การทำภาพเคลื่อนไหวบนคอมพิวเตอร์โดยใช้เทคนิคมอร์ฟฟิง

โดย

นางสาวนิติพร อมรเลิศวิทย์ รหัส 34501006
นางสาวอรนุช ธนกิจลีลาเจริญ รหัส 34501027
นายเอกชัย รัตนดิลกชัย รหัส 34501032

ปัญหาพิเศษฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต

สาขาคณิตศาสตร์ประยุกต์
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2537

COMPUTER GRAPHIC ANIMATION BY MORPHING ALGORITHM

BY

MISS NITIPORN AMORNLERTWIT	CODE	34501006
MISS ORANUCH THANAKITLELACHAROEN	CODE	34501027
MR. AKEKACHAI RATTANADILOKCHAI	CODE	34501032

A Special Project Submitted in Partial Fulfillment
of the Requirement of the Degree of Bachelor of Science

Department of Mathematics and Computer Science
Faculty of Science
King Mongkut's Institute of Technology Chaokhumtaharn
Ladkrabang

1994

ปัญหาพิเศษเรื่อง	การทำภาพเคลื่อนไหวบนคอมพิวเตอร์โดยใช้เทคนิคมอร์ฟฟิง
ชื่อนักศึกษา	1. นางสาวนิติพร อมรเลิศวิทย์ รหัส 34501006
	2. นางสาวอรนุช ธนกิจลีลาเจริญ รหัส 34501027
	3. นายเอกชัย รัตนดิลกชัย รหัส 34501032
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษา	อาจารย์ศรัณย์ อินทโกสุม

ปัญหาพิเศษฉบับนี้ กรรมการสอบปัญหาพิเศษได้ตรวจพิจารณาแล้วจึงอนุมัติให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต สาขาคณิตศาสตร์ประยุกต์ ประจำปีการศึกษา 2537

(รองศาสตราจารย์ภัคคินี ชิตสกุล) รักษาการหัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะกรรมการโครงงานพิเศษ

(อาจารย์วีระชัย ตันยะสิทธิ์) ประธานกรรมการ

(อาจารย์พรชัย เจนจิระพงศ์เวช) กรรมการ

(อาจารย์ศรัณย์ อินทโกสุม) กรรมการและอาจารย์ที่ปรึกษา

บทคัดย่อ

ปัญหาพิเศษฉบับนี้เสนอโปรแกรมประยุกต์สำหรับการทำภาพเคลื่อนไหวบนคอมพิวเตอร์ โดยใช้เทคนิคมอร์ฟฟิง ซึ่งเทคนิคนี้ประกอบด้วยหลักการ 3 อย่าง คือ การทวีน การวอร์ป และการดิสโซลว์ โดยเลือกใช้การเก็บภาพในรูปแบบแฟ้มข้อมูลทีจีเอ (TGA) โดยมีขั้นตอนในการดำเนินการพัฒนาคือ ศึกษาทฤษฎีที่เกี่ยวข้อง การออกแบบโปรแกรม และการพัฒนาโปรแกรม นี้ จะพัฒนาโดยใช้ภาษา C++ เป็นส่วนใหญ่ ผลที่ได้รับก็คือ จะได้ชุดของภาพที่ได้จากการประมวลผลในรูปแบบทีจีเอ และสามารถนำภาพเหล่านั้นมาสร้างเป็นภาพเคลื่อนไหวบนคอมพิวเตอร์จากรูปต้นทางไปเป็นรูปปลายทาง

Abstract

The Purpose of this project is to build application program for making graphic animation on computer by morphing algorithm . This morphing algorithm consists of 3 methods; the first one is tweening algorithm , the second one is warping algorithm and the last one is dissolving algorithm . The output of file format is TGA file format . The development process studied are the retated theories , software design and system implementation which mostly use C++ language programming as a development tool . The result is graphic animation made from source image to target image .

กิตติกรรมประกาศ

ปัญหาพิเศษนี้สำเร็จได้เพราะความช่วยเหลือของบุคคลต่างๆ ดังนี้

1. อาจารย์ศรัณย์ อินทโกสุม
   ท่านช่วยแนะนำเทคนิคการเขียนโปรแกรมและช่วยแนะนำการจัดทำเอกสารคู่มือปัญหาพิเศษ
2. อาจารย์ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
   ท่านช่วยประสาทวิชาความรู้ต่างๆ ที่นำมาใช้ในการทำปัญหาพิเศษ
3. บิดา มารดา และผู้มีอุปการคุณ
   ท่านเป็นผู้ส่งเสริมและสนับสนุนทางด้านการศึกษา

ขอขอบพระคุณในความกรุณาเป็นอย่างสูง

คณะผู้จัดทำ

สารบัญรูปภาพ

สารบัญ

บทที่ 1

ความสำคัญ/ที่มาของปัญหาพิเศษ

เนื่องจากปัจจุบันคอมพิวเตอร์ เข้ามามีบทบาทในชีวิตประจำวันมากมาย ทั้งด้าน ธุรกิจ การศึกษา การสื่อสาร หรือ แม้แต่วงการบันเทิง ภาพยนตร์โฆษณาทางทีวี ซึ่งปัจจุบันนิยมใช้เทคนิคของการตกแต่งภาพและการสร้างภาพเคลื่อนไหวโดยคอมพิวเตอร์เพื่อกระตุ้นความสนใจของผู้พบเห็น และการสร้างภาพเคลื่อนไหวบนคอมพิวเตอร์เป็นศาสตร์แขนงหนึ่งที่น่าสนใจค้นคว้าเป็นอย่างมาก

ในการสร้างภาพเคลื่อนไหวบนคอมพิวเตอร์นั้นมีเทคนิคที่น่าสนใจมาก ก็คือ เทคนิคการทำมอร์ฟฟิง ซึ่งเป็นเทคนิคใหม่ โดยจะทำการปรับ และสร้างรูปภาพขึ้นมาเพื่อประกอบเป็นภาพเคลื่อนไหว โดยจะพยายามให้เกิดความรู้สึกว่าวัตถุหนึ่งค่อย ๆ แปรเปลี่ยนเป็น อีกวัตถุหนึ่งอย่างค่อยเป็นค่อยไป

วัตถุประสงค์ของปัญหาพิเศษ

- เพื่อให้เข้าใจหลักการทำงานของภาพเคลื่อนไหวบนคอมพิวเตอร์โดยใช้เทคนิคมอร์ฟฟิง
- สร้างโปรแกรมต้นแบบในการจำลองการทำงานมอร์ฟฟิง ที่ใช้กันอยู่ในปัจจุบัน
- เป็นแนวทางของรูปแบบปัญหาพิเศษเพื่อจะใช้ในปีต่อไป

ขั้นตอนในการดำเนินงาน

- ปรับพื้นฐานความรู้เรื่อง การประมวลผลรูปภาพ
- ศึกษาทฤษฎีที่เกี่ยวข้องกับการทำมอร์ฟฟิง
- ศึกษาการจัดเก็บและการเรียกใช้ ของแฟ้มข้อมูลรูปภาพ (Graphic File Format) ต่าง ๆ
- ศึกษาการทำงานของภาพเคลื่อนไหว และการจัดเก็บภาพเคลื่อนไหว
- ออกแบบโปรแกรมต้นแบบ
- ทดสอบการทำงานและปรับปรุงแก้ไขโปรแกรมต้นแบบ

ขอบเขตของปัญหาพิเศษ

- โปรแกรมจะพัฒนาขึ้นด้วยภาษา C++
- โปรแกรมจะทำงานเพื่อสร้างภาพในแต่ละรูป (Frame) ลงไปเก็บในรูปแบบ TGA แบบ 24 บิต (RGB)
- ภาพต้นแบบนั้นจะต้องมีขนาด 320 * 200 ในรูปแบบ TGA แบบ 24 บิต (RGB)
- ผู้ใช้สามารถกำหนดจำนวนรูป (Frame) ในการสร้างภาพเคลื่อนไหวแต่ละชุดได้
- โปรแกรมจะมีการติดต่อกับผู้ใช้ (User Interface) เป็นภาษาไทยและสามารถทำงานด้วยเมาส์ได้ ในรูปแบบกราฟฟิคที่ความละเอียด 800*600 จำนวนสี 256 สี

ประโยชน์ที่คาดว่าจะได้รับ

- ได้โปรแกรมต้นแบบในการทำภาพเคลื่อนไหวโดยใช้เทคนิคมอร์ฟฟิง
- เป็นการนำคณิตศาสตร์มาประยุกต์เพื่อใช้งานจริง
- ได้เครื่องมือเพื่อการพัฒนา โปรแกรมประยุกต์ (Application) ต่อไปในอนาคต
- เป็นแนวทางของรูปแบบปัญหาเพื่อจะใช้ในปีต่อไป

บทที่ 2

Morphing : การเกลื่อนกลืนภาพ

มอร์ฟฟิง (Morphing) เป็นการแสดงภาพที่เปลี่ยนวัตถุหนึ่งให้เป็นอีกวัตถุหนึ่ง ซึ่งเราจะเห็นตัวอย่างของมอร์ฟฟิงได้ทุกหนทุกแห่งทั้งในภาพยนตร์และโทรทัศน์ ผลการมอร์ฟฟิงจะมีแรงผลักดันให้ใช้ประโยชน์ได้เหนือการเปลี่ยนแปลงอย่างมหัศจรรย์ที่เราได้พบเห็นกันมา มอร์ฟฟิงเป็นการย่อคำ ตัดจากคำเดิมว่า Metamorphosing หรือการเปลี่ยนรูปร่าง ซึ่งเป็นคำศัพท์จากสาขาชีววิทยาที่อธิบายกลไกของธรรมชาติที่ทำหน้าที่เปลี่ยนตัวหนอนที่น่าเกลียดน่าชังให้กลายเป็นผีเสื้อแสนสวย หรือเปลี่ยนลูกอ๊อดให้กลายเป็นกบ

มอร์ฟฟิงก็คือ การเกลื่อนกลืนภาพที่มีการจัดเรียงลำดับให้เห็นความเปลี่ยนแปลงทีละเล็กทีละน้อยจนกลายเป็นภาพใหม่ เช่นจากคนกลายเป็นมนุษย์หมาป่า จากค้างคาวกลายเป็นผีดูดเลือด ในเชิงเทคนิคการเกลื่อนกลืนภาพประกอบด้วยอัลกอริทึม 3 อย่าง คือ ทวีนนิ่ง(Tweening),วอร์ปปิง (Warping), และดิสโซลว์ (Dissolve)

ทวีนนิ่ง มาจากคำเต็มว่า In-Betweening หรือการแทรกภาพเพื่อต่อเชื่อมภาพอื่นอีกสองภาพ ถ้าภาพสองภาพนั้นเป็นภาพเดียวกัน เช่นเป็นภาพม้าซึ่งถ่ายจากมุมมองคนละทิศ เมื่อนำมาวางเรียงกันก็เป็นเพียงภาพเคลื่อนไหวธรรมดา แต่ถ้าเปลี่ยนภาพม้าลายให้กลายเป็นเสือโคร่งก็จะกลายเป็นมอร์ฟฟิง

วอร์ปปิง เป็นการเปลี่ยนแปลงรูปร่างจากออบเจกต์ต้นทางไปเป็นออบเจกต์ปลายทางในลักษณะที่มีการดึงยืดขยาย และบีบอัดลงไปในขอบเขตที่กำหนด

ดิสโซลว์ เป็นการเปลี่ยนสีจากออบเจกต์หนึ่งไปเป็นสีของออบเจกต์ปลายทาง ซึ่งก็คือเป็นการเกลื่อนกลืนทางสีนั่นเอง

รูปแบบการสร้างภาพแบบมอร์ฟฟิง

รูปแบบการสร้างภาพแบบมอร์ฟฟิงสามารถแบ่งได้เป็น 2 ประเภท คือ มอร์ฟฟิง 2 มิติ (Image Morphing) และมอร์ฟฟิง 3 มิติ(Object Morphing) มอร์ฟฟิง 2 มิติ เป็นแอพพลิเคชันของดิจิตอลอิมเมจวอร์ปปิง (Digital Image Warping) ซึ่งเกี่ยวกับการดึงยืดขยาย และการบิดรูป

ของภาพเริ่มต้นให้มีรูปร่างเปลี่ยนไปเป็นภาพสุดท้าย ในขณะเดียวกันองค์ประกอบของแต่ละภาพก็ต้องเปลี่ยนจากภาพเริ่มต้นไปเป็นภาพสุดท้าย สำหรับการควบคุมนั้น จะแบ่งภาพออกเป็นส่วนเล็กๆ ซึ่งจะถูกแม็บ (Map) ไปในแต่ละส่วนที่กำหนด สำหรับมอร์ฟฟิง 3 มิติ จะมีความคล้ายคลึงกัน คือ ภาพเริ่มต้นและภาพสุดท้าย จะต้องเป็นรูปร่าง 3 มิติ ความยากในมอร์ฟฟิง 3 มิติ ก็คือการมอร์ฟวัตถุซึ่งมีโครงสร้างแตกต่างกัน ยกตัวอย่างเช่น การมอร์ฟวัตถุรูปทรงโดนัท ไปเป็นวัตถุรูปทรงสี่เหลี่ยมผืนผ้า ปัญหาที่เกิดขึ้นได้แก่ รูของโดนัทและด้านของสี่เหลี่ยมซึ่งมีปัญหาในการมอร์ฟ

1. การประมวลผลแบบอิมเมจมอร์ฟฟิง (มอร์ฟฟิงแบบ 2 มิติ)

เทคนิคสำหรับมอร์ฟฟิงแบบ 2 มิติ ส่วนมากมาจากดิจิตอลอิมเมจวอร์ปปิง (Digital Image Warping) ซึ่งเป็นส่วนหนึ่งของอิมเมจโพรเซสซิง (Image Processing) อิมเมจวอร์ปิง (Image Warping) นี้เป็นเทคนิคหลักในการมอร์ฟฟิง เท็กเจอร์แม็บปิง (Texture Maping) เป็นแอพลิเคชันหนึ่งในหลายๆ แอพลิเคชันพื้นฐานสำหรับอิมเมจวอร์ปปิง วิธีการนี้ทำได้โดยการใช้สามเหลี่ยม หรือช่องตาข่าย โดยนิยามความสัมพันธ์ว่า เท็กเจอร์ (Textures) ในภาพจะเปลี่ยนแปลงไปอย่างกลมกลืนจากภาพเริ่มต้นไปเป็นภาพสุดท้าย

ข้อดีและข้อเสียของการประมวลผลแบบอิมเมจมอร์ฟฟิง

สามารถสร้างภาพได้สะดวก ในกรณีที่ภาพต้นแบบเป็นภาพแบบภาพจริงหรือภาพถ่าย กล่าวคือ ไม่จำเป็นที่จะต้องใช้การแปลงภาพให้กลับมาเป็นแบบเวกเตอร์ หรือแบบรายละเอียดของภาพก่อน

ภาพที่ได้นั้นจะมี รายละเอียดของแสงเงาไม่สมจริง ทั้งนี้ขึ้นอยู่กับภาพต้นทาง ภาพปลายทาง และการกำหนดจุดภาพนั่นเอง

2. การประมวลผลแบบออปเจกต์มอร์ฟิง (มอร์ฟฟิงแบบ 3 มิติ)

การประมวลผลแบบออปเจกต์มอร์ฟฟิง จะเป็นการสร้างภาพทั้งหมดโดยการใช้การคำนวณของคอมพิวเตอร์ โดยระบุรูปร่าง ขนาด สี และ ลักษณะของวัตถุใน 3 มิติ แล้วทำการปรับรายละเอียดของสิ่งที่ถูกนำมาสร้างภาพนี้ ให้เปลี่ยนไปหารูปร่างของวัตถุปลายทางตามที่ระบุไว้ โดยผลลัพธ์ที่ได้ออกมานั้นก็จะเป็นรายละเอียดของภาพที่ถูกสร้างขึ้นระหว่างเฟรมที่ต้องการนั่นเอง แล้วจึงนำรายละเอียดของภาพเหล่านี้ไปสร้างเป็นภาพบนจอภาพเพื่อแสดงผลต่อไป

ภาพอินพุททั้งหมดที่ต้องการ จำเป็นต้องถูกกำหนดในรูปแบบที่เป็น 3 มิติ ซึ่งโดยทั่วไปจะใช้ปริมาณแบบเวกเตอร์เข้ามาประยุกต์ใช้ในการเก็บรูปแบบที่เป็น 3 มิติ และเพื่อเพิ่มความสมจริงของภาพ ก็จะสามารถนำภาพที่สร้างขึ้นมาไปผ่านขั้นตอนการจำลองแสงเงา (Ray Tracing) เนื่องจากผลลัพธ์ที่ได้เป็นรูปแบบ ไม่ใช่รูปภาพของวัตถุนั่นเอง

ข้อดีและข้อเสียของการประมวลผลแบบออปเจกต์มอร์ฟฟิง

สามารถสร้างภาพที่มีรายละเอียดสูง มีความสมจริงและมีรูปแบบของแสงเงาที่ถูกต้อง

ในการประมวลผลแบบออปเจกต์มอร์ฟฟิง ถ้าต้องการให้รูปมีคุณภาพสูง จำเป็นที่จะต้องใช้เวกเตอร์เป็นปริมาณมากในการเก็บภาพ ซึ่งทำให้สิ้นเปลืองหน่วยความจำที่จะใช้ประมวลผลและใช้เวลาในการประมวลผลสูงมาก การประมวลผลแบบออปเจกต์มอร์ฟฟิงนี้ จึงมักไม่ค่อยมีให้พบเห็นนักบนคอมพิวเตอร์ทั่วไป แต่จะพบได้บนคอมพิวเตอร์ที่ทำงานเฉพาะด้าน (Work Station) เท่านั้น

<u>หมายเหตุ</u>

สำหรับในการทำปัญหาพิเศษนี้ เมื่อพิจารณาถึงความเหมาะสมในด้านต่าง ๆ คือ ภาพต้นแบบที่สามารถหาได้ในรูปแบบทีจีเอ ประกอบกับอุปกรณ์ต่าง ๆ ที่มีอยู่ จึงเลือกทำแบบอิมเมจมอร์ฟฟิง

3. อัลกอริทึมของการทวีนนิ่ง

หลักการของการทวีนนิ่งนี้จะอาศัยการแทรก (Interpolation) จุดยอดต่าง ๆ ของรูปหลายเหลี่ยมที่เป็นตารางควบคุมซึ่งอยู่ในภาพต้นทางและภาพปลายทาง ในการทวีนแต่ละครั้ง ก็จะได้จุดยอดของรูปเหลี่ยมชุดใหม่เกิดขึ้น ซึ่งจะมีจำนวนจุดยอดเท่ากับจำนวนจุดยอดในภาพต้นทางและปลายทาง ส่วนจุดยอดใหม่ที่คำนวณได้จะอยู่ระหว่างจุดยอดของภาพต้นทางและปลายทาง และขึ้นอยู่กับค่าถ่วงน้ำหนัก (Weight) ในการทวีนแต่ละครั้ง

ในการกำหนดจำนวนครั้งในการทวีน ขึ้นอยู่กับการกำหนดจำนวนเฟรมทั้งหมดที่ต้องการให้เป็นชุดของภาพเคลื่อนไหว โดยในที่นี้จะมีการรวมเอาภาพต้นแบบมาเป็นภาพแรกและภาพสุดท้ายของชุดภาพเคลื่อนไหว ดังนั้นจำนวนครั้งในการทวีนจะเป็น จำนวนเฟรมทั้งหมด - 2

ขั้นตอนในการทวีนนิ่งคือ

1. ตรวจสอบจำนวนจุดยอดในภาพต้นทางและภาพปลายทางว่ามีจำนวนเท่ากันหรือไม่ ถ้าเท่ากัน ก็ทำข้อ 2. ต่อไป ถ้าไม่เท่าก็ออกไป
2. กำหนดจำนวนครั้งที่ต้องการทวีน นั่นคือกำหนดจำนวนเฟรมที่ต้องการนั่นเอง
3. คำนวณหาชุดของจุดยอดที่เป็นผลจากการทวีนนิ่งในแต่ละครั้ง ดังนี้

$$TP_i.x = S.x + \frac{(i*(D.x - S.x))}{N+1}$$

$$TP_i.y = S.y + \frac{(i*(D.y - S.y))}{N+1}$$

โดยที่

$TP_i.x, TP_i.y$	คือ	จุดที่เป็นผลจากการทวีนจุดยอดลำดับที่ i
$S.x, S.y$	คือ	จุดยอดของภาพต้นทาง
$D.x, D.y$	คือ	จุดยอดของภาพปลายทาง
N	คือ	จำนวนครั้งในการทวีน

4. การวอร์ป

4.1 ประวัติของวอร์ปปิง

ประวัติของวอร์ปปิงจะย้อนกลับไปยังโครงการอวกาศในทศวรรษที่ 60 เมื่อนาซาได้ถ่ายภาพโลกเอาไว้ แต่เมื่อนำภาพมาต่อกันเข้านาซาพบว่าต่อกันให้ถูกต้องไม่ได้เลย เพราะว่าความแตกต่างในเรื่องความสูง มุม เวลา และการมองเวลาถ่ายภาพแต่ละครั้ง จากผลนี้นักวิจัยที่นาซาได้พัฒนาอัลกอริทึมที่ปฏิบัติต่อข้อมูลดิจิตอลเสมือนเป็นจุดต่างๆ บนพื้นผิวที่ สามารถยืดออกเพื่อให้เข้ากับเซตของจุดที่อ้างอิงได้ และเรียกวิธีนี้ว่าอิมเมจรีจิสเตชัน (Image Registration) ภาพที่ได้จึงได้มาจากการยืดขยายและดึงให้มาปะติดปะต่อเป็นภาพของโลก จุดนี้เป็นจุดเริ่มต้นอันยิ่งใหญ่ของรูทีน วอร์ปปิง

วอร์ปปิง ได้รับการปรับปรุงในทศวรรษที่ 70 เมื่อยาน ไวกิ้ง 2 (Viking 2) ลงบนดาวอังคาร โชคไม่ดีที่ยานไวกิ้งลงเป็นมุม และกล้องจับลงด้านล่าง ภาพที่ได้มาดูเหมือนเหวใหญ่ มีขอบฟ้าโค้งมาก ภาพทั้งหมดของดาวอังคารจึงได้นำมาผลักดันผ่านอัลกอริทึมของ วอร์ปปิง เพื่อแก้ไขให้ถูกต้องก่อนจะออกแสดงต่อประชาชน

มีการใช้วอร์ปปิงทางการแพทย์ด้วย ในวิธีการที่เรียกว่า ดิจิตอลสับแทร็กชั่นแองจิโอกราฟฟี (Digital Subtraction Angiography) หรือการถ่ายภาพระบบเส้นโลหิต โดยการลบภาพเชิงดิจิตอล วิธีการทำงานจะใช้ภาพรังสีเอ็กซ์ของคนไข้สองชุด ชุดแรกถ่ายก่อน และชุดหลังถ่ายหลังจากฉีดสีย้อมเข้าไปในเส้นเลือด สีย้อมไม่ยอมให้รังสีเอกซ์ส่องผ่าน ระบบโลหิตจะกลายเป็นสีดำสนิทบนแผ่นฟิล์ม แผ่นฟิล์มแรกจะนำมาซ้อนแผ่นหลังเพื่อลบภาพรุงรัง เช่น กระดูก อวัยวะต่างๆ เหลือไว้เพียงระบบเส้นโลหิตเท่านั้น ปัญหาที่ตามมาก็คือ เทคนิคจะใช้งานได้ก็ต่อเมื่อคนไข้ไม่ขยับเขยื้อน ซึ่งก็เป็นเรื่องที่เป็นไปไม่ได้เนื่องจากคนไข้ยังหายใจอยู่ตลอดเวลา ช่างเทคนิคพิเศษจำเป็นต้องดึงยืดขยายโดยกำหนดเครื่องหมายเด่นบนพื้นที่ให้ตรงกัน เมื่อนั้นก็จะได้ภาพระบบเส้นโลหิตในร่างกายของคนไข้

4.2 อัลกอริทึมของการวอร์ปปิง

เป็นการเปลี่ยนแปลงรูปร่างของวัตถุต้นทางไปเป็นรูปร่างของวัตถุปลายทาง ซึ่งจะเป็นการทำให้ภาพค่อย ๆ เพี้ยนไปจากเดิมในแต่ละเฟรมทีละน้อย ๆ สำหรับการเปลี่ยนแปลงที่เกิดขึ้นนี้จะเป็นไปในลักษณะที่จุดของเฟรมแรกถูกย้ายไปให้อยู่ในตำแหน่งใหม่ในเฟรมสุดท้าย ซึ่งก็คือความหมายของคำว่าวอร์ปปิงนั่นเอง วิธีหนึ่งในการทำวอร์ปปิงบนคอมพิวเตอร์ ก็คือ การแบ่งภาพออกเป็นชิ้นส่วนของรูปหลายเหลี่ยม (polygon) เล็ก ๆ ซึ่งรูปหลายเหลี่ยมแต่ละรูปก็จะถูกสร้างขึ้นจากจุดโคออร์ดิเนตที่ลากต่อกันเป็นรูปหลายเหลี่ยมนั้น เมื่อรูปหลายเหลี่ยมและจุดยอดของมันมีการย้ายตำแหน่ง จุดต่าง ๆ ที่อยู่ภายในรูปหลายเหลี่ยมก็จะถูกย้ายเช่นกัน และจำนวนของพิกเซลที่อยู่ในรูปหลายเหลี่ยมก็จะเปลี่ยนแปลงไปด้วยและอัลกอริทึมนี้จะใช้ Triangle Mapping แม็บจุดต่าง ๆ ที่อยู่ภายในรูปหลายเหลี่ยมจากภาพต้นทางไปยังภาพปลายทาง

หลักการของ Triangle Mapping

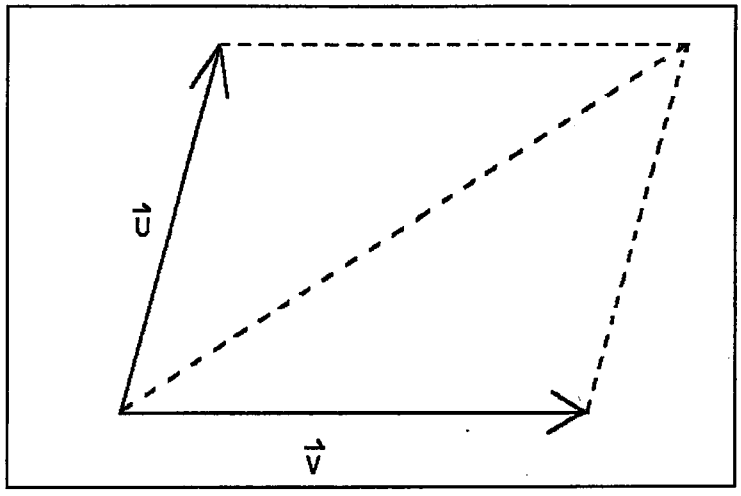
เนื่องจากเราไม่สามารถที่จะกำหนดได้ว่าจะแม็บจุดใดในภาพต้นทางให้ไปเป็นจุดในภาพปลายทางที่ต้องการ แต่ทราบแน่ ๆ ว่าจุดที่ต้องการในภาพปลายทางเป็นจุดใด ดังนั้นในการแม็บจุด ก็จะทำในลักษณะที่กลับกัน ก็คือ ทราบจุดในภาพปลายทาง แล้วแม็บไปยังภาพต้นทางเพื่อหาว่าจุดใดในภาพต้นทางที่จะแม็บไปยังภาพปลายทาง

ขั้นตอนในการแม็บ

1. กำหนดขอบเขตของรูปสามเหลี่ยมในภาพต้นทางและในภาพปลายทาง
   - ในภาพต้นทางประกอบด้วยจุดยอด 3 จุด คือ S[0] , S[1] , S[2]
   - ในภาพปลายทางประกอบด้วยจุดยอด 3 จุด คือ d[0] , d[1] , d[2]

2. คำนวณหาพื้นที่ทั้งหมดในภาพปลายทาง d[0] , d[1] , d[2] โดยใช้หลักการของ การคูณเวกเตอร์แบบครอส-โปรดัก (cross product) กำหนดเวกเตอร์ $\vec{U}$ และ $\vec{V}$ เป็นเวกเตอร์ใด ๆ ทำการครอสระหว่างเวกเตอร์ 2 เวกเตอร์ $(\vec{U}\times\vec{V})$ ดังนั้น

$\|\vec{U}\times\vec{V}\|$ = พื้นที่สี่เหลี่ยมด้านขนานที่มีด้านคู่ขนานแทนด้วยเวกเตอร์ $\vec{U}$ และ $\vec{V}$ ดังรูปที่ 2.1



รูปที่ 2.1 แสดงการครอสโปรดัก ระหว่าง $\vec{U}$ และ $\vec{V}$

ดังนั้น ถ้าต้องการพื้นที่ ของรูปสามเหลี่ยมก็จะได้ว่า แบ่งพื้นที่สี่เหลี่ยมด้านขนานที่ได้จากการครอสเวกเตอร์ด้วยเส้นทแยงมุมก็จะได้พื้นที่สามเหลี่ยม 2 รูปที่มีพื้นที่เท่ากัน จะได้

$$\text{พื้นที่สามเหลี่ยม} = \frac{\|\vec{U}\times\vec{V}\|}{2}$$

กำหนด $\vec{U}=(u_1,u_2,u_3)$; $\vec{V}=(v_1,v_2,v_3)$

แต่เนื่องจากทำในระบบ 2 มิติ ดังนั้น u_3 และ v_3 เท่ากับ 0 ดังนั้น

จากความรู้เรื่องค่ากำหนด (determinant) สามารถหา ค่า $\vec{U}\times\vec{V}$ ได้โดย

$$\vec{U}\times\vec{V}=\begin{vmatrix} i & j & k \\ u_1 & u_2 & 0 \\ v_1 & v_2 & 0 \end{vmatrix}$$

$$=(u_1v_2-u_2v_1)\vec{k}$$

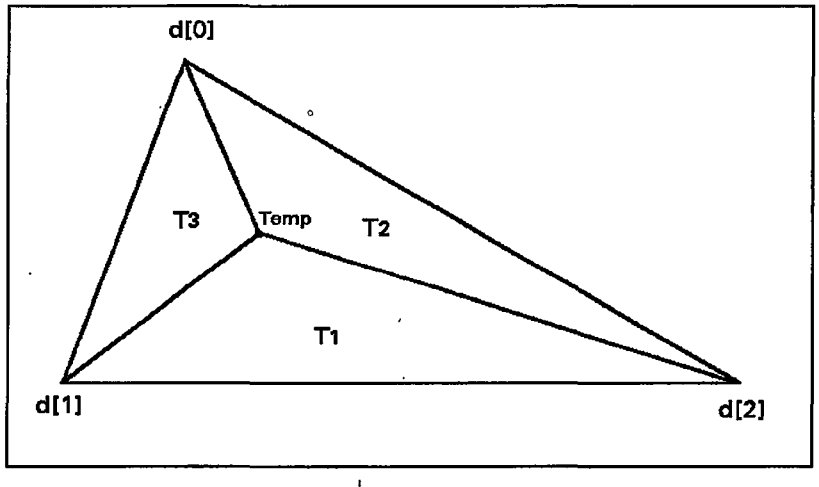
$$\|\vec{U}\times\vec{V}\|=(u_1v_2-u_2v_1)$$

พื้นที่สามเหลี่ยม = $\frac{(u_1 v_2 - u_2 v_1)}{2}$

ในที่นี้แทน $\vec{U}$ = (d[0] , d[1])

$\vec{V}$ = (d[2] , d[1])

3. หาค่าถ่วงน้ำหนัก ณ แต่ละจุดที่พิจารณาในภาพปลายทางว่าจุดที่พิจารณานั้นอยู่ใกล้ไกลจากจุดอ้างอิง d[0] , d[1] , d[2] เพียงไร โดยคิดเป็นอัตราส่วนระหว่างพื้นที่สามเหลี่ยมที่พิจารณากับพื้นที่สามเหลี่ยมทั้งหมดในภาพปลายทางดังนี้จุดพิจารณาให้เป็น Temp โดยการหาพื้นที่สามเหลี่ยมโดยใช้หลักการของครอส-โปรดักของเวกเตอร์ เช่นเดิม ซึ่งจะได้อัตราส่วนของพื้นที่สามเหลี่ยม 3 ส่วนที่รวมกันได้ 1 แสดงได้ดังนี้



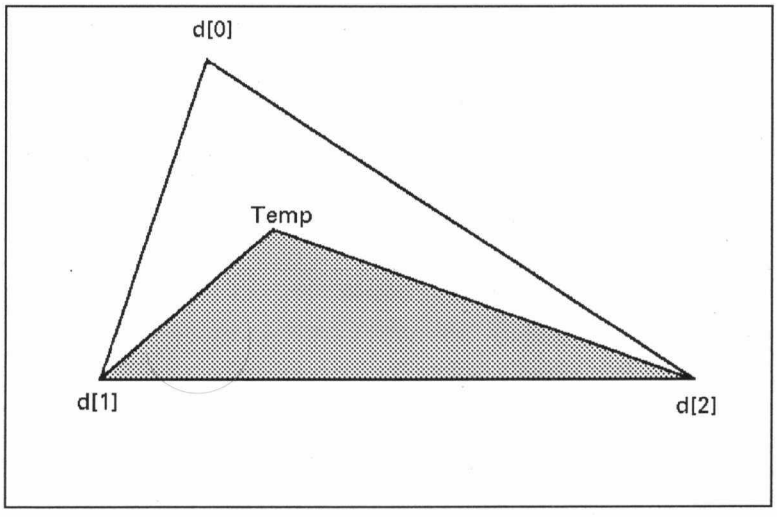
รูปที่ 2.2

จากรูปที่ 2.2 T1 + T2 + T3 = พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

a1 = พื้นที่สามเหลี่ยมของ (Temp,d[1],d[2]) / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

= T1 / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

ดังรูปที่ 2.3

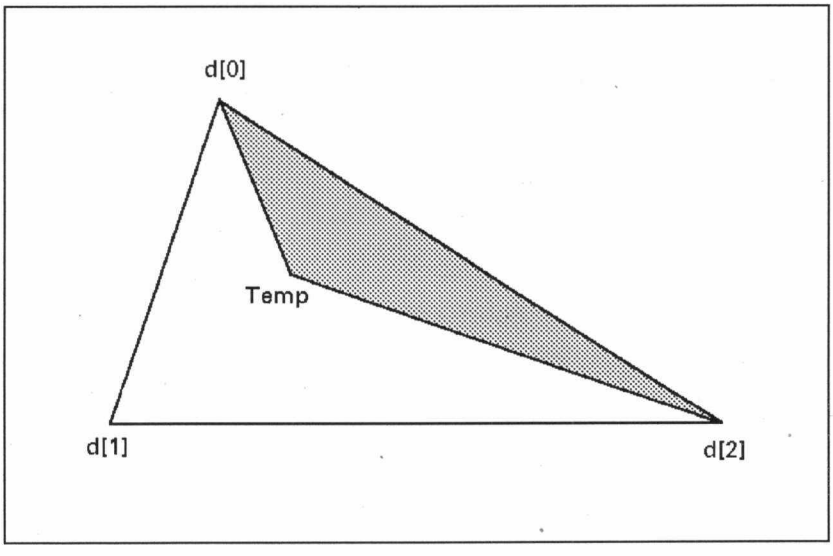


รูปที่ 2.3 รูปแสดงพื้นที่สามเหลี่ยม (Temp,d[1],d[2])

a2 = พื้นที่สามเหลี่ยมของ (Temp,d[0],d[2]) / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

= T2 / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

ดังรูปที่ 2.4

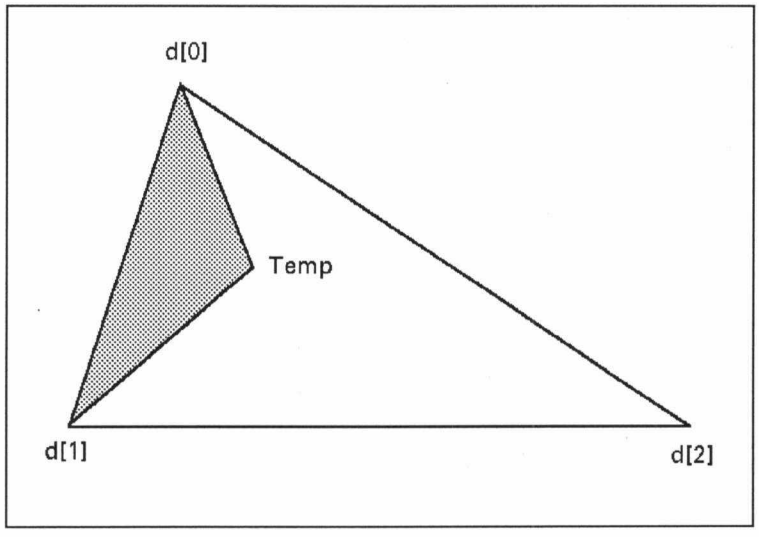


รูปที่ 2.4 รูปแสดงพื้นที่สามเหลี่ยม (Temp,d[0],d[2])

a3 = พื้นที่สามเหลี่ยมของ (Temp,d[0],d[1]) / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

= T3 / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

ดังรูปที่ 2.5



รูปที่ 2.5 รูปแสดงพื้นที่ (Temp,d[0],d[1])

ดังนั้นจะได้ว่า

ถ้า จุดที่พิจารณานั้นอยู่ใกล้จุด d[0] จะได้ว่า a1 มีค่าใกล้ ๆ 1

ถ้า จุดที่พิจารณานั้นอยู่ใกล้จุด d[1] จะได้ว่า a2 มีค่าใกล้ ๆ 1

ถ้า จุดที่พิจารณานั้นอยู่ใกล้จุด d[2] จะได้ว่า a3 มีค่าใกล้ ๆ 1 และ

a1+a2+a3 = (T1 + T2 + T3) / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

= พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2]) / พื้นที่สามเหลี่ยมทั้งหมด (d[0],d[1],d[2])

= 1

4. คำนวณหาจุดที่อยู่ในภาพต้นทาง ซึ่งจุดเหล่านั้นจะอยู่ห่างจากจุด S[0] ,S[1] ,S[2] ด้วยอัตาส่วน a1 , a2 , a3 ตามลำดับ ดังนี้

$$newpoint.x = \left(\left(S[0].x * a_1\right) + \left(S[1].x * a_2\right) + \left(S[2].x * a_3\right)\right)$$
$$newpoint.y = \left(\left(S[0].y * a_1\right) + \left(S[1].y * a_2\right) + \left(S[2].y * a_3\right)\right)$$

5. อัลกอริทึมของดิสโซวล์

ดิสโซวล์ซึ่งเป็นการเชื่อมโยงภาพทั้งสองที่ทำวอร์ปปิงและทวีนนิ่งให้เป็นภาพเดียว ก็คือ การผสมสีของภาพทั้งสองเข้าด้วยกันในระบบสี 24 บิต การทำดิสโซวล์เป็นการกระทำกับค่าของ RGB ของแต่ละพิกเซลในแต่ละเฟรมของลำดับการเปลี่ยนแปลงนั้นค่า RGB ของพิกเซลต่าง ๆ จะเป็นอิสระจากกันในแต่ละเฟรม ซึ่งอยู่ระหว่างเฟรมเริ่มต้นและเฟรมสุดท้าย โดยค่าของ RGB ของแต่ละพิกเซล จะได้รับการคำนวณแบบเฉลี่ยน้ำหนักระหว่างค่า RGB ของพิกเซลที่ได้จากเฟรมแรกและเฟรมสุดท้าย โดยที่การคำนวณแบบถ่วงน้ำหนักนี้จะเป็นเปอร์เซนต์ของความสมบูรณ์ของการเปลี่ยนแปลงนั้น ซึ่งจะสามารถคำนวณค่าสีที่ใหม่ที่ได้จากการดิสโซวล์ ดังนี้

$$dest[i] = \left(p_1[i] * \left(1 - factor\right)\right) + \left(p_2[i] * factor\right).$$

โดยที่ $dest[i]$ เป็น ค่าสีของพิกเซลที่ i ที่คำนวณได้

$p_1[i]$ เป็น ค่าสีของพิกเซลที่ i ของรูปต้นทาง

$p_2[i]$ เป็น ค่าสีของพิกเซลที่ i ของรูปปลายทาง

$factor$ เป็น ลำดับของเฟรมที่กำลังทำงาน / (จำนวนเฟรมทั้งหมด - 1)

และทำการคำนวณเช่นเดียวกันนี้กับทุกองค์ประกอบของสี ได้แก่ สีแดง สีเขียว และ สีน้ำเงิน ตัวอย่างเช่น ในเฟรมซึ่งอยู่ตรงกลางระหว่างเฟรมเริ่มต้นและเฟรมสุดท้าย ค่าขององค์ประกอบที่เป็นสีแดงของแต่ละพิกเซลจะได้รับการคำนวณโดยคูณองค์ประกอบที่เป็นสีแดงของพิกเซลนั้นในเฟรมแรกด้วย 0.5 และคูณองค์ประกอบสีแดงของพิกเซล นั้นในเฟรมสุดท้ายด้วย 0.5 และนำค่าทั้งสองบวกเข้าด้วยกัน

สมมติกำหนดจำนวนเฟรมทั้งหมดไว้ 6 เฟรม สำหรับเฟรมซึ่งเกิดขึ้นในลำดับที่ 1 ของเฟรมทั้งหมดก็จะใช้แฟกเตอร์ในการคูณเข้าองค์ประกอบของสีในเฟรมแรกและเฟรมสุดท้ายด้วย 0.75 และ 0.25 ตามลำดับ ด้วยวิธีการคำนวณเดียวกันนี้ จะต้องนำไปคำนวณกับค่าขององค์ประกอบสีเขียว และสีน้ำเงินสำหรับข้อมูลทั้งหมดที่ต้องแสดงขึ้นในเฟรมที่อยู่ระหว่างเฟรมแรกและเฟรมสุดท้าย ดังนั้น เฟรมที่อยู่ใกล้กับเฟรมเริ่มต้น หรือเฟรมที่อยู่ใกล้กับเฟรมสุดท้ายก็จะมีลักษณะของภาพที่คล้ายกับเฟรมเริ่มต้นและเฟรมสุดท้ายตามลำดับ

6. ขั้นตอนในการมอร์ฟ

1. เริ่มต้นจากมีภาพต้นแบบ 2 ภาพ คือภาพต้นทางและปลายทาง

2. กำหนดตารางควบคุมของภาพต้นทางและภาพปลายทางจากนั้นกำหนดจำนวนภาพทั้งหมดภาพทั้งหมดที่ต้องการในการสร้างชุดภาพเคลื่อนไหว

3. ทำการทวีนหาจุดยอดของตารางควบคุมตามจำนวนที่กำหนด

4. ทำการวอร์ปภาพต้นทางและภาพปลายทางด้วยตารางควบคุมเดียวกัน จากการวอร์ปนี้จะได้ภาพภาพ 2 ภาพ ที่มีลักษณะเหมือนกันทุกประการ แต่ต่างกันที่สี

4. นำภาพที่ได้จากการทำการวอร์ปทั้ง 2 ภาพนี้ มาทำการดิสโซลว์กัน ซึ่งเมื่อถึงขั้นตอนนี้ก็จะได้ภาพเอาพุทที่จะนำมาประกอบกันเป็นชุดของภาพเคลื่อนไหว

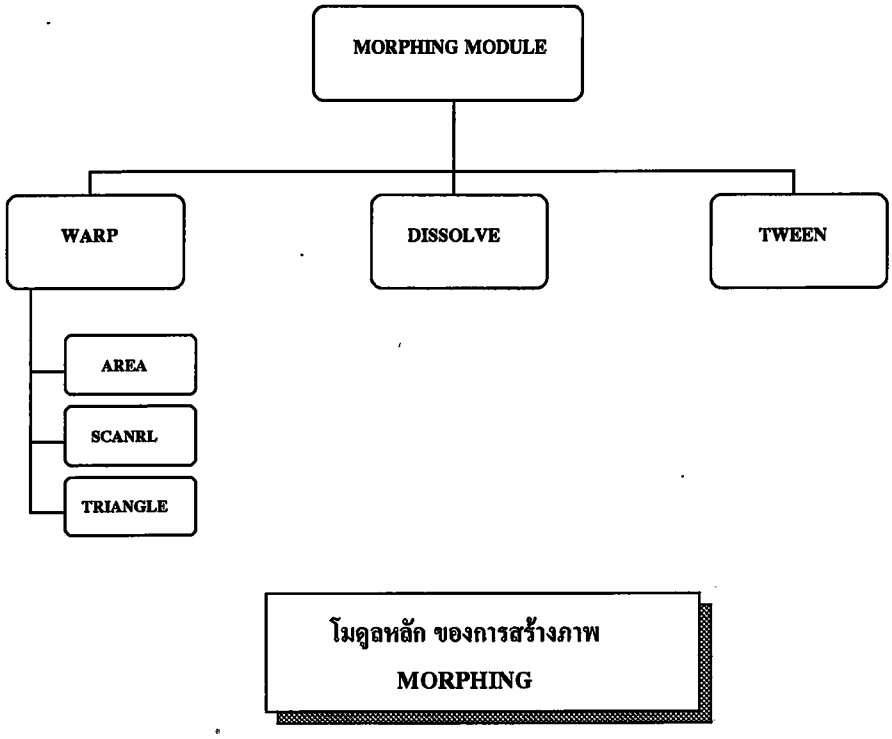
5. ทำเช่นเดียวกับข้อ 3 - 5 ไปเรื่อย ๆ จนครบตามจำนวนที่กำหนด

บทที่ 3

การออกแบบระบบและการติดต่อกับผู้ใช้

การออกแบบระบบ

โปรแกรมการสร้างภาพมอร์ฟฟิงนี้ ได้ถูกออกแบบไว้โดยแยกกันเป็นสองส่วน คือ ส่วนของการสร้างภาพ และส่วนที่ติดต่อกับผู้ใช้ โดยในส่วนของการสร้างภาพนั้นจะประกอบไปด้วยโมดูล(MODULE) หลักๆ ดังนี้



รูปที่ 3.1 รูปแสดงโมดูลหลัก ของการสร้างภาพ มอร์ฟฟิง

ซึ่งแต่ละโมดูลมีหน้าที่การทำงานดังนี้

- *void Warp(PICTURE *s,PICTURE *d,MESH *sm,MESH *dm);*

  ซึ่งเป็นโมดูลหลักของการสร้างภาพในขั้นตอนการวอร์ป

- *void dissolve(PICTURE *a,PICTURE *b,PICTURE *c,int now,int all);*

  ซึ่งเป็นโมดูลหลักของการสร้างภาพในขั้นตอนการดิสโซล์ว

- *void tween(char *filename1,char *filename2,int amount);*

  ซึ่งเป็นโมดูลหลักของการสร้างชุดกรอบภาพ สำหรับการมอร์ฟ

- *float area(POINT *a,POINT *b,POINT *c);*

  เป็นฟังก์ชันสำหรับการหาพื้นที่สามเหลี่ยม

- *void scanRL(POINT p[3],int Left[200],int Right[200]);*

  เป็นฟังก์ชันสำหรับหาจุดขอบซ้ายขวาของสามเหลี่ยม

*void TriangleMap(POINT *,POINT *,unsigned char *spic[3],byte *dpic[3],int,int);*

เป็นฟังก์ชันสำหรับวอร์ปรูปในสามเหลี่ยมย่อยๆ

และในโปรแกรมส่วนการติดต่อกับผู้ใช้นั้นได้มีการออกแบบโดยใช้หลักการการออกแบบเชิงวัตถุ(Object-Oriented Design) โดยมีคลาส(Class) ที่สำคัญดังนี้

- *EVENT*เป็นคลาสของการควบคุมการติดต่อกับผู้ใช้
- *EVENT_HELP* เป็นคลาสลูก(Derived class) ของ EVENT เป็นคลาสของการควบคุมการติดต่อ กับผู้ใช้ที่มีข้อความช่วยเหลือ
- *FONT* เป็นคลาสของตัวอักษรภาษาไทย เพื่อใช้ในการแสดงผล
- *BUTTON* เป็นคลาสของปุ่ม(Button) เพื่อการรับคำสั่ง
- *TBUTTON* เป็นคลาสลูกของคลาส BUTTON แต่มีการแสดงข้อความบนปุ่มเป็นภาษาไทย

และมีคลาสที่ใช้สำหรับการคำนวณเพื่อสร้างภาพ และการทำงานอื่นๆอีกดังนี้

- *memBlock* เป็นคลาสของบล็อคหน่วยความจำ ซึ่งอาจจะขอบริการจาก หน่วยความจำยืดขยาย (Extended memory) หรือ หน่วยความจำสมือน(Virtual memory)ก็ได้
- *PICTURE* เป็นคลาสของรูปภาพ ในรูปแบบการเก็บแบบแม่สี(R G B)
- *MESH* เป็นคลาสของตารางกรอบภาพ

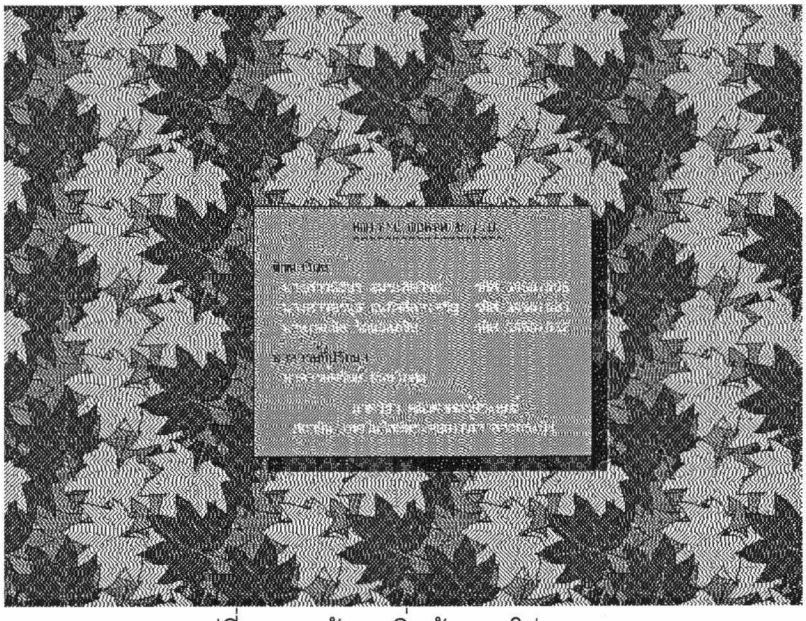
ส่วนการติดต่อผู้ใช้

ในการเรียกใช้งานโปรแกรมนั้น ผู้ใช้จะต้องทำงานอยู่บนระบบปฏิบัติการดอส 3.30 ขึ้นไป และ มีหน่วยความจำยืดขยายว่างไม่น้อยกว่า 1 ล้านตัวอักษร และเรียกใช้

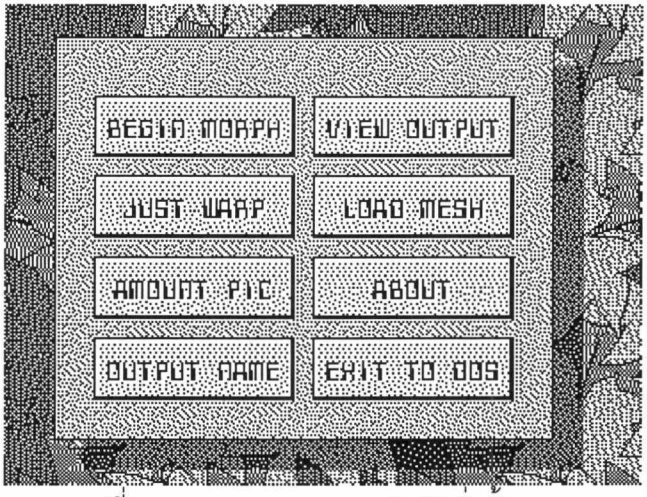
```
C:IMORPH>MORPH <FILENAME1.TGA> <FILENAME2.TGA>

โดย    <FILENAME1.TGA> คือชื่อของแฟ้มภาพที่เป็นภาพตั้งต้น ต้นทาง
       <FILENAME2.TGA> คือชื่อของแฟ้มภาพที่เป็นภาพตั้งต้น ปลายทาง
```

หมายเหตุ ภายในซับไดเรกเตอรี MORPH จะต้องมีแฟ้ม LEAVES.BMP , SMALL?.FON และ SVGA256.BGI บรรจุอยู่ด้วย

ในส่วนของการติดต่อกับผู้ใช้นั้น เป็นการติดต่อในรูปแบบกราฟฟิค(Graphic User Interface) ที่ออกแบบขึ้นเอง โดยจะสั่งงานต่างๆจะใช้เมาส์ หรือคีย์บอร์ดก็ได้ โดยการสั่งงานจะทำการกด(Click) เมาส์ที่ตำแหน่งปุ่มต่างๆที่สร้างไว้ โดยจะมีข้อความช่วยเหลือภาษาไทยแสดงไว้ในส่วนท้ายของจอเพื่อเป็นการอธิบายทำงานของปุ่มนั้นๆ ดังรูปด้านล่าง



รูปที่ 3.2 หน้าจอเริ่มต้นของโปรแกรม



รูปที่ 3.3 ภาพขยายแสดงปุ่มรับสั่งทั้งหมด

จากรูปที่ 3.3 แสดงปุ่มรับคำสั่งทั้งหมด โดยแต่ละปุ่มมีหน้าที่ดังนี้

ข้อความบนปุ่ม	**หน้าที่ของปุ่ม**
BEGIN MORPH	เริ่มต้นทำการ MORPH ภาพ ตามค่าตัวแปรที่ตั้งไว้
JUST WARP	เริ่มต้นทำการ WARP ภาพ ตามค่าตัวแปรที่ตั้งไว้
AMOUNT PIC	เปลี่ยนปริมาณของชุดภาพทั้งหมด ที่จะทำการสร้าง
OUTPUT PREFIX	เปลี่ยนชื่อขึ้นต้นของแฟ้มภาพผลลัพธ์
VIEW OUTPUT	แสดงชุดภาพผลลัพธ์ที่ได้สร้างขึ้น
LOAD MESH	ทำการกำหนดชื่อตารางควบคุม
ABOUT	แสดงรายชื่อผู้จัดทำ และอาจารย์ที่ปรึกษา
QUIT	สิ้นสุดการทำงาน และกลับสู่ดอส

เมื่อจะทำการเลือกการทำงานตามปุ่มใดๆนั้น สามารถกระทำได้สองทาง ก็คือ

- ทำการเลื่อนเมาส์ ให้ลูกศรบนจอภาพทับลงบนจุดนั้นๆ แล้วทำการกดปุ่มซ้ายมือบนเมาส์
- ทำการกดคีย์ด่วน(HOT KEY) โดยกดตัวอักษรที่แสดงเป็นสีสว่างบนปุ่มนั้นๆ

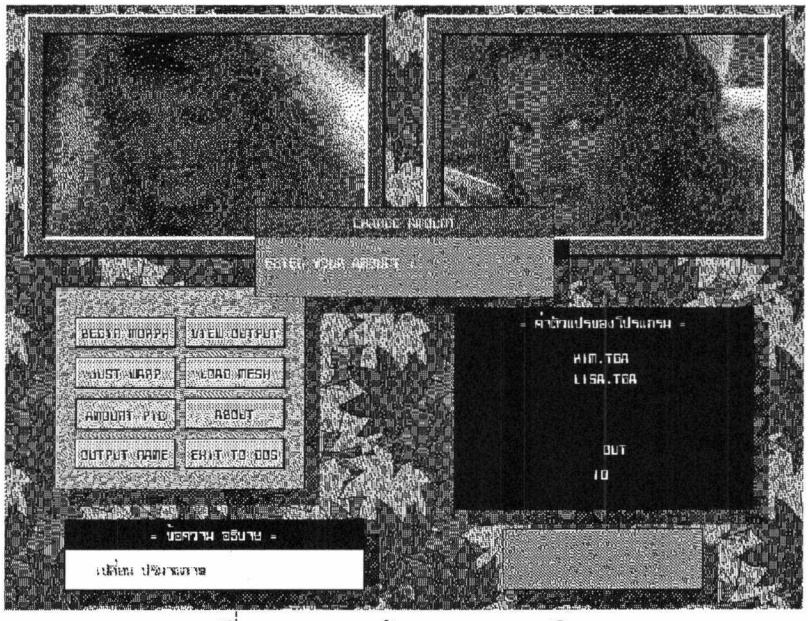
เมื่อมีการเลือกปุ่ม BEGIN MORPH ซึ่งเป็นการสั่งให้ทำการเริ่มต้น MORPH แล้วจะมีหน้าต่างใหม่แสดงขึ้นมาดังรูป

รูปที่ 3.4 แสดงหน้าจอ ขณะที่ทำการคำนวณ MORPH

ในขณะที่ทำการ MORPH นั้น จะมีหน้าต่างเพื่อแสดงรูปสุดท้ายที่ได้คำนวณผ่านไป อยู่ที่กลางหน้าจอ(สังเกตได้ที่ หัวของหน้าต่างจะแสดงลำดับของเฟรม) และสามารถยกเลิกการคำนวณนี้ได้โดยการกด <ESC>

ส่วนในการเลือกคำสั่ง JUST WARP นั้นการทำงานก็จะคล้ายคลึงกับการทำการ MORPH ต่างกันเพียงผลลัพธ์ของรูปที่ได้ จะเป็นรูปที่ผ่านเพียงขั้นตอนการวอร์ปเท่านั้น ดังนั้นจึงขอละที่จะกล่าวถึงการใช้งานไว้

การเลือกคำสั่ง AMOUNT PIC ก็จะมีหน้าต่างขึ้นมาให้กรอกจำนวนเฟรมที่ต้องการโดยจะกรอกได้นั้งแต่ 3 ถึง 32000 ดังรูป



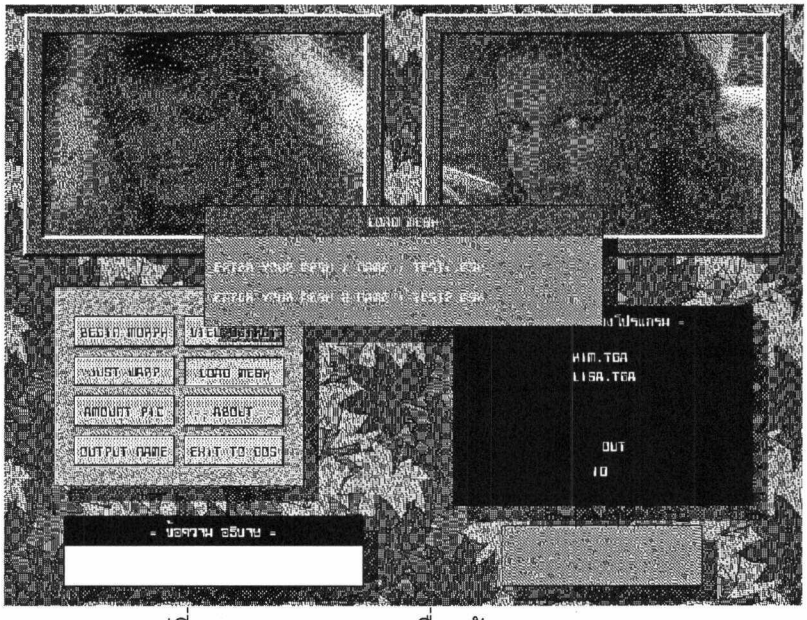
รูปที่ 3.5 แสดงหน้าจอการถามปริมาณภาพ

เมื่อมีการเลือกการ OUTPUT NAME นั้น จะมีหน้าต่างเพื่อให้กรอกชื่อขึ้นต้นของภาพผลลัพธ์ ซึ่งไม่ควรจะมีความยาวเกิน 5 ตัวอักษร โดยที่ภาพผลลัพธ์นั้นจะมีชื่อแฟ้มเป็น OUT001.TGA, OUT002.TGA, OUT003.TGA, ... จนครบจำนวนภาพที่ต้องการ

รูปที่ 3.6 แสดงการถามชื่อขึ้นต้นภาพผลลัพธ์

เมื่อมีการกดคีย์ VIEW OUTPUT ก็จะเป็นการแสดงชุดภาพที่สร้างขึ้น ในลักษณะภาพต่อเนื่อง โดยจะแสดงเป็นภาพขาวดำ เพื่อความรวดเร็วในการคำนวณและแสดงผล

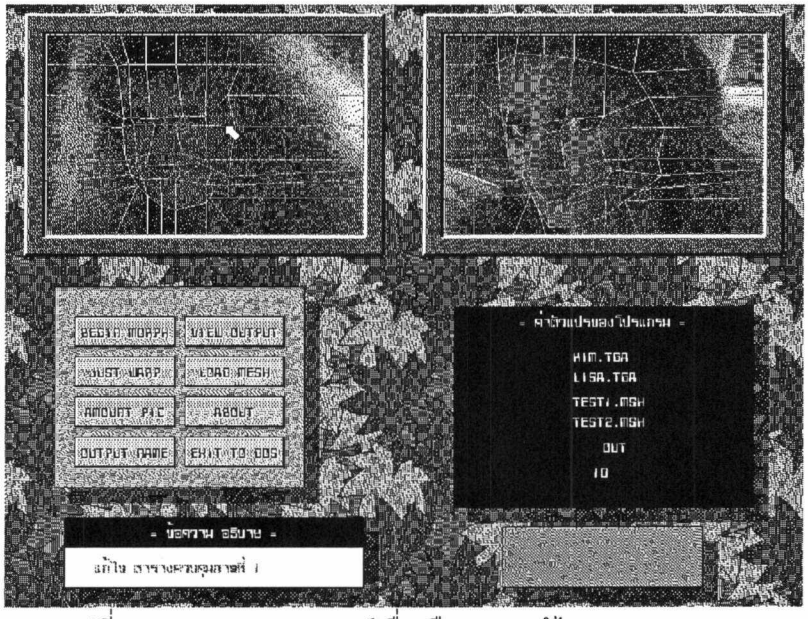
เมื่อมีการกดคีย์ LOAD MESH ก็จะเป็นการทำการกำหนดชื่อแฟ้มของตารางควบคุมใหม่ โดยที่ ถ้ามีแฟ้มข้อมูลที่ชื่อตรงกับชื่อที่กำหนดให้แล้วก็จะทำการอ่านแฟ้มที่มีอยู่นั้นมาแก้ไขต่อ แต่ถ้าไม่มีแฟ้มข้อมูลที่ชื่อตรงกับชื่อที่กำหนดให้แล้วก็จะทำการสร้างแฟ้มตารางควบคุมนั้นขึ้นใหม่ โดยจะมีหน้าต่างแสดงผลดังรูป



รูปที่ 3.7 แสดงการกรอกชื่อแฟ้มตารางควบคุม

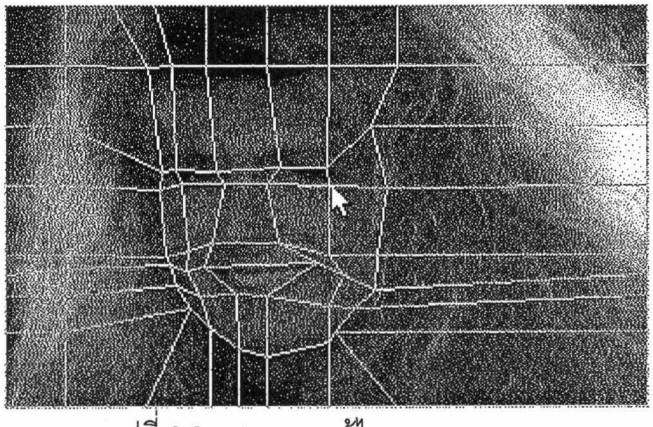
สำหรับการเลือกคำสั่ง ABOUT ก็จะเป็นการแสดงหน้าจอเริ่มต้นอีกครั้งหนึ่ง และการเลือกคำสั่ง QUIT ก็จะเป็นการเลิกการทำงาน และกลับไปยังระบบปฏิบัติการ

ส่วนการ แก้ไขตารางควบคุมนั้นสามารถทำได้โดยการเลื่อนเมาส์ไปทับที่รูปที่ต้องการจะทำการแก้ไขตาราง แล้วกดปุ่มซ้ายบนเมาส์ดังรูปด้านล่าง



รูปที่ 3.8 แสดงการกดเมาส์เพื่อเลือกการแก้ไขตารางควบคุม

หลังจากที่ทำการกดปุ่มซ้ายของเมาส์ที่บริเวณรูปแล้ว หน้าจอก็จะแปลี่ยนเป็นหน้าจอที่ใช้สำหรับการแก้ไขตารางควบคุมดังรูปด้านล่าง



รูปที่ 3.9 แสดงการแก้ไขตารางควบคุม

โดยวิธีการแก้ไขตารางทำได้โดย การกดปุ่มซ้ายของเมาส์ขณะที่ลูกศรทับอยู่บนจุดตัดของตารางใดๆ และขยับเมาส์ไปยังจุดที่ต้องการแล้วจึงจะปล่อยปุ่มซ้าย และเพิ่มเส้นของตารางโดยการกดปุ่มขวาของเมาส์ขณะที่ลูกศรทับอยู่บนเส้นที่ขอบภาพทางด้านบนและด้านซ้ายมือ เพื่อเป็นการเพิ่มเส้นตามแนวตั้งและแนวนอนตามลำดับ

และกดคีย์บอร์ด ปุ่ม 'Q' หรือ <ESC> เพื่อออกจากการแก้ไขโดยไม่ต้องเก็บข้อมูล, ปุ่ม 'S' เพื่อการจัดเก็บแฟ้มข้อมูลโดยไม่ออกจากการแก้ไข และปุ่ม 'X' เพื่อการจัดเก็บพร้อมทั้งออกจากการแก้ไข

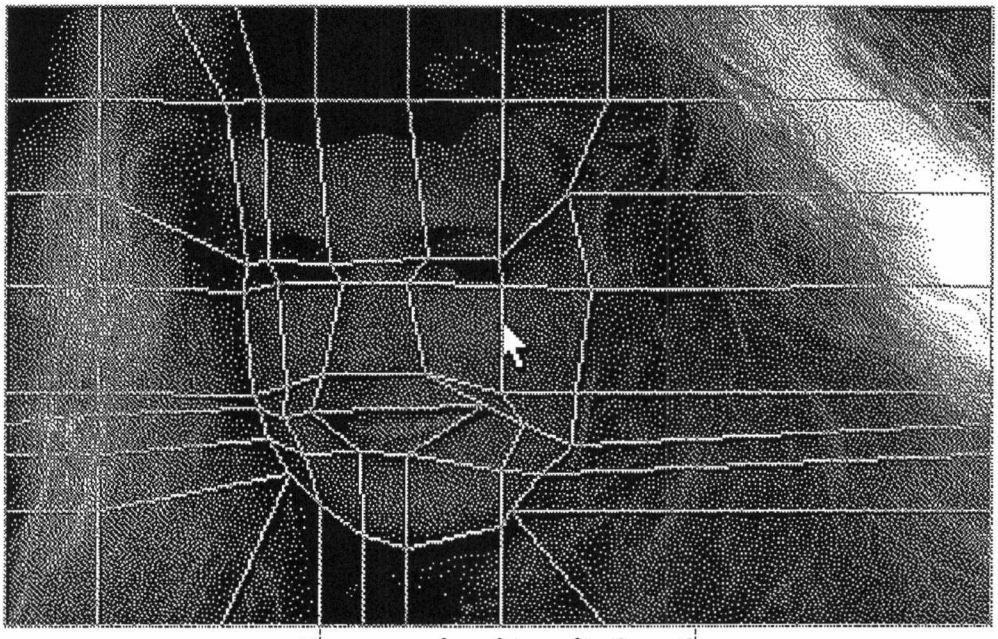
บทที่ 4

การประเมินผลระบบ

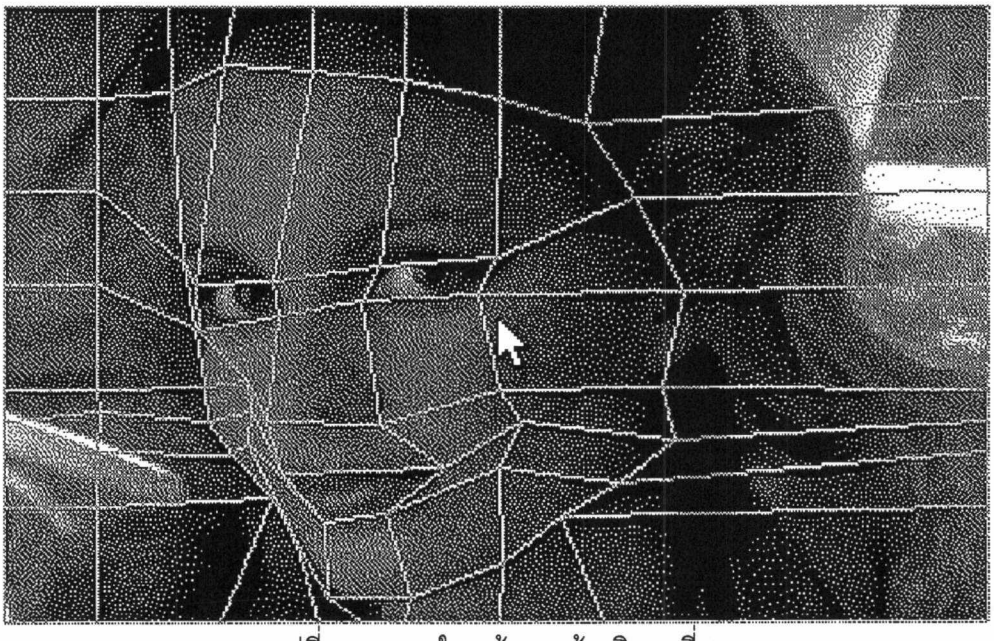
โปรแกรมต้นแบบที่ได้จากการพัฒนาระบบการทำภาพเคลื่อนไหวบนคอมพิวเตอร์ โดยใช้เทคนิคมอร์ฟฟิง เป็นโปรแกรมที่ทำงานบนดอส(DOS) โดยโปรแกรมต้นแบบนี้จะทำการประมวลผลมอร์ฟภาพ 2 ภาพ ซึ่งมีรูปแบบเป็นทีจี่เอ(TGA) จากนั้นจะให้ผู้ใช้ทำการกำหนดจุดควบคุม ซึ่งก็คือการกำหนดจุดยอดของรูปด้วยการลากเส้นตัดกันในแนวนอนและแนวตั้งเป็นตารางที่สัมพันธ์กันทั้งสองรูป โดยใช้เมาส์คลิกที่ขอบของหน้าต่างสำหรับการสร้างเส้นและขยับจุดควบคุมต่างๆ ขั้นตอนต่อมาก็คือ กำหนดจำนวนเฟรมที่ต้องการ

อย่างไรก็ตาม โปรแกรมต้นแบบที่พัฒนาขึ้นนี้ ยังมีข้อที่ควรปรับปรุงและแก้ไข ดังนี้

1. ในการกำหนดจุดและตารางจุดควบคุมนั้น เมื่อมีการลากเส้นในรูปแล้ว ไม่สามารถแก้ไขเปลี่ยนแปลงหรือยกเลิกเส้นที่ลากได้
2. ในการกำหนดจุดและกรอบควบคุมนั้นกระทำได้ยุ่งยาก เนื่องจากต้องกำหนดให้ภาพ ต้นทางและภาพปลายทางมีความสัมพันธ์กันอย่างเหมาะสม เช่นจุดควบคุมต่างๆ ที่อยู่ รอบใบหน้าของรูปผู้หญิงคนที่ 1 จะเป็นชุดเดียวกับจุดควบคุมที่อยู่รอบใบหน้าของรูปผู้หญิงคนที่ 2 ดังนั้นการเปลี่ยนจากโครงหน้าของผู้หญิงคนที่ 1 ไปเป็นโครงหน้าของผู้หญิงคนที่ 2 จึงเกิดจากการควบคุมของจุดเหล่านี้ ดังนั้นจึงต้องมีความแม่นยำในการกำหนดขอบเขตของตำแหน่ง อวัยวะต่างๆ ที่อยู่บนหน้า เช่น ตา จมูก ปาก เป็นต้น ให้ตรงกัน (ดูรูปที่ 4.1 และ รูปที่ 4.2)
3. การใช้หน่วยความจำของโปรแกรมต้นแบบมีการใช้หน่วยความจำมาก และถ้าหากไม่ใช้หน่วย ความจำยืดขยาย จะทำให้โปรแกรมทำงานได้ช้า เนื่องจากต้องเสียเวลาในการอ่านและเขียนข้อมูลระหว่างหน่วยความจำหลักกับฮาร์ดดิสก์
4. โปรแกรมต้นแบบนี้ ใช้เวลาในการประมวลผลประมาณ 25 วินาทีต่อภาพ โดยทำการรันบนเครื่อง ไมโครคอมพิวเตอร์ ระดับ 486 DX2-50



รูปที่ 4.1 ภาพใบหน้าของผู้หญิงคนที่ 1



รูปที่ 4.2 ภาพใบหน้าของผู้หญิงคนที่ 2

บทที่ 5

สรุปผลและเสนอแนะ

สรุปผล

ผลที่ได้จากการพัฒนาระบบการทำภาพเคลื่อนไหวบนคอมพิวเตอร์ โดยใช้เทคนิคมอร์ฟฟิง คือจะได้ชุดของภาพที่ประกอบด้วยภาพต้นทาง ภาพที่ได้จากการทำการมอร์ฟ และภาพปลายทาง ซึ่งจำนวนภาพทั้งหมดที่ได้จะขึ้นอยู่กับ การกำหนดจำนวนเฟรมของผู้ใช้ โดยภาพทั้งหมดที่นำมาแสดงนั้น จะเห็นถึงความเปลี่ยนแปลงทีละเล็กละน้อย จากภาพต้นทางกลายไปเป็นภาพปลายทาง โดยภาพในเฟรมแรกๆ ที่อยู่ใกล้กับภาพต้นทางจะมีลักษณะคล้ายคลึงกับภาพต้นทาง ส่วนภาพที่อยู่เฟรมทางตอนท้ายใกล้กับภาพปลายทางก็จะมีลักษณะคล้ายคลึงกับภาพปลายทาง ส่วนภาพเคลื่อนไหวที่ได้จะมีลักษณะกลมกลืนกันมากน้อยเพียงใดขึ้นอยู่กับความสัมพันธ์ในการกำหนดจุดและตารางเส้นควบคุมของภาพต้นทางและปลายทาง และการกำหนดจำนวนเฟรมทั้งหมดที่ใช้ในการทำมอร์ฟฟิง

ข้อจำกัดของระบบ คือ โปรแกรมต้นแบบนี้ทำการรันบนดอสในโหมดกราฟฟิกที่ความละเอียด 800x600 พิกเซลที่ 256 สี โดยประมวลผลกับภาพขนาด 320 x 200 แบบ 24 บิต หรือ 16 ล้านสี และติดต่อกับผู้ใช้โดยใช้เมาส์(MOUSE) และสามารถกำหนดจำนวนเฟรมที่ต้องการให้แสดงได้ขึ้นอยู่กับขนาดของเนื้อที่ว่างในอุปกรณ์บันทึกข้อมูล ซึ่งภาพแต่ละภาพจะมีขนาด 200,000 ตัวอักษร

ข้อเสนอแนะ

1. ในการกำหนดจุดและตารางเส้นควบคุมนั้น เมื่อมีการลากเส้นในรูปแล้วควรยกเลิกเส้นที่ลากได้
2. ควรเพิ่มเติมความสามารถในการใช้ไฟล์รูปแบบอื่น นอกเหนือจากรูปแบบทีจีเอ เช่น รูปแบบกิฟ (GIF) , บีเอ็มพี(BMP)
3. ควรเพิ่มเติมความสามารถในการประมวลผลภาพได้ที่ความละเอียดอื่นๆ นอกเหนือจากที่ความะเอียด 800 x 600 พิกเซล

4. เนื่องจากโปรแกรมต้นแบบได้ออกแบบโดยเขียนโปรแกรมในเชิงวัตถุ(OOP: Object Oriented Programming) ดังนั้นจึงสามารถนำคลาส(CLASS) ซึ่งออกแบบไว้ไปพัฒนาต่อ เพื่อประยุกต์กับระบบที่จะถูกพัฒนาต่อไปได้

ภาคผนวก ก.

Warping Algorithm

```
void Warp(PICTURE *sp,PICTURE *dp,MESH *sm,MESH *dm)
{
        int i,j,s_max_y,d_max_y,s_min_y,d_min_y;
        byte *src[3],*dst[3];
        POINT s[3],d[3];
        for (i=0;i<dm->amount_y;i++)
        {
                s_min_y=sm->p[0][i].y;s_max_y=sm->p[0][i+1].y;
                d_min_y=dm->p[0][i].y;d_max_y=dm->p[0][i+1].y;
                for (j=0;j<dm->amount_x;j++)
                {
                        if (sm->p[j][i+1].y>s_max_y)
                                s_max_y=sm->p[j][i+1].y;
                        if (sm->p[j][i].y<s_min_y)
                                s_min_y=sm->p[j][i].y;
                        if (dm->p[j][i+1].y>d_max_y)
                                d_max_y=dm->p[j][i+1].y;
                        if (dm->p[j][i].y<d_min_y)
```

```
                d_min_y=dm->p[j][i].y;
        }
        for (j=0;j<=2;j++)
        {
                src[j]=(byte*)malloc(320*(s_max_y-s_min_y+1));
                if (src[j]==NULL)
                {
                        puts("ERROR!!");
                        exit(-1);
                }
                dst[j]=(byte*)malloc(320*(d_max_y-d_min_y+1));
                if (dst[j]==NULL)
                {
                        puts("ERROR!!");
                        exit(-1);
                }
        }
        long int s_length=320l*(s_max_y-s_min_y+1);
        long int d_length=320l*(d_max_y-d_min_y+1);
        sp->r.copy_r(src[0],320u*s_min_y,s_length);
        sp->g.copy_r(src[1],320u*s_min_y,s_length);
        sp->b.copy_r(src[2],320u*s_min_y,s_length);
        dp->r.copy_r(dst[0],320u*d_min_y,d_length);
```

```
dp->g.copy_r(dst[1],320u*d_min_y,d_length);
dp->b.copy_r(dst[2],320u*d_min_y,d_length);
for (j=0;j<dm->amount_x;j++)
{
        d[0].x=dm->p[j][i].x;d[0].y=dm->p[j][i].y;
        d[1].x=dm->p[j][i+1].x;d[1].y=dm->p[j][i+1].y;
        d[2].x=dm->p[j+1][i].x;d[2].y=dm->p[j+1][i].y;
        s[0].x=sm->p[j][i].x;s[0].y=sm->p[j][i].y;
        s[1].x=sm->p[j][i+1].x;s[1].y=sm->p[j][i+1].y;
        s[2].x=sm->p[j+1][i].x;s[2].y=sm->p[j+1][i].y;
        TriangleMap(s,d,src,dst,s_min_y,d_min_y);

        d[0].x=dm->p[j+1][i].x;d[0].y=dm->p[j+1][i].y;
        d[1].x=dm->p[j][i+1].x;d[1].y=dm->p[j][i+1].y;
        d[2].x=dm->p[j+1][i+1].x;d[2].y=dm->p[j+1][i+1].y;
        s[0].x=sm->p[j+1][i].x;s[0].y=sm->p[j+1][i].y;
        s[1].x=sm->p[j][i+1].x;s[1].y=sm->p[j][i+1].y;
        s[2].x=sm->p[j+1][i+1].x;s[2].y=sm->p[j+1][i+1].y;
        TriangleMap(s,d,src,dst,s_min_y,d_min_y);
}
dp->r.copy_w(dst[0],320u*d_min_y,d_length);
dp->g.copy_w(dst[1],320u*d_min_y,d_length);
dp->b.copy_w(dst[2],320u*d_min_y,d_length);
```

```
            for (j=0;j<=2;j++)
            {
                free(src[j]);
                free(dst[j]);
            }
        }
}       /* End Warp */
```

ตัวแปรของฟังก์ชัน

```
int i,j,s_max_y,d_max_y,s_min_y,d_min_y;
byte *src[3],*dst[3];
POINT s[3],d[3];
```

ฟังก์ชันที่ถูกเรียกใช้

```
void TriangleMap(POINT *,POINT *,unsigned char *spic[3],byte *dpic[3],int,int);
```

Triangle Map Algorithm

```
void TriangleMap(POINT *s,POINT *d,byte *spic[3],byte *dpic[3],int s_min_y,int d_min_y)
{
        int left_edge[201];
        int right_edge[201];
        int min_y,max_y,i,j;
        float a1,a2,a3,a;

        scanRL(d,left_edge,right_edge);
        min_y=d[0].y;max_y=d[0].y;
        for (i=1;i<=2;i++)
        {
                if (d[i].y<min_y)
                        min_y=d[i].y;
                if (d[i].y>max_y)
                        max_y=d[i].y;
        }
        a=area(&d[0],&d[1],&d[2]);
        for (i=min_y;i<=max_y;i++)
        {
                for(j=left_edge[i];j<=right_edge[i];j++)
```

```
                {
                        POINT temp,newpoint;
                        temp.x=j;temp.y=i;
                        a1=area(&temp,&d[1],&d[2])/a;
                        a2=area(&d[0],&temp,&d[2])/a;
                        a3=area(&d[0],&d[1],&temp)/a;

                        newpoint.x=round(s[0].x*a1+s[1].x*a2+s[2].x*a3);
                        newpoint.y=round(s[0].y*a1+s[1].y*a2+s[2].y*a3);

                        dpic[0][(i-d_min_y)*320+j]=spic[0][((newpoint.y-s_min_y)*320)+
                        newpoint.x];
                        dpic[1][(i-d_min_y)*320+j]=spic[1][((newpoint.y-s_min_y)*320)+
                        newpoint.x];
                        dpic[2][(i-d_min_y)*320+j]=spic[2][((newpoint.y-s_min_y)*320)+
                        newpoint.x];
                }
        }
}
```

ตัวแปรของฟังก์ชัน

```
        int left_edge[201];
        int right_edge[201];
        int min_y,max_y,i,j;
        float a1,a2,a3,a;
```

ฟังก์ชันที่ถูกเรียกใช้

```
float area(POINT *a,POINT *b,POINT *c);

void scanRL(POINT p[3],int Left[200],int Right[200]);
```

Dissolve Algorithm

```
void dissolve(PICTURE *a,PICTURE *b,PICTURE *c,int now,int all)
{
        unsigned char *p1,*p2,*dest;
        long i;
        float factor=(float)now/(float)all;

        p1=(unsigned char *)malloc(6400);
        p2=(unsigned char *)malloc(6400);
        dest=(unsigned char *)malloc(6400);

        for (i=0;i<64000l;i+=6400)
        {
                a->r.copy_r(p1,i,6400);
                b->r.copy_r(p2,i,6400);
                calc(p1,p2,dest,factor);
                c->r.copy_w(dest,i,6400);
        }
        for (i=0;i<64000l;i+=6400)
        {
                a->g.copy_r(p1,i,6400);
                b->g.copy_r(p2,i,6400);
                calc(p1,p2,dest,factor);
                c->g.copy_w(dest,i,6400);
        }
        for (i=0;i<64000l;i+=6400)
        {
                a->b.copy_r(p1,i,6400);
```

```
            b->b.copy_r(p2,i,6400);
            calc(p1,p2,dest,factor);
            c->b.copy_w(dest,i,6400);
        }
}
```

ตัวแปรของฟังก์ชัน

```
unsigned char *p1,*p2,*dest;
long i;
float factor=(float)now/(float)all;
```

ฟังก์ชันที่ถูกเรียกใช้

```
void calc(unsigned char p1[6400],unsigned char p2[6400],unsigned char dest[6400],float
factor);
```

Tween Algorithm

```
void tween_calc(MESH *a,MESH *b,MESH *c,float factor)
{
        int i,j;

        c->amount_x=a->amount_x;c->amount_y=a->amount_y;

        for (j=0;j<=a->amount_y;j++)
        for (i=0;i<=a->amount_x;i++)
        {
                c->p[i][j].x=(a->p[i][j].x*(1.0-factor))+(b->p[i][j].x*factor);
                c->p[i][j].y=(a->p[i][j].y*(1.0-factor))+(b->p[i][j].y*factor);
        }
}

void tween(char *filename1,char *filename2,int amount)
{
        MESH m1,m2,out;
        char f[13];
        int i;

        // Create first and last frame before //

        m1.load(filename1);m2.load(filename2);
        m1.save("1.tmp");
        itoa(amount,f,10);strcat(f,".tmp");m2.save(f);

        for (i=2;i<=amount-1;i++)
```

```
        {
                itoa(i,f,10);strcat(f,".tmp");
                tween_calc(&m1,&m2,&out,(float)(i-1)/(amount-1));
                out.save(f);
        }
}
```

ตัวแปรของฟังก์ชัน

```
MESH m1,m2,out;
char f[13];
int i;
```

ฟังก์ชันที่ถูกเรียกใช้

```
void tween_calc(MESH *a,MESH *b,MESH *c,float factor);
```

Morph Algorithm

```
void morph(void)
{
        PICTURE a,b,c,d,e;
        MESH m1,m2,m3;
        char f[13];
        char filename[13];
        int i;

        a.load(LEFT_PIC);b.load(RIGHT_PIC);
        m1.load(LEFT_MESH);m2.load(RIGHT_MESH);
        tween(LEFT_MESH,RIGHT_MESH,AMOUNT_FRAME);

        strcpy(f,"1");strcat(f,".tga");
        strcpy(filename,SUFFIX);strcat(filename,f);
        a.save(filename);
        itoa(AMOUNT_FRAME,f,10);strcat(f,".tga");
        strcpy(filename,SUFFIX);strcat(filename,f);
        b.save(filename);
        for (i=2;i<=AMOUNT_FRAME-1;i++)
        {
                itoa(i,f,10);strcat(f,".tmp");
                m3.load(f);
                Warp(&a,&c,&m1,&m3);Warp(&b,&d,&m2,&m3);
                dissolve(&c,&d,&e,i-1,AMOUNT_FRAME-1);
                itoa(i,f,10);strcat(f,".tga");
                strcpy(filename,SUFFIX);strcat(filename,f);
                e.save(filename);
```

```
        }
        clear_mesh(AMOUNT_FRAME);
}
```

ตัวแปรของฟังก์ชัน

```
        PICTURE a,b,c,d,e;
        MESH m1,m2,m3;
        char f[13];
        char filename[13];
        int i;
```

ฟังก์ชันที่ถูกเรียกใช้

```
        void dissolve(PICTURE *a,PICTURE *b,PICTURE *c,int now,int all);
        void Warp(PICTURE *s,PICTURE *d,MESH *sm,MESH *dm);
        void tween(char *filename1,char *filename2,int amount);
        void clear_mesh(int);
```

คลาสต่าง ๆ ที่เกี่ยวข้อง

```
#define XMS_USED 0
#define DSK_USED 1
#define EMS_USED 2
extern XMSstruct XMSctrlblock;

memBlock::memBlock(void)
{
}

memBlock::memBlock(long size)
{
        init(size);
}

memBlock::~memBlock(void)
{
        free();
};

void memBlock::init(long size)
{
        long int size_k=(size/1024)+((size%1024)? 1:0);
        if ((XMSinstalled())&&(XMSgetFree()>=size_k))
        {
                handle=XMSalloc((int)size_k);
                mem_type=XMS_USED;
        }
```

```
if ((!XMSinstalled())||(XMSgetFree()<size_k)||(handle==0))
{
        unsigned int i,j;
        FILE *fp;

        randomize();
        do{
                for (i=0;i<=7;i++)
                {
                        j=random(10);SwapName[i]=j+48;
                }
                SwapName[8]=0;
                strcat(SwapName,".TMP");
        }while(fopen(SwapName,"rb")!=NULL);

        fp=fopen(SwapName,"wb");
        for (i=0;i<=size-1;i++)
                if (fputc(0,fp)==EOF)
                {
                        puts("Insufficient disk space");
                        fclose(fp);
                        unlink(SwapName);
                        exit(-1);
                }
        fclose(fp);

        mem_type=DSK_USED;
}
```

```
        amount=size;
}

void memBlock::free(void)
{
        if (mem_type==XMS_USED)
        {
                XMSfree(handle);
        }
        else
        if (mem_type==DSK_USED)
        {
                remove(SwapName);
        }
}

void memBlock::read(void far *s)
{
        if (mem_type==XMS_USED)
        {
                XMSctrlblock.length=(unsigned long) amount;
                XMSctrlblock.s_handle=handle;
                XMSctrlblock.s_offset=(unsigned long) 0L;
                XMSctrlblock.t_handle=0;
                XMSctrlblock.t_offset=(unsigned long) s;

                XMScopy(&XMSctrlblock);
        }
        else
```

```
        {
                FILE *fp;
                unsigned i;
                char *temp;

                fp=fopen(SwapName,"rb");

                for (i=0;i<=amount;i+=1024)
                {
                        if (i+1024<amount)
                                fread(((char*)s)+i,1024,1,fp);
                        else
                                fread((temp+i),(int)(amount%1024),1,fp);
                }
                fclose(fp);
        }
}

void memBlock::write(void far *s)
{
        if (mem_type==XMS_USED)
        {
                XMSctrlblock.length=(unsigned long) amount;
                XMSctrlblock.s_handle=0;
                XMSctrlblock.s_offset=(unsigned long) s;
                XMSctrlblock.t_handle=handle;
                XMSctrlblock.t_offset=(unsigned long) 0L;

                XMScopy(&XMSctrlblock);
```

```
        }
        else
        {
                unsigned i;
                char *temp;
                FILE *fp;
                fp=fopen(SwapName,"rb+");

                temp=(char *)s;

                for (i=0;i<=amount;i+=1024)
                {
                        if (i+1024<amount)
                                fwrite((temp+i),1024,1,fp);
                        else
                                fwrite((temp+i),(int)(amount%1024),1,fp);
                }

                fclose(fp);
        }
}

void memBlock::copy_r(void far *s,long start,long dist)
{
        if (mem_type==XMS_USED)
        {
                XMSctrlblock.length=(unsigned long) dist;
                XMSctrlblock.s_handle=handle;
                XMSctrlblock.s_offset=(unsigned long) start;
```

```
                XMSctrlblock.t_handle=0;
                XMSctrlblock.t_offset=(unsigned long) s;

                XMScopy(&XMSctrlblock);
        }
        else
        {
                FILE *fp;
                unsigned i;
                char *temp;

                temp=(char *)s;

                fp=fopen(SwapName,"rb");

                fseek(fp,start,SEEK_SET);
                for (i=0;i<=dist;i+=1024)
                {
                        if (i+1024<dist)
                                fread((temp+i),1024,1,fp);
                        else
                                fread((temp+i),(int)(dist%1024),1,fp);
                }
                fclose(fp);
        }
}

void memBlock::copy_w(void far *s,long start,long dist)
{
```

```
    if (mem_type==XMS_USED)
    {
        XMSctrlblock.length=(unsigned long) dist;
        XMSctrlblock.s_handle=0;
        XMSctrlblock.s_offset=(unsigned long) s;
        XMSctrlblock.t_handle=handle;
        XMSctrlblock.t_offset=(unsigned long) start;

        XMScopy(&XMSctrlblock);
    }
    else
    {
        unsigned i;
        char *temp;
        FILE *fp;
        temp=(char *)s;

        fp=fopen(SwapName,"rb+");
        fseek(fp,start,SEEK_SET);
        for (i=0;i<=dist;i+=1024)
        {
            if (i+1024<dist)
                fwrite((temp+i),1024,1,fp);
            else
                fwrite((temp+i),(int)(dist%1024),1,fp);
        }
        fclose(fp);
    }
}
```

```
PICTURE::PICTURE(void)
{
        r.init(64000l);g.init(64000l);b.init(64000l);
}

PICTURE::~PICTURE(void)
{
        r.free();g.free();b.free();
}

int PICTURE::load(char *s)
{
        FILE *fp;
        unsigned char header[18],data[3200][3],_r[3200],_g[3200],_b[3200];
        int i,j;

        fp=fopen(s,"rb");
        if (fp==NULL)
        {
                puts("Error! file NOT FOUND!!");
                return -1;
        }

/*      Read header away */
        fread(header,18,1,fp);

        for (i=0;i<=19;i++)
        {
```

```
                fread(data,3200,3,fp);
                for(j=0;j<3200;j++)
                {
                        _r[j]=data[j][0];
                        _g[j]=data[j][1];
                        _b[j]=data[j][2];
                }
                r.copy_w(_r,i*3200u,3200);
                g.copy_w(_g,i*3200u,3200);
                b.copy_w(_b,i*3200u,3200);
        }
        fclose(fp);
        return 0;
}

int PICTURE::save(char *s)
{
        FILE *fp;
        unsigned char data[3200][3],_r[3200],_g[3200],_b[3200];
        unsigned char header[18]={0,0,2,0,0,0,0,0,0,0,0,0,0x40,1,0xc8,0,0x18,0x20};
        int i,j;


        fp=fopen(s,"wb");
        if (fp==NULL)
        {
                puts("Error! file NOT FOUND!!");
                return -1;
        }
```

```
/*      Read header away */
        fwrite(header,18,1,fp);

        for (i=0;i<=19;i++)
        {
                r.copy_r(_r,i*3200u,3200);
                g.copy_r(_g,i*3200u,3200);
                b.copy_r(_b,i*3200u,3200);

                for(j=0;j<3200;j++)
                {
                        data[j][0]=_r[j];
                        data[j][1]=_g[j];
                        data[j][2]=_b[j];
                }
                fwrite(data,3200,3,fp);
        }
        fclose(fp);
      return 0;
}

MESH::MESH(void)
{
}

MESH::MESH(char *s)
{
        load(s);
```

```
}

int MESH::load(char *s)
{
        FILE *fp;
        int i,j;

        fp=fopen(s,"r");
        if (fp==NULL)
        {
                puts("Error! file NOT FOUND!!");
                return -1;
        }
        fscanf(fp,"%d %d",&amount_x,&amount_y);

        for (i=0;i<=amount_y;i++)
        for (j=0;j<=amount_x;j++)
                if (fscanf(fp,"%d %d",&(p[j][i].x),&(p[j][i].y))==EOF)
                {
                        puts("Error! file DAMAGE");
                        return -1;
                }
        fclose(fp);
        return 0;
}

int MESH::save(char *s)
{
        FILE *fp;
```

```
        int i,j;

        fp=fopen(s,"w");
        if (fp==NULL)
        {
                puts("Error! file NOT FOUND!!");
                return -1;
        }

        fprintf(fp,"%d\n%d\n",amount_x,amount_y);

        for (i=0;i<=amount_y;i++)
        for (j=0;j<=amount_x;j++)
                if (fprintf(fp,"%d %d\n",p[j][i].x,p[j][i].y)==EOF)
                {
                        puts("Error! file DAMAGE");
                        return -1;
                }

        fclose(fp);
        return 0;
}

void MESH::show(int x,int y)
{
        int i,j;

        for (j=0;j<=amount_y;j++)
        for (i=0;i<=amount_x-1;i++)
```

```
                line(x+p[i][j].x,y+p[i][j].y,x+p[i+1][j].x,y+p[i+1][j].y);

        for (i=0;i<=amount_x;i++)
        for (j=0;j<=amount_y-1;j++)
                line(x+p[i][j].x,y+p[i][j].y,x+p[i][j+1].x,y+p[i][j+1].y);
}

void MESH::sort(void)
{
        int i,j,k,temp;


        for (k=0;k<=amount_y;k++)
        for (j=0;j<=amount_x-1;j++)
        for (i=0;i<=amount_x-1;i++)
                if (p[i][k].x>p[i+1][k].x)
                {
                        temp=p[i][k].x;
                        p[i][k].x=p[i+1][k].x;
                        p[i+1][k].x=temp;
                }

        for (i=0;i<=amount_x;i++)
        for (j=0;j<=amount_y-1;j++)
        for (k=0;k<=amount_y-1;k++)
                if (p[i][k].y>p[i][k+1].y)
                {
                        temp=p[i][k].y;
                        p[i][k].y=p[i][k+1].y;
```

```
                    p[i][k+1].y=temp;
                }
}
EVENT::EVENT(void)
{
        last=0;
}

void EVENT::add(POINT *p1,POINT *p2,char hot, void (*func)(void))
{
        memo[last].p1.x=p1->x;memo[last].p1.y=p1->y;
        memo[last].p2.x=p2->x;memo[last].p2.y=p2->y;
        memo[last].hotkey=hot;
        memo[last].func=func;
        last++;
}

void EVENT_HELP::add(POINT *p1,POINT *p2,char hot, void (*func)(void),char str[])
{
        strcpy(message[last],str);
        EVENT::add(p1,p2,hot,func);
}

char * EVENT_HELP::msg(POINT *p)
{
        int i;
        char c[]="";
        for(i=last;i>=0;i--)
        {
```

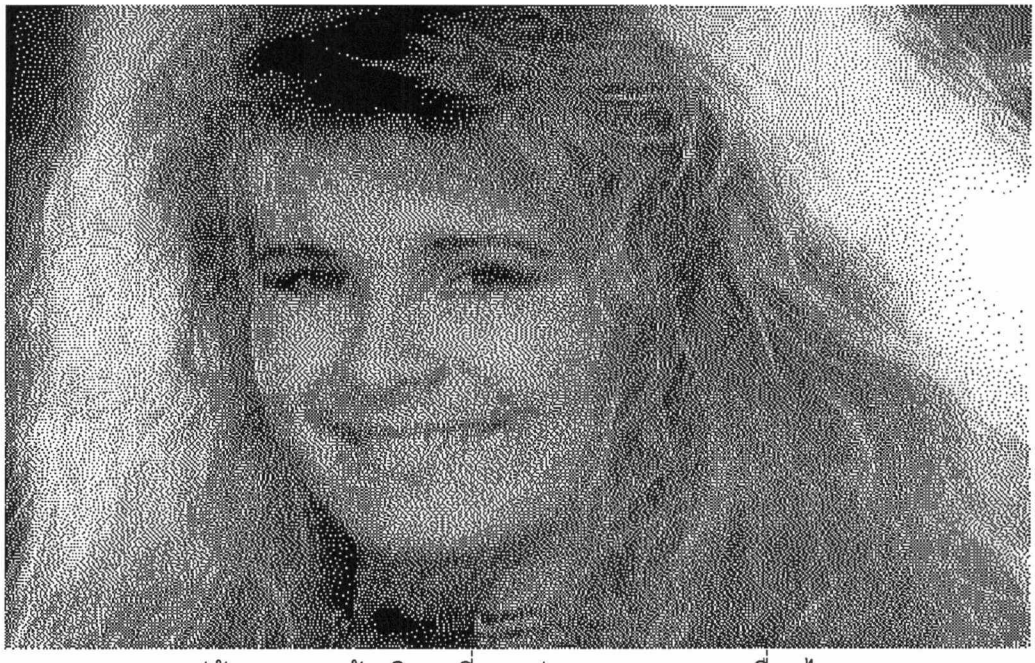
```
                if ((p->x>=memo[i].p1.x)&&(p->x<=memo[i].p2.x))
                if ((p->y>=memo[i].p1.y)&&(p->y<=memo[i].p2.y))
                {
                        return message[i];
                }
        }
        return c;
}


void EVENT::check_mouse(POINT *p)
{
        int i,j=0;
        for(i=last;i>=0&&j!=1;i--)
        {
                if ((p->x>=memo[i].p1.x)&&(p->x<=memo[i].p2.x))
                if ((p->y>=memo[i].p1.y)&&(p->y<=memo[i].p2.y))
                {
                        (*(memo[i].func))();
                        j=1;
                }
        }
}

void EVENT::check_key(char c)
{
        int i,j=0;
        for(i=last-1;i>=0&&j!=1;i--)
        {
```

```
                if (memo[i].hotkey==c)
                {
                        (*(memo[i].func))();
                        j=1;
                }
        }
}
```

ภาคผนวก ข.

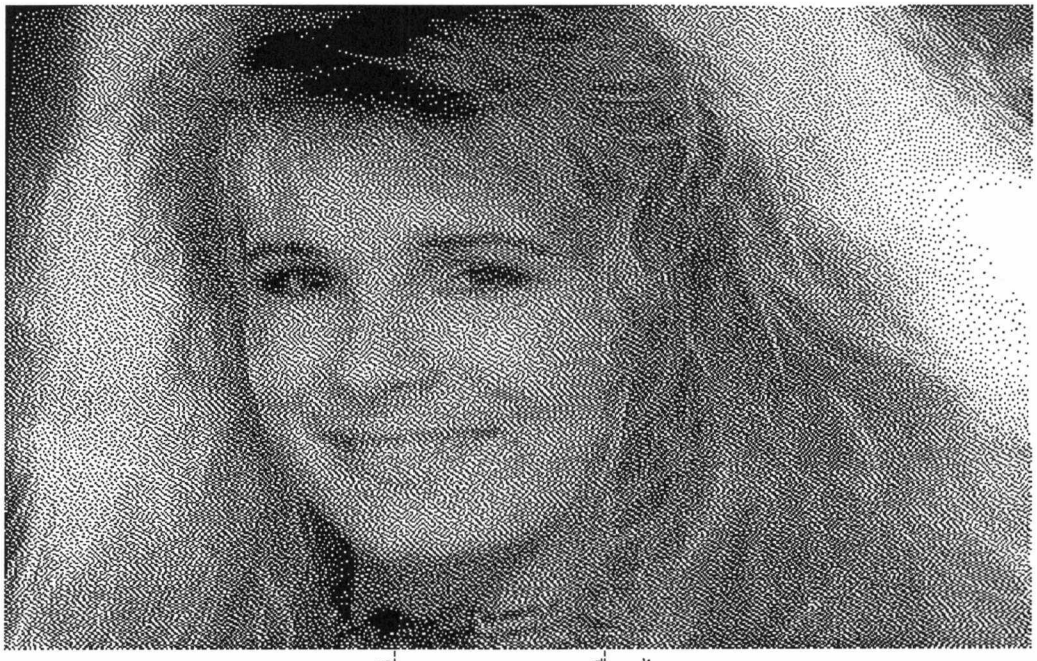
ตัวอย่างของชุดภาพที่สร้างได้

ภาคผนวกนี้เป็นตัวอย่างของชุดภาพเคลื่อนไหวที่สร้างได้จากโปรแกรมต้นแบบโดยทำการสร้างชุดภาพเคลื่อนไหวจำนวนทั้งหมด 10 รูป

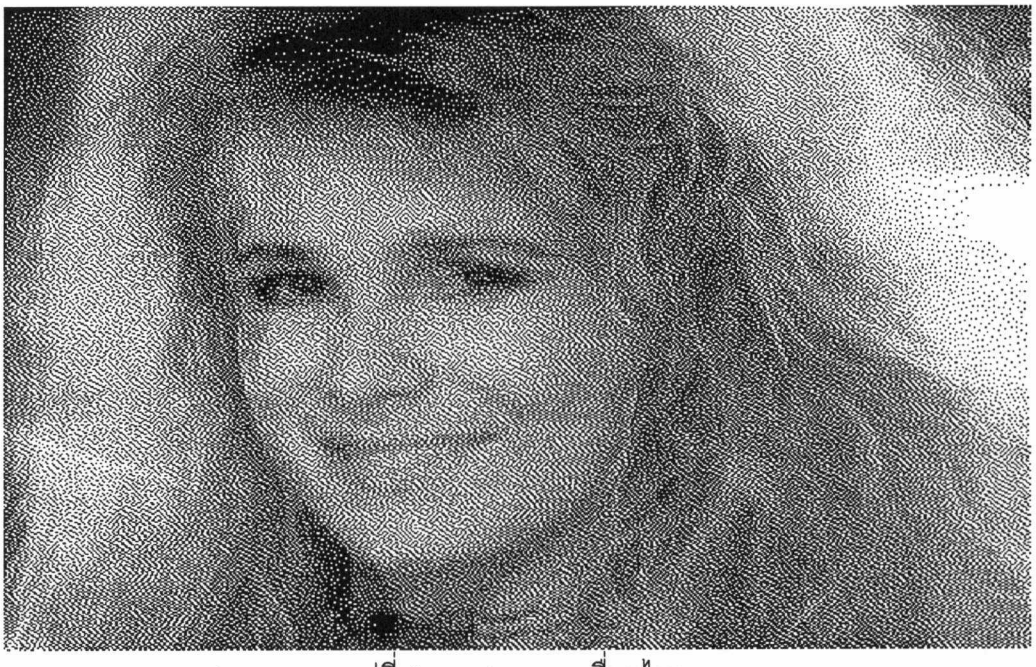


รูปต้นแบบของผู้หญิงคนที่ 1 (รูปแรกของชุดภาพเคลื่อนไหว)

รูปต้นแบบของผู้หญิงคนที่ 2 (รูปที่ 10ของชุดภาพเคลื่อนไหว)

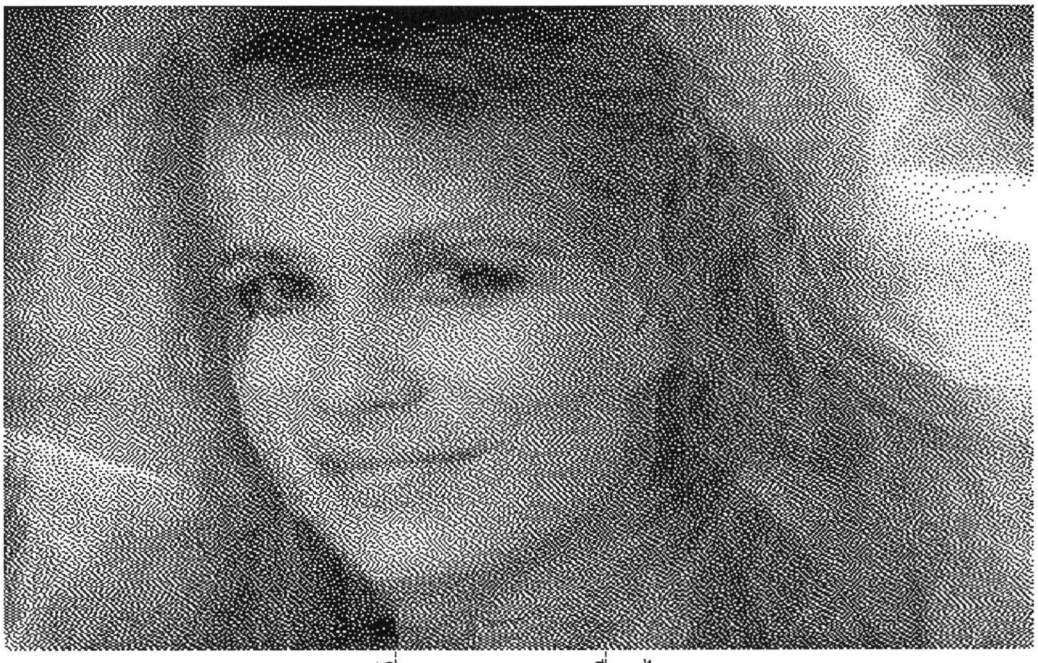


รูปที่ 3 ของชุดภาพเคลื่อนไหว



รูปที่ 4 ของชุดภาพเคลื่อนไหว

รูปที่ 5 ของชุดภาพเคลื่อนไหว



รูปที่ 6 ของชุดภาพเคลื่อนไหว

รูปที่ 7 ของชุดภาพเคลื่อนไหว

รูปที่ 8 ของชุดภาพเคลื่อนไหว

รูปที่ 9 ของชุดภาพเคลื่อนไหว

รูปที่ 10 ของชุดภาพเคลื่อนไหว

ภาคผนวก ค

การติดตั้งระบบ

ฮาร์ดแวร์

- เครื่องคอมพิวเตอร์ที่มี CPU ตั้งแต่รุ่น 386 ขึ้นไป
- มีเนื้อที่ว่างในฮาร์ดดิสค์อย่างน้อย 2 เมกกะไบต์
- มีหน่วยความจำยืดขยายอย่างน้อย 1 เมกกะไบต์
- ไมโครซอฟต์เมาส์ หรือที่เข้ากันได้
- จอภาพแบบ ซุเปอร์วีจีเอ ที่สามารถทำงานที่ความละเอียด 800*600 ที่ 256 สี ได้

ซอฟท์แวร์

- โปรแกรมระบบปฏิบัติการดอส รุ่น 3.30 ขึ้นไป
- โปรแกรมจัดการหน่วยความจำยืดขยาย

ไฟล์ที่ใช้ทำงาน

- MORPH.EXE : แฟ้มโปรแกรมหลัก
- SVGA256.BGI : แฟ้มไดรฟ์เวอร์จอภาพซุเปอร์วีจีเอ
- DESKTOP.BMP : แฟ้มภาพบิตแมพของวอลล์เปเปอร์
- MORPH.FON : แฟ้มตัวอักษรภาษาไทย
- ????????.TGA : แฟ้มภาพต้นฉบับสำหรับการมอร์ฟ
- ????????.MSH : แฟ้มตารางควบคุมสำหรับการมอร์ฟ

บรรณานุกรม

- เกษมสันต์ พานิชการ, C++ และหลักการของ OOP ระดับเริ่มต้น , ซีเอ็ดยูเคชัน , 2537
- Scott Anderson , **MORPHING MAGIC** , SAMS ,1993
- Valerie Hall , **INTRODUCTION TO MORPHING** , 1993