โปรแกรมแต่งหน้าจากภาพถ่ายวีดีโอ

VDO IMAGE MAKEUP PROGRAM

จัดทำโดย

1. นายธรากร ชงเชื้อ ปี 4 D รหัส 321118
2. นายบันเทิง พันธุวสิ ปี 4 D รหัส 321160

อาจารย์ที่ปรึกษา

1. รศ.ดร.ชม กิ่มปาน
2. อ.สมศักดิ์ มัตะเถา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต
สาขาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2535

ปริญญานิพนธ์ปีการศึกษา 2545
ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
เรื่อง โปรแกรมแต่งภาพใบหน้าจากภาพถ่ายวิดีโอ

ผู้จัดทำ

นาย ธรากร ชงเชื้อ
นาย บันเทิง พันคุวสี

.....อาจารย์ที่ปรึกษา
(อ. สมศักดิ์ มิตะถา)

.....อาจารย์ที่ปรึกษา
(รศ.ดร. ชม กิ้มปาน)

ปีการศึกษา 2535

โปรแกรมแต่งภาพใบหน้าจากภาพวิดีโอ

โดย

นาย ธรากร ชงเชื้อ

นาย บันเทิง พันธุวสี

อาจารย์ที่ปรึกษา

รศ.ดร. ชม กิ้มปาน

อ. สมศักดิ์ มิตะถา

สารบัญ

โปรแกรมแต่งภาพใบหน้าจากภาพถ่ายวีดีโอ

ผู้จัดทำ
1. นายธรากร ชงเชื้อ
2. นายบันเทิง พันธุวสิ

รศ.ดร. ชม กิ้ม ปาน
อ. สมศักดิ์ มิตะถา อาจารย์ที่ปรึกษา

ปีการศึกษา 2535

บทคัดย่อ

ในระบบปฏิบัติการที่สนับสนุนโปรแกรมกราฟฟิกส์ คือ MICROSOFT WINDOWS มีลักษณะการติดต่อกับผู้ใช้แบบรูปภาพ และมีความสามารถในการติดต่อกับโปรแกรมอื่น ๆ บนวินโดวส์เอง จึงสามารถนำภาพที่ได้จากโปรแกรมบนวินโดวส์มาใช้กับโปรแกรมอื่น ๆ อีก ได้

ดังนั้นในโครงงานชิ้นนี้ เป็นการสร้างแอพพลิเคชันโปรแกรมที่ทำงานบน MS WINDOWS โดยมีลักษณะการทำงาน คือ ทำการดึงภาพของใบหน้าที่เก็บจากกล้องวีดีโอออกมาแสดง แล้วสามารถเลือกทำการเปลี่ยนทรงผม เปลี่ยนสีผม หรือเปลี่ยนใบหน้าใหม่จนกว่าผู้ใช้จะพอใจ มีการส่งผ่าน อินพุท เอาท์พุท ผ่านทางคลิปบอร์ดของวินโดวส์ หรือเก็บใส่ไฟล์เพื่อให้สามารถนำภาพต้นแบบที่เปลี่ยนทรงผม หรือสีผม ออกไปใช้ในโปรแกรมอื่น ๆ โดยโครงงานนี้ อาจนำไปใช้ประโยชน์ด้านการตกแต่งใบหน้า ร้านทำผม การสืบหาใบหน้าคนร้าย ซึ่งเป็นการแต่งภาพจากภาพถ่ายจริง

VDO Image Makeup Program

BY

1. MR.TARAKORN CHONGCUA
2. MR.BUNTERNG PUNTUWASEE

DR. CHOM KIMPAN

SOMSAK MITTATA ADVISOR

ACADEMIC YEAR 1992

ABSTACT

Microsoft Windows is graphics operating system. It communicate to user with Graphics User Interface (GDI) and has ability to connect with others application.So graphics output can be used by another program.

This project is application program run on MS WINDOWS. Program works with face image that bring from VDO camera. Program can select face to display and change hair or hair color by users. Output and input of this program can communicate with another program by bitmap file or clipboard. This application can be used in hair cut or robber detect that can see real image.

บทที่ 1 พื้นฐานของวินโดวส์

วินโดวส์คืออะไร

วินโดวส์เป็น operating environment ที่รันใต้ดอส เป็นระบบกราฟฟิก สามารถรันงานหลายงานพร้อมกันได้ วินโดวส์ได้เตรียม routine ภายในไว้มากมายทำให้การใช้ pull-down menu , scroll bar, dialog boxes , icons และระบบการติดต่อกับผู้ใช้แบบกราฟฟิกอื่นๆที่ช่วยให้การใช้งานง่าย

วินโดวส์ถือว่าจอแสดงผล แป้นพิมพ์ เม้าส์ เครื่องพิมพ์ พอร์ทสื่อสาร และเวลาของระบบ เป็น device independent ทำให้สามารถรันโปรแกรมได้โดยไม่ขึ้นกับ ฮาร์ดแวร์

ความสามารถของวินโดวส์

วินโดวส์มีข้อดีเหนือกว่าระบบดอสมากทั้งในแง่ของผู้ใช้และผู้พัฒนาโปรแกรม วินโดวส์มีคุณลมบัติที่สำคัญ 3 ประการคือ

- ระบบการติดต่อกับผู้ใช้แบบกราฟฟิก(graphics-oriented user interface)
- ความสามารถในการทำงานหลายอย่างได้พร้อมกัน(multitasking)
- การทำงานที่ไม่ต้องขึ้นกับ ฮาร์ดแวร์(hardware independence)

ซึ่งทั้ง 3 สิ่งนี้ไม่ใช่สิ่งใหม่ แต่การนำทั้งสามสิ่งมารวมเข้าด้วยกันในระบบไมโครคอมพิวเตอร์นับว่าเป็นแนวความคิดใหม่

ระบบการติดต่อกับผู้ใช้ที่เป็นมาตราฐาน (Standardized User Interface)

ในคุณลมบัติหลัก 3 ประการของวินโดวส์ ระบบการติดต่อกับผู้ใช้ที่เป็นกราฟฟิกดูจะเป็นสิ่งที่สังเกตได้ง่ายที่สุดและสำคัญที่สุดสำหรับผู้ใช้ การติดต่อกับผู้ใช้จะใช้รูปภาพแทนไดร์ฟ ไฟล์ ไดเร็กทอรี และแทนคำสั่งและการกระทำต่างๆ วินโดวส์โปรแกรมส่วนมากจะสามารถใช้ได้ทั้งแป้นพิมพ์และเม้าส์ แม้ว่าจะสามารถใช้แป้นพิมพ์ควบคุมการทำงานของโปรแกรมได้ แต่การใช้เม้าส์จะช่วยให้สะดวกและง่ายกว่ามาก

การทำงานหลายอย่างได้พร้อมกัน (multitasking)

ระบบมัลติทาสกิง คือระบบที่สามารถทำให้มีหลายๆโปรแกรม หรือโปรแกรมเดียวกันหลายชุดสามารถทำงานพร้อมกัน ผู้ใช้สามารถเปลี่ยนสลับการทำงานไปมาระหว่างโปรแกรมต่างๆได้ เปลี่ยนขนาดของหน้าต่างและสามารถแลกเปลี่ยนข้อมูลระหว่างหน้าต่างได้

การจัดการหน่วยความจำ (Memory Management)

หน่วยความจำเป็นทรัพยากรที่สำคัญมากในระบบวินโดวส์ ถ้ามีโปรแกรมมากกว่าหนึ่งโปรแกรมทำงานในเวลาเดียวกัน แต่ละโปรแกรมต้องแบ่งหน่วยความจำกันใช้โดยไม่ทำให้หน่วยความจำไม่พอ เมื่อโปรแกรมใหม่เริ่มการทำงานในขณะที่โปรแกรมเก่าเลิกการทำงาน

หน่วยความจำอาจแยกเป็นส่วนๆได้ วินโดวส์สามารถรวบรวมหน่วยความจำที่ว่างมาใช้ได้

การจัดลำดับของข้อมูลเข้า

ทรัพยากรที่ใช้ร่วมกันที่สำคัญอีกอย่างหนึ่งก็คือข้อมูลจากแป้นพิมพ์และเม้าส์ โปรแกรมจะไม่สามารถจัดการกับข้อมูลโดยตรงได้ ต้องใช้ฟังก์ชันของวินโดวส์แทน ในระบบวินโดวส์แต่ละโปรแกรมจะไม่จัดการอ่านข้อมูลจากแป้นพิมพ์หรือเม้าส์เอง วินโดวส์จะเป็นผู้รับข้อมูลเข้าทั้งหมดจากแป้นพิมพ์ เม้าส์ และเวลาของระบบ จัดไว้ในคิวของระบบ ซึ่งคิวจะเปลี่ยนทิศทางของข้อมูลเข้าไปยังโปรแกรมที่เหมาะสม โดยการลอกข้อมูลจากคิวของระบบไปยังคิวของโปรแกรม ดังนั้นเมื่อโปรแกรมต้องการอ่านข้อมูลเข้าก็จะมาอ่านได้จากคิวของมันและส่งข่าวสารไปยังหน้าต่างที่ถูกต้อง

ข่าวสาร (messages)

หลักการในการใช้ข้อมูลในระบบการทำงานหลายอย่างพร้อมกัน ก็คือระบบข่าวสารของวินโดวส์โดยมองจากโปรแกรม ข่าวสารสามารถชี้ให้เห็นว่ามีเหตุการณ์ที่น่าสนใจเกิดขึ้น ซึ่งอาจจะจำเป็นหรือไม่จำเป็นต้องโต้ตอบ โดยเหตุการณ์อาจเกิดขึ้นจากผู้ใช้ เช่นการกดปุ่มเม้าส์หรือการเลื่อนเม้าส์ การเปลี่ยนขนาดหน้าต่างหรือการเลือกเมนู ข่าวสารอาจเกิดจากโปรแกรมเป็นผู้สร้างเองก็ได้ เช่น โปรแกรมต้องการเขียนหน้าจอแสดงผลใหม่ ก็จะส่งข่าวสาร"update window" วินโดวส์เองก็สามารถส่งข่าวสารได้ เช่นกรณีของข่าวสาร"close session" ซึ่งจะส่งให้โปรแกรมเมื่อต้องการเลิกการทำงานของโปรแกรม

การไม่ขึ้นกับอุปกรณ์ (Device Independence)

ความสามารถอีกอย่างของวินโดวส์ก็คือการไม่ขึ้นกับอุปกรณ์อาร์ดแวร์ วินโดวส์ทำให้โปรแกรมเมอร์ไม่จำเป็นต้องคอยกังวลในการจัดการกับอุปกรณ์เช่น จอแสดงผลหรือเครื่องพิมพ์ทุกชนิด เพราะจะมีไดรเวอร์คอยจัดการแทน ซึ่งไดรเวอร์ส่วนหนึ่งจะมาจากผู้ผลิตอุปกรณ์นั้นๆอีกส่วนวินโดวส์จะจัดเตรียมไว้ให้ โปรแกรมจะติดต่อกับวินโดวส์แทนที่จะต้องติดต่อกับอุปกรณ์โดยตรงทำให้โปรแกรมไม่จำเป็นต้องรู้ว่าใช้ เครื่องพิมพ์ชนิดไหนอยู่

แนวคิดและวิธีการทำงานของวินโดวส์

สามารถแบ่งหลักการทำงานของวินโดวส์ออกเป็น 2 ส่วนใหญ่ๆคือ คุณสมบัติของวินโดวส์ซึ่งมองเห็นได้ เช่น เมนู ไอคอน เป็นต้น และการดำเนินงานเบื้องหลังเช่น การใช้ข่าวสาร การเข้าถึงฟังก์ชัน เป็นต้น สำหรับผู้พัฒนาวินโดวส์โปรแกรมที่จะสื่อสารได้อย่างมีประสิทธิภาพระหว่างกัน คุณสมบัติทั้งหมดของวินโดวส์ได้ถูกตั้งชื่อและวิธีใช้ ดังต่อไปนี้

หน้าต่าง (window)

สำหรับผู้ใช้ หน้าต่างของวินโดวส์คือส่วนที่เป็นสี่เหลี่ยมของจอแสดงผลซึ่งใช้ในการติดต่อ

ระหว่างผู้ใช้กับโปรแกรมที่สร้างหน้าต่างนั้น สำหรับโปรแกรม หน้าต่างเป็นทรัพยากรสิ่งหนึ่งของจอแสดงผลที่อยู่ภายใต้การควบคุมของโปรแกรม โปรแกรมสร้างและควบคุมทุกอย่างเกี่ยวกับหน้าต่างหลักซึ่งรวมทั้งขนาดและรูปร่าง เมื่อผู้ใช้เริ่มการทำงานของโปรแกรมหน้าต่างจะถูกสร้างขึ้นมา แต่ละครั้งที่ผู้ใช้เลือกคำสั่งบนหน้าต่างโปรแกรมจะทำงานตามคำสั่งนั้น การปิดหน้าต่างทำให้โปรแกรมเลิกทำงาน

ตัวอย่างเช่น ในขณะที่หน้าต่างแทนระบบติดต่อกับผู้ใช้แบบกราฟฟิก มันก็เป็นการรวมของการทำงานพร้อมๆกันและการไม่ขึ้นกับฮาร์ดแวร์ ซึ่งสามารถทำให้วินโดวส์โปรแกรมทำงานซ้อนกันได้โดยการแบ่งจอแสดงผลเป็นหน้าต่างที่ต่างกัน ผู้ใช้สามารถป้อนข้อมูลเข้าโปรแกรมที่ต้องการได้โดยตรงผ่านระบบการทำงานหลายอย่างพร้อมกัน โดยใช้แป้นพิมพ์หรือเม้าส์เพื่อเลือกโปรแกรมหนึ่งจากโปรแกรมที่กำลังทำงานพร้อมกัน วินโดวส์จะรับข้อมูลเข้าของผู้ใช้และให้ทรัพยากรที่จำเป็น เช่น โปรเซสเซอร์ ตามที่โปรแกรมต้องการ

การติดต่อที่มองเห็น (Visual Interface)

วินโดวส์โปรแกรมมีคุณสมบัติและการทำงานเหมือนกันคือ ขอบ(borders),คอนโทรลบ๊อกซ์(control boxes),อะเบาท์บอกซ์(About boxse) และอื่นๆ ซึ่งทำให้การเรียนรู้วิธีใช้โปรแกรมจากโปรแกรมหนึ่งไปยังอีกโปรแกรมหนึ่งสามารถทำได้ง่าย

ขอบ (Border)

หน้าต่างของวินโดวส์ทั้งหมดจะมีขอบรอบๆ ขอบประกอบด้วยเส้นที่เป็นกรอบหน้าต่างขอบแสดงให้รู้ว่าหน้าต่างไหนกำลังทำงานอยู่ เมื่อมีหลายหน้าต่างซ้อนๆกัน ถ้าใช้เม้าส์ชี้ไปที่ขอบแล้วกดปุ่มซ้าย ผู้ใช้สามารถเปลี่ยนขนาดของหน้าต่างได้

แถบแสดงชื่อโปรแกรม (Title bar)

แถบแสดงชื่อโปรแกรมจะแสดงชื่อของโปรแกรมที่ด้านบนของหน้าต่าง โดยจะอยู่ทางด้านบนตรงกลางของหน้าต่าง มีประโยชน์ในการทำให้รู้ว่าโปรแกรมไหนกำลังทำงานอยู่

คอนโทรลบ๊อกซ์ (Control Box)

วินโดวส์โปรแกรมใช้คอนโทรลบ๊อกซ์ ซึ่งเป็นรูปกล่องสี่เหลี่ยมเล็กๆตรงมุมบนซ้าย มีขีดอยู่หนึ่งขีดภายใน คลิ๊กเม้าส์บนคอนโทรลบ๊อกซ์จะทำให้หน้าต่างแสดงรายการคำสั่งของระบบ

รายการคำสั่งของระบบ (System Menu)

รายการคำสั่งของระบบจะทำงานเมื่อใช้เม้าส์คลิ๊กบนคอนโทรลบ๊อกซ์ และจะเตรียมคำสั่งพื้นฐานที่จัดการเกี่ยวกับหน้าต่าง เช่น

- คำสั่งกลับสู่ขนาดเดิม(Restore)
- คำสั่งย้ายหน้าต่าง(Move)

- คำสั่งเปลี่ยนขนาดหน้าต่าง(Size)

  คำสั่งลดหน้าต่างเป็นภาพสัญญลักษณ์ที่เรียกว่า ไอคอน (Minimize)

- คำสั่งขยายขนาดหน้าต่างให้ใหญ่เต็มจอแสดงผล(Maximize)

- คำสั่งปิดหน้าต่าง(Close) เป็นต้น

ช่องลดขนาดหน้าต่างและช่องขยายขนาดหน้าต่าง (Minimize Box,Maximize Box)

มุมบนขวาของวินโดวส์โปรแกรมจะแสดงสัญญลักษณ์ช่องสี่เหลี่ยมที่มีรูปลูกศรเล็กๆ 2 ช่อง ช่องหนึ่งจะมีรูปลูกศรชี้ลงเรียกว่าช่องลดขนาดข้อมูล(Minimize Box) เมื่อใช้เม้าส์คลิ๊กที่ช่องนี้จะทำให้หน้าต่างลดขนาดลงเป็นภาพสัญญลักษณ์ที่เรียกว่าไอคอน(icon) อีกช่องจะมีรูปลูกศรชี้ขึ้นเรียกว่าช่องขยายขนาดหน้าต่าง(Maximize Box) เมื่อใช้เม้าส์คลิ๊กที่ช่องนี้จะทำให้ขนาดของหน้าต่างขยายจนเต็มจอแสดงผล ซึ่งจะทำให้หน้าต่างนี้ซ้อนทับหน้าต่างอื่นๆที่กำลังทำงานอยู่ในขณะนั้น

แถบเลื่อนแนวตั้งและแนวนอน (Vertical Scroll Bar,Horizontal Scroll Bar)

แถบเลื่อนแนวตั้ง(Vertical Scroll Bar) จะอยู่ทางด้านขวาใต้ช่องขยายขนาดหน้าต่างลงมา ส่วนแถบเลื่อนแนวนอน(Horizontal Scroll Bar)จะอยู่ที่ด้านล่างของแต่ละหน้าต่าง แถบเลื่อนจะมีรูปลูกศรชี้ตรงข้ามกันที่ปลายทั้งสองข้าง และจะมีช่องหน้าต่างใสกับแถบสี ซึ่งช่องหน้าต่างใสจะแสดงข้อมูลที่แสดงผลในปัจจุบันเทียบกับข้อมูลทั้งหมดซึ่งแสดงด้วยแถบสี การใช้เม้าส์คลิ๊กที่รูปลูกศรแต่ละอันจะแสดงข้อมูลหนึ่งบรรทัดสำหรับแนวตั้งหรือหนึ่งคอลัมน์สำหรับแนวนอน การใช้เม้าส์คลิ๊กบนช่องหน้าต่างใสและลากจะทำให้จอภาพแสดงผลข้อมูลเปลี่ยนไปยังส่วนต่างๆของข้อมูลแสดงผลของโปรแกรม แถบเลื่อนแนวตั้งจะช่วยให้การแสดงผลของข้อมูลเอกสารที่มีหลายหน้าทำได้รวดเร็ว แถบเลื่อนแนวนอนจะช่วยให้การแสดงผลของข้อมูลที่มีหลายคอลัมน์ที่มีขนาดเกินกว่าจอภาพจะแสดงได้ทำได้รวดเร็ว

แถบแสดงรายการคำสั่ง (Menu Bar)

ทุกวินโดวส์โปรแกรมจะมีแถบแสดงรายการคำสั่งอยู่ใต้แถบแสดงชื่อโปรแกรม (Title bar) แถบแสดงรายการคำสั่งใช้สำหรับเลือกรายการคำสั่งหรือรายการคำสั่งย่อย การเลือกจะทำได้โดยการชี้เม้าส์และคลิ๊กบนรายการคำสั่ง หรือโดยการใช้คีย์ด่วน(hot key) โดยการกดแป้น ALT และตัวอักษรตัวหนาที่ขีดเส้นใต้ในคำสั่ง เช่น "F" ในคำสั่ง File แถบรายการคำสั่งจะมีคำสั่งขอความช่วยเหลืออยู่ด้วย (Help)

ส่วนแสดงผลของโปรแกรม (Client Area)

เป็นส่วนที่ใหญ่ที่สุดของแต่ละหน้าต่าง ส่วนแสดงผลของโปรแกรมเป็นส่วนแสดงผลส่วนแรกของโปรแกรม การจัดการส่วนแสดงผลของโปรแกรมเป็นหน้าที่ของโปรแกรมประยุกต์ จะ

มีแต่โปรแกรมประยุกต์เท่านั้นที่สามารถส่งข้อมูลออกไปที่ส่วนแสดงผลของโปรแกรมได้

บทที่ 2 แนะนำบิทแมพกราฟฟิกส์สำหรับวินโดวส์

(Introducing bitmapped graphics for Windows)

ถึงแม้มีความเข้าใจในด้านกราฟฟิกส์ของมันดี ในตัววินโดวส์เองก็ยังให้เครื่องมือที่น่าประหลาดใจในช่วงที่จำกัด สำหรับทำงานกับ บิทแมพกราฟฟิกส์ มันสามารถวาดแต่ไม่ใช่การทาสีจริง ๆ บิทแมพกราฟฟิกส์เป็นทางที่คอมพิวเตอร์เกี่ยวข้องกับการมองเห็นจริง

รูป 2-1 เป็นบิทแมพกราฟฟิกส์ ที่ถูกแสดงโดยวินโดวส์แอพพลิเคชัน มันแสดงสิ่งที่สำคัญหลายอย่างเกี่ยวกับวินโดวส์ สิ่งที่สำคัญที่สุดคือวินโดวส์มีความน่าสนใจในด้านการมองเห็น และมันยังแสดงถึงผู้ใช้สำหรับปุ่มสีเทาเล็ก ๆ (grey button)

วินโดวส์ซอฟแวร์ ไม่ได้บอกมากนักในการรวม Krazy Kat หรือรายละเอียดในการระบายสี ในขณะที่วินโดวส์สามารถทำได้ ทางเดินระหว่างไฟล์ของภาพกับซอฟแวร์ไม่ได้ถูกบอกไว้ให้ดี ซึ่งเป็นสิ่งที่จะกล่าวต่อ ๆ ไป

รูป 2-1 บิทแมพกราฟฟิกส์ในวินโดวส์-Ignatz,Krazy Kat,และ ปุ่มควบคุม

กราฟฟิกส์ส่วนมากที่เกี่ยวข้องกับวินโดวส์ เป็นเวคเตอร์กราฟฟิกส์ (vector graphics) ค่าของมันจะเกี่ยวกับการวาดเส้นและวัตถุที่ถูกเติมเต็ม (filled object) วินโดวส์ไอคอน (icons) ที่ผู้ใช้รวบควมไว้ เป็นภาพบิทแมพที่มีขนาดเล็กมาก ไอคอนเป็นรูปแบบของไฟล์แบบง่าย ๆ วินโดวส์สนับสนุนรูปแบบบิทแมพของตัวเองคือ BMP ฟอร์แมต ที่ใช้ใน Wallpaper ของวินโดวส์ แต่มีหลายเหตุผลที่มันไม่เหมาะสำหรับการเก็บภาพใหญ่ ๆ

นอกเหนือจากการสนับสนุนภาพบิทแมพ วินโดวส์ไม่ได้ให้เราจัดการกับภาพได้ง่ายนัก โดยเฉพาะอย่างยิ่งมันไม่ได้ให้บิทแมพรูปแบบต่าง ๆ ที่ใช้งานจริงเป็นส่วนของรีลอร์ท (resource)

หัวข้ออีกหัวข้อหนึ่งเป็นเนื้อหาในการจัดการความซับซ้อนของภาพขนาดใหญ่ ภาพใด้วินโดวส์ จริง ๆ แล้ววินโดวส์มีทางเลือกของระบบพอสมควรในการเลือกการแสดงผล, การจัดการ, การพิมพ์ และการจัดการบิทแมพ แต่การใช้งานไม่ชัดเจนนัก และข้อจำกัดยังไม่ถูกจัดการได้ดี

ถ้ามีทางเลือกระหว่างภาพ 2-2 กับ Exel spreadsheet ที่ทำงานอยู่ คนส่วนใหญ่จะเลือกภาพ 2-2 แต่ถ้ามีใครไม่ชอบภาพที่แสดงโดยแอพพลิเคชัน แน่นอน เขาจะสามารถแทนได้ด้วยภาพของเขาเอง เป็นสิ่งสำคัญมากของบิทแมพกราฟฟิกส์ที่มีภาพอยู่ภายใน

รูป 2-2 แอพพลิเคชันที่แสดงภาพ GIF

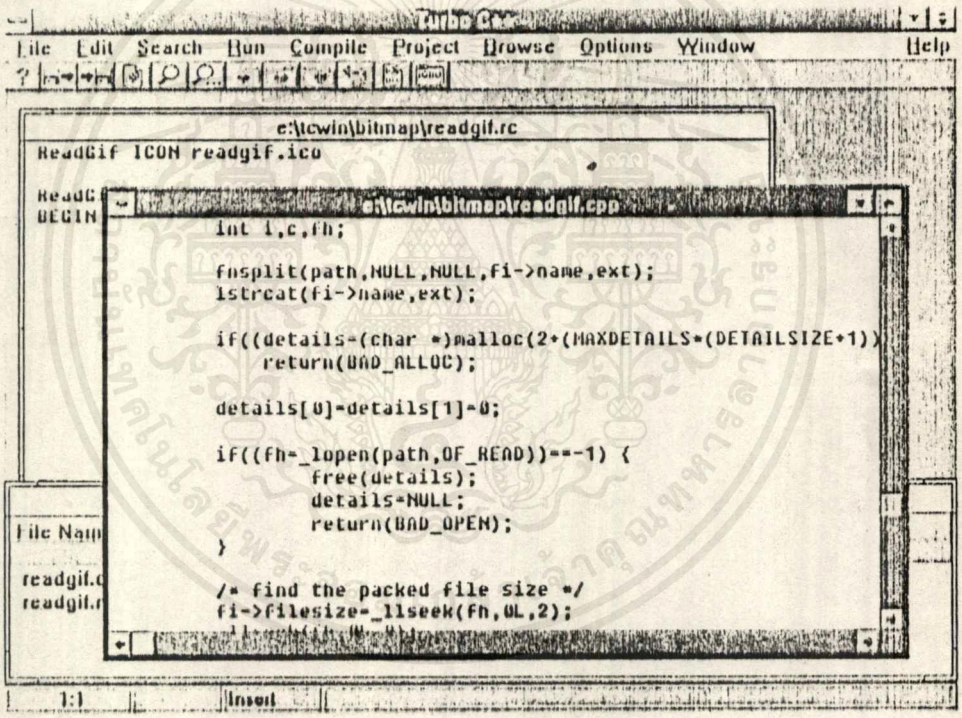
2.1 เครื่องมือที่ใช้ในการพัฒนางาน (Development tools)

สิ่งหนึ่งที่ผู้ผลิตตัวแปล (compiler) ภาษา C ชอบที่จะอ้างว่าเหตุผลในการเขียนซอฟแวร์ คือการเริ่มต้นที่ลึกลับ และส่วนใหญ่จะประกอบไปด้วยเครื่องหมายวรรคตอน ผู้ผลิตตัวแปลภาษา C จึงสร้างตัวแปลภาษา C บน DOS ให้ครอบคลุมทั้งหมด จึงไม่มีตัวแปลภาษา C คู่ใด compile โปรแกรมที่ยาวกว่า 8 บรรทัดในลักษณะเดียวกัน

โปรดีที่ภาษา C บนวินโดวส์ไม่มีความเป็นศัตรูกันของ 2 ฝ่าย ส่วนมากโค้ดที่พัฒนาโดยคอมไพเลอร์ตัวหนึ่ง สามารถคอมไพล์โดยตัวแปลภาษาตัวอื่นได้

โปรแกรมของโครงงานนี้เขียนโดยคอมไพเลอร์ของ Borland คือ Turbo C++ for Windows ซึ่งเป็นคอมไพเลอร์ภาษา C ที่สามารถใช้ edit, compile และ run application โดยไม่ต้องเปลี่ยนไปมาระหว่างดอสกับวินโดวส์ ข้อเสียที่สำคัญของ Turbo C++ for Windows คือจะมี bug ซึ่งมีผลทำให้เกิดแอพพลิเคชันผิดพลาด (unrecovable application error) หรือ ความผิดพลาดในโหมดป้องกัน (protected mode fault) ซึ่งขึ้นอยู่กับเวอร์ชันของวินโดวส์ด้วย

โค้ดที่สร้างโดย Turbo C++ for Windows ไม่มีปัญหาเหมือนตัวคอมไพเลอร์ ซึ่งสามารถสร้างแอพพลิเคชันที่คงทนและใช้งานได้ดี รูป 2-3 แสดงถึง Turbo C++ for Windows ในขณะที่เกิด unrecovable application error



รูป 2-3 ตัวอย่างหน้าจอของบอร์แลนด์ C++ สำหรับวินโดวส์

เมื่อที่จะเริ่มงานนี้ เราจะทำการคอมไพล์โปรแกรมโดยใช้ small memory model หรืออาจใช้ medium memory model ถ้าโปรแกรมมีขนาดใหญ่ขึ้น มีผลไม่ดีเล็กน้อยเมื่อใช้ medium memory model กับงานที่ใช้ small memory model ในวินโดวส์ หรืออาจทำการเริ่ม medium memory model ถ้าไม่ลนใจจะเปลี่ยน model ภายหลัง

small memory model ต้องการให้ทั้งโค้ดและข้อมูล (data) อยู่ในเซกเมนต์ (segment) เดียวที่มีขนาด 64 K ซึ่งไม่เป็นข้อจำกัดมากนักเพราะวินโดวส์สามารถละบางส่วนไว้ในแอพพลิเคชันแบบ small model เช่น รีสอร์ท ไว้บนดิสค์ จนกว่าจะต้องการใช้งาน การอ้างแอดเดรสทุกแบบของ small model program ทำโดยใช้พอยท์เตอร์ขนาด 16 บิท

medium memory model อนุญาตให้มีหลาย ๆ โค้ดเซกเมนต์ (multiple code segment) ไม่มีโค้ดเซกเมนต์ที่มีขนาดใหญ่กว่า 64 K วินโดวส์จะโหลดเซกเมนต์ที่ต้องการขณะทำงาน

แอพพลิเคชันที่ทำงานกับบิทแมพขนาดใหญ่ ต้องการหน่วยความจำและเกี่ยวข้องกับโปรเซสเซอร์มาก ถึงแม้ medium memory model จะลดความสามารถของแอพพลิเคชันลงแต่ก็จะดีกว่า ถ้าเปรียบเทียบกับจำนวนหน่วยความจำและ overhead ของโปรเซสเซอร์ที่ทำงานขยายภาพขนาดกลางของไฟล์แบบ GIF

สิ่งที่สองที่จะทำการตรวจสอบเกี่ยวกับคอมไพเลอร์ภาษา C ที่จะทำงานกับโค้ดในโครงงานนี้คือ callback function เอนาเบิ้ล smart callback option ในคอมไพเลอร์ callback function เป็นบิทของโปรแกรมที่เราบอกให้วินโดวส์เรียกใช้ เช่น ถ้าเราสร้างไดอะลอกซ์ เราผ่านแอดเดรสที่เหมาะสมของ callback ซึ่งวินโดวส์เรียกอย่างไม่เป็นทางการว่า procinst ถ้าโปรแกรมทำงานกับคอมไพเลอร์ที่ไม่สนับสนุน smart callback ให้ทำการประกาศฟังก์ชันให้ชัดเจน

สิ่งที่เพิ่มเติมเกี่ยวกับ Microsoft-compatible Windows C compiler คือต้องมีการคอมไพล์รีสอร์ท โครงงานนี้ใช้ Resource workshop ที่มากับ Turbo C++ for Windows หรืออาจใช้เครื่องมือจัดการรีสอร์ทตัวอื่น ๆ ซึ่งจะเหมือนกับคอมไพเลอร์ภาษา C คือ resource script จะมีความ portable ระหว่างสภาวะแวดล้อมในการพัฒนาโปรแกรมบนวินโดวส์ที่ต่างกัน

2.2 วินโดวส์และสี (Windows and color)

มีหลายสิ่งที่ควรเข้าใจเกี่ยวกับวินโดวส์และการที่มันทำงานกับภาพบิทแมพ วินโดวส์และความเข้าใจที่แปลกของวินโดวส์ อาจถูกเรียกให้ทำงานโดยไม่กี่เวิร์ด (word) ภายใน

วินโดวส์สามารถรับช่วงการใช้มอนิเตอร์ได้อย่างน่าประทับใจ เพราะมันใช้ไดร์เวอร์ที่สามารถโหลดขึ้นมาเชื่อมต่อระหว่างเครื่องวาดภาพของตัวเอง (เรียกว่า

GDI Graphic device interface) และการ์ดแสดงผล ดังนั้นวินโดวส์จึงสามารถ
วันได้ตั้งแต่เต็มหน้าจอของ desktop publishing แบบโมโนโครม จนถึงการ์ดแสดง
ผลแบบ 24 บิท ซึ่งให้สีได้เหมือนจริง โดยสภาวะทั่วไปจะรันบนโหมด 16 สี ขนาด
640 จุด คูณ 480 จุด ซึ่งคือ VGA ซึ่งเป็นโหมดกราฟฟิกส์เชิงการค้า

โหมด VGA 16 สี อนุญาติให้แสดงสีที่แตกต่างกัน 16 สีในเวลาเดียวกัน
ซึ่งมาจากจานสี (palette) ประมาณ 256,000 สี

ปัญหาที่อยู่ภายในจานสีที่จำกัด เช่นในสภาวะแวดล้อมแบบมัลติทาสก์กิง
(multitasking environment) มองเห็นได้ง่าย ถ้าทุกแอพพลิเคชันมีอิสระใน
การตั้งค่าจานสีเอง วินโดวส์จะทำงานไม่เหมาะสม เช่น สมมุติว่าแอพพลิเคชันที่อยู่
ด้านหน้าตัดสินใจที่จะตั้งจานสีเป็นสีขาวหมดโดยเว้นไว้ 1 จานสี ต่อมาวินโดวส์เกิดมี
ไดอะลอกซ์แจ้งว่า print maneger มีปัญหา ไดอะลอกบ๊อกซ์ก็จะเป็นสีขาวทั้งหมด
ทำให้เรามองเห็นปัญหาที่เกิด

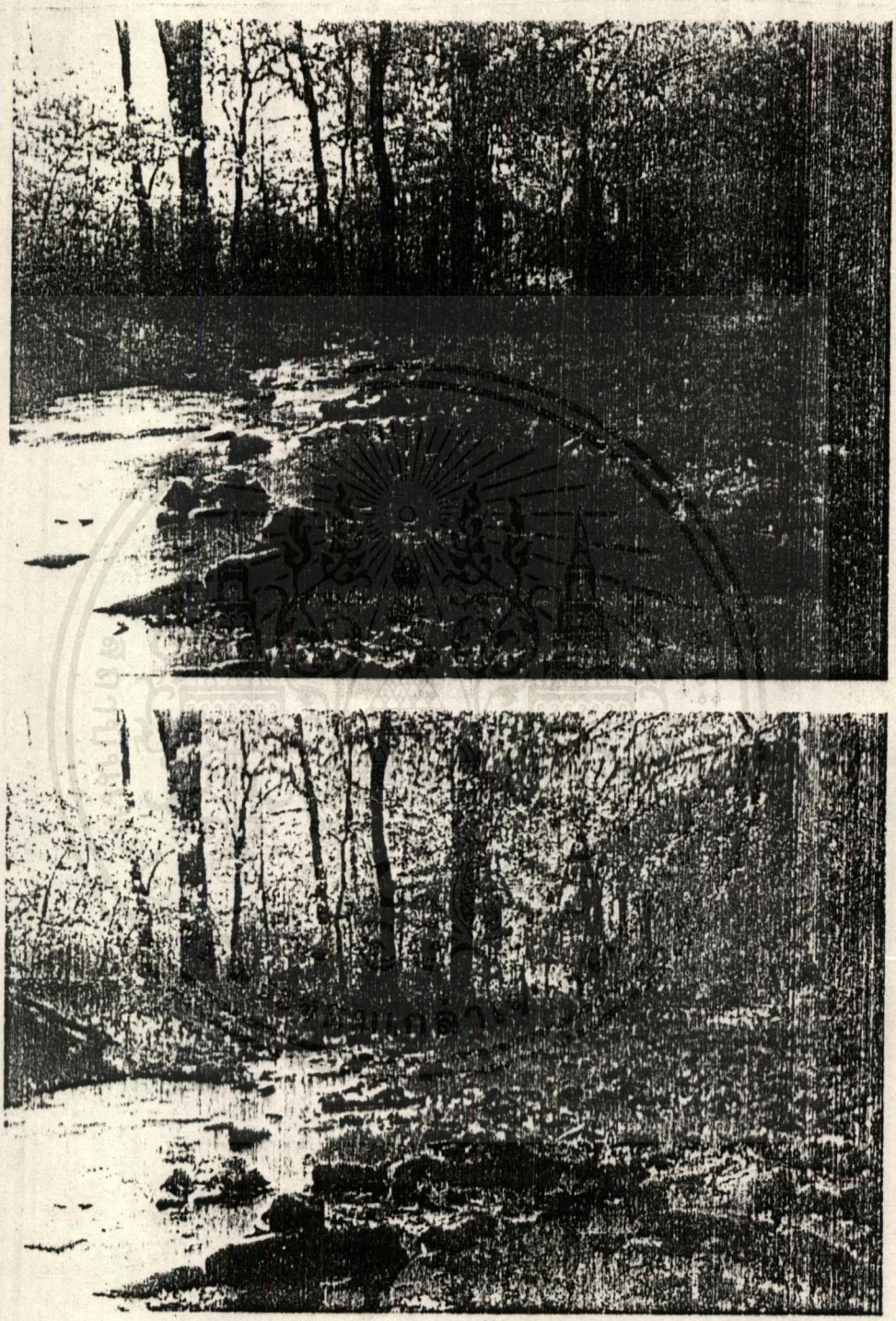
วินโดวส์จัดการปัญหานี้โดยไม่อนุญาติให้ทำการเปลี่ยนจานสีของฮาร์ดแวร์
โดยจะมีจานสีที่สงวนไว้ 20 จานสี ในการแสดงผล 16 สี เพียงแค่ 16 สีแรกที่มีอยู่จริง
เมื่อแอพพลิเคชันร้องขอที่จะระบายบางสิ่งด้วยสีเขียว วินโดวส์จะหาสีในจานสีที่สงวนไว้
ที่เหมือนกับสีเขียวมากสุดและนำมาใช้

ถ้าเราให้วินโดวส์แสดงภาพบิทแมพแสดงสีมากกว่าที่มีอยู่ หรือที่มีสีไม่ใกล้เคียง
กับจานสีสงวนของวินโดวส์ มันจะทำการ remap ภาพเพื่อหาจานสีที่เหมาะสม ผลที่ได้
บางครั้งไม่ดีนัก รูป 2-4 แสดงบิทแมพ 256 สี บนไดร์เวอร์ 16 สี ของวินโดวส์

กล่าวให้ชัดเจนขึ้นก็คือ ไม่มีวิธีที่จะทำให้การแสดงผล 16 สีแสดงสีมากกว่า
16 สี ไม่เป็นการดีที่จะเปลี่ยนแปลงจานสีที่วินโดวส์มีอยู่ มีทางที่จะหลอกลวงเรียกว่า
การทำดิชเชอร์ (Dithering) ถึงแม้เป็นการลวงที่ไม่สมบูรณ์แบบแต่ก็เป็นวิธีจัดการที่
ใช้งานได้จริง ดูแนวคิดพื้นฐานของดิชเชอร์ที่ใช้ให้การแสดงภาพสีเต็มให้เป็น 16 สี

คิดง่าย ๆ ว่า ดิชเชอร์จะหลอกสายตาเราเพื่อให้เห็นสีที่ไม่มีอยู่โดยการ
ผสมสี ดังเช่น ถ้าเราต้องการจะทาสีเขียว แต่มีสีน้ำเงินและสีเหลือง เราอาจจะผสม
สี 2 กระป๋องนี้ เปิดผาเสปรย์แล้วนำไปใช้งาน คอมพิวเตอร์สามารถทำสิ่งที่คล้ายกัน
นี้โดยการ ดิชเชอร์ มันทำการระบายสีทั้งหมดเป็นสีน้ำเงินและระบายหลาย ๆ จุดเป็น
สีเหลือง ห่างออกไปสักหน่อยผลของมันดูเป็นสีเขียว

ภาพ 2-5 แสดงบิทแมพจากภาพ 2-4 หลังจากถูกดิชเชอร์ในการแสดงผล
ทำให้ภาพดูมีความสำคัญกว่าในเรื่องของสี



รูป 2-4 เมื่อวินโดวส์แสดงภาพ 256 สีบน driver 16 สี

รูป 2-5 การดิชเชอร์บิทแมพจากรูป 2-4

การ์ด Super VGA ส่วนใหญ่สามารถสนับสนุนการทำงานของวินโดวส์ใน ไดร์เวอร์แบบ 256 สี ถึงแม้จะถูกวินโดวส์สงวนสีเอาไว้ 20 สี แต่ก็ยังมีสีเหลืออยู่ถึง 236 สี แอพพลิเคชันใหม่ ๆ ใช้ประโยชน์ส่วนนี้เพื่อจะตั้งค่าสีในทางที่ต้องการ ดังนี้ จึงทำให้เราสามารถแสดงภาพถ่ายที่จำเป็นได้ในวินโดวส์ ดังในกรณีของภาพ 2-2

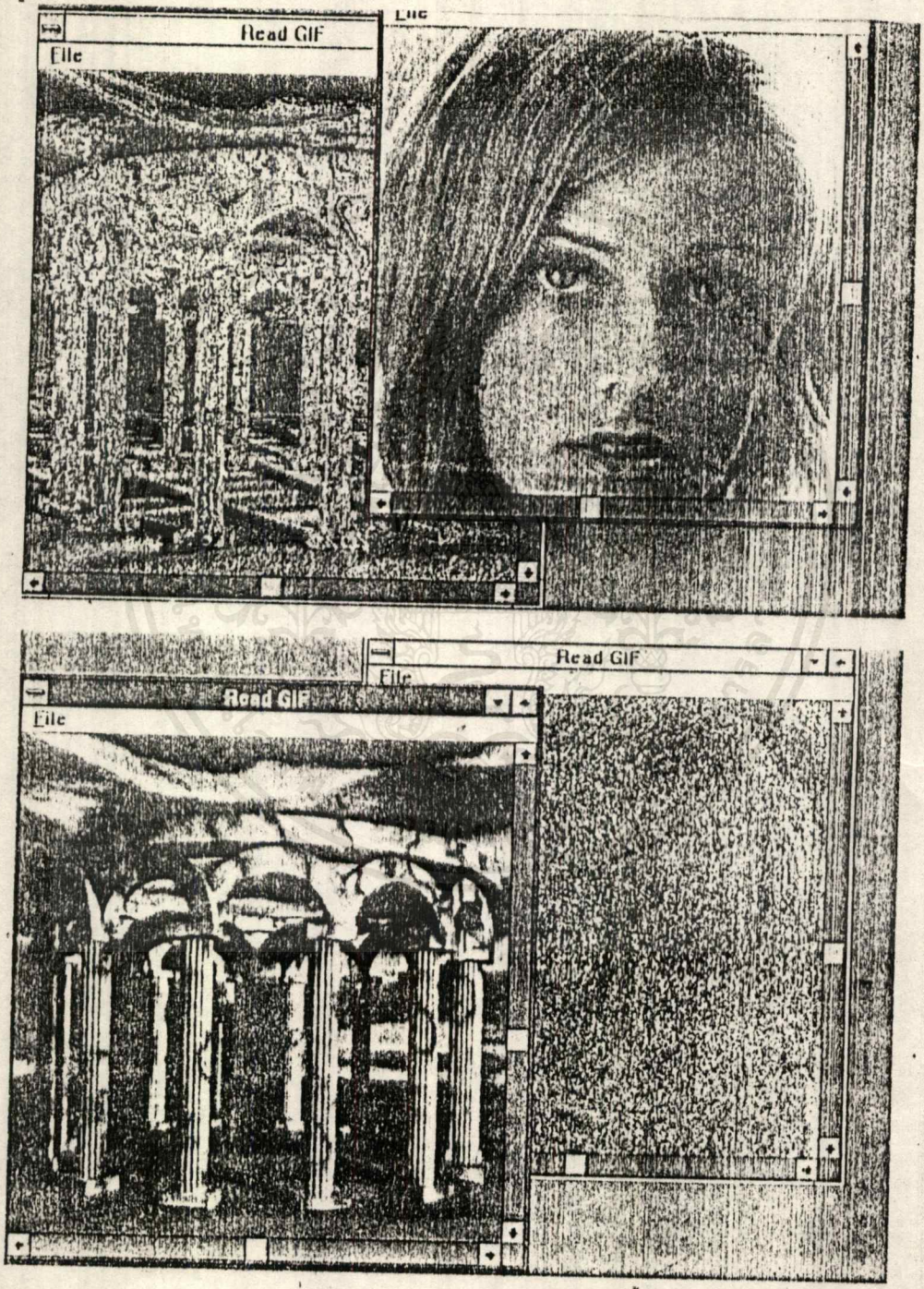
จึงเป็นโอกาสดีที่จะเห็นสิ่งที่เกิดขึ้นเมื่อแอพพลิเคชัน 2 ตัวประชันกันใน จานสีที่จำกัด รูป 2-6 แสดงภาพ GIF พร้อมกัน 2 ภาพของแอพพลิเคชัน ซึ่งเป็น ภาพที่ต่างกัน ในกรณีนี้ตัวที่แอคทีฟอยู่จะแสดงภาพที่ถูกต้องส่วนตัวที่ไม่แอคทีฟถูกบังคับให้ ใช้ค่าจานสีที่ผิดพลาด

อาจจะไม่มีค่าอะไรเลยสำหรับผลเสียที่เกี่ยวกับการใช้ไดร์เวอร์ของวินโดวส์ แบบ 256 สี ไดร์เวอร์นี้ต้องการหน่วยความจำมากเพียงสำหรับการดูแล screen มัน ทำให้การอัพเดทสกรีนช้าลง ซึ่งเป็นสิ่งที่ควรเข้าใจเมื่อเราใช้คอมพิวเตอร์ที่ช้าอยู่แล้ว โปรแกรมในโครงงานนี้สามารถรันบนการแสดงผลทั่วไปได้ ทั้งการ์ดแสดงผลแบบใช้ ไดร์เวอร์ 16 สีและ 256 สี ส่วนโหมดการแสดงผลก่อนนี้ จะต้องไปแก้ไขโปรแกรมก่อน

มันไม่มีผลพิเศษใด ๆ เมื่อเราแสดงบิทแมพ 256 สีบนการ์ดแสดงผล 256 สี วินโดวส์จะทำการลวงจานสีของมันเองสำหรับอนุชาติการแสดงผลของ 20 สีที่สงวนไว้

ผลที่เกิดคือ มันจะหา 20 สีสงวนในแหล่งของบิทแมพและทำการรีแมพ ในภาพส่วนใหญ่ที่มี 20 สีหายไปจาก 256 สีเนื่องจากสีสงวนไม่ทำให้สังเกตได้

ถ้าเรานำภาพบิทแมพ 256 สีขึ้นมาในโปรแกรม Paintbrush ของวินโดวส์ โดยใช้ไดร์เวอร์ 256 สีและทำการจัดเก็บไปยังไฟล์ใหม่ การจัดเรียงสีของชุดจานสีจะถูกเปลี่ยน นี่เป็นผลที่อาจไม่สังเกตเห็นได้ ยกเว้นในกรณีที่ร้ายแรงที่ทุกสีในแหล่งของภาพถูกจำกัดออกอย่างสำคัญจาก 20 สีสงวน

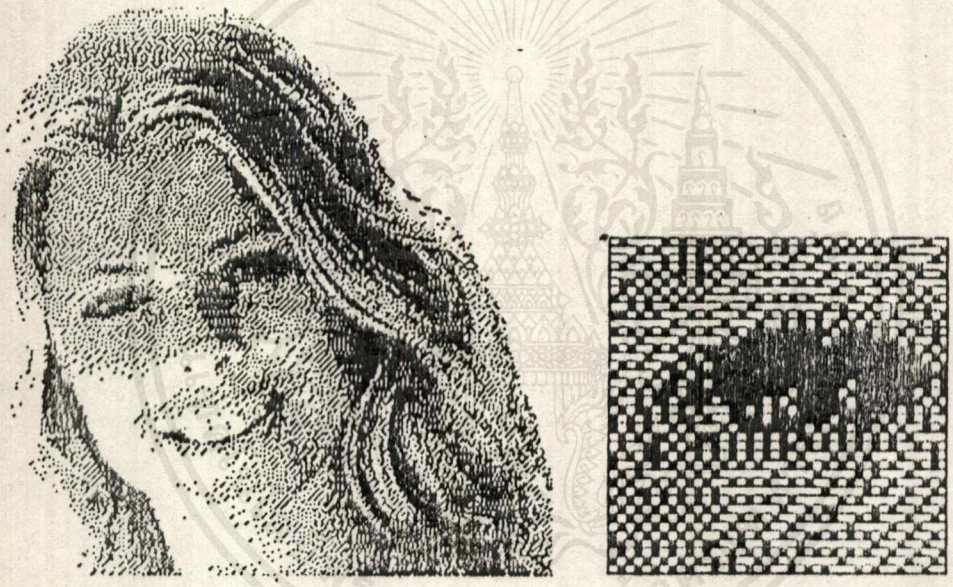


รูป 2-6 ผลของการที่มี 1 แอพพลิเคชันควบคุมจานสีทั้งหมด 256 สีบนวินโดวส์

2.3 ทำความเข้าใจบิทแมพ (Unterstanding Bitmaps)

ที่ผ่านมาได้บรรยายถึงบิทแมพในฐานะของวัตถุ (objects) โดยปราศจากการบรรยายว่าจริง ๆ แล้วบิทแมพคืออะไร นี่เป็นในด้านของวินโดวส์บนพื้นฐานที่ว่าวินโดวส์เข้าใจบิทแมพเป็นอย่างดีและจะบอกให้เรารู้ในสิ่งที่วินโดวส์ต้องการ โดยการให้การจัดการ (handle) แก่ผู้ใช้ที่จะจัดการกับงานภายใน

ชนิดที่ง่ายที่สุดของบิทแมพกราฟฟิกส์คือบิทแมพแบบโมโนโครม โดยแสดงตัวอย่างของภาพบิทแมพแบบโมโนโครมในภาพ 2-7



รูป 2-7 บิทแมพกราฟฟิกส์แบบโมโนโครม

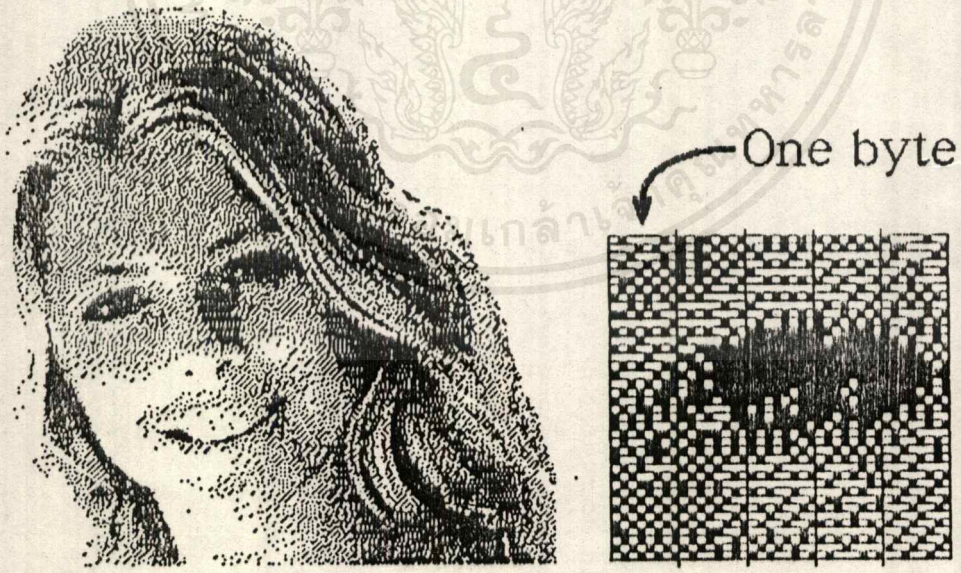
ในด้านขวาของภาพ 2-7 ภาพทั้งหมดสามารถมองเห็นเป็นการประกอบกันของหน่วยสี่เหลี่ยมเรียกว่า pixels ในบิทแมพแบบขาวดำแต่ละพิกเซลคือ เปิดและปิด (on and off)

ในความเป็นจริงบิทแมพเป็นเมตริกซ์ 2 มิติของพิกเซล อย่างไรก็ตามเนื่อง

จากบิทแมพส่วนใหญ่จะแสดงบน มอนิเตอร์ หรือ การพิมพ์ โดยทั่วไปสามารถบรรยายบิทแมพว่าประกอบไปด้วยชั้นของเส้น (scan line) ที่ถูกจับเข้ามาเป็นแถวของจุดในแนวตั้ง เมื่อเราแสดงบิทแมพบนมอนิเตอร์ซึ่งเป็นอุปกรณ์สำหรับบิทแมพ แต่ละบรรทัดของบิทแมพถูกจัดการโดยหนึ่งบรรทัดของมอนิเตอร์

ในสภาวะแวดล้อมของ PC บิทแมพไม่ใช่บิทแมพจริง ๆ แต่เป็น ไบท์แมพ (bytemaps) ภาพ 2-7 ก็เป็นภาพที่ถูกแสดงเหมือนภาพ 2-8

เป็นการสะดวกในการคิดว่าแต่ละจุดในบิทแมพแบบโมโนโครมเป็นชิ้นที่ไม่ต่อเนื่องกัน (discrete entity) ในความเป็นจริงแล้วมันเป็นการรวมกลุ่มกันเนื่องว่าแต่ละไบท์ในหน่วยความจำที่ใช้กับบิทแมพแบบโมโนโครมมีบิทแมพจริง ๆ 8 จุด เพราะมี 8 บิทใน 1 ไบท์ ในบิทแมพแบบสี แต่ละจุดมี 1 สีแทนที่จะเป็นการเปิดและปิด



รูป 2-8 การแสดงโครงสร้างภาพในของบิทแมพแบบโมโนโครม

โมโนโครมบิทแมพถูกแสดงได้เพียงแบบเดียว เรียกว่าเป็น single-image plane พื้นผิวสมมุติของไบท์เรียงกันราวกับเป็นแต่ละจุดของเมตริกซ์ บิทแมพที่แสดงสีสามารถแสดงได้หลายวิธี วิธีการจัดการกับสีสะท้อนให้เห็นถึงฮาร์ดแวร์ที่ใช้แสดงสี ที่ทำงานกับบิทแมพ

ทางที่ตรงที่สุดในการแสดงสีถูกกำหนดโดยแต่ละจุดใช้สิ่งที่เรียกว่าค่า RGB แต่ละจุดถูกกำหนดโดยไบท์ที่มี 8 บิทสำหรับข้อมูลของสีทั้งหมดขนาด 24 บิทต่อ 1 จุด ซึ่งโดยทั่วไปมักเรียกว่า สีแบบ 24 บิท ในโมเดลของสีชนิดนี้แต่ละจุดสามารถแสดงสีได้ประมาณ 16 ล้านสีที่ต่างกัน

ข้อเสียของการแสดงสีขนาด 24 บิทนี้ทำให้ไฟล์ภาพมีขนาดใหญ่ ภาพขนาด 640 จุด คูณ 480 จุด ใช้การแสดงสีแบบ 24 บิท ต้องการหน่วยความจำไว้เก็บภาพ 900 K

เมื่อเร็ว ๆ นี้ ฮาร์ดแวร์แสดงผลที่แสดงภาพสีชนิดนี้ได้โดยตรงมีราคาเรียกได้ว่าไม่แพงนัก เร็ว ๆ นี้การ์ด Super VGA ส่วนใหญ่สนับสนุนสิ่งที่เรียกว่า high color (16 bit color) และสามารถแสดงภาพแบบ 24 บิท ส่วนใหญ่การ์ดเหล่านี้มีไดรเวอร์ของวินโดวส์ที่จะทำงานในโหมดสีสูง ๆ

การ์ดแสดงผลส่วนใหญ่ที่ถูกออกแบบให้ทำงานในสภาพแวดล้อมของ PC ใช้สิ่งที่เรียกว่า สีของจานสี (palette color) เพื่อที่จะเก็บรักษาความต้องการในการใช้หน่วยความจำเพื่อแสดงผลให้ลดลงเพื่อที่จะจัดการกับความใหญ่โตของภาพ สีของจานสีเป็นข้อจำกัดสำหรับสิ่งที่จะทำงานกับบิทแมพกราฟฟิกส์ มันยังเป็นการลงทุนที่ดีที่จะอนุญาตให้กราฟฟิกส์แบบสีสามารถถูกแสดงผลโดยใช้หน่วยความจำแสดงผลน้อยกว่าที่ควจะเป็นสำหรับค่า RGB

เริ่มด้วยการดูการ์ดแสดงผลแบบ 16 สีบนโหมด VGA จุดแต่ละจุดถูกแทนด้วยหมายเลขแทนที่จะเป็นค่า RGB ตัวเลขเป็นตัวชี้ไปยังตารางของค่า RGB ซึ่งเป็นจานสีของการ์ดแสดงผล แต่ละตัวเลขมีค่าระหว่าง 0 ถึง 15 และจึงมี 16 ชุดของจานสี ดังนั้นถ้าจานสีแรกเป็นสีเขียว เราต้องใส่ค่า 0 ลงในตำแหน่งที่ต้องการให้แสดงสีเขียว

โหมด 16 สี จำกัดความสามารถจากจอแสดงผลไปยังส่วนที่ใช้วาดภาพ เช่น วินโดวส์และไปยังการดิชเชอร์ภาพสี กับการใช้โหมด 256 สีที่มีอยู่ ทำให้เราสามารถแสดงภาพถ่ายสีที่เชื่อถือได้

ในบิทแมพ 256 สี แต่ละจุดถูกแสดงโดย 1 ไบท์ซึ่งสามารถมีค่าได้ 256 ค่าที่แตกต่างกัน โดยทั่วไปจะอ้างถึงบิทแมพ 256 สีในฐานะที่มีสีขนาด 8 บิท (eight-bit

colour) เพราะแต่ละจุดถูกแสดงโดย 8 บิท บิทแมพที่มีสีระหว่าง 4 ถึง 128 สีมีข้อแม้ในการจัดเก็บได้หลายอย่าง ซึ่งดูได้ในแต่ละหัวข้อไฟล์ฟอร์แมต

เพื่อให้ได้ประโยชน์มากขึ้น มาดูปัญหาของการจัดเก็บบิทแมพ 16 สีหรือที่เรียกว่าภาพแบบ 4 บิท เราสามารถใช้กลยุทธในการจัดเก็บที่ใช้กับภาพ 8 บิทและทำให้ง่ายโดยการไม่สนใจ 4 บิทบน การทำเช่นนี้เป็นสิ่งสะดวก แต่มีความสูญเปล่าของหน่วยความจำจำนวนมาก ในความเป็นจริงแล้วเป็นวิธีเดียวกับที่ไฟล์ GIF ถูกจัดเก็บ

วินโดวส์จัดเก็บภาพ 16 สีโดยใช้ nibble stacking ซึ่ง nibble คือ 4 บิทหรือครึ่งไบท์ ซึ่งครึ่งหนึ่งของ nibble คือ crumb

ในวินโดวส์ บิทแมพขนาด 16 สี วิธีที่ข้อมูลของบิทแมพถูกเก็บคือ จุดแรกของบรรทัดถูกเก็บใน 4 บิทบนในไบท์แรก จุดที่สองถูกเก็บใน 4 บิทล่างของไบท์แรก จุดที่สามถูกเก็บใน 4 บิทบนของไบท์ที่สอง และต่อ ๆ ไปอีก แต่ละ 4 บิทแสดงค่าที่ชี้ไปยังจานสี 16 สีของวินโดวส์

เป็นการง่ายในการทำความเข้าใจและง่ายต่อการอ้างถึงวิธีการเก็บกับสีขนาด 4 บิท แต่มันมีข้อเสียหลักอยู่ 2 ข้อ ข้อแรกคือมันไม่ยืดหยุ่น ตัวอย่างคือมันต้องการให้ภาพขนาด 32 สี ถูกเก็บราวกับว่าเป็นภาพ 256 สี เพราะเนื่องจากไม่มีสิ่งที่เป็น nibble แบบ 5 บิท ข้อที่สองคือวิธีนี้ไม่ได้เป็นรูปแบบเดียวกับที่การ์ดแสดงผลของ PC จัดการกับหน่วยความจำแสดงผล บิทแมพใด ๆ ที่ถูกจัดเก็บในฟอร์แมตนี้ต้องถูกแปลงไปยังรูปแบบที่ใช้แสดงบิทแมพ

ในความเป็นจริงแล้วปัญหาในข้อหลังไม่ได้เป็นจริงเสียทีเดียว การ์ดแสดงผลที่ออกใหม่หลายรุ่นที่เป็น VGA มีส่วนของโหมดการแสดงผล 16 สีของวินโดวส์ ซึ่งหน่วยความจำของมันถูกจัดเรียงเป็น stack nibble ซึ่งไม่ว่ากรณีใด ๆ การจัดการแปลงเป็นหน้าที่ของวินโดวส์ไม่ใช่ของผู้ใช้หรือโปรแกรมเมอร์

วิธีที่การ์ดแสดงผลส่วนใหญ่จัดการกับกราฟฟิกส์แบบ 16 สีเป็นเรื่องน่าอัศจรรย์ โดยใช้การจัดการที่เป็นระนาบของการ์ดแสดงผล เหตุผลที่ต้องทำกับวิธีการทำงานจริง ๆ ของการ์ดเป็นสิ่งที่วินโดวส์ป้องกันไม่ให้ผู้ใช้เข้าไปยุ่งเกี่ยว สิ่งที่เราต้องการรู้ทั้งหมดเกี่ยวกับบิทแมพก็คือส่วนที่เป็นโครงสร้าง

ในการจัดโครงสร้างของระนาบบิทแมพ ภาพที่สมบูรณ์ถูกเก็บเป็นระนาบแบบโมโนโครมของภาพหลายระนาบ มีระนาบ 1 ระนาบสำหรับบิทแต่ละบิทของสี ดังเช่นแต่ละจุดในภาพที่ประกอบด้วยตัวเลขแบบ 4 บิท เลข 4 บิทถูกตั้งขึ้นโดยเป็นระนาบละ 1 บิท ดังนี้การทำงานกับบิทแรกในมุมบนซ้ายของภาพเป็นการนำบิทแรก

ของระนาบแรกและให้น้ำหนักเป็น 1 บิทแรกของระนาบที่ 2 ให้น้ำหนักเป็น 2 บิทแรกของระนาบที่ 3 ให้น้ำหนักเป็น 4 และอื่น ๆ

2.4 ไฟล์ภาพบิทแมพ (Bitmapped image file)

เราสามารถจับภาพจากการ์ด VGA มาตรฐานบนโหมด 256 สีลงในไฟล์ในแผ่นดิสค์แบบง่าย ๆ โดยใช้โค้ดภาษา C ดังนี้

```
FILE      *fp;
if((fp=fopen("PICTURE.BIN","wa")) != NULL) {
     fwrite(MK_FP(0xa000, 0x0000),1,6400,fp);
     fclose(fp);
```

โค้ดนี้ทำการก๊อปปี้ข้อมูลขนาด 64,000 ไบท์อย่างง่าย ๆ จากตำแหน่งของหน่วยความจำซึ่งเป็นที่เริ่มของ screen buffer ของโหมดการแสดงผล ในกรณีนี้คือ A000:0000H เราสามารถแสดงภาพนี้โดยแสดงบนการ์ด VGA ต่าง ๆ กันโดยให้การ์ดอยู่ในโหมดกราฟฟิกส์เดียวกันและก๊อปปี้ค่าของไฟล์นั้นกลับไปยังสกรีนบัฟเฟอร์

การสร้างไฟล์แบบนี้ยังไม่ได้พิจารณาถึงสภาพความเป็นจริง มันไม่ได้มองถึงความเป็นจริงที่ว่าภาพนี้มีสีของจานสีไฟล์ที่มีรูปแบบง่าย ๆ นี้ไม่สามารถจัดเก็บได้ มันจะไม่มองถึงความเป็นจริงที่ว่าภาพหลาย ๆ ภาพมีช่องว่างจำนวนมากอยู่ภายในภาพ ถ้าเรากลับไปมองภาพ 2-1 เราจะเห็นว่านอกจาก Ignatz, Krazy Kat, และ high-tech brick ส่วนมากของภาพเป็นสีขาว

ในความเป็นจริงรูปแบบของไฟล์บิทแมพเป็นโครงสร้างข้อมูลที่ซับซ้อน ในการเริ่มต้นกับรูปแบบเช่น GIF หรือ PCX ซึ่งสามารถเก็บค่าสีของจานสีของภาพ รูปแบบของไฟล์ภาพส่วนใหญ่มีวิธีในการบีบ (compress) ภาพบิทแมพที่เกี่ยวข้องกับข้อมูลที่ซ้ำซ้อน (redundant data) ซึ่งคือพื้นที่ของบิทแมพที่มีจุดจำเพาะ (identical pixels) จำนวนมาก ภาพ 2-1 ประกอบด้วยบิทแมพอย่างหยาบ ๆ ที่ใช้หน่วยความจำ 414,720 ไบท์ ซึ่งต้องการเพียง 5,356 ไบท์ของเนื้อที่ดิสค์ในการจัดเก็บไฟล์แบบ GIF

คิดแบบง่าย ๆ แล้ว ภาพบิทแมพที่มีการย่อนั้นทำงานโดยการแทนค่าที่ซ้ำซ้อนด้วยโทเคน (tokens) ดังตัวอย่างต่อไปนี้คือ แถวบนของภาพ 2-1 ซึ่งเป็นสีขาวทั้งหมดประกอบไปด้วยไบท์จำเพาะจำนวน 288 ไบท์ ในการย่อภาพนี้เราสามารถแทน

ส่วนนี้ด้วยชนิดของโค้ดที่ถูกจัดเรียงมาก่อนแล้วซึ่งเป็นผลบอกว่า "วางข้อมูล 288 ไบท์ที่มีสีขาวและทำมันอย่างรวดเร็ว"

มีบางบริเวณของภาพ 2-1 ที่ไม่ใช่ข้อมูลที่ซ้ำซ้อนกัน ซึ่งถูกจัดการโดยโค้ดที่ต่างกันที่บอกว่า "วางไบท์ถัดไปอย่างที่มันเป็น เพราะเราไม่สามารถบีบมันลงได้"

โค้ดชนิดแรกเรียกว่า run of byte ส่วนโค้ดชนิดที่สองเรียกว่า string field ในบิทแมพที่ซับซ้อนจะมีโค้ดทั้ง 2 ชนิดอยู่รวมกันซึ่งเรียกว่า run length encoding

runlength encoding มีอยู่ด้วยกันหลายชนิด เราสามารถออกแบบขั้นตอนของ run length encoding ที่มีประสิทธิภาพที่สุดสำหรับภาพเฉพาะบางชนิด ซึ่งการออกแบบขั้นตอนของ run length encoding นี้ก็สามารถดูได้ในรูปแบบของไฟล์ต่าง ๆ

ในทางปฏิบัติแล้ว run length encoding มีประสิทธิภาพน้อยลงถ้าภาพนั้น ๆ มีความซับซ้อนมากขึ้น โดยบ่อยครั้งภาพที่มีขนาด 256 สีที่ถูกแสกนเข้ามานั้นสร้าง run length encoding ที่มีขนาดใหญ่กว่าบิทแมพที่ไม่ได้ถูกย่อขนาดข้อมูล ปรากฏการณ์ที่เกิดขึ้นนี้เรียกว่า การย่อข้อมูลตามธรรมชาติ (native compression) ซึ่งเกิดขึ้นเพราะข้อมูลมีความซับซ้อนเพียงพอที่จะไม่ต้องทำการย่อขนาด อย่างไรก็ตามในการเพิ่มโค้ดที่มีฟิลด์ของ run length encoding จริง ๆ แล้วก็ทำให้ภาพใหญ่ขึ้นเพียงเล็กน้อย

ภาพ 256 สีที่ถูกแสกนและรูปแบบอื่นที่ซับซ้อน สามารถถูกย่อข้อมูลอย่างมีประสิทธิภาพโดยใช้สิ่งที่เรียกว่าเป็น การย่อข้อมูลโดยใช้สตริง (string table compression) ซึ่งเป็นพื้นฐานของรูปแบบไฟล์ GIF และแพคเกจ PKZIP

โดยทั่ว ๆ ไปแล้ว ไฟล์ที่มีสีขนาด 24 บิทมักจะเป็น native compression รูปแบบการจัดเก็บของภาพ 24 บิทจะถูกเก็บโดยไม่มีการย่อขนาด

บทที่ 3 การโปรแกรมกับสี

(COLOR PROGRAMMING)

Windows application programming interface (API) ได้เตรียม build in ฟังก์ชันสำหรับโปรแกรมทางด้านสีไว้อย่างมากมาย และเพื่อให้ใช้ประโยชน์ได้อย่างเต็มที่จึงต้องทำความเข้าใจก่อนว่า Windows จัดการสีอย่างไร

Windows ใช้สีอย่างไร

ปรกติถ้า run Windows ใน mode VGA จะมี 16 สีโดยสีต่าง ๆ เกิดจาก electron gun 3 ตัว ที่จะแสดงสี แดง,เขียว และน้ำเงิน ซึ่งแต่ละสีนั้นสามารถปรับความเข้มได้ตั้งแต่ 0 จนถึงเข้มเต็มที่ ดังนั้นจึงสามารถผสมสีได้มากมายนับไม่ถ้วน

The digital limitation

monitor นั้นเป็น analog device คือสามารถปรับ RGB gun ได้ไม่จำกัดตั้งแต่ไม่ติดเลยจนถึงติดเต็มที่ แต่คอมพิวเตอร์นั้นเป็น digital device ในการเก็บข้อมูลเกี่ยวกับสีที่แสดงบนจอนั้นจะอยู่ในส่วนของ memory ที่เรียกว่า display memory บน VGA graphics card ซึ่ง memory ส่วนนี้มีขนาดจำกัด นั่นหมายถึงว่า จำนวนสีที่สามารถแสดงได้ก็มีจำนวนจำกัดด้วย

4 bit/pixel

แต่ละ pixel บนจอจะใช้ 4 บิตของ display memory ในการเก็บ ฉะนั้นจึงสามารถแสดงสีที่แตกต่างกันได้ทั้งสิ้น 2*2*2*2 = 16 สี VGA graphics card จึงสามารถแสดงสีได้ทั้งหมด 16 สี ใน VGA mode 640*480 ใช้ 307,200 pixels จึงต้องการ memory 1,228,800 บิต หรือ 153,600 ไบท์ ซึ่ง display memory ขนาด 256 K ใน VGA card จึงเพียงพอที่จะแสดง mode 640*480 16 สี

The digital-analog pipeline

VGA graphics adapter จะเป็นตัวเชื่อมโยงไปยัง display monitor

ซึ่งเป็น analog โดย adapter นี้จะต้องสามารถบอก RGB gun แต่ละตัวได้ว่าต้องใช้ความเข้มขนาดเท่าไรเพื่อแสดงแต่ละสี adapter จะใช้วิธีเปรียบเทียบค่าดังกล่าวจากตาราง application และ Windows 'GDI routine จะบอกไปยัง graphics card โดย software จากนั้น graphics card จึงจะไปบอก display monitor อีกทีว่า RGB แต่ละตัวควรมีความเข้มเท่าไร เป็นการแปลงจาก digital ไปเป็น analog

The lookup table

สีทั้ง 16 สี (Entry) จะเป็นตัวระบุเพื่อแปลงเป็นขนาดความเข้มของ RGB gun ซึ่งบางทีจะเรียกว่า index หรือ color index ในแต่ละ Entry ในตารางการค้นหานั้นจะใช้เลข 6 บิท สำหรับที่จะเก็บค่าของโวลเตจที่ใช้สำหรับที่จะให้ความเข้มที่ต่างกันของปืนยิงอิเลคตรอน RGB แต่ละบิทใน 6 บิทนั้นสามารถที่จะเปิดหรือปิดก็ได้ซึ่งทำให้เรามีสถานะอยู่ 2 สถานะในแต่ละบิท ดังนั้นจำนวนรวมทั้งหมดเท่าที่เป็นไปได้จะเก็บลงใน 6 บิท นี้คือ 64 (2*2*2*2*2*2 = 64) ค่าที่ได้นั้นจะเป็นของแต่ละปืนที่ใช้มีพิสัยอยู่ในช่วง 0 ถึง 63 หรือมี 64 ค่า เมื่อนำค่าของปืนยิงอิเลคตรอน 3 ตัวมารวมกัน จะสร้างสีที่แตกต่างกันได้ 262,144 สี นั่นคือ จอ VGA สามารถที่จะแสดงสีได้ตั้งแต่ 16 จนถึง 262,144 สี ซึ่งสี 16 สีที่เป็นสีที่เราอ้างถึงโดยทั่วไปเราจะเรียกว่า Hardware palette

The VGA system palette

เมื่อเรามีการเรียกใช้ Windows นั้นจะมีการตั้งค่าในตอนแรก 16 entry ใน Hardware Lookup table พวกนี้จะเป็นค่าเริ่มต้นของสีที่ใช้ในระบบ เราเรียกว่า System palette เราสามารถเขียนโปรแกรม Graphics บน DOS เพื่อที่จะทำการแก้ไขปรับปรุงการเซทค่าใน Hardware ได้โดยง่าย และทำให้เราสามารถที่จะใช้สีทั้งหมดที่มีอยู่ของ VGA ทั้ง 262,144 สี ได้อย่างเต็มที่

แต่สำหรับการเขียน Application ที่เป็น Graphics บน Windows นั้น Windows จะไม่ยอมให้เราเข้าถึง Hardware ได้โดยตรงจาก Application ของเรา เนื่องจากไม่มีฟังก์ชันที่เราสามารถเรียกใช้จาก GDI เพื่อที่จะแก้ไขตาราง hardware lookup table เราจึงต้องใช้โหมดโปรเทค (DPMI) สำหรับการเซทระดับลงขึ้นไปในวินโดวส์ โปรแกรมวินโดวส์จะประกาศการเซท RGB gun ตอนเริ่ม

ต้นไม้ การกำหนด GDI จะไม่มีความหมายสำหรับการเปลี่ยนฮาร์ดแวร์ทั้งหมด แม้แต่ฟังก์ชัน SetSystemPaletteUse() จะสามารถแปลได้เพียง 14 จาก 16 entry ที่เข้ามาเท่านั้น

Dithering

เมื่อแอพพลิเคชันของคุณได้กำหนดสีและเรียกใช้ฟังก์ชัน RGB() มันจะคืนค่า COLORREF กลับมา ค่านี้เป็น Long integer ที่ประกาศไว้ใน WINDOWS.H การเรียก RGB() จะต้องการตัวแปรคือ ค่าของสีตั้งแต่ 0 ถึง 255 เช่น ถ้าต้องการสีขาวสว่างจะต้องเรียก RGB(255,255,255) สีเทาจะเรียก RGB(191,191,191) สีเหลืองจะเรียก RGB(255,255,0) ลุดท้ายคำสั่ง RGB(127,127,0) จะให้สีน้ำตาลออกมา ค่าสีที่วินโดวส์อนุญาติให้ใช้ความเข้มมีเพียง 64 ค่าที่เพิ่มขึ้นเท่านั้น เป็นผลมาจากตาราง Hardware lookup table กำหนดเพียง 6 bit ที่ควบคุมความเข้ม จึงทำให้เซทได้เพียง 64 แบบ(2,2,2,2,2,2)

ถ้าคุณเจาะจงจะใช้ RGB() สำหรับแปรงสีหรือใส่สี ในโปรแกรมกราฟฟิกส์เกินกว่า 16 สี ที่จอ VGA จะรับได้วินโดวส์จะจำลองสีเหล่านี้เอง จะมีสองสีหรือมากกว่านั้นในพิคเซลที่แตกต่างจะรวมกันเป็นรูปแบบสีใหม่ขึ้นมา ทำให้ตามองเห็นเป็นสีใหม่ วิธีการจะทำให้แอพพลิเคชันของคุณไม่ขึ้นกับฮาร์ดแวร์ เมื่อเรียก RGB() จะสร้างแปรงที่ปรกติให้ เรียก dynamic dithering วินโดวส์จะแสดงสีของแบล็คกราวด์ให้อัตโนมัติ ทำให้ภาพกราฟฟิกส์ของคุณไม่ขึ้นกับอุปกรณ์อื่น โปรแกรมของคุณจะต้องการสีหรือกำหนด แต่ละวินโดวส์ ได้อย่างแท้จริงหรือจะสร้างเป็นสีอื่น ๆ ขึ้น

Dynamic dithering VS. static dithering

การจะใช้ RGB() ในการทำภาพสี คุณจะต้องสร้างแพลทสีสำหรับแอพพลิเคชันที่จะใช้ในการอธิบาย logical palette

คุณจะระบุสีใน logical palette โดยใช้ GDI ฟังก์ชัน CreatePalette() คุณจะเรียก logical palette เพื่อแสดงผลโดยการเรียกคำสั่ง SelectPalette() ใช้สีหนึ่งเพื่อเป็นตัวแสดงว่าคุณใช้สีใดอยู่ใน logical palette หมายเลขของสีจะระบุใน palette

การใช้งานในโหมด 256 สี

สำหรับ Super VGA, 8514/A, XGA และ graphics adapter อื่น ๆ มี memory อย่างเพียงพอที่จะเล่นในโหมด 256 สีได้ แต่ละ pixel เราแทนด้วย 8 bit ตัวอย่างเช่น ในโหมด 640x480x256 สี เราต้องการ 2,457,600(640x480x8) bit หรือเท่ากับ 307,200 byte ของ memory ที่ใช้แสดงผล ส่วนโหมด 800x600x256 สี ต้องการ 480,000 byte และในโหมด 1024x768x256 สี ต้องการ 686,432 byte ด้วยเหตุผลนี้เองที่การ์ดหลาย ๆ ชนิดถึงมี memory ขนาดเป็นเมกะไบท์เลยทีเดียว

ถ้า Windows รันในโหมด 256 สี เมื่อมีการลงสีเราสามารถจะกำหนดสีอื่น ๆ นอกจาก 16 สีเดิมที่เป็น default Windows จะใช้ hardware ในการสร้างสีใหม่ให้คุณ

Windows จะจอง hardware color 20 ส่วนแรกสำหรับการ run ในโหมด 256 สีบนจอ Super VGA, 8514/A หรือ XGA ถ้าเราสร้าง logical palette ในการหา custom color นั้น Windows จะสร้างให้ใหม่ใน hardware lookup table มันจะเป็น 20 สีแรกซึ่งเป็นสีเข้มไม่มีลักษณะของ dithered color

ลักษณะ Palette ที่ได้

หลังจากที่เราได้ใช้ฟังก์ชัน CreatePalette() และ SelectPalette() ในการสร้าง logical palette แล้ว เราสามารถที่จะเรียกฟังก์ชัน RealizePalette() เมื่อส่งให้ GDI ทำการ map custom colors ไปให้ hardware ถ้า application เรารันบนจอ Super VGA, 8514/A หรือ XGA ในโหมด 256สี ฟังก์ชัน RealizePalette() จะ adjust ตัว hardware register ของ graphics card ให้สร้าง table lookup ใหม่ อย่างไรก็ตามถ้าโปรแกรมของเรารันบนจอ VGA custom

Advanced color-mixing techniques

เราสามารถประยุกต์ใช้การปรับสีให้มีหลาย ๆ อัตราส่วน เพื่อให้แสดงภาพต่าง ๆ ได้ดีขึ้นเช่น ภาพสามมิติ แสงและเงาของวัตถุ เป็นต้น

Gradients

เราสามารถปรับ Gradients ได้ 2 วิธีคือ

- Single-hue gradients เช่น เริ่มต้นเรามี RGB(3,0,0) เป็น

สีแดงมืด เมื่อเราเพิ่ม argument แรกจนเป็น RGB(255,0,0) เราจะสามารถสร้างเงาของสีแดงจากแดงมืดเป็นแดงสว่าง

- Multiple-hue gradients เราสามารถเพิ่ม argument ตัวหนึ่งในขณะที่ลด argument อีกตัวหนึ่งก็ได้ หรือปรับ argument ตัวหนึ่งในขณะที่คง argument อีกตัวหนึ่งไว้ที่ค่าที่ไม่เทากับ 0 เช่น เปลี่ยนจาก RGB(255,255,0) ไปเป็น RGB(0,255,0) เราจะได้การเปลี่ยนจากสีเหลืองไปเป็นสีเขียว

Hue-value-chroma

ในแต่ละสีนั้นจริง ๆ แล้วเราสามารถพิจารณาได้เป็น 3 ส่วน คือ

- Hue หมายถึง สีที่เห็น เช่น แดง,เขียว,ส้ม และอื่น ๆ ซึ่งเป็นผลทางด้านของคลื่นแสงที่มากระทบกับ retina ของตา

- Value หมายถึง ความมืดหรือสว่างที่ปรากฏ สีแดงก็ยังคงเป็นแดงแต่ต่างกันที่ความสว่างว่ามากหรือน้อย

- Chroma หมายถึง ความเข้มของสีที่ปรากฏ

โดยใช้ GDI's control กับ RGB gun เราสามารถสร้างสีได้มากมาย โดยการปรับความสว่าง (value) หรือปรับ chroma เพื่อแสดงระยะใกล้ไกล

บทที่ 4 การโปรแกรมในโหมดกราฟฟิกส์ของวินโดวส์

(GRAPHICS PROGRAMMING FOR WINDOWS)

กราฟฟิกส์ของ Microsoft Windows จะมี library ที่สมบูรณ์อยู่แล้ว ซึ่ง function เหล่านี้สามารถถูกเรียกใช้โดย application ในขณะ run Graphical Environment เหล่านี้รวมเรียกว่า GDI (Graphics device interface) และ routine เหล่านี้รวมเรียกว่า GDI EXE dynamic-link library (DLL)

4.1 กราฟฟิกส์ดีไวซ์อินเทอร์เฟส (GDI)

GDI เป็น graphic ของ Windows ซึ่งสามารถส่ง graphics output ออกไปที่ screen,memory,printer และอุปกรณ์อื่น ๆ ได้ โดยในทางปฏิบัติจะมี context เป็นตัวบอกลักษณะที่แท้จริงของ output device ปัจจุบัน

Devices เป็นศูนย์กลางของ input และ output ได้แก่ keyboard, mouse และ timer ในการที่จะ call function ทาง graphic ใด ๆ จะต้องมีการ provide device context ไปยัง GDI

Device contexts เป็นการแสดง output device และ device driver ของมัน

Display-contexts เป็น type หนึ่งของ device context ซึ่งแสดง window บน screen จะแสดงถึง graphics output จะถูกเขียนลงบน client area ของ window บน screen อย่างไร

4.2 ดิสเพลย์คอนเท็กซ์ (Display Context)

เมื่อไรที่มีการสร้าง display-context สำหรับ window บน screen ตัว Windows จะเริ่มจุด origin(0,0) ไว้ที่มุมบนซ้ายสุดของ client area และ reset clipping rectangle ให้ match กับมิติของ client area โดยมี default attribute เพียง 2 อย่าง

Default attributes

Creating a display context ในการสร้าง display context โดยปรกติจะ call function "GetDC()" ของ GDI ซึ่งเป็นการ pass handle ของ window ที่ใช้เป็น output โดย GetDC() จะ return handle ไปที่ display-context ที่ถูกสร้างขึ้น ซึ่งจะใช้เมื่อมีการ call graphic function ใน GDI

Releasing a display-context ใช้ function "ReleaseDC()" ในการปลดปล่อย display-context

Saving and restoring a display-context ใช้ function "SaveDC" ในการ save และ restore display-context ที่ถูก save บน stack ก็จะใช้ function "RestoreDC()"

Compatible device context เป็นการจำลอง display-context ใน Windows programming โดยปรกติจะเป็น block ของ memory สำหรับใช้เหมือน bitmap

4.3 เครื่องมือในการวาดภาพ (Drawing tools)

ได้แก่ pens, brushes, fonts โดยจะมี Drawing tool function ต่าง ๆ สำหรับการสร้าง, เลือก และทำลาย drawing tools

Drawing tools functions ประกอบด้วย

1. Creating drawing tools ในการสร้าง pen ใหม่จะใช้ CreatePen() ในการสร้าง solid brush จะใช้ CreateSolidBrush() ในการสร้าง patterned brush จะใช้ CreateHatchBrush() หรือ CreatePatternBrush() นอกจากนี้ยังมี function GetStockObject() ในการเลือก drawing tools รูปแบบต่าง ๆ

2. Using drawing tools ในการใช้ drawing tools ใหม่ที่ถูกสร้างขึ้นมา เลือกมันให้เข้าไปใน display-context จะเรียก function SelectObject()

3. Deleting drawing tools จะใช้ function DeleteObject() ในการทำลาย custom drawing tools ที่สร้างขึ้นมา

Drawing attribute function

เป็นการ specialized routines ที่ modify วิธีการปฏิบัติของ drawing tools โดยที่ function เหล่านี้จะเปลี่ยนองค์ประกอบของ background

, drawing mode, text color และองค์ประกอบ bitblt ประกอบด้วย

1. องค์ประกอบของ background

SetBkColor() ใช้ set สีของ background

SetBkMode() ใช้ set background mode

2. Drawing modes

SetROP2() ใช้เพื่อเปลี่ยน Boolean logic ที่ GDI ใช้สำหรับ pens และ สำหรับการตกแต่ง filled object ภายใน ซึ่งประกอบด้วย Boolean operator เช่น OR, XOR, AND, NOT

3. BitBlt stretching attributes

SetStretchBltMode() ใช้ในการผ่าน image จาก clipboard เนื่องจาก GDI สามารถแผ่ขยายหรือหด bitmap ใด ๆ ที่คุณ copy มากับ bitblt function

BitBlt() ใช้ในการ copy bitmap ในขนาดเดียวกัน

4. TextColor

SetTextColor() ใช้ในการเปลี่ยนสีของ text

Pens

ใช้ function CreatePen() ในการสร้าง pen drawing tools ซึ่งจะระบุถึง style, width และ สีที่คุณต้องการ

default pen จะเป็นสีดำ, solid และหนา, pixel

Brushes

CreateSolidBrush() ใช้ในการสร้าง brush drawing tool ส่วน default brush เป็น solid white

Fonts

ใช้ function TextOut() โดยที่ TextOut() จะใช้ font ปัจจุบัน โดยที่ default ของ Windows จะเป็น san-serif, proportional-spaced ซึ่งเรียกว่า "System"

ถ้าต้องการ font ใหม่ต้องใช้ function GetStockObject() นอกจากนี้สามารถ modify วิธีการ display GDI fonts ก็สามารถใช้ CreateFont()

เพื่อระบุขนาด, style, weight และ attribute อื่น ๆ

Color

จะถูกใช้โดย pens, brushes และ fonts วิธีที่ง่ายที่สุดในการระบุสีคือการใช้ RGB()

RGB descriptions

RGB() จะมี argument 3 ตัว แสดงถึงสีแดง, เขียว และสีน้ำเงิน โดย RGB(0,0,0) จะได้สีดำ, RGB(255,255,255) จะได้สีขาว, RGB(0,0,255) จะได้สีน้ำเงิน และ RGB(127,127,127) จะได้สีเทา

Output operations

Line, rectangle, polygon and ellipse ใช้ในการสร้างเส้น, สี่เหลี่ยม, วงกลม และรูปทึบ

1. MoveTo() เป็นการกำหนดจุดเริ่มต้นของ pen
2. LineTo() เป็นการลากเส้นจากจุดเริ่มต้นไปยัง position ที่กำหนด
3. Polyline() เป็นการวาด set ของ connected line segments
4. LineDDA() สามารถใช้หา coordinate xy ของแต่ละ pixel บน line ได้ ๆ
5. Rectangle() ใช้วาดสี่เหลี่ยมโดยใช้ 4 จุด
6. RoundRect() ใช้วาด round-cornered rectangle โดยใช้จุด 4 จุด
7. Polygon() ใช้สร้าง polygon ที่มีด้าน > 1 ด้าน array ของ point ที่ใช้งาน
8. Ellipse() ใช้สร้างวงกลมและวงรี

Bitmap

Bitmap คือ array หรือ matrix ของ bit ใน RAM ซึ่งเป็นตัวแทนของ image ซึ่งสามารถที่จะวาดอะไรก็ได้ลงบน bitmap เหมือนกับวาดลงใน main window ใน application สามารถที่จะ copy bitmap จาก memory ไปยัง screen, จาก screen ไปยัง memory ได้ โดยมี function ที่ใช้ดังนี้

1. CreateCompatibleDC() ใช้สร้าง memory-display context สำหรับซ่อน bitmap

2. CreateCompatibleBitmap() ใช้สร้าง bitmap compatible พร้อมกับ display-context

3. SelectObject() ใช้ในการเลือก bitmap ไปใน display-context หรือ ไปใน memory-display context

4. BitBlt() ใช้ copy bitmap จาก display context หนึ่ง ๆ หรือ device context หนึ่ง ๆ ไปยังอีกอันหนึ่ง

5. DeleteObject() ใช้ทำลาย bitmap

6. StretchBlt() ใช้ copy และเปลี่ยนขนาดของ bitmap เพื่อให้พอดีกับ target display-context หรือ device context

7. GetObject() ใช้เรียกรายละเอียดจาก bitmap header เกี่ยวกับความกว้าง, ความลึก และอื่น ๆ

Clipboard

เป็น buffer ที่ถูก maintain โดย Windows ขณะ run มันมีประโยชน์สำหรับการ share data ระหว่าง application โดยมี function ต่าง ๆ ดังนี้

1. CreateCompatibleDC() ใช้สร้าง memory-display context เพื่อที่จะ hold bitmap สำหรับ clipboard

2. CreateCompatibleBitmap() ใช้สร้าง bitmap สำหรับการ read/write clipboard

3. OpenClipboard() ใช้ในการเปิด clipboard

4. EmptyClipboard() ลบ context ภายในของ clipboard

5. SetClipboardData() ใช้ผ่าน bitmap handle ไปยัง clipboard

6. CloseClipboard() ปลดปล่อย clipboard สำหรับให้ใช้ใน application อื่น ๆ

7. GetClipboardData() ใช้เรียก bitmap handle จาก clipboard

Metafiles

Metafiles เป็น collection of GDI function ซึ่งจะถูกเก็บไว้สำหรับ playback ครั้งล่าสุด คุณสามารถใช้ metafile ใน application ที่ต้องติดต่อกับ graphics เพื่อเก็บ action ของ end-user คล้าย regen function

การสร้าง Metafile

สร้าง Metafile โดยใช้ function "CreateMetaFile()"

การเขียน Metafile

ส่ง output ในลักษณะ graphic ไปยัง Metafile โดยให้เรียก graphics routines ของ GDI นอกจากว่าจะให้ metafile device-context จัดการแทน display-context แล้วก็ draw line, rectangles, ellipses บน metafile

การ Lock Metafile

พอเก็บ statements ซึ่งต้องเก็บ Metafile โดยเรียก function PlayMetaFile() แล้ว Windows จะ read metafile และจะ execute แต่ละ function ก็เรียกจากตรงนี้ ต้องมีการ close Metafile ก่อนที่จะเล่นมัน หรือเก็บลงใน disk

การลบ Metafile

หลังจากเล่น metafile เสร็จ จะต้อง call function DeleteMetaFile() เพื่อทำลาย RAM version

Fonts and Text

ประกอบด้วย function ต่าง ๆ ดังนี้

1. TextOut() ใช้ในการ display text string โดยใช้ font ปัจจุบัน

2. GetStockObject() ใช้ในการ retrieve handle ไปยัง system font

3. SelectObject() ใช้ในการเลือก font ไปยัง display-context หรือ device context

4. SetTextColor() set สีของ text (เฉพาะ foreground)

5. SetBkColor() set สีของ text cel (เป็นสี background)

6. SetBkMode() set background ของ text cel ให้เป็น opaque

หรือโปร่งใส

7. CreateFont() สร้าง logical font ด้วยความสูงที่ระบุ, น้ำหนัก, style และอื่น ๆ

Printer

ประกอบด้วย function ต่าง ๆ ดังนี้

1. GetProfileString() ใช้เรียกรายละเอียดของ printer จากไฟล์ WIN.INI

2. LoadLibrary() ใช้ load driver ของ printer จาก disk

3. CreateDC() สร้าง memory-device context โดยใช้ default ของ printer

4. GetProcAddress() เรียก address ของ function ของ driver เพื่อที่จะสามารถแก้ไข default ที่กำหนดไว้ได้

5. FreeLibrary() ปล่อย printer driver ออกมาจาก memory

6. GetDeviceCaps() เรียก capability ทั้งหมดของ printer

7. Escape() เริ่มต้น print งาน ส่งงานที่ printer ไปยัง spooler ของ Windows

บทที่ 5 ไฟล์บิทแมพของวินโดวส์ - รูปแบบ BMP

(WINDOWS BITMAP FILE - BMP FORMAT)

Bitmap ไฟล์เป็นรูปแบบของไฟล์ที่สร้างขึ้นเพื่อแสดง และ การเก็บภาพบน Microsoft Windows รูปแบบของ Bitmap file สามารถเก็บรูปภาพ (image) ได้ทั้ง 1 บิท 4 บิท 8 บิท และ 24 บิท ต่อหนึ่งจุด การเก็บในแบบ 1 บิท 4 บิท 8 บิท เป็นการเก็บแบบ color map ในขณะการเก็บแบบ 24 บิท เป็นการเก็บสีต่าง ๆ โดยตรง

ในแต่ละไฟล์จะประกอบด้วย file header, bitmap header, color map (color map นี้จะไม่นำมาใช้อะไรถ้าเป็นการเก็บแบบ 24 บิท) และ ภาพ (image) ภาพที่เก็บโดยใช้ format นี้สามารถลดขนาดของข้อมูลได้โดยใช้วิธี RLE

File Header

ในแต่ละไฟล์จะมี File header ซึ่งมีรูปแบบตามตารางดังต่อไปนี้

Offset	Size	Name	Description
0	2	bfType	ASCII "BM"
2	4	bfSize	Size in longwords (4-byte units) of the file
6	2	bfReserved1	zero
8	2	bfReserved2	zero
10	4	bfOffBits	Byte offset after header where image begins

ในฟิลด์ของ bfOffBits จะเป็นระยะทางตั้งแต่ไบท์ที่ 14 จนถึงบิทแรกของรูปภาพ เพื่อความสะดวกเมื่อไม่ต้องการทราบหรือไม่ต้องการอ่าน bitmap header

การแยกระหว่าง OS/2 format กับ Windows 3.x format ทำได้โดยการตรวจสอบ offset 14 ซึ่ง ถ้าเป็น 12 คือ OS/2 format แต่ถ้าเป็น 40 หมายความว่าเป็น Windows format

Windows 3.x Bitmap Header

ภายหลังจาก file header แล้วจะตามด้วย bitmap header มีรายละเอียดดังต่อไปนี้

Offset	Size	Name	Description
14	4	biSize	Size of this header, 40 bytes.
18	4	biWidth	Image width in pixels.
22	4	biHeight	Image height in pixels.
26	2	biPlanes	Number of image planes, must be 1
28	2	biBitCount	bits per pixel 1, 4, 8 or 24
30	4	biCompression	Compression type, below.
34	4	biSizeImage	Size in bytes of compressed image.
38	4	biXPelsPerMeter	Horizontal resolution, in pixels/meter.
42	4	biYPelsPerMeter	Vertical resolution, in pixels/meter.
46	4	biClrUsed	Number of colors used, below.
50	4	biClrImportant	Number of "important" colors.
54	4*N	bmiColors	Color map.

Color map

การสร้างภาพโดยการเก็บแบบ 1 บิท 4 บิท 8 บิท ต่อหนึ่งจุดนั้นจำเป็นต้องมี color map ขนาดของ color map จะมีได้คือ 2, 16, 256 แต่อย่างไรก็ดี ถ้าไม่ใช้สีครบทั้งหมด สามารถลดขนาดของ color map ได้ ในฟิลด์ของ biClrUsed ถ้าไม่เป็นศูนย์จะหมายถึงจำนวนของสีที่ใช้ใน color map แต่ถ้าฟิลด์นี้เป็นศูนย์ หมายความว่าใช้สีทั้งหมด ถ้าเป็นการเก็บแบบ 24 บิทต่อหนึ่งจุดจะไม่จำเป็นจะต้องใช้ color map เพราะเป็นการเก็บแบบ RGB โดยตรงอยู่แล้ว ขนาดของฟิลด์ biClrUsed ไม่จำเป็นจะต้องเป็นศูนย์ และใช้ฟิลด์นี้เป็นตัวบอกขนาดของ color map

เนื่องจากการแสดงภาพนั้น สีที่แสดงได้ถูกจำกัดโดยความสามารถของอุปกรณ์ ในฟิลด์ biClrImportant ถ้าไม่ใช่ศูนย์ จะบอกถึงจำนวนสีที่ต้องการสำหรับการแสดงให้ได้ภาพที่สมบูรณ์ Color map จะประกอบด้วยข้อมูล 4 ไบท์ดังนี้

Offset	Name	Description
0	rgbBlue	Blue value for color map entry.
1	rgbGreen	Green value for color map entry.
2	rgbRed	Red value for color map entry.
3	rgbReserved	Zero

Windows 3 Bitmap Data

Bitmap data เป็นข้อมูลที่จะต่อมาจาก color map ข้อมูลนี้อาจเป็นชนิดที่ยังไม่ได้ลดขนาดข้อมูลหรือ ถ้าเป็นการเก็บแบบ 4 บิทหรือ 8 บิทต่อหนึ่งจุดสามารถลดขนาดข้อมูลได้โดยวิธี RLE

Bitmap with one bit per pixel

ข้อมูลในแต่ละจุดจะมีขนาดหนึ่งบิทและรวมเก็บไว้ โดยเก็บ 8 จุดต่อหนึ่งไบท์ High bit ของแต่ละไบท์จะเป็นจุดที่อยู่ทางซ้ายสุด

Bitmap with four bits per pixel

ถ้าเป็นรูปที่ยังไม่มีการลดขนาดของข้อมูลจะถูกเก็บรวมกัน 2 จุดต่อหนึ่งไบท์ 4 บิทแรกจะเป็นของจุดที่อยู่ทางซ้ายสุด ในแต่ละแถวจะถูกกั้นด้วยข้อมูลจำเพาะอีก 4 ไบท์

ถ้าเป็นข้อมูลที่ทำการลดขนาดข้อมูลโดยการใช้รูปแบบการเข้ารหัส RLE ซึ่งมีวิธีต่าง ๆ อีกคือ Repeating group, Literal group, และ Special group

Repeating group จะเป็นข้อมูล 2 ไบท์ ไบท์แรกจะเป็นจำนวนของจุดและไบท์ที่สองจะเป็นคู่ของจุด ตัวอย่างเช่น 05h 24h จะหมายความถึง 2 4 [illegible] 4 2

Literal group จะเริ่มด้วยจำนวนจุด และ ตามต่อมาด้วยจุดที่ตามมา จำนวนจุดอย่างน้อยที่สุด 3 ถ้าเป็น 1 หรือ 2 สามารถใช้การเข้ารหัสแบบ Repeating group ข้อมูลจะถูกล้อมด้วยตัวเลขศูนย์เป็นจำนวนคู่ของไบท์ ตัวอย่างเช่น 00h 05h 12h 34h 50h 00h เป็นการแทนความหมายของ 1 2 3 4 5 ชุดตัวเลข 00 00 หมายถึงสิ้นสุดแถว 00 01 หมายถึงสิ้นสุด Bitmap และถ้ามีตัวเลขชุดพิเศษคือ 00 02 xx yy ตามมาด้วยจะหมายถึง delta positon คือมีรูปต่อไปอีกทางขวา xx จุดและลงล่าง yy จุด

Bitmap with eight bits per pixel

ถ้าเป็นข้อมูลที่ไม่มีการลดขนาดจะเก็บจุดละหนึ่งไบท์แต่ละแถวจะล้อมรอบด้วยข้อมูลขนาด 4 ไบท์ แต่ถ้าทำการลดขนาดโดยใช้การเข้ารหัสแบบ RLE ซึ่งจะมีวิธีการอีก 3 วิธีเช่นกันคือ Repeating group, Literal group, และ Special group

Repeating group จะเป็นข้อมูล 2 ไบท์ ไบท์แรกจะเป็นจำนวนของจุดและไบท์ที่สองจะเป็นค่าของจุด ตัวอย่างเช่น 05h 24h จะหมายความถึง 24h 24h 24h 24h 24h

Literal group จะเริ่มด้วยจำนวนจุด และ ตามต่อมาด้วยจุดที่ตามมา จำนวนจุดอย่างน้อยที่สุด 3 ข้อมูลเหล่านี้จะถูกล้อมด้วยตัวเลขศูนย์เป็นจำนวนเลขคู่ของไบท์ ตัวอย่างเช่น 00h 05h 12h 34h 56h 78h 9ah 00h จะหมายถึง 12h 34h 56h 78h 9ah ข้อมูลพิเศษที่ตามมามีความหมายเช่นเดียวกับการเก็บแบบ 4 บิท

Bitmap with 24 bits per pixel

แต่ละจุดจะประกอบด้วยข้อมูลขนาด 3 ไบท์จะเป็นค่าของสีต่าง ๆ ทั้ง 3 สี คือ แดง, เขียว และ น้ำเงิน ในแต่ละแถวจะล้อมด้วยเลขศูนย์ขนาด 4 ไบท์

บทที่ 6 ส่วนโครงงานและผลการทดลอง

โครงงานนี้เป็นการสร้างแอพพลิเคชันไว้ใช้ในการแต่งภาพใบหน้าและทรงผม โดยทำการเก็บภาพจากกล้อง VDO ซึ่งภาพที่นำมาใช้งานเป็นภาพที่มีสีจำนวน 256 สี ซึ่งในการทำงานจะทำงานในโหมดสีสูงในวินโดวส์

6.1 ฮาร์ดแวร์ที่ใช้ (Hardware)

อุปกรณ์ที่ใช้ทำการจับภาพมาทำการประมวลผลในโครงงานนี้ใช้ การ์ด Image Grabber ซึ่งเป็นการ์ด Interface PC กับ สัญญาณวิดีโอ ให้มาเป็นภาพบนจอภาพ

คุณสมบัติของการ์ด PV-6200

คอมพิวเตอร์และการ์ดวีจีเอที่ใช้

- ใช้สล๊อต 16 บิท ของคอมพิวเตอร์ IBM PC/AT หรือเครื่องคอมแพทิเบิล 1 ช่อง
- ใช้หน่วยความจำ 1 เมกกะไบท์ ซึ่งอยู่ระหว่าง 1 เมกกะไบท์กับ 16 เมกกะไบท์
- รีจิสเตอร์I/Oทั้งหมด ใช้ 2 แอดเดรส เขียนข้อมูลผ่านรีจิสเตอร์ข้อมูลไปยังรีจิสเตอร์I/O โดยใช้รีจิสเตอร์อินเดกซ์ และเคลื่อนย้ายข้อมูลแบบ 8 บิท
- การ์ดวีจีเอที่ใช้ต้องมีคอนเน็คเตอร์ 26 ขา

หน่วยความจำแสดงผลภาพ

- การ์ด PV-6200 มีหน่วยความจำแสดงผลภาพขนาด 768 กิโลไบท์ โดยแบ่งเป็น 1024 (แนวนอน) x 512 (แนวตั้ง) x 12 บิทต่อจุด(pixel)
- ใช้หน่วยความจำแสดงผลภาพแบบ 2 พอร์ท เพื่อแปลงภาพวิดีโอไปแสดงบนจอวีจีเอ
- แต่ละจุดประกอบด้วย 12 บิท ความสว่าง 8 บิท และสัญญาณสี 4 บิท (สัญญาณสี U 2 บิท, V 2 บิท) ความละเอียดของสีเท่ากับ 2**21(2 ล้าน)สี

การจับภาพ

เลือกต้นกำเนิดสัญญาณภาพสามารถได้ 3 สัญญาณเข้า ระบบ NTSC หรือ PAL

- สัญญาณเข้าเป็น คอมโพสิทวีดีโอ(composite video) หรือ เอสวีดีโอ(S-video)
- อิมพีแดนซ์ของสัญญาณเข้าสามารถเลือกจากดิพสวิทช์ว่าจะใช้ 75โอห์มเทอร์มิเนเตอร์หรือไม่
- การปรับแต่งภาพใช้ซอฟท์แวร์ควบคุม

- สามารถปรับขนาดของภาพได้โดยอิสระทั้งแนวนอนและแนวตั้ง
- ใช้เวลา 1/30 วินาทีในการจับภาพในระบบNTSC และ 1/25 วินาทีในระบบ PAL

การแสดงช่องหน้าต่างและสี

- ใช้ "window key" และ "color key" ในการแสดงภาพจากวิดีโอบนจอวีจีเอ
- window key ใช้กำหนดขนาดภาพวิดีโอ จาก 1/64 ถึง 64/64 แสดงบนจอวีจีเอ
- color key ใช้เลือกสีจาก 256 สีของวีจีเอ เพื่อแทนที่จุดที่สีเหมือนกับ color key ด้วยข้อมูลจากภาพวิดีโอ

การควบคุมการแสดงภาพ

- ภาพวิดีโอแพนได้ 4 จุด เลื่อนได้ 1 เส้น
- ขยายภาพได้ 2 เท่าทั้งแนวนอน และแนวตั้ง
- window key ใช้กับภาพวิดีโอที่ย่อหรือไม่ย่อก็ได้

สัญญาณออก

- ให้สัญญาณออก 31.5 KHz non-interlaced บนจอวีจีเอมาตราฐาน ประกอบด้วยสัญญาณสีแดง,สีเขียว,สีน้ำเงิน สัญญาณซิงค์แนวนอนและแนวตั้ง ใช้กับจอมัลติซิงค์ได้
- สัญญาณภาพมีความละเอียด 720(แนวนอน) x 480(แนวตั้ง)
- อิมพีแดนซ์ของสัญญาณออก 75โอห์ม
- โปรแกรมปรับความคมชัด,ความสว่าง,และความเข้มของสีแดง,สีเขียว,สีน้ำเงินได้

ซอฟท์แวร์

- Dynamic Link Library (DLL) บน MS Window 3.0 ความละเอียด 640 (แนวนอน)x 480(แนวตั้ง)
- ไลบรารี่ภาษา C สำหรับ DOS 3.0 ขึ้นไป มีการทำงานเหมือนกับ DLL ความละเอียด 640 (แนวนอน)x 480(แนวตั้ง)
- ตัวแปลคำสั่ง สำหรับ DOS 3.0 ขึ้นไป มีการทำงานเหมือนกับ DLL ความละเอียด 640 (แนวนอน)x 480(แนวตั้ง)
- โปรแกรมอรรถประโยชน์ เช่น เปลี่ยนรูปแบบของแฟ้มภาพ

รูปแบบของแฟ้มภาพ

- TGA กำหนดโดย Truevision
- MMP กำหนดโดย IBM
- VMP กำหนดโดย Chips & Technologies

ตัวการ์ดPV-6200

- มีขนาด 3/4 ของการ์ดอะแดปเตอร์ของ IBM PC/AT ขนาด 255(ยาว) x 117(สูง)x 23(กว้าง)
- แรงดัน.....กระแส

  +5.....2.5A

  +12.....0.4A

  -12.....0.05A

6.2 เทอร์โบ C++ สำหรับวินโดวส์

ในชุดของเทอร์โบ C++ สำหรับวินโดวส์จะประกอบด้วยแผ่นดิสเกตต์ขนาด 5.25 นิ้ว ความจุ 1.2 เมกะไบต์ 5 แผ่น และมีหนังสือคู่มืออีก 4 เล่มคือ Turbo C++ 3.0 for Windows User's Guide และ Programmer's Guide รวมกัน 1 เล่ม , Resource Workshop User's Guide , Turbo Debugger 3.0 for Windows User's Guide, Object Windows for C++ User's Guide

<u>ฮาร์ดแวร์และซอฟต์แวร์ที่จำเป็น</u>

1.เครื่องคอมพิวเตอร์ที่ใช้ซีพียู 286, 386 และ 486

2.หน่วยความจำหลัก 640 กิโลไบต์ และหน่วยความจำเอ็กซ์เทนเดตที่ต้องมีอีกอย่างน้อย 2 เมกะไบต์

3.จอภาพต้องเป็นอีจีเอ หรือวีจีเอขึ้นไป

4.ฟลอปปี้ดิสค์ 1 ตัวและฮาร์ดดิสค์ซึ่งต้องมีที่ว่างอย่างน้อย 16 เมกะไบต์

5.เม้าส์ เพื่อการใช้งานโปรแกรมได้อย่างเต็มที่

6.ไมโครซอฟต์วินโดวส์ 3.0 หรือ 3.1 และดอสตั้งแต่เวอร์ชัน 3.0 ขึ้นไป

<u>การติดตั้งโปรแกรม</u>

เทอร์โบ C++ ได้เตรียมโปรแกรม INSTALL.EXE สำหรับติดตั้งมาให้เรียบร้อย โดย INSTALL จะก็อบปี้ไฟล์ทั้งหมดที่จำเป็นลงในฮาร์ดดิสค์ให้ พร้อมทั้งจะสร้างไดเรคทอรีทั้งหมดที่จำเป็นให้ด้วย โดยจะต้องเรียกวินโดวส์ให้ทำงานก่อน ขณะที่ใช้โปรแกรมเมเนเจอร์

เป็นเชลล์โปรแกรมอยู่ ถ้าใช้โปรแกรมตัวอื่นเป็นเชลล์จะต้องยกเลิกออปชันไม่ให้สร้างเทอร์โบซีกรุ๊ป

การติดตั้งเริ่มต้นอาจทำได้โดยเรียกวินโดวส์ที่ดอสดังนี้

```
C:\>WIN A:INSTALL
```

หรืออาจใช้คำสั่ง RUN โดยกด ALT-F + R ขณะที่อยู่ในโปรแกรมเมเนเจอร์ให้รัน INSTALL.EXE ก็ได้ หลังจากที่ INSTALL เริ่มทำงานโปรแกรมจะอนุญาตให้ผู้ใช้แก้ไขชื่อไดเรคทอรีต่างๆก่อนที่จะเริ่มติดตั้ง ในระหว่างที่ก๊อบปี้ไฟล์จะมีรูปมิเตอร์แสดงเปอร์เซนต์ที่ติดตั้ง เมื่อการติดตั้งเสร็จก็จะมีเทอร์โบซีกรุ๊ปสร้างให้เรียบร้อย

เริ่มต้นใช้งาน

วิธีเรียกเทอร์โบ C++ มาทำงานนั้น ทำได้โดยการคลิกเมาส์สองครั้งที่ไอคอนของตัวเทอร์โบ C++ ก็จะเข้าสู่โปรแกรมทันที ถ้ามีโปรเจกต์หลายอันก็อาจกำหนดให้แต่ละโปรเจกต์แทนด้วยไอคอนได้โดยใช้คำสั่ง New (ALT-F + N) จากโปรแกรมเมเนเจอร์และพิมพ์ TCW ตามด้วยชื่อโปรเจกต์ไฟล์ที่ต้องการ

การคอมพายล์และรันโปรแกรม

เริ่มจากโหลดไฟล์ซึ่งเป็นโปรเจกต์ไฟล์ชื่อ WHELLO.PRJ โดยเลือก Project:Open Project จากนั้นกด ALT-F9 รอสักครู่โปรแกรมก็จะถูกคอมพายล์และลิงค์เรียบร้อย ทดลองรันโปรแกรมที่ได้โดยกด CTRL-F9 ก็จะได้โปรแกรมที่ทำงานบนวินโดวส์

6.3 ผลการทดลอง

ความสามารถของโปรแกรม

โปรแกรมใช้ในการแต่งภาพใบหน้าที่เป็นโครงงานนี้มีความสามารถต่าง ๆ ดังนี้ คือ

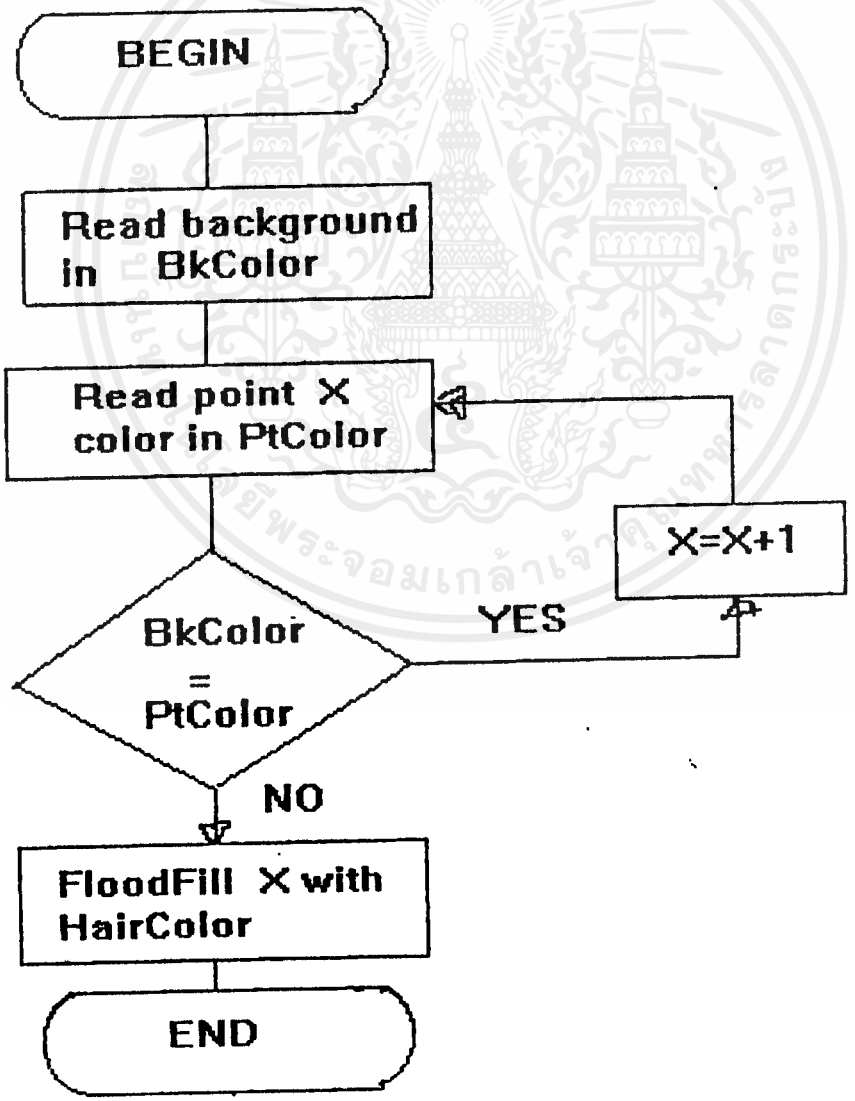
1. เปลี่ยนสีทรงผมของใบหน้า
2. เปลี่ยนทรงผมของใบหน้า หรือทำการรวมทรงผมกับใบหน้า
3. เปลี่ยนส่วนประกอบของใบหน้าคือ ตา และ จมูก
4. การแสดงผล การจัดเก็บ การอ่านข้อมูลจากไฟล์บิทแมพ (BMP)
5. การผ่านข้อมูล input/output ผ่านทางคลิปบอร์ดเพื่อจะนำภาพที่แต่งแล้วไปใช้งานโดยแอพพลิเคชันอื่น

1. การเปลี่ยนสีทรงผม

เป็นการเปลี่ยนสีทรงผมของภาพใบหน้าที่เป็นต้นแบบโดยตัดทรงผมของใบหน้าแยกออกมาต่างหากก่อนจะทำการเปลี่ยนสีทรงผม แล้วจึงนำภาพกลับไปรวมกันอีกทีหนึ่ง โดยสีทรงผมที่มีให้ในการเปลี่ยนสีคือ

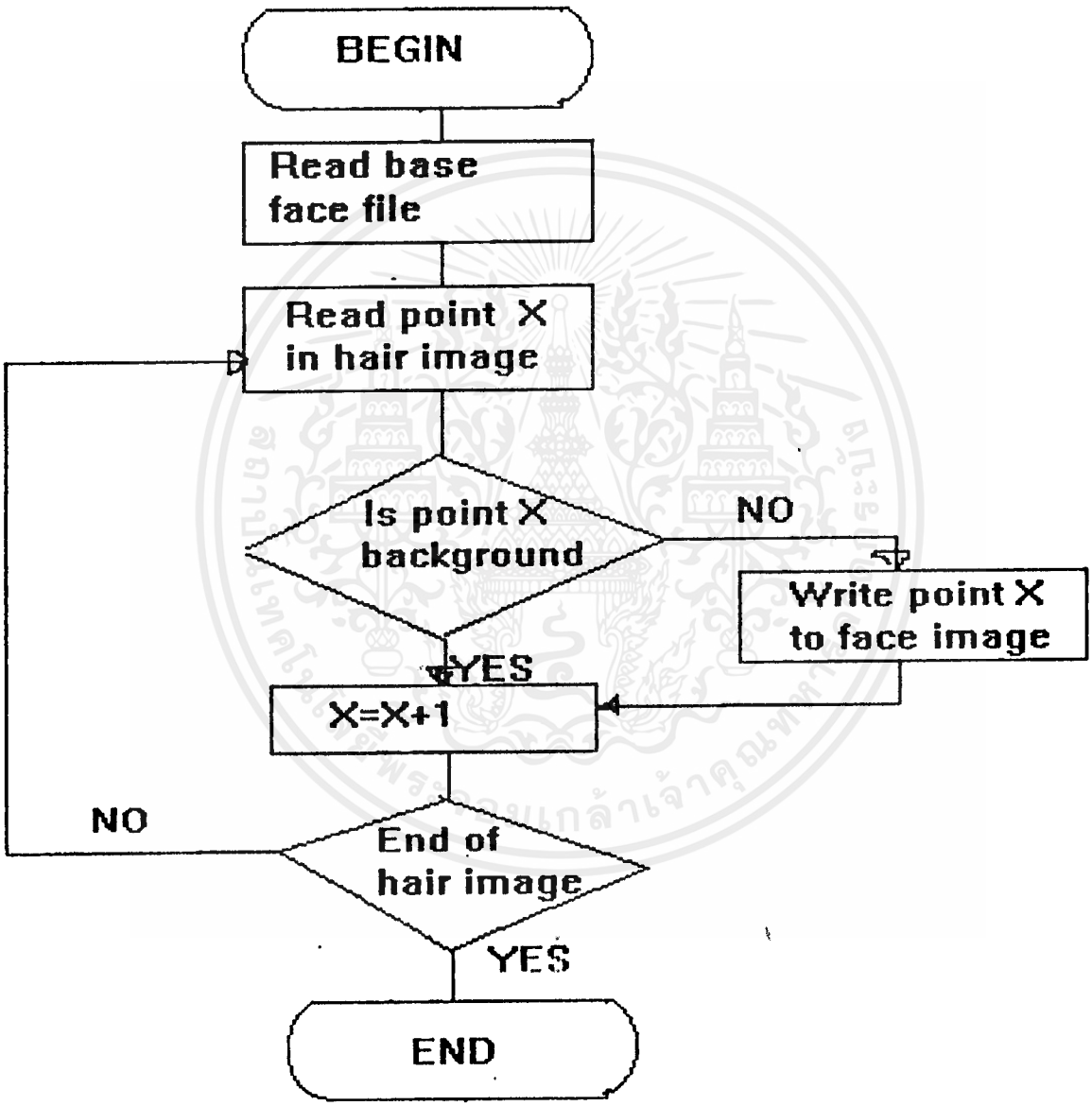
- สีดำ (Black)
- สีน้ำตาลสว่าง (Bright Brown)
- สีน้ำตาล (Brown)
- สีน้ำตาลมืด (Dark Brown)
- สีน้ำตาลแดง (Red-Brown)

การทำงานของโปรแกรมในการเปลี่ยนสีทรงผมของภาพสามารถเขียนเป็นโฟลวชาร์ตได้ดังนี้คือ



2. การเปลี่ยนหรือการรวมทรงผมเข้ากับใบหน้า

ในส่วนนี้เป็นการรวมทรงผมที่อยู่ด้านขวาของ screen เข้ากับภาพใบหน้าในด้านซ้ายของ screen โดยก่อนที่จะทำงานนี้ต้องมีภาพทรงผม และภาพพื้นหน้าของใบหน้านั้น ๆ ก่อน



3. เปลี่ยนส่วนประกอบของใบหน้า ส่วน ตา และ จมูก

นำภาพ ตา หรือจมูก ของภาพต่าง ๆ มารวมกันแล้วสามารถเปลี่ยน ตา หรือจมูกให้กับภาพได้มีขั้นตอนการทำงานดังนี้

1. อ่านไฟล์ของส่วนประกอบต้นแบบ (ตา, จมูก)

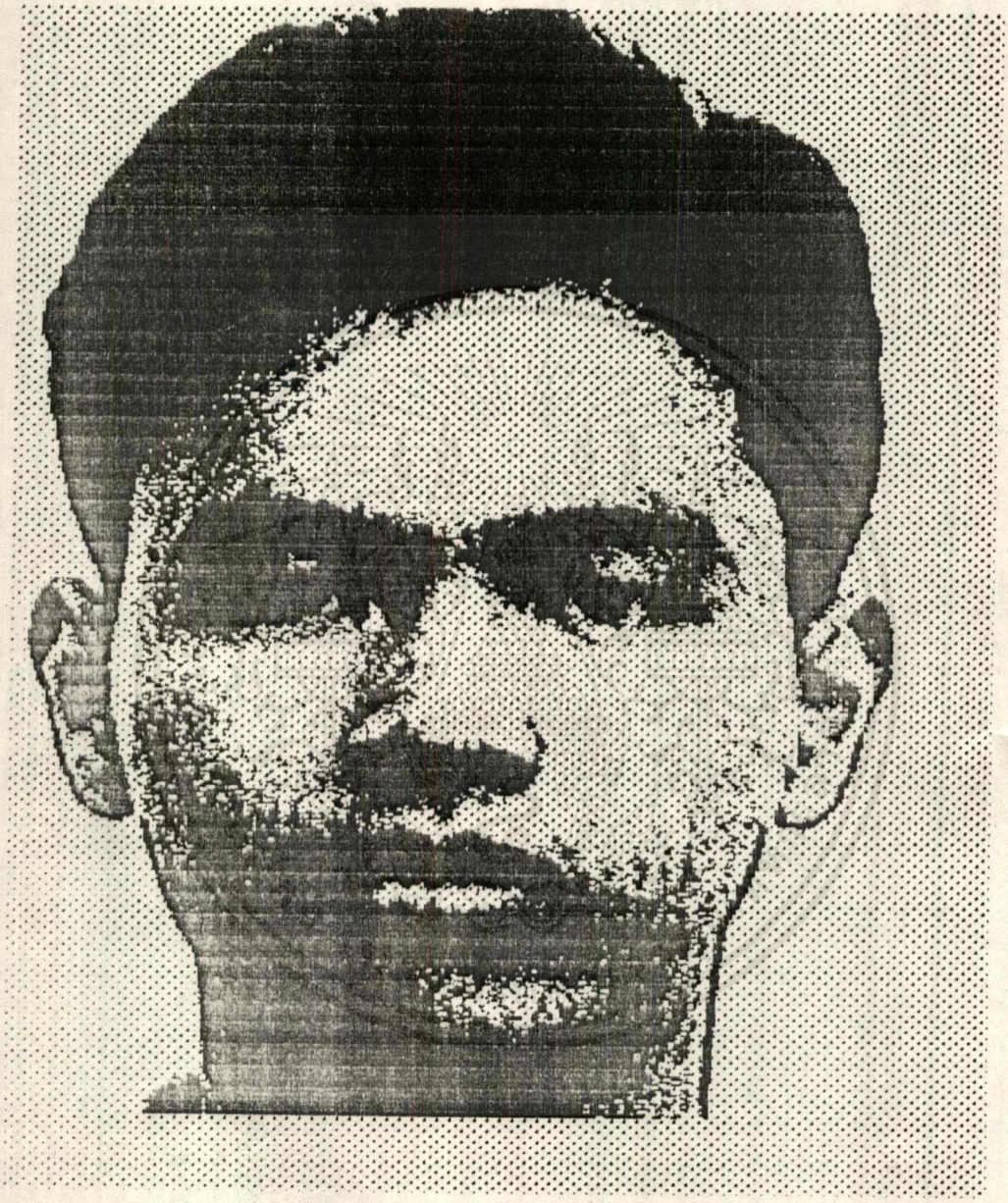
2. ทำการลบส่วนประกอบนั้น ด้วยสีพื้นใบหน้าที่อ่านมาจากข้อ 1

3. อ่านไฟล์ของส่วนประกอบใหม่ที่ต้องการจะเปลี่ยนให้ใบหน้า

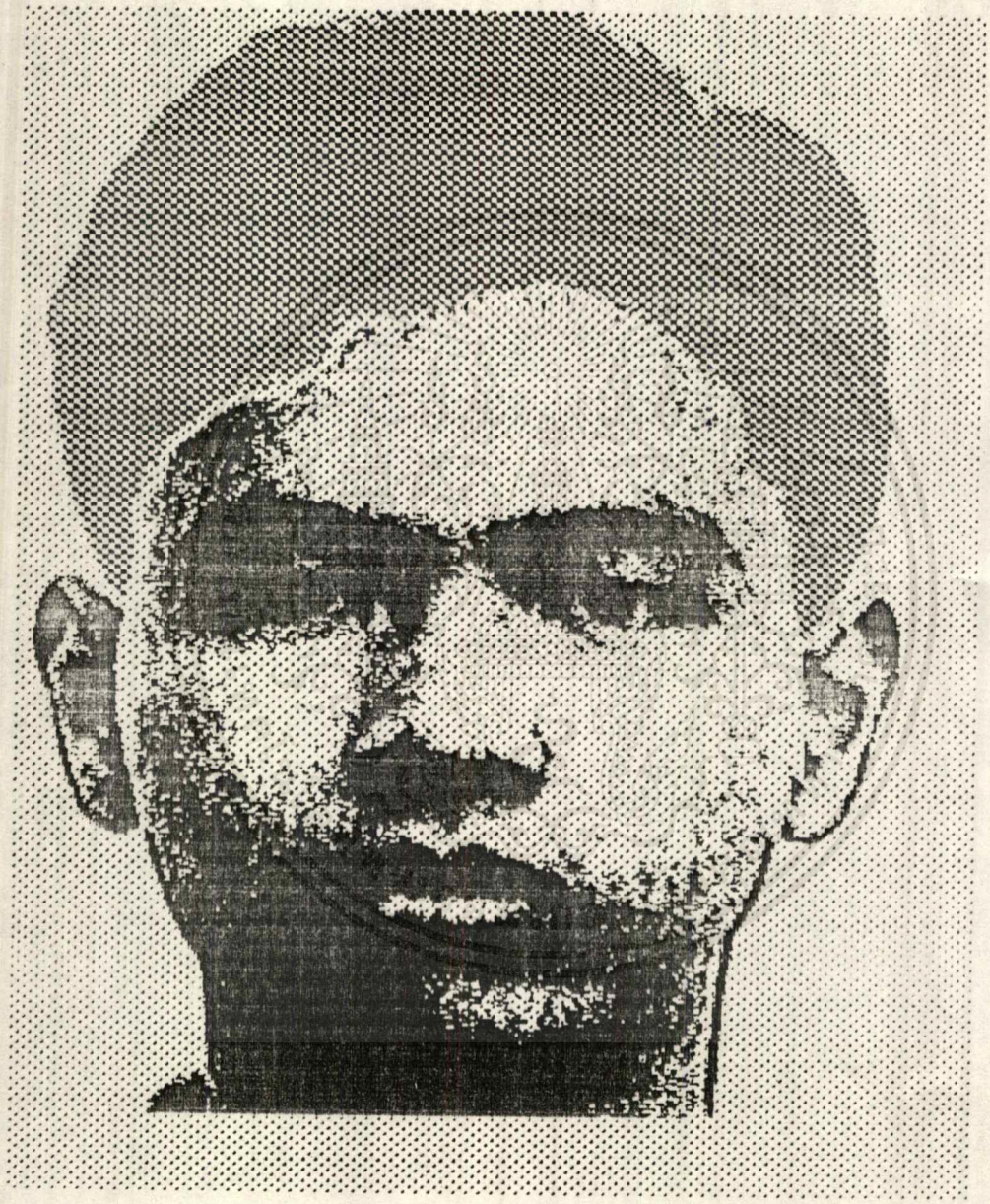
4. อ่านค่าจุดแต่ละจุดบนภาพที่อ่านขึ้นมา ถ้าไม่ใช่สีพื้นให้เขียนจุดนั้นลงบนภาพใบหน้าที่ตำแหน่งเดียวกับมุมบนซ้ายของส่วนประกอบแรก แต่ถ้าเป็นสีพื้นก็จะไม่ทำงาน

5. เพิ่มค่าตัวนับของจุด ถ้ายังไม่จบไฟล์ กลับไปทำข้อ 4

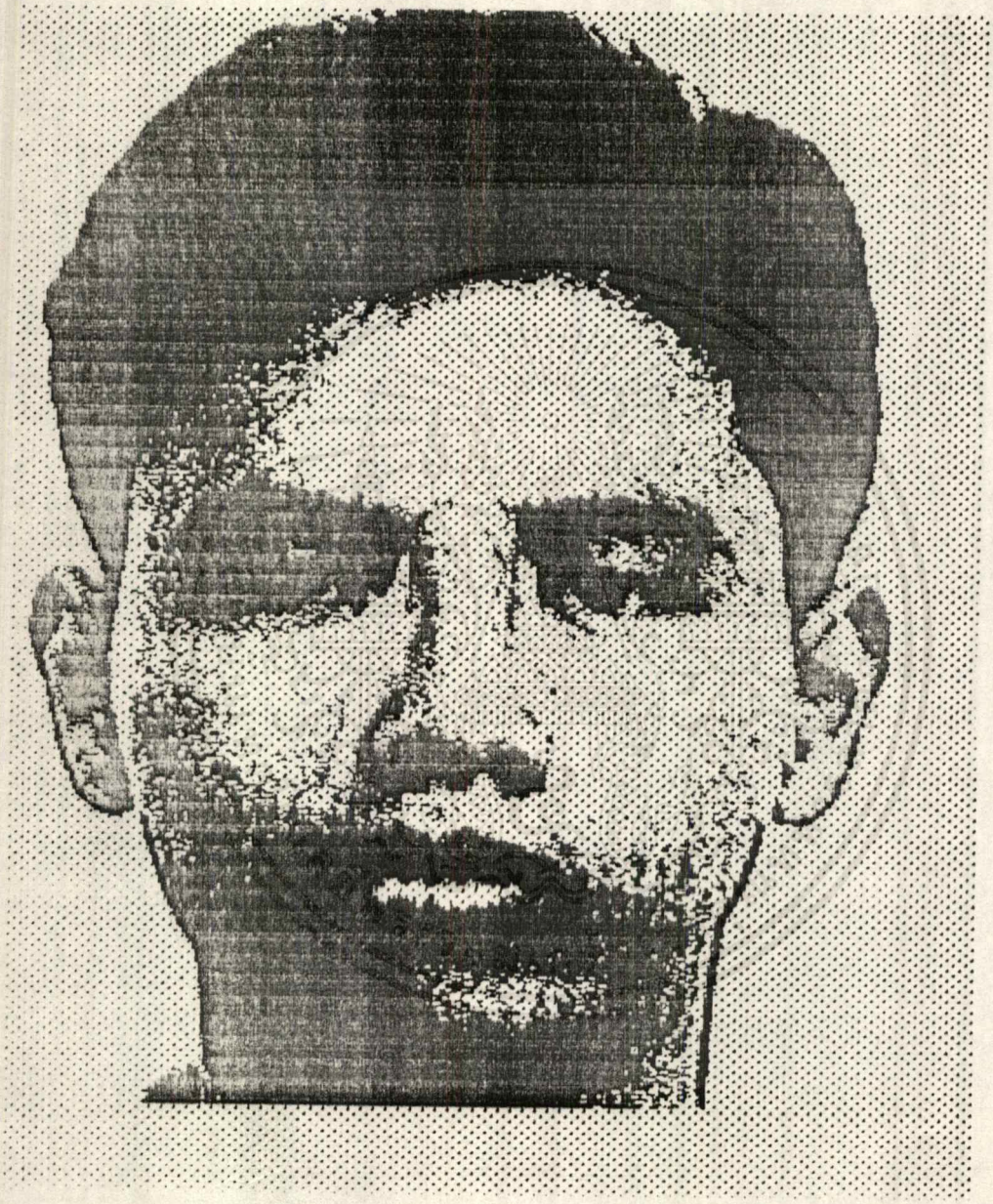
นอกจากนี้ โปรแกรมสามารถทำการอ่านภาพบิทแมพทั่วไปที่มีรูปแบบ BMP ซึ่งจะทำการ ดีซเซอร์ ภาพนั้น ๆ ถ้าจำนวนสีของระบบมีไม่พอกับสีของภาพ รวมถึงการทำงานผ่านคลิปบอร์ดเพื่อนำไปใช้งานกับแอพพลิเคชันอื่น ๆ



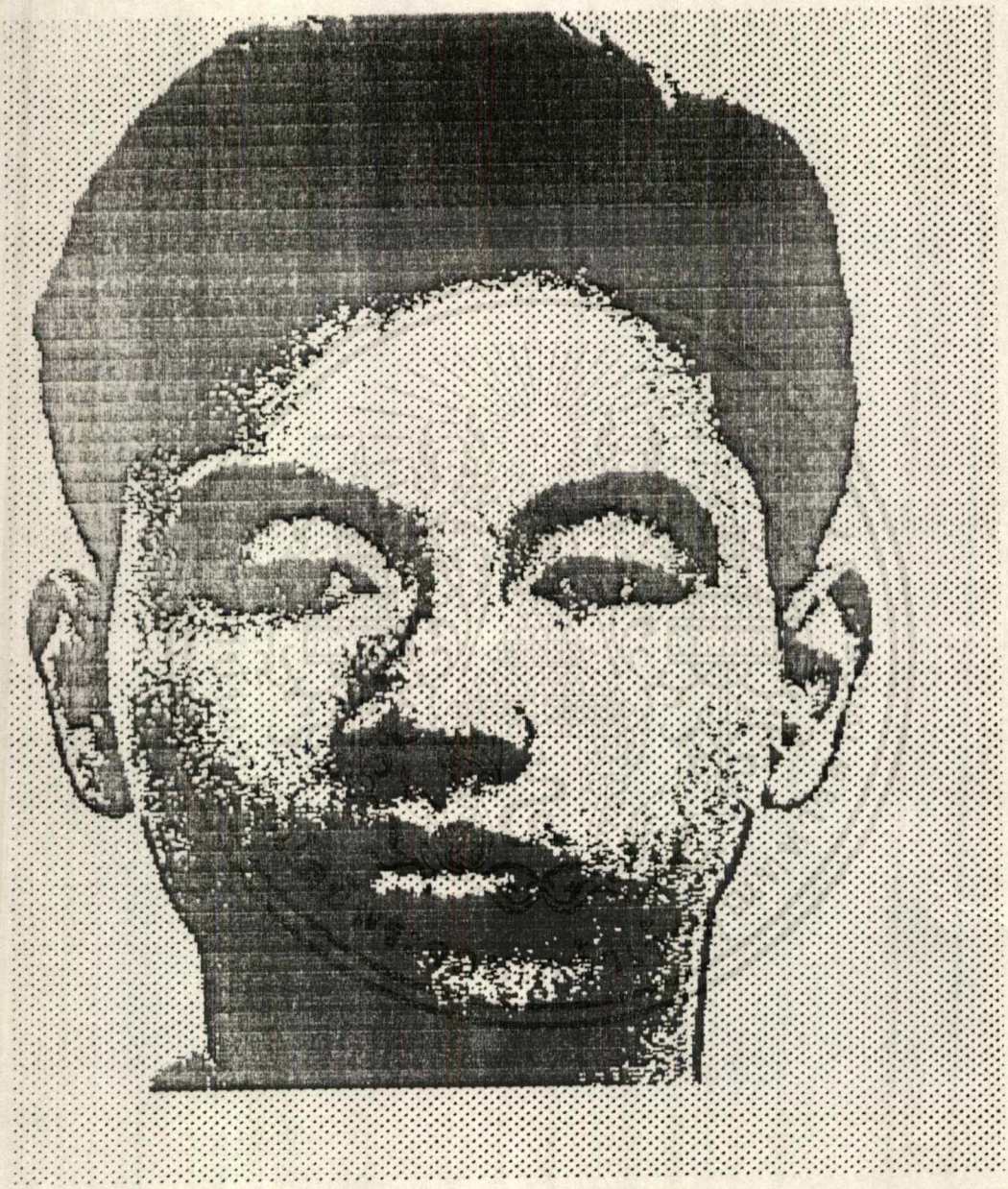
ภาพต้นแบบ



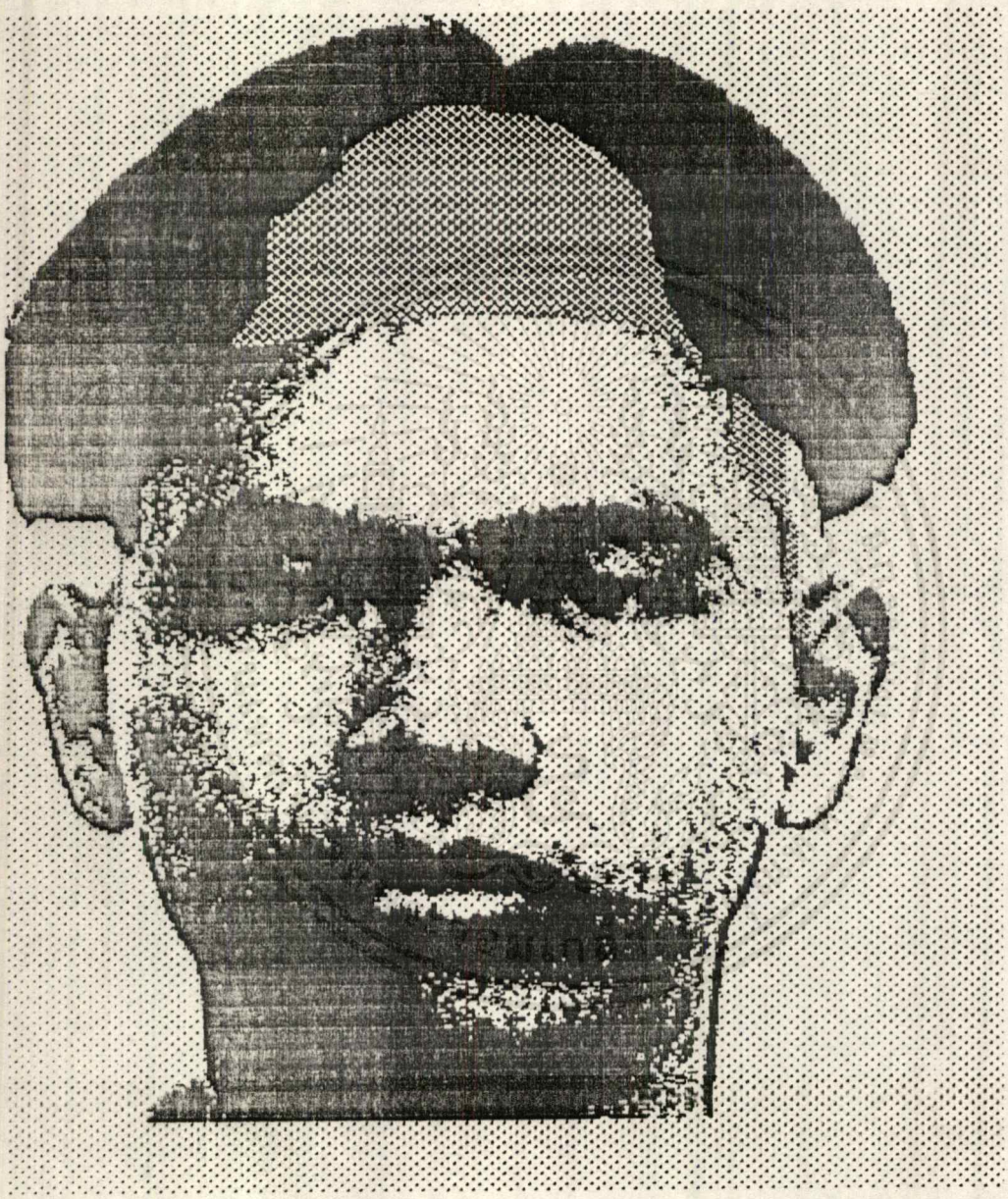
ภาพต้นแบบ หลังการเปลี่ยนสีผม



ภาพต้นแบบ หลังการเปลี่ยนส่วนประกอบ คือ จมูก



ภาพต้นแบบ หลังการเปลี่ยนส่วนประกอบ คือ ตา



ภาพต้นแบบ หลังการเปลี่ยนทรงผม

6.1 สรุปผลและวิจารณ์การทดลอง

สำหรับการเขียนโปรแกรมแอพพลิเคชัน ที่ทำการแต่งภาพใบหน้าบนวินโดวส์ มีปัญหาหลักที่พบอยู่ 2 ส่วนใหญ่ ๆ คือ ส่วนการใช้สี และ ส่วนของการแต่งภาพ โดยจะแยกพิจารณาดังนี้

ปัญหาส่วนการใช้สี

1. การเลือกสีให้เหมาะสม เช่น สีของทรงผม
2. การค้นหาส่วนที่ต้องการโดยใช้การเปรียบเทียบสี
3. การที่รูปภาพที่เหมือนจริงนั้น มีสีอยู่มากมาย แม้จะเป็นบริเวณเดียวกัน เช่น จุดที่ใกล้เคียงกันบนพื้นหน้านั้นยังมีค่า RGB ต่างกัน
4. เนื่องจากภาพที่ประกอบขึ้นใหม่นั้น บางภาพประกอบกันได้ไม่พอดี จึงมีการเลือกสีพื้นที่เหมาะสมเติมแต่งภาพที่ขาดไปบางส่วน ดังนั้นถ้าเลือกสีได้ไม่ใกล้เคียงแล้วทำให้ภาพดูไม่เหมือนจริง

ปัญหาส่วนการแต่งภาพ

1. การเคลื่อนย้ายส่วนประกอบต่าง ๆ ของใบหน้านั้น จะทำให้พอดีกับภาพใหม่ได้ยาก เนื่องจากต้องทำการจับภาพมาให้ได้อัตราส่วนเดียวกัน และ คนทุกคนมีขนาดของส่วนประกอบใบหน้าที่ไม่เท่ากันด้วย

ดังนั้น ความสมบูรณ์ของโครงงานนี้จึงขึ้นอยู่กับความสมบูรณ์การแก้ไขปัญหา 2 ข้อนี้

ภาคผนวก ก

โปรแกรมภาษา C

ไฟล์ HEADER

PROOOO.H

```
#if !defined(zRCFILE)
  LONG FAR PASCAL zWndProc(HWND,unsigned,WORD,LONG);
  int PASCAL WinMain(HANDLE,HANDLE,LPSTR,int);
  HWND zInitMainWindow(HANDLE);
  BOOL zInitClass(HANDLE);
static void zResetFunc(HWND);
static void zSystemMessage(void);
static void zPasteToClipboard(HWND);
static void zPasteFromClipboard(HWND);
static void zDisplayFace(HWND,int);
static void zDisplayBaseFace(HWND,int);
static void zDisplayHair(HWND,int);
static void zSelectFaceName(char *FaceName[5],int);
static void zSelectHairName(char *HairName[5],int);
static void zPaintHair(HWND,int,int);
static void zPasteToFace(HWND);
static void zOpenFile(HWND,int);
#endif


#define IDM_Open            1
#define IDM_Save            2
#define IDM_Print           3
#define IDM_GetInfo         4
#define IDM_Exit            5
#define IDM_Copy            6
#define IDM_Paste           7
```

```
#define IDM_Face                8
#define IDM_Hair                9
#define IDM_Eye                 10
#define IDM_Nose                11
#define IDM_Mouth               12

#define IDM_HairColor           13
#define IDM_Black               14
#define IDM_DarkBrown           15
#define IDM_Brown               16
#define IDM_LightBrown          17
#define IDM_RedBrown            18

#define IDM_Go                  30

#define IDM_FMGo                31

#define IDM_HairMode            34
#define IDM_FaceMode            35

#define IDM_About               36
#define IDM_License             37
#define IDM_GeneralHelp         38

#define Black           1
#define DarkBrown       2
#define Brown           3
#define LightBrown      4
#define RedBrown        5
#define IDS_Caption             1
```

```
#define IDS_Source              2
#define IDS_SignOn              3
#define IDS_Warning             4
#define IDS_NoMouse             5
#define IDS_About               6
#define IDS_AboutText           7
#define IDS_License             8
#define IDS_LicenseText         9
#define IDS_Help                10
#define IDS_HelpText            11
#define IDS_Completed           12
#define IDS_ClpbdText           13
#define IDS_Error               14
#define IDS_ClpbdError          15
#define IDS_System              16
#define IDS_SystemText          17
#define IDS_Width               18
#define IDS_Depth               19
#define IDS_ClpbdHandle         20
#define IDS_WD                  21
```

ไฟล์ RESOURCE
PROOOO.RC

```
#define zRCFILE
#include <d:\tcwin\include\WINDOWS.H>
#include "PROOOO.H"

PROOOO MENU
  BEGIN
  POPUP "&File"
    BEGIN
      MENUITEM  "&Open...", IDM_Open
      MENUITEM  "&Save", IDM_Save
      MENUITEM  "&Print..", IDM_Print
      MENUITEM  SEPARATOR
      MENUITEM  "Get Info", IDM_GetInfo
      MENUITEM  SEPARATOR
      MENUITEM  "E&xit", IDM_Exit
    END
  POPUP "&Edit"
    BEGIN
      MENUITEM "&Copy", IDM_Copy
      MENUITEM "&Paste", IDM_Paste
    END
  POPUP "&Change"
    BEGIN
      MENUITEM  "Fa&ce...", IDM_Face
      MENUITEM  "&Hair...", IDM_Hair
```

```
    MENUITEM  "&Eye... ", IDM_Eye, GRAYED
    MENUITEM  "&Nose.. ", IDM_Nose, GRAYED
    MENUITEM  "&Mouth..", IDM_Mouth, GRAYED \
  END
POPUP "&HairDone"
  BEGIN
    POPUP "&HairColor"
     BEGIN
MENUITEM "Black..", IDM_Black
MENUITEM "DarkBrown...", IDM_DarkBrown
MENUITEM "Brown...", IDM_Brown
MENUITEM "LightBrown...", IDM_LightBrown
MENUITEM "RedBrown...", IDM_RedBrown
     END
    MENUITEM SEPARATOR
    MENUITEM " &GO.... ", IDM_Go
  END
POPUP "&FaceDone"
  BEGIN
    MENUITEM "&Change..." , IDM_FMGo, GRAYED
  END
POPUP "* &Mode * "
  BEGIN
    MENUITEM "&HairMode", IDM_HairMode, CHECKED
    MENUITEM SEPARATOR
    MENUITEM "&FaceMode", IDM_FaceMode
  END
POPUP "\a&Help"
  BEGIN
    MENUITEM "&About...", IDM_About
```

```
        MENUITEM "&Writer...", IDM_License
        MENUITEM SEPARATOR
        MENUITEM  "&Help...", IDM_GeneralHelp
      END
    END

 STRINGTABLE
   BEGIN
      IDS_Caption      "PROGRAM MAKEUP"
      IDS_Source       "PR0000.C"
      IDS_SignOn       "Here-is-how-it's-done capability to process
 hair cut and hair color"
      IDS_Warning      "This program want more developement"
      IDS_NoMouse      "No mouse found.  Some features of this
 demonstration program may require a mouse."
      IDS_About        "About this application process with hair
 and face"
      IDS_AboutText    "This demonstration program provides face
 and hair
 functions for applications running under Windows.  "
      IDS_License      "This is Project II of Image Application"
      IDS_LicenseText "This is a Project of Tarakorn and Bunterng"
      IDS_Help         "General Help:  PR0000.C"
      IDS_HelpText     "Select non-grayed items from the menu to
 work with various hair and face functions.  Use Paste from the
 Edit menu to import an image from the Clipboard. "
      IDS_Completed    "Task completed OK"
      IDS_ClpbdText    "The image was copied successfully to the
 Clipboard."
      IDS_Width        "Incoming image is wider than current window.
```

```
  Resize image? [Yes: resize.] [No: right edge will be clipped.]"
     IDS_Depth      "Incoming image is longer than current window.
  Resize image? [Yes: resize.] [No: bottom edge will be clipped.]".
     IDS_WD         "Image is winder and longer than current window.
  Resize image? [Yes: resize.] [No: right and bottom edges will be
 clipped.]"
     IDS_Error      "Runtime error"
     IDS_ClpbdHandle "An error occurred while attempting to grab
 the handle of the bitmap in the Clipboard.  No data was pasted
 from the Clipboard."
     IDS_ClpbdError  "An error occurred while attempting to
 create a compatible bitmap.No data was copied to the clipboard."
     IDS_System      "System Commands"
     IDS_SystemText  "You attempted to resize, move, maximize, or
 minimize the window.This program filters and disable system
 commands."
   END

FileOpen DIALOG 18, 4, 130, 145
CAPTION "FileOpen"
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
     LTEXT "Open File:", -1, 3, 3, 35, 10
     EDITTEXT 101, 3, 15, 100, 12, ES_AUTOHSCROLL
     LTEXT "File in", -1, 3, 33, 25, 10
     LTEXT "", 102, 34, 33, 90, 12
     CONTROL "", 103, "LISTBOX", LBS_STANDARD, 4, 47, 70, 75
     CONTROL "Cancel", IDCANCEL, "BUTTON", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_GROUP, 91, 126, 34, 14
     CONTROL "Open", IDOK, "BUTTON", BS_PUSHBUTTON | WS_CHILD |
```

```
WS_VISIBLE | WS_GROUP, 53, 126, 34, 14
     ICON "View", -1, 0, 129, 16, 16, WS_CHILD | WS_VISIBLE
END

InfoBox DIALOG 18, 20, 187, 103
CAPTION "File Infomation"
STYLE DS_MODALFRAME | WS_POPUP | WS_DLGFRAME
BEGIN
     RTEXT "Name:", -1, 9, 9, 53, 8, SS_RIGHT | WS_CHILD |
WS_VISIBLE | WS_GROUP
     RTEXT "Dimemsions:", -1, 9, 21, 53, 8, SS_RIGHT |
WS_CHILD | WS_VISIBLE | WS_GROUP
     RTEXT "Colours:", -1, 9, 33, 53, 8, SS_RIGHT | WS_CHILD |
WS_VISIBLE | WS_GROUP
     RTEXT "Packed size:", -1, 9, 45, 53, 8, SS_RIGHT | WS_CHILD
 | WS_VISIBLE | WS_GROUP
     RTEXT "Unpacked size:", -1, 9, 57, 53, 8, SS_RIGHT | WS_CHILD
| WS_VISIBLE | WS_GROUP
     RTEXT "Comments:", -1, 9, 69, 53, 8, SS_RIGHT | WS_CHILD |
WS_VISIBLE | WS_GROUP
     LTEXT "", 101, 73, 9, 84, 8, WS_CHILD| WS_VISIBLE| WS_GROUP
     LTEXT "", 102, 73, 21, 84, 8, WS_CHILD| WS_VISIBLE| WS_GROUP
     LTEXT "", 103, 73, 33, 109, 8, WS_CHILD| WS_VISIBLE| WS_GROUP
     LTEXT "", 104, 73, 45, 109, 8, WS_CHILD| WS_VISIBLE| WS_GROUP
     LTEXT "", 105, 73, 57, 109, 8, WS_CHILD| WS_VISIBLE| WS_GROUP
     LTEXT "None", 106, 73, 69, 109, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
     ICON "View", -1, 0, 88, 16, 16, WS_CHILD | WS_VISIBLE
     PUSHBUTTON "Ok", IDOK, 157, 83, 24, 14, WS_CHILD | WS_VISIBLE
 | WS_TABSTOP
```

```
END

NameBox DIALOG 100, 56, 120, 58
CAPTION "File name"
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
     LTEXT "Save to:", -1, 12, 8, 63, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
     CONTROL "", 101, "EDIT", ES_LEFT |ES_UPPERCASE | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_TABSTOP, 12, 16, 95, 12
     PUSHBUTTON "Ok", IDOK, 62, 33, 38, 14, WS_CHILD | WS_VISIBLE
 | WS_TABSTOP
     PUSHBUTTON "Cancel", IDCANCEL, 18, 33, 38, 14, WS_CHILD |
WS_VISIBLE | WS_TABSTOP
     ICON "View", -1, -1, 42, 16, 16, WS_CHILD | WS_VISIBLE
END


View ICON
BEGIN
'00 00 01 00 01 00 20 10 10 00 00 00 00 00 A8 01'
'00 00 16 00 00 00 28 00 00 00 20 00 00 00 20 00'
'00 00 01 00 04 00 00 00 00 00 40 01 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
'00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00 80 80'
'80 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
'00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 FF FF'
'FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
'FF FF FF FF CC CC CF FF FF FF FF FF FF FF FF FF'
'FF FF FF CC CF FF CC CF FF FF FF FF FF FF FF FF'
```

```
 'FF FF CC CF FF FF FF CC CF FF FF FF FF FF FF'
 'FF FC CF FF FF FF FF FC CC FF FF FF FF FF FF'
 'FF FC FF FF FF FF F9 FF FC CF FF FF FF FF FF'
 'FF FF FF FC CC FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FC FF FF F9 F9 9F 9F FF FF FF FF FF'
 'FF FF FF FC CC FF FF 9F FF 9F FF FF FF FF FF'
 'FF FF FF FF FC FF FF FF FF FF FF FF FF FF FF'
 'FF CC CF FF FC FF FF FF CC CF FF FF FF FF F0 FF'
 'FF CF CF FF FC FF FF FC CF CF FF 0F FF FF F0 00'
 'FF CC CF FF FF FF FF FC FC CF FF 0F FF FF FF F0'
 '0F CC FF FF FF FF FF FC CC FF F0 F0 FF FF FF FF'
 '00 F0 00 0F 00 00 00 0F FF F0 0F F0 FF FF FF FF'
 'F0 00 FF F0 0F FF FF F0 00 00 FF F0 FF FF 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00'
END

PROOOO ICON
BEGIN
 '00 00 01 00 01 00 20 20 10 00 00 00 00 00 E8 02'
 '00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00'
 '00 00 01 00 04 00 00 00 00 00 00 02 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
 '00 00 80 00 80 00 80 80 00 00 80 80 80 00 C0 C0'
 'C0 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
 '00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 99 99'
 '99 99 99 99 99 99 99 99 99 99 99 99 99 99 BB 0B'
```

'BB BB BB BB BB BB BB BB BB BB BB BB BB B9 99'

'99 99 9C CC 99 99 99 99 99 99 99 99 9B B9 9C'

'CC 99 CC CC C9 99 99 99 99 99 99 99 9B B9 CC'

'CC C9 CC CC C9 99 99 99 99 99 99 99 9B B9 CC'

'CC C9 9C CC 99 99 99 99 99 99 99 99 9B B9 9C'

'CC CC C9 C9 99 99 99 99 99 99 99 99 9B B9 99'

'99 99 EC C9 99 99 99 99 99 99 99 99 9B B9 99'

'99 99 9C CC 99 99 99 99 99 99 99 99 9B B9 99'

'99 99 99 CC 99 99 99 99 99 99 99 99 9B B8 99'

'99 99 99 9C CC 99 99 99 99 99 99 99 9B B8 99'

'99 99 99 99 CC C9 99 99 99 99 99 99 9B B8 00'

'99 99 99 99 9C CC 99 99 99 99 99 99 9B B0 08'

'99 99 99 99 9C CC C9 99 99 99 99 99 9B B0 00'

'89 99 99 99 9C CC CC 99 99 99 99 99 9B B0 00'

'00 99 99 99 9C C9 CC C9 99 99 99 99 9B B9 00'

'00 89 99 99 9C CC 9C CC 99 99 99 99 9B B9 90'

'00 08 99 99 98 CC 89 CC C9 99 99 99 9B B9 90'

'00 00 09 99 8C CC C9 9C CC 99 99 99 9B B9 99'

'00 00 00 98 99 CC C9 89 CC C9 99 99 9B B9 99'

'99 00 00 09 98 9C CC 99 9C C9 99 99 9B B9 99'

'99 00 00 00 89 9C CC 98 9C CC 99 99 9B B9 99'

'99 99 00 00 99 98 CC C9 98 CC C9 99 9B B9 99'

'99 99 90 00 09 89 CC C9 89 8C CC 99 9B B9 99'

'99 99 90 00 00 98 9C CC 99 98 CC 99 9B B9 99'

'99 99 99 90 00 00 9C CC 99 89 9C 99 9B B9 99'

'99 99 99 90 00 00 08 9C C8 99 89 99 9B B9 99'

'99 99 99 99 00 00 00 9C C9 98 99 99 9B B9 99'

'99 99 99 99 99 00 00 08 99 89 99 99 9B B9 99'

'99 99 99 99 99 90 00 09 99 89 99 99 9B B9 99'

'99 99 99 99 99 90 00 00 00 99 99 99 9B BB BB'

```
'BB BB BB BB BB BB BB BB BB BB BB BB BB BB 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
'00 00 00 00 00 00 00 00 00 00 00 00 00 00 38 00'
'00 00 60 00 00 00 70 00 00 00 7C 00 00 00 3C 00'
'00 00 1E 00 00 00 1F 00 00 00 0F C0 00 00 03 E0'
'00 00 03 F0 00 00 00 F0 00 00 00 78 00 00 00 7C'
'00 00 00 1F 00 00 00 1F 80 00 00 0F C0 00 00 03'
'E0 00 00 01 E0 00 00 01 FC 00 00 00 00 00'
END
```

ไฟล์หลักภาษา C

PROOOO.CPP

```
/*
Include files
*/

#include <WINDOWS.H>
#include <io.h>
#include <dir.h>
#include <stdio.h>
#include <alloc.h>
#include "PROOOO.H"
#include "view.h"

typedef struct {
int red_ref;
int green_ref;
int blue_ref;
} RGB_REF;


BOOL FAR PASCAL FileOpenProc(HWND hWnd,WORD message,WORD wParam,
LONG lParam);
BOOL FAR PASCAL ShowInfoProc(HWND hWnd,WORD message,WORD wParam,
LONG lParam);
BOOL FAR PASCAL GetNameProc(HWND hWnd,WORD message,WORD wParam,
LONG lParam);

int ShowFrame(HWND hWnd,HDC hdc,char huge *p,int x,int y,
```

```
char *palette,int bits);
int DitherPicture(FILEINFO *fi);
int FileOpenDialog(HANDLE hInst,HWND hWnd,char *name,POFSTRUCT
pofIn,int DialogStatus);
int GetFileName(HWND hWnd,char *b);
int putline(char huge *p,unsigned int n);
int getline(char huge *p,unsigned int n);
int getbuffer(FILEINFO *fi);

void freebuffer(void);
void ShowInfo(HWND hWnd,FILEINFO *fi);
void ReadNotBk(HWND hWnd, int x1, int y1, int x2, int y2);
void FMDisplay(HWND hWnd, int Mode);

static RGB_REF FindRGB(int Color);

/*
Static variables visible throughout this file
*/

HANDLE hInst;
HWND MainhWnd;          // Main windows handle
HANDLE hAccel;
HCURSOR hPrevCursor;
HCURSOR hHourGlass;
int MessageRet;
int MousePresent;
char lpCaption[51];
int Max= 50;
char lpMessage[250];
```

```
int MaxText= 249;
DWORD ActualLtGray= 0;
DWORD ActualDKGray= 0;
short ImageWidth= 0;
short ImageDepth= 0;
int FaceNo=1;   // Index to call face
int HairNo=1;           //      ..      hair
int EyeNo=1;
int NoseNo=1;
int MaxFaceFile=2; // number of face file
int MaxHairFile=2;      // number of hair file
int MaxEyeFile=2;
int MaxNoseFile=2;
int xBeginFace=0;
int yBeginFace=0;
int xFaceWidth=320;
int yFaceWidth=470;
int xBeginHair=320;
int yBeginHair=0;
int xHairWidth=319;
int yHairWidth=470;
int FirstHairColor=Black;
int OldHairColor=Black;
int HairColor=Black;
int HairMode=0;// mode operation
int FaceMode=1;
int Mode=0;             // begin with hair
int DIALOG_STATUS;
int DialogOff=0;
int DialogOn=1;
```

```
int AllScreen=0;     // 3 status to select picture to display
int LeftScreen=1;
int RightScreen=2;
int DisplayStatus;// status to display
int Eye=0;
int Nose=1;
static FILEINFO fi;
static char szFileName[129];
static char szFileSpec[129]="";
static int SortRGBAxis;

char masktable[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
char bittable[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};

GLOBALHANDLE details=NULL;
GLOBALHANDLE imagehandle=NULL;

static POFSTRUCT pof;
OFSTRUCT of;

unsigned int linewidth,imagebits,pagedepth;

extern char szAppName[];
extern char szFileExt[];

FARPROC lpfnDlgProc;
HDC hdc;
PAINTSTRUCT ps;
RECT rect;
HBRUSH hBrush;
```

```
HPEN hPen;
HMENU hMenu;
char far *p,b[STRINGSIZE+1];
static int vpos,hpos;
int i,vsize,hsize,vjump,hjump;
/*
Entry point for the application
*/

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstane,
LPSTR lpCmdLine, int nCmdshow)
{
  MSG msg;
  HWND hWndPrev;

hWndPrev = FindWindow("PROOOO", NULL);
if (hWndPrev != NULL)
  {
  BringWindowToTop(hWndPrev);
  return 0;
  }


hInst = hInstance;
if (!zInitClass(hInstance)) return FALSE;
MainhWnd = zInitMainWindow(hInstance);
if (!MainhWnd) return FALSE;
ShowWindow(MainhWnd, nCmdshow);
UpdateWindow(MainhWnd);
```

```
zDisplayFace(MainhWnd,FaceNo); // display face in first window
zDisplayHair(MainhWnd,HairNo); //    ..    hair    ..

hAccel = LoadAccelerators(hInstance,"PROOOO");
MousePresent = GetSystemMetrics(SM_MOUSEPRESENT);
if (!MousePresent)
  {
  LoadString(hInst,IDS_Warning,lpCaption,Max);
  LoadString(hInst,IDS_NoMouse,lpMessage,MaxText);
  MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
  }


while (GetMessage(&msg,0,0,0))
  {
  if(TranslateAccelerator(MainhWnd, hAccel, &msg))
 continue;
  TranslateMessage(&msg);
  DispatchMessage(&msg);
  }
return(msg.wParam);
}

/*
Switcher for incoming message
*/

LONG FAR PASCAL zWndProc(HWND hWnd, unsigned message,
 WORD wParam, LONG lParam)
{
```

```
HMENU  hMenu;

hMenu=GetMenu(hWnd);
switch (message)
  {
  case WM_COMMAND:
 switch(wParam)
{
case IDM_Open:
   DisplayStatus=AllScreen;
   zOpenFile(hWnd,DialogOn);
   break;      // break FILE_OPEN
case IDM_Save:
        if(GetFileName(hWnd,b)==IDOK)  {
     hHourGlass=LoadCursor(NULL,IDC_WAIT);
     hPrevCursor=SetCursor(hHourGlass);
     fi.getline=getline;
     if(WriteFile(&fi,b) != SUCCESS)  {
  remove(b);
  SetCursor(hPrevCursor);
  MessageBox(hWnd ,"Error writing file",ERRORSTRING,MB_OK |
 MB_ICONSTOP);
     }  else SetCursor(hPrevCursor);
        }
        return(0);
 case IDM_Print:break;
 case IDM_GetInfo:
        pof=&of;
        if(FileOpenDialog(hInst,hWnd,szFileName,pof,DialogOn)) {
}
```

```
    fi.putline=putline;
    hHourGlass=LoadCursor(NULL,IDC_WAIT);
    hPrevCursor=SetCursor(hHourGlass);
    if(GetInfo(&fi,szFileName) == SUCCESS)  {
SetCursor(hPrevCursor);
ShowInfo(hWnd,&fi);
    }
    else  {
SetCursor(hPrevCursor);
MessageBox(hWnd,"Error getting information",
      ERRORSTRING,MB_OK | MB_ICONSTOP);
    }
       }
break;

case IDM_Exit:PostQuitMessage(0);
break;
case IDM_Copy:zPasteToClipboard(hWnd);
break;
case IDM_Paste:zPasteFromClipboard(hWnd);
break;
case IDM_Face:          DisplayStatus=LeftScreen;
zDisplayFace(hWnd,FaceNo);
break;
case IDM_Hair:                      if(Mode==HairMode)  {
  HairColor=Black;
  zDisplayHair(hWnd,HairNo);
}
break;
case IDM_Eye:if(Mode==FaceMode)
```

```
    FMDisplay(hWnd,Eye);
  break;
  case IDM_Nose:if(Mode==FaceMode)
    FMDisplay(hWnd,Nose);
  break;

  case IDM_Black:                    if(Mode==HairMode)
    zPaintHair(hWnd,Black,OldHairColor);
  break;
  case IDM_DarkBrown:                if(Mode==HairMode)
    zPaintHair(hWnd,DarkBrown,OldHairColor);
  break;
  case IDM_Brown:                    if(Mode==HairMode)
    zPaintHair(hWnd,Brown,OldHairColor);
  break;
  case IDM_LightBrown:               if(Mode==HairMode)
    zPaintHair(hWnd,LightBrown,OldHairColor);
  break;
  case IDM_RedBrown:                 if(Mode==HairMode)
    zPaintHair(hWnd,RedBrown,OldHairColor);
  break;
  case IDM_Go:                       if(Mode==HairMode)
    zPasteToFace(hWnd);
  break;
  case IDM_FaceMode:Mode=FaceMode;
  FMDisplay(hWnd,Eye);
  if(GetMenuState(hMenu,IDM_HairMode,MF_CHECKED)) {
   CheckMenuItem(hMenu,IDM_HairMode,MF_UNCHECKED);
   CheckMenuItem(hMenu,IDM_FaceMode,MF_CHECKED);
   EnableMenuItem(hMenu,IDM_Hair,MF_GRAYED);
```

```
  EnableMenuItem(hMenu,IDM_HairColor,MF_GRAYED);
  EnableMenuItem(hMenu,IDM_Go,MF_GRAYED);
  EnableMenuItem(hMenu,IDM_FMGo,MF_ENABLED);
  EnableMenuItem(hMenu,IDM_Nose,MF_ENABLED);
  EnableMenuItem(hMenu,IDM_Eye,MF_ENABLED);
 }
 break;
 case IDM_HairMode:Mode=HairMode;
 HairNo--;
 zDisplayHair(hWnd,FaceNo--);
 zPaintHair(hWnd,HairColor,OldHairColor);
 if(GetMenuState(hMenu,IDM_FaceMode,MF_CHECKED)) {
  CheckMenuItem(hMenu,IDM_FaceMode,MF_UNCHECKED);
  CheckMenuItem(hMenu,IDM_HairMode,MF_CHECKED);
  EnableMenuItem(hMenu,IDM_Eye,MF_GRAYED);
  EnableMenuItem(hMenu,IDM_Nose,MF_GRAYED);
  EnableMenuItem(hMenu,IDM_FMGo,MF_GRAYED);
  EnableMenuItem(hMenu,IDM_HairColor,MF_ENABLED);
  EnableMenuItem(hMenu,IDM_Go,MF_ENABLED);
  EnableMenuItem(hMenu,IDM_Hair,MF_ENABLED);
 }
 if(HairNo==0)
    HairNo=MaxHairFile;
 break;
 case IDM_About:
   LoadString(hInst,IDS_About,lpCaption,Max);
   LoadString(hInst,IDS_AboutText,lpMessage,MaxText);
   MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
   break;
 case IDM_License:
```

```
    LoadString(hInst,IDS_License,lpCaption,Max);
    LoadString(hInst,IDS_LicenseText,lpMessage,MaxText);
    MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
    break;
  case IDM_GeneralHelp:
    LoadString(hInst,IDS_Help,lpCaption,Max);
    LoadString(hInst,IDS_HelpText,lpMessage,MaxText);
    MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
    break;
  default:
    return(DefWindowProc(hWnd, message, wParam, lParam));
    }
    break;

  case WM_INITMENUPOPUP:
    if (lParam == 1)
   {
   }
  break;

  case WM_KEYDOWN:
    switch (wParam)
   {
   case VK_RETURN:break;
   case VK_ESCAPE:break;
   case VK_PRIOR:break;
   case VK_NEXT: break;
   case VK_HOME:   break;
   case VK_LEFT:break;
   case VK_UP:break;
```

```
    case VK_DOWN: break;
    case VK_RIGHT:   break;
    case VK_END:break;
    case VK_TAB:break;
    default:
        return(DefWindowProc(hWnd, message, wParam, lParam));
    }
     break;


  case WM_PAINT:
    hdc=BeginPaint(hWnd,&ps);
            if(imagehandle==NULL)  {
          GetClientRect(hWnd,&rect);
                hBrush=GetStockObject(LTGRAY_BRUSH);
                SelectObject(hdc,hBrush);
          hPen=GetStockObject(NULL_PEN);
                SelectObject(hdc,hPen);
                Rectangle(hdc,rect.left,rect.top,rect.right,rect.
bottom);
            }
    else  {
                if((p=GlobalLock(imagehandle)) != NULL)  {
      if(!ShowFrame(hWnd,hdc,p,hpos,vpos,fi.palette,imagebits))
   MessageBox(hWnd,"Error allocating memory",ERRORSTRING,MB_OK |
MB_ICONSTOP);
      GlobalUnlock(imagehandle);
         }  else  MessageBox(hWnd,"Error locking memory",
ERRORSTRING,MB_OK | MB_ICONSTOP);
            }
```

```
  EndPaint(hWnd,&ps);
  return(0);
        case WM_DESTROY:
          freebuffer();
          PostQuitMessage(0);
          break;          // break  WM_DESTROY
case WM_SYSCOMMAND:
  if ((wParam & 0xfff0)== 'SC_MOVE)
 {
     zSystemMessage(); break;
 }
  if ((wParam & 0xfff0)== SC_SIZE)
 {
     zSystemMessage(); break;
 }
  if ((wParam & 0xfff0)== SC_MINIMIZE)
 {
     zSystemMessage(); break;
 }
  if ((wParam & 0xfff0)== SC_MAXIMIZE)
 {
     zSystemMessage(); break;
 }

  default:
 return(DefWindowProc(hWnd, message, wParam, lParam));
  }
return FALSE;
}
```

```
/*
Initialize the attributes od the window class
*/

BOOL zInitClass(HANDLE hInstance)
{
  WNDCLASS WndClass;
WndClass.style= 0;
WndClass.lpfnWndProc= zWndProc;
WndClass.cbClsExtra= 0;
WndClass.cbWndExtra= 0;
WndClass.hInstance= hInstance;
WndClass.hIcon= LoadIcon(NULL,IDI_EXCLAMATION);
WndClass.hCursor= LoadCursor(NULL,IDC_ARROW);
WndClass.hbrBackground= CreateSolidBrush(RGB(255,255,255));
WndClass.lpszMenuName= "PR0000";
WndClass.lpszClassName= "PR0000";
return RegisterClass(&WndClass);
}

/*
Create the main window
*/

HWND zInitMainWindow(HANDLE hInstance)
{
  HWND hWnd;
LoadString(hInstance,IDS_Caption,lpCaption,Max);
hHourGlass= LoadCursor(NULL,IDC_WAIT);
hWnd = CreateWindow(
```

```
  "PROOOO",
  lpCaption,
  WS_OVERLAPPED | WS_THICKFRAME | WS_SYSMENU | WS_MINIMIZEBOX |
     WS_MAXIMIZEBOX | WS_CLIPCHILDREN,
  0,0,
  639,470,
  0, 0,
  hInstance, (LPSTR)NULL);
return hWnd;
}


/*
THE CORE FUNCTION OF THE APPLICATION
*/


LPSTR lstrchr(LPSTR s,char c)
{
     while(*s)  {
          if(c==*s)  return(s);
          s=AnsiNext(s);
     }
     return(NULL);
}

LPSTR lstrrchr(LPSTR s,char c)
{
     LPSTR     st;
```

```
    st=s+lstrlen(s);

    do  {
        if(c==*st)  return(st);
        st=(AnsiPrev(s,st));
    }  while(st > s);
}
/* do memcmp */
int lmemcmp(LPSTR s1,LPSTR s2,int n)
{
    while(n--)  if(*s1++ != *s2++)  return(1);
    return (0);
}

/* do memcpy */
void lmemcpy(LPSTR dest,LPSTR source,int n)
{
    while(n--)  *dest++=*source++;
}

/* do memset */
void lmemset(LPSTR dest,int c,int n)
{
    while(n--)  *dest++=c;
}

/* handle message to the Getinfo display dialog */
BOOL FAR PASCAL ShowInfoProc(HWND hWnd,WORD message,WORD wParam,
LONG lParam)
{
```

```
        char b[64];
        switch(message)  {
             case WM_INITDIALOG:
          SetDlgItemText(hWnd,INFO_NAME,fi.name);

                  sprintf(b,"%u x %u",fi.width,fi.depth);
          SetDlgItemText(hWnd,INFO_DIMENSIONS,b);
                  if(fi.bits <= 8)  sprintf(b,"%u",1<<fi.bits);
                  else  lstrcpy(b,"Infinite");
          SetDlgItemText(hWnd,INFO_COLOURS,b);

                  sprintf(b,"%lu bytes",fi.imagesize);
          SetDlgItemText(hWnd,INFO_FILESIZE,b);

                  sprintf(b,"%lu bytes",fi.imagesize);
          SetDlgItemText(hWnd,INFO_IMAGESIZE,b);

          SetDlgItemText(hWnd,INFO_COMMENTS,fi.comments);

                  return(TRUE);
             case WM_COMMAND:
                  switch(wParam)  {
    case IDOK:
 EndDialog(hWnd,TRUE);
                          return(TRUE);
                  }

             }
             return(FALSE);
}
```

```
void pinsert(char far * far *array,int number)
{
}

void psort(char far * far *array,int number)
{
}

BOOL FAR PASCAL GetNameProc(HWND hWnd,WORD message,WORD wParam,
LONG lParam)
{
     HWND dlgH;
     char b[145];
     int  r;

     switch(message)  {
          case WM_INITDIALOG:
               sprintf(b,"UNTITLED%s",szFileExt);
       dlgH=GetDlgItem(hWnd,GETNAME_EDIT);
               SetWindowText(dlgH,b);
               return(TRUE);
          case WM_COMMAND:
               switch(wParam)  {
                    case IDOK:
       dlgH=GetDlgItem(hWnd,GETNAME_EDIT);
       GetWindowText(dlgH,b,STRINGSIZE);
               fnsplit(b,NULL,NULL,szFileName,NULL);
               lstrcat(szFileName,szFileExt);
               if(access(b,0)==0)  {
    r=MessageBox(hWnd,"Overwrite the existing file?","",
```

```
MB_YESNOCANCEL | MB_ICONQUESTION);
                    if(r==IDNO)  return(TRUE);
                    else if(r==IDCANCEL)  {
 EndDialog(hWnd,IDCANCEL);
                         return(TRUE);
                    }
               }

        AnsiLower(szFileName);
               case IDCANCEL:
    EndDialog(hWnd,wParam);
                    return(TRUE);
          }
  break;
     }

     return(FALSE);

}

/* open file function */
static void zOpenFile(HWND hWnd,int DialogStatus)
{

       freebuffer();

      pof=&of;
      hpos=vpos=0;

      if(FileOpenDialog(hInst,hWnd,szFileName,pof,DialogStatus)){
```

```
fi.putline=putline;
if(GetInfo(&fi,szFileName)==SUCCESS)  {
   if(getbuffer(&fi))  {
      hHourGlass=LoadCursor(NULL,IDC_WAIT);
      hPrevCursor=SetCursor(hHourGlass);
      if(ReadFile(&fi,szFileName)==SUCCESS)  {
 hdc=GetDC(hWnd);
 i=(GetDeviceCaps(hdc,PLANES) * GetDeviceCaps(hdc,BITSPIXEL));
 if(i>=16)  i=24;
 ReleaseDC(hWnd,hdc);
 hMenu=GetMenu(hWnd);

 if(fi.bits>1)  {
     if(1)  {
fi.getlino=getline;
if(DitherPicture(&fi)!=SUCCESS)
   MessageBox(hWnd,"Error dithering image",
     ERRORSTRING,MB_OK | MB_ICONSTOP);
      }
      else if(fi.bits>8)  {
i=(1<<i);
if(Quantize(fi.width,fi.depth,&i,fi.palette,getline)
  !=SUCCESS)
    MessageBox(hWnd,"Error quantizing image",
      ERRORSTRING,MB_OK | MB_ICONSTOP);
      }
 }
SetCursor(hPrevCursor);
InvalidateRect(hWnd,NULL,FALSE);
 SendMessage(hWnd,WM_PAINT,0,0L);
```

```
        }
        else  {
   SetCursor(hPrevCursor);
   MessageBox(hWnd,"Error reading file",ERRORSTRING,MB_OK |
MB_ICONSTOP);
   freebuffer();
        }
     }  else MessageBox(hWnd,"Error allocating memory",
ERRORSTRING,MB_OK | MB_ICONSTOP);
 }  else MessageBox(hWnd,"Error getting information",
ERRORSTRING,MB_OK | MB_ICONSTOP);
      }
      GetClientRect(hWnd,&rect);
      vsize=rect.bottom-rect.top;
      if(fi.depth>vsize)
 SetScrollRange(hWnd,SB_VERT,0,fi.depth-vsize,TRUE);
      else
 SetScrollRange(hWnd,SB_VERT,0,1,TRUE);
      hsize=rect.right-rect.left;
      if(fi.width>hsize)
 SetScrollRange(hWnd,SB_HORZ,0,fi.width-hsize,TRUE);
      else
 SetScrollRange(hWnd,SB_HORZ,0,1,TRUE);

}

/* the OpenFile dialog */
int FileOpenDialog(HANDLE hInst,HWND hWnd,char *name,POFSTRUCT
pofIn,int DialogStatus)
{
```

```
     FARPROC dlgProc;
     int r;
     if(DialogStatus==DialogOn)  {
DIALOG STATUS=DialogOn;
lstrcpy(szFileSpec,"*");
lstrcat(szFileSpec,szFileExt);
pof=pofIn;
dlgProc=MakeProcInstance((FARPROC)FileOpenProc,hInst);
r=DialogBox(hInst,"FileOpen",hWnd,dlgProc);
FreeProcInstance(dlgProc);
     }
     else {          // CASE NO DIALOG BOX
DIALOG_STATUS=DialogOff;
pof=pofIn;
lstrcat(szFileSpec,szFileExt);
dlgProc=MakeProcInstance((FARPROC)FileOpenProc,hInst);
r=DialogBox(hInst,"FileOpen",hWnd,dlgProc);
     }
     lstrcpy(name,szFileName);
     return(r);
}


/* handle message to the OpenFile dialog */
BOOL FAR PASCAL FileOpenProc(HWND hWnd,WORD message,WORD wParam,
LONG lParam)
{
     char cLastChar;
     int nEditLen;
     switch(message)   {
```

```
    case WM_INITDIALOG:
     SendDlgItemMessage(hWnd,IDD_FNAME,EM_LIMITTEXT,80,0L);
     DlgDirList(hWnd,szFileSpec,IDD_FLIST,IDD_FPATH,0x4010);
     SetDlgItemText(hWnd,IDD_FNAME,szFileSpec);
     if(DIALOG_STATUS==DialogOff)
       SendMessage(hWnd,WM_COMMAND,IDOK,0L);
     return(TRUE);
    case WM_COMMAND:
         switch(wParam)  {
     case IDD_FLIST:
   switch(HIWORD(lParam))  {
        case LBN_SELCHANGE:
   if(DlgDirSelect(hWnd,szFileName,IDD_FLIST))
        lstrcat(szFileName,szFileSpec);
   SetDlgItemText(hWnd,IDD_FNAME,szFileName);
   return(TRUE);
      case LBN_DBLCLK:
   if(DlgDirSelect(hWnd,szFileName,IDD_FLIST) && DIALOG_STATUS==
  DialogOn)  {
        lstrcat(szFileName,szFileSpec);
        DlgDirList(hWnd,szFileName,IDD_FLIST,IDD_FPATH,0x4010);
        SetDlgItemText(hWnd,IDD_FNAME,szFileSpec);
   }
   else  {
        SetDlgItemText(hWnd,IDD_FNAME,szFileName);
        SendMessage(hWnd,WM_COMMAND,IDOK,0L);
   }
   return(TRUE);
         }
         break;
```

```
        IDL_
          if(HIWORD(lParam)==EN_CHANGE)
      EnableWindow(GetDlgItem(hWnd,IDOK),(BOOL)SendMessage(LOWORD(
lParam),WM_GETTEXTLENGTH,0,0L));
          return(TRUE);
    case IDOK:
      GetDlgItemText(hWnd,IDD_FNAME,szFileName,80);
      nEditLen=lstrlen(szFileName);
      cLastChar=*AnsiPrev(szFileName,szFileName+nEditLen);

      if(cLastChar!='\\' || cLastChar!=':')
lstrcat(szFileName,szFileSpec);

        if(lstrchr(szFileName,'*') || lstrchr(szFileName,'?'))  {
     if(DlgDirList(hWnd,szFileName,IDD_FLIST,IDD_FPATH,0x4010)) {
        lstrcpy(szFileSpec,szFileName);
        SetDlgItemText(hWnd,IDD_FNAME,szFileSpec);
     }  else MessageBeep(0);
     return(TRUE);
        }
        lstrcat(lstrcat(szFileName,"\\"),szFileSpec);

        if(DlgDirList(hWnd,szFileName,IDD_FLIST,IDD_FPATH,0x4010)){
     lstrcpy(szFileSpec,szFileName);
     SetDlgItemText(hWnd,IDD_FNAME,szFileSpec);
     return(TRUE);
        }
        szFileName[nEditLen]=0;
        if(-1==OpenFile(szFileName,pof,OF_READ | OF_EXIST)) {
     lstrcat(szFileName,szFileExt);
```

```
    if(-1==OpenFile(szFileName,pof,OF_READ | OF_EXIST))  {
 MessageBeep(0);
 return(TRUE);
    }
       }
       lstrcpy(szFileName,AnsiNext(lstrrchr(pof->szPathName,
'\\')));
       OemToAnsi(szFileName,szFileName);
       EndDialog(hWnd,TRUE);
                                return(TRUE);
 case IDCANCEL:
      EndDialog(hWnd,FALSE);
                                return(TRUE);
                }
             }
            return(FALSE);
}

/* quantize an image to create a palette */
int Quantize(int width,int depth,int *colourcount, char
*OutputColorMap,int (*proc)(char huge *p,unsigned int n))
{
     char far *p,far *linebuffer;
     unsigned int i,j;
     int Index;
     int NewColorMapSize;
     int NumOfEntries;
     unsigned long memS;
     long cRed,cGreen,cBlue;
     char huge *lpstr;
```

```
    NEWCOLOUR far *NewColorSubdiv;
    QCOLOUR far *ColorArrayEntries, far *QuantizedColor;
    HANDLE memH;

    memS=((long)sizeof(QCOLOUR)*(long)COLOR_ARRAY_SIZE)+
         ((long)sizeof(NEWCOLOUR) * 256L)+
  ((long)linewidth)+
         ((long)sizeof(QCOLOUR far *)*(long)COLOR_ARRAY_SIZE);

    if((memH GlobalAlloc(GMEM_MOVEABLE,memS)) = NULL)
         return(BAD_ALLOC);
    if((lpstr=(char huge *)GlobalLock(memH))==(char huge *)NULL){
         GlobalFree(memH);
         return(BAD_ALLOC);
    }

    ColorArrayEntries=(QCOLOUR far *)lpstr;
    NewColorSubdiv=(NEWCOLOUR far *)(lpstr+((long)sizeof(QCOLOUR)
*(long)COLOR_ARRAY_SIZE));
    linebuffer=(char far *)(lpstr+(((long)sizeof(QCOLOUR)*
COLOR_ARRAY_SIZE)+((long)sizeof(NEWCOLOUR) * 256L)));

    for(i=0;i<COLOR_ARRAY_SIZE;i++)  {
         ColorArrayEntries[i].RGB[RGB_RED] = i >> (2 *
BITS_PER_PRIM_COLOR);
  ColorArrayEntries[i].RGB[RGB_GREEN] = (i >> BITS_PER_PRIM_COLOR)
 & MAX_PRIM_COLOR;
         ColorArrayEntries[i].RGB[RGB_BLUE] = i & MAX_PRIM_COLOR;
         ColorArrayEntries[i].Count = 0L;
    }
```

```
        for(i=0;i<depth;++i)  {
             if((proc)(linebuffer,i))  {
          GlobalUnlock(memH);
                GlobalFree(memH);
                return(CANCEL_FUNCTION);
             }
             p=linebuffer;
             for(j=0;j<width;++j)  {
                Index = (((int)p[WRGB_RED] >> (8 -
BITS_PER_PRIM_COLOR)) << (2 * BITS_PER_PRIM_COLOR)) +
                        (((int)p[WRGB_GREEN] >> (8-
BITS_PER_PRIM_COLOR)) << BITS_PER_PRIM_COLOR) +
                        ((int)p[WRGB_BLUE] >> (8
 BITS_PER_PRIM_COLOR));
                ColorArrayEntries[Index].Count++;
                p+=RGB_SIZE;
             }
        }
        for(i=0;i<256;i++)  {
             NewColorSubdiv[i].QuantizedColors = (QCOLOUR far *)NULL;
             NewColorSubdiv[i].Count = 0L;
             NewColorSubdiv[i].NumEntries = 0;
             for(j=0;j<RGB_SIZE;++j)  {
                NewColorSubdiv[i].RGBMin[j] = 0;
                NewColorSubdiv[i].RGBWidth[j] = 255;
    }
        }
        for(i=0;i<COLOR_ARRAY_SIZE;i++)
             if(ColorArrayEntries[i].Count > 0)  break;
        QuantizedColor = NewColorSubdiv[0].QuantizedColors =
&ColorArrayEntries[i];
```

```
        NumOfEntries = 1;
        while(++i < COLOR_ARRAY_SIZE)
            if(ColorArrayEntries[i].Count > 0)  {
                QuantizedColor->Pnext=&ColorArrayEntries[i];
                QuantizedColor = &ColorArrayEntries[i];
                NumOfEntries++;
            }
        QuantizedColor->Pnext = (QCOLOUR far *)NULL;

        NewColorSubdiv[0].NumEntries = NumOfEntries;
        NewColorSubdiv[0].Count = (long)width * (long)depth;
        NewColorMapSize = 1;
        lpstr+=((long)sizeof(QCOLOUR)*(long)COLOR_ARRAY_SIZE)+
               ((long)sizeof(NEWCOLOUR) * 256L)+
               ((long)linewidth);

        dividemap(NewColorSubdiv,*colourcount,&NewColorMapSize,
    lpstr);
        if(NewColorMapSize < *colourcount)  {
            for(i=NewColorMapSize;i<*colourcount;i++)
          OutputColorMap[(i*3)+RGB_RED]=
          OutputColorMap[(i*3)+RGB_GREEN]=
          OutputColorMap[(i*3)+RGB_BLUE]=0;
        }
        for(i=0;i<NewColorMapSize;i++)  {
            if((j=NewColorSubdiv[i].NumEntries) > 0)  {
                QuantizedColor = NewColorSubdiv[i].QuantizedColors;
                cRed=cGreen=cBlue=0;
                while(QuantizedColor)  {
                    QuantizedColor->NewColorIndex=i;
```

```
                    cRed+=QuantizedColor->RGB[RGB_RED];
        cGreen+=QuantizedColor->RGB[RGB_GREEN];
        cBlue+=QuantizedColor->RGB[RGB_BLUE];
                    QuantizedColor=QuantizedColor->Pnext;
                }
                OutputColorMap[(i*RGB_SIZE)+RGB_RED]=(int)(cRed<<
(8-BITS_PER_PRIM_COLOR))/j;
                OutputColorMap[(i*RGB_SIZE)+RGB_GREEN]=(int)(cGreen
<<(8-BITS_PER_PRIM_COLOR))/j;
                OutputColorMap[(i*RGB_SIZE)+RGB_BLUE]=(int)(cBlue<<
(8-BITS_PER_PRIM_COLOR))/j;
            }
        }
        GlobalUnlock(memH);
        GlobalFree(memH);
        *colourcount = NewColorMapSize;
        return(SUCCESS);
}

int Dividemap(NEWCOLOUR far *NewColorSubdiv,int ColorMapSize,
int *NewColorMapSize,char huge *lpstr)
{
        unsigned int i,j;
        int MaxSize,Index=0;
        unsigned int NumEntries,MinColor,MaxColor;
        long Sum,Count;
        QCOLOUR far *QuantizedColor, far *far *SortArray;
        while(ColorMapSize > *NewColorMapSize)  {
            MaxSize=-1;
```

```
            for(i=0;i<*NewColorMapSize;i++)  {
                for(j=0;j<3;j++)  {
                    if(((int)NewColorSubdiv[i].RGBWidth[j]) >
 MaxSize && NewColorSubdiv[i].NumEntries > 1)  {
                        MaxSize = NewColorSubdiv[i].RGBWidth[j];
  Index = i;
                        SortRGBAxis = j;
                    }
                }
            }

            if(MaxSize == -1)  return(1);
            SortArray=(QCOLOUR far * far *)lpstr;
            for(j=0,QuantizedColor=NewColorSubdiv[Index].
 QuantizedColors; j < NewColorSubdiv[Index].NumEntries &&
 QuantizedColor != (QCOLOUR far *)NULL ; j++ , QuantizedColor=
 QuantizedColor->Pnext)

                SortArray[j]=QuantizedColor;

            psort((char far * far *)SortArray,NewColorSubdiv[Index].
 NumEntries);

            for(j=0;j<NewColorSubdiv[Index].NumEntries-1;j++)
                SortArray[j]->Pnext = SortArray[j+1];
            SortArray[NewColorSubdiv[Index].NumEntries-1]->Pnext=
 (QCOLOUR far *)NULL;
            NewColorSubdiv[Index].QuantizedColors=QuantizedColor=
 SortArray[0];
            Sum = NewColorSubdiv[Index].Count / 2 - QuantizedColor
```

```
->Count;
            NumEntries    1;
   Count =QuantizedColor->Count;
            while((Sum-=QuantizedColor ->Pnext ->Count) >= 0 &&
                  QuantizedColor->Pnext !  (QCOLOUR far *)NULL &&
                  QuantizedColor->Pnext->Pnext !=(QCOLOUR far *)NULL)
{
                  QuantizedColor = QuantizedColor ->Pnext;
                  NumEntries++;
                  Count+=QuantizedColor->Count;
            }
            MaxColor = QuantizedColor->RGBISort[RGBAxis];
            MinColor = QuantizedColor->Pnext ->RGBISort[RGBAxis];
            MaxColor <<=(8-BITS_PER_PRIM_COLOR);
            MinColor <<=(8-BITS_PER_PRIM_COLOR);

   NewColorSubdiv[*NewColorMapSize].QuantizedColors =QuantizedColor
->Pnext;
            QuantizedColor->Pnext = (QCOLOUR far *)NULL;
            NewColorSubdiv[*NewColorMapSize].Count =Count;
   NewColorSubdiv[Index].Count -= Count;
            NewColorSubdiv[*NewColorMapSize].NumEntries
NewColorSubdiv[Index].NumEntries - NumEntries;
            NewColorSubdiv[Index].NumEntries    NumEntries;
            for(j=0;j<3;j++)  {
                  NewColorSubdiv[*NewColorMapSize].RGBMin[j]
NewColorSubdiv[Index].RGBMin[j];
                  NewColorSubdiv[*NewColorMapSize].RGBWidth[j] =
NewColorSubdiv[Index].RGBWidth[j];
            }
```

```
    NewColorSubdiv[*NewColorMapSize].RGBWidth[SortRGBAxis] =
            NewColorSubdiv[*NewColorMapSize].RGBMin[SortRGBAxis] +
            NewColorSubdiv[*NowColorMapSize].RGBWidth[SortRGBAxis] -
            MinColor;
              NewColorSubdiv[*NewColorMapSize].RGBMin[SortRGBAxis]
= MinColor;

              NewColorSubdiv[Index].RGBWidth[SortRGBAxis] =
MaxColor - NewColorSubdiv[Index].RGBMin[SortRGBAxis];

              (*NowColorMapSize)++;
        }

        return(1);
}

/* add one detail to the details arrays */
void AddDetail(GLOBALHANDLE dtls,LPSTR s)
{

        LPSTR p,pr;
        int far *n;
        if((pr=GlobalLock(dtls)) != NULL)  {
              n=(int far *)pr;
    if(*n < MAXDETAILS)  {
                    p=pr+sizeof(int)+(*n)*(DETAILSIZE+1);
                    lstrcpy(p,s);
                    *n=*n+1;
              }
              GlobalUnlock(dtls);
```

```
    }
}

int ShowFrame(HWND hWnd,HDC hdc,char huge *p,int x,int y,
char *palette,int bits)
{
    LOGPALETTE   *pLogPal;
    PBITMAPINFO  pDibInfo;
    HANDLE  hPal=NULL;
    RECT  rect;
    char huge *pr,*pal;
    int  DestX,DestY,nWidth,nHeight,i,j,n,r=FALSE;

    hHourGlass=LoadCursor(NULL,IDC_WAIT);
    hPrevCursor=SetCursor(hHourGlass);

    GetClientRect(hWnd,&rect);

    n=1<<bits;
    j=max(n,256);
    if((pLogPal=(LOGPALETTE *)LocalAlloc(LMEM_FIXED,sizeof(
LOGPALETTE)+(j*sizeof(PALETTEENTRY)))) != NULL) {
  pLogPal->palVersion=0x0300;
  pLogPal->palNumEntries=j;
  pal=palette;
  for(i=0;i<j;i++)  {
       pLogPal->palPalEntry[i].peRed=*pal++;
       pLogPal->palPalEntry[i].peGreen=*pal++;
       pLogPal->palPalEntry[i].peBlue=*pal++;
       pLogPal->palPalEntry[i].peFlags=0;
```

```
}
hPal=CreatePalette(plogPal);
LocalFree((HANDLE)pLogPal);

SelectPalette(hdc,hPal,0);
RealizePalette(hdc);
   }

   if((pDibInfo=(PBITMAPINFO)LocalAlloc(LMEM_FIXED,sizeof(
BITMAPINFOHEADER)+j*sizeof(RGBQUAD))) != NULL) {
pDibInfo->bmiHeader.biSize=(long)sizeof(BITMAPINFOHEADER);
pDibInfo->bmiHeader.biWidth=(long)fi.width;
pDibInfo->bmiHeader.biHeight=(long)fi.depth;
pDibInfo->bmiHeader.biPlanes=1;
pDibInfo->bmiHeader.biBitCount=bits;
pDibInfo->bmiHeader.biCompression=0L;
pDibInfo->bmiHeader.biSizeImage=0L;
pDibInfo->bmiHeader.biXPelsPerMeter=0L;
pDibInfo->bmiHeader.biYPelsPerMeter=0L;
pDibInfo->bmiHeader.biClrUsed=0L;
pDibInfo->bmiHeader.biClrImportant=0L;

pal=palette;
for(i=0;i<j;++i)  {   // checked
      pDibInfo->bmiColors[i].rgbRed=*pal++;
      pDibInfo->bmiColors[i].rgbGreen=*pal++;
      pDibInfo->bmiColors[i].rgbBlue=*pal++;
      pDibInfo->bmiColors[i].rgbReserved=0;
}
i=min(fi.depth,rect.bottom-rect.top);
```

```
  p-=((long)y*(long)linewidth);

  pr=p+(long)(fi.depth-i)*(long)linewidth;

  if(DisplayStatus==LeftScreen)
  {
DestX=0;
DestY=0;
nWidth=xFaceWidth;
nHeight=i;
  }
  else if(DisplayStatus==RightScreen)
  {
DestX=xBeginHair;
DestY=0;
nWidth=xHairWidth;
nHeight=i;
  }
  else if(DisplayStatus==AllScreen)
  {
DestX=0;
DestY=0;
nWidth=min(fi.width,rect.right-rect.left);
nHeight=i;
  }
  SetDIBitsToDevice(hdc,DestX,DestY,nWidth,nHeight,x,0,0,i,
(LPSTR)pr,pDibInfo,DIB_RGB_COLORS);
  LocalFree((HANDLE)pDibInfo);
  r=TRUE;
     }
```

```
    if(hPal != NULL)  DeleteObject(hPal);
    SetCursor(hPrevCursor);
    return(r);
}

/* allocate buffer */
int getbuffer(FILEINFO *fi)
{
    unsigned long size;

    if(fi->bits==1)  {
        imagebits=1;
        linewidth=pixels2bytes(fi->width);
        if(linewidth & 0x0003)  linewidth=(linewidth | 3)+1;
  size=(long)linewidth*(long)(fi->depth+1);
    }
    else if(fi->bits >1 && fi->bits <=4)  {
  imagebits=4;
  linewidth=pixels2bytes(fi->width)<<2;
  if(linewidth & 0x0003)  linewidth=(linewidth | 3)+1;
  size=(long)linewidth*(long)(fi->depth+1);
    }
    else if(fi->bits >4 && fi->bits <=8)  {
  imagebits=8;
  linewidth=fi->width;
  if(linewidth & 0x0003)  linewidth=(linewidth | 3)+1;
  size=(long)linewidth*(long)(fi->depth+1);
    }
    else {
  imagebits=24;
```

```
  linewidth=fi->width*RGB_SIZE;
  if(linewidth & 0x0003)  linewidth (linewidth | 3)+1;
  size=(long)linewidth*(long)(fi->depth+1);
     }
     pagedepth=fi->depth;
     if((imagehandle=GlobalAlloc(GMEM_MOVEABLE,size))==NULL)
return(FALSE);
     else  return(TRUE);
}

void freebuffer()
{
     if(imagehandle != NULL)
          imagehandle=GlobalFree(imagehandle);
}

int putline(char huge *p,unsigned int n)
{     char huge *pr,huge *p1;
     unsigned int i;
     if((p1=GlobalLock(imagehandle)) != NULL)  {
          pr=p1+(long)linewidth*(long)(pagedepth-n-1);
          for(i=0;i<linewidth;++i)  *pr++=*p++;
          GlobalUnlock(imagehandle);
     }
     return(0);
}

int getline(char huge *p,unsigned int n)
{
     char huge *pr,huge *p1;
```

```
    unsigned int i;

    if((p1=GlobalLock(imagehandle)) != NULL)  {
        pr=p1+(long)linewidth*(long)(pagedepth-n-1);
        for(i=0;i<linewidth;++i)  *p++=*pr++;
        GlobalUnlock(imagehandle);
    }

    return(0);
}

/* display a Get info box for the image in fi */
void ShowInfo(HWND hWnd,FILEINFO *fi)
{
    FARPROC   dlgProc;

    dlgProc=MakeProcInstance((FARPROC)ShowInfoProc,hInst);
    DialogBox(hInst,"InfoBox",hWnd,dlgProc);
    FreeProcInstance(dlgProc);
}

int GetFileName(HWND hWnd,char *b)
{
    FARPROC   dlgProc;
    int  r;

    dlgProc=MakeProcInstance((FARPROC)GetNameProc,hInst);
    r=DialogBox(hInst,"NameBox",hWnd,dlgProc);
    lstrcpy(b,szFileName);
```

```
        char *ps,*pr,*p,*linebuffer;
        unsigned long size;
        unsigned int ditherlinewidth;
        unsigned int s,i,j,n,x;

        ditherlinewidth=pixels2bytes(fi->width)<<2;
        if(ditherlinewidth & 0x0003)  ditherlinewidth=(
ditherlinewidth | 3)+1;
        size=(long)ditherlinewidth*(long)(fi->depth+1);

        if((dh=GlobalAlloc(GMEM_MOVEABLE,size))==NULL)
return(BAD_ALLOC);
        if((pd=GlobalLock(dh))==NULL)  {
                GlobalFree(dh);
                return(BAD_ALLOC);
        }
        if((linebuffer=(char *)malloc(fi->width*RGB_SIZE))==NULL) {
                GlobalUnlock(dh);
                GlobalFree(dh);
                return(BAD_ALLOC);
        }

        if((p=(char *)malloc(linewidth))==NULL)  {
                free(linebuffer);
                GlobalUnlock(dh);
                GlobalFree(dh);
                return(BAD_ALLOC);
        }
        for(i=0;i<fi->depth;++i)  {
                if(fi->getline != NULL)  {
```

```
            if((fi->getline)(p,i))  {
                free(linebuffer);
                GlobalUnlock(dh);
                GlobalFree(dh);
                return(CANCEL_FUNCTION);
            }
        }
        pr=linebuffer;
        ps=p;
        if(fi->bits>1 && fi->bits<=4)  {
            for(j=x=0;j<fi->width;)  {
                n=(p[x] >> 4) & 0x00ff;
                lmemcpy(pr,fi->palette+n*RGB_SIZE,RGB_SIZE);
                pr+=RGB_SIZE;
                ++j;
                if(j<fi->width)  {
                    n=(p[x] & 0x000f);
                    lmemcpy(pr,fi->palette+n*RGB_SIZE,
RGB_SIZE);
                    pr+=RGB_SIZE;
                }
                ++j;
                ++x;
            }
        }
        else if(fi->bits>4 && fi->bits<=8)  {
            for(j=0;j<fi->width;++j)  {
                n=p[j];
                lmemcpy(pr,fi->palette+n*RGB_SIZE,RGB_SIZE);
                pr+=RGB_SIZE;
```

```
        }
    }
    else if(fi->bits==24)  {
        for(j=0;j<fi->width;++j)  {
            pr[RGB_RED]=ps[WRGB_RED];
            pr[RGB_GREEN]=ps[WRGB_GREEN];
            pr[RGB_BLUE]=ps[WRGB_BLUE];
            ps+=RGB_SIZE;
            pr+=RGB_SIZE;
        }
    }

    pr=linebuffer;
    pt=pd+(long)(fi->depth-i-1)*(long)ditherlinewidth;

    for(j=x=0;j<fi->width;)  {
        a=bayerpattern[i & 0x0007][j & 0x0007]<<2;
        n=0;
        if(pr[RGB_RED] > a) n |= 0x10;
        if(pr[RGB_GREEN] > a ) n |= 0x02;
        if(pr[RGB_BLUE] > a) n |= 0x04;
        pr+=RGB_SIZE;
        pt[x]=(n & 0x00f) << 4;
        ++j;
        if(j < fi->width)  {
            a=bayerpattern[i & 0x0007][j & 0x0007]<<2;
            n=0;
            if(pr[RGB_RED] > a )  n |= 0x01;
            if(pr[RGB_GREEN] > a)  n |= 0x02;
            if(pr[RGB_BLUE] > a)  n |= 0x04;
```

```
                    pr+=RGB_SIZE;
                    pt[x] |= (n & 0x000f);
                }
                ++j;
                ++x;
            }
        }
        free(p);
        free(linebuffer);

        GlobalUnlock(dh);

        freebuffer();

        imagehandle=dh;
        linewidth=linewidth=ditherlinewidth;
        imagebits=4;
        fi->imagesize=(long)linewidth*(long)fi->depth;
        fi->bits=3;

        lmemcpy(fi->palette,fixedpalette,8*RGB_SIZE);

        return(SUCCESS);
}


/*
Copy graphics content of client area to the Clipboard

*/
```

```
static void zPasteToClipboard(HWND hActiveWnd)
{
  HDC hDC, hMemDC;
  HBITMAP hBitmap, hPrevBitmap;
  int xwidth, ydepth;
  RECT rcClientArea;
GetClientRect(hActiveWnd, &rcClientArea);
xwidth= rcClientArea.right;
ydepth= rcClientArea.bottom;
hDC= GetDC(hActiveWnd);
hMemDC= CreateCompatibleDC(hDC);
hBitmap=
        CreateCompatibleBitmap(hDC, xwidth, ydepth);
if (hBitmap)
  {
  hPrevBitmap= SelectObject(hMemDC, hBitmap);
  BitBlt(hMemDC,0,0,xwidth,ydepth,hDC,0,0,SRCCOPY);
  if (OpenClipboard(hActiveWnd))
{
EmptyClipboard();
SetClipboardData(CF_BITMAP, hBitmap);
CloseClipboard();
}
  SelectObject(hMemDC, hPrevBitmap);
  LoadString(hInst,IDS_Completed,lpCaption,Max);
  LoadString(hInst,IDS_ClpbdText,lpMessage,MaxText);
  MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
  }
else
  {
```

```
    MessageBeep(0);
    LoadString(hInst,IDS_Error,lpCaption,Max);
    LoadString(hInst,IDS_ClpbdError,lpMessage,MaxText);
    MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
    }
  DeleteDC(hMemDC);
  ReleaseDC(hActiveWnd, hDC);
  return;
  }

  /*
  Paste graphics from the Clipboard into the window client area
  */

  static void zPasteFromClipboard(HWND hActiveWnd)
  {
    BITMAP bmPaste;
    RECT rcClientArea;
    int xwidth, ydepth;
    HDC hDC, hMemDC;
    HBITMAP hBitmap, hPrevBitmap;
    int Selection;
    int xDest, yDest;

  GetClientRect(hActiveWnd,&rcClientArea);
  xwidth= rcClientArea.right;
  ydepth= rcClientArea.bottom;
  hDC= GetDC(hActiveWnd);
  hMemDC= CreateCompatibleDC(hDC);
```

```
if (OpenClipboard(hActiveWnd))
  {
  if (!(hBitmap= GetClipboardData(CF_BITMAP)))
 {
 CloseClipboard();
 LoadString(hInst,IDS_Error,lpCaption,Max);
 LoadString(hInst,IDS_ClpbdHandle,lpMessage,MaxText);
 MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
 DeleteDC(hMemDC); ReleaseDC(hActiveWnd,hDC); return;
 }
  GetObject(hBitmap,
    sizeof(BITMAP),(LPSTR) &bmPaste);
  ImageWidth= bmPaste.bmWidth; ImageDepth= bmPaste.bmHeight;
  if ((bmPaste.bmWidth > xwidth) && (bmPaste.bmHeight > ydepth))
 {
 LoadString(hInst,IDS_Source,lpCaption,Max);
 LoadString(hInst,IDS_WD,lpMessage,MaxText);
 Selection= MessageBox(GetFocus(),lpMessage,lpCaption,
MB_ICONQUESTION | MB_YESNOCANCEL);
 switch (Selection)
{
case IDCANCEL: CloseClipboard(); DeleteDC(hMemDC);
ReleaseDC(hActiveWnd,hDC); return;
case IDYES:    xDest= xwidth; yDest= ydepth;
goto SHRINK_PASTE;
case IDNO:goto CLIP_PASTE;
default:goto CLIP_PASTE;
}
 }
  if (bmPaste.bmWidth > xwidth)
```

```
 {
    LoadString(hInst,IDS_Source,lpCaption,Max);
 LoadString(hInst,IDS_Width,lpMessage,MaxText);
 Selection= MessageBox(GetFocus(),lpMessage,lpCaption,
MB_ICONQUESTION | MB_YESNOCANCEL);
    switch (Selection)
{
case IDCANCEL: CloseClipboard(); DeleteDC(hMemDC);
ReleaseDC(hActiveWnd,hDC); return;
case IDYES:    xDest= xwidth; yDest= bmPaste.bmHeight;
goto SHRINK_PASTE;
case IDNO:goto CLIP_PASTE;
default:goto CLIP_PASTE;
}
 }
if (bmPaste.bmWidth > ydepth)
 {
    LoadString(hInst,IDS_Source,lpCaption,Max);
 LoadString(hInst,IDS_Depth,lpMessage,MaxText);
 Selection= MessageBox(GetFocus(),lpMessage,lpCaption,
MB_ICONQUESTION | MB_YESNOCANCEL);
    switch (Selection)
{
case IDCANCEL: CloseClipboard(); DeleteDC(hMemDC);
ReleaseDC(hActiveWnd,hDC); return;
case IDYES:    xDest= bmPaste.bmWidth; yDest= ydepth;
goto SHRINK_PASTE;
case IDNO:goto CLIP_PASTE;
default:goto CLIP_PASTE;
}
```

```
 }
CLIP_PASTE:
hPrevBitmap= SelectObject(hMemDC, hBitmap);
BitBlt(hDC,
 0, 0,
 bmPaste.bmWidth,
 bmPaste.bmHeight,
 hMemDC,
 0, 0,
 SRCCOPY);
goto FINISHED_PASTE;
SHRINK_PASTE:
hPrevBitmap= SelectObject(hMemDC, hBitmap);
StretchBlt(hDC,
  0, 0,
  xDest, yDest,
  hMemDC,
     0, 0,
  bmPaste.bmWidth,
  bmPaste.bmHeight,
              SRCCOPY);

FINISHED_PASTE:
  SelectObject(hMemDC,hPrevBitmap);

  CloseClipboard();
  }
DeleteDC(hMemDC);
ReleaseDC(hActiveWnd,hDC);
return;
```

```
}

/*
Display message if user attempts to move or resize window
*/

static void zSystemMessage(void)
{
MessageBeep(0);
LoadString(hInst,IDS_System,lpCaption,Max);
LoadString(hInst,IDS_SystemText,lpMessage,MaxText);
MessageBox(GetFocus(),lpMessage,lpCaption,MB_OK);
return;
}

/*
DISPLAY FACE IMAGE AT LEFT POSITION (FaceNo is static)INC. FaceNo
*/
static void zDisplayFace(HWND hWnd,int CountFace)
{

if(FaceNo==0)
  FaceNo=MaxFaceFile;
CountFace=FaceNo;
hHourGlass=LoadCursor(NULL,IDC_WAIT); // hour glass cursor
hPrevCursor=SetCursor(hHourGlass);
switch(CountFace)  {// select file name to szFileSpec
  case 1: lstrcpy(szFileSpec,"z2563");
break;
  case 2:       lstrcpy(szFileSpec,"z2564");
```

```
break;
  default : lstrcpy(szFileSpec,"z2563");

}
FaceNo++;          // increase static var. of face index
if(FaceNo>MaxFaceFile)
  FaceNo=1;

DisplayStatus=LeftScreen;// Display Face Only Left Screen
zOpenFile(hWnd,DialogOff);
SetCursor(hPrevCursor);
}

static void zDisplayBaseFace(HWND hWnd,int CountFace)
{

if(FaceNo==0)
  FaceNo=MaxFaceFile;
CountFace=FaceNo;
hHourGlass=LoadCursor(NULL,IDC_WAIT); // hour glass cursor
hPrevCursor=SetCursor(hHourGlass);
switch(CountFace)  {// select file name to szFileSpec
  case 1: lstrcpy(szFileSpec,"b2563");
break;
  case 2: lstrcpy(szFileSpec,"b2564");
break;
  default : lstrcpy(szFileSpec,"b2563");
}
FaceNo++;          // increase static var. of face index
if(FaceNo>MaxFaceFile)
```

```
  FaceNo=1;
DisplayStatus=LeftScreen;// Display Face Only Left Screen
zOpenFile(hWnd,DialogOff);

SetCursor(hPrevCursor);
}

/*
DISPLAY HAIR IMAGE AT RIGHT POSITION  (HairNo is static)INC HairNo
*/

static void zDisplayHair(HWND hWnd,int CountHair)
{
if(HairNo==0)
  HairNo=MaxHairFile;
hHourGlass=LoadCursor(NULL,IDC_WAIT);
hPrevCursor=SetCursor(hHourGlass);
switch(CountHair)  {
  case 1: lstrcpy(szFileSpec,"h2563");
break;
  case 2: lstrcpy(szFileSpec,"h2564");
break;
  default : lstrcpy(szFileSpec,"h2563");

}
HairNo++;   // increase static var. of hair index
if(HairNo>MaxHairFile)
  HairNo=1;

DisplayStatus=RightScreen;
```

```
zOpenFile(hWnd,DialogOff);
SetCursor(hPrevCursor);
}

void FMDisplay(HWND hWnd, int Mode)
{
hHourGlass=LoadCursor(NULL,IDC_WAIT);
hPrevCursor=SetCursor(hHourGlass);

if(Mode==Eye)  {// if must display eye
  if(EyeNo<1)  EyeNo=MaxEyeFile;
  switch(EyeNo)  {
  case 1: lstrcpy(szFileSpec,"e2563");
break;
  case 2: lstrcpy(szFileSpec,"e2564");
break;
  default : lstrcpy(szFileSpec,"e2563");
  }
EyeNo++;   // increase static var. of eye index
if(EyeNo>MaxEyeFile)
  EyeNo=1;
}
else  {                  // if display nose
  if(NoseNo<1)  NoseNo=MaxNoseFile;
  switch(NoseNo)  {
  case 1: lstrcpy(szFileSpec,"n2563");
break;
  case 2: lstrcpy(szFileSpec,"n2564");
break;
  default : lstrcpy(szFileSpec,"n2563");
```

```
  }
NoseNo++;   // Increase static var. of eye index
if(NoseNo>MaxNoseFile)
  NoseNo=1;
}

DisplayStatus=RightScreen;
zOpenFile(hWnd,DialogOff);

SetCursor(hPrevCursor);

}

static RGB_REF FindRGB(int Color)
{
  RGB_REF r_value;
  int RedRef,GreenRef,BlueRef;
     switch(Color)
     {
      case(Black):
    RedRef=0;
    GreenRef=0;
    BlueRef=0;
    break;
      case(DarkBrown):
    RedRef=128;
    GreenRef=64;
    BlueRef=64;
    break;
      case(Brown):
```

```
    RedRef=128;
    GreenRef=64;
    BlueRef=0;
    break;
     case(LightBrown):
    RedRef=128;
    GreenRef=128;
    BlueRef=64;
    break;
      case(RedBrown):
    RedRef=192;
    GreenRef=64;
    BlueRef=0;
    break;
 }
 r_value.red_ref=RedRef;
 r_value.green_ref=GreenRef;
 r_value.blue_ref=BlueRef;
 return(r_value);
}

/*
Change color to hair
*/

static void zPaintHair(HWND hWnd,int HColor,int LColor)
{                       // LColor is reference color
    PAINTSTRUCTps;
    DWORDBkColor, BkRed, BkGreen, BkBlue;
    inti, j, dColor, dRed, dGreen, dBlue;
```

```
    RGB_REF rgb_st;
    HBRUSHhBrush;

OldHairColor=HairColor;   // LColor is OldHairColor
BeginPaint(hWnd, &ps);
hHourGlass=LoadCursor(NULL,IDC_WAIT);
hPrevCursor=SetCursor(hHourGlass);
rgb_st=FindRGB(HColor);     // Find RGB of new color
ps.hdc=GetDC(hWnd);
BkColor=GetPixel(ps.hdc,xBeginHair+2,yBeginHair+2);// Read bk.
/* Find RGB of hair background */
BkRed=GetRValue(BkColor)%256;
BkBlue=GetBValue(BkColor)%256;
BkGreen=GetGValue(BkColor)%256;

hBrush=CreateSolidBrush(RGB(rgb_st.red_ref,rgb_st.green_ref,
rgb_st.blue_ref));
SelectObject(ps.hdc,hBrush);
for(j=yBeginHair;j<yBeginHair+yHairWidth;j++)
  for(i=xBeginHair;i<xBeginHair+xHairWidth;i++)
  {
    dColor=GetPixel(ps.hdc,i,j);// Read hair ...destination
// Find RGB of hair (destination)
    dRed=GetRValue(dColor)%256;
    dBlue=GetBValue(dColor)%256;
    dGreen=GetGValue(dColor)%256;

    if(dRed!=BkRed || dGreen!=BkGreen || dBlue!=BkBlue)
     {  FloodFill(ps.hdc,i,j,RGB(BkRed,BkGreen,BkBlue));
i+=xHairWidth;
```

```
j+=yHairWidth;
      }
  }
HairColor=HColor;
SetCursor(hPrevCursor);
ReleaseDC(hWnd,ps.hdc);
EndPaint(hWnd,&ps);
}

/*
Paste hair to face
*/

static void zPasteToFace(HWND hWnd)
{
hHourGlass=LoadCursor(NULL,IDC_WAIT);
hPrevCursor=SetCursor(hHourGlass);
FaceNo--;
if(FaceNo==0)  FaceNo=MaxFaceFile;
zDisplayBmpFace(hWnd,FaceNo);
ReadNotBk(hWnd, xBeginHair, yBeginHair, xBeginHair+xHairWidth-20,
yBeginHair+yHairWidth-150);
SetCursor(hPrevCursor);
}

void ReadNotBk(HWND hWnd, int x1, int y1, int x2, int y2)
{
HDC hDC;
int i, j;
DWORD  PtColor, PtRed, PtGreen, PtBlue;
```

```
hDC=GetDC(hWnd);
for(j=y1;j<=y2;j++)
  for(i=x1;i<=x2;i++)  {
    PtColor=GetPixel(hDC,i,j);// Read hair ...destination
    PtRed=GetRValue(PtColor)%256;
    PtGreen=GetGValue(PtColor)%256;
    PtBlue=GetBValue(PtColor)%256;

    if(PtRed!=255 || PtGreen!=255 || PtBlue!=255) // if not Bk.
SetPixel(hDC,i-xBeginHair,j,RGB(PtRed,PtGreen,PtBlue));
  }

ReleaseDC(hWnd,hDC);
}
```

ไฟล์ PROJECT

BMP.CPP

```
#include <windows.h>
#include <stdio.h>
#include <alloc.h>
#include <dir.h>
#include "view.h"

typedef struct {
        char id[2];
        long filesize;
        int reserved[2];
        long headersize;
        long infoSize;
        long width;
        long depth;
        int biPlanes;
        int bits;
        long biCompression;
        long biSizeImage;
        long biXPelsPerMeter;
        long biYPelsPerMeter;
        long biClrUsed;
        long biClrImportant;
        } BMPHEAD;

typedef struct {
        char id[2];
```

```
        long filesize;
        int reserved[2];
        long headersize;
        long infoSize;
        int width;
        int depth;
        int biPlanes;
        int bits;
        } BMPCOREHEAD;

char szAppName[33]="View BMP";
char szFileExt[]=".BMP";

extern unsigned int linewidth,pagedepth;
extern char masktable[];
extern char bittable[];

int ReadFile(FILEINFO *fi,char *path)
{
     FILE *fp;
     BMPHEAD bmp;
     BMPCOREHEAD *bmc;
     char *linebuffer;
     int i,bytes;

     if((fp=fopen(path,"rb"))==NULL) return(BAD_OPEN);

    if(fread((char *)&bmp,1,sizeof(BMPHEAD),fp)==sizeof(BMPHEAD)){

     if(!memcmp(bmp.id,"BM",2)) {
```

```
        if(bmp.infoSize==12L) {
              bmc=(BMPCOREHEAD *)&bmp;
              fi->width=bmc->width;
              fi->depth=bmc->depth;
              fi->bits=bmc->bits;
              fseek(fp,(long)sizeof(BMPCOREHEAD),SEEK_SET);
        }
        else {

              fi->width=(int)bmp.width;
              fi->depth=(int)bmp.depth;
              fi->bits=bmp.bits;
        }

        /* work out the line width */
        if(fi->bits==1) bytes=pixels2bytes(fi->width);
        else if(fi->bits==4) bytes=pixels2bytes(fi->width)<<2;
        else if(fi->bits==8) bytes=fi->width;
        else if(fi->bits==24) bytes=fi->width*3;

        if(bytes & 0x0003) {
              bytes |= 0x0003;
              ++bytes;
        }
        if((linebuffer=(char *)malloc(max(fi->width,linewidth)))
==NULL){
              fclose(fp);
              return(BAD_ALLOC);
        }
        fseek(fp,bmp.headersize,SEEK_SET);
```

```
        for(i=0;i<fi->depth;++i) {
            if(fread(linebuffer,1,bytes,fp) !=bytes) {
                free(linebuffer);
                fclose(fp);
                return(BAD_READ);
            }

            if(fi->putline!=NULL) {
                if((fi->putline)(linebuffer,fi->depth-i-1)) {
                    free(linebuffer);
                    fclose(fp);
                    return(CANCEL_FUNCTION);
                }
            }
        }
        free(linebuffer);
        fclose(fp);
        return(SUCCESS);
    }
    else {
        fclose(fp);
        return(BAD_FILE);
    }
    }
    else {
        fclose(fp);
        return(BAD_READ);
    }
}
```

```
int GetInfo(FILEINFO *fi,char *path)
{
     FILE *fp;
     BMPHEAD bmp;
     BMPCOREHEAD *bmc;
     char ext[MAXEXT];
     int i,n;

     fnsplit(path,NULL,NULL,fi->name,ext);
     lstrcat(fi->name,ext);

     fi->comments[0]=0;

     if((fp=fopen(path,"rb"))==NULL)     return(BAD_OPEN);

     fseek(fp,0L,SEEK_END);
     fi->filesize=ftell(fp);
     fseek(fp,0L,SEEK_SET);
     lmemcpy(fi->palette,"\000\000\000\377\377\377",6);

     if(fread((char *)&bmp,1,sizeof(BMPHEAD),fp)==sizeof(BMPHEAD))
          if(!lmemcmp(bmp.id,"BM",2)) {

               if(bmp.infoSize==12L) {
                    bmc=(BMPCOREHEAD *)&bmp;
                    fi->width=bmc->width;
                    fi->depth=bmc->depth;
                    fi->bits=bmc->bits;
                    fseek(fp,(long)sizeof(BMPCOREHEAD),SEEK_SET);
               }
```

```
            else {

                fi->width=(int)bmp.width;
                fi->depth=(int)bmp.depth;
                fi->bits=bmp.bits;
            }
            fi->background=0;

            fi->imagesize=(long)pixels2bytes(fi->width)*
(long)fi->bits*(long)fi->depth;
            if(fi->bits==1)
             lmemcpy(fi->palette,"\000\000\000\377\377\377",6);

            if(fi->bits>1 && fi->bits <=8)  {
               n=1<<fi->bits;
               for(i=0;i<n;++i) {
                fi->palette[(i*RGB_SIZE)+RGB_BLUE]=fgetc(fp);
                fi->palette[(i*RGB_SIZE)+RGB_GREEN]=fgetc(fp);
                fi->palette[(i*RGB_SIZE)+RGB_RED]=fgetc(fp);
                if(bmp.infoSize!=12L)  fgetc(fp);
                }
            }
            else if(fi->bits==24)  {
                 for(i=0;i<256;++i)
                  lmemset(fi->palette+(i*RGB_SIZE),i,RGB_SIZE);
            }

            fclose(fp);
            return(SUCCESS);
        }
```

```
            else {
                  fclose(fp);
                  return(BAD_FILE);
            }
      }
      else {
            fclose(fp);
            return(BAD_FILE);
      }
}

int WriteFile(FILEINFO *fi,char *path)
{
      FILE *fp;
      BMPHEAD bmp;
      char *linebuffer;
      int i,bytes;
      if((fp=fopen(path,"wb"))==NULL)  return(BAD_CREATE);

      if(fi->bits==1)  bytes=pixels2bytes(fi->width);
      else if(fi->bits>1 && fi->bits<=4)  bytes=pixels2bytes(
fi->width)<<2;
      else if(fi->bits>4 && fi->bits<=8)  bytes=fi->width;
      else bytes=fi->width*RGB_SIZE;
      if(bytes & 0x0003)  {
            bytes |= 0x0003;
            ++bytes;
      }
      if((linebuffer=(char *)malloc(max(fi->width,linewidth)))==
NULL){
```

```
        fclose(fp);
        remove(path);
        return(BAD_ALLOC);
    }

    lmemset((char *)&bmp,0,sizeof(BMPHEAD));
    lmemcpy(bmp.id,"BM",2);
    if(fi->bits==1)  {
        bmp.headersize=62L;
        bmp.filesize=bmp.headersize+(long)bytes*(long)fi->depth;
    }
    else if(fi->bits>1 && fi->bits<=4)  {
        bmp.headersize=118L;
        bmp.filesize=bmp.headersize+(long)bytes*(long)fi->depth;
    }
    else if(fi->bits>4 && fi->bits<=8)  {
        bmp.headersize=1078L;
        bmp.filesize=bmp.headersize+(long)bytes*(long)fi->depth;
    }
    else  {
        bmp.headersize=54L;
        bmp.filesize=bmp.headersize+(long)bytes*(long)fi->depth;
    }

    bmp.width=(long)fi->width;
    bmp.depth=(long)fi->depth;
    bmp.infoSize=0x28L;
    if(fi->bits==1)  bmp.bits=1;
    else if(fi->bits>1 && fi->bits<=4)  bmp.bits=4;
    else if(fi->bits>4 && fi->bits<=8)  bmp.bits=8;
```

```
    else  bmp.bits=24;
    bmp.biPlanes=1;
    bmp.biCompression=0L;

    if(fwrite((char *)&bmp,1,sizeof(BMPHEAD),fp) !=sizeof(BMPHEAD
))(
        free(linebuffer);
        fclose(fp);
        remove(path);
        return(BAD_WRITE);
    }
    if(bmp.bits==1)
        fwrite("\000\000\000\000\377\377\377\000",1,8,fp);
    else if(bmp.bits==4)  {
        for(i=0;i<16;++i)  {
            fputc(fi->palette[i*RGB_SIZE+RGB_BLUE],fp);
            fputc(fi->palette[i*RGB_SIZE+RGB_GREEN],fp);
            fputc(fi->palette[i*RGB_SIZE+RGB_RED],fp);
            fputc(0,fp);
        }
    }
    else if(bmp.bits==8)  {
        for(i=0;i<256;++i)  {
            fputc(fi->palette[i*RGB_SIZE+RGB_BLUE],fp);
            fputc(fi->palette[i*RGB_SIZE+RGB_GREEN],fp);
            fputc(fi->palette[i*RGB_SIZE+RGB_RED],fp);
            fputc(0,fp);
        }
    }
```

```
    for(i=0;i<fi->depth;++i)  {
        if((fi->getline)(linebuffer,fi->depth-i-1))  {
            free(linebuffer);
            fclose(fp);
            remove(path);
            return(CANCEL_FUNCTION);
        }

        if(bmp.bits==1)  {
      if(fwrite(linebuffer,1,bytes,fp)!=bytes)  {
   free(linebuffer);
   fclose(fp);
   remove(path);
   return(BAD_WRITE);
      }
}
else if(bmp.bits==4)  {
      if(fwrite(linebuffer,1,bytes,fp)!=bytes)  {
   free(linebuffer);
   fclose(fp);
   remove(path);
   return(BAD_WRITE);
      }
}
else if(bmp.bits==8)  {
      if(fwrite(linebuffer,1,bytes,fp)!=bytes)  {
   free(linebuffer);
   fclose(fp);
   remove(path);
   return(BAD_WRITE);
```

การใช้งานโปรแกรม

เมื่อเริ่ม run โปรแกรมจะโหลดภาพใบหน้ามาไว้ด้านซ้าย และภาพทรงผมมาไว้ด้านขวามือ

การเปลี่ยนสีผม

ให้โปรแกรมอยู่ในโหมด HairMode โดยดูที่เมนู Mode แล้วจึงทำการเลือกเมนู HairDone และเลือก HairColor ซึ่งจะมีเมนูให้เลือกสีผม 5 สี คือ

สีดำ (Black)

สีน้ำตาลเข้ม (DarkBrown)

สีน้ำตาล (Brown)

สีน้ำตาลสว่าง (BrightBrown)

สีน้ำตาลแดง (Red-Brown)

เมื่อทำการเปลี่ยนสีทรงผมแล้ว ทำให้ภาพทรงผมด้านขวามือมีสีตามที่เลือกไว้ แต่ที่ใบหน้านั้นยังมีสีทรงผมเดิม จึงต้องทำงานต่อดังนี้ คือ เลือกเมนู HairDone และเลือกเมนู Go หลังจากนั้นโปรแกรมจะโหลดภาพพื้นหน้านั้น ๆ นำมารวมกับภาพทรงผมด้านขวามือ

การเปลี่ยนส่วนประกอบของใบหน้า

ให้โหมดของโปรแกรมอยู่ในโหมด FaceMode โดยดูที่เมนู Mode ซึ่งเมื่ออยู่ใน FaceMode แล้วโปรแกรมจะมีภาพใบหน้าอยู่ด้านซ้าย ส่วนภาพส่วนประกอบจะเป็นภาพตา หรือ จมูก ซึ่งอยู่ด้านขวามือนั้นสามารถเลือกได้ในเมนู Change

การนำภาพส่วนประกอบของใบหน้า เช่น ตา ไปประกอบกับใบหน้า ต้องเลือกเมนู FaceDone แล้วเลือกเมนู Change อีกทีหนึ่ง

การใส่ค่าในคลิปบอร์ด และการนำค่าออกจากคลิปบอร์ด

เลือกเมนู Edit ถ้านำค่ามาจากคลิปบอร์ดเลือก Paste แต่ถ้าจะนำค่าใส่คลิปบอร์ดเลือก Copy

การทำงานเกี่ยวกับไฟล์

เลือกเมนู File สำหรับทำงานเกี่ยวกับไฟล์ โดยจะมีเมนู Save สำหรับเก็บภาพเป็นรูปแบบ BMP เมนู Open สำหรับอ่านไฟล์ BMP ส่วนเมนู GetInfo สำหรับดูข้อมูลเรื่อง ขนาด และ สีของไฟล์

ภาคผนวก ข

คู่มือการใช้งาน

คู่มือการใช้งานโปรแกรม

โปรแกรมนี้ทำงานภายใต้วินโดวส์ โดยการทำงานจะเริ่มต้นด้วยการโหลดภาพใบหน้า และภาพทรงผมขึ้นมา ส่วนไฟล์ที่จะโหลดขึ้นมานั้น เป็นฟอร์แมต BMP ที่มีชื่ออยู่ในโมดูล SelectFaceName, SelectHairName ซึ่งเป็นการเลือกชื่อของภาพใบหน้าและทรงผม ส่วนภาพตาและจมูกนั้น จะมีชื่ออยู่ในโมดูล FMDisplay โดยที่โปรแกรมจะเซตค่าเริ่มต้นของไฟล์ภาพต่าง ๆ ดังนี้

ชนิดของภาพ	จำนวน	ชื่อ (.BMP)
ใบหน้า	2	Z2563
		Z2564
พื้นหน้า	2	B2563
		B2564
ตา	2	E2563
		E2564
จมูก	2	N2563
		N2564

ภาพที่เป็นต้นแบบนั้น เป็นภาพขนาด 256 สี แต่ถ้าการแสดงผลของฮาร์ดแวร์น้อยกว่า 256 สีนั้น จะมีโมดูลของโปรแกรมทำหน้าที่แสดงสีโดยการดิชเชอร์

การเลือกโหมดแสดงผลของโปรแกรม ควรเลือกโหมดที่สูงกว่า 256 สีในการแสดงผลภาพขนาด 256 สี เนื่องจากถ้าเลือกโหมด 256 สีแล้วจะเกิดการผิดพลาดของค่าจานสีในภาพที่ถูกแสดงก่อน ดังรูปในหน้า 14

ก่อนเริ่มเข้าโปรแกรม

ผู้ใช้ต้องนำภาพใบหน้าที่ถ่ายจากกล้องวีดีโอ และนำภาพส่วนประกอบต่าง ๆ คือ ทรงผม ตา จมูก โดยอาจทำการวาดภาพส่วนประกอบต่าง ๆ เอง หรือ อาจตัดมาจากใบหน้าจริง โดยใช้โปรแกรมในการวาดรูปเช่น Painbrush เข้าช่วย หรืออาจนำมาแต่ภาพใบหน้าแล้วใช้ภาพส่วนประกอบที่มีอยู่เดิม ซึ่งสิ่งที่สำคัญคือ โปรแกรมจ์จะทำงานกับไฟล์ที่ถูกตั้งชื่อไว้ตามที่กล่าวมาแล้วเท่านั้น และก่อนเข้าโปรแกรมนั้นจะต้องเซตโหมดการทำงานของจอในโหมด 6 หมื่นสีเป็นอย่างน้อย

การนำภาพใบหน้าลงในโปรแกรม ทำได้ 2 วิธีคือ

1. ทำการลบไฟล์เก่าที่เป็นภาพใบหน้าออก แล้วใส่ภาพใบหน้าใหม่ลงไปโดยตั้งชื่อให้ตรงกับที่ตั้งไว้ในโปรแกรม

2. ทำการเพิ่มชื่อลงในโมดูล SelectFaceName และเปลี่ยนค่า MaxFaceFile ซึ่งเป็นตัวแปร static ตามจำนวนไฟล์ภาพใบหน้าที่มี แล้วทำการคอมไพล์โปรแกรมใหม่

หมายเหตุ การใส่ภาพใบหน้าลงในโปรแกรมทุกภาพนั้นจะต้องใส่ภาพที่เป็นพื้นหน้าที่ไม่มีผมอยู่ด้วย โดยการแต่งภาพในโปรแกรมแต่งภาพ ตัวอย่าง เช่น ภาพใบหน้าคือ Z2563 จะใช้ภาพทรงผม B2563

การเพิ่มภาพทรงผม ตา หรือ จมูก มี 2 วิธีเช่นเดียวกับภาพใบหน้า โดยอาจทำการลงไฟล์เก่าออกแล้วนำไฟล์ใหม่ไปใส่ในชื่อเดียวกัน หรือการแก้ค่าในโมดูลของโปรแกรมซึ่งมีข้องแตกต่างในชื่อโมดูลคือ

SelectHairName สำหรับทรงผม

FMDisplay สำหรับ ตา จมูก

ส่วนความแตกต่างของตัวแปร static ที่บอกจำนวนของภาพคือ

ทรงผม - MaxHairFile

ตา - MaxEyeFile

จมูก - MaxNoseFile

การเพิ่มส่วนประกอบนั้นไม่ต้องใส่ภาพพื้นหน้าลงไปเหมือนการเพิ่มภาพใบหน้า แต่มีข้อแม้ที่สำคัญคือ สีพื้นของภาพส่วนประกอบทุกภาพต้องเป็นสีขาวเท่านั้น

```
        }
    }
    else if(bmp.bits==24)  {
        if(fwrite(linebuffer,1,bytes,fp)!=bytes)  {
      free(linebuffer);
      fclose(fp);
      remove(path);
      return(BAD_WRITE);
        }
    }
    }
    free(linebuffer);
    fclose(fp);
    return(SUCCESS);
}
```

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงด้วยดี เนื่องด้วยการอนุเคราะห์จาก อาจารย์ สมศักดิ์ มิตะถา และ อาจารย์ ดร. ชม กิ้มปาน และขอขอบคุณ นาย ไพบูลย์ กาญจนบัตร นักศึกษาภาควิชาวิศวกรรมคอมพิวเตอร์ ชั้นปีที่ 3 ที่ให้การสนับสนุนด้านอุปกรณ์กล้องวิดีโอและหนังสือที่ใช้ในการทำงานเป็นอย่างดี

จึงใคร่ขอขอบคุณท่านผู้ได้กล่าวนามมา ณ. ที่นี้