Vrije Universiteit Brussel

FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT OF ELECTRONICS AND INFORMATICS

Computersystemen

Project Computersystemen

Pieter Meiresone, Adrien T'Kint

Academiejaar 2013-2014

Inhoudsopgave

1 Overzicht.

Het programma is een JPG decoder die beelden behandelt met formaat van 320x200 en gecompresseerd onder een YUV 4:2:0 kleurenruimte.

Voor het werken in teamverband is er gebruik gemaakt van een GitHub-repository [1].

2 Functionaliteit van de applicatie.

De applicatie kan JPG-beelden met een dimensie van 320x200 gemaakt in Paint openen. Door de complexiteit van de JPG-standaard kunnen wij enkel deze beelden ondersteunen[1].

De bestandsnaam van de afbeelding die geopend moet worden, moet hardcoded in de file MAIN.ASM geplaatst worden. Dit gebeurt bij het label 'BESTANDSNAAM' in het data-segement. Merk op dat de bestandsnaam niet meer als 8 karakters mag bevatten. Het JPG-bestand moet geplaatst worden in dezelfde folder als de source code. Vervolgens moet de gebruiker het programma compileren met het commando *nmake*, nadien kan hij het programma runnen met het commando MAIN.EXE.

Vervolgens wordt de foto getoond op het scherm, de gebruiker kan het programma afsluiten door op de enter-toets te duwen.

3 High-level en abstract design.

Het implementeren van een JPG-decoder is een sequentieel proces waarin een duidelijk aantal verschillende stappen in zijn te onderscheiden:

1. Openen en inlezen van het bestand, herkennen van de signalisatieparameters om de tabellen en de gecomprimeerde data in segmenten weg te schrijven. Omdat er bij een JPG-file bit stuffing wordt toegepast, worden bij het wegschrijven van de bitstream in zijn segment de ff00-woorden vervangen door ff.

2. Vier Binary trees opstellen uit geextraeerde tabellen die het aantal leaf-nodes per row geven.

3. Vervolgens worden de verschillende MDU-blokken uit de datastream gehaald. Per MDU worden er volgende operaties achtereenvolgens uitgevoerd:

[1] Afbeeldingen van externe bronnen kunnen dus ook geopend worden door deze eerst in Paint te openen en vervolgens met behulp van Paint op te slaan.

(a) Entropy decoding.

i. De coden waaruit de gecomprimeerde data bestaat komen overeen met leaf-codes. Elke code wordt dus vergeleken totdat er een gelijke leaf-code wordt teruggevonden. De DC waarden worden met hun eigen tabellen gevonden en worden behandeld relatief tov. het voorgaande waarde.

ii. De rangen van de leaf-codes laten toe om hun overeenkomstige symbolen te vinden. Een symbool geeft het aantal nullen die weggeschreven moeten worden en het aantal bits van de gecomprimeerde data die het volgende van nul verschillend getal vormen.

iii. De van nul-verschillende waarden die met een 0 beginnen moeten als negatieve beschouwd worden: ze worden omgezet in hun two's complement notatie.

(b) De gegevens bevinden zich nu in een zigzag ordening. Deze ordening wordt omgezet naar een matrixvoorstelling.

(c) De elementen in de matrices worden gedequantizeerd.

(d) De inverse discrete cosinus transformatie wordt uitgevoerd. Hiervoor maken we gebruik van een discrete cosinus transformatie die enkel gebruik maakt van integers en geen floating points [3].

(e) De gegevens, gecompresseerd onder een 4:2:0 downsampling, worden terug geconverteerd een 4:4:4 verhouding.

(f) De YCbCr kleurenruimte wordt omgezet naar een RGB-kleurenruimte. Dit gebeurt volgens volgens formules 1 - 3 [2].

$$R = Y + 1.402 * (Cr - 128) \tag{1}$$
$$G = Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128) \tag{2}$$
$$B = Y + 1.772 * (Cb - 128) \tag{3}$$

(g) Voor elke RGB-waarde wordt er een index bepaalt van een kleur in het kleurenpalet die deze het best benadert.

(h) Deze index wordt weggeschreven in een apart data segment van $64kB$ dat dienst doet als screenbuffer.

4. De screenbuffer wordt weggeschreven in het videogeheugen.

4 Specifieke problemen, en hun oplossingen.

Beperkt segmentgeheugen.

Hiervoor werken we MDU per MDU zodat er op geen enkel moment te veel geheugen wordt gebruikt. Anders zou bijvoorbeeld juist voor de inverse Cosine Transform 99840 words moeten opgeslaan worden, wat alleen te verwezenlijken is met behulp van extern geheugen.

$$20 * 13 * MDU * \frac{(4Y + 1Cb + 1Cr)blokken}{MDU} * 64\frac{Words}{block} = 99840words$$

Door het beperkt geheugen kan het ook voorvallen dat er bepaalde JPG-afbeeldingen niet geopend kunnen worden.

Geen ondersteuning voor floating points.

In de Intel 8086 kan er enkel met integer-waarden gerekend worden. Vermits een standaard DCT met floating points werkt, is er gebruik gemaakt van een integer DCT[3]. Verder is er voor formules 1 - 3 gebruik gemaakt

van volgende variant:

$$R' = 100 * Y + 140 * (Cr - 128) \tag{4}$$
$$G' = 100 * Y - 34 * (Cb - 128) - 71 * (Cr - 128) \tag{5}$$
$$B' = 100 * Y + 177 * (Cb - 128) \tag{6}$$

Vervolgens worden R', G' en B' gedeeld door 100. Dit geef benaderende waarden voor R, G en B.

Meerdere ASM-files.

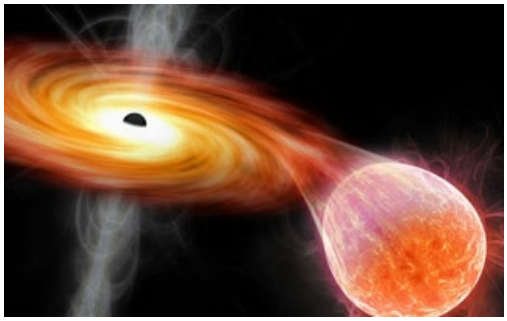
Om lange ASM files met duizenden lijnen code te vermijden, is de source code van het programma opgesplitst in verschillende files. Hierdoor werden de commando's in de makefile te lang. Door verscheidene ASM-files te combineren in een library konden we het aantal argumenten beperken.

Kleinigheden

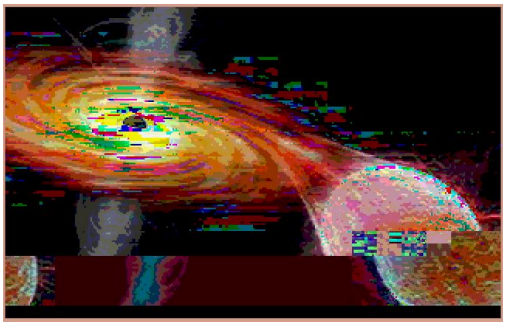
- De lengte van het te inlezen bestand is beperkt tot 8 karakteren: beschouw alleen de 8 eerste karakteren en vervangt de 2 laatste door ~ 1.
- Jump out of range: Het bereik van de jumps gaat van -126 tot +127. Heb jij meer lijnen tussen je vertreklijn en je label, definieer dan een tussenligende label welke verwijst naar uw eindbesteming.
- De codes (waaruit de bitstream bestaat) zijn niet (byte-)gealigneerd. Om de codes te kunnen vergelijken met de leaf-nodes heeft men moeten zorgen dat de bitstream tot de bit na in stuk kon gekapt worden.

5 Kwaliteit van de JPG decoder.

De kwaliteit van onze JPG-decoder bespreken we aan de hand van enkele voorbeelden. Op figuur 1 is het resultaat van 3 foto's te zien. Links staat telkens het origineel, in een resolutie van 320x200. Rechts staat een printscreen van het resultaat in DOSBox.



(a) Foto 1.



(b) Foto 1 resultaat.

(c) Foto 2.

(d) Foto 2 resultaat.

(e) Foto 3.

(f) Foto 3 resultaat.

Figuur 1: Kwaliteitsbespreking.

Algemeen is de kwaliteit van de afbeeldingen slecht, dit kan echter verklaard worden:

- Door gebruik te maken van een integer DCT worden er nauwkeurigheidsfouten gemaakt. Tests in Matlab hebben uitgewezen dat de relatieve fout geïntroduceerd door de integer DCT t.o.v. de werkelijke DCT soms meer als 10% bedraagt. Deze fout kan verder verkleind worden door gebruik te maken van nauwkeurige integer DCT's, te vinden op [3].
- Bij het gebruik van formules 4 - 6 worden er ook afrondingsfouten gemaakt.

A Referenties

Referenties

[1] *GitHub repository.* `https://github.com/MichelineIsImpossible/JPGEditor/`

[2] *JPEG File Interchange Format.* (1992) Geraadpleegd op 2 januari 2014 via http://www.w3.org/Graphics/JPEG/jfif3.pdf

[3] *Integer DCT.* Geraadpleegd op 2 januari 2014 via http://fgiesen.wordpress.com/2013/11/04/bink-2-2-integer-dct-design-part-1/ en http://fgiesen.wordpress.com/2013/11/10/bink-2-2-integer-dct-design-part-2/.

[4] Bruylants, T. & Temmermans, F. & Jansen, B. & Schelkens, P. (2008) *Een baseline JPEG encoder.*

[5] *JPEG Huffman Coding Tutorial.* Geraadpleegd op 2 januari 2014 via http://www.impulseadventure.com/photo/jpeg-huffman-coding.html