Introduksjon

Denne gjennomgangen skal være tilstrekkelig til å komme i gang med å kunne lese, skrive og manipulere bitmap-filer fra et C++ program. Formatet til bitmapfiler gjennomgås først, deretter følger en gjennomgang av et C++ program som leser og manipulerer en bitmapfil (en 24 bits dib).

Informasjonen er stort sett hentet fra [Kile & Fiskvik, 09] og [Wikipedia-bmp,09].

Bitmap formatet

Bitmap formatet også kalt DIB (DIB = device-independent bitmap) brukes til å representere og lagre bilder digitalt. Normalt er hvert bildeelement (picture element / pixel / piksel) representert av 3 byte (24 bits) der hver byte henholdsvis representerer fargene rød, grønn og blå – betegnet som RGB (Red, Green, Blue) verdi.

Fargen til hver piksel er en blanding av fargene rødt, grønt og blått med ulik intensitet. Intensiteten bestemmes av verdiene på de tre bytene. Hver av disse kan variere mellom 0 og 255 (dvs. 256 ulike verdier). Dette gir 256*256*256 (>16,7 millioner) mulige farger. Dersom RGB-verdien til et piksel settes til {0,0,0} vil pikselen fremstå som svart, setter vi verdien lik {255,255,255} vil pikselen bli hvit. Setter vi for eksempel R=G=B=80 gir dette en mørk grå mens R=G=B=200 gir en lys grå. Det finnes dermed 256 nyanser av grått (inkludert svart og hvitt).

Normalt består en bmp fil av et hode (*header*) på 54 byte (se imidlertid [Wikipedia-bmp,09] for mer informasjon om dette). Deretter følger pikseldata bestående av tre bytes per piksel. Overordnet format på .bmp filer:

Et bitmapbilde er organisert som rader med piksler. Antall piksler per rad skal være delelig på 4 for at hver linje skal utgjøre et helt antall 32 bits enheter. I de tilfeller der antall piksler ikke er delelig på fire vil det legges til et antall (1-3) overflødige/ekstra bytes (kalles *byte stuffing*). Når vi leser bitmapfila må dette tas hensyn til ved at de overflødige bytene fjernes/overses.

Altså: Hvis angitt bredde på bitmapfila ikke er delelig med 4, vil det ligge 1, 2 eller 3 ekstra bytes som gjør at bredden blir delelig med 4.

Dersom angitt bredde er 400 piksler vil det ikke ligge noen ekstra bytes. Hvis bredden derimot er 401 piksler vil det ligge 1 ekstra bytes (per linje). Ved lesing av bildet må vi derfor vite om det finnes ekstra bytes slik at disse kan overses. Tilsvarende ved skriving – her må evt. ekstra bytes legges til.

Disse ekstra bytene har som regel verdien 0. Legger vi til 3 slike, og man ikke tar hensyn til ekstra bytes ved lesing, vil programmet feilaktig anta at dette er en svart piksel.

Ved lesing av et bitmap finner vi antall ekstra bytes ved å beregne bredden (i antall piksler) modulus 4. Dette vil gi 0, 1,2 eller 3 som resultat.

Bitmaphode (*header*)

En bitmapfil er egentlig en sekvens av bytes som vi kan lese vha. standard filoperasjoner i C++. Vi kan se på dette som en strøm av bytes inn til vårt program. ”Hodet” består (normalt) av 54 byte. Her ligger blant annet informasjon om hvor i bytestrømmen pikseldata begynner – denne verdien kalles *offset*(siden hodet normalt er på 54 byte vil *offset* normalt være lik 54).

I tillegg finner man bredden og høyden på bildet (i antall piksler) i hodet. Det finnes også en del annen informasjon som vi ikke trenger å forholde oss til her. *Offset* ligger fra og med byte nummer 10 i hodet mens *width* og *height* ligger fra og med byte nummer 18 (alle disse er på 4 byte hver).

Når vi har funnet *offset* kan vi flytte filpekeren slik at vi fortsetter å lese pikseldata fra og med dette byte-nummeret.

Bildets oppbygging

Informasjonen i .bmp fila er organisert slik at første piksel/byte-rad (som ligger etter *headeren*) representerer nederste pikselrad i bildet, andre piksel/byte-rad representerer nest nederste pikselrad osv.

Hvordan lese og manipulere en bitmapfil fra et C++ program

I dette delkapitlet gjennomgås følgende:

1. En bitmapfil (24 bits) leses binært fra filsystemet. Nødvednig *headerinformasjon* lagres i variabler. Hver enkelt piksel leses og legges i et Pixel-objekt. Alle Pixel-objekter legges i en pikseltabell (en vektor).
2. Bildet endres ved å endre på noen av pikslene som ligger i pikseltabellen (tegner f.eks. et svart kvadrat i bildet).
3. Den endrede pikseltabellen skrives til en ny bitmapfil. Her må vi passe på å også skrive en korrekt header til fila før pikslene skrives.

Beskrivelsen er laget slik at du skal kunne utvikle ditt eget program basert på dette. Det meste av nødvendig kode er gitt og du kan starte med å lage et tomt C++ prosjekt og lage en .cpp fil. Les gjennom notatet og forsøk å få dette til å fungere. Lag for eksempel din egen .bmp fil i Paint. Pass på å lagre denne bitmapen som 24-biters punktgrafikk (*.bmp).

Tips: fra et C++ program kan du enkelt vise frem en bitmap vha. kommandoen:

```
. . .
system(”start minbitmap.bmp”);
. . .
```

1) Lese bitmapfil

Vi starter med å lese innholdet, både header og pikseldata, fra en navngitt bitmap fil. Fila leses binært. Bitmap-formatet sier at *offset*-verdien, dvs. i hvilken byte pikseldataene starter, ligger i byte nummer 10 i *headeren*. Etter at fila er åpnet finner du denne slik (først noen variabeldeklarasjoner):

```
int main() {
  fstream infile;
  string innfilnavn = "figur1.bmp";
```

```
string utfilnavn = "figur1-ny.bmp";

unsigned int offset;        //Angir starten på pikslene
unsigned int width;         //Bildets bredde
unsigned int height;        //Bildets høyde
unsigned char header[54];   //Kan holde på hele headeren
unsigned char enPixel[3];   //Bestående av 3 byte

//En tabell som vil inneholde alle pikslene i fila:
vector<Pixel> pixelTabell;

//Åpner bitmapfila m.m.
infile.open(innfilnavn.c_str(), ios::binary | ios::in);
if (infile.fail()) {
    cout << "Feil ved åpning av fil" << endl;
} else {
    //Leser hele headeren slik at den kan brukes i ny fil:
    infile.read((char *)header,54);
    //Søker frem til 10. byte (fra start):
    infile.seekg(10, ios::beg);
    infile.read((char*)&offset,4);
    . . .
```

Her leser vi 4 byte fra og med byte nummer 10 – dette gjøres om til en **unsigned int** (et heltall uten fortegn). Husk! *unsigned* indikerer at variabelen ikke kan inneholde et negativt tall.

Variabelen *offset* vil etter dette inneholde bytenummeret til første piksel (normalt lik 54 siden bmp-headeren som regel er på 54 bytes).

På samme måte kan man finne bildets bredde og høyde, i antall piksler, slik (ligger fra og med byte nummer 18):

```
infile.seekg(18, ios::beg);
infile.read((char*)&width,4);     //i antall piksler
infile.read((char*)&height,4);    //i antall piksler
```

Her leses 4 byte fra byte nummer 18 og legges inn i variabelen width. Funksjonen read() krever at adressen til *width castes* (omgjøres) til en char-peker – derfor (char *). Lese-operasjonen kunne alternativt vært skrevet slik (egentlig mer korrekt C++ syntaks):

```
infile.read(reinterpret_cast<char *>(&width), sizeof(unsigned int));
```

Vi kan nå lese pikslene og legge dem i en enkel tabell (en vektor). Dette betyr at man må holde orden på pikslenes rad og kolonner i forhold til plassering i tabellen/vektoren.

Som vi så over er vektoren deklarert slik:

```
vector<Pixel> pixelTabell;
```

Her deklareres en tabell (en vektor) som kan holde på et vilkårlig antall Pixel-objekter. Her er det naturlig å se på hver piksel som leses fra fila som et objekt. For hver piksel som leses opprettes derfor et nytt objekt av type Pixel som legges til vektoren. Pixel-klassen må lages og kan se slik ut:

```
class Pixel {
public:
    //Konstruktør med flere argumenter:
```

```
        Pixel(int r, int g, int b);
        //Funksjon for å endre fargene til en piksel:
        void edit(int r, int g, int b);
        //Funksjoner for å lese ut R,G og B verdiene:
        int getR();
        int getG();
        int getB();

private:
        //Medlemsvariabler:
        int r;
        int g;
        int b;
};
```

NB! Her er kun klassedefinisjonen tatt med. Du må selv skrive kode for klassens medlemsfunksjoner som f.eks. konstruktøren **Pixel(int r, int g, int b)**, **getR()** osv.

Vektoren **pixelTabell** er en dynamisk tabell (dvs. den kan vokse etter behov) der du kan legge til et vilkårlig antall Pixel-objekter. Vi ser at Pixel-klassen inneholder tre medlemsvariabler (*red*, *green* og *blue*) av type int. Som vi ser er det tatt med en konstruktør som tar parametre tilsvarende medlemsvariablene. Andre medlemsfunksjoner og variabler kan også være aktuelle.

Vi er nå klar til å lese pikslene fra fila. Starter med å plassere filpekeren til korrekt byte (der første piksel ligger):

```
  //Plasserer filpeker til starten av pikseldata:
  infile.seekg(offset);
```

Alle bytene i .bmp fila ligger etter hverandre og antall bytes som leses vil da være lik 3*bredde*høyde:

```
  //NB!Bredden (i antall piksler) på bildet skal være delelig på 4.
  //Dersom bredden på bildet er delelig på 4 vil stuffing være lik 0,
  //ellers vil den bli 1,2 eller 3.
  int stuffing = width % 4;
  //pix holder rede på antall leste piksler per linje
  int pix=0;
  //Leser pikslene (3 byte per piksel):
  for(int i=0; i < (int)(height*width); i++){
      //Leser tre bytes (en piksel) i slengen:
      infile.read((char *)enPixel,3);
      //Øker antall piksler for denne linja med 1:
      pix++;

      //Når antall piksler er lik bredden sjekker vi for evt. stuffing:
      if((pix == width)){
            //p settes lik filpekeren
            int p = infile.tellp();

            //Søker forbi evt. stuffbytes:
            infile.seekg(p + stuffing);
            pix=0;
      }

      //Opprett et nytt pikselobjekt og legg i vektor:
```

```
        Pixel nyPixel((int)enPixel[2], (int)enPixel[1], (int)enPixel[0]);

        pixelTabell.push_back(nyPixel);
    }
```

Bredden på bitmappen (i antall piksler) skal være delelig med 4. Dersom dette ikke er tilfelle vil det ligge noen ekstra bytes på slutten av hver linje. Disse må overses ved lesing (ved skriving til fil må slike bytes legges til).

Det er også slik at første ”byterad ” i fila tilsvarer nederste pikselrad i bildet, andre ”byterad” tilsvarer nest nederste pikselrad i bildet osv. Dette betyr at første piksel (med indeks 0) i pixelTabell tilsvarere pikselen i nedre venstre hjørne av bildet, andre piksel (med indeks 1) i pixelTabell tilsvarer pikselen til høyre for denne osv.

Legg merke til hvordan objektet nyPixel opprettes. Hver piksel som leses fra fila består av tre bytes, en verdi for rød (0-255), en for grønn (0-255) og en for blå (0-255). Disse verdiene legges først i en char-tabell (enPixel). Husk at char egentlig er en 1-bytes heltallstype som kan holde på verdier fra -127 til og med +127. Bruker vi **unsigned char** fungerer dette som en byte som kan holde på verdier fra og med 0 til og med 255 (som er det vi trenger).

Verdiene i denne tabellen brukes til å opprette et Pixel-objekt nyPixel. Rød-verdien ligger i det 3. elementet, grønn-verdien ligger i det andre mens blå-verdien ligger i det første (indeks = 0). Det nye Pixel-objektet legges (kopieres) så til vektoren pixelTabell vha. `push_back(nyPixel)`.

Når for-løkka er fullført vil alle pikslene ligge i vektoren. Bildet er nå representert som en tabell med Pixel-objekter.

2) Endre pikslene

Det er nå relativt enkelt å endre på pikselverdiene og eventuelt skrive disse tilbake til en ny fil. La oss se på et enkelt eksempel der vi tegner et svart kvadrat (med kanter på _width antall piksler) på et gitt sted i bildet (gitt av _row og _col).

```
//Kaller fra main:
settMerke(pixelTabell, 10, 10, 50, width);
```

Dette gir et kvadrat på 50 piksler i rad 10 kolonne 10. Vi sender også med bredden på bildet i antall piksler. Funksjonen kan for eksempel se slik ut (legges utenfor main()):

```
void settMerke(vector<Pixel> &pixelTabell, int _row, int _col, int _width,
               int _rowwidth) {
      // _row og _col er antatt å være 1-basert!
      // _width er bredden på bildet i antall piksler.
      // idx representerer indeks i tabell:
      int idx;
      //sr=start row, sc=start column
      //er=end row, ec=end column
      int sr = _row, sc=_col, er=sr+_width, ec=sc+_width;

      for (int i=sr; i < er; i++) {
            for (int j=sc; j < ec; j++) {
                  //Finner indeks i tabell som svarer til
                  //gitt rad (i) og kolonne (j):
                  idx = (i - 1) * _rowwidth + (j - 1);
```

```
                pixelTabell[idx].edit(0,0,0);
            }
        }
}
```

Når vi skriver den endrede pixelTabellen til en ny fil er det viktig at vi skriver en korrekt BMP-header til den nye fila. For å være sikker på at man får korrekt header kan man ta vare på *headeren* fra den originale fila (vist over):

```
. . .
unsigned char header[54]; //Gir plass til headeren.
//Leser hele headeren slik at den kan brukes i ny fil:
infile.read((char *)header,54);
. . .
```

3) Skrive de endrede pikslene til en ny .bmp fil

For å gjøre dette åpner vi en ny binærfil for skriving.

Kode:

```
//Skriver manipulert innhold til ny fil:
fstream outfile;
outfile.open(utfilnavn.c_str(), ios::binary | ios::out);
//Skriver headeren:
outfile.write((char *)header, 54);

//"Hjelpebytes":
unsigned char utdata[3];
unsigned char stuffBytes[3] = {0, 0, 0};

//Plasserer skrivepeker (byte 54):
outfile.seekp(offset, ios::beg);

//pix holder rede på antall skrevne piksler per linje
pix=0;

//Skriver alle pikslene fra tabellen til den nye fila:
for(int j=0; j < (int)pixelTabell.size(); j++){
      utdata[0] = pixelTabell[j].getB();
      utdata[1] = pixelTabell[j].getG();
      utdata[2] = pixelTabell[j].getR();

      outfile.write((char *)utdata, 3);
      pix++;
      //Legger til evt. stuffing:
      if((pix == width)){
            outfile.write((char *)stuffBytes, stuffing);
            pix=0;
      }
}
infile.close();
outfile.close();
```

Vi skriver opprinnelig header til den nye fila. Har man endret på bildets bredde og/eller høyde må *headeren* også endres i forhold til dette.

Her bruker vi en char-tabell med plass til tre bytes. For hver runde i for-løkka fylles denne med informasjon fra pikseltabellen. Deretter skrives denne til fila (outfile.write((char *)utdata, 3)).

Variabelen stuffing har her samme verdi som tidligere siden vi her antar at bildets bredde og høyde ikke er endret. Vi ser at det legges til et antall 0-bytes dersom antall piksler per rad ikke er delelig med 4.

Referanser

[Kile & Fiskvik, 09] Rapport - Bildebehandling, Avlesing av digitalt display, Martin Kile og Daniel Fiskvik, 2009.

[Wikipedia-bmp, 09] BMP file format.
http://en.wikipedia.org/wiki/BMP_file_format#Related_formats