En model for programmer
Version 1.0

Henrik Bærbak Christensen
Datalogisk Institut
Aarhus Universitet

Februar 2012

Dette undervisningsmateriale beskriver en model for, hvordan programmer er opbygget.

Materialet er skrevet til brug i det gymnasiale forsøgsfag Informationsteknologi. Seneste version af dette undervisningsmateriale kan findes på `http://www.imhotep.dk`. Tak til Elisabeth Husum for en kritisk gennemgang.

1 Interaktion og opførsel

Et program, der kører på din computer eller din smartphone, er designet til *i samspil med dig at gøre noget for dig*. For eksempel så bruger du et tekstbehandlingsprogram, fordi du kan skrive tekst ind i det, og programmet kan hjælpe med at gemme det, rette stavefejl, sætte det pænt typografisk op osv. Altså er der et *samspil* mellem dig (som taster løs på tastaturet flytter rundt i teksten med musen, vælger menupunkter osv.) og programmet, der *gør noget* for dig (viser teksten på skærmen, markerer stavefejl, printer teksten osv.)

Fordi dette er kernen i de fleste programmer[1], har vi to begreber at slå fast:

[1] Det skal lige tilføjes, at der er mange programmer som *ikke* har interaktion med mennesker, men kun med andre programmer. F.eks. programmer i bankerne til at regne renter ud på dine konti ved årets udgang. Men dem ser vi lige bort fra her.

Interaktion: Interaktion er samspillet mellem dig og programmet, typisk ved hjælp af tastaturet (indsæt tekst, kald funktioner via shortcuts, enter, slettetast osv.) og musen (markér ting, flyt cursor, skalér, osv.)

Opførsel og tilstand: Et program udviser *opførsel*[2], det vil sige, at det har handlinger,som du kan bede det om at blive udføre, og programmet har en *tilstand*, altså det data som det indeholder. Et tekstbehandlingsprograms tilstand vil f.eks. være den tekst som det bearbejder, og typisk opførsel være at kunne gøre tekst fed, ændre det til en overskrift, printe teksten ud, søge efter stavefejl osv.

Softwareudviklere er de professionelle, som skriver programmer, så du forhåbentlig har en god interaktion med et program, der tilbyder den opførsel, du ønsker. Til det formål skaber udviklerne en *model* som basis for programmets opførsel og tilstand, og ofte vil denne model "skinne igennem", fordi interaktionen skal tillade dig at få fat i programmets tilstand og opførsel. Jeg kommer tilbage til en mere præcis forklaring af, hvad en model er for et program, men lad os starte med at lave et forsøg, som viser, hvordan to programmer, som åbenlyst har samme formål, bruger en mere eller mindre avanceret model. Den simple model er let at forstå og kræver en simpel interaktion, mens den mere avancerede model er en del sværere at forstå og interagere med. Til gengæld for den ekstra indsats du skal bruge på at forstå programmets model, får du så et program, som kan meget mere.

2 Et forsøg med tegneprogrammer

Til det her forsøg skal du bruge en Windows baseret computer og to tegneprogrammer. Det ene er standard på enhver Windows computer og hedder Paint. Det andet er et freeware program, der hedder TwistedBrush Open Studio.

TwistedBrush Open Studio er et avanceret tegneprogram, der fås i en freeware-version. Programmet hentes på følgende adresse:

http://www.pixarra.com/download.html

Der er mange typer tegneprogrammer. Nogle er beregnet til at redigere fotografier og andre til at lave tekniske tegninger af for eksempel maskindele eller elektroniske kredsløb. Både Paint og TwistedBrush er tegneprogrammer beregnet til at lave "kunst"to dimensionelle billeder, som er (mere eller mindre) pæne at se på.

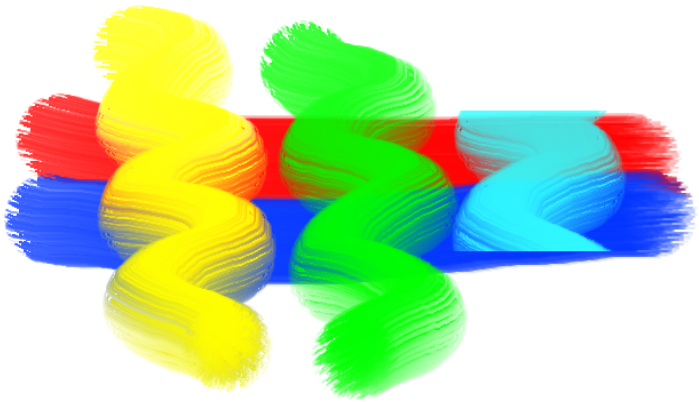
Forsøget går ud på at efterligne en kunstmalers brug af oliemaling til at male noget som ligner billedet nedenfor. Altså:

- en tyk rød streg oliemaling

[2]Softwareudviklere bruger ofte den alternative betegnelse *funktionalitet* for opførsel.

- en tyk blå streg oliemaling
- en tyk gul streg som zigzagger gennem den røde og blå. Når man maler med våd oliemaling, vil farverne naturligt blande sig, så den gule streg vil tvære noget af den røde og blå streg ud. Se specielt på den nederste del af den gule streg, hvor der tydeligt er blandet blåt i.
- en tyk grøn zigzag streg, men hvor maleren har ladet den blå og røde måling tørre først, så der ikke sker nogen opblanding men hvor det blå og røde derimod kan skinne igennem, hvor den grønne maling er tynd. Se igen på den nederste del af stregen hvor der ikke er noget blåt at spore.
- en tyk lysblå zigzag streg, hvor maleren har lagt en maske over, altså klippet et rektangel ud af et stykke karton, lagt det over maleriet, og så malet på det. Dermed er det naturligvis kun der, hvor der er hul i kartonen, at maleriet får en lysblå zigzag streg.

Altså skal vi ende med noget, der ligner billedet herunder.



Forsøg 1: Brug Windows tegneprogram Paint til at efterligne de fem streger. Hvis I har adgang til både Windows XP og Vista/7 så prøv opgaven på alle versioner af Windows.

Forsøg 2: Brug tegneprogram "TwistedBrush " til at efterligne de fem streger.

For hvert forsøg notér om det er muligt at:

- lave brede streger, som ligner oliemaling (struktur af en pensel i strøget, det hvide papir skinner igennem, hvor malinglaget er tyndt)?
- lave den gule streg, så noget af den røde og blå maling bliver blandet op i det gule?
- lave den grønne streg, så der ikke sker opblanding af rød og blå maling, men måske den underliggende maling skinner igennem?
- lave en maske så kun et rektangulært udsnit af lysblå maling kommer på maleriet?

Lav forsøget, inden du bladrer videre!

Her er resultatet af mit eget forsøg.

- Windows XP's Paint program. Det er ikke muligt at lave noget, der ligner overhovedet. Paint kan kun male med tykke streger og giver slet ikke mulighed for opblanding af farver eller masker...

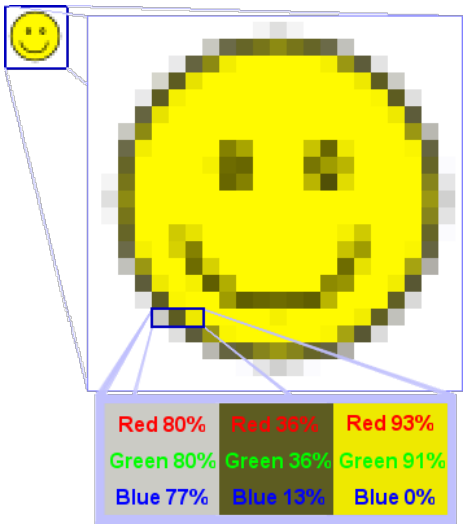
- Windows 7's Paint program. Her går det noget bedre. Man kan vælge et penselværktøj, så der kommer noget, der ligner oliemaling, og man kan også skimte farven nedenunder. Men som du kan se, så er der ingen opblanding mellem farverne, specielt den gule streg viser ingen tegn på at trække noget rød eller blå med og ligner den grønne streg. Endvidere ser malingen underlig ud, der hvor "ziggerne zagger". Den lysblå streg gennem masken har jeg forsøgt lave ved først at tegne den et andet sted, og dernæst klippe et rektangel ud (cut) og derefter flytte det hen over de røde og blå streger (paste). Det er imidlertid ikke lykkedes særlig godt, fordi Paint programmet også har flyttet noget af den hvide baggrund med....

- TwistedBrush. Billedet på side 3 er lavet ved hjælp af TwistedBrush, så det program efterligner efter bedste evne rigtig oliemaling. Det er en smule indviklet, så du er velkommen til at bruge gennemgangen fra afsnit 3.2, som forklarer, hvordan jeg er kommet frem til billedet.

Alle de tre programmer har samme formål: at tegne to dimensionelle billeder, så hvorfor denne forskel? Svaret ligger i, at den model, som softwareudviklerne har bygget ind i programmerne, er forskellige. Windows XP's Paints model er relativ simpel, og det er derfor begrænset, hvad man kan gøre. TwistedBrush har en langt mere avanceret model, som på den ene side er lidt sværere at forstå, men på den anden side tillader brugeren at lave langt mere avancerede billeder.

3 Modeller for tegneprogrammer

Ok, tegneprogrammer har brug for at vise og manipulere to-dimensionelle billeder, så lad os starte med at finde ud af, hvordan et billede vises på en tv- eller computerskærm.



Figur 1: Bitmap af en smiley.

Hvis du tager et forstørrelsesglas og går helt tæt på en computerskærm vil, du opdage, at billedet består af uhyre små lysende firkanter. En sådan lysende firkantet kaldes en *pixel*. En pixel kan have alle mulige farver, men "hele firkanten" har altid én og kun én farve. Pixel'erne er sat op i et to- dimensionelt koordinatsystem, som du kender det fra matematik med en 1. og 2. akse. 1. aksen går altid fra 0 og til det antal pixels, der er i bredden på din skærm; og 2. aksen igen fra 0 til det antal pixels, der er i højden på din skærm. Fx. har et fuld HD TV 1920 pixels i bredden og 1080 pixels i højden, min computerskærm hjemme har 1600x1200 pixels, og i begyndelsen af 1990'erne var 640x400 pixels meget almindeligt.

Så derfor kan en skærm vise et billede ved at farvelægge de enkelte pixels, som det fremgår af figur 1. Et sådan koordinatsystem af pixels kaldes et *raster billede* eller et *bitmap*.

3.1 Model 1: Bitmap og værktøjer

Simple tegneprogrammer som Windows Paint program har en model som vi kan betegne som "bitmap plus værktøjer", hvilket vil sige, at programmet indeholder et bitmap og tilbyder noget opførsel (pakket ind i "værktøjer"), der kan ændre på farveværdierne af de enkelte pixels i bitmappet. Lad os kigge på dem en af gangen.

3.1.1 Bitmap

Udvikleren programmerer informationsteknologisk et **samfund af objekter**, der repræsenterer et bitmap. "Samfund" lyder måske mærkeligt, men ligesom det samfund, vi lever i, kun fungerer hvis vi samarbejder konstruktivt og følger normer og regler, ja så fungerer et programs objektsamfund kun, hvis alle

følger spillereglerne. Så lad os nu kigge på, hvordan vi kan lave et bitmap objektsamfund.

Det mest fundamentale objekt er en pixel. En pixel har en bestemt farve, og vi skal naturligvis kunne ændre, hvilken farve den har. Det additive farveprincip anvendes, det vil sige at alle farver kan blandes af de tre primær farver, rød (R), grøn (G) og blå (B). Altså kan vi gemme farven som tre procenttal, der angiver hvor "tændt" farven er. Hvis vi vil have en helt knaldrød farve så skal vi gemme RGB = (100%, 0%, 0%) det betyder fuld lysstyrke i den røde farve, og slukket for grøn og blå. Hvis man vil have en mørk rød farve, så skruer man ned for den røde farve, f.eks. RGB = (50%, 0%, 0%) På samme måde er knaldblå RGB = (0%, 0%, 100%); hvid er RGB = (100%, 100%, 100%), osv. Se figur 1.

Opgave: Hvordan gemmer man farven sort?

Opgave: Man kan lave en rød mørkere ved at skrue ned for intensiteten af rød, f.eks. fra 100% til 50%, men hvordan laver man en lysere rød? Hint: Prøv at starte farveeditoren i Paint, vælg den helt røde farve og skru så op for lysstyrken og se hvordan (R,G,B) værdierne ændrer sig.

Derfor kan vi lave et objekt, som er en model af en pixel på en computerskærm:

```
objekt: Pixel
  attribut:
     farve: (R, G, B);
  handling:
     ændr-farve-til (R, G, B);
```

Det, som ovenstående beskriver, er et objekt, Pixel, som har en *tilstand*, nemlig dens farve repræsenteret ved (R,G,B) værdierne, og som har en *opførsel*, nemlig en handling ændr-farve-til som kan skifte farven på pixlen.

Så en knaldrød pixel vil have tilstanden [farve: (100%,0%,0%)]. Hvis programmet gerne vil gøre den knaldblå istedet, så skal den udføre handlingen ændr-farve-til(0%,0%,100%) på den pixel.

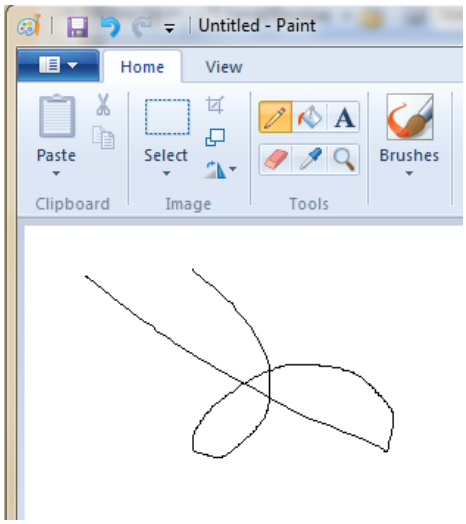
Ok, det var en enkelt pixel, men vi har jo brug for mange af dem. Så vi skal også bruge et bitmap objekt.

```
objekt: Bitmap
  attribut:
     indhold: koordinatsystem (1600,1200) af pixel;
  handling:
     hent-pixel-i-punkt( X, Y );
```

Det ovenstående beskriver, er et objekt, Bitmap, hvis tilstand er et 1600x1200 koordinatsystem af pixels, og som tillader, at man kan få fat i den enkelte pixel f.eks. på koordinatpunkt (100,230) ved hjælp af handlingen hent-pixel-i-punkt(100,230).

Så hvis programmøren vil se farven på pixel i position (133,754), så kan han gøre det i noget denne stil

```
(R,G,B) = bitmap.hent-pixel-i-punkt(133,754).farve;
```



Figur 2: Blyantsværktøjet i brug.

Ovenstående læses sådan her "farve værdierne (R,G,B) tildeles værdierne af attributten 'farve' i den pixel, som er resultatet af at udføre handlingen hent-pixel-i-punkt(133,754) på objektet 'bitmap'."

Generelt bruger man "punktum" i mange programmeringssprog til at udtrykke, at et bestemt objekt skal udføre en handling. Hjemme kan jeg bede min søn, Magnus, om at tømme opvaskemaskinen ved at sige "Magnus, du skal tømme opvaskemaskinen." I programmeringssprog, og i denne note, har vi på samme måde brug for at kunne udtrykke, at et objekt skal udføre en handling. Derfor vil min bøn til Magnus se sådan ud i et programmeringssprog:

```
Magnus.tøm-opvaskemaskine;
```

Opgave: Tegn nogle streger i Paint og zoom nu ind på stregerne til 400%. Nu kan du tydeligt se de enkelte pixels i det bitmap, der er tegnet i.

3.1.2 Værktøjer

Tegneprogrammer har *værktøjer* til at tegne med. Disse værktøjer er fundamentet for interaktionen med tegneprogrammet. Det mest simple i Windows Paint er værktøjet "blyant". Det bruger man til at tegne frihåndsstreger med, som det ses på figur 2.

Blyantsværktøjet er også et objekt med tilstand og handlinger.

```
objekt: Blyantsværktøj
  attribut:
    farve: (R,G,B);
  handling:
    mus-knap-ned( X, Y );
    mus-flyt-til( X, Y );
    mus-knap-op( X, Y );
```

Tilstanden er en farve, nemlig den som blyanten skal tegne med. Handlinger er der tre af, som er lidt specielle, idet de bliver kaldt, når du bruger musen. Lad os sige, at du vil tegne en skrå frihåndsstreg i dit Paint program. Det gør du ved at flytte musen til det ene hjørne (lad os sige koordinaterne (23,24)), trykke museknappen ned og så flytte musen (med knappen nedtrykket) til det andet hjørne (f.eks. koordinaterne (348, 351)), hvorefter du så slipper museknappen. Det Paint vil gøre er, at omsætte museklikkene (altså din interaktion med programmet) til værktøjets handlinger, så ovenstående bliver noget i denne stil

```
værktøjsobjekt.mus–knap–ned(23,24);
værktøjsobjekt.mus–flyt–til(25,28);
værktøjsobjekt.mus–flyt–til(27,29);
værktøjsobjekt.mus–flyt–til(32,31);
(og så videre...)
værktøjsobjekt.mus–knap–op(348,351);
```

Ok, så Paint oversætter dine musebevægelser til handlinger på blyantsværktøjet, men hvordan får vi så tegnet en streg? Det sker, fordi blyantsværktøjets handlinger samarbejder med bitmap objektet! Så programmøren har lavet handlingen "mus-flyt-til(X,Y)" så den ændrer farven på de pixels, der ligger på koordinaten (X,Y):

```
Blyantsværktøj: mus–flyt–til(X,Y) {
  bitmap.hent–pixel–i–punkt(X,Y).
          ændr–farve–til( værktøj.farve );
}
```

Puha, den ser lidt kringlet ud. Hvad står der? Der står: "I blyantsværktøjets "mus-flyt-til" handling sker der følgende: hent pixel på position (X,Y) i bitmappet og udfør dette pixel objekts handling ændr-farve-til[3] og farven, der skal ændres til, er den farve som værktøjets farve attribut indeholder." Altså ved at skifte værktøjets farve attribut til blå så tegner værktøjet for fremtiden med en blå farve.

Opgave: Paint har også nogle værktøjer til at tegne f.eks. rektangler. Hvad tror du dette værktøjs mus-knap-ned, mus-flyt-til og mus-knap-op handlinger gør?

3.1.3 Penselværktøj

Nu kan du måske se, hvordan Windows 7's Paint kan lave noget, som ligner olie- og penselmaling? Programmørerne har lavet et penselværktøj med samme tilstand og handlinger som blyantsværktøjet, men med en anden procedure, for hvad der skal ske i dets mus-flyt-til handling. Denne procedure er meget mere kompliceret, dels fordi den skal hente mange flere pixels i omegnen af (X,Y) (fordi en pensel er meget bredere end bare en pixel), og dels fordi den skal variere farven, den skal tegne med (fordi noget maling kommer i et tyndt lag så selvom værktøjets farve er knaldrød (100%, 0%, 0%), så skal nogle pixels farves i en lysere rød farve, så det ser ud som om, det hvide papir skinner igennem).

[3] Ok, det er lidt mere kompliceret, fordi Paints værktøj jo laver en fuldt optrukket linje, så faktisk tegner den en streg fra sidste (X0,Y0) koordinat og til den nye (X,Y).

Men da Paint er et simpelt program, har programmørerne valgt en simpel procedure/algoritme for, hvordan oliemaling opfører sig. Det er langt mere simpelt at lave en procedure, som smører maling ud i den farve, som værktøjet har, uden at tage hensyn til den farve pixelen i bitmappet allerede har.

Så proceduren i Windows 7's penselværktøj er noget ala

```
Penselværktøj: mus–flyt–til(X,Y) {
  I alle pixels (A,B) i en radius omkring (X,Y) {
     hvis (A,B) er tæt på (X,Y) så {
       bitmap.hent–pixel–i–punkt(A,B).
          ændr–farve–til( værktøj.farve );
     }
     hvis (A,B) er langt fra (X,Y) så {
      lyserefarve = gør–lysere( værktøj.farve );
       bitmap.hent–pixel–i–punkt(A,B).
          ændr–farve–til( lyserefarve );
    }
  }
}
```

som udtrykker, at farven er tyndere/lysere langt væk fra penslens centrum.

Selvom det er en mere kompliceret procedure end blyantsværktøjet, så bemærk, at den ikke tager hensyn til den farve, som pixelen allerede har. Så hvis vi tegner med en gul pensel, vil pixlen blive gul eller lysgul, ligegyldig om den var rød eller blå før.

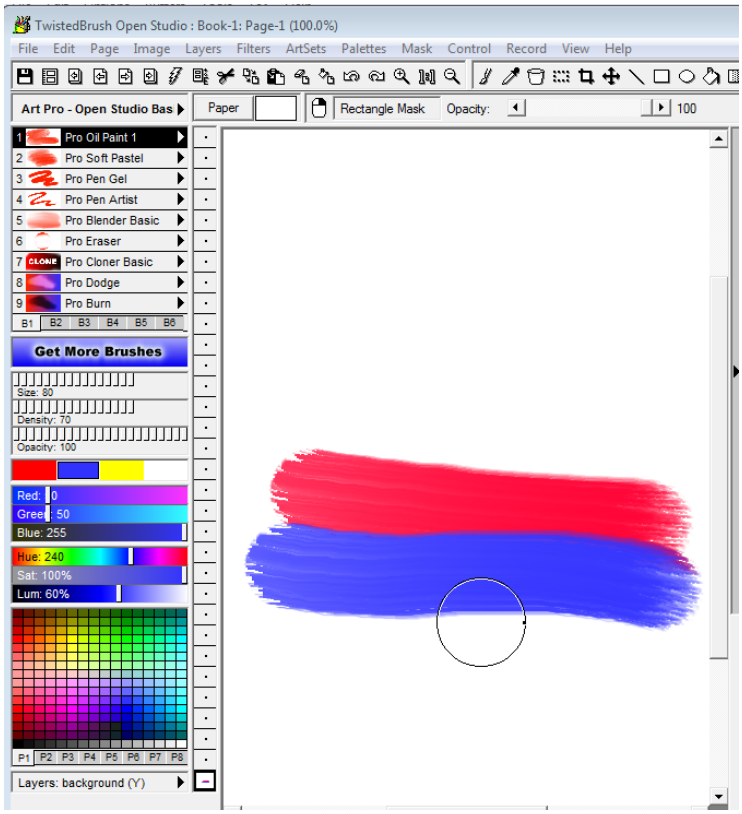
TwistedBrush's penselværktøjs procedure er meget mere kompliceret, fordi den bliver nødt til at regne ud, hvilken farve pixelen skal have, baseret på både penslens farven, hvor langt væk en pixel er fra penslens centrum, hvilken farve pixlen allerede har, og faktisk også i hvilken retning penslen bevæger sig. Når den gule streg tegnes gennem den våde blå streg, skubber man jo noget blå maling med, som blandes op og bliver blå-grøn.

3.1.4 Opsummering af Model 1

Windows Paint programmer bruger bitmap og værktøjs-modellen som grundlag. Det vil sige en model, hvor der er et objekt, der repræsenterer et bitmap, der igen består af et koordinatsystem af pixel-objekter. Tilsammen beskriver disse objekter den tilstand, som maleriet er i.

Bitmappens tilstand ændres ved hjælp af værktøjer, som igen er objekter: blyantsværktøj, oliemalingværktøj, tegn-rektangelværktøj, osv. De procedurer/algoritmer, som er tilknyttet de enkelte værktøjers handlinger, samarbejder med bitmap og pixel objekterne og ændrer derigennem bitmappets tilstand. F.eks. vil et blyantsværktøjs "mus-flyt-til(X,Y)" handling ændre farvetilstanden af pixel objektet på koordinat (X,Y).

De procedurer der er programmeret til et værktøjs handlinger, kan være mere eller mindre simple. TwistedBrush's procedurer er meget komplicerede og foretager en masse beregninger for at finde den rette farve værdi af de pixels, den behandler. Paint's procedurer derimod er meget mere simple og ligner derfor i mindre grad rigtig oliemaling.



Figur 3: Rød og blå oliemaling.

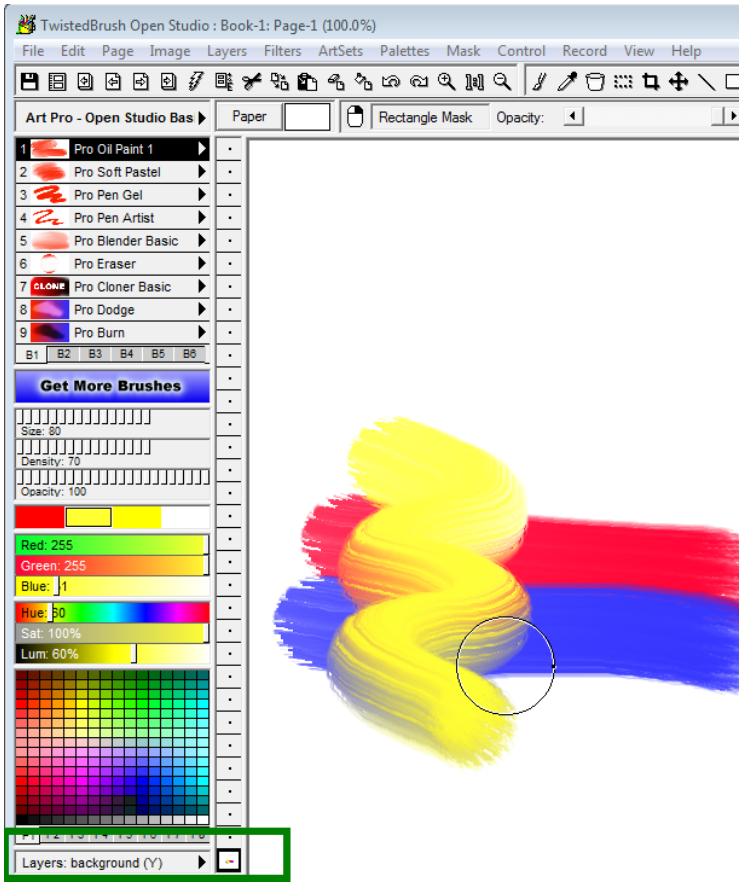
3.2 Model 2: Lag og masker

TwistedBrush benytter også bitmap og værktøjsmodellen, men den har nogle ekstra objekter i sin model, der gør, at den kan noget mere, men også er noget mere kompliceret. Det er lidt ligesom en elektrisk skruemaskine: den kan en del mere end en skruetrækker, men er lidt mere vanskelig at bruge.

TwistedBrush har ikke kun et bitmap, men *flere* bitmaps som ligger i lag "oven-på hinanden." Disse bitmaps hedder *lag* (engelsk: layer). Det maleri, du ser på din skærm, er altså ikke resultatet af kun et enkelt bitmap, men derimod summen af et antal bitmaps, som er lagt oven på hinanden. Det bliver vist lidt abstrakt, så lad os lave nogle forsøg med TwistedBrush.

I figur 3 har jeg tegnet en vandret rød og blå oliemaling streg i TwistedBrush.

I figur 4 har jeg tegnet den gule zigzag streg over den røde og blå, og vi ser at malingen bliver smurt ud. Det sker, fordi jeg har tegnet den gule streg *i det samme lag*. Nederst til venstre har jeg markeret en grøn kasse om grænsefladens knapper og felter, som muliggør interaktion med lag modellen. Der står "Layers: Background", og til højre herfor er der faktisk et stærkt formindsket billede af lagets indhold. Oven over den er der en række knapper, der bruges til at vælge, hvilket lag TwistedBrush's værktøjer skal arbejde på.

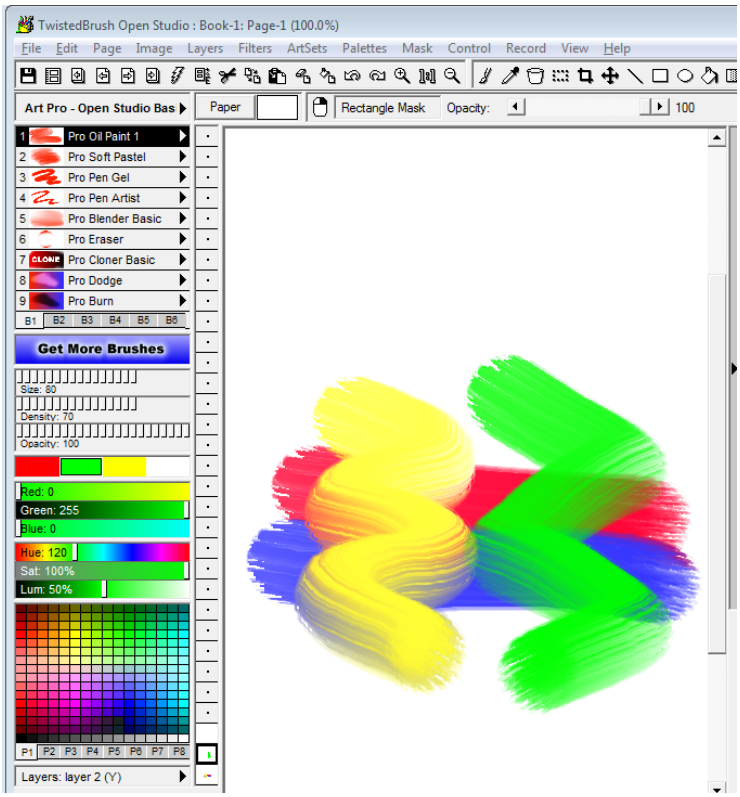


Figur 4: Angivelse af lag og knapper til at skifte lag.

Derfor kan jeg klikke på laget lige ovenover, hvorefter der står "Layers: 2", og jeg kan nu tegne min grønne streg som vist i figur 5. Da den bliver tegnet i sit eget lag, er der derfor ingen rød og blå maling nedenunder og derfor heller ingen opblanding.

Prøv at klikke på knappen, der repræsenterer baggrundslaget. Knappen virker dels til at vælge, hvilket lag man vil tegne i, og dels til at vælge, om det skal være synligt eller ej. I figur 6 har jeg gjort baggrundslaget usynligt, og kun den grønne streg træder frem.

Ok, sidste opgave, lad os lave en maske. I figur 7 har jeg startet: jeg har valgt lag 3 (knappen markeret med nederste grønne firkant) og derefter klikket på "rektangel maske" knappen (øverste grønne firkant). Derefter har jeg brugt musens *højre* museknap til at tegne et rektangel. Det har fået en mørk skakternet markering. Det betyder, at der er lagt en maske netop dér. Men jeg vil have et hul, jeg kan tegne igennem, ikke et lille rektangel jeg ikke kan tegne på. Derfor vælger jeg menupunktet "mask -> invert mask", hvorefter masken bliver "modsat". Nu kan jeg tegne mit lysblå oliestrøg, men kun dér, hvor penslen passerer hullet i masken, kommer der faktisk maling ned på laget, som det ses i figur 8.



Figur 5: Grøn oliemaling i lag 2.

For at se den endelige figur, skal jeg vælge menupunktet "menu->Display mask", der gør, at masken ikke direkte kan ses i laget, og mit maleri bliver som i figur 9.

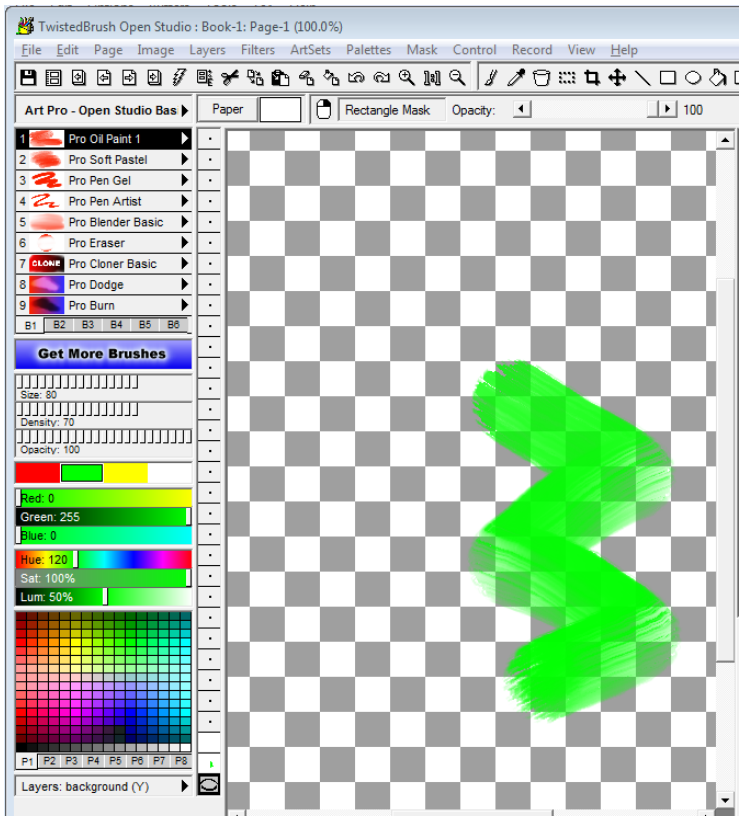
3.2.1 Objekter i modellen

TwistedBrush's model er mere avanceret, i og med den introducerer lag og masker og at et billedes tilstand ikke blot er givet ved et enkelt bitmap, men derimod givet ved summen af tilstand i alle de lag, som der er tegnet på.

Lad os først kigge på objekter for lag og masker.

Et **lag** har næsten samme definition som et bitmap, men den har en yderligere tilstand, nemlig om den er synlig eller ej. Altså

```
objekt: Lag
  attribut:
     synlig:  sand/falsk;
     indhold: koordinatsystem (1600,1200) af pixel;
  handling:
     hent-pixel-i-punkt( X, Y );
     sæt-synlighed( sand/falsk værdi );
```



Figur 6: Baggrundslaget gjort usynligt.

Så da vi "slukkede" for lag 1 i gennemgangen før, så udførte vi faktisk:

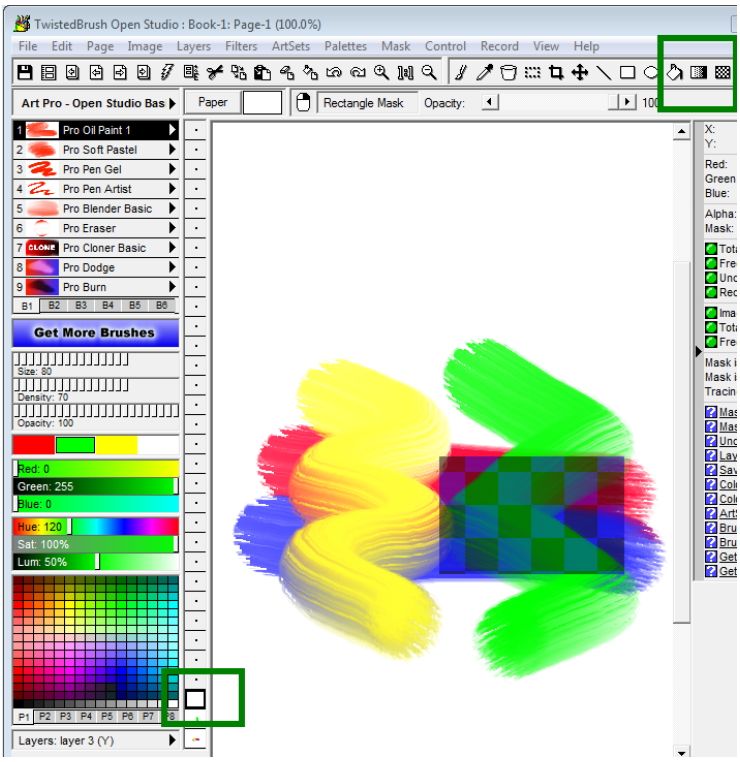
```
lag1.sæt-synlighed( falsk );
```

Det billede, som vi ser på skærmen, når vi arbejder med TwistedBrush, er naturligvis stadigvæk et bitmap, men istedet for at den farve, en pixel på (X,Y) har, blot er givet ved et enkelt pixel objekt, ja så skal man faktisk gå igennem alle lag ovenfra, indtil man finder en pixel, der er farvelagt. Hvis man ikke finder en farvelagt pixel, når man er nået helt ned til det nederste baggrundslag, så gengiver TwistedBrush en hvid pixel[4].

En **maske** har igen næsten samme definition som et bit map, men her gemmer man ikke (R,G,B) værdier i hver koordinatposition. Istedet gemmer man en sand/falsk værdi, som angiver, om værktøjerne kan ændre på tilstanden af de enkelte pixel objekter i laget eller ej. Så, hvis maskens tilstand er sand i koordinat (X,Y), så må værktøjet påvirke pixel objektet i (X,Y); hvis masken er falsk i (X,Y), ja så kan værktøjet ikke ændre pixlen.

Opgave: Lav en beskrivelse af et maske objekt på samme form som beskrivelserne af bitmap og lag ovenfor.

[4]Faktisk arbejder pixel objekter i TwistedBrush med en fjerde "kanal" ud over R, G, og B, nemlig den såkaldte *alfa kanal*, som angiver, hvor transparent den givne pixel er. Hvis den er 100% transparent, er den helt gennemsigtig.



Figur 7: En maske i lag 3.

3.2.2 Opsummering af Model 2

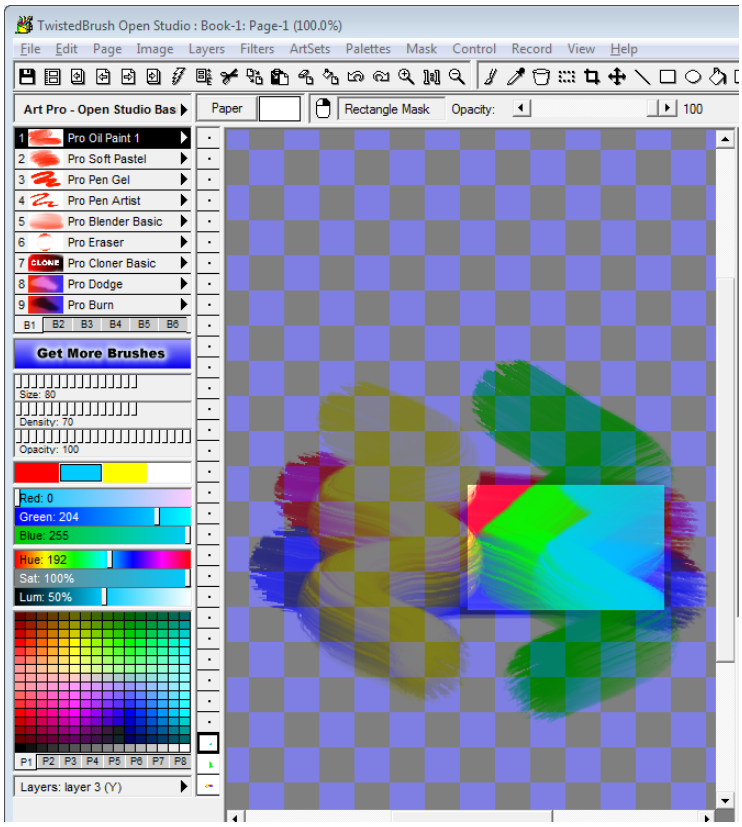
TwistedBrush bruger lag og værktøjsmodellen som grundlag. Ideen med værktøjer er identisk med den i model 1, blot har TwistedBrush en hel del mere komplicerede procedurer for at male pixels.

Lag modellen adskiller sig fra bitmap modellen. I stedet for at et billedes tilstand entydigt er givet ved tilstanden af en enkelt pixel, så er den summen af de pixels, som ligger i alle lagene. Altså, farven af et billedpunkt (X,Y) er givet ved at kigge på alle pixels i punkt (X,Y) i alle synlige lag.

Endvidere tilbyder TwistedBrush masker. En maske er et objekt, der kan afgrænse, hvilke pixels i et lag, der bliver påvirket fra et værktøjs procedurer. Specielle maskeværktøjer gør det muligt at manipulere sand/falsk tilstanden af maskens koordinatpunkter.

Bemærk, at man faktisk godt kan bruge TwistedBrush uden at bruge lag og masker. Hvis man gør det, kan man naturligvis ikke lave nær så avancerede billeder—man kan faktisk ikke meget mere end med Paint.

Bemærk også, at hvis man som bruger af TwistedBrush kun har forstået bitmap og værktøjsmodellen (den jeg kalder model 1), så har TwistedBrush pludselig et stor mængde knapper og menupunkter, som blot er meget forvirrende, fordi man ikke ved, hvad de skal bruges til. Og hvis man ved et uheld kommer til at skifte lag, mens man laver et billede, så vil det for brugeren se ud som



Figur 8: Den inverterede maske i lag 3 samt en lysblå streg.

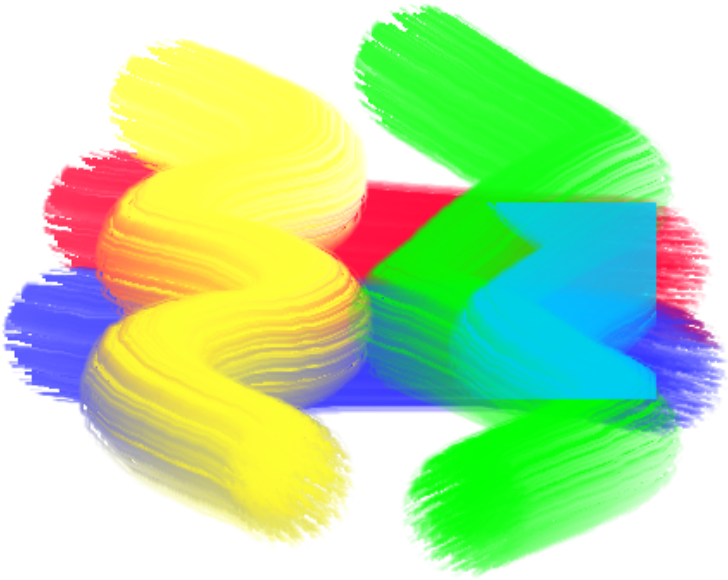
om programmet virker "forkert". F.eks. hvis man skifter til et lag, som er gjort usynligt—så kan man pludselig slet ikke tegne noget som helst! Konklusionen fra brugeren kan så (ukorrekt) blive at programmet "ikke dur."

4 En model for et program

> **Program:** Et program er struktureret som et samfund af interagerende objekter. Hvert objekt har en rolle at spille. Hvert objekt tilbyder en service/en opførsel, som bruges af de øvrige objekter i samfundet [Budd, 2002].

Med ordet "samfund" betegner vi alle de objekter, som indgår i et program. Så TwistedBrush's samfund er alle lag objekter, aktive maske objekter og alle værktøjsobjekter. Desuden har TwistedBrush et koordinerende objekt hvis tilstand afspejler, hvilket lag du lige i øjeblikket tegner i, hvilket værktøj du har valgt, om der er en maske aktiveret i laget, osv.

Alle objekter er karakteriserede ved to ting: deres *tilstand* og deres *handlinger*. Alle objekter interagerer med hinanden for samlet at give dig et korrekt og velfungerende program. De gør det ved at udføre de handlinger, der er defineret



Figur 9: Det endelige "maleri".

for det givne objekt. F.eks. så har et pixel objekt en "ændr-farve-til" handling, men det har et lag ikke. Derimod har et lag objekt en handling "hent-pixel-i-punkt" som et pixel objekt ikke har. Dermed spiller objekterne hver deres *rolle* i samfundet: pixel objekterne sørger for farvetilstanden af billedets punkter, laget for at opbevare pixelobjekterne.

På den måde kan et program sammenlignes med hvordan vi mennesker organiserer virksomheder [Christensen, 2010]. F.eks. har læger, sygeplejersker og portører deres helt specielle roller for at få et sygehus til at fungere som en samlet organisation. Og med den givne rolle en person er i, kommer der ansvar og specielle handlinger man kan og bør udføre.

I interaktive programmer skal brugeren kunne interagere med programmets objekter. Typisk vil et moderne program have vinduer med synlige objekter, menupunkter, knapper, højreklik-menuer og andre muligheder; og typisk vil disse interaktionsmuligheder direkte kalde handlinger i programmets objekter.

> **Model:** Ved et programs model forstår vi en bestemt opdeling i objekter i programmet samt et valg af, hvilke roller objekterne skal udfylde i objekt-samfundet. Til hver rolle/objekt er knyttet handlinger og tilstand.

Softwareudviklere har ret frie hænder, når de skal lave en model for et program, og derfor kan det gøres på mange forskellige måder. Ofte vil nogle være mere intuitive og hensigtsmæssige end andre; og nogle modeller vil gøre ting mulige, som andre modeller vil gøre næsten umulige. F.eks. er der ting som er helt umuligt at lave i Paint.

Det er essentielt for korrekt og optimal brug af et program, at man forstår den model, som programmets objekt-samfund er bygget op omkring. Det er ligeledes essentielt, at man har en forståelse for hvordan, man interagerer med modellens objekter.

I praksis vil softwareudviklere ofte tage udgangspunkt i objekter og roller fra "den virkelige verden." F.eks. vil det give god mening, at et økonomiprogram kan håndtere *konto* objekter, fordi bogholdere kender det begreb på forhånd og dermed fra starten har en grundlæggende forståelse for programmets model. Omvendt så kan det også være meget begrænsende, hvis man ikke opfinder nye objekter til sine modeller. F.eks. bruger stort set alle avancerede tegneprogrammer lag i deres model, men en traditionel kunstmaler tænker nok ikke rigtigt om sit maleri som en stabel individuelle billeder oven på hinanden. Et andet eksempel er tekstbehandlingsprogrammer, som har en meget avanceret model, der ikke har meget til fælles med hverken papir og blyant eller den gode gamle mekaniske skrivemaskine.

5 Opgaver

1. Vælg et program, du kender godt. Prøv at gætte på, hvilken model programmet har: hvilke objekter bruges til at gemme dets tilstand; hvilke handlinger har disse objekter mon; og hvordan interagerer du med dem via programmets brugergrænseflade?

2. Kan du huske et program, som du har/havde svært ved at bruge (eller har du et ældre familiemedlem, som har svært ved at bruge IT)? Analysér om det kan skyldes, at du/dit familiemedlem har svært ved at forstå den model, som programmet anvender.

3. Vi har trykt bøger siden 1400 tallet, og igennem disse mange år er typografi blevet et højt udviklet håndværk med en forfinet æstetik, som blandt andet kommer til udtryk i håndteringen af skrifttyper—hvordan ser tekst bedst og smukkest ud? Denne note er skrevet i et meget gammelt tekstbehandlingsprogram der hedder LaTeX(udtales "latek"). Det har meget fokus på at overholde den typografiske æstetik. Som eksempel så se på følgende ord i skrifttypen *Times*:

En fisk.

Bemærk specielt hvordan "fi" ser ud? Det er faktisk ikke, som man skulle forvente, to bogstaver—det er et sammensat tegn. Det hedder en ligatur. LaTeX's model håndterer naturligvis ligaturer og andre avancerede typografiske objekter, for at sikre, at det udskrevne dokument er af høj typografisk kvalitet.

Prøv at skrive "En fisk." i dit foretrukne tekstbehandlingsanlæg i skriftypen "Times" (eller "Times New Roman") i 36 punkt, og se om programmets model har ligatur objekter.

Litteratur

[Budd, 2002] Budd, T. (2002). *An Introduction to Object-Oriented Programming*. Addison-Wesley.

[Christensen, 2010] Christensen, H. B. (2010). *Flexible, Reliable Software—Using Patterns and Agile Development*, chapter 15. CRC Press.