.eks. beregnes fra gradienten til bildet som

$$D(x, y) = \frac{1}{1 + \kappa ||\nabla u_0(x, y)||^2}\,, \tag{4}$$

med κ som en passende valgt konstant. Diffusjonsligningen (1) endres da til

$$\frac{\partial u}{\partial t} = \nabla \cdot (D\nabla u), \tag{5}$$

og løses for Ω med f.eks. Dirichletbetingelsen $u(x, y) = u_0(x, y)$ på $\partial\Omega$. Merk at det da må lages egne numeriske skjemaer for denne modifiserte ligningen. Merk også at hvis $D = 1$ overalt, vil (5) bli identisk med (1) med $h = 0$.

Denne utvidede modellen kan også på forskjellig vis brukes til å forbedre oppførselen av flere av de tidligere anvendelsene, så her åpnes et hav av nye muligheter. . .

3 Oppgave

3.1 Gjennomføring

Prosjektet gjøres i grupper à 1–4 studenter. Én på gruppen lager en privat fork GitLab-repoet og inkluder din de andre på gruppen som «developer»s. La både kode og rapport bo i repoet. Gjør hyppige og små nok commits til repoet slik at det blir mulig å følge utviklingen av prosjektet i ettertid, også når man ikke har en fungerende løsning (bruk gjerne «branches»). Det vil altså i praksis si flere commits enn man strengt tatt ville gjort i en realistisk utviklingssituasjon.

[2] `https://opencv.org/`

3.2 Minimumsløsning

Implementer det eksplisitte skjemaet (2) og anvendelsene glatting (avsnitt 2.1), inpainting (avsnitt 2.2) og kontrastforsterkning for gråtonebilder (avsnitt 2.3) beskrevet over. Beskriv problemstillingen, løsningen og resultater i form av eksempelbilder i en velformet rapport. Rapporten skal inneholde en lenke til det forkede repoet.

3.3 Utvidelser

Det er ubegrenset med muligheter for å utvide besvarelsen og dermed gjøre den bedre:

- Implementer løsningen også for fargebilder (enkel liten utvidelse).
- Implementer de øvrige anvendelsene beskrevet i avsnitt 2.4–2.9 (de er stort sett beskrevet i rekkefølge av økende kompleksitet).
- Implementer det implisitte numeriske skjemaet (3) vha. glisne matriser («sparse matrices»)[3] og prøv det for alle de implementerte anvendelsene.
- Lag en applikasjon med grafisk brukergrensesnitt som gir brukeren mulighet til å utføre alle de implementerte operasjonene interaktivt. Bruk f.eks. PyQt5.[4]

3.4 Vurderingskriterier

Ved vurderingen av prosjektoppgaven vil det bli lagt vekt på

- hvilke metoder som er implementert
  - gråtonebilder/fargebilder
  - eksplisitt/implisitt etc.
  - typer randverdier for $\partial\Omega$ og $\partial\Omega_i$
- hvilke anvendelser som er implementert
  - glatting
  - inpainting
  - kontastforsterkning (flere mulige varianter)
  - demosaicing
  - sømløs kloning
  - farge til gråtone-konvertering
  - rekonstruksjon og visualisering av hdr-bilder
  - anonymisering av bilder med ansikter
  - kantbevarende glatting
  - gui-applikasjon

[3] `https://docs.scipy.org/doc/scipy/reference/sparse.html`
[4] `https://www.riverbankcomputing.com/software/pyqt/intro`

- kvalitet på koden, herunder
  - struktur og gjenbruk av kode
  - gjenbrukbarhet i form av moduler
  - dokumentasjon i form av velformede doc-strings
  - variabelnavn
  - automatiserte (enhets)tester med tilhørende rapportering (f.eks. med `coverage`)
- kvalitet på rapporten, herunder
  - struktur
  - språk
  - formler
  - referanser
  - kryssreferanser
  - figurer
  - tabeller
  - kodelistinger
- prosessen (slik den fremkommer av git-historikken)

4 Innlevering

Rapporten innleverers som PDF i Inspera innen fredag 24. april 2020 kl. 1530. Rapporen *skal* inneholde en lenke til GitLab-repoet. GitLab-repoet må minimum få leve til etter at sensuren er gitt (og lenger dersom man skal kunne klage på karakteren).

Referanser

[1] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003.

[2] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 249–256. ACM, 2002.

[3] Ali Alsam and Mark S Drew. Fast colour2grey. In *Color and Imaging Conference*, volume 2008, pages 342–346. Society for Imaging Science and Technology, 2008.

[4] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, pages 369–378, 1997.