การบีบอัดข้อมูลภาพ
(IMAGE DATA COMPRESSION)

โดย

นายอภิพล คุณาภิบาล

นายอัคเรศ แดงสุวรรณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเลคทรอนิคส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

การลดข้อมูลภาพ

(IMAGE DATA COMPRESSION)

โดย

นายอภิพล คุณาภิบาล

นายอัคเรศ แตงสุวรรณ

อาจารย์ที่ปรึกษา

รศ.ดร. มนัส สังวรศิลป์

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาอิเลคทรอนิคส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชา วิศวกรรมอิเลคทรอนิคส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การลดข้อมูลภาพ

ผู้จัดทำ

1. นาย อภิพล คุณาภิบาล 36014549
2. นาย อัคเรศ แตงสุวรรณ 36014562

(รศ.ดร. มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การลดข้อมูลภาพ

IMAGE DATA COMPRESSION

1. นาย อภิพล คุณาภิบาล 36014549

2. นาย อัครเรศ แตงสุวรรณ 36014562

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการทดสอบได้

(รศ.ดร.มนัส สังวรศิลป์)

อาจารย์ที่ปรึกษา

การลดข้อมูลภาพ

อภิพล คุณาภิบาล
อัคเรศ แตงสุวรรณ
รศ.ดร.มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2539

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการศึกษาพื้นฐานของภาพในระบบดิจิตอลและทฤษฎีต่างๆ เกี่ยวกับการลดข้อมูลภาพซึ่งจะศึกษาทั้งหมด 2 วิธี คือการลดข้อมูลภาพด้วยวิธีการของฮัฟแมน (HUFFMAN) และการประยุกต์ใช้ทฤษฎีการส่งข้อมูลภาพโทรทัศน์ในระบบเอ็นทีเอสที (NTSC) ซึ่งทั้ง 2 วิธี เป็นวิธีการที่แพร่หลายพอสมควรและใช้งานได้ดีและมีการสร้างโปรแกรมเพื่อจำลองทฤษฎีดังกล่าวขึ้น โดยเป็นโปรแกรมที่ใช้งานเกี่ยวกับคอมพิวเตอร์ส่วนบุคคลพร้อมระบบปฏิบัติการไมโครซอฟท์วินโดวส์

Image data compression

APIPOL GUNABHIBAL
AKARET TAENGSUWANN
Assoc. prof. Dr. MANAS SANGWORASILP
Academicc year 1996

Abstract

This thesis is study about principle of digital images and data compression theory.The thory is studoed that have 2 methods HUFFMAN and NTSC system transmission theory .Both is well known method and use to real performance.The applications is designed to simulate theory.The applications work on personal computer and run under Microsoft windows.

สารบัญ

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความหมายและประโยชน์การประมวลผลภาพ

การประมวลผลภาพ (image processing) หมายถึงการใช้ขั้นตอนหรือกรรมวิธีใดๆ มากระทำกับภาพโดยมีวัตถุประสงค์ให้ได้ภาพใหม่ตามที่ต้องการ

ในยุคปัจจุบันเป็นยุคของข่าวสารข้อมูล การประมวลผลภาพค่อนข้างจะมีประโยชน์มาก เพราะมีเนื้อหาที่กว้างและอีกทั้งข้อมูลในยุคนี้มักจะอยู่ในรูปภาพหรือเสียงข้อมูลในลักษณะภาพค่อนข้างมีความสำคัญอย่างยิ่ง ดังนั้นในปริญญานิพนธ์ฉบับนี้จึงมุ่งเน้นการศึกษาในด้านการประมวลผลภาพ ข้อมูลเหล่านี้อาจแสดงอยู่ในรูปของจอโทรทัศน์หรือจอภาพซึ่งข้อมูลเหล่านี้สามารถนำมาใช้ได้อย่างกว้างขวางไม่ว่าจะเป็นข้อมูลจากดาวเทียมสำรวจพื้นโลกซึ่งสามารถนำมาใช้ประโยชน์ในการพัฒนาและรักษาสภาพแวดล้อมให้ดีขึ้นหรือระบบฐานข้อมูลของแฟ้มประวัติบุคคลวิ่งมีทั้งใบหน้าและประวัติของบุคคลนั้นๆหรือภาพจากการเอ็กซ์เรย์ส่วนต่างของร่างกายมนุษย์เพื่อใช้ประโยชน์ในทางการแพทย์เช่นกัน

1.2 วัตถุประสงค์ของปริญญานิพนธ์

ปริญญานิพนธ์เล่ม นี้จะศึกษาพื้นฐานของภาพและ ลักษณะการประมวลผลภาพแบบ เบื้องต้นเพื่อให้เข้าใจการประมวลผลภาพดียิ่งขึ้น

1.3 ขอบเขตของปริญญานิพนธ์

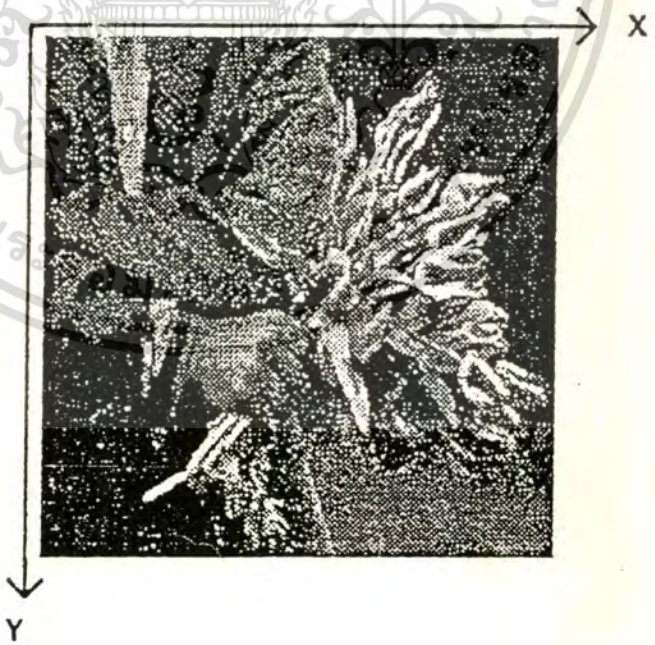
จากหัวข้อ 1.2 เราจะพิจารณาภาพนิ่งแบบดิจิตอลและมีสี 256 สี โดยจะศึกษาในหัวข้อการลดข้อมูลภาพด้วยวิธีของฮัฟแมนและทฤษฎีการส่งภาพของโทรทัศน์ในระบบเอ็นทีเอสซี

บทที่ 2
หลักการประมวลผลภาพ
(Image processing)

ในการประมวลผลสัญญาณภาพด้วยระบบคอมพิวเตอร์ จำเป็นต้องเปลี่ยนข้อมูลหรือสัญญาณภาพที่อยู่ในรูปอนกลอก ให้เป็นสัญญาณทางดิจิตอล เพื่อประโยชน์ในการคำนวณและประมวลผลได้ง่าย ในบทนี้ จะกล่าวถึง ความหมายของภาพในระบบดิจิตอลและคณิตศาสตร์ที่เกี่ยวข้อง

2.1 ความหมายและนิยามของภาพในระบบดิจิตอล

ภาพ (Image) ในเชิงคณิตศาสตร์จะหมายถึง ฟังก์ชัน 2 มิติ f(x,y) โดย x และ y เป็นแกนพิกัดในระนาบ 2 มิติ ค่าฟังก์ชัน f(x,y) จะเป็นสัดส่วนกับความสว่างหรือความเข้มของภาพที่ตำแหน่ง (x,y) ซึ่งเราเรียกว่า ระดับสีเทา (Gray level) ในรูปที่ 2.1 แสดงให้เห็นถึงระนาบและจุดพิกัดของภาพ ซึ่งปกติเราจะให้จุดกำเนิดของแกนพิกัด (Coordinate) อยู่ทางมุมบนซ้ายของภาพ



รูปที่ 2.1 ระนาบและพิกัดที่ใช้ในระบบภาพ

ภาพ 2 มิติ ที่แทนด้วยฟังก์ชัน f(x,y) โดย x และ y เป็นแกนในระนาบของภาพ ค่าของ ฟังก์ชันที่จุด (x,y) คือความเข้มของแสงที่จุดนั้น เนื่องจากแสงเป็นพลังงานรูปหนึ่ง ดังนั้น f(x,y) ต้องไม่เป็นศูนย์ และมีค่า (finite) นั้นคือ

$$0 < f(x,y) < \infty \qquad \ldots (2.1.1)$$

โดยธรรมชาติของแสง ซึ่งจะต้องมีแหล่งกำเนิดแสงและส่วนที่สะท้อนแสง ดังนั้นเราสามารถแยกฟังก์ชัน f(x,y) ออกเป็น 2 ส่วนคือ อิลูมินชั่นคอมโพเนนท์ (illuminstion component) และ รีแฟล็กแทนท์คอมโพเนนท์ (reflectant component) จะได้ว่า

$$f(x,y) = I(x,y) \times r(x,y) \qquad \ldots (2.1.2)$$

เมื่อ

$$0 < i(x,y) < \infty \qquad \ldots (2.1.3)$$

และ

$$0 < r(x,y) < 1 \qquad \ldots (2.1.4)$$

สมการ (2.4) แสดงให้เห็นว่า ฟังก์ชันการสะท้อนถูกจำกัดขอบเขตระหว่าง 0 (ซึ่งหมายถึงการดูดซึมสมบูรณ์) และ 1 (ซึ่งหมายถึง การสะท้อนโดยสมบูรณ์) ธรรมชาติของ i(x,y) ขึ้นอยู่กับแหล่งกำเนิดแสง ในขณะที่ r(x,y) ขึ้นอยู่กับวัตถุที่สะท้อนแสงมาเข้าตา

ดังที่กล่าวมาแล้ว ความเข้มของภาพที่จุด (x,y) เราเรียกว่า ระดับสีเทา (Gray level) l จากสมการที่ (2.2) ถึง (2.4) จะเห็นว่า l ควรอยู่ในช่วง

$$L_{min} \leq l \leq L_{max} \qquad \ldots (2.1.5)$$

ในทางทฤษฎี L_{min} ต้องมีค่าบวก ในขณะที่ L_{max} ต้องมีค่าน้อยกว่าอนันต์ ในทางปฏิบัติ $L_{min} = L_{min} r_{min}$ และ $L_{max} = L_{max}\, r_{max}$ ช่วงของ (L_{min} , L_{max}) เราเรียกว่าช่วงของระดับสีเทา ในทางปฏิบัติโดยใช้หลักคณิตศาสตร์ เรานิยมปรับช่วง (L_{min} , L_{max}) ให้เป็นช่วง (0,L) โดย L=0 หมายถึง ดำสนิท และ L=1 หมายถึงขาว

2.2 การสุ่มแบบสม่ำเสมอและควอนไตเซชั่น

(Unifrom sampling and Quantization)

เพื่อที่จะประมวลสัญญาณภาพด้วยระบบคอมพิวเตอร์ ฟังก์ชันของภาพ f(x,y) จะถูกทำให้เป็นสัญญาณไม่ต่อเนื่อง ทั้งระนาบของภาพ ซึ่งเราเรียบกว่า การสุ่มภาพ (Image sampling) ของฟังก์ชันที่ได้เรียกว่า การควอนไตเซชั่นระดับสีเทา (gray level quantization)

สมมุติว่าสัญญาณภาพต่อเนื่อง f(x,y) ถูกดิจิไตซ์ ในระนาบ X Y เป็นช่วงเท่าๆกัน เราสามารถจัด f(x,y) ให้อยู่ในรูปเมตริกซ์ ขนาด NxN ได้ดังสมการ (2.4.1)

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,N-1) \end{bmatrix} \quad \dots (2.4.1)$$

ทางขวาของสมการ จะเรียกว่าภาพดิจิตอล และทุกๆสมาชิกของเมตริกซ์จะเรียกว่า พิกเซล จากขบวนการสร้างภาพดิจิตอลข้างต้น จะเห็นว่า เราต้องทราบขนาดความละเอียดของภาพ NxN พิกเซล และจำนวนระดับของ Gray level ในทางปฏิบัติการทำควอนไตเซชั่นในระบบภาพดิจิตอล จะเป็นค่าของ 2 ยกกำลังจำนวนเต็ม คือ

$$N = 2^n \quad \dots (2.4.2)$$

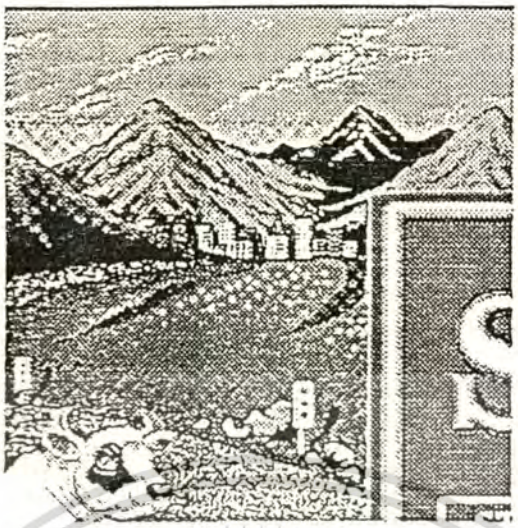
และ

$$G = 2^m \quad \dots (2.4.3)$$

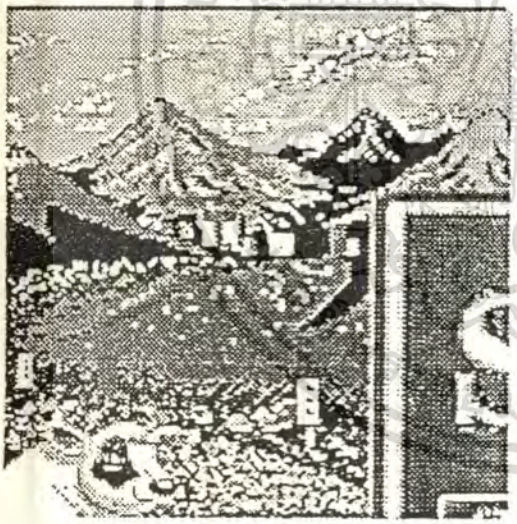
เมื่อ G คือ จำนวนระดับของ Gray level ดังนั้นจำนวน บิท (bit) ที่ใช้ในการเก็บภาพหนึ่งภาพ ที่ถูกดิจิไตซ์ คือ

$$B = N \times N \times m \text{ บิท} \quad \dots (2.4.4)$$

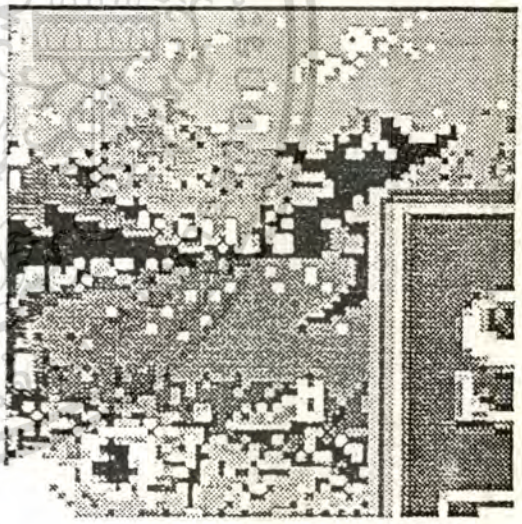
ดังตัวอย่างภาพขนาด 128 x 128 Pixel และระดับ Gray level จำนวน 256 ระดับ ต้องใช้หน่วยความจำขนาด 131,072 บิท ในรูปที่ 2.4 ได้แสดงการเปรียบเทียบภาพเมื่อลดความละเอียดของภาพลง และตาราง 2.1 แสดงจำนวนไบท์ ที่ใช้ในการเก็บภาพ เมื่อ N และ M เปลี่ยนไป



256*256



128*128



64*64

รูป 2.4 เปรียบเทียบภาพเมื่อลดความละเอียดของภาพลง

ตาราง 2.1 จำนวน BYTE ที่ใช้ในการเก็บภาพ เมื่อ N และ m เปลี่ยนไป

N \ m	1	2	3	4	5	6	7	8
32	128	256	512	512	1024	1024	1024	1024
64	512	1024	2048	2048	4096	4096	4096	4096
128	2048	4096	8192	8192	16384	16384	16384	16384
256	8192	16384	32786	32786	65536	65536	65536	65536
512	32786	65536	131072	131072	262144	262144	262144	262144

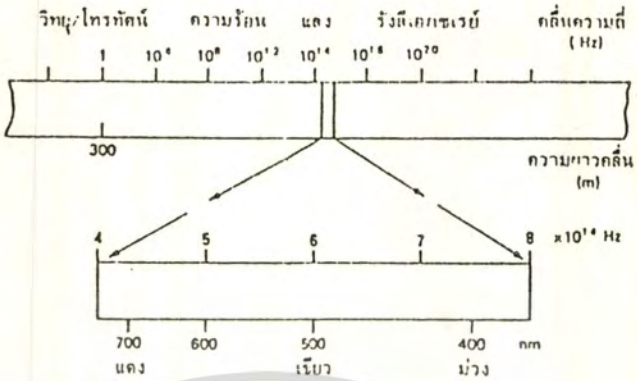
2.3 พื้นฐานของสี

2.3.1 แสงเป็นคลื่นแม่เหล็กไฟฟ้า

แสงเป็นคลื่นแม่เหล็กไฟฟ้าอย่างหนึ่งเช่นเดียวกับคลื่นวิทยุต่างกันตรงที่ความถี่และความยาวคลื่นอยู่คนละย่านความถี่กันเท่านั้น คลื่นวิทยุกระจายเสียง FM อยู่ในย่านความถี่ 88-108 เมกะเฮิรตซ์ คลื่นที่ใช้แพร่ภาพโทรทัศน์อยู่ในย่านความถี่ไม่เกิน 850 เมกะเฮิรตซ์ แต่คลื่นแสงอยู่ในย่าน ความถี่ 400 ล้าน - 800 ล้านเมกะเฮิรตซ์ ซึ่งสูงยิ่งกว่าคลื่นวิทยุ

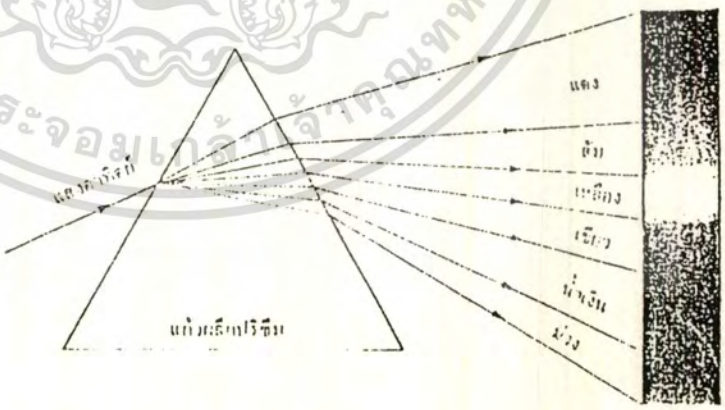
ถ้าวัดเป็นความยาวคลื่นในย่านความถี่วิทยุกระจายเสียงหรือย่านแพร่ภาพโทรทัศน์ความยาวคลื่นจะวัดได้เป็นเซนติเมตรหรือเป็นเมตร แต่สำหรับคลื่นแสงความยาวคลื่นจะยิ่งสั้นเป็น ไมครอน (ความยาว 1 ไมครอนเท่ากับ 1 ส่วนล้านส่วนของเมตร)

ผู้อ่านคงจะนึกภาพออกว่าคลื่อนแสงมีความถี่สูงเพียงใดเมื่อเทียบกับคลื่นวิทยุ พิจารณารูปที่ 2.3 จะเห็นว่าย่านความถี่คลื่นวิทยุอยู่คนละย่านความถี่กับคลื่นแสงแต่มีคุณสมบัติเป็นคลื่นแม่เหล็กไฟฟ้าเหมือนกันแสงมีความเร็ว 186,000 ไมล์ต่อวินาที่ หรือ 300,000 กิโลเมตรต่อวินาที และมีความถี่เป็นสัดส่วนผกผันกับความยาวคลื่น

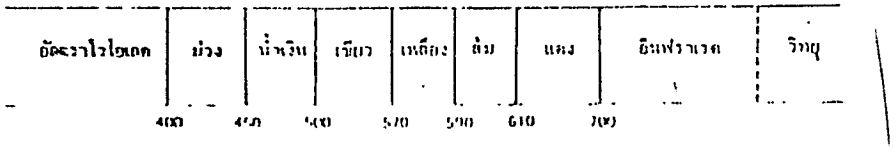


รูปที่ 2.3 ย่านความถี่ของคลื่นแสงเทียบกับคลื่นอื่น

ในปี ค.ศ. 1666 เซอร์ไอแซก นิวตัน ได้ค้นพบว่า เมื่อลำแสงอาทิตย์ส่องผ่านแก้วผลึกปรซึม (glass prism) แสงที่ทะลุผ่านแท่งแก้วไปยังอีกด้านหนึ่งจะกระจายแยกออกเป็นแสงสีต่างๆดังรูปที่ 2.4 แสดงว่าแสงอาทิตย์ที่เรามองเห็นเป็นสีขาวปรือสีใสนั้น แท้ที่จริงแล้วประกอบด้วยแสงสีต่างๆหลายสี แสงแต่ละสีที่เห็นนั้นต่างก็มีความถี่หรือความยาวคลื่นใกล้เคียงกัน



รูปที่ 2.4 การแยกแสงอาทิตย์ออกเป็นแสงสีรุ้งโดยใช้แก้วปริซึม



รูปที่ 2.5 แสงในย่านความยาวคลื่นที่ตามนุษย์สามารถมองเห็นได้ ความยาวคลื่นเป็นมิลลิไมครอน

จากรูปที่ 2.5 จะเห็นว่าแสงสีม่วง (violet) มีความยาวคลื่นประมาณ 400 มิลลิไมครอนแสงสีแดงมความยาวคลื่นประมาณ 700 มิลลิไมครอน (1 micron = 1000 millimicron) แสงในย่านความยาวคลื่น 400 - 1,000 มิลลิไมครอนนี้เท่านั้นที่ตามนุษย์มอง เห็นได้แสงที่มีความยาวคลื่นอยู่นอกย่านที่ตามองเห็นนั้นได้แก่แสงอัลตราไวโอเลต (ultraviolit แปลว่า มีความถี่สูงเลยแสงสีม่วงขึ้นไป) และแสงอินฟราเรด (infrared แปลว่า มีความถี่ต่ำกว่าแสงสีแดงลงมา)

ขอบเขตของควา ยาวคลื่นของแสงสีต่างๆมีค่าประมาณดังนี้

สีม่วง	ประมาณ	400-450	มิลลิไมครอน
สีน้ำเงิน	ประมาณ	450-500	มิลลิไมครอน
สีเขียว	ประมาณ	500-570	มิลลิไมครอน
สีเหลือง	ประมาณ	570-590	มิลลิไมครอน
สีส้ม	ประมาณ	590-610	มิลลิไมครอน
สีแดง	ประมาณ	610-700	มิลลิไมครอน

ขอบเขตที่ต่อเชื่อมกันระหว่างสีไม่สามารถชี้เจาะจงลงไปได้แน่นอนว่าแต่ละสีจะสิ้นสุดลงตรงไหน เพราะต่างก็ค่อยๆจางลงแล้วเข้ามาเชื่อมติดต่อกันและกัน

เส้นที่แบ่งเขตนั้นเป็นเส้นที่เราจะประมาณเอาว่าสีหนึ่งๆได้สิ้นสุดลง (เพราะไม่สามารถรู้ได้แน่ชัดว่าสิ้นสุดลงตรงไหน) ตัวเลขที่กล่าวมานี้อาจมีค่าแตกต่างกันบ้างตามตำราแต่ละเล่ม

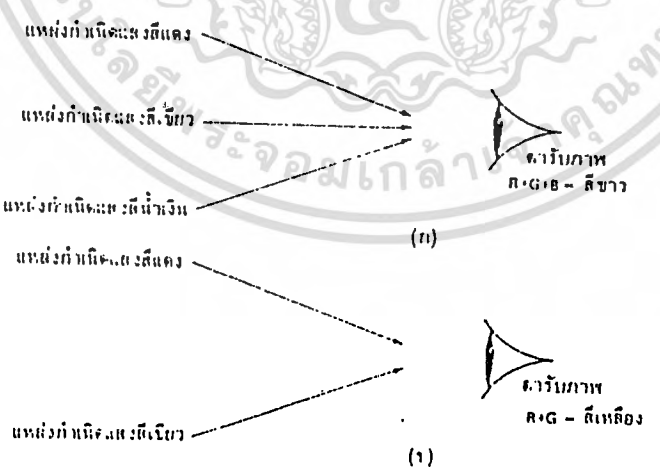
2.3.2 การผสมสีแสง

เราทราบแล้วว่าแสงที่มองเห็นเป็นสีต่างๆนั้นมีความถี่หรือความยาวคลื่นไม่เท่ากันการที่ตาเรามองเห็นสีสันก็คือ เรามองเห็นแสงความถี่ต่างๆกัน แสงจึงเป็นพื้นฐานของเรื่องสีในระบบโทรทัศน์ สีการสร้างภาพสีและการกำเนิดสัญญาณภาพสีอาศัยหลักการผสมสีแสง ซึ่งเป็นการผสมเชิงบวก (additive mixing)

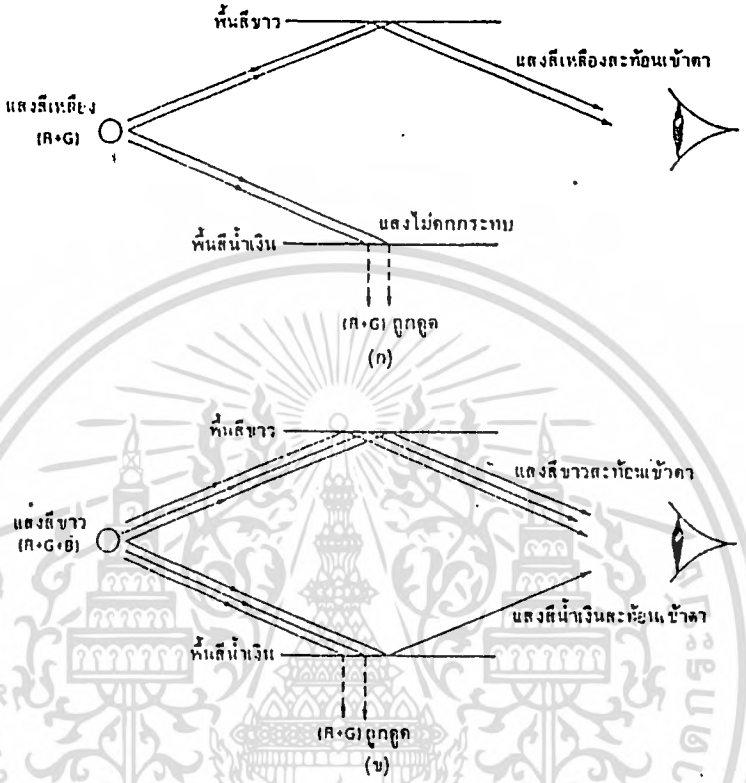
แม่สี แสง (primary colour) มีอยู่ 3 สีคือ สีแดง สีเขียว และสีน้ำเงินเรานิยมเขียนย่อๆดังนี้คือสีแดง เขียนว่า R สีเขียว เขียนว่า G และ สีน้ำเงิน เขียนว่า B

ในรูปที่ 2.6 (ก) แสดงลลำแสงสีแดง เขียว และ น้ำเงิน ผสมกันแล้วตาจะมองเห็นเป็นสีขาว ส่วนรูปที่ 2.6 (ข) ลำแสงสีแดงผสมกับสีเขียวตาจะมองเห็นเป็นสีเหลืองสีที่เกิดขึ้นจากการผสมแม่แสงเราเรียกว่าสีผสม (complementary colour) การผสมสีแสงที่กล่าวมาข้างต้นเป็นการผสมเชิงบวก

การมองเห็นสีของวัตถุเช่น สีทา สีพ่น เกิดจากการผสมสีแสงเชิงลบ (subtroctive mixing) เหตุที่เรียกเช่นนี้ก็เพราะว่าวัตถุเหล่านี้ดูดแสงอื่นๆไว้เสียหมดยกตัวอย่างเช่นเวลาเราเห็นวัตถุชิ้นหนึ่งมีสีน้ำเงินหมายความว่าแสงที่สะท้อนเข้าตาเราเป็นแสงสีน้ำเงินนั่นคือเหตุที่เราได้เห็นเป็นสีน้ำเงินก็เพราะว่าวัตถุอันนั้นดูดสีอื่นไว้หมดเว้นแต่สีน้ำเงินที่ไม่ถูกดูดจึงสะท้อนเข้าตาให้เราเห็นได้ในทำนองเดียวกันสีต่างๆหลายสีผสมกันจะเห็นเป็นสีดำก็เพราะว่ามันดูดแสงทุกสีไว้หมดการดูดหรือลบออกนี้เองที่เราเรียกว่าเป็นการผสมเชิงลบ



รูปที่ 2.6 ตามองเห็นสีซึ่งเกิดจากการผสมเชิงบวก

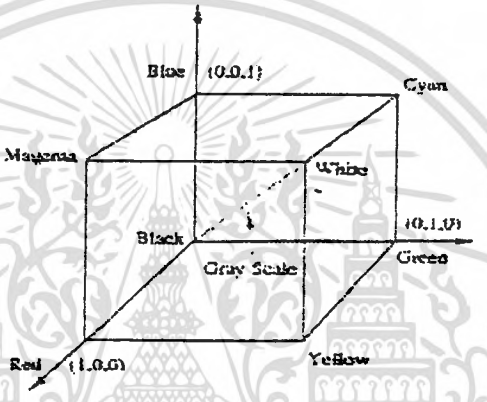


รูปที่ 2.7 ตามองเห็นสีวัตถุซึ่งเกิดการผสมเชิงลบ

จากรูปที่ 2.7 (ข) เมื่อเราฉายแสงสีขาวกระทบพื้นที่ทาสีขาวแสงสีขาวจะสะท้อนทำให้ตามองเห็นเป็นสีขาวเมื่อฉายแสงสีขาวกระทบพื้นทาสีน้ำเงินแสงน้ำเงินจะสะท้อนส่วนแสงสีแดงและเขียวจะถูกดูดไว้หมดตาจึงมองเห็นเป็นสีน้ำเงินในทำนองเดียวกันพิจารณารูปที่ 2.7 (ก) เราฉายแสงสีเหลืองกระทบพื้นสีขาวไม่มีสีใดถูกดูไๆว้แสงสีเหลืองจึงสะท้อนเข้าตาทำให้ตาเห็นเป็นสีเหลืองแทนที่จะเป็นสีขาวแต่เมื่อเราฉายแสงสีเหลือง (ซึ่งประกอบด้วยแสงสีแดงและแสงสีเขียว) กระทบพื้นสีน้ำเงินแสงสีแดงและเขียวจะถูกดูดได้ จึงไม่มีแสงสะท้อนทำให้ตามองเห็นเป็นสีดำ

โมเดล RGB

ในโมเดลนี้สีแต่ละสีจะปรากฏในรูปของสีปฐมภูมิ (แดง , เขียว , น้ำเงิน) โมเดลนี้มีโครงสร้างเป็นแกนคาร์ทีเซียน โดออดิเตต (Cartesian coordinate) มีลักษณะเป็นทรงลูกบาศก์ดังแสดงในรูปที่ 2.9 ค่า Red , Green และ Blue จะอยู่ที่มุมทั้งสาม และค่า คราม , มาเกนตา และ เหลืองจะอยู่ที่มุมทั้งสามที่เหลือสีดำจะอยู่ที่จุดกำเนิด เพื่อความสะดวกเราจะสมมุติให้ค่าสีถูกนอมอลไลซ์ ทั้งสามสีให้มีค่าอยู่ในช่วง 0 ถึง 1 ลูกบาศก์ที่แสดงในรูปที่ 2.9 จึงเป็นลูกบาศก์หนึ่งหน่วย



รูปที่ 2.9 แสดง โมเดลสี RGB

ภาพในโมเดล RGB ประกอบด้วยภาพสามระนาบที่เป็นอิสระจากกันสำหรับแต่ละสีปฐมภูมิ เมื่อป้อนเข้าไปในมอนิเตอร์ที่เป็นแบบ RGB ภาพทั้งสามสีจะรวมตัวกันที่จะภาพกลายเป็นภาพสีผสม ดังนั้นการใช้โมเดล RGB ในการประมวลผลภาพนั้นจะสมเหตุสมผลเมื่อภาพถูกแยกออกโดยธรรมชาติให้อยู่ในเทอมของทั้งสามสีกล้องภาพสีส่วนใหญ่ที่ให้ภาพสีดิจิตอลจะอยู่ในรูปแบบของโมเดล RGB ดังนั้นโมเดลนี้จึงเป็นโมเดลที่สำรคัญมากสในการประมวลผลภาพ

ตัวอย่างที่ดีตัวอย่างหนึ่งของการใช้ประโยชน์โมเดล RGB คือ ในการประมวลผลภาพถ่ายทางอากาศ และภาพถ่ายดาวเทียม ภาพจะได้มาจาก เซนเซอร์ภาพ (image sensor) ที่ทำงานที่หเปกตรัม (spectrum) ต่งกันอย่างเช่นภาพที่ได้จากดาวเทียม LANSAT จะประอกบด้วยภาพ 4 ภาพ แต่ละภาพเป็นภาพของฉากเดียวกันแต่จะต่างสเปกตรัมกัน โดย 2 ใน 4 จะเป็นของสีเขียวกับสีแดงและอีก 2 จะเป็นของอินฟราเรด (infrared) ซึ่งในแต่ละระนาบภาพจะมีควา มหมายทางกายภาพต่างกับซึ่งการประมวลผลจะใช้โมเดล RGB

ปัญหาในการปรับปรุงภาพสีโดยเฉพาะในหน้าของมนุษย์ที่บางส่วนถูกบังด้วยเงามืดการใช้ฮิสโตแกรมอิควอไลเซชัน (Histogram Equalization) เป็นเครื่องมือทางอุดมคติเพราะว่าลักษณะของภาพที่เป็น 3 ระนาบและการทำฮิสโตแกรมอิควอไลเซชันกระทำเฉพาะค่าความเข้มเท่านี้วิธีการทำฮิสโตแกรมอิควอไลเซชันโดยแยกทำในแต่ละระนาบภาพอย่างอิสระส่วนของภาพที่ถูกซ่อนภายใต้เงาก็จะถูกปรับปรุงอย่างไรก็ตามความเข้มในแต่ละระนาบนี้จะแตกต่างกัผลจากการทำฮิสโตแกรมอิคอไลเซชันแยกสามระนาบภาพจะเปลี่ยนแปลงอัตราส่วนของความเข้มระหว่างสีปฐมภูมิซึ่งอัตราส่วนนี้เป็นคุณสมบัติที่สำคัญของสี ทำให้สีของผิวหนังที่ได้จากการทำฮิสโตแกรมอิควอไลเซชัน จะดูไม่เป็นธรรมชาติ โมเดลสีอื่นๆที่จะกล่าวตามมาสามารถที่จะแก้ปัญหานี้ได้

โมเดลสี CMY

ตามที่ ได้กล่าวมาแล้วว่า ครราม , มาเกนตา และเหลืองเป็นสีทุติยภูมิของแสงแต่เป็นสีปฐมภูมิของเม็ดสี (pigment) ผิวที่เคลือบด้วยสีครามเมื่อถูกฉายด้วยแสงขาวแสงสีแดงจะไม่สะท้อนออกมาจากผิววัตถุนั่นแสดงว่าสีครามจะแยกสีแดงออกจากแสงขาว ทำให้เหลือเฉพาะแสงสีเขียวกับสีน้ำเงิน

อุปกรณ์ส่วนมากทำให้ภาพสีบนกระดาษ เช่น พรินเตอร์สีและเครื่องถ่ายเอกสารสีต้องการข้อมูล CMY มิฉะนั้นจะต้องมีตัวเปลี่ยนข้อมูล RGB เป็น CMY ในตัวมัน การเปลี่ยนแปลงนี้ใช้สูตรง่ายๆคือ

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} I \\ I \\ I \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (2.4.1)$$

เราสมมติว่าค่าของสีจะถูกนอร์ มอลไลซ์ให้อยู่ในช่วง 0 ถึง 1 สมการ 2.4.1 แสดงว่าแสงที่สะท้อนจากวัสดุสีครามจะไม่ประกอบด้วยสีแดงจาก C=1 - R ในทำนองเดียวกัน มาเกนตา จะไม่ประกอบด้วยสีเขียวและเหลือง จะไม่ประกอบด้วยสีน้ำเงิน สมการ 2.4.1 สามารถกล่าวได้อีกอย่างหนึ่งว่าเราสามารถได้ค่า RGB จาก CMY ได้อย่างง่ายดายโดยแต่ละค่า CMY จาก 1 ในการประมวลภาพ

โมเดลนี้จะมีประโยชน์ในการเปลี่ยนแปลงข้อมูลเพื่อใช้ในการเชื่อมต่อกับอุปกรณ์ฮาร์ดก็อปปี (Hardcopy)

โมเดล YIQ

YIQ โมเดลใช้ในการแพร่ภาพโทรทัศน์ YIQ เป็นการบันทึกค่า RGB เพื่อใช้ในการส่งที่ให้มีประสิทธิภาพ และรักษาความเข้ากันได้ compatibility) กับมาตรฐานโทรทัศน์ขาวดำ องค์ประกอบ Y ของ YIQ โมเดลจะให้ข้อมูลภาพทั้งหมดที่ต้องการโดยโทรทัศน์ขาวดำการแปลงจาก RGB เป็น YIQ ถูกกำหนดไว้ดังนี้

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (2.4.2)$$

ในการที่จะได้ค่า RGB คืนมาจากค่า YIQ สามารถทำได้โดยการทำเมทริกซ์ผกผัน (Inverse Matrix) YIQ โมเดลถูกออกแบบมาเพื่อให้เข้ารับระบบการมองเห็นของมนุษย์ที่มีปฏิกริยาตอบสนองต่อการเปลี่ยนแปลงของความส่องสว่างมากกว่าการเปลี่ยนแปลง ฮิว หรือความอิ่มตัว ดังนี้ในมาตรฐาน YIQ ต้องการแถบความถี่ปฏิบัติงานที่กว้างหรือจำนวนบิตมาก ในกรณีของภาพดิจิตอลให้กับการแสดงค่า Y และแถบความถี่ปฏิบัติงาน ที่แคบกว่าสำหรับการแสดงค่า I และ Q

บทที่ 3

ทฤษฎีและหลักการลดข้อมูลภาพ

เทคนิคการลดข้อมูล คือ เทคนิคที่ใช้ในการปริมาณของพื้นที่ที่ใช้เก็บข้อมูล ทำให้ลดเวลาการส่งข้อมูลหรือลดแบนด์วิดธ์ที่ต้องใช้ในการส่งข้อมูลไดเทคนิคการลดขนาดข้อมูลมี 2 ชนิด กว้างๆคือ

1.เทคนิคที่ไม่สามารถนำข้อมูลกลับคืนมาได้เหมือนเดิมทุกประการซึ่งมีชื่อเรียกต่างๆ กันเช่น Entropy reduction, Lossy compression

2.เทคนิคที่สามารถนำข้อมูลกลับคืนมาได้ เหมือนเดิมทุกประการ ซึ่งมีชื่อเรียกต่างๆกัน เช่น Noiseless coding, Redundancy reduction,Lossless compression

1.เทคนิคที่ไม่สามารถนำข้อมูลกลับคืนมาได้เหมือนเดิมทุกประการ

การลดขนาดข้อมูลเกิดขึ้น เนื่องจากการตัดทอนข่าวสาร (Information) บางส่วนทิ้งไปในระดับที่ยอมรับได้ ข่าวสารที่ทิ้งไปไม่สามารถนำกลับคืนมาได้ ทำให้ไม่สามารถนำข้อมูลกลับมาได้เหมือนเดิมทุกประการ เช่น การส่งพัลส์ 1 ลูก จากเครื่องส่งผ่านส่งไปยังเครื่องรับถ้าต้องการให้เครื่องรับ รับพัลส์ได้เหมือนเดิมทุกประการ ต้องใช้แบนด์วิดธ์กว้างถึงอินฟีนิตี้ เพื่อส่งสเปคตรัมทุกๆฮาโมนิค แต่จะส่งเพียงฮาโมนิคแรกๆ เพียงไม่กี่ฮาโมนิคซึ่งเป็นสเปคตรัมที่พลังงานของพัลส์ส่วนใหญ่สะสมอยู่ ทางเครื่องรับสามารถรับพัลส์ได้ มีรูปร่างใกล้เคียงกับพัลส์จริงๆ เท่านั้น แต่เพียงพอที่จะแยกแยะได้ว่าเป็นพัลส์หรือไม่การลดขนาดข้อมูลเกิดขึ้น เนื่องจากการจำกัดสเปคตรัมให้เหลือเพียงพอที่เครื่องรับสามารถแยกแยะได้นั่นคือ ใช้แบนด์วิดธ์ ในการส่งพัลส์ลงเทคนิคนี้เหมาะสมสำหรับการใช้งานกับข้อมูลประเภทภาพและเสียงซึ่งยอมให้การตัดทอนข้อมูลบางส่วนได้

2.เทคนิคที่สามารถนำข้อมูลกลับคืนมาได้เหมือนเดิมทุกประการ

เราอาจคิดว่า ข้อมูล(data) ประกอบไปด้วยข่าวสารส่วนที่ซ้ำ (Redundancy) การลดขนาดข้อมูลทำได้โดยการตัดส่วนที่ซ้ำหรือลดส่วนที่ซ้ำให้เหลือน้อยที่สุด ซึ่งส่วนที่ซ้ำสามารถนำกลับคืนมาได้ นั่นคือข้อมูลสามารถนำกลับคืนมาได้เหมือนเดิมทุกประการ เช่นถ้าต้องการส่งข้อมูล'0000001240001111' ไปให้เครื่องรับโดยกำหนดให้ส่งข้อมูลครั้งละตัวตามปรกติต้องส่งข้อมูล17

ครั้ง แต่เราอาจนำข้อมูลส่วนที่ซ้ำมาเข้ารหัส โดยกำหนดข้อมูลพิเศษขึ้นมาตัวหนึ่งเพื่อ เป็นสัญลักษณ์บอกกับเครื่องรับว่าถ้าได้รับข้อมูลพิเศษตัวนี้ข้อมูลที่ตามมา 2 ตัวคือ จำนวนข้อมูลที่ซ้ำและค่าของข้อมูลตามลำดับ ถ้ากำหนดให้ข้อมูลพิเศษคือ A ข้อมูล '000000 ' เข้ารหัสเป็น 'A60' ข้อมูล '0000' เข้ารหัส เป็น '440' ข้อมูล '1111' เข้ารหัสเป็น 'A41' ข้อมูลทั้งหมดเมื่อผ่านการเข้ารหัสแล้วมีลักษณะดังนี้ 'A60124A40A41' ซึ่งส่งข้อมูลเพียง 12 ครั้ง เครื่องรับก็สามารถ รับข้อมูลได้เหมือนเดิมทุกประการ และประหยัดเวลาที่ใช้ส่งได้ 5 ครั้ง การเข้ารหัสข้อมูลในบักษณะที่เรียกว่า Run Length Encoding เทคนิคนี้เหมาะสำหรับข้อมูลประเภทฐานข้อมูลและข้อมูลประเภทตัวอักษรเป็นต้น

การลดขนาดข้อมูลสามารถอธิบายได้โดย ทฤษฎีข่าวสาร (Information Theory) มีศัพท์หลายคำที่เกี่ยวข้องกับเทคนิคการลดขนาดข้อมูลเช่น ข่าวสาร ข่าวสารเฉลี่ย (entropy) และส่วนที่ซ้ำ

3.1 ข่าวสาร ปริมาณข่ายวสารเฉลี่ย และส่วนที่ซ้ำ

ข่าวสาร (Information) ในความหมายของ ทฤษฎีข่าวสาร คือปริมาณความไม่แน่นอนของเหตุการณ์ (measure of uncertainty) เช่น สำหรับเหตุการณ์ (event) เกิดค่อนข้างแน่นอนเหตุการณ์นั้นมีข่าวสารน้อย สำหรับเหตุการณีที่เกิดไม่มากครั้ง แสดงว่าเหตุการณ์นั้นมีข่าวสารมาก

การวัดปริมาณข่าวสาร ของเหตุการณ์คำนวณได้จาก

$$I_i = - \log_2 P_i \text{ บิท}$$

โดยที่ P_i คือความน่าจะเกิด (Probability) ของ เหตุการณ์ I

ปริมาณข่าวสาร ของเหตุการณ์ I บอกเราว่าควรจะใช้เนื้อที่ในการเก็บเหตุการณ์ i เท่าไร ตัวอย่างเช่น เหตุการณ์ x มีความน่าจะเกิดเท่ากับ 1/8 แล้วแสดงว่า เนื้อที่ที่ใช้เก็บเหตุการณ์ x คือ 3 บิท

ปริมาณข่าวสารเฉลี่ยของเหตุการณ์ทั้งหมดหรือ entropy คำนวณได้จาก

$$H = - (P_1 \log_2 P_1 + P_2 \log_2 P_2 + \ldots + P_m \log_2 P_m)$$

โดยที่ M คือเหตุการณ์ที่เป็นไปได้ทั้งหมด

ปริมาณข่าวสารเฉลี่ย บอกเราว่าควรจะเก็บข้อมูลทั้งหมดด้วยเนื้ออย่างน้อยที่สุดเท่าไร เช่น ข้อมูล ' e ' มีความน่าจะเกิดเท่ากับ 1 / 16 ดังนั้นข้อมูล 'eeeee ' ใช้เนื้อที่ในการเก็บเท่ากับ 20 บิท ข่าวสารเฉบี่ย มีเงื่อนไขที่น่าสนใจ 2 กรณีคือ

1. เมื่อ M = 1 คือมีเหตุการณ์เดียว ดังนั้น $P_1 = 1$ ทำให้ H = 0 หมายความว่าไม่มีข่าวสารเฉลี่ยของเหตุการณ์ที่ทราบแน่นอนแล้ว
2. เมื่อ $P_1 = 0$ ทำให้ H = 0 หมายความว่าไม่มีข่าวสารเฉบี่ยของเหตุการณ์ที่เป็นไปไม่ได้

นั่นคือ $H = - P \log_2 P = 0$ (เมื่อ P-> 0)

นิยามของส่วนที่ซ้ำ คือ ผลต่างระหว่างปริมาณของข้อมูลและข่าวสารเฉลี่ย

$$R = \log_2 M - H$$

ส่วนที่ซ้ำเกิดขึ้น เนื่องจากการกระจายของข้อมูลไม่เท่ากัน คือความน่าจะเกิดของข้อมูลทุกตัวไม่เท่ากัน ถ้าการกระจายของข้อมูลเท่ากัน แล้ว $H = \log_2 M$ ซึ่ง H จะมีค่าสูงสุดดังนั้น R= 0 แต่ ข้อมูล โดยทั่วไปมักกระจายไม่เท่ากัน

ดังนั้นเทคนิคการลดขนาดข้อมูลจะใช้ประโยชน์ จากการกระจายไม่เท่ากันของข้อมูลนี้ เพื่อลดขนาดของข้อมูล โดยการเข้ารหัสข้อมูลด้วยรหัสที่เท่ากับ หรือ ต่างจาก entropy น้อยที่สุดตัวอย่างของรหัสที่ใช้ในการเข้ารหัสข้อมูล คือ ฮัฟแมน (Huffman)

3.2 อัลกอริทึมของฮัฟแมน

ผู้ที่คิดวิธีการลดขนาดข้อมูลนี้ก็คือ นาย D.A.Huffman ศิษย์เก่าจาก MIT เขาได้เสนอกระบวนการ " เข้ารหัส " แบบนี้มาตั้งแต่ต้นทศวรรษ 1950 สำหรับหลักการสำคัญของการลดขนาดข้อมูลนี้ก็คือ การลดจำนวนบิตที่ใช้มอนตัวอักษรที่มีการใช้บ่อยๆ และเพิ่มจำนวนบิตที่ใช้แทนตัวอักษรที่มีการใช้น้อยๆ

การทำงานชนิดนี้อยู่บนพื้นฐานความจริงที่ว่า ในประโยคหรือในไฟล์ข้อมูลที่เป็นภาษาอังกฤษหนึ่งๆจะมีการใช้อักษรภาษาอังกฤษซ้ำไปซ้ำมาเสมอ และบางตัวก็ไม่ได้ใช้บ่อยสักเท่าไร ถ้าภาษาอังกฤษตัวหนึ่งๆถูกแทนด้วยเลขฐานสองจำนวน 8 บิต แล้วเราพบว่าในไฟล์ข้อมูลของเรามีตัวอักษร " a " มากที่สุดและพบว่าตัวอักษร " z " เป็นตัวอักษรที่พบน้อยตามวิธีการของ ฮัฟแมน เราก็อาจจะแทนอักษร " a " ด้วยรหัส 4 บิต และให้อักษณ " z " แทนด้วยรหัส 14 บิต เป็นต้น

ปัจจุบันได้มีโปรแกรมแอพพลิเคชันหลายๆชนิดที่ได้ประยุกต์ใช้วิธีการลดขนาดของ ฮัฟแมน นี้ เช่น การลดขนาดแบบ MNP-5 ของ โมเด็ม ซึ่งได้ใช้การลดขนาดข้อมูล ฮัฟแมน แบบไดนามิก การเข้ารหัสแบบ Shannon-Fano การเข้ารหัสบางส่วนของโปรแกรม PKZIP การทำงานตอนท้ายของการลดขนาดแบบ JPEG เป็นต้น

การสร้างรหัสของ ฮัฟแมน ก่อนอื่นอ่านข้อมูลหนึ่งรอบ เพื่อหาความถี่ของแต่ละข้อมูลจากนั้นเรียงลำดับของข้อมูลโดยข้อมูล ที่มีความถี่มากอยู่ด้านหนึ่ง และข้อมูลที่มีความถี่ น้อยอยู่ด้านหนึ่ง และกำหนดให้ข้อมูลแต่ละตัวคือโหนด (node) ใดๆของทรี (tree) แล้วนำมาสร้าง ทรี ตามวิธีต่อไปนี้

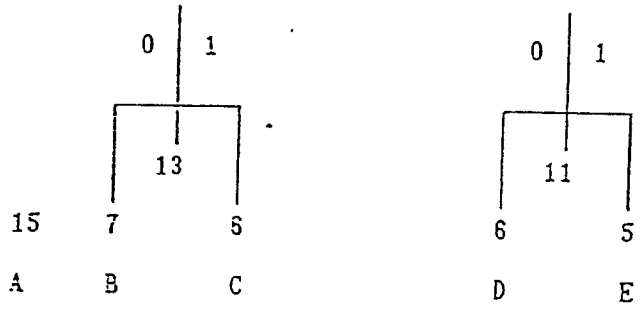
1. นำเอาข้อมูลหรือโหนดที่มีความถี่น้อยที่สุดมาทำเป็น โหนดลูก (child node) ของ โหนดพ่อ (parent node) ซึ่งความถี่ของโหนดพ่อนี้จะเท่ากับผลรวมของความถี่ของโหนดลูกทั้งสอง
2. โหนดพ่อ จะถูกนำไปรวมกับโหนด อื่นๆที่เหลือและโหนดลูกที่ใช้แล้วทั้งสองจะไม่ถูกนำมาใช้อีก
3. โหนดลูก หนึ่งจะถูกกำหนดให้เป็นบิท 0 และอีกโหนดเป็นบิท 1
4. กระทำซ้ำตั้งแต่ 1 จนกระทั่งเหลือ โหนดเดียว เรียกว่า ราก (root) ของทรี

ตัวอย่างเช่น ให้ข้อมูลมีลักษณะดังนี้

"AEDBAACBADCABECADACEDADACAEBABABEADABAD"

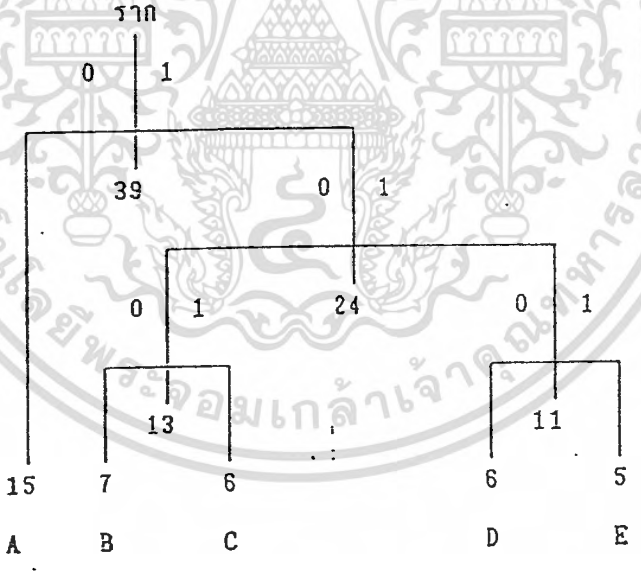
ความถี่	15	7	6	6	5
	A	B	C	D	E

โหนดที่มีน้ำหนักน้อยที่สุด คือ D และ E ทั้งสองโหนด จะถูกเชื่อมให้เป็น โหนดพ่อซึ่งมีน้ำหนัก11 แล้วเรากำหนดให้ โหนด D มีค่าเป็น 0 และโหนด E มีค่าเป็น 1 รอบที่สอง B และ C จะถูกนำมาเชื่อมต่อเป็น โหนดพ่อ ใหม่อีก ซึ่งมีน้ำหนักเป็น 13 ซึ่งจะเป็นดังรูปที่ 3.1



รูปที่ 3.1 ฮัฟแมนทรี เมื่อผ่านไป 2 รอบ

รอบต่อมา 2 โหนด ที่มีน้ำหนักน้อยที่สุด คือ โหนดพ่อ ของ B/C กับ D/E จะถูกเชื่อมเข้าเป็นโหนดพ่อ หลังจากนั้นจะเหลือโหนดเพียง 21 โหนด เมื่อนำมารวมกันจะได้ รากของทรีซึ่งมีลักษณะดังรูป 3.2



รูปที่ 3.2 ผลรวมของฮัฟแมนทรี

การหารหัสของข้อมูลทำได้โดยการ เดินทางจากรากไปยังข้อมูลตัวนั้นผ่าน โหนดใดให้จำค่าของโหนดนั้นไว้ รหัสของข้อมูลคือ ค่าของโหนดที่จำไว้ตามลำดับ ดังนั้นรหัสของข้อมูลคือ

A = 0

B = 100

C = 101

D = 110

E = 111

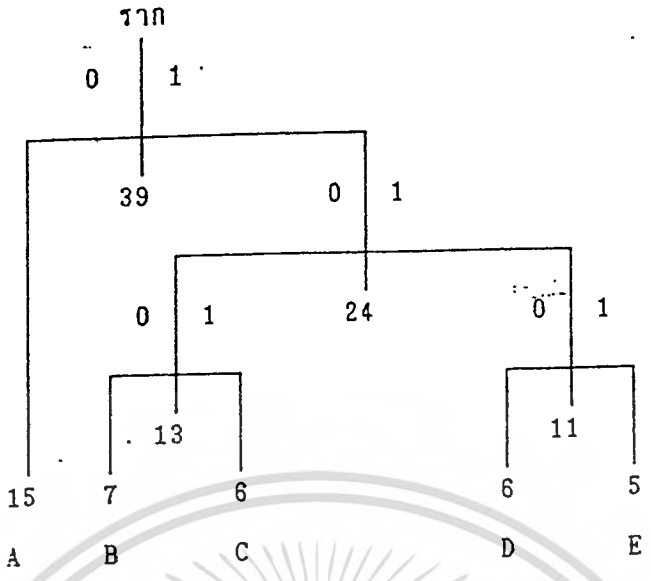
หลังจากได้รหัสของข้อมูลแล้วจึงอ่านข้อมูลอีกรอบเพื่อลดขนาดข้อมูลโดยการเข้ารหัสข้อมูลทั้งหมดโดยการแทนข้อมูลด้วยรหัสของข้อมูลดังนั้นจะได้รหัสของข้อมูลมีลักษณะเรียงเป็นลำดับเช่นเดียวกับลำดับของข้อมูลดังรูป 3.3

"0 111 110 100 0 0 101 ....." รหัสของข้อมูลมีลักษณะเป็นบิท

"A E D B A A C ...." ข้อมูลมีลักษณะเป็นไบท์

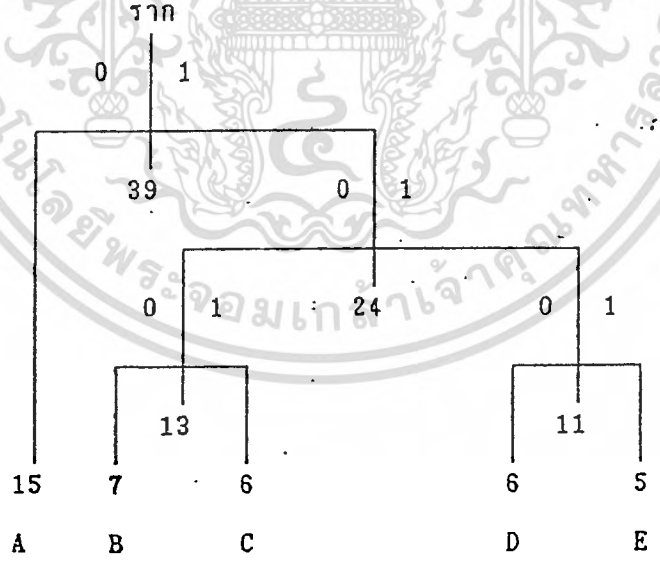
รูปที่ 3.3 ตัวอย่างการเข้ารหัสข้อมูล

สำหรับการขยายขนาดข้อมูลกลับทำได้โดย
ที่ใช้ในการลดขนาดข้อมูล โดยจะอ่านรหัสเข้ามาทีละบิทและเริ่มเดินทางจากรากไปยังกิ่งที่มีค่าเท่ากับรหัส 1 บิทที่อ่านเข้ามา และกระทำเช่นนี้ต่อไปจนกระทั่งไม่มี โหนดย่อยลงไปอีกก็จะได้ข้อมูลกลับคืนมาหนึ่งตัวหลังจากนั้น เริ่มเดินทางจากรากใหม่ จนกระทั่งได้ข้อมูลกลับคืนมาจนครบหมด จากรหัสของข้อมูลข้างต้น บิทตัวแกรคือ 0 ดังนั้นสามารถถอดรหัสออกมาได้เป็น A ดังรูปที่ 3.4



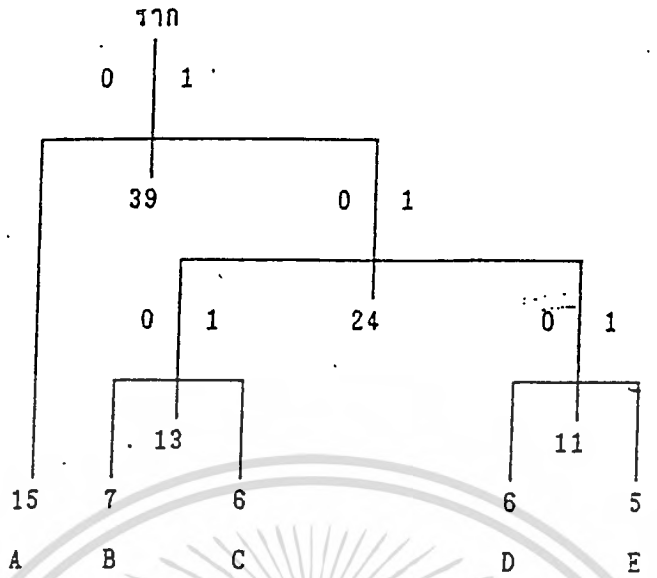
รูปที่ 3.4 แสดงการถอดรหัสเมื่อได้รับรหัส 0

จากนั้นอ่านรหัสข้อมูลบิทที่ 2 คือ 1 เมื่อเดินทางจากรากไปยังทรี จะมีลักษณะดังรูป 3.5



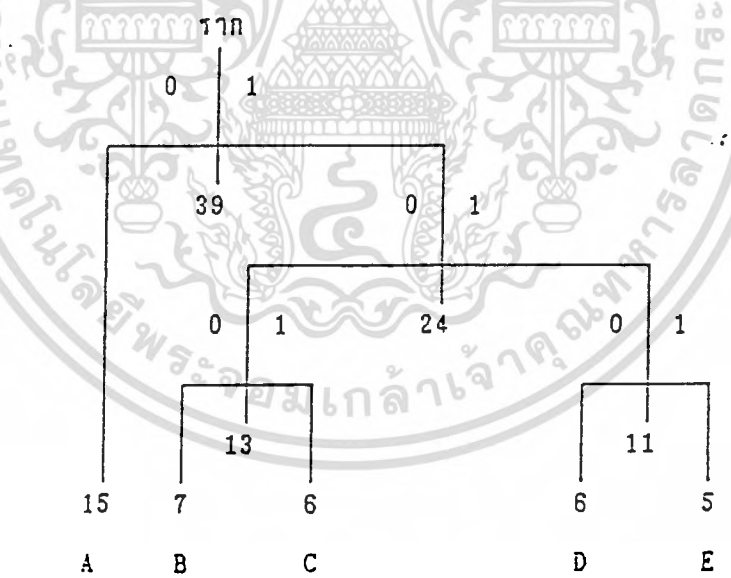
รูปที่ 3.5 แสดงการถอดรหัสเมื่อได้รับรหัส 1

จากนั้นอ่านรหัสข้อมูลบิทที่ 3 คือ 1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 3.6



รูปที่ 3.6 แสดงการถอดรหัสเมื่อได้รับรหัส 11

จากนั้นอ่านรหัสข้อมูลบิทที่ 1 คือ 1 เมื่อเดินทางจากรากไปยังทรีจะมีลักษณะดังรูป 3.7



รูปที่ 3.7 แสดงการถอดรหัสเมื่อได้รับรหัส 111

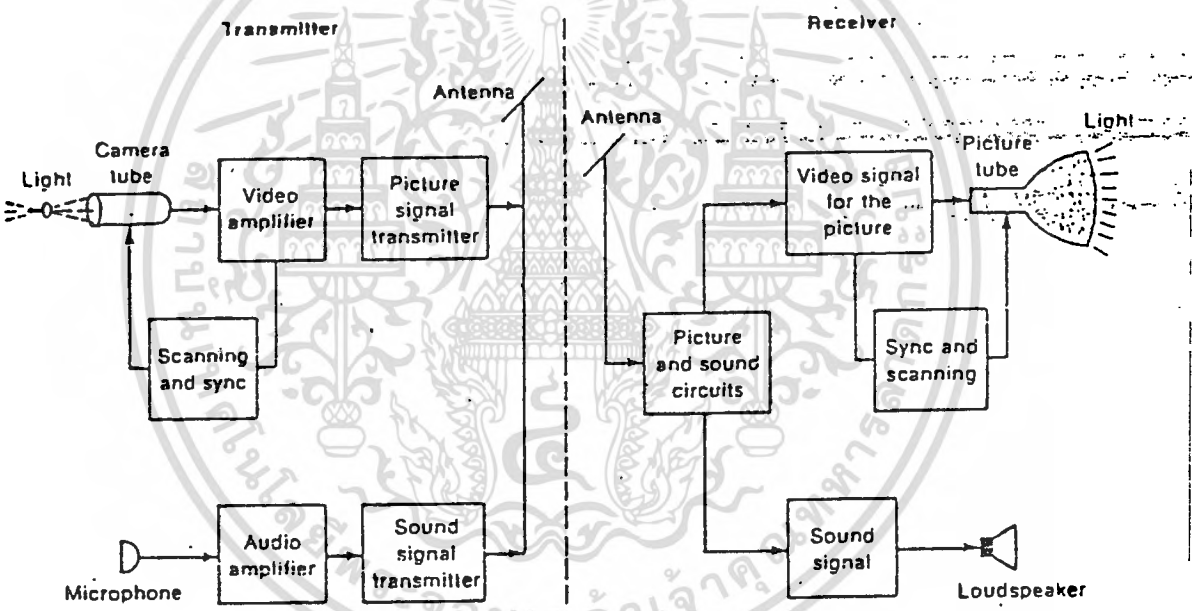
ก็จะสามารถถอดรหัสได้เป็น 10 และจะกระทำเช่นนี้ต่อไปจนกระทั่งไม่มีรหัสข้อมูลเหลือ แล้วจะได้ข้อมูลกลับมาเหมือนเดิมทุกประการ

บทที่ 4

ทฤษฎีการส่งและรับโทรทัศน์ระบบ NTSC

4.1ระบบเบื้องต้นของการส่งและรับโทรทัศน์

โทรทัศน์ในปัจจุบันนี้ใช้ระบบซึ่งทำงานด้วยระบบอีเลคทรอนิคส์ล้วนๆ หลอดถ่ายภาพ เช่น Image orthicon, Plumbicon, Vidicon ได้นำมาใช้ทำกล้องถ่ายโทรทัศน์ ส่วนด้านรับ ใช้หลอดภาพ (cathode ray tube) หลอดเหล่านี้ทำงานด้วยการสะแกนของลำอีเลคตรอนดังแสดงให้เห็นการทำงานด้วยบล๊อกไดอะแกรม ในรูปที่ 4.1

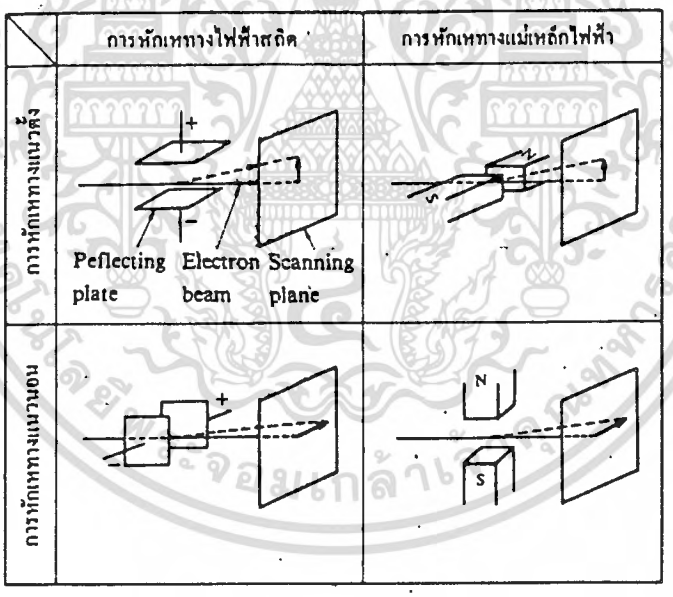


รูปที่ 4.1 ส่วนสำคัญทางด้านส่งและด้านรับ

4.2 การสะแกนและเรื่องที่เกี่ยวข้อง

ภาพบนจอหลอดภาพของเครื่องรับโทรทัศน์สีโดยทั่วไป จะประกอบด้วยเส้นขวางเล็กๆ แนวนอนเป็นจำนวนมาก ซึ่งแต่ละเส้นเหล่านี้ มีทั้งส่วนที่ดำสนิทหรือมีสีเข้ม ส่วนที่ดำจางหรือมีสีจางและส่วนที่สว่างมากปะปนกันอยู่ เส้นขวางเล็กๆ ในแนวนอนเหล่านี้ มีชื่อเรียกว่า เส้นสะแกน เส้นเหล่านี้ประกอบด้วยจุดเล็กๆ ซึ่งมีทั้งมืดและสว่างปะปนกันอยู่ ภาพที่ปรากฏบนจอหลอดภาพ

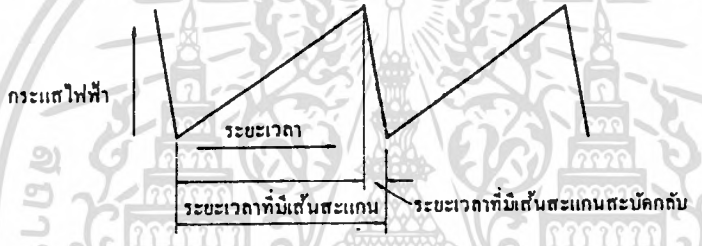
ประกอบด้วยจุดเล็กๆ ที่มีระดับของความสว่างแตกต่างกันเป็นจำนวนมาก จุดเล็กๆ เหล่านี้ เรียกว่า ส่วนประกอบของภาพ หรือ picture element ซึ่งมีส่วนสัมพันธ์กับความละเอียดของภาพเช่นเดียวกับจุดดำหรือจุดสีเล็กๆในรูปภาพของสิ่งตีพิมพ์ ภาพที่เห็นบนจอหลอดภาพ จะมองดูละเอียดน่าดูหากจำนวนจุดเล็กๆ หรือจำนวนเส้นสะแกนแนวนอนมากเพียงพอ ด้วยเหตุนี้ โทรทัศน์ระบบยุโรป ซึ่งมีจำนวนเส้นสะแกน 625 เส้น ต่อภาพ จึงให้ภาพที่มองดูละเอียดกว่าโทรทัศน์ระบบอเมริกัน ซึ่งจำนวนเส้นสะแกนเพียง 525 เส้น ต่อภาพ เท่านั้น อย่างไรก็ตาม ภาพที่เห็นบนจอหลอดภาพจะมองละเอียดหรือหยาบไม่น่าดูอย่างไรนั้น ยังขึ้นอยู่กับส่วนประกอบอีกหลายอย่าง เช่น ความสว่างของภาพและระยะทางที่มองดูภาพ เป็นต้น สำหรับโทรทัศน์ระบบอเมริกัน ซึ่งมีจำนวนเส้นสะแกนน้อยกว่าจำนวนเส้นสะแกนของโทรทัศน์ระบบยุโรป อันอาจทำให้มองเห็นภาพมีความละเอียดน้อยกว่า แต่ถ้าหากมองดูภาพในระยะทางห่างประมาณสี่ถึงแปดเท่าของความสูงของภาพแล้ว ก็จะรู้สึกว่าเป็นภาพพอใช้ได้เหมือนกัน



รูปที่ 4.2.1 ทฤษฎีของการหักเหทางไฟฟ้าสถิต และทางแม่เหล็กไฟฟ้า

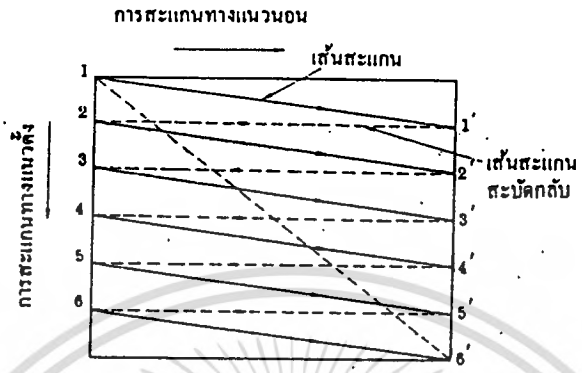
จุดที่เห็นสว่างในจอหลอดภาพของเครื่องรับโทรทัศน์ เกิดขึ้นเพราะอีเล็คตรอนที่หลุดออกไปจากแคโทด และถูกดึงดูดให้วิ่งเป็นลำไปกระทบแอโนดหรือจอหลอดภาพ ซึ่งฉาบวัสดุเรืองแสง

บางชนิดเอาไว้ จุดที่มีการกระทบกัน ก็จะมองเห็นเป็นจุดสว่างขึ้นที่จอ การสะแกนก็คือ การทำให้จุดสว่างนี้เคลื่อนที่ไปในจังหวะที่ต้องการ ซึ่งในเรื่องของเครื่องรับโทรทัศน์ ก็ต้องการให้จุดสว่างนี้เคลื่อนที่ไปในแนวนอนและแนวตั้ง โดยอาศัยความเข้มของสนามแม่เหล็กเข้าช่วยเหลือ ทำให้เกิดการดึงดูดหรือผลักกันกับอีเล็คตรอน ในหลักการ การทำให้เกิดการดึงดูดหรือการผลักกันกับอีเล็คตรอนนี้ อาจทำได้โดยวิธีการหักเหของไฟฟ้าสถิต (electrostatic deflection) หรือวิธีการหักเหของแม่เหล็กไฟฟ้า (electromagnetic deflection) ตามที่แสดงไว้ในรูปที่ 4.2.1 ซึ่งวิธีการหลังนี้ เป็นที่นิยมกันมากในทางปฏิบัติ สนามแม่เหล็กนี้เกิดขึ้นโดยการปล่อยกระแสไฟฟ้ารูปฟันเลื่อย ตามที่แสดงไว้ในรูปที่ 4.2.2 ให้ไหลฝ่านขดลวดของการหักเห (deflection coil) ที่พันอยู่รอบๆ

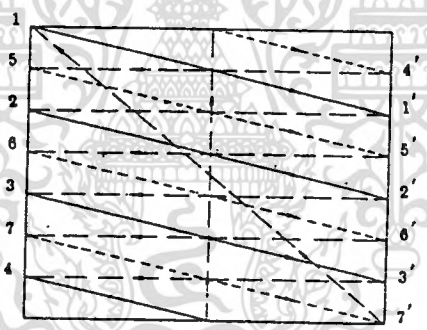


รูปที่4.2.2 รูปร่างของกระแสรูปฟันเลื่อย

คอหลอดภาพ ซึ่งมีอยู่สองชุดด้วยกันคือ ขดลวดที่พันอยู่รอบคอหลอดภาพในแนวนอนชุดหนึ่ง และขดลวดที่พันอยู่รอบคอหลอดภาพในแนวตั้งอีกชุดหนึ่ง สำหรับโทรทัศน์ระบบยุโรป ความถี่ของการกระแสรูปฟันเลื่อยที่ไหลผ่านขอลวดของการหักเหในแนวตั้ง จะมีค่าเพียง 50 เฮิร์ทซ์ เท่านั้น โดยปกติ การสะแกนจะเริ่มต้นขึ้นโดยการทำให้จุดสว่างบนจอหลอดภาพเคลื่อนที่จากซ้ายมือด้านบนของจอไปทางขวามือในแนวนอน ซึ่งเมื่อถึงตำแหน่งขวามือสุด ก็จะถูกเบนต่ำลงเล็กน้อย อันเป็นผลจากการที่มีกระแสรูปฟันเลื่อยไหลผ่านขดลวดของการหักเหในแนวตั้ง แล้วก็จะกลับไปตั้งต้นใหม่ทางซ้ายมือเพื่อเคลื่อนที่มาทางขวามือในแนวนอนอีก เป็นอยู่เช่นนี้เรื่อยๆ จนกระทั่งจุดสว่างนั้นไปถึงตำแหน่งขวามือสุดของจอหลอดภาพ จึงเป็นอันเสร็จสิ้นการสะแกนภาพนิ่งภาพหนึ่ง หรือเรียกกันว่า เฟรมหนึ่ง ตามที่แสดงไว้ในรูปที่ 4.2.3



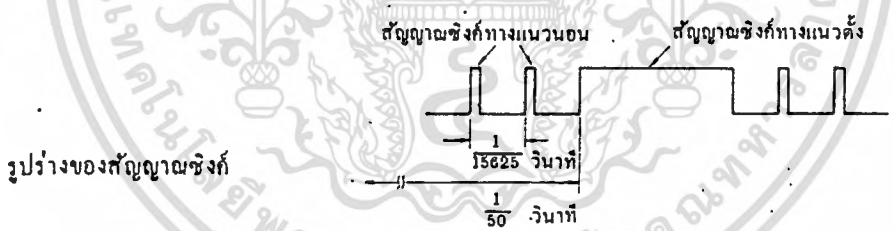
รูปที่4.2.3 การสะแกนจากซ้ายไปขวาและจากบนลงล่าง



รูปที่ 4.2.4 การสะแกนไขว้กัน (interlace scanning)

หลังจากนั้น ลำอีเล็คตรอนก็จะกลับไปตั้งต้นใหม่ทางซ้ายมือด้านบนสุดของจอหลอดภาพอีก เพื่อสะแกนภาพนิ่งอันดับถัดไป อย่างไรก็ตาม เพื่อลดอาการกระพริบของภาพการสะแกนภาพนิ่งแต่ละภาพ มักนิยมจัดทำสองครั้งในแบบของการสะแกนไขว้กัน ซึ่งเรียกว่า interlace scanning ตามที่แสดงไว้ในรูปที่4.2.4 โดยกำหนดให้ภาพนิ่งหนึ่งเฟรม (frame) ประกอบด้วยภาพนิ่งสองฟิลด์ (field) และเริ่มต้นด้วยการสะแกนภาพนิ่งฟิลด์เส้นคี่ก่อน เมื่อเสร็จสิ้นถึงตำแหน่งขวามือล่างสุดของจอหลอดภาพแล้ว จึงกลับไปตั้งต้นใหม่ทางด้านซ้ายมือบนสุดของจอ แล้วเริ่มต้นสะแกนภาพนิ่งฟิลด์เส้นคู่ต่อไป จนถึงตำแหน่งขวามือล่างสุด หลังจากนั้น จึงจะเริ่มต้นสะแกนภาพนิ่งอันดับอื่นต่อ

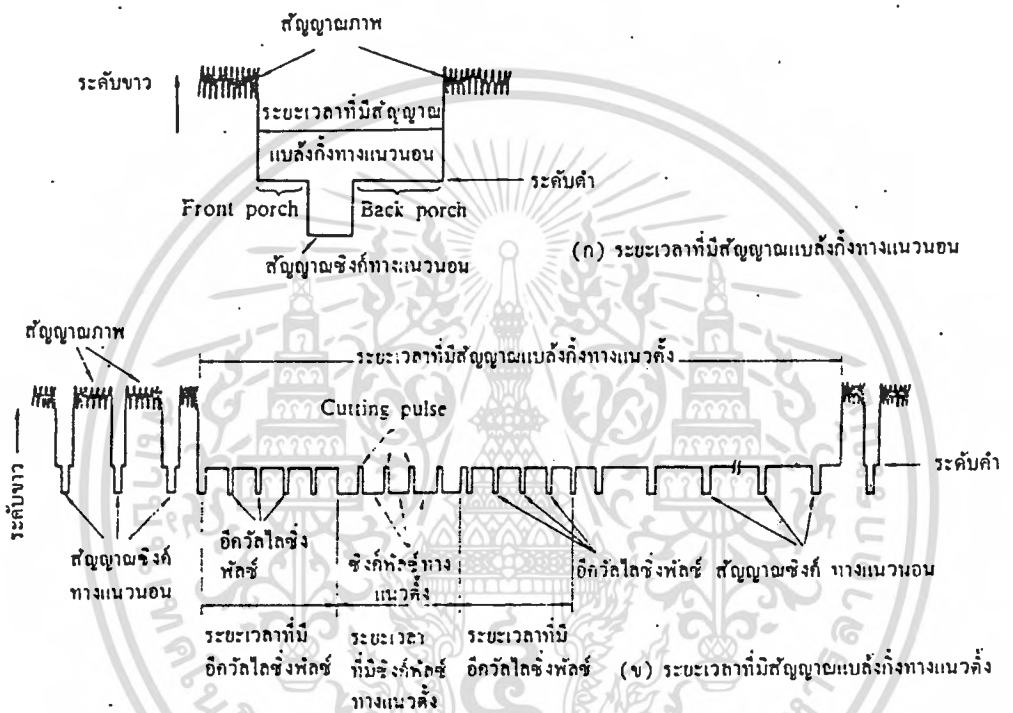
ไปใหม่ ฉะนั้น ภาพนิ่งหนึ่งภาพหรือภาพนิ่งหนึ่งเฟรม จึงประกอบด้วยฟิลด์เส้นสะแกนเส้นคี่และฟิลด์เส้นสะแกนเส้นคู่ สำหรับโทรทัศน์ระบบยุโรป ซึ่งใช้เส้นสะแกน 625 เส้น ต่อภาพ และหนึ่งภาพ ต่อวินาที นั้น ภาพนิ่งแต่ละภาพหรือแต่ละเฟรมจะประกอบด้วยเส้นสะแกนแนวนอน 625 เส้น ภาพนิ่งแต่ละฟิลด์ จะมีเส้นสะแกนแนวนอนครึ่งหนึ่งของ 625 เส้น หรือ 312 ½ เส้น ภาพนิ่ง แต่ละภาพนี้ จะเกิดขึ้นภายในระยะเวลา 1/25 วินาที ความถี่ของกระแสรูปฟันเลื่อยที่ใช้ในการหักเหทางแนวนอน ซึ่งในระยะเวลา 1/25 วินาที จะเกิดเส้นสะแกน 625 เส้น จะมีค่า (625) (25) หรือ 15,600 เฮิร์ทซ์ ส่วนความถี่ของกระแสรูปฟันเลื่อยที่ใช้ในการหักเหทางแนวตั้ง ซึ่งใช้เวลาในการสะแกนจากบนสุดมาล่างสำหรับฟิลด์หนึ่งๆ เพียง 1/50 วินาที จะมีค่า 50 เฮิร์ทซ์ การสะแกนภาพนิ่งตามที่กล่าวมาแล้วนี้ จะกระทำติดต่อกันไปเรื่อยๆ โดยจะมีจำนวนภาพนิ่งหรือจำนวนเส้นสะแกนต่อภาพ กับจำนวนภาพต่อวินาทีแตกต่างกันไปตามแต่ชนิดของระบบของระบบโทรทัศน์ที่ใช้ ภาพที่มาปรากฏบนจอหลอดภาพของเครื่องรับโทรทัศน์ จึงมีผลคล้ายกับการฉายภาพนิ่ง ซึ่งแต่ละภาพมีความแตกต่างกันบ้างเพียงเล็กน้อย เป็นจำนวนหลายๆ ภาพต่อหนึ่งวินาที ด้วยเหตุที่สายตาของคนเรามีคุณลักษณะพิเศษในเรื่องของ persistence of vision จึงทำให้ผู้ชมโทรทัศน์สามารถมองเห็นภาพบนจอหลอดภาพของเครื่องรับโทรทัศน์ เป็นภาพเคลื่อนไหวติดต่อกันไปตลอดเวลา



รูปที่ 4.2.5 รูปร่างของสัญญาณซิงค์

เรื่องที่สำคัญอีกอย่างหนึ่งของการส่งและการรับโทรทัศน์ก็คือ จะต้องสามารถหาวิธีการซึ่งทำให้การสะแกนของภาพที่เกิดขึ้นในกล้องโทรทัศน์นั้น เกิดขึ้นพร้อมกันกับการสะแกนของภาพที่จอหลอดภาพของเครื่องรับโทรทัศน์ หรือทำให้ความถี่ของกระแสรูปฟันเลื่อยของวงจร หักเหทางแนวนอนและแนวตั้งทางกล้องโทรทัศน์เท่ากันตลอดเวลา กับความถี่ของวงจรหักเหทางแนวนอนและแนวตั้งทางจอหลอดภาพของเครื่องรับโทรทัศน์ หากความถี่ของกระแสรูปฟันเลื่อยในวงจร

ทางเครื่องรับโทรทัศน์ ก็จะพบว่า ภาพจะล้มหรือไม่มีภาพทางเครื่องรับโทรทัศน์ การทำให้ ความถี่ของกระแสรูปฟันเลื่อยทางด้านเครื่องส่งโทรทัศน์เท่ากันตลอดเวลา กับความถี่ของกระแสรูปฟันเลื่อยทางด้านเครื่องรับโทรทัศน์นี้ เรียกว่า เกิดการเข้าจังหวะ (synchronization) ขึ้น ในทางปฏิบัติสถานีโทรทัศน์จะต้องส่งสัญญาณชนิดหนึ่งเรียกว่า สัญญาณซิงค์ (synchronizing signal หรือ sync pulse signal) ไปพร้อมกับสัญญาณภาพ ตามที่แสดงไว้ในรูปที่4.2.5 และรูปที่ 2.4.6 สัญญาณซิงค์นี้จะประกอบด้วยสัญญาณซิงค์ทางแนวนอน (horizontal synchronizing signal)



รูปที่ 4.2.6 รูปร่างของสัญญาณซิงค์ที่ใช้ในการส่งโทรทัศน์

ซึ่งมีความถี่ 15,625 เฮิร์ทซ์ หรือจะมี sync pulse ครั้งหนึ่งในทุกๆ ครั้งที่มีเส้นสะแกนในแนวนอนกับสัญญาณซิงค์ทางแนวตั้ง (vertical synchronizing signal) ซึ่งมีความถี่ 50 เฮิร์ทซ์ หรือจะมี sync pulse ครั้งหนึ่งในขณะที่มีการสะแกนฟิลด์เส้นคี่หรือฟิลด์เส้นคู่เสร็จสิ้นลง สัญญาณซิงค์เหล่านี้จะส่งไปพร้อมๆกับสัญญาณภาพในช่วงระยะเวลาของเส้นสะแกนสะบัดกลับ หรือช่วงระยะเวลาที่เส้นสะแกนกำลังหันกลับไปเริ่มต้นใหม่ (flyback period)

ในทางปฏิบัติสถานีโทรทัศน์ขาวดำจะต้องส่งสัญญาณต่างๆ หลายอย่างออกอากาศไปให้เครื่องรับโทรทัศน์ เพื่อทำให้เกิดภาพขาวดำที่จอหลอดภาพของเครื่องรับโทรทัศน์ในลักษณะเดียว

กันและพร้อมกันกับการสะแกนภาพของกล้องโทรทัศน์ สัญญาณต่างๆ สำหรับทำให้เกิดภาพขาวดำเหล่านี้ แสดงไว้แล้วในรูปที่ 4.2.6 ซึ่งประกอบด้วย

- สัญญาณภาพ (video signal)

- สัญญาณแบล้งกิ้ง (blanking signal)

- สัญญาณซิ้งค์ (synchronizing signal)

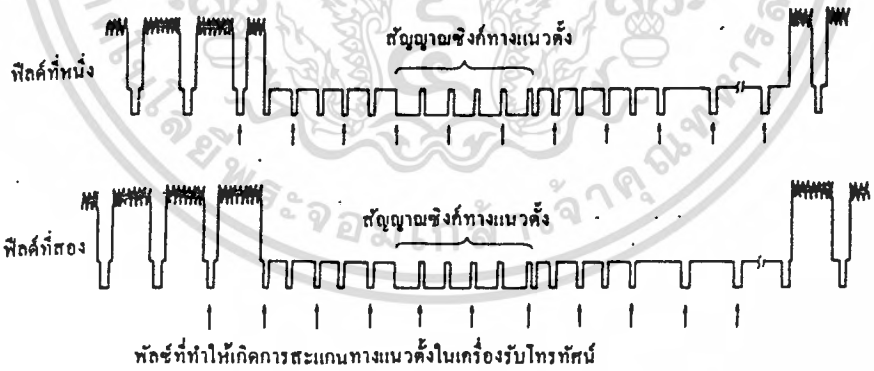
- สัญญาณอีควัลไลซิ่ง (equalizing signal)

สัญญาณต่างๆ ตามรูปนี้จะรวมอยู่เป็นรูปร่างเดียวกัน ซึ่งเรียกว่า สัญญาณภาพรวม (composition video signal) แล้วใช้คลื่นพาห์ของภาพเป็นตัวพาออกอากาศ รวมกับคลื่นพาห์ของสัญญาณเสียง เหตุผลและความจำเป็นในการใช้สัญญาณต่างๆ มีดังต่อไปนี้

(ก) สัญญาณภาพ (video signal) และสัญญาณเสียง (sound signal) เป็นสัญญาณที่ใช้เพื่อทำให้เกิดภาพขาวดำที่จอหลอดภาพ และมีเสียงที่ลำโพงเครื่องรับโทรทัศน์ตามต้องการ สัญญาณภาพ (video signal) บางครั้งเรียกว่าสัญญาณส่องสว่าง (brightness signal หรือ luminance signal)

(ข) สัญญาณแบล้งกิ้ง (blanking signal) เป็นสัญญาณที่ใช้เพื่อลบเส้นสะแกนสะบัดกลับทั้งในแนวนอนและในแนวตั้ง เพื่อมิให้เป็นที่สังเกตเห็นได้ชัดทางจอหลอดภาพ รูปที่ 4.2.6 (ก) เป็นรูปขยายของระยะเวลาที่มีสัญญาณแบล้งค์กิ้งทางแนวนอน (horizontal blanking signal) และในช่วงระยะเวลาที่มีสัญญาณแบล้งกิ้งทางแนวนอนนี้ ก็จะส่งสัญญาณซิงค์ทางแนวนอน(horizontal synchronizing signal) ไปด้วย และจะอยู่ในระดับดำสนิทมากกว่าสัญญาณแบล้งกิ้ง ส่วนที่เหลื่อมล้ำกันระหว่างแบล้งกิ้งพัลซ์กับซิงค์พัลซ์นี้ จะมีอยู่สองส่วนตามรูปที่แสดงไว้ ส่วนหน้าเรียกว่า front porch และส่วนหลังเรียกว่า back porch สำหรับโทรทัศน์ระบบอเมริกัน ความถี่ของกระแสรูปฟันเลื่อยที่ไหลผ่านขดลวดของการหักเหในแนวนอนมีค่า 15,750 เฮิร์ทซ์ ฉะนั้น ในระยะเวลา 1/15 ,750 วินาที หรือ 63.5 ไมโครวินาที จะต้องเกิดเส้นสะแกนสะบัดกลับครั้งหนึ่ง จึงจำเป็นต้องใช้แบ

ล้งกิ้งพัลซ์ทางแนวนอนครั้งหนึ่ง โดยมีขนาดประมาณ 10 ไมโครวินาที ส่วนรูปที่ 4.2.6 นั้น เป็นรูปขยายของระยะเวลาที่มีสัญญาณแบล้งกิ้งทางแนวตั้ง (vertical blanking signal) สำหรับโทรทัศน์ระบบอเมริกัน ทุกๆ ระยะเวลา 1/60 วินาที หรือ 16.66 ไมโครวินาที จำเป็นต้องให้มี แบล้งค์กิ้งพัลซ์ ทางแนวตั้งครั้งหนึ่งโดยมีขนาดประมาณ 1,250 ไมโครวินาที ในระยะเวลาที่มี แบล้งค์กิ้งพัลซ์ทางแนวตั้งนี้ ก็จะส่งสัญญาณซิงค์ทางแนวตั้งออกไปด้วย และเพื่อประโยชน์ในการช่วยทำให้สัญญาณซิงค์ทางแนวตั้ง ยังคงมีรูปร่างเหมือนเดิม หลังจากแยกออกมาจากสัญญาณซิงค์ทางแนวนอนทางเครื่องรับโทรทัศน์แล้วจะนิยมส่งอีควัลไลซิงพัลซ์ (equalizing pulses) กับ คัตติ้งพัลซ์ (cutting pulses) ไปด้วยตามรูปที่ 4.2.6 ความถี่ของอีควัลไลซิงพัลซ์และคัทติ้งพัลซ์นี้ จะมีค่าเป็นสองเท่าของความถี่สัญญาณซิงค์ทางแนวนอน เพื่อช่วยให้การสะแกนแบบหนึ่งเฟรมแบ่งออกเป็นสองฟีลด์ทางด้านเครื่องรับโทรทัศน์ เป็นไปอย่างถูกต้องเหมาะสม จุดตั้งต้นของสัญญาณซิงค์ทางแนวนอนและสัญญาณซิงค์ทางแนวตั้งนี้ จะต้องมีส่วนสัมพันธ์กันอย่างเหมาะสม คือ เมื่อหมดการสะแกนฟีลด์หนึ่งๆ แล้ว จะต้องเกิดขึ้นพร้อมกันเพื่อทำการสะแกนฟีลด์ต่อๆไป ตามที่ได้แสดงไว้ในรูป 4.2.7 มิฉะนั้น การสะแกนไขว้กันทางเครื่องรับโทรทัศน์อาจไม่เป็นไปในจังหวะที่ถูกต้องได้



รูปที่ 4.2.7 พัลซ์ที่เกิดขึ้นในระยะเวลาที่มีสัญญาณแบล้งค์กิ้งทางแนวตั้ง ในฟีลด์ที่หนึ่งและฟีลด์ที่สอง

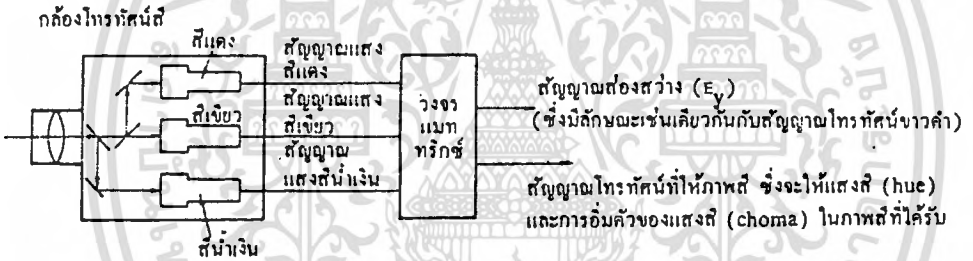
(ค) สัญญาณซิงค์ (synchronizing signal) เป็นสัญญาณที่ใช้เพื่อช่วยทำให้ความถี่ของกระแสรูปฟันเลื่อยใช้ในวงจรของการหักเหทางแนวนอนกับวงจรของการหักเหทางแนวตั้งของเครื่องส่งโทรทัศน์มีค่าตรงกันกับที่ใช้ในเครื่องรับโทรทัศน์ อันจะมีผลทำให้การสะแกนของภาพทางด้านเครื่องส่งโทรทัศน์ตรงกันกับทางด้านเครื่องรับโทรทัศน์ตลอดเวลา สัญญาณซิงค์ทางแนวนอนจะมีความถี่เท่ากันกับความถี่ของกระแสรูปฟันเลื่อยที่ใช้ในวงจรของการหักเหทางแนวนอน และสัญญาณซิงค์ทางแนวตั้งก็จะมีความถี่เท่ากับความถี่ของกระแสฟันเลื่อยที่ใช้ในวงจรการหักเหทางแนวตั้ง เนื่องจากความถี่ของสัญญาณซิงค์นี้เท่ากับความถี่ของสัญญาณแบล้งค์กิ้ง จึงจำเป็นต้องป้องกันการรบกวนที่อาจเกิดขึ้น โดยจำเป็นต้องกำหนดขนาดของซิงค์พัลซ์ให้น้อยกว่าขนาดของแบล้งค์กิ้งพัลซ์ กล่าวคือ ทำให้ซิ้งค์พัลซ์ทางแนวนอนมีขนานเพียง 5ไมโครวินาที และซิงค์พัลซ์ทางแนวตั้งมีขนาดเพียง 190 ไมโครวินาทีเท่านั้น นอกจากนี้ ยังใช้วิธีส่งซิงค์พัลซ์เหล่านี้ปะปนกับแบล้งค์กิ้งพัลซ์ โดยทำให้ฐานของซิ้งค์พัลซ์อยู่ทับขอบบนของแบล้งกิ้งพัลซ์ ก็จะเป็นระดับดำมืดสนิทและไม่ทำให้เกิดการรบกวนภาพท่จอหลอดภาพแต่ประการใด

(ง) สัญญาณอีควัลไลซิ่ง (equalizing signal) เป็นสัญญาณที่ใช้เพื่อช่วยทำให้สัญญาณซิงค์ทางแนวตั้งยังคงมีรูปร่างดีเหมือนเดิม หลังจากแยกออกมาจากสัญญาณซิงค์ทางแนวนอนในเครื่องรับโทรทัศน์แล้ว สัญญาณ นี้มีความถี่เป็นสองเท่าของสัญญาณซิงค์ทางแนวนอน ซึ่งจะช่วยทำให้การสะแกนไขว่กันทางเครื่องรับโทรทัศน์เป็นไปโดยเรียบร้อย รวมทั้งสัญญาณซิงค์ทางแนวนอนก็จะไม่ขาดหายไปในช่วงเวลาที่มีสัญญาณซิงค์ทางแนวตั้งอีกด้วย ขนาดของอีควัลไลซิ่งพัลซ์ ก็มีขนาดประมาณซิงค์พัลซ์ทางแนวตั้ง คือประมาณ 190 ไมโครวินาที หรือประมาณ สามเท่าของซิงค์พัลซ์ทางแนวนอน นอกจากนี้ ยังนิยมแบ่งพัลซ์นี้ออกเป็นพัลซ์เล็กๆ ตามรูปที่ 4.2.6 เพื่อทำให้เกิดซิงค์ทางแนวนอนครั้งหนึ่ง ในทุกๆ สองครั้งที่มีพัลซ์เล็กๆ เหล่านี้

4.3 สัญญาณโทรทัศน์สีประกอบด้วยอะไรบ้าง

ตามที่ได้กล่าวแล้วความต้องการในเรื่องการให้บริการโทรทัศน์สีในเขตบริการที่มีเครื่องรับโทรทัศน์ขาวดำอยู่ด้วย มีอยู่ด้วยกันสามประการ ประการแรกคือ เรื่องของ compatibility กับ reverse compatibility ประการที่สอง คือสัญญาณโทรทัศน์สีที่ส่งออกอากาศ จะต้องประกอบด้วยส่วนที่เป็นสัญญาณส่องสว่าง (luminance signal) กับส่วนที่เป็นสัญญาณโทรทัศน์ที่ให้ภาพสี (chrominance signal) ประการสุดท้าย ก็คือ ความถี่ของสัญญาณโทรทัศน์ที่ให้ภาพสี (chrominance

signal) จะต้องมีขอบเขตอยู่ในความถี่เดียวกันกับสัญญาณส่องสว่าง (luminance signal) ในทางปฏิบัติ กล้องโทรทัศน์สีจะช่วยเหลือทำให้เกิดสัญญาณแสงสีแดง-แสงสีเขียว-แสงสีน้ำเงิน ขึ้น ซึ่งหากจะทำการส่งสัญญาณแสงสีเหล่านี้ไปยังเครื่องรับโทรทัศน์สีโดยตรง จะเป็นการยุ่งยาก จึงใช้วิธีการนำสัญญาณแสงสีทั้งสามเหล่านี้ มาผสมกันในวงจรพิเศษ ซึ่งเรียกว่า แมททริกซ์ (matrix) เพื่อทำให้เกิดเป็นสัญญาณใหม่สองสัญญาณคือ สัญญาณโทรทัศน์ขาวดำ หรือสัญญาณส่องสว่าง (luminance signal) และสัญญาณโทรทัศน์ที่ให้ภาพสี (chrominance signal) ตามที่ได้แสดงไว้แล้วในรูปที่ 4.3.1 เมื่อสถานีโทรทัศน์ส่งสัญญาณต่างๆ เหล่านี้ออกอากาศ เครื่องรับโทรทัศน์ขาวดำ ก็จะรับแต่เฉพาะสัญญาณโทรทัศน์ขาวดำหรือ ส่วนที่เป็นสัญญาณส่องสว่าง (luminance signal) ส่วนเครื่องรับโทรทัศน์สี ก็จะรับสัญญาณโทรทัศน์ทั้งหมด และทำให้เกิดเป็นภาพสีขึ้นที่จอหลอดภาพต่อไป



รูปที่ 4.3.1 สัญญาณที่ได้จากกล้องโทรทัศน์สี

กล่าวโดยสรุป สถานีโทรทัศน์จะต้องส่งสัญญาณต่างๆ ออกอากาศไปให้เครื่องรับโทรทัศน์เพื่อทำให้เกิดภาพสีขึ้นที่จอหลอดภาพของเครื่องรับโทรทัศน์สี และภาพขาวดำที่จอหลอดภาพของเครื่องรับโทรทัศน์ขาวดำ กับมีเสียงที่ลำโพงของเครื่องรับโทรทัศน์ ดังต่อไปนี้

- สัญญาณเสียง (sound signal)

- สัญญาณโทรทัศน์ขาวดำ หรือสัญญาณส่องสว่าง (luminance signal)

- สัญญาณโทรทัศน์ที่ให้ภาพสี (chrominance signal)

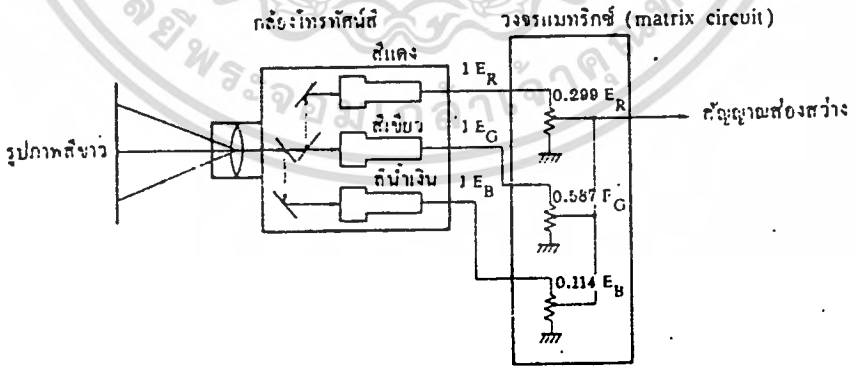
- สัญญาณซิ้งค์,แบล้งค์กิ้ง และอีควัลไลซิ่ง (synchronizing , blanking , and equalizing signals)

- สัญญาณซิ้งค์ของภาพสี (color sync signal)

เหตุผลและความจำเป็นในการที่ต้องส่งสัญญาณต่างๆ เหล่านี้ มีดังต่อไปนี้

(ก) สัญญาณเสียง (sound signal) เพื่อทำให้เกิดเสียงที่ลำโพงเครื่องรับโทรทัศน์ตามต้องการ

(ข) สัญญาณโทรทัศน์ขาวดำ หรือสัญญาณส่องสว่าง (luminance signal) สัญญาณนี้ คือ สัญญาณภาพ (video signal or brightness signal) ในเรื่องของโทรทัศน์ขาวดำนั่นเอง กล้องโทรทัศน์สีในห้องส่งโทรทัศน์ จะช่วยทำให้เกิดสัญญาณแสงสีแดง-แสงสีเขียว-แสงสีน้ำเงิน ขึ้น ซึ่งเราอาจใช้สัญลักษณ์ E_R , E_G , E_B แทนค่าสัญญาณทางไฟฟ้าเป็นโวลท์ที่ได้จากหลอดสีแดง-หลอดสีเขียว-หลอดสีน้ำเงิน ตามลำดับ สัญญาณทั้งสามนี้ จะผ่านวงจรพิเศษ เรียกว่า วงจรแมททริกซ์ (matrix)ตามที่ได้แสดงไว้ในรูปที่ 4.3.2



รูปที่4.3.2 สัญญาณส่องสว่าง

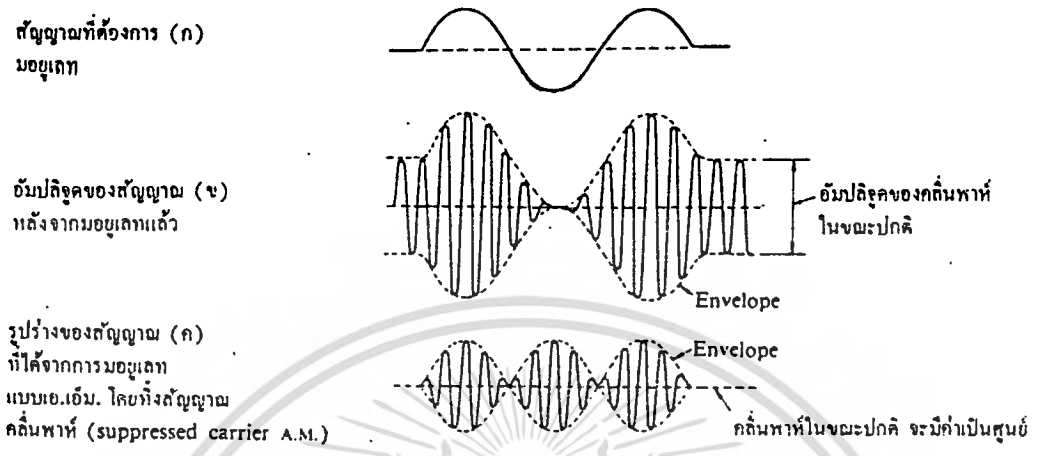
เพื่อทำให้เกิดสัญญาณส่องสว่าง(brightness or video signal, or luminance signal) Ey โดยมีส่วนผสมของแสงสีทั้งสามในอัตราส่วนที่แน่นอน ดังต่อไปนี้

$$E_Y = 0.299\,E_R + 0.587\,E_G + 0.114\,E_B$$

(ค) สัญญาณโทรทัศน์ที่ให้ภาพสี(chrominance signal) สัญญาณนี้เป็นสัญญาณที่เครื่องส่งโทรทัศน์สีได้ทำการส่งไปยังเครื่องรับโทรทัศน์ เพื่อช่วยเหลือทำให้เกิดภาพสีขึ้นทางจอหลอดภาพของเครื่องรับโทรทัศน์สี คุณลักษณะของสัญญาณนี้ จะขึ้นอยู่กับระบบของโทรทัศน์สี ซึ่งมีอยู่สามระบบคือ โทรทัศน์สีระบบNTSC, โทรทัศน์สีระบบPAL และโทรทัศน์สีระบบSECAM สัญญาณโทรทัศน์ที่ให้ภาพสีในโทรทัศน์สีระบบNTSC จะประกอบด้วยสัญญาณสีสองสัญญาณที่รวมกันอยู่ในรูปของ amplitude-modulated signal suppressed carrier โดยจะต้องทำให้มุมระหว่างคลื่นพาห์ทั้งสองต่างกันเก้าสิบองศา สัญญาณสีทั้งสองสัญญาณนี้ เครื่องรับโทรทัศน์สีจะมีวงจรพิเศษแยกออกมาเพื่อนำไปใช้ในการควบคุมเรื่องแสงสี (hue) และการอิ่มตัวของแสงสี (saturation) ของภาพสีที่ปรากฏบนจอหลอดภาพ สัญญาณโทรทัศน์ที่ให้ภาพสีนี้ จะใช้คลื่นพาห์ของสัญญาณ หรือคัลเลอร์ซับแคริเออร์ (color subcarrier) ที่ความถี่ 3.58 เมกกะเฮิร์ทซ์และจะต้องรวมกันกับสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่าง (luminance signal) เพื่อใช้คลื่นพาห์ของเครื่องส่งโทรทัศน์สีนำออกอากาศเพื่อส่งต่อไปให้ถึงเครื่องรับโทรทัศน์สี วิธีการส่งสัญญาณโทรทัศน์ที่ให้ภาพสีรวมไปกับสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่าง โดยการใช้คลื่นพาห์ของสัญญาณ สีนี้เป็นวิธีการพิเศษทางไฟฟ้า ซึ่งเรียกว่า multiplex transmission

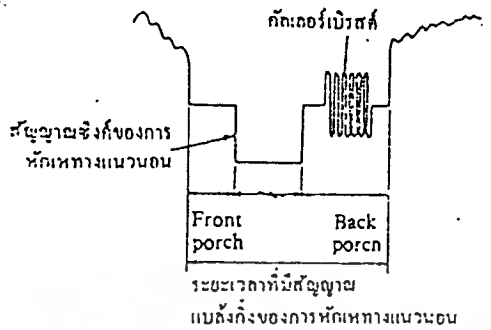
(ง) สัญญาณซิ้งค์,แบล้งค์กิ้ง และอีควัลไลซิ่ง (synchronizing , blanking , and equalizing signals) สัญญาณดังกล่าวนี้ มีลักษณะเช่นเดียวกันกับสัญญาณซิงค์ สัญญาณแบล้งค์กิ้ง และสัญญาณอีควัลไลซิ้ง ในเรื่องของโทรทัศน์ขาวดำทุกประการ

(จ) สัญญาณซิงค์ของภาพสี (color sync carrier) เนื่องจากสัญญาณโทรทัศน์ที่ให้ภาพสี ตามข้อ (ค)อยู่ในรูปของ amplitude-modulated signal (suppressed carrier) ตามที่แสดงไว้ในรูปที่4.3.3



รูปที่ 4.3.3 รูปร่างของสัญญาณที่ได้จากการมอยูเลทแบบ เอ.เอ็ม. โดยทิ้งสัญญาณคลื่นพาห์ (suppressed carrier amplitude modulated)

ซึ่งเมื่อเครื่องรับโทรทัศน์สีรับได้แล้ว ก็จำเป็นต้องใช้คลื่นพาห์คัลเลอร์ซับแคริเออร์ (color subcarrier) ที่เหมือนกันกับที่ใช้ในเครื่องส่งโทรทัศน์ด้วย ฉะนั้น เครื่องรับโทรทัศน์สีจึงจำเป็นต้องมีวงจรผลิตคลื่นพาห์ คัลเลอร์ซับแคริเออร์ ที่ต้องการขึ้น เพื่อทำให้ คลื่นพาห์ คัลเลอร์ซับแคริเออร์ที่ใช้ในด้านเครื่องส่งโทรทัศน์กับด้านเครื่องรับโทรทัศน์สี มีความถี่และเฟส (phase angle) ที่ถูกต้องตามกัน เครื่องส่งโทรทัศน์จึงจำเป็นต้องส่ง สัญญาณซิงค์ของภาพสี (color sync carrier) ไปให้เครื่องรับโทรทัศน์โดยส่งไปในส่วนของ back porch ของซิงค์พัลซ์ทางแนวนอน ตามที่แสดงไว้ในรูปที่ 4.3.4 และรูปที่ 4.3.5 ซึ่งเรียกสัญญาณซิงค์ของภาพสีนี้ว่า คัลเลอร์เบิร์สต์ (color burst)

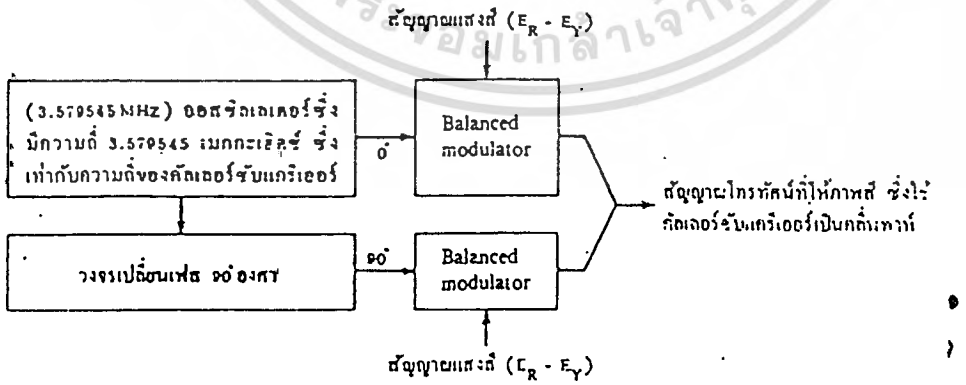


รูปที่ 4.3.4 คัลเลอร์เบิรสต์

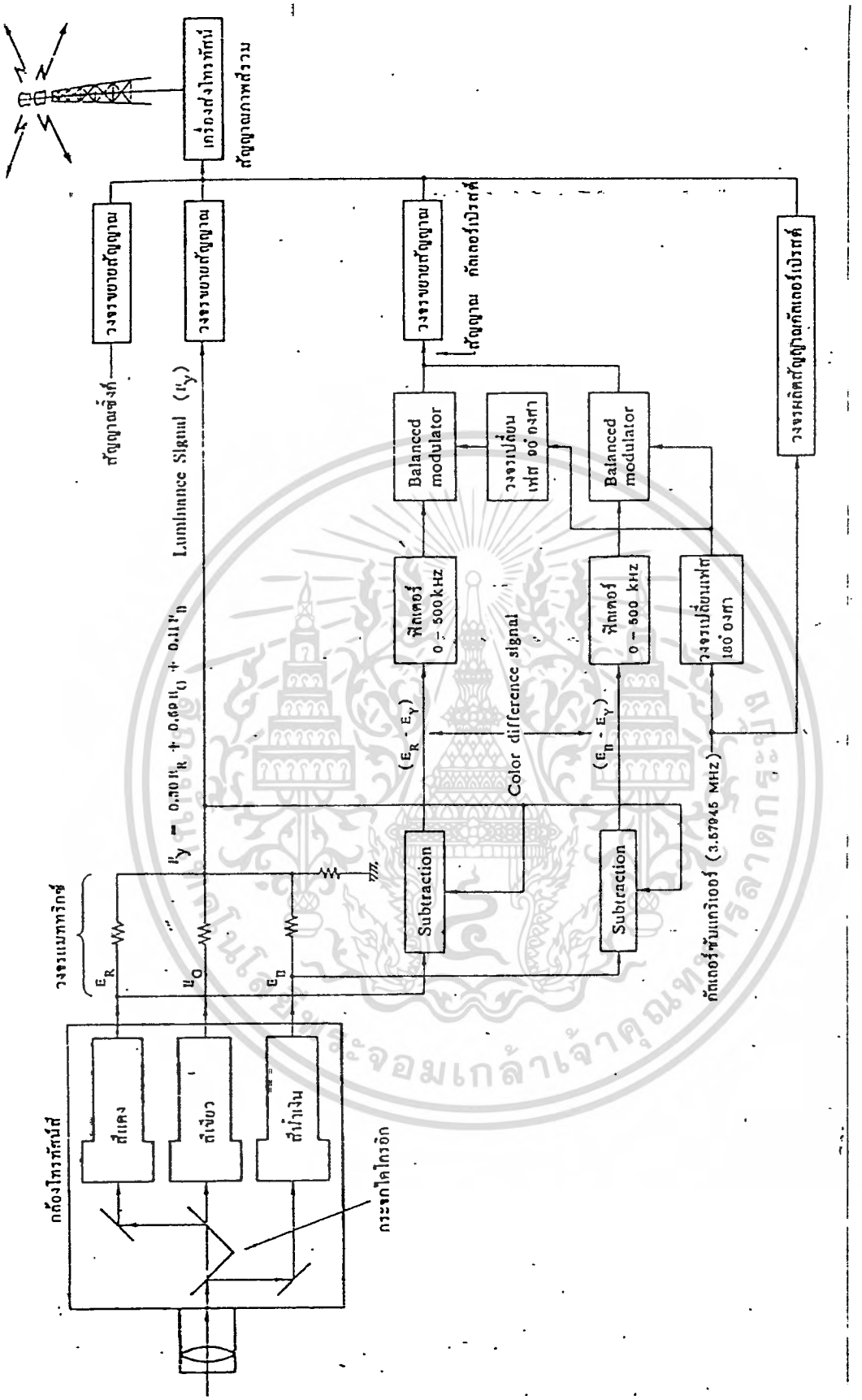
ความแตกต่างของโทรทัศน์สีระบบต่างๆ อยู่ที่วิธีการและวงจรในการส่งสัญญาณโทรทัศน์ที่ใช้ภาพสีรวมไปกับสัญญาณโทรทัศน์ขาวดำ หรือสัญญาณส่องสว่างเท่านั้น ซึ่งเป็นเรื่องที่จะได้กล่าวถึงต่อไป

4.4 โทรทัศน์สีระบบเอ็น.ที.เอส.ซี. (NTSC color TV system)

โทรทัศน์สีระบบแรกของโลก คือโทรทศน์สีระบบอเมริกัน เอ็น.ที.เอส.ซี. ซึ่งเป็นโทรทัศน์ของสหรัฐอเมริกา และต่อมาก็ได้แพร่หลายไปในอีกหลายประเทศที่ใช้โทรทัศน์ระบบอเมริกัน 525 เส้น ต่อภาพ 30 ภาพ ต่อวินาที หลักการของการส่งโทรทัศน์สีระบบนี้ ได้แสดงไว้แล้วในรูปที่ 4.4.1 และรูปที่ 4.4.2



รูปที่ 4.4.1 การมอดูเลทสัญญาณแสงสีในโทรทัศน์ระบบ เอ็น.ที.เอส.ซี.



รูปที่ 4.4.2 แผนผังการส่งโทรทัศน์สีในระบบ N.T.S.C.

โดยกล้องโทรทัศน์สีจะทำให้เกิดสัญญาณทางไฟฟ้าจากแสงสีแดง-แสงสีเขียว-แสงสีน้ำเงิน มีขนาด E_R , E_G , E_B ตามลำดับ แสงสีทั้งสามนี้จะผสมกันในวงจรพิเศษที่เรียกว่า วงจรแมททริกซ์ (matrix circuit) การบวกลบผสมสีทางไฟฟ้าของแสงสีแดง-แสงสีเขียว-แสงสีน้ำเงิน เหล่านี้จะทำให้เกิดสัญญาณโทรทัศน์ขาวดำกชหรือสัญญาณส่องสว่าง (luminance signal) E_Y กับสัญญาณโทรทัศน์ที่ให้ภาพสี (chrominance signal) สองสัญญาณคือ inphase color signal voltate E_I และ quadrature- phase color signal voltate E_Q โดยมีส่วนผสมของ

Luminance signal

$$E_Y = 0.299\,E_R + 0.587\,E_G + 0.114\,E_B$$

Inphase color signal voltate

$$E_I = 0.74\,(E_R - E_Y) - 0.27\,(E_B - E_Y)$$

$$= 0.60\,E_R - 0.28\,E_G - 0.32\,E_B$$

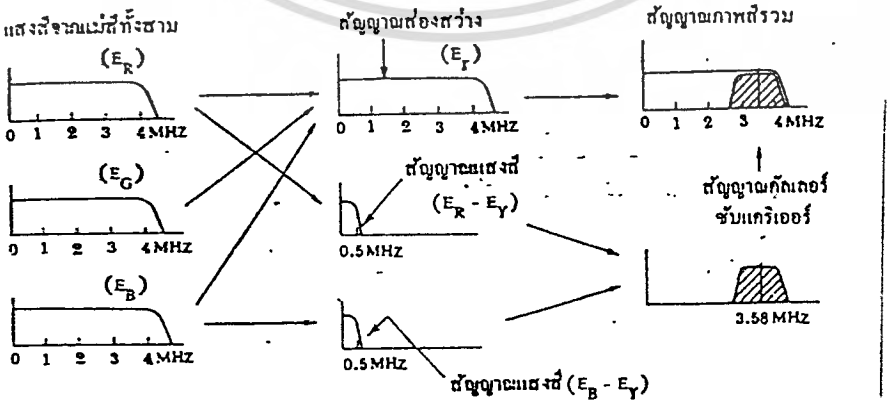
Quadrature- phase color signal voltate

$$E_Q = 0.48\,(E_R - E_Y) + 0.41\,(E_B - E_Y)$$

$$= 0.21\,E_R - 0.52\,E_G + 0.31\,E_B$$

เพื่อหลีกเลี่ยงการลบกวนที่อาจเกิดขึ้นในการส่งสัญญาณทางไฟฟ้าเหล่านี้ จากเครื่องส่งโทรทัศน์ไปยังเครื่องรับโทรทัศน์ จึงใช้วิธีการ double modulation AM - AM กล่าวคือสัญญาณภาพสีทั้งสองสัญญาณจะมีคลื่นพาห์ของตัวเองโดยเฉพาะ เรียกว่าคัลเลอร์ซับแคริเออร์ (color subcarrier) สัญญาณโทรทัศน์ให้ภาพสี E_I กับ E_Q จะผ่านเข้าไปยัง encoder ซึ่งมี balanced modulator อยู่สองชุด แต่ละชุดจะมี คัลเลอร์ซับแคริเออร์ (color subcarrier) ซึ่งมีความถี่เดียวกัน แต่มี เฟส (phase

angle) ต่างกันอยู่ 90 องศา ผลลัพธ์ที่ได้จาก balanced modulator ทั้งสองนี้ก็คือ สัญญาณโทรทัศน์ที่ให้ภาพสีทางไฟฟ้าสองชุด ซึ่งแต่ละชุดจะอยู่ในรูปของ amplitude-modulated suppressed-carrier double-sidebands ตามที่แสดงไว้ในรูป 4.3.3 สัญญาณเหล่านี้จะนำไปรวมกับสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่างกับสัญญาณอื่นๆ ก่อน แล้วจึงจะใช้คลื่นพาห์ในเครื่องส่งโทรทัศน์นำออกอากาศในวิธีการของ amplitude modulation ตามได้แสดงไว้แล้วในรูปที่ 4.3.3 สัญญาณที่เกิดขึ้นจะอยู่ในรูปร่างของ amplitude-modulated double-sidebands ซึ่งกว้างประมาณข้างละ 4 เมกกะเฮิร์ทซ์ แต่การนำผ่านวงจร vestigial sideband filter จะช่วยลดไซด์แบนด์ด้านต่ำ (lower sideband) ลงบ้าง และจะส่งไซด์แบนด์ด้านสูง (upper sideband) เต็มที่ ซึ่งจะทำให้มีแบนด์วิดธ์ ทั้งสิ้น (overall RF bandwidth) ประมาณ 6 เมกกะเฮิร์ทซ์ ซึ่งเท่ากับแบนด์วิดธ์ของช่องโทรทัศน์ขาวดำ ในระบบอเมริกันพอดี สำหรับแบนด์วิดธ์ ของสัญญาณโทรทัศน์ที่ให้ภาพสี E_I กับ E_Q นั้น ของการทดลองได้พบว่า สายตาของคนเรา จะสังเกตการเปลี่ยนแปลงในเรื่อง สัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่าง ได้ง่ายกว่าและดีกว่าการเปลี่ยนแปลงในเรื่องของสัญญาณโทรทัศน์ให้ภาพสี โดยเฉพาะในส่วนประกอบเล็กๆ ของภาพบนจอโทรทัศน์ ไม่จำเป็นต้องทำให้รายละเอียดของสัญญาณโทรทัศน์ที่ให้ภาพสีมีมากเท่ากับ รายละเอียดของสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่าง ฉะนั้น โทรทัศน์สีระบบ NTSC จึงกำหนดให้สัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่าง รายละเอียดของสัญญาณภาพได้กว้างเต็มที่ประมาณ 4.2 เมกกะเฮิร์ทซ์ ส่วนสัญญาณโทรทัศน์ที่ให้ภาพสี E_I ซึ่งแสดงลักษณะคุณสมบัติของแสงสีเขียวน้ำเงิน (cyan) หรือแสงสีส้ม(orange color) จะมีรายละเอียดได้กว้างประมาณ 1.5 เมกกะเฮิร์ทซ์ กับสัญญาณโทรทัศน์ที่ให้ภาพสี E_Q ซึ่งแสดงลักษณะคุณสมบัติของแสงสีม่วง(purple) หรือแสงสีเขียว(green) นั้น สามารถกำหนดให้มีรายละเอียดได้ประมาณ 0.5 เมกกะเฮิร์ทซ์ ก็เป็นการเพียงพอแล้ว ตามที่ได้แสดงไว้ในรูปที่ 4.4.3



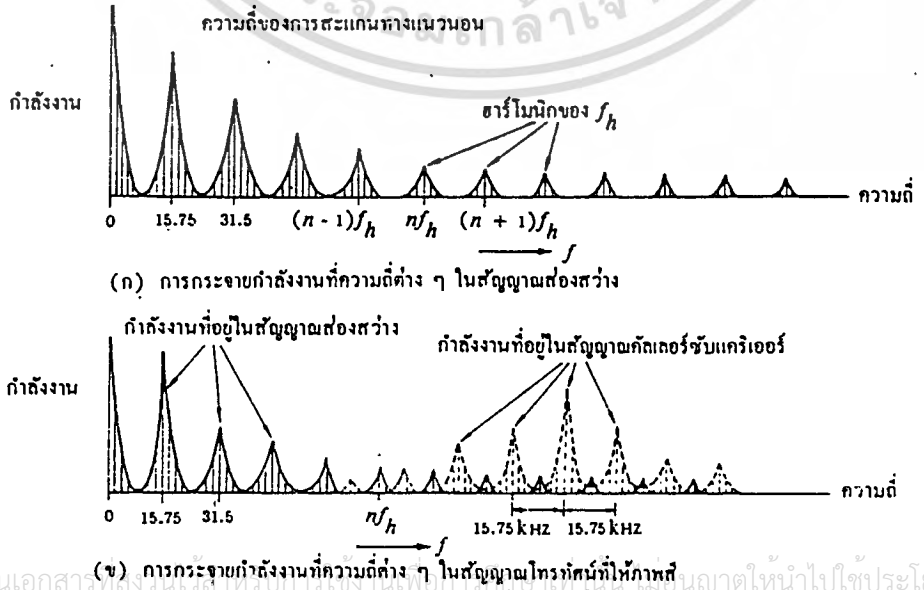
รูปที่ 4.4.3 สัญญาณส่องสว่างและสัญญาณโทรทัศน์ที่ให้ภาพสี ของโทรทัศน์สีระบบ N.T.S.C.

การส่งโทรทัศน์สีระบบ NTSC จากเครื่องส่งโทรทัศน์มายัง เครื่องรับโทรทัศน์นี้ สัญญาณโทรทัศน์ภาพสี จะใช้คลื่นพาห์ของภาพสีหรือคัลเลอร์ซับแคริเออร์ จึงจำเป็นต้องส่งคัลเลอร์ซับแคริเออร์ นี้มายังเครื่องโทรทัศน์ด้วย การเลือกใช้คัลเลอร์ซับแคริเออร์ ก็จำเป็นจะต้องพิถีพิถันและมีหลักเกณฑ์ เหมือนกัน ซึ่งพอสรุปได้ดังนี้ คือ เนื่องจากรายละเอียดของสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่างจะมีความถี่ระหว่าง 0-4 เมกกะเฮิร์ทซ์ ซึ่งมีกำลังงานรวมกลุ่มกันอยู่ในบริเวณความถี่ 15,750 เฮิร์ทซ์ ซึ่งเป็นค่าของความถี่ของการสะแกนแนวนอน หากเลือกใช้คัลเลอร์ซับแคริเออร์ให้มีค่าเท่ากับ ฮาร์โมนิก (harmonic) ต่างๆ ของความถี่ ตามที่กล่าวแล้วนี้ จะทำให้ลดอาการรบกวนที่เกิดขึ้นระหว่างความถี่ของคัลเลอร์ซับแคริเออร์ กับความถี่ของความแตกต่างระหว่างคลื่นพาห์ ของเสียง กับคลื่นพาห์ของภาพได้ ฉะนั้นในหลักการจึงมีข้อกำหนดดังนี้

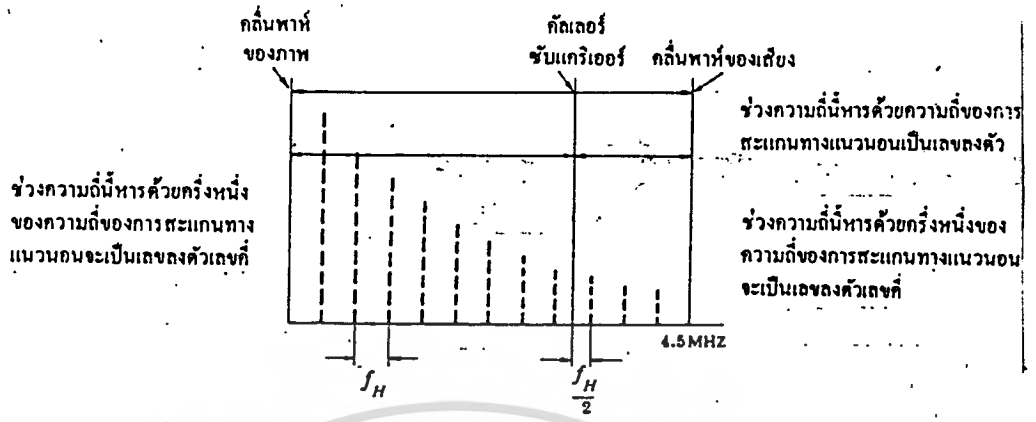
(ก) กำหนดให้ ความถี่ของความแตกต่างระหว่างคลื่นพาห์ ของเสียง กับคลื่นพาห์ของภาพ (sound carrier spacing relative to video carrier) มีค่าเท่ากับฮาร์โมนิกที่ 286 ของความถี่ในการ สะแกนทางแนวนอน

(ข) กำหนดให้ความถี่ของคัลเลอร์ซับแคริเออร์ มีค่าเท่ากับ ฮาร์โมนิกที่ 455 ของครึ่งหนึ่งของความถี่ในการสะแกนทางแนวนอน

หากกำหนดให้ความถี่ของการสะแกนทางแนวนอนมีค่า 15,750 เฮิร์ทซ์ เราจะพบว่าความถี่ของคัลเลอร์ซับแคริเออร์นี้ จะมีค่า(455)(15,750)/(2) เฮิร์ทซ์ หรือ 3.58 เมกกะเฮิร์ทซ์ ตามที่แสดงไว้ในรูปที่ 4.4.4 และรูปที่ 4.4.5



รูปที่ 4.4.4 การกระจายกำลังงานที่ความถี่ต่างๆ ในสัญญาณส่องสว่าง และในสัญญาณโทรทัศน์ที่ให้ภาพสี



รูปที่ 4.4.5 ความสัมพันธ์ของความถี่คลื่นพาห์ของภาพ ความถี่คลื่นพาห์ของเสียง และความถี่ของคัลเลอร์ซับแคริเออร์ในระบบโทรทัศน์สี

อย่างไรก็ตามมาตรฐานของโทรทัศน์สีระบบ NTSC นี้ จะมีส่วนที่แตกต่างกันบ้างเล็กน้อยกับมาตรฐานของโทรทัศน์ขาวดำ ในระบบอเมริกันเหมือนกัน เพราะสาเหตุที่ต้องการลดการรบกวนในการเลือกใช้ความถี่ของคัลเลอร์ซับแคริเออร์นี้ เมื่อความแตกต่าง ระหว่างคลื่นพาห์ของเสียงกับคลื่นพาห์ของภาพในโทรทัศน์ระบบอเมริกัน มีค่าความถี่ 4.5 เมกกะเฮิร์ทซ์ เราจะได้ตัวเลขที่แน่นอนของค่าความถี่ต่างๆ ดังนี้

ความถี่ของการสะแกนทางแนวนอนจะมีค่า(4.5)/(286) เมกกะเฮิร์ทซ์ หรือ 15,734.296 เฮิร์ทซ์ ความถี่ของคัลเลอร์ซับแคริเออร์ จะมีค่า (455) (15,734.296)/(2) เฮิร์ทซ์ หรือ 3.579545 เมกกะเฮิร์ทซ์

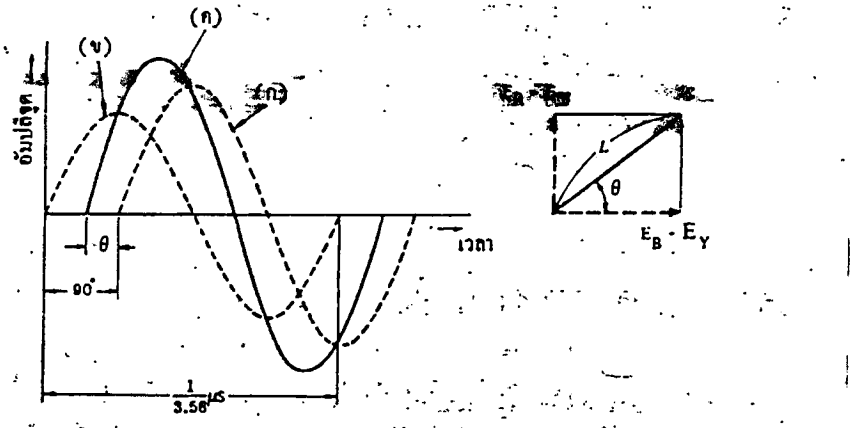
ความถี่ของฟีลด์ มีค่า (15,734.296) /(262.5) เฮิร์ทซ์ หรือ 59.94 เฮิร์ทซ์

ตัวเลขที่แน่นอนในเรื่องความถี่ของการสะแกนทางแนวนอนและในเรื่องความถี่ของฟีลด์สำหรับโทรทัศน์สีจะมีค่าใกล้เคียงกับ 15,750 เฮิร์ทซ์ และ 60 เฮิร์ทซ์ ซึ่งเป็นตัวเลขที่ใช้ในมาตรฐานโทรทัศน์ขาวดำ ระบบอเมริกันมาก และไม่ทำให้เกิดปัญหาในเรื่องการซิงค์สัญญาณต่างๆ แต่ประการใด ในทางปฏิบัติ

รูปที่ 4.4.6 เป็นสัญญาณภาพสีรวม (colorplexed composite video signal) ที่เครื่องส่งโทรทัศน์สีต้องส่งไปให้เครื่องรับโทรทัศน์ ซึ่งประกอบด้วยสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่าง (luminance signal)

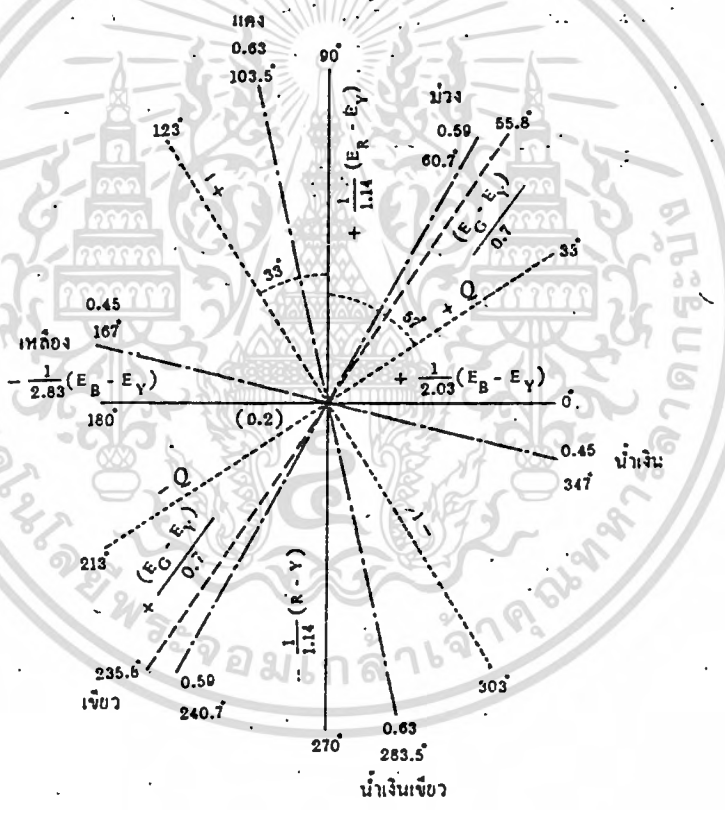
รูปที่ 4.4.6 สัญญาณส่องสว่างกับสัญญาณโทรทัศน์ที่ให้ภาพสี (color composit signal)

สัญญาณโทรทัศน์ที่ให้ภาพสีสองสัญญาณซึ่งอยู่ในรูปร่างของ amplitude-modulated suppressed-carrier double-sidebands สัญญาณซิงค์ สัญญาณแบล้งค์กิ้ง สัญญาณอีควัลไล้ซิ้งค์และสัญญาณซิงค์ของภาพสีที่เรียกว่า คัลเลอร์เบิร์สต์ (color burst) ทางด้านเครื่องรับโทรทัศน์ ก็มีการแยกสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่างออกจากสัญญาณโทรทัศน์ที่ให้ภาพสีและจะมีวงจรคัลเอร์ดีมอยูเลเตอร์ (color demodulator) สองชุด



(ก) สัญญาณเอ้าท์พุทของ balanced modulator ซึ่งมี $(E_B - E_Y)$ เป็นสัญญาณอินพุท
(ข) สัญญาณเอ้าท์พุทของ balanced modulator ซึ่งมี $(E_R - E_Y)$ เป็นสัญญาณอินพุท
(ค) สัญญาณโทรทัศน์ที่ให้ภาพสี ซึ่งเป็นผลรวมของสัญญาณ (ก) กับ (ข)

รูปที่ 4.4.7 รูปร่างของสัญญาณโทรทัศน์ที่ให้ภาพสี ซึ่งใช้คัลเลอร์ซับแคริเออร์เป็นคลื่นพาห์ (ขยายให้เห็นส่วนประกอบต่างๆ ชัดเจน)



รูปที่ 4.4.8 เฟสและแอมปลิจูดของสัญญาณโทรทัศน์ที่ให้ภาพสี ซึ่งมีการอิ่มตัวของแสงสีประมาณ 100 %

ซึ่งใช้คัลเลอร์ซับแคริเออร์ที่มีความถี่เดียวกัน แต่มีเฟสต่างกันอยู่ 90 องศา เพื่อทำให้เกิดสัญญาณโทรทัศน์ที่ให้ภาพสี E_I กับ E_Q ความสัมพันธ์ของสัญญาณโทรทัศน์ที่ให้ภาพสีทั้งสองคือ $(E_R - E_Y)$ และ $(E_B - E_Y)$ กับ สัญญาณคัลเลอร์ซับแคริเออร์ ได้แสดงไว้แล้วในรูปที่ 4.4.7 เมื่อนำสัญญาณโทรทัศน์ขาวดำหรือสัญญาณส่องสว่างกับสัญญาณโทรทัศน์ที่ให้ภาพสีเหล่านี้มาผสมกันในวงจรแมททริกซ์ (matrix circuit)ก็จะได้สัญญาณของแสงสีแดง-แสงสีเขียว-แสงสีน้ำเงิน เพื่อป้อนให้แคโถดของหลอดภาพ และทำให้เกิดภาพสีขึ้นที่จอหลอดภาพในเครื่องรับโทรทัศน์สีต่อไป

บทที่ 5

การทดลอง

5.1 การทดลองแปลงไฟล์ภาพสี เป็น ไฟล์ภาพ NTSC

วัตถุประสงค์

1. เพื่อทำการทดสอบผลที่เกิดขึ้นเมื่อภาพสีถูกแปลงเป็น NTSC
2. เพื่อทำการพิจารณาขนาดไฟล์ภาพสี เป็น ไฟล์ภาพ NTSC

ขั้นตอนการทดลอง

1.ใช้โปรแกรมของปริญญานิพนธ์ที่ทำงานบนวินโดว์ทดลองภาพ colorbar โดยได้ภาพ colorbar จากฟังก์ชั่นในโปรแกรม

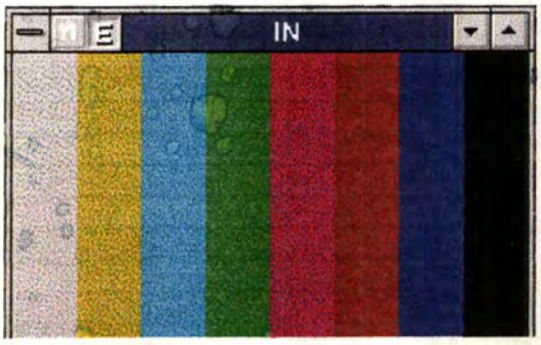
2.ใช้โปรแกรมของปริญญานิพนธ์ที่ทำงานบนวินโดว์นำภาพ colorbar copy จากหน้าต่างวินโดว์ in ไปหน้าต่าง วินโดว์ work

3. ใช้โปรแกรมของปริญญานิพนธ์ที่ทำงานบนวินโดว์นำภาพ colorbar ไปแปลงเป็น NTSC โดยมีใน menu ให้เลือก จากนั้นถ่ายภาพของหน้าต่างวินโดว์ in และ work

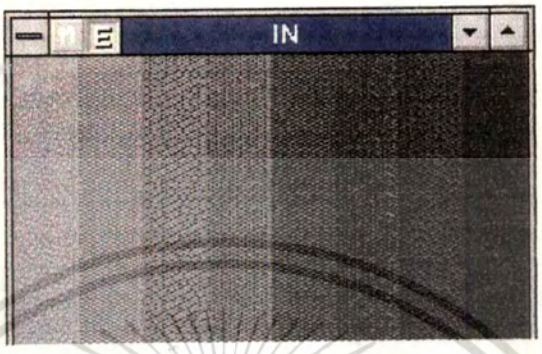
4. พิจารณาขนาดภาพ NTSC กับภาพสีต้นฉบับ

5.เปลี่ยนภาพโดยใช้ฟังก์ชั่นอ่านไฟล์โปรแกรมในโปรแกรมปริญญานิพนธ์ แล้วทำการทดลองซ้ำ ข้อ 2 โดยเปลี่ยนภาพ colorbar เป็นภาพไฟล์ที่อ่านเข้ามา

ผลการทดลอง



(ก.)



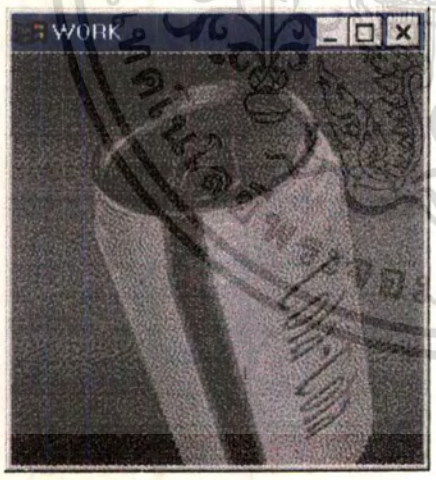
(ข.)

(ค.)

รูปที่ 5.1 แสดงการแปลงเป็นภาพ NTSC ของ colorbar
(ก) ภาพต้นฉบับ (ข) ภาพที่แปลงเป็น NTSC + (ค) ภาพที่แปลงเป็น NTSC -

(ก.)

(ข.)



(ค.)

รูปที่ 5.2 การแปลงเป็นภาพ NTSC ของไฟล์ภาพ

ก) ภาพต้นฉบับ (ข) ภาพ NTSC+ (ค) ภาพ NTSC-

	ขนาดภาพต้นฉบับ	ขนาดภาพ NTSC+	ขนาดภาพ NTSC-
ภาพ colorbar	65536 * 3	65536	65536
ไฟล์ภาพ	65536 * 3	65536	65536

สรุปผลการทดลอง

ในการแปลงภาพสีเป็นภาพ NTSC นั้นภาพสีเกิดจากการรวมกันของแม่สี 3 สี คือ เขียว แดง น้ำเงิน ขนาดของภาพสีนั้นจะใหญ่กว่าขนาดของภาพ NTSC เนื่องจากภาพ NTSC เกิดภาพคล้าย gray level ที่เป็นระนาบเดียวและแม้ว่าจะใช้ไฟล์ภาพ หรือ colorbar ผลที่ได้ ออกมาก็เช่นเดียวกัน นอกจากนี้ การแปลงภาพเป็นภาพ NTSC+ (เป็นภาพที่นำสัญญาณ Y+ กับ C) ได้ภาพออกมาเหมือนกับ NTSC- (เป็นภาพที่นำสัญญาณ Y- กับ C) ซึ่ง NTSC+ กับ NTSC- ใช้ในการแปลงกลับเป็นภาพสี

5.2 การทดลองแปลงไฟล์ภาพ NTSC เป็น ไฟล์ภาพสี

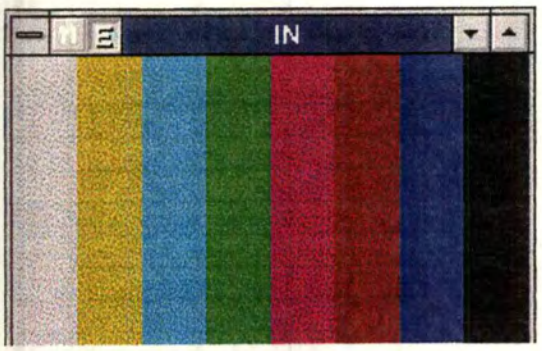
วัตถุประสงค์

1. เพื่อทำการทดสอบผลที่เกิดจากการแปลงไฟล์ภาพ NTSC เป็นไฟล์ภาพสี โดยใช้ในการแปลง mode ต่างๆ
2. เพื่อศึกษาและเปรียบเทียบภาพก่อนการแปลง และภาพหลังการแปลงเป็นสัญญาณภาพ NTSC

ขั้นตอนการทดลอง

1. ใช้โปรแกรมของปริญญานิพนธ์ที่ทำงานบนระบบวินโดว์ โดยนำภาพที่ได้จากการทดลองที่ 5.1 คือ ภาพ colorbar ที่แปลงเป็น NTSC ในจอหน้าต่างวินโดว์ in และ work มาใช้
2. ใช้โปรแกรมของปริญญานิพนธ์ที่ทำงานบนระบบวินโดว์ มาใช้โดยแปลงภาพ ในข้อ 1 โดยใช้ฟังก์ชั่นแปลงกลับ ซึ่งมีในโปรแกรมทดลองแปลงภาพด้วย mode 0 พิจารณาภาพที่เกิดขึ้นกับภาพต้นฉบับ พร้อมถ่ายภาพผลการทดลอง
3. ใช้โปรแกรมของปริญญานิพนธ์ที่ทำงานบนระบบวินโดว์ ทดลองแปลงภาพโดยใช้ mode ต่างๆ ตั้งแต่ 1 ถึง 5 ตามลำดับ เปรียบเทียบกับต้นฉบับ พร้อมถ่ายภาพการทดลอง
4. เปลี่ยนภาพการทดลองเป็นไฟล์ภาพอื่นๆ แทน colorbar แล้วทำการทดลองตามข้อ 2 และ 3

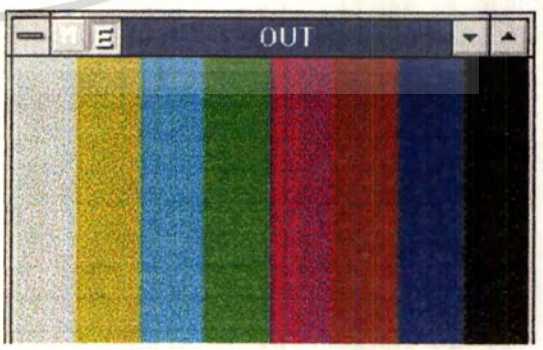
ผลการทดลอง



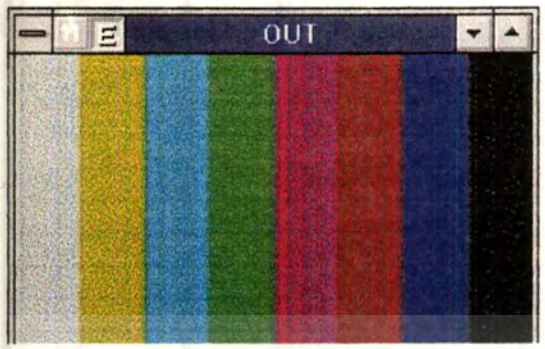
(ก.)

(ข.)

(ค.)

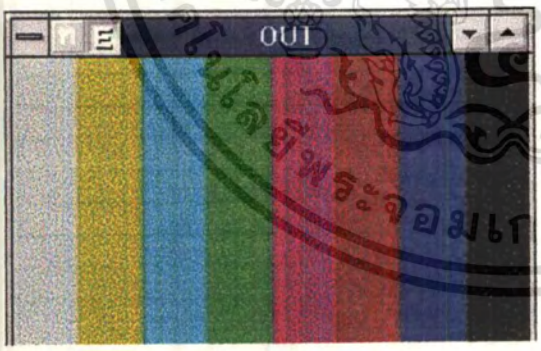


(ง.)



(ข.)

(ฉ.)



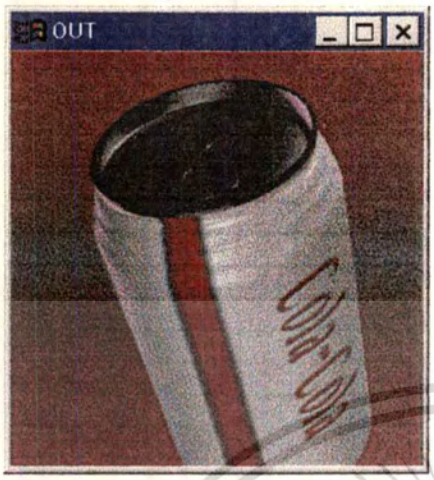
(ช.)

รูปที่ 5.3 แสดงการแปลงภาพ NTSC กลับเป็นภาพสี โดยใช้ colorbar
(ก) ภาพต้นฉบับ (ข) ภาพที่เกิดจากการแปลงกลับ mode 0
(ค) - (ช) ภาพแสดงการแปลงกลับด้วย mode 1-5

(ก.)

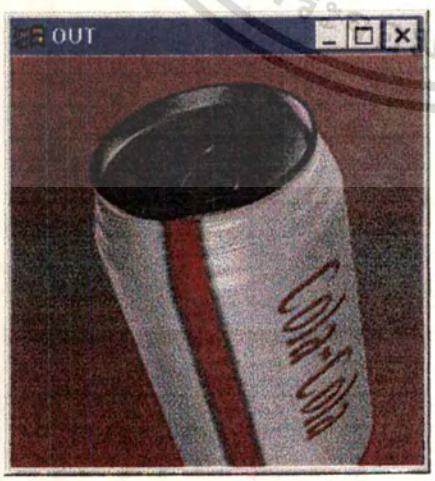
(ข.)

(ค.)

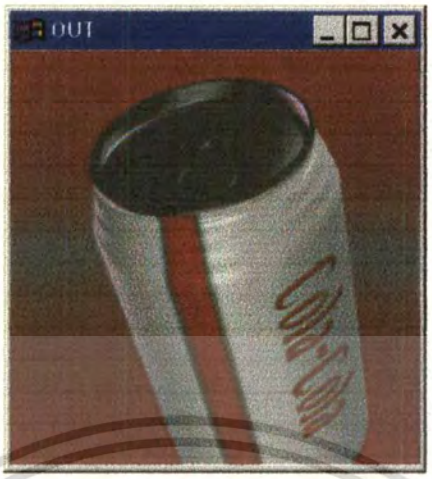


(ง.)

(จ.)



(ฉ.)



(ช.)

รูปที่ 5.4 แสดงการแปลงกลับเป็นภาพสี โดยใช้ไฟล์ภาพ
(ก) ภาพต้นฉบับ (ข) ภาพที่เกิดจากการแปลงกลับโดย mode 0
(ค) - (ช) ภาพที่เกิดจากการแปลงกลับโดย mode 1 - 5

ผลการทดลอง RMS ERROR

mode	0	1	2	3	4	5
colorbar	2.2644	1.983	2.127	1.906	1.505	2.2
file ภาพ	1.9152	2.1987	1.7164	2.1542	2.049	2.056

สรุปผลการทดลอง

ในการแปลงภาพ NTSC กลับเป็นภาพสีนั้น ภาพที่เกิดขึ้นจะมีความใกล้เคียงกับภาพต้นฉบับขึ้นกับวิธีการแปลงกลับใน mode 0 ถึง mode 5 ตามลำดับ เนื่องจากใน mode 0 จะใช้ bandpass สัญญาณ NTSC เพื่อให้ได้สัญญาณ C, mode 1จะมี line comb ในการกรองสัญญาณ NTSC, mode 2 จะไม่มี line comb แต่มี frame comb ในการกรองสัญญาณ C, mode 4 มี frame comb กรองสัญญาณ C และกรองสัญญาณ Y, mode 5 จะมี line comb, frame comb กรองสัญญาณ C และกรองสัญญาณ Y เพราะฉะนั้น mode5 จะมีการทำงานทุกขั้นตอน และเมื่อทำการวัดค่า RMS ERROR ผลที่ได้มีค่าต่ำมากในทุกๆโหมด แม้เปลี่ยนภาพจาก colorbar เป็นไฟล์ภาพอื่นๆ ได้ผลการทดลองเช่นเดียวกัน

บทที่ 6
บทสรุปและวิจารณ์

การประมวลผลภาพเพื่อลดข้อมูลภาพสีโดยวิธีการแปลงเป็นสัญญาณ NTSC จะประกอบไปด้วยขั้นตอนหลักๆ 2 ขั้นตอน คือ การแปลงภาพสี R, G, B เป็น NTSC และการแปลงภาพ NTSC เป็นภาพสี R, G, B

6.1 การแปลงภาพสี R, G, B เป็นภาพ NTSC

ซึ่งมีภาพ NTSC ประกอบไปด้วยสัญญาณ Y, I, Q มีวิธีการแปลงจาก R, G, B เป็น Y, I, Q โดยใช้สมการ

$$Y = 0.3\,r + 0.59\,G + 0.11\,B \quad ...(6.1\,A)$$

$$I = 0.6\,r - 0.28\,G - 0.32\,B \quad ...(6.1\,B)$$

$$Q = 0.21\,r - 0.52\,G + 0.31\,B \quad ...(6.1\,C)$$

การแปลงโดยวิธีดังกล่าว ใช้ความรวดเร็วสูง แต่ในการที่จะสร้างสัญญาณ NTSCได้ I, Q ต้องเปลี่ยนเป็น C โดยนำ I, Q หาร 2 แล้วบวก ofs (level / 2 = 127) จากนั้นจึงนำมารวมกันด้วยค่าคงที่ต่างๆตามตำแหน่งจุดภาพ แล้วจึงรวมกับสัญญาณ Y เพื่อไม่ให้เกิด saturate จะทำการลด seale โดยนำไปคูณ 140/255 แล้วบวกกับ 60 สำหรับในการแปลงกลับ จำเป็นต้องใช้สัญญาณ Y - C (NTSC -) ด้วย

6.2 การแปลงภาพ NTSC กลับเป็นภาพสี R, G, B

ในการแปลงสัญญาณกลับเป็นR, G, B ประกอบไปด้วยขั้นตอนหลักๆ 2 ขั้นตอน คือ การแยกสัญญาณ NTSC เป็น Y, I, G และการแปลงสัญญาณ Y, I, G เป็น R, G, B

(ก) การแยกสัญญาณ NTSC เป็น Y, I, G มีวิธีการแยกสัญญาณ 6 แบบ โดยแต่ละแบบจะมีขั้นตอนที่ต่างกันดังนี้

mode	bandpass	line comb	frame comb 1	frame comb 2
mode 0	O	X	X	X
mode 1	O	O	X	X
mode 2	O	X	O	X
mode 3	O	O	O	X
mode 4	O	X	O	O
mode 5	O	O	O	O

ขั้นตอนต่างๆ ใน mode มีหน้าที่ดังนี้

1. bandpass แยกสัญญาณ C ออกจาก NTSC+ (เป็นสัญญาณที่ได้จาก Y + C และแสดงภาพในหน้าต่าง วินโดว์ in)

2. line comb เป็นการจัดสัญญาณNTSC ก่อนที่จะทำการ bandpass เพื่อให้ได้สัญญาณ C ที่ดีขึ้น โดยสัญญาณที่ใช้เป็นสัญญาณ NTSC+

3. frame comb 1 เป็นการสร้างสัญญาณ C จากการนำสัญญาณ NTSC+ และสัญญาณ NTSC- มาทำการลบกัน

4. frame comb 2 เป็นการสร้างสัญญาณ Y จากการนำสัญญาณ NTSC+ และสัญญาณ NTSC- มาทำการบวกกัน

ในการที่แยกสัญญาณ C เพื่อไปทำการลบออกจาก NTSC+ แล้วได้สัญญาณ Y จากนั้นจะนำสัญญาณ C ไปแยกเป็นสัญญาณ I, Q เพราะฉะนั้นจะได้สัญญาณ Y, I, Q ออกมา ข้อดีของการมีขั้นตอน

แยกสัญญาณมาก ภาพสีก็จะยิ่งชัดขึ้น แต่ว่าจะต้องใช้ทั้ง สัญญาณ NTSC+ และสัญญาณ NTSC- ในการสร้างภาพสี

(ข) การสร้างสัญญาณ R, G, B โดยใช้ Y, I, Q เมื่อได้ Y, I, Qมาแล้วใช้สมการ

$$R = 1.0\,Y + 0.95\,I + 0.62\,Q \qquad \ldots(6.1.2A)$$

$$G = 1.0\,Y - 0.28\,I - 0.64\,Q \qquad \ldots(6.1.2B)$$

$$B = 1.0\,Y - 1.11\,I + 1.72\,Q \qquad \ldots(6.1.2C)$$

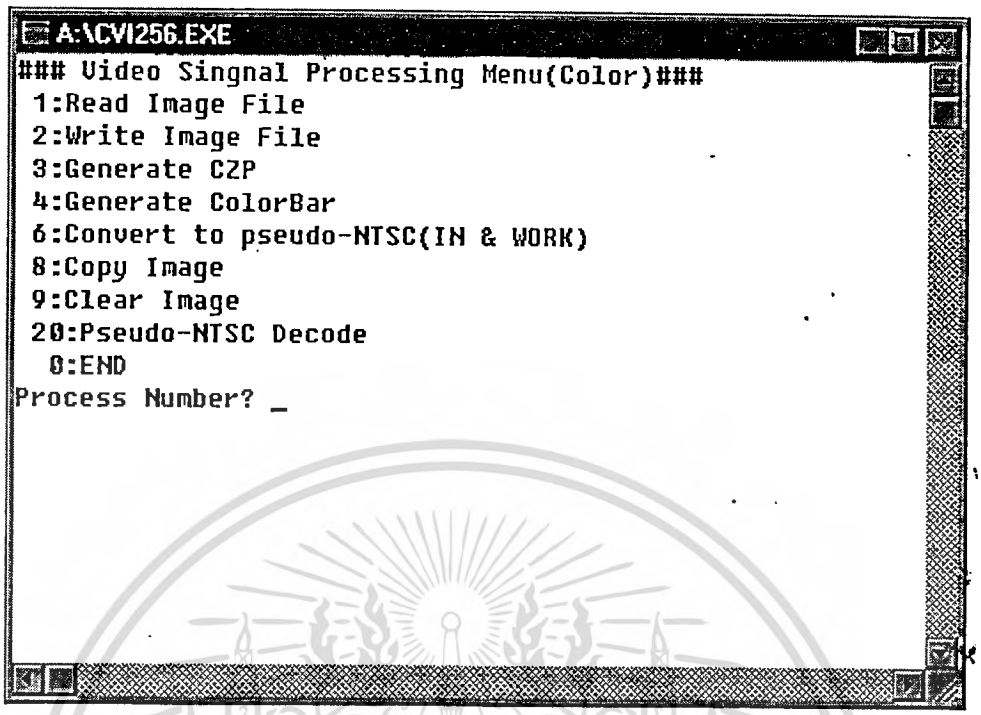
การนำ Y, I, Q ที่จะมาใช้ของสมการ ต้องพิจารณาให้ดีเพราะ I, Q ที่ได้จาก NTSC เป็นสัญญาณ I/2 , Q/2

ภาคผนวก ก

เมนูของโปรแกรม

ลักษณะโครงสร้างหน้าจอของโปรแกรมปริญญานิพนธ์ มีฟังก์ชั่นให้เลือกหลายแบบ ตามหน้าที่ ซึ่งแสดงได้ดังนี้

1. read image file คือ อ่านข้อมูลไฟล์ภาพไปแสดงที่หน้าต่าง วินโดวส์ in , out และwork โดยเราสามารถระบุได้
2. write immage file คือ ทำการนำข้อมูลภาพที่แสดงบนหน้าต่างวินโดวส์ ต่าง ๆ บรรจุข้อมูลลงไฟล์ได้
3. grerate CZP คือ การสร้างภาพวงกลมซ้อน ๆ กัน เพื่อใช้ในการทดลอง
4. generate colorbar คือ การสร้างภาพแท่งสี 7 แท่ง เพื่อนำไปใช้ในการทดลอง
5. convert to pseudo - NTSC (IN & WORK) คือ ทำการแปลงสัญญาณภาพสี R , G , B เป็นภาพ NTSC โดยนำภาพจาก IN และWORK ไปใช้ โดย out put ที่ออกมา ที่หน้าต่าง IN จะเป็น NISC -
6. clear image คือ การลบภาพที่แสดงบนหน้าต่างวินโดวส์ IN , OUT และWORK
7. pseudo - NTSC decode คือ การแปลงสัญญาณ NTSC ไปเป็นภาพสีโดยภาพจากหน้าต่าง IN และหน้าต่าง WORK จะแปลงไปออกหน้าต่าง OUT ออกมาเป็นภาพสี R , G , B
8. copy image คือ การนำภาพจากหน้าต่างวินโดวส์ หนึ่ง ไปอีกวินโดวส์หนึ่ง
9. end เป็นการแสดงสิ้นสุดการใช้งานโปรแกรมนี้ จะพบว่า โปรแกรมปริญญานิพนธ์นี้ แสดงบนวินโดวส์ จึงใช้งานง่าย



รูปที่ ก.1. แสดง ภาพเมนูของ โปรแกรม

ภาคผนวก ข.

การแปลงเป็นภาพ NTSC

การแปลงภาพสี R , G , B เป็นภาพ NTSC

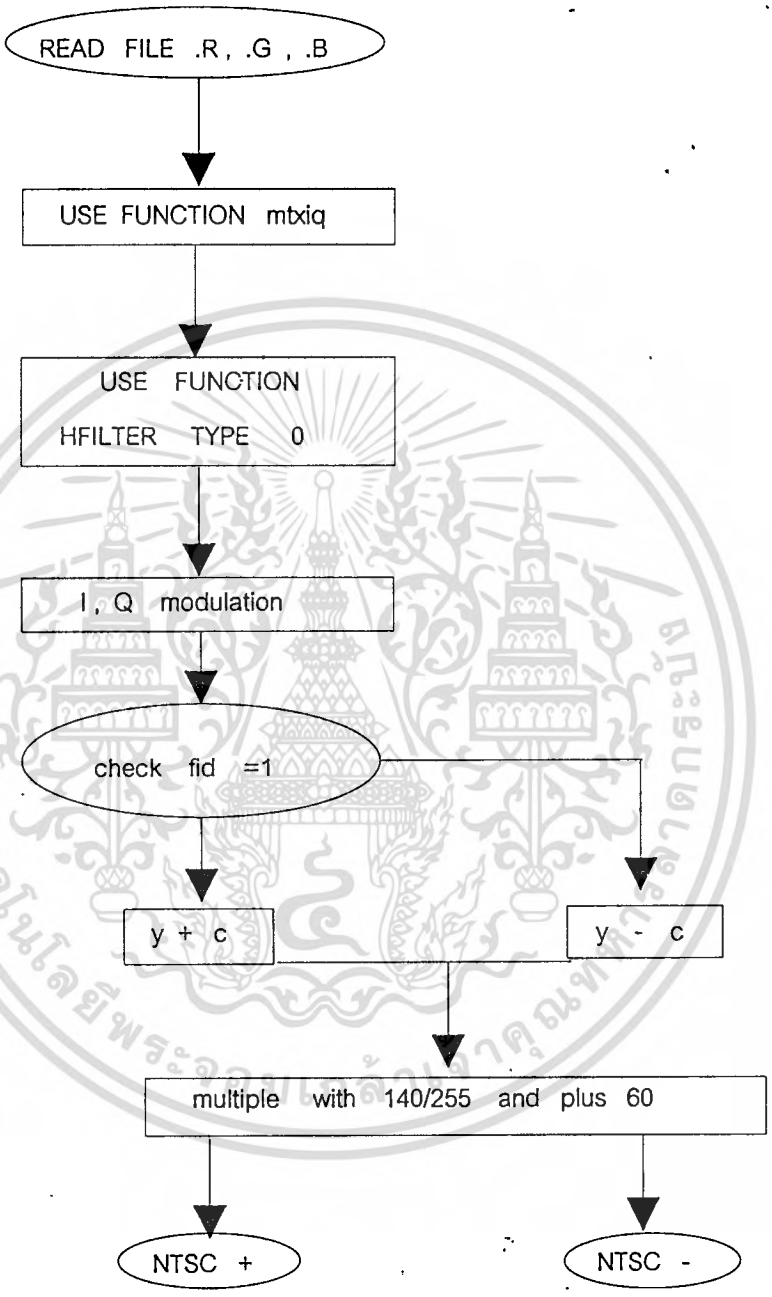
หลักการในการแปลงไฟล์ภาพสี เป็นไฟล์ภาพ NTSC แสดงได้ดังรูปที่ ข.1 โดยมีขั้นตอนต่าง ๆ ดังต่อไปนี้ นำไฟล์ภาพสี R ,G ,B ที่แยกออกมา 3 ระนาบ มาแทนในฟังก์ชั่น mtxiq ซึ่งฟังก์ชั่นนี้ มีลักษณะดังนี้

$$Y = 0.3R + 0.59G + 0.11B$$

$$I = 0.6R - 0.28G - 0.32B$$

$$Q = 0.21R - 0.52G + 0.31B$$

แต่ I,Q ที่ใช้ ต้องทำการ หารด้วย 2 แล้วบวกกับ ofs (มีค่าเท่ากับ 128) เพราะฉะนั้น จะได้ Y, I/2 + ofs , Q/2 + ofs ออกมาจากนั้นนำ I/2 + ofs และ Q/2 + ofs มา mod รวมกัน แต่ก่อนที่จะ mod ต้องนำทั้งสองไปผ่านฟังก์ชั่น hfilter ชนิด o (low pass filter) เพื่อไม่ให้เกิดการอะไรท์ซิ่งค์ ซึ่งสัญญาน I รวมกับ Q จะได้สัญญาณ C ออกมา แล้วตรวจสอบตัวแปร fld ว่าเท่ากับ 1 หรือไม่ ถ้าใช่นำสัญญาณ Y มาบวกกับ C แต่ถ้าไม่ใช่ นำสัญญาณ Y มาลบกับ C และนำสัญญาณที่ออกมาคูณกับ 140 / 255 แล้วบวกกับ 60 เพื่อเป็นการลด scale ไม่ให้เกิดการ saturate เพราะฉะนั้น จะได้สัญญาณ NTSC ออกมา



รูปที่ ข . 1 . แสดงขั้นตอนการแปลงเป็นสัญญาณ NTSC

ตัวอย่างขั้นตอนการคำนวณหาสัญญาณ C จากสัญญาณ I , Q

สมมติให้ $I = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ $Q = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$

สมการที่เราใช้คือ $C = 2(mi)I + 2(mq)Q$

โดย mi หาได้จาก การชี้ตำแหน่งในเมตริกซ์ md

mq หาได้จาก การชี้ตำแหน่งในเมตริกซ์ md เช่นเดียวกัน

md = (1 , 0 , -1 , 0)

โดยสมการ mi = md [xx % 4]

mq = md [(xx+3)%4]

โดย xx = x + my และ my = (y % 2) * 2

คำนวณตามสมการต่างๆ

1. หาค่า my ได้ $I = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ $\begin{matrix} 2 = my \\ 0 = my \end{matrix}$ $Q = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$ $\begin{matrix} 2 = my \\ 0 = my \end{matrix}$

2. หาค่า xx ได้ xx ของ $I = \begin{bmatrix} 3 & 4 \\ 3 & 4 \end{bmatrix}$ xx ของ $Q = \begin{bmatrix} 4 & 4 \\ 2 & 2 \end{bmatrix}$

3. หา $mi = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$ $mq = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

4. ได้ C $= \begin{bmatrix} -2 & 4 \\ -6 & 8 \end{bmatrix}$

ภาคผนวก ค.

การแปลงภาพ NTSC กลับเป็นภาพสี

การแปลงไฟล์ภาพ NTSC กลับเป็น ไฟล์ภาพสี R , G, B

หลักการในการแปลงไฟล์ภาพกลับของภาพ NTSC มีขั้นตอนที่สำคัญ หลายขั้นตอน และในแต่ละ mod ก็จะมีขั้นตอนที่แตกต่างกันดังนี้

1. mode o จะนำ NTSC + (สัญญาณ Y + C ที่หน้าต่างวินโดวส์ IN) มาผ่านฟังก์ชั่น hfilter ชนิด 2 (bandpass) เพื่อจะได้สัญญาณ C + ofs ออกมา แล้วนำไปลบด้วย ofs (lever /2 = 128) จะได้สัญญาณ C ออกมา จากนั้น นำ NTSC + ไปลบด้วย C ที่ได้แล้วปรับแต่ง scale ด้วยการลบด้วย 60 คูณ 255/140 จึงได้สัญญาณ Y ออกมา ในการที่จะหา I ,Q ต้อง demodulation สัญญาน C และนำสัญญาณที่ออกมาไปผ่านฟังก์ชั่น hfilter ชนิด 1 (low pass filter) เพื่อไม่ให้เกิดการอะไรท์ซิ่งค์ นอกจากนี้ต้องปรับ scale ด้วยการคูณ 255/140 จึงได้ I/2 + ofs , Q/2 + ofs ออกมา เพราะฉะนั้น สามารถหาค่า Y , I ,Q แล้วจึงนำไปแทนใน ฟังก์ชั่น Imtxiq ซึ่งมีสมการดังนี้

$$R = 1.0\,Y + 0.95I + 0.62Q \qquad \text{.....(ค.1 A)}$$

$$G = 1.0\,Y - 0.28I - 0.62Q \qquad \text{.....(ค.1 B)}$$

$$B = 1.0\,Y - 1.11I + 1.72Q \qquad \text{.....(ค.1 C)}$$

จึงได้สัญญาณภาพสี R , G , B ออกมา

2. mode 1 จะทำการ line comb ก่อน คือจะนำ NTSC + ไปผ่านฟังก์ชั่น iqvfil ซึ่งฟังก์ชั่นนี้มีหน้าที่ทำให้สัญญาณ C ที่ออกมาหลังจาก bandpass เหมือนภาพต้นฉบับมากขึ้น โดยมีสมการดังต่อไปนี้

$$\frac{\text{array}[I]}{2} - \left[\frac{\text{array}[I+1] + \text{array}[I-1]}{4}\right] + \text{ofs} = \text{array}[I]$$

โดยที่ $I = 0$ $\qquad \text{array}[I-1] = \text{ofs}$

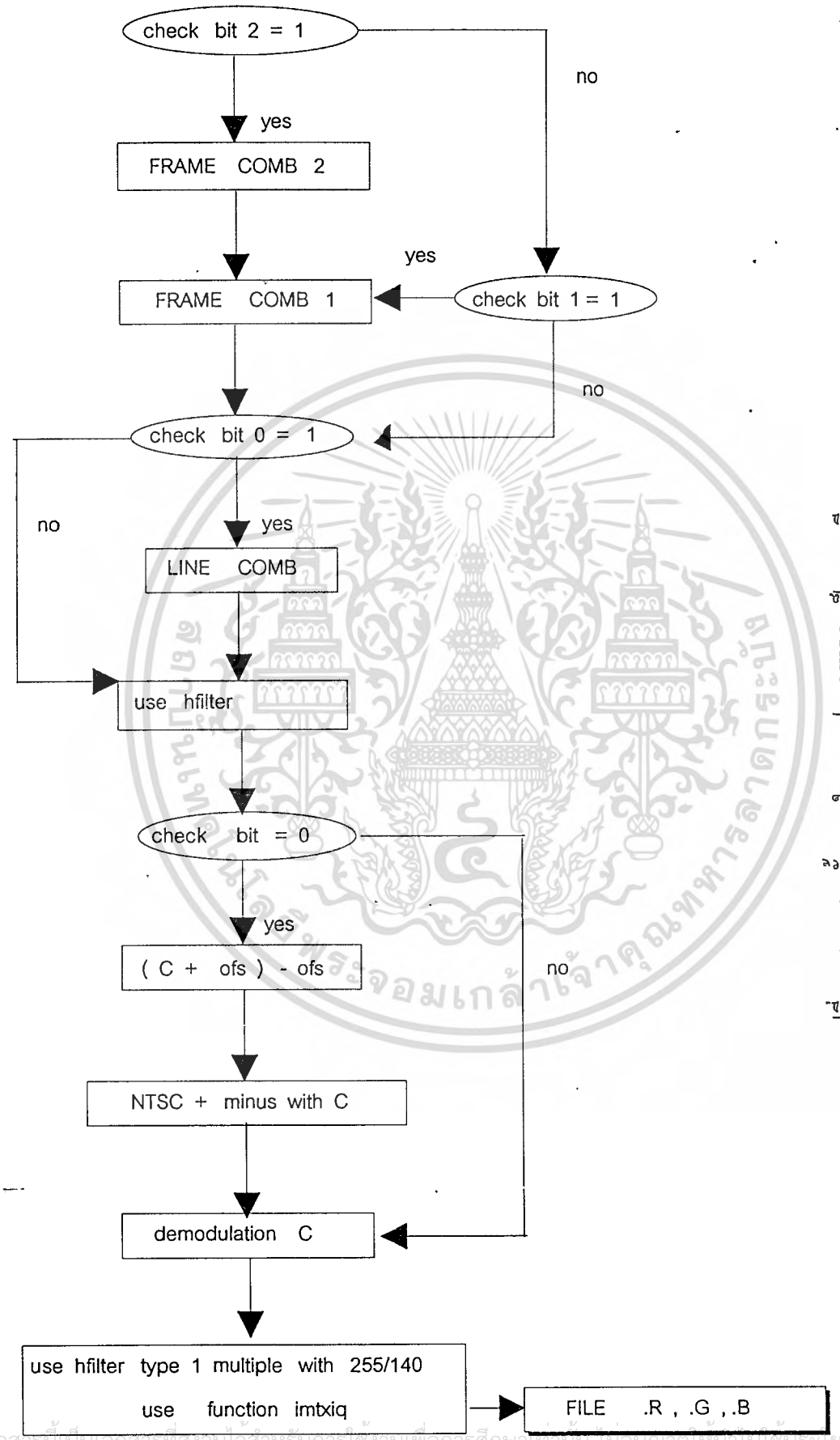
$I = Y\_size$ $\qquad \text{array}[I+1] = \text{ofs}$

$$\frac{ofs * level}{2} = 128$$

เมื่อผ่านฟังก์ชั่นนี้แล้ว ขั้นต่อไปทำตามขั้นตอน mode o

3. mode 2 จะมีขั้นตอน frame comb 1 ซึ่งเป็นขั้นตอนในการหาสัญญาณ C โดยการนำภาพ NTSC + ในจอวินโดวส์ IN กับ ภาพ NTSC - ในจอวินโดวส์ OUT มาทำการลบกัน แล้วหารด้วย 2 จึงได้ C ออกมา ต่อจากนั้นทำตาม mode o
4. mode 3 จะมีทั้งขั้นตอน frame comb 1 และ line comb ซึ่งต้องใช้สัญญาณภาพทั้ง NTSC + ในจอวินโดวส์ IN และภาพ NTSC - ในจอวินโดวส์ WORK ต่อจากนั้น ทำตามขั้นตอน mode o
5. mode 4 มีขั้นตอน frame comb 2 ซึ่งเป็นการหาสัญญาณ Y จากการนำภาพ NTSC + ในจอวินโดวส์ IN และภาพ NTSC - ในจอวินโดวส์ WORK มาทำการรวมกัน แล้วหารด้วย 2 จะได้สัญญาณ Y ออกมา แล้วนำไปทำ frame comb 1 แต่ไม่มีการทำ line comb ต่อจากนั้น ทำตาม mode o แต่ไม่มีการนำสัญญาณ NTSC + ไปลบกับสัญญาณ C
6. mode 5 จะมีทุกขั้นตอนตั้งแต่ frame comb 2 , frame comb 1 , line comb แล้วไปทำตามขั้นตอนของ mode o แต่ไม่มีการนำสัญญาณ NTSC + ไปลบกับสัญญาณ C

ให้	mode o	=	000	mode 1	=	001
	mode 2	=	010	mode 3	=	011
	mode 4	=	100	mode 5	=	101



รูปที่ ค.1. แสดงขั้นตอนในการแปลง NTSC เป็น ภาพสี

ภาคผนวก ง.

รายละเอียดและซอร์สโค้ด ของโปรแกรม

ง.1 รายละเอียดของโปรแกรม

ง.1.1 ทำงานบนเครื่องคอมพิวเตอร์ที่มีโปรแกรมไมโครซอฟท์วินโดวส์รุ่น 3.1 ขึ้นไป ควรมีหน่วยความจำอย่างน้อย 8 เมกะไบท์

ง.1.2 สามารถนำภาพที่แยกเป็นจุด R , G , B แล้วมาทำงาน
สามารถเซฟไฟล์ข้อมูลที่แปลงมาแล้วได้
สามารถสร้าง colorbar และรูปวงกลม
สามารถแปลงเป็น NTSC และแปลงกลับ

ง.2 ซอร์สโค้ดของโปรแกรม

cvideo5.c	เป็นฟังก์ชั่นหลักของคปรแกรม
ntsc.c	เป็นฟังก์ชั่นสร้างสัญญาณภายในแบบ ntsc
ntsc_de.c	เป็นฟังก์ชั่นสร้างภาพสีจากสัญญาณแบบ ntsc
hfilter.c	เป็นฟังก์ชั่นใช้ในการกรองภาพแบบต่าง ๆ
czp.c	เป็นฟังก์ชั่นสร้างภาพตัวอย่างแบบขาวดำ
colorb.c	เป็นฟังก์ชั่นสร้างภาพตัวอย่างแบบแถบสี
dirwin.c	เป็นฟังก์ชั่นสำหรับสร้างหน้าจอในแบบวินโดวส์

ภาคผนวก จ.

เกณฑ์การวัดความเหมือนจริงของภาพ(Image Fidelity)

ในการลดข้อมูลภาพนั้น จะมีส่วนหนึ่งที่เกิดผิดพลาดหรือสูญเสียไป ข้อผิดพลาดที่เกิดขึ้นนี้จะมีผลในตอนที่สร้างภาพ กลับคืนมา (Reconstruction) และค่าความผิดพลาดจะอยู่ในช่วงหนึ่งที่สามารถยอมรับได้ ดังนั้นเกณฑ์การวัดความเหมือนจริงของภาพ สามารถนำมาใช้วัดประสิทธิภาพของระบบได้ ตัวอย่างเกณฑ์ที่นิยมใช้ในการวัดคุณภาพของภาพคือ ค่า root-mean-square (rms) ของ error ระหว่างข้อมูลภาพอินพุทและข้อมูลภาพ เอาท์พุต เมื่อกำหนดให้ข้อมูลภาพอินพุตประกอบด้วย อาเรย์ขนาด N*N ของจุดภาพ f(x,y) โดยที่ x,y มีค่าเป็น 0,1,...N-1 แต่ละจุดภาพมีค่าของระดับสี เป็นเลขจำนวนเต็มของ 2^m เมื่อ m เป็นจำนวนบิตของเลข ไบนารี

สำหรับทุกค่า x และ y ในช่วง 0,1,...N-1 ค่า error ระหว่างจุดภาพอินพุตและเอาท์พุตคือ

$$e(x,y) = g(x,y) - f(x,y) \qquad \ldots(จ.1)$$

เมื่อ f(x,y) คือค่าอินพุตอิมเมต ณ จุด x,y ใดใด

g(x,y) คือค่าเอาท์พุตอิมเมต ณ จุด x,y ใดใด

ค่าเฉลี่ยของ error กำลังสองของภาพ (mean square error) คือ

$$e^2 = (1/N^2) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} (e(x,y))^2$$

$$= (1/N^2) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} [\, g(x,y) - f(x,y) \,]^2 \quad \ldots(จ.2)$$

ดังนั้นค่า rms ของ error จึงสามารถเขียนได้ดังนี้ คือ

$$e_{rms} = [\, e^2 \,]^{\frac{1}{2}} \qquad \ldots(จ.3)$$

จากวิธีการที่ใช้ในการวัดความเหมือนจริงของภาพที่กล่าวมา ไม่สามารถที่จะบ่งบอก หรือใช้เป็นเกณฑ์ในการพิจารณาได้เพียงอย่างเดียว ในกรณีของภาพ เอาท์พุต ซึ่งมีสายตาของมนุษย์เป็นตัวรับภาพจากจอภาพอีกทีหนึ่ง ระบบการมองเห็นของมนุษย์จะมีลักษณะพิเศษ กล่าวคือ ระบบการมองเห็นของตาจะไวต่อ ความเข้มแสง ในลักษณะของ Logarithmic ดังนั้น error ในบริเวณที่เป็นที่มืดของภาพจะเห็นได้ชัดกว่า eroor ที่อยู่ในบริเวณที่สว่าง และระบบการมองเห็นยังไวต่อการเปลี่ยนแปลงอย่างทันทีทันใดของระดับเทาด้วย error ที่อยู่บนขอบหรือใกล้ๆ ขอบของภาพจะเห็นได้ชัดมากกว่า error ที่อยู่ในโครงสร้างที่เป็นฉากหลังของภาพ ด้วยเหตุนี้ถึงแม้ว่าภาพสองภาพจะมีค่าของ rms error เท่ากัน แต่อาจจะปรากฏความแตกต่างของคุณภาพของการมองเห็นที่แตกต่างกันได้

```
#include <stdio.h>
#include <stdlib.h>
#ihclude <alloc.h>
#include "params.h"

#define IN     1
#define OUT    2
#define WORK   3

unsigned char huge *image_in[3];
unsigned char huge *image_out[3];
unsigned char huge *image_work[3];
unsigned char huge *image[3];


void color_image_init()
{
int i;

for(i=0;i<3;i++){
 image_in[i]  =(unsigned char huge *)farcalloc(X_SIZE,Y_SIZE);
 image_out[i] =(unsigned char huge *)farcalloc(X_SIZE,Y_SIZE);
 image_work[i]=(unsigned char huge *)farcalloc(X_SIZE,Y_SIZE);

 if((image_in[i]==NULL)||(image_out[i]==NULL)||(image_work[i]==NULL)){
        printf("Memory not enough. \n");
        exit(0);
 }
```

```
}
  coefh =(float huge *)farcalloc((size_t)MAX_TAP,sizeof(float));
  coefv =(float huge *)farcalloc((size_t)MAX_TAP,sizeof(float));
  coefhv=(float huge *)farcalloc((size_t)MAX_TAP*MAX_TAP,sizeof(float));

  if((coefh==NULL)||(coefv==NULL)||(coefhv==NULL)){
  printf("Memory not enough.\n");
  exit(0);
  }
}

void image_read(image,xsize,ysize,filename)
unsigned char huge *image;
int xsize;
int ysize;
char *filename;
{
        FILE *fp;

        if((fp=fopen(filename,"rb")) == NULL)
        {
                printf("open error (read) \n");
                exit(-1);
        }
        fread(image,xsize,ysize,fp);
        fclose(fp);
}
```

```
 void image_write(image,xsize,ysize,filename)
 unsigned char huge *image;
 int xsize;
 int ysize;
 char *filename;
 {
        FILE *fp;

        if((fp=fopen(filename,"wb")) == NULL)
        {
            printf("open error (write)\n");
            exit(-1);
        }
        fwrite(image,xsize,ysize,fp);
        fclose(fp);
}


void image_copy(image_in,image_out)
unsigned char huge *image_in;
unsigned char huge *image_out;
{
 long i;

 for(i=0;i<(long)Y_SIZE*X_SIZE;i++)
        image_out[i]=image_in[i];
}
void image_clear(image)
```

```
 unsigned char huge *image;
 {
  long i;

  for(i=0;i<(long)Y_SIZE*X_SIZE;i++)
        image[i]=0;
 }


void color_image_read(cimage,xsize,ysize,filename)
unsigned char huge *cimage[3];
int     xsize;
int             ysize;
char            *filename;
{
        int   i;
        char  fname[80];
        static char  ext[3] = {'r','g','b'};

        for(i=0;i<3;i++){
                sprintf(fname,"%s.%c",filename,ext[i]);
                image_read(cimage[i],xsize,ysize,fname);
        }
}

void color_image_write(cimage,xsize,ysize,filename)
```

```
unsigned char huge *cimage[3];
int     xsize;
int             ysize;
char            *filename;
{
        int   i;
        char  fname[80];
        static char  ext[3] = {'r','g','b'};

        for(i=0;i<3;i++){
                sprintf(fname,"%s.%c",filename,ext[i]);
                image_write(cimage[i],xsize,ysize,fname);
        }
}

void main(argc,argv)
int  argc;
char *argv[];
{
        static int command=-1;
        char    source[80];
        char            destin[80];
        char            fname[80];
        int             m,n;
        float           f;
        int             h,h1,h2;
        int             v,v1,v2;
```

```
float       a,b;
int         i,j;
int         ctl;
int         level;
float       loop;
float       fx,fy,px,py,deg;
float       qn[2];
long        nbyte;

color_image_init();
InitCDisplay(argc,argv);

while(command){
        printf("### Video Singnal Processing Menu(Color)###\n");
        printf(" 1:Read Image File\n");
        printf(" 2:Write Image File\n");
        printf(" 3:Generate CZP\n");
        printf(" 4:Generate ColorBar\n");
        printf(" 6:Convert to pseudo-NTSC(IN & WORK)\n");
        printf(" 8:Copy Image\n");
        printf(" 9:Clear Image\n");
          printf(" 20:Pseudo-NTSC Decode\n");
        printf("  0:END\n");
        printf("Process Number? ");
        scanf("%d",&command);
        switch(command){
                case 1:
                 printf(" Filename(read)? ");
```

```
                scanf("%s",source);
                printf("Which window(1:IN,2:OUT,3:WORK)?");
                scanf("%d",&n);
                switch(n){
                        case 1:
                                color_image_read(image_in,X_SIZE,Y_SIZE,source);

DisplayCImage(image_in[0],image_in[1],image_in[2],IN);//wrong code
                                break;
                        case 2:

color_image_read(image_out,X_SIZE,Y_SIZE,source);

DisplayCImage(image_out[0],image_out[1],image_out[2],OUT);
                                break;
                        case 3:

color_image_read(image_work,X_SIZE,Y_SIZE,source);

DisplayCImage(image_work[0],image_work[1],image_work[2],WORK);
                                break;
                        default:
                                break;
                }
                break;
               case 2:
                printf(" Which window(1:IN,2:OUT,3:WORK)? ");
                scanf("%d",&n);
```

```
                printf("Filename(write)? ");
                scanf("%s",destin);
                switch(n){
                        case 1:
                                color_image_write(image_in,X_SIZE,Y_SIZE,destin);
                                break;
                        case 2:
color_image_write(image_out,X_SIZE,Y_SIZE,destin);
                                break;
                        case 3:
color_image_write(image_work,X_SIZE,Y_SIZE,destin);
                                break;
                        default:
                                break;
                }
                break;
              case 3:
                printf(" CZP shape(1:circle,2:eclipse)? ");
                scanf("%d",&m);
                if((m<1)||(m>2)) break;
                printf("Which window(1:IN,2:OUT,3:WORK)? ");
                scanf("%d",&n);
                switch(n){
                        case 1:
                                czp(image_in[0],m);
                                image_copy(image_in[0],image_in[1]);
```

```
                    image_copy(image_in[0],image_in[2]);

DisplayCImage(image_in[0],image_in[1],image_in[2],IN);
                    break;
                case 2:
                    czp(image_out[0],m);
                    image_copy(image_out[0],image_out[1]);
                    image_copy(image_out[0],image_out[2]);

DisplayCImage(image_out[0],image_out[1],image_out[2],OUT);
                    break;
                case 3:
                    czp(image_work[0],m);
                    image_copy(image_work[0],image_work[1]);
                    image_copy(image_work[0],image_work[2]);

DisplayCImage(image_work[0],image_work[1],image_work[2],WORK);
                    break;
                default:
                    break;
            }
            break;
          case 4:
            printf(" Colorbar type(1:Full,2:Split)? ");
            scanf("%d",&m);
            if((m<1)||(m>2))break;
            printf(" Which window(1:IN,2:OUT,3:WORK)? ");
            scanf("%d",&n);
```

```
switch(n){
        case 1:

colorbar(image_in[0],image_in[1],image_in[2],HIGH/4*3,m-1);

DisplayCImage(image_in[0],image_in[1],image_in[2],IN);
                break;
        case 2:

colorbar(image_out[0],image_out[1],image_out[2],HIGH/4*3,m-1);

DisplayCImage(image_out[0],image_out[1],image_out[2],OUT);
                break;
        case 3:

colorbar(image_work[0],image_work[1],image_work[2],HIGH/4*3,m-1);

DisplayCImage(image_work[0],image_work[1],image_work[2],WORK);
                break;
        default:
                break;
    }
    break;
  case 6:
    printf(" Converting IN to pseudo-NTSC\n");
    ntsc_enc(image_in[0],image_in[1],image_in[2],image_out[0],0);
    printf(" Converting WORK to pseudo-NTSC\n");
```

```
ntsc_enc(image_work[0],image_work[1],image_work[2],image_out[1],1);
                    for(i=0;i<3;i++){
                        image_copy(image_out[0],image_in[i]);
                        image_copy(image_out[1],image_work[i]);
                    }
                    DisplayCImage(image_in[0],image_in[1],image_in[2],IN);

DisplayCImage(image_work[0],image_work[1],image_work[2],WORK);
                    break;
                  case 8:
                    printf(" 1: IN-->OUT \n");
                    printf(" 2: IN-->WORK \n");
                    printf(" 3: OUT-->IN \n");
                    printf(" 4: OUT-->WORK \n");
                    printf(" 5: WORK-->IN \n");
                    printf(" 6: WORK-->OUT \n");
                    scanf("%d",&n);
                    switch(n){
                        case 1:
                            for(i=0;i<3;i++){
                                image_copy(image_in[i],image_out[i]);
                            }

DisplayCImage(image_out[0],image_out[1],image_out[2],OUT);
                            break;
                        case 2:
                            for(i=0;i<3;i++){
```

```
                    image_copy(image_in[i],image_work[i]);
                }

DisplayCImage(image_work[0],image_work[1],image_work[2],WORK);
                    break;
                case 3:
                    for(i=0;i<3;i++){
                        image_copy(image_out[i],image_in[i]);
                    }

DisplayCImage(image_in[0],image_in[1],image_in[2],IN);
                    break;
                case 4:
                    for(i=0;i<3;i++){
                        image_copy(image_out[i],image_work[i]);
                    }

DisplayCImage(image_work[0],image_work[1],image_work[2],WORK);
                    break;
                case 5:
                    for(i=0;i<3;i++){
                        image_copy(image_work[i],image_in[i]);
                    }

DisplayCImage(image_in[0],image_in[1],image_in[2],IN);
                    break;
                case 6:
                    for(i=0;i<3;i++){
```

```
                    image_copy(image_work[i],image_out[i]);
                }

DisplayCImage(image_out[0],image_out[1],image_out[2],OUT);
                    break;
                default:
                    break;
            }
            break;
        case 9:
            printf(" 1: Clear IN \n");
            printf(" 2: Clear OUT \n");
            printf(" 3: Clear WORK ");
            scanf("%d",&n);
            switch(n){
                case 1:
                    for(i=0;i<3;i++){
                        image_clear(image_in[i]);
                    }

DisplayCImage(image_in[0],image_in[1],image_in[2],IN);
                    break;
                case 2:
                    for(i=0;i<3;i++){
                        image_clear(image_out[i]);
                    }

DisplayCImage(image_out[0],image_out[1],image_out[2],OUT);
```

```
                    break;
                case 3:
                    for(i=0;i<3;i++){
                        image_clear(image_work[i]);
                    }

DisplayCImage(image_work[0],image_work[1],image_work[2],WORK);
                    break;
                default:
                    break;
            }
            break;
            case 10:
            printf(" Type of YC separation filter ?\n");
            printf(" 0:C=Horizontal,Y=NTSC-C\n");
            printf(" 1:C=Horizontal+Line Comb,Y=NTSC-C\n");
            printf(" 2:C=Horizontal+Frame Comb,Y=NTSC-C\n");
            printf(" 3:C=Horizontal+Line Comb+Frame Comb,Y=NTSC-C\n");
            printf(" 4:C=Horizontal+Frame Comb,Y=Frame Comb\n");
            printf(" 5:C=Horizontal+Line Comb+Frame Comb,Y=Frame Comb");
            scanf("%d",&n);
            if((n<0)||n>5)break;

ntsc_dec(n,image_in[0],image_work[0],image_out[0],image_out[1],image_out[2]);
            DisplayCImage(image_out[0],image_out[1],image_out[2],OUT);
            break;
        case 0:
            printf(" Bye-bye.....\n");
```

```
            break;
          default:
            printf(" Not defined number\n");
            break;
          }
          EndDisplay();
        }

}
```

```
#include "params.h"

colorbar(image_r,image_g,image_b,level,type)

unsigned char huge image_r[Y_SIZE][X_SIZE];
unsigned char huge image_g[Y_SIZE][X_SIZE];
unsigned char huge image_b[Y_SIZE][X_SIZE];
int  level;
int  type;
{
        int   i,j,width,height;
        height=Y_SIZE/2;
        width=X_SIZE/8;

        for(i=0;i<Y_SIZE;i++){
                for(j=0;j<X_SIZE;j++){
                        if((type!=0)&&(i>=height)){
                        if(((j>=0)&&(j<2*width))||
                                        ((j>=4*width)&&(j<6*width)))
                                image_r[i][j]=0;
                        else
                                image_r[i][j]=level;
                        if((j>=0)&&(j<4*width))
                                image_g[i][j]=0;
                        else
                                image_g[i][j]=level;
                        if(((j>=0)&&(j<width))||
                                        ((j>=2*width)&&(j<3*width))||
                                        ((j>=4*width)&&(j<5*width))||
                                        ((j>=6*width)&&(j<7*width)))
                                image_b[i][j]=0;
                        else
```

```
                image_b[i][j]=level;
            }else{
             if(((j>=0)&&(j<2*width))||
                        ((j>=4*width)&&(j<6*width)))
                image_r[i][j]=level;
             else
                image_r[i][j]=0;
             if((j>=0)&&(j<4*width))
                image_g[i][j]=level;
             else
                image_g[i][j]=0;
            if(((j>=0)&&(j<width))||
                ((j>=2*width)&&(j<3*width))||
                ((j>=4*width)&&(j<5*width))||
                ((j>=6*width)&&(j<7*width)))
                image_b[i][j]=level;
             else
                image_b[i][j]=0;
             }
          }
       }
    }
```

```
#include <math.h>
#include "params.h"

#define  PI     3.141592
#define  PIXEL     1.0

czp(image_out,shape)
unsigned char huge image_out[Y_SIZE][X_SIZE];
int  shape;
{
        int   i,j;
        double x,y,cx,cy,d;

        if(shape != 1){
              cx = (float)PI/(float)X_SIZE;
              cy = (float)PI/(float)Y_SIZE;
        }
        else{
              if(X_SIZE>Y_SIZE){
               cx = (float)PI/((float)Y_SIZE/(float)PIXEL);
               cy = (float)PI/(float)Y_SIZE;
              }
              else{
               cx=(float)PI/((float)X_SIZE/(float)PIXEL);
               cy=(float)PI/(float)X_SIZE;
              }
        }
        for(j=0;j<Y_SIZE;j++){
          for(i=0;i<X_SIZE;i++){
              x=i-(X_SIZE/2);
              y=j-(Y_SIZE/2);
              d=cx*x*x + cy*y*y;
```

```
                d=112.0*cos(d) + 128.0;
                image_out[j][i]=d;
            }
        }
    }
```

```
#include <stdio.h>
#include <stdlib.h>
#include "params.h"

ntsc_enc(image_r,image_g,image_b,ntsc,fld)

unsigned char huge image_r[Y_SIZE][X_SIZE];
unsigned char huge image_g[Y_SIZE][X_SIZE];
unsigned char huge image_b[Y_SIZE][X_SIZE];
unsigned char huge ntsc[Y_SIZE][X_SIZE];
int  fld;
{
        int   x,y,xx,my;
        int         filter_type;
        int   mi,mq,mc,comp,ofs;
        unsigned char *image_y,*image_i,*image_q;
        static int md[4]={ 1, 0, -1, 0};

        image_y=(unsigned char *)malloc((size_t)X_SIZE*Y_SIZE);
        image_i=(unsigned char *)malloc((size_t)X_SIZE*Y_SIZE);
        image_q=(unsigned char *)malloc((size_t)X_SIZE*Y_SIZE);
        if(image_y==NULL||image_i==NULL||image_q==NULL){
                printf(" Memory not enough.\n");
                exit(0);
        }
        ofs=LEVEL/2;
        filter_type=0;
        mtxiq(image_r,image_g,image_b,image_y,image_i,image_q);
        for(y=0;y<Y_SIZE;y++){
                hfilter(&image_i[(long)y*X_SIZE],filter_type);
                hfilter(&image_q[(long)y*X_SIZE],filter_type);
        }
```

```
        for(y=0;y<Y_SIZE;y++){
                my=(y%2)*2;
                for(x=0;x<X_SIZE;x++){
                        xx=x+my;
                        mi=md[xx%4];
                        mq=md[(xx+3)%4];
                        mc=2*mi*((int)image_i[(long)y*X_SIZE+x] - ofs)
                                +2*mq*((int)image_q[(long)y*X_SIZE+x]-ofs);
                        if(fld==1) mc=-mc;
                        comp=(int)image_y[(long)y*X_SIZE+x]+mc;
                        comp=(double)comp*140./255.+60;
                        if(comp<LOW)comp=LOW;
                        if(comp>HIGH)comp=HIGH;
                        ntsc[y][x]=comp;
                }
        }
        free(image_y);
        free(image_i);
        free(image_q);
}

mtxiq(image_r,image_g,image_b,image_y,image_i,image_q)
unsigned char huge image_r[Y_SIZE][X_SIZE];
unsigned char huge image_g[Y_SIZE][X_SIZE];
unsigned char huge image_b[Y_SIZE][X_SIZE];
unsigned char huge image_y[Y_SIZE][X_SIZE];
unsigned char huge image_i[Y_SIZE][X_SIZE];
unsigned char huge image_q[Y_SIZE][X_SIZE];
{
        int   x,y;
        int       yy,ii,qq;
        double rx,gx,bx,ofs;
```

```
ofs = LEVEL/2;
for(y=0;y<Y_SIZE;y++){
        for(x=0;x<X_SIZE;x++){
                rx=image_r[y][x];
                gx=image_g[y][x];
                bx=image_b[y][x];
                yy=0.30*rx+0.59*gx+0.11*bx;
                ii=(0.60*rx-0.28*gx-0.32*bx)/2.0+ofs;
                qq=(0.21*rx-0.52*gx+0.31*bx)/2.0+ofs;
                if(yy<LOW)yy=LOW;
                if(yy>HIGH)yy=HIGH;
                if(ii<LOW)ii=LOW;
                if(ii>HIGH)ii=HIGH;
                if(qq<LOW)qq=LOW;
                if(qq>HIGH)qq=HIGH;
                image_y[y][x]=yy;
                image_i[y][x]=ii;
                image_q[y][x]=qq;
        }
}
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "params.h"

ntsc_dec(mode,ntsc1,ntsc2,image_r,image_g,image_b)
int  mode;
unsigned char huge ntsc1[Y_SIZE][X_SIZE];
unsigned char huge ntsc2[Y_SIZE][X_SIZE];
unsigned char huge image_r[Y_SIZE][X_SIZE];
unsigned char huge image_g[Y_SIZE][X_SIZE];
unsigned char huge image_b[Y_SIZE][X_SIZE];
{
        int   line_comb,frame_comb1,frame_comb2;
        int   x,y,xx,my;
        int         filter_type;
        int   mi,mq,mci,mcq,comp0,comp1;
        int         wk0,wk1,wk2,wk3,ofs;
        unsigned char buf1[X_SIZE],buf2[X_SIZE];
        unsigned char *nbuf1,*nbuf2;
        static int md[4]={ 1, 0, -1, 0};

        nbuf1=(unsigned char *)malloc((size_t)X_SIZE*Y_SIZE);
        nbuf2=(unsigned char *)malloc((size_t)X_SIZE*Y_SIZE);
        if(nbuf1==NULL||nbuf2==NULL){
                printf(" Memory not enough.\n");
                exit(0);
        }
        ofs=LEVEL/2;
        line_comb =mode&0x1;
        frame_comb1 =mode&0x2;
        frame_comb2 =mode&0x4;
        for(y=0;y<Y_SIZE;y++){
```

```
for(x=0;x<X_SIZE;x++){
        wk0=ntsc1[y][x];
        wk1=ntsc2[y][x];
        if(frame_comb2){
         wk2=(wk0+wk1)/2;
         if(wk2<LOW)wk2=LOW;
         if(wk2>HIGH)wk2=HIGH;
         nbuf1[(long)y*X_SIZE+x]=wk2;
        }
        else{
                nbuf1[(long)y*X_SIZE+x]=wk0;
        }
        if(frame_comb1||frame_comb2) {
         wk2=(wk0-wk1)/2+ofs;
         if(wk2<LOW)wk2=LOW;
         if(wk2>HIGH)wk2=HIGH;
         nbuf2[(long)y*X_SIZE+x]=wk2;
        }
        else{
                nbuf2[(long)y*X_SIZE+x]=wk0;
        }
        }
}
if(line_comb){
        for(x=0;x<X_SIZE;x++){
         for(y=0;y<Y_SIZE;y++){
                buf1[y]=nbuf2[(long)y*X_SIZE+x];
         }
         iqvfil(buf1);
         for(y=0;y<Y_SIZE;y++){
                nbuf2[(long)y*X_SIZE+x]=buf1[y];
         }
```

```
            }
        }
        for(y=0;y<Y_SIZE;y++){
         for(x=0;x<X_SIZE;x++)
                buf1[x]=nbuf2[(long)y*X_SIZE+x];
         filter_type=2;
         hfilter(buf1,filter_type);
         for(x=0;x<X_SIZE;x++)
                nbuf2[(long)y*X_SIZE+x]=buf1[x];
        }
        if(!frame_comb2){
                for(y=0;y<Y_SIZE;y++){
                        for(x=0;x<X_SIZE;x++){
                        wk1=nbuf1[(long)y*X_SIZE+x];
                        wk2=(int)nbuf2[(long)y*X_SIZE+x]-ofs;
                        wk0=((double)(wk1-wk2)-60.)*255./140.;
                        if(wk0<LOW)wk0=LOW;
                        if(wk0>HIGH)wk0=HIGH;
                        image_r[y][x]=(unsigned char)wk0;
                        }
                }
        }else{
                for(y=0;y<Y_SIZE;y++){
                        for(x=0;x<X_SIZE;x++){
                        wk0=((double)nbuf1[(long)y*X_SIZE+x]-60.)*255./140.;
                        if(wk0<LOW)wk0=LOW;
                        if(wk0>HIGH)wk0=HIGH;
                        image_r[y][x]=(unsigned char)wk0;
                        }
                }
        }
    for(y=0;y<Y_SIZE;y++){
```

```
        my=(y%2)*2;
        for(x=0;x<X_SIZE;x++){
                xx=x+my;
                mi=md[xx%4];
                mci=mi*((int)nbuf2[(long)y*X_SIZE+x]-ofs);
                mci+=ofs;
                mq=md[(xx+3)%4];
                mcq=mq*((int)nbuf2[(long)y*X_SIZE+x]-ofs);
                mcq+=ofs;
                if(mci<LOW)mci=LOW;
                if(mci>HIGH)mci=HIGH;
                if(mcq<LOW)mcq=LOW;
                if(mcq>HIGH)mcq=HIGH;
                buf1[x]=mci;
                buf2[x]=mcq;
        }
        filter_type=1;
        hfilter(buf1,filter_type);
        hfilter(buf2,filter_type);
        for(x=0;x<X_SIZE;x++){
                mci=((double)buf1[x]-ofs)*255./140.+(double)ofs;
                if(mci<LOW)mci=LOW;
                if(mci>HIGH)mci=HIGH;
                mcq=((double)buf2[x]-ofs)*255./140.+(double)ofs;
                if(mcq<LOW)mcq=LOW;
                if(mcq>HIGH)mcq=HIGH;
                image_g[y][x]=mci;
                image_b[y][x]=mcq;
        }
}
imtxiq(image_r,image_g,image_b);
free(nbuf1);
```

```
        free(nbuf2);
}

iqvfil(arry)
unsigned char arry[];
{
 int  i,i0,i1;
 int        buff[Y_SIZE];
 int        val0,val1,val2,ofs;

 ofs=LEVEL/2;
 for(i=0;i<Y_SIZE;i++){
        i0=i-1;
        i1=i+1;
        val1=arry[i];
        if(i0<0)      val0=ofs;
        else                        val0=arry[i0];
        if(i1>Y_SIZE-1) val2=ofs;
        else        val2=arry[i1];
        buff[i]=val1/2-(val0+val2)/4+ofs;
        if(buff[i]<LOW)buff[i]=LOW;
        if(buff[i]>HIGH)buff[i]=HIGH;
 }
 for(i=0;i<Y_SIZE;i++){
        arry[i]=buff[i];
 }
}
imtxiq(image_r,image_g,image_b)
unsigned char huge image_r[Y_SIZE][X_SIZE];
unsigned char huge image_g[Y_SIZE][X_SIZE];
unsigned char huge image_b[Y_SIZE][X_SIZE];
{
```

```
int   i,j,k;
double yy,ii,qq;
int   rx,gx,bx,ofs;

ofs=LEVEL/2;
for(j=0;j<Y_SIZE;j++){
        for(i=0;i<X_SIZE;i++){
                yy=image_r[j][i];
                ii=((int)image_g[j][i]-ofs)*2;
                qq=((int)image_b[j][i]-ofs)*2;
                rx=1.00*yy+0.95*ii+0.62*qq;
                gx=1.00*yy-0.28*ii-0.64*qq;
                bx=1.00*yy-1.11*ii+1.72*qq;
                if(rx<LOW)rx=LOW;
                if(rx>HIGH)rx=HIGH;
                if(gx<LOW)gx=LOW;
                if(gx>HIGH)gx=HIGH;
                if(bx<LOW)bx=LOW;
                if(bx>HIGH)bx=HIGH;
                image_r[j][i]=rx;
                image_g[j][i]=gx;
                image_b[j][i]=bx;
        }
}
}
```

```
int     nCmdShow;
{
        int i;
        _InitEasyWin();
        if(!hPrevInstance)
                if(!InitApplication(hInstance))
                        return(0);
        if(!InitInstance(hInstance,nCmdShow))
                return(0);

        main();
        PostQuitMessage(0);
        return(0);
}
InitDisplay(argc,argv)
int  argc;
char     *argv[];
{
        if(bitspixel==8){
                pal_mode=GRAY;
                InitPal();
        }
}
InitCDisplay(argc,argv)
int  argc;
char     *argv[];
{
        if(bitspixel==8){
                pal_mode=COLOR;
                InitPal();
        }
}
```

```
EndDisplay()
{
}

DisplayImage(image,position)
unsigned char huge *image;
int                                position;
{
        switch(position){
         case 1:
                ColorDisplay("IN",image,image,image,1);
                break;
         case 2:
                ColorDisplay("OUT",image,image,image,2);
                break;
         case 3:
                ColorDisplay("WORK",image,image,image,3);
                break;
        }
}
DisplayCImage(image_r,image_g,image_b,position)
unsigned char huge *image_r;
unsigned char huge *image_g;
unsigned char huge *image_b;
int                            position;
{
        switch(position){
              case 1:
                ColorDisplay("IN",image_r,image_g,image_b,1);
                break;
         case 2:
```

```
                ColorDisplay("OUT",image_r,image_g,image_b,2);
                break;
        case 3:
                ColorDisplay("WORK",image_r,image_g,image_b,3);
                break;
        }
}
ColorDisplay(title,image_r,image_g,image_b,position)
char        *title;
unsigned char huge *image_r;
unsigned char huge *image_g;
unsigned char huge *image_b;
int                         position;
{
        gimage_r=image_r;
        gimage_g=image_g;
        gimage_b=image_b;
        SendMessage(hGWnd[position-1],WM_COMMAND,0,0);
}

long FAR PASCAL MainWndProc(hWnd,message,wParam,lParam)
HWND   hWnd;
UINT    message;
WORD   wParam;
LONG   lParam;
{
        HDC             hDC,hMemDC;
        long             i,j;
        PAINTSTRUCT      ps;
//      unsigned char huge *ps_r,*ps_g,*ps_b;
        unsigned char huge *ps_r;
        unsigned char huge *ps_g;
```

```
    unsigned char huge *ps_b;
    char huge       *pd;
    //LPSTR             lpMem,lp_r,lp_g,lp_b;
    unsigned char huge *lpMem,*lp_r,*lp_g,*lp_b;
    //HANDLE          hDB,h_r,h_g,h_b;
    HANDLE          hDB;
DWORD  h_r,h_g,h_b;
    static          HBITMAP hBitmap[3][4];
    static int    winNo=0;
    static long     lWidth;
    int             *color16bit;

    switch(message){
     case WM_COMMAND:
          if(hWnd==hGWnd[0])winNo=0;
          else if (hWnd==hGWnd[1]) winNo=1;
          else if (hWnd==hGWnd[2]) winNo=2;
          else return(0);
          hDC=GetDC(hWnd);
     // add by ling
          if(bitspixel==16||bitspixel==24)
          {
               if(bitspixel==16) lWidth = X_SIZE*2L;
else lWidth = X_SIZE*3L;
               if(lWidth&3) lWidth = (lWidth&0xfffffc)+4;
               hDB=GlobalAlloc(GMEM_MOVEABLE,(DWORD)lWidth*Y_SIZE);
               if(hDB == NULL) printf("\nCan't allocate memory.");
          }
     // end add
          else hDB=GlobalAlloc(GMEM_MOVEABLE,(DWORD)X_SIZE*Y_SIZE);
          lpMem=GlobalLock(hDB);
          ps_r=gimage_r;
```

```
ps_g=gimage_g;
ps_b=gimage_b;
if(bitspixel==8)
{
        SelectPalette(hDC,hPal,FALSE);
        RealizePalette(hDC);
        pd=lpMem;
        if(pal_mode==GRAY)
                for(i=0;i<(long)X_SIZE*Y_SIZE;i++)
                {
                        j=(long)*ps_g++ *(npal-1)/HIGH;
                        *pd++=p_table[j];
                }
        else
        {
                h_r=GlobalAlloc(GMEM_MOVEABLE,(DWORD)X_SIZE*Y_SIZE);
                h_g=GlobalAlloc(GMEM_MOVEABLE,(DWORD)X_SIZE*Y_SIZE);
                h_b=GlobalAlloc(GMEM_MOVEABLE,(DWORD)X_SIZE*Y_SIZE);
                lp_r=GlobalLock(h_r);
                lp_g=GlobalLock(h_g);
                lp_b=GlobalLock(h_b);
                dither(ps_r,lp_r,CLEVELR);
                dither(ps_g,lp_g,CLEVELG);
                dither(ps_b,lp_b,CLEVELB);
                for(i=0;i<(long)X_SIZE*Y_SIZE;i++)
                {
                        j=(int)*lp_r++ *CLEVELG*CLEVELB+(int)*lp_g++
*CLEVELB+(int)*lp_b++;
                        *pd++=p_table[j];
                }
                GlobalUnlock(h_r);
                GlobalUnlock(h_g);
```

```
                    GlobalUnlock(h_b);
                    GlobalFree(h_r);
                    GlobalFree(h_g);
                    GlobalFree(h_b);
                }
                SetBitmapBits(hBitmap[winNo][0],(DWORD)X_SIZE*Y_SIZE,lpMem);
            }
        // add by ling for 16 bit and 24 bit display
            if(bitspixel == 16)
            {
                for(j=0;j<Y_SIZE;j++)
                {
                    color16bit = lpMem+(lWidth*j);
                    for(i=0;i<X_SIZE;i++,color16bit++,pd+=2,ps_b++,ps_g++,ps_r++)
                    {
                        *color16bit  = (((unsigned int) (*ps_r))>>2)<<10;
                        *color16bit += (((unsigned int) (*ps_g))>>3)<<5;
                        *color16bit += ((unsigned int) (*ps_b))>>3;
                    }
                }
                SetBitmapBits(hBitmap[winNo][0],(DWORD)lWidth*Y_SIZE,lpMem);
            }
            if(bitspixel == 24)
            {
                for(j=0;j<Y_SIZE;j++)
                {
                    pd = lpMem+(lWidth*j);
                    for(i=0;i<X_SIZE;i++,pd+=3,ps_b++,ps_g++,ps_r++)
                    {
                        pd[0] = *ps_b;
                        pd[1] = *ps_g;
pd[2] = *ps_r;
```

```
                }
            }
            SetBitmapBits(hBitmap[winNo][0],(DWORD)lWidth*Y_SIZE,lpMem);
}
// end add

            if(bitspixel == 32)
            {
                for(j=0;j<4;j++){
                  pd=lpMem;
                  for(i=0;i<(long)X_SIZE*Y_SIZE/4;i++){
                        *pd++=*ps_b++;
                        *pd++=*ps_g++;
                        *pd++=*ps_r++;
                        *pd++;
                  }
                  SetBitmapBits(hBitmap[winNo][j],(DWORD)X_SIZE*Y_SIZE,lpMem);
                }
            }
            GlobalUnlock(hDB);
            GlobalFree(hDB);
            ReleaseDC(hWnd,hDC);
            InvalidateRect(hWnd,NULL,TRUE);
            break;
case WM_CREATE:
            hDC=GetDC(hWnd);
            if(winNo==0){
                bitspixel=GetDeviceCaps(hDC,BITSPIXEL);
                if(bitspixel!=8&&bitspixel!=32&&bitspixel!=16&&bitspixel!=24){
                  printf("cannot support your display.\n");
                  return(1);
                }
```

```
        }
        j=bitspixel/8;
        hMemDC=CreateCompatibleDC(hDC);
        if(bitspixel==16||bitspixel==24)
        {
                j=1;
        }
        for(i=0;i<j;i++){
                hBitmap[winNo][i]=CreateCompatibleBitmap(hDC,X_SIZE,Y_SIZE/j);
                SelectObject(hMemDC,hBitmap[winNo][i]);
                SelectObject(hMemDC,GetStockObject(BLACK_BRUSH));
                Rectangle(hMemDC,0,0,X_SIZE,Y_SIZE/j);
        }
        DeleteDC(hMemDC);
        ReleaseDC(hWnd,hDC);
        InvalidateRect(hWnd,NULL,TRUE);
        winNo++;
        if(winNo==3){
                winNo=0;
        }
        break;
case WM_PAINT:
        if(hWnd==hGWnd[0])winNo=0;
        else if (hWnd==hGWnd[1])winNo=1;
        else if (hWnd==hGWnd[2])winNo=2;
        else return 0;
        BeginPaint(hWnd,&ps);
        hMemDC=CreateCompatibleDC(ps.hdc);
        if(bitspixel==8){
                SelectPalette(ps.hdc,hPal,FALSE);
                RealizePalette(ps.hdc);
        }
```

```
                    j = bitspixel/8;
                    if(bitspixel==16||bitspixel==24) j=1;
                    for (i=0;i<j;i++){
                        SelectObject(hMemDC,hBitmap[winNo][i]);
                        BitBlt(ps.hdc,0,Y_SIZE/j*i,X_SIZE,Y_SIZE/j,hMemDC,0,0,SRCCOPY);
                    }
                    DeleteDC(hMemDC);
                    EndPaint(hWnd,&ps);
                    break;
                case WM_DESTROY:
                    PostQuitMessage(0);
                    break;
                default:
                    return(DefWindowProc(hWnd,message,wParam,lParam));
            }
            return(0);
}
BOOL InitInstance(hInstance,nCmdShow)
HANDLE  hInstance;
int     nCmdShow;
{
        RECT  rect;
        int     i;

        rect.top=0;
        rect.left=0;
        rect.bottom=rect.top+Y_SIZE;
        rect.right=rect.left+X_SIZE;

        AdjustWindowRect(&rect,WS_OVERLAPPEDWINDOW,FALSE);
        for(i=0;i<3;i++){
        /*hGWnd[i]=CreateWindow("ImageClass","Image Processing",
```

```
                WS_OVERLAPPEDWINDOW,X_IN_POS,Y_IN_POS+i*Y_POS_STEP,
                rect.right-rect.left,rect.bottom-rect.top,NULL,NULL,hInstance,NULL);*/
               hGWnd[i]=CreateWindow("ImageClass","Image Processing",
                WS_OVERLAPPEDWINDOW,X_IN_POS+i*X_POS_STEP,Y_IN_POS,
                rect.right-rect.left,rect.bottom-rect.top,NULL,NULL,hInstance,NULL);

               if(hGWnd[i]==NULL)
                return(FALSE);
    }
    for(i=0;i<3;i++){
                SetWindowText(hGWnd[i],wTitle[i]);
                ShowWindow(hGWnd[i],nCmdShow);
                UpdateWindow(hGWnd[i]);
    }
    return(TRUE);
}

BOOL InitApplication(hInstance)
HANDLE  hInstance;
{
                WNDCLASS  wc;

                wc.style=CS_HREDRAW|CS_VREDRAW;
                wc.lpfnWndProc=MainWndProc;
                wc.cbClsExtra=0;
                wc.cbWndExtra=0;
                wc.hInstance=hInstance;
                wc.hIcon=LoadIcon(hInstance,"image");
                wc.hCursor=LoadCursor(NULL,IDC_ARROW);
                wc.hbrBackground=GetStockObject(WHITE_BRUSH);
                wc.lpszMenuName="MainMenu";
                wc.lpszClassName="ImageClass";
```

```
                }
        }
        else{
                if(nFreeCol<CLEVELR*CLEVELG*CLEVELB){
                        printf("Palette not enough.\n");
                        exit(0);
                }
                npal=CLEVELR*CLEVELG*CLEVELB;
                for(i=0;i<npal;i++){
                 pLogPal->palPalEntry[i].peRed   =((i/CLEVELG/CLEVELB)%CLEVELR)*HIGH/(CLEVELR-1);
                 pLogPal->palPalEntry[i].peGreen =((i/CLEVELB)%CLEVELG)*HIGH/(CLEVELG-1);
                 pLogPal->palPalEntry[i].peBlue =(i%CLEVELB)*HIGH/(CLEVELB-1);
                 pLogPal->palPalEntry[i].peFlags  =0;
                }
        }
        pLogPal->palNumEntries=npal;
        hPal=CreatePalette(pLogPal);
        LocalUnlock(hLocPal);
        LocalFree(hLocPal);
        hBitmap = CreateCompatibleBitmap(hDC,npal,1);
        hMemDC=CreateCompatibleDC(hDC);
        SelectObject(hMemDC,hBitmap);
        SelectPalette(hMemDC,hPal,FALSE);
        RealizePalette(hMemDC);
        hMem=GlobalAlloc(GHND,npal);
        lpMem=GlobalLock(hMem);
        for(i=0;i<npal;i++)
                SetPixel(hMemDC,i,0,PALETTERGB(
                        pLogPal->palPalEntry[i].peRed,
                        pLogPal->palPalEntry[i].peGreen,
                        pLogPal->palPalEntry[i].peBlue));
        GetBitmapBits(hBitmap,(long)npal,lpMem);
```

```
for(i=0;i<npal;i++)
        p_table[i]=*lpMem++;
 GlobalUnlock(hMem);
 DeleteObject(hBitmap);
 GlobalFree(hMem);
}
```

```
#include "params.h"
static int error[X_SIZE],error1[X_SIZE],error2[X_SIZE];
dither(image_in,image_out,nq)
unsigned char huge image_in[Y_SIZE][X_SIZE];
unsigned char huge image_out[Y_SIZE][X_SIZE];
int                                     nq;
{
        int  i,j;
        int       d;

        for(i=0;i<X_SIZE;i++){
                d_quantize((int)image_in[0][i],&image_out[0][i],nq);
                        error1[i]=(int)image_in[0][i]-d_iquantize(image_out[0][i],nq);
                }
        for(i=0;i<X_SIZE;i++){
                d_quantize((int)image_in[1][i],&image_out[1][i],nq);
                        error2[i]=(int)image_in[1][i]-d_iquantize(image_out[1][i],nq);
        }
        for(i=2;i<Y_SIZE;i++){
                d_quantize((int)image_in[i][0],&image_out[i][0],nq);
                error[0]=(int)image_in[i][0]-d_iquantize(image_out[i][0],nq);
                d_quantize((int)image_in[i][1],&image_out[i][1],nq);
                error[1]=(int)image_in[i][1]-d_iquantize(image_out[i][1],nq);
                for(j=2;j<X_SIZE-2;j++){
                 d=(error1[j-2]+error1[j-1]*3+error1[j]*5
                 +error1[j+1]*3+error1[j+2]
                 +error2[j-2]*3+error2[j-1]*5+error2[j]*7
                 +error2[j+1]*5+error2[j+2]*3
                 +error[j-2]*5+error[j-1]*7)/48;
                 d_quantize((int)image_in[i][j]+d,&image_out[i][j],nq);
                 error[j]=(int)image_in[i][j]+d-d_iquantize(image_out[i][j],nq);
                }
```

```
                d_quantize((int)image_in[i][X_SIZE-2],&image_out[i][X_SIZE-2],nq);
                error[X_SIZE-2]=(int)image_in[i][X_SIZE-2]-d_iquantize(image_out[i][X_SIZE-2],nq);
                d_quantize((int)image_in[i][X_SIZE-1],&image_out[i][X_SIZE-1],nq);
                error[X_SIZE-1]=(int)image_in[i][X_SIZE-1]-d_iquantize(image_out[i][X_SIZE-1],nq);
                for(j=0;j<X_SIZE;j++){
                        error1[j]=error2[j];
                        error2[j]=error[j];
                }
        }
}
d_quantize(in,pout,nq)

int    in;
unsigned char huge *pout;
int    nq;
{
        *pout=(float)in*(nq-1)/HIGH+0.5;
        if(*pout<0)   *pout=0;
        if(*pout>nq-1) *pout=nq-1;
}
d_iquantize(in,nq)
unsigned char huge in;
int                    nq;
{
 return((int)in*HIGH/(nq-1));
}
```

```
#include "params.h"

static   int  filtsize[3] = {16,16,32};
static   double coef[3][32]={
                                { 0.25684, 0.23079, 0.16224, 0.07573, 0.00000, -0.04403,
                                 -0.05076, -0.02997, 0.00000, 0.02182, 0.02668, 0.01640,
                                  0.00000, -0.01247, -0.01539, -0.00948,
                                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

                                { 0.50781, 0.32031, 0.00000, -0.10547, 0.00000, 0.06250,
                                  0.00000, -0.04297, 0.00000, 0.03125, 0.00000, -0.02344,
                                  0.00000, 0.01563, 0.00000, -0.02344,
                                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

                                { 0.49219, 0.0, -0.31748, 0.0, 0.0, 0.0, 0.09729, 0.0,
                                  0.0, 0.0, -0.04972, 0.0, 0.0, 0.0, 0.02706, 0.0,
                                  0.0, 0.0,-0.010554, 0.0, 0.0, 0.0,-0.008889, 0.0,
                                  0.0, 0.0, -0.002462, 0.0, 0.0, 0.0, 0.001041, 0.0} };

hfilter(arry,type)
unsigned char arry[];
int         type;
{
        int                 i,j,k;
        int                 i0,i1;
        unsigned char buff[X_SIZE];
        double   sum,val0,val1,min,max,ofs;

        min=LOW;
        max=HIGH;
        ofs=LEVEL/2;
        for(i=0;i<X_SIZE;i++){
```

```
        sum=((double)arry[i]-ofs)*coef[type][0];
        for(k=1;k<filtsize[type];k++){
                i0=i-k;
                i1=i+k;
                if(i0<0) val0=0;
                else val0=(int)arry[i0]-ofs;
                if(i1>X_SIZE-1) val1 = 0;
                else val1=(int)arry[i1]-ofs;
                sum+=(double)(val0+val1)*coef[type][k];
        }
        sum+=ofs;
        if(sum<min)sum=min;
        if(sum>max)sum=max;
        buff[i]=sum;
    }
    for(i=0;i<X_SIZE;i++)
            arry[i]=buff[i];
}
```

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้จะสำเร็จลงไม่ได้ถ้าปราศจากความช่วยเหลือจาก รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา ในการให้ความช่วยเหลือและชี้แนะข้อผิดพลาดต่างๆ อย่างมาก จนสามารถทำปริญญานิพนธ์ได้ตามเป้าหมาย ขอขอบคุณ อ. เทอดศักดิ์ ลิ่วหาทอง ในการศึกษาและแนะนำอัลกอริทึมต่างๆของโปรแกรม ขอขอบคุณพี่ ธีระณัฐ ที่ทุ่มเทแรงกายและแรงใจช่วยในการศึกษาภาษาที่เขียนโปรแกรม ขอขอบคุณอาจารย์ทุกท่านที่ประสิทธิประสาทวิชาให้ และขอขอบคุณเพื่อนๆที่ได้ช่วยกันออกแรงในการที่ทำให้ปริญญานิพนธ์เล่มนี้เป็นรูปเป็นร่างขึ้นมาได้ สุดท้ายนี้ขอขอบคุณบิดามารดาผู้เล็งเห็นความสำคัญของการศึกษา ต้องขอขอบคุณทุกท่านอย่างสูงมา ณ. ที่นี้ด้วย

อภิพล คุณาภิบาล

อัครเรศ แตงสุวรรณ

หนังสืออ้างอิง

1. IMAGE PROCESSING THEORY , ALGORITHMS & ARCHITECTURES
   M.A. SID - AHMED MC GRAW - HILL INTERNATIONAL 1995 PAGE 377 - 511
2. TEXT BOOK OF COLOUR TELEVISION ENGINEET\RING ดร. ธวัช เมฆสวรรค์
   โยชิคาซึ ซาวามูระ สำนักพิมพ์ ดวงกมล 2528 หน้า 23 - 63
3. ทฤษฎีโทรทัศน์ หลักการและเครื่องรับโทรทัศน์ ประกิจ ตังติศานนท์ สถาบันเทคโนฯ
   เจ้าคุณทหารลาดกระบัง 2527 หน้า 1 - 16