บทที่ 4

การทดสอบโปรแกรม

ในบทที่ผ่านมาได้กล่าวถึงโครงสร้างภายในและความสัมพันธ์ของโครงสร้างข้อมูลของระบบจัดการภาพเคลื่อนไหวที่คลังโปรแกรม TAM จัดเตรียมไว้ให้ ในบทนี้จะทำการทดสอบประสิทธิภาพของระบบโดยพัฒนาโปรแกรมตรวจสอบความเร็วในการสร้างภาพเคลื่อนไหวของสไปรต์

<u>โปรแกรมทดสอบประสิทธิภาพ</u>

ในคลังโปรแกรมมีการกำหนดโครงสร้างข้อมูลต่างๆไว้หลายชนิด แต่มีเพียงบางชนิดเท่านั้นที่ผู้เขียนโปรแกรมจำเป็นต้องรู้จักและนำไปใช้ เนื่องจากในคลังโปรแกรมกำหนดให้จัดการข้อมูลของระบบได้โดยการเรียกใช้ฟังก์ชันต่างๆที่จัดเตรียมไว้ให้และใช้ค่าแฮนเดิลสำหรับโครงสร้างข้อมูลที่ต้องการเท่านั้น ตัวอย่างโปรแกรมต่อไปนี้เป็นการพัฒนาโปรแกรมเพื่อตรวจสอบความเร็วในการสร้างภาพเคลื่อนไหวของสไปรต์ที่มีขนาดต่างๆกัน โดยกำหนดความต้องการของระบบดังนี้

1. ต้องการสไปรต์ 2 ชนิดคือ
   - 1.1 สไปรต์ชนิด 1 ภาพ มีได้หลายขนาด (กว้าง x สูง) ดังนี้
     10x10, 20x20, 30x30, ... ,100x100, 150x150, 200x200 โดยมีหน่วยเป็นจุดภาพ (pixels)
     ในโปรแกรมจะสร้างเป็นภาพวงกลมตามขนาดที่ระบุ
   - 1.2 สไปรต์ชนิดหลายภาพ มีขนาดเดียว คือ 50x50 จุดภาพ
     ในโปรแกรมจะสร้างเป็นภาพดาว 5 แฉก โดยแต่ละภาพจะหมุน (rotate) ทีละ 10 องศา ดังนั้นจึงใช้ภาพดาว 35 ภาพ (ภาพที่ 0,10,20, ..., 350 องศา)
2. สามารถเลือกจำนวนสไปรต์ในระบบได้ตั้งแต่ 1 ถึง 300 ตัว และจำนวนสไปรต์มากที่สุด (maximum) โดยในระบบนี้ต้องการ 500 ตัว (เป็นกรณีทดสอบ)
3. ให้มีการรายงานเวลาที่ตัวจักรทำภาพเคลื่อนไหวทำงานในแต่ละกรณีเช่น
   - 3.1 เวลาที่ใช้ในการส่งข้อความไปให้กับสไปรต์ (Broadcast message)
   - 3.2 เวลาที่ใช้ในการทำภาพเคลื่อนไหวต่อ 1 ครั้ง
   - 3.3 เวลาที่ใช้ในการวาดใหม่ทั้งจอภาพ (Refresh ภาพพื้นหลังและสไปรต์ทุกตัว)

3.4 เวลาที่ใช้ในการทำภาพเคลื่อนไหวในกรณีที่ดีที่สุด (เมื่อสไปรต์ทุกตัวมาอยู่ที่ตำแหน่งเดียวกัน)

จากความต้องการของระบบดังกล่าวนำมาวิเคราะห์และออกแบบโปรแกรมได้ดังนี้

1. การสร้างสไปรต์ทั้งสองชนิดตามที่ต้องการนั้นกระทำได้ดังนี้

1.1 สร้างคลังภาพสำหรับสไปรต์ชนิด 1 ภาพ และเป็นรูปวงกลม

สร้างภาพวงกลมแต่ละขนาดตามที่ระบุโดยใช้ฟังก์ชันของวินโดวส์ (Windows API) จากนั้นนำแต่ละภาพมาเก็บไว้ในคลังภาพและสร้างเป็นลำดับภาพ (Sequence) โดยใช้ฟังก์ชันใน TAM ดังนี้

```
#define       NUMSIZE   12
HDC           hDC, hmDC;
HBITMAP       hBmp[ NUMSIZE ];
HBMTAB        BmpTab [ NUMSIZE ];
HSEQUENCE     hSeq [ NUMSIZE ];
int           I;

hDC = GetDC(NULL);
hmDC = CreateCompatibleDC(hDC);
for ( i=0; i<NUMSIZE; i++) {
     hBmp[i] = CreateCompatibleBitmap(hDC, w[i], h[i]);  // กำหนดค่า w[], h[] ไว้แล้ว
     SelectObject(hmDC, hBmp[i]);
     PatBlt(hmDC, 0,0,w[i],h[i],BLACKNESS);
     Ellipse(hmDC,0,0,w[i],h[i]);
     // สร้างคลังภาพ
     BmpTab[i] = amRegisterCompatibleBitmap( (HBITMAP far*) &hBmp[i] );
     // สร้างลำดับภาพ (Sequence)
     hSeq[i]   = amCreateSequence(BmpTab[i], 0,0);
}
```

1.2 สร้างคลังภาพสำหรับสไปรต์ชนิดหลายภาพขนาด กว้างxสูง = 50x50 จุดภาพ

สร้างภาพหลายเหลี่ยมรูปดาวขนาดตามที่ระบุโดยใช้ฟังก์ชันของวินโดวส์ (Windows API) จากนั้นนำแต่ละภาพมาเก็บไว้ในคลังภาพและสร้างเป็นลำดับภาพ (Sequence) โดยใช้ฟังก์ชันใน TAM ดังนี้

```
HBITMAP  hBmpStar [35];
POINT    pStar[10] = {0,20, 20,20, 25,0, 30,20, 49,20, 30,30, 49,49, 25,37, 0,49, 20,30};
POINT    np[10];
int      i, ang0;
HSEQUENCE  hSeqStar;
HBMTAB     hStarTab;

for (i=0;i<35;i++, ang0+=10) {
   hBmpStar[i] = CreateCompatibleBitmap(hDC, 50,50);
   SelectObject(hmDC, hStar[i]);
   // หมุนจุดของ Polygon ที่ใช้สร้างรูปดาวเป็นมุม ang0 (ครั้งละ 10 องศา)
   amRotatePolygon((LPPOINT) &pStar, (LPPOINT) &np, 10, 25,25, ang0);
   PatBlt(hmDC, 0,0,50,50,BLACKNESS);
   Polygon(hmDC, (LPPOINT) &np, 10);
}
// สร้างคลังภาพ
hStarTab = amRegisterCompatibleBitmap( (HBITMAP far*) &hBmpStar[0] );
// สร้างลำดับภาพ
hSeqStar = amCreateSequence(hStarTab, 0,0);
```

2. สร้างสไปรต์ตามจำนวนที่ต้องการ

เมื่อสร้างคลังภาพและลำดับภาพสำหรับสไปรต์แต่ละชนิดแล้วจึงนำไปใช้ในการสร้างสไปรต์ขึ้นมาโดยใช้ฟังก์ชันดังนี้

```
#define      MAXSPRITE   500
HENVIRON   hEnv;
HSPRITE    hSprite[MAXSPRITE];
int        i, x, y, nPriority;

// สร้างสภาพแวดล้อมของระบบ
hEnv  = amInitEnv (hWnd);
// สร้างสไปรต์
```

```
for (i=0;i<NumSprite;i++) {
    hSpr [i] =  amCreateSprite("ClassOfSprite",
                                NULL,
                                NULL,
                                hEnv,
                                nPriority,
                                SPR_DEFAULT,
                                x, y,
                                hWnd,
                                (WNDPROC)  SpriteProc);
    // เลือกลำดับภาพให้สไปรต์
    amSelectSequence (hSpr[i],  hSeq);
}
```

3. การจับเวลาการทำงานและรายงานผล

การจับเวลาการทำงานของตัวจักรทำภาพเคลื่อนไหวในโปรแกรมทดสอบนี้ ฟังก์ชันที่ใช้ในการจับเวลานั้นจะใช้ฟังก์ชันในกลุ่มของมัลติมีเดียคือ timeGetTime() ซึ่งจะให้ค่าของเวลาตั้งแต่เริ่มดำเนินงานระบบวินโดวส์จนถึงปัจจุบันมีหน่วยเป็นมิลลิวินาทีและให้ค่าละเอียดถึง 1 มิลลิวินาที แต่ถ้าใช้ฟังก์ชัน GetCurrentTime() จะให้ค่าเวลาต่างกันเพียง 55 มิลลิวินาที เท่านั้น

3.1 เวลาที่ใช้ในการส่งข้อความไปให้กับสไปรต์ (Broadcast message)

การควบคุมสไปรต์ทำได้โดยการส่งข้อความคำสั่งหรือเหตุการณ์ไปให้สไปรต์เหล่านั้น ซึ่งทำได้โดยฟังก์ชัน amBroadCastMessage() นอกจากนี้ยังสามารถควบคุมสไปรต์โดยตรงด้วยฟังก์ชันก็ได้แต่ถ้าไม่จำเป็นก็ไม่ควรใช้วิธีนี้เนื่องจากจะทำให้เกิดการสับสนในการเขียนโปรแกรม (ผู้เขียน) การจับเวลาในการส่งข้อความให้สไปรต์ทำได้ดังนี้

```
DWORD   t1, t2;
int         nBroadCastTime;

switch (msg) {
    case  WM_TIMER :
          t1 = timeGetTime();
          amBroadCastMessage (hEnv, msg, wParam, lParam);
```

```
            t2 = timeGetTime();
            nBroadCastTime = (int)  (t2-t1);
            break;
        . . .
    }
```

3.2 เวลาที่ใช้ในการทำภาพเคลื่อนไหวต่อ 1 ครั้ง

ฟังก์ชันในการทำภาพเคลื่อนไหวคือ amAnimate() ซึ่งในโปรแกรมทดสอบนี้จะให้ทำการเคลื่อนไหวภาพตามข้อความ WM_TIMER ที่ได้รับ ดังนั้นจากตัวอย่างที่ผ่านมานำมาเพิ่มเติมได้ดังนี้

```
    int     nAnimateTime;

    switch (msg) {
      case  WM_TIMER :
            t1 = timeGetTime();
            amBroadCastMessage (hEnv, msg, wParam, lParam);
            t2 = timeGetTime();
            nBroadCastTime = (int) (t2-t1);
            t1 = timeGetTime();
            amAnimate(NULL, hEnv, NORMAL);
            t2 = timeGetTime();
            nAnimateTime = (int)  (t2-t1);
            break;
        . . .
    }
```

3.3 เวลาที่ใช้ในการวาดใหม่ทั้งจอภาพ

การวาดใหม่ทั้งจอภาพสามารถทำได้โดยใช้ฟังก์ชัน amRefreshAll() ฟังก์ชันนี้จะทำการวาดภาพพื้นหลัง (background) และสไปรต์ทุกตัวลงบนหน่วยความจำที่เป็น offscreen แล้วจึงนำออกไปที่จอภาพของวินโดว์แม่ การจับเวลาการวาดใหม่ทำได้ดังนี้

```
int     nRefreshTime;
switch (msg) {
        case  WM_COMMAND :
              switch (wParam) {
                 case IDM_REFRESH :
                      t1 = timeGetTime();
                      amRefreshAll (NULL, hEnv);
                      t2 = timeGetTime();
                      nRefreshTime = (int) (t2-t1);
                      break;
              }
        }
```

3.4 การจับเวลาในกรณีที่ดีที่สุด

กรณีดีที่สุดได้แก่การที่สไปรต์ทุกตัวเคลื่อนที่มาอยู่ในตำแหน่งเดียวกัน (ทับกันหมด) ดังนั้นการบังคับให้สไปรต์มาอยู่ในตำแหน่งเดียวกันทำได้โดยใช้ฟังก์ชัน amSetSpriteXY() จากนั้นจึงจับเวลาของฟังก์ชัน amAnimate()

```
switch (msg) {
case WM_COMMAND :
     switch (wParam) {
        case IDM_BESTCASE :
             for (i=0;i<NumSprite;i++) {
                amSetSpriteXY(hSprite[i], 100,100);
             }
             t1 = timeGetTime();
             amAnimate(NULL, hEnv, NORMAL);
             t2 = timeGetTime();
             nBestCaseTime = (int) (t2-t1);
             break;
        . . .
     }
```

จากตัวอย่างนี้ ถ้าไม่ทราบว่ามีสไปรต์อยู่ในวินโดว์แม่กี่ตัว การใช้วงวน for อาจทำให้โปรแกรมหยุดทำงานได้ วิธีแก้ปัญหานี้ทำได้โดยใช้ฟังก์ชัน amBroadCastMessage() โดยส่งข้อความไปให้สไปรต์ทุกตัวทำการตอบสนองต่อข้อความที่ส่งไปให้ดังตัวอย่างนี้

ที่วินโดว์แม่ :

```
case IDM_BESTCASE :
    amBroadCastMessage(hEnv, WM_USER, hWnd, MAKELONG(100,100));
    t1 = timeGetTime();
    amAnimate(NULL, hEnv, NORMAL);
    t2 = timeGetTime();
    . . .
```

ที่วินโดว์ประจำตัวสไปรต์ :

```
LONG FAR PASCAL SpriteProc (HWND hWnd, unsigned msg, WORD wParam, LONG lParam)
{
    HSPRITE hSprite;
    switch (msg) {
      case WM_USER :
        hSprite = amGetSpriteHandle(hWnd);
        amSetSpriteXY ( hSprite, LOWORD(lParam), HIWORD(lParam));
        amUpdateSprite( hSprite );
        break;
      . . .
    }
}
```

<u>การพัฒนาโปแกรม</u>

จากการวิเคราะห์และออกแบบระบบดังกล่าวนำมาเขียนโปรแกรมได้โดย

1. เขียนโปรแกรมหลัก (.C)
2. เขียน Resource Script (.RC)
3. เขียน Definition File (.DEF)
4. ทำการแปล (Compile) โปรแกรม .C ด้วย C Compiler และ Resource Script ด้วย Resource Compiler

5. ทำการเชื่อมโยงโปรแกรมที่แปลแล้วเข้ากับคลังโปรแกรมชนิด Import Library ของ TAM ชื่อ TAM.LIB (import library เป็น library ที่เก็บเลขดัชนีชี้ตำแหน่งของฟังก์ชันใน DLL)

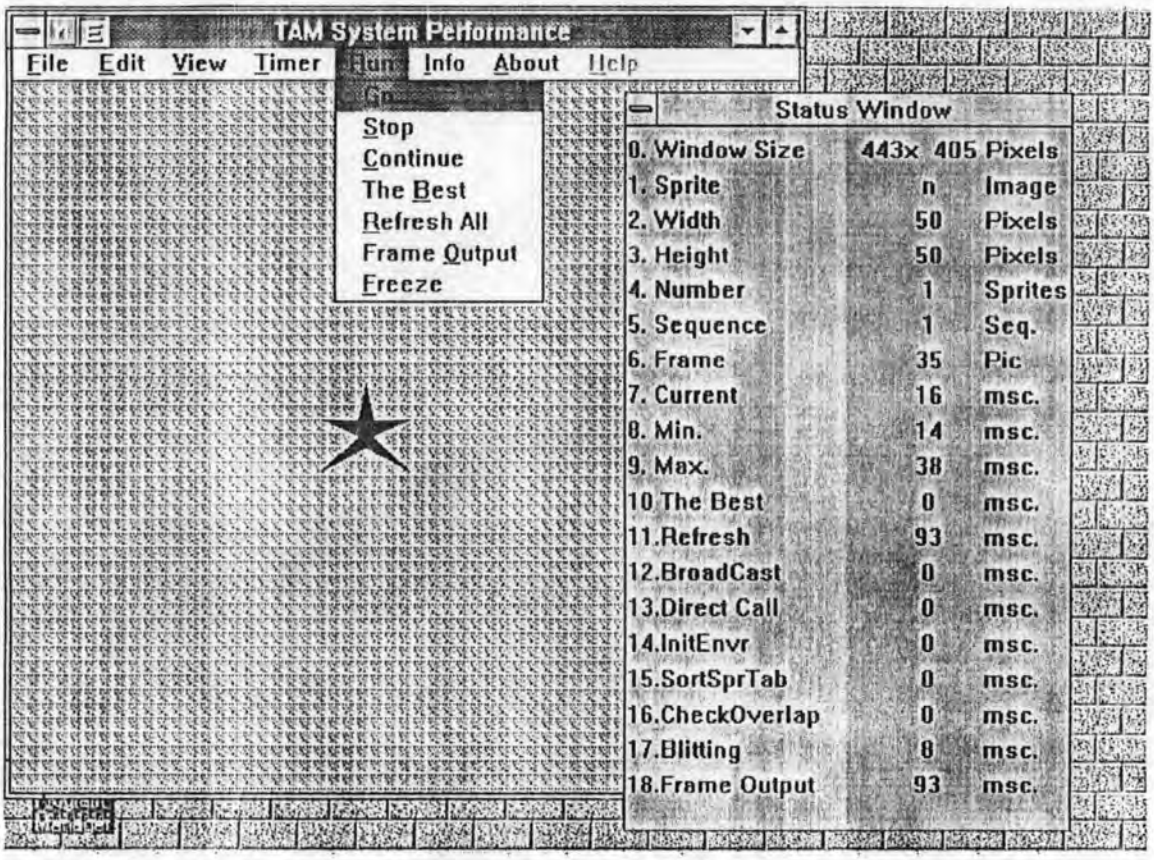
เมื่อเขียนโปรแกรมในส่วนต่างๆเรียบร้อยแล้วจึงนำไปแปลโปรแกรมโดยใช้ Compiler ภาษาซี สำหรับสร้างโปรแกรมภายใต้วินโดวส์

ตัวอย่างต่อไปนี้เป็นการเรียกใช้ Compiler ของบริษัทไมโครซอฟต์
ถ้าโปรแกรมที่พัฒนาขึ้นมาชื่อ Program.C

```
CL      -c -AS -Gws -Os -W3 -Zpe   Program.C
LINK    /NOD /AL:16 Program ,,,libw slibcew TAM,  Program.DEF
RC      Program.RC  Program.EXE
```

ในกรณีที่ใช้ Compiler ของบริษัทบอร์แลนด์ ให้ใส่ชื่อแฟ้มที่เกี่ยวข้องในการแปลและเชื่อมโยงลงใน Project File ตามวิธีการสร้าง Project File ของ Integrated Development Environment (IDE) ของตัว Compiler โดยชื่อแฟ้มที่จะใส่ใน Project File ได้แก่ Program.C Program.RC Program.DEF และ TAM.LIB

เมื่อทำการสร้างโปรแกรมตามข้อกำหนดเรียบร้อยแล้วจะได้โปรแกรมประยุกต์ดังนี้



รูปที่ 4.1 ตัวอย่างวินโดว์ของโปรแกรมทดสอบประสิทธิภาพ

จากรูปที่ 4.1 เป็นวินโดว์ของโปรแกรมทดสอบประสิทธิภาพตามที่กำหนดความต้องการไว้ เมื่อเรียกโปรแกรมขึ้นมาจะปรากฏวินโดว์หลักชื่อ "TAM System Performance" และมีวินโดว์บอกสถานะชื่อ "Status Window" วินโดว์หลักจะทำหน้าที่รับการติดต่อจากผู้ใช้งานรวมทั้งเป็นวินโดว์ที่มีสไปรต์เป็นสมาชิก การรายงานผลการทดสอบประสิทธิภาพจะปรากฏที่วินโดว์ชื่อ "Status Window" ในทันทีที่มีการทำงานเช่น เมื่อมีการวาดจอภาพและสไปรต์ทุกตัวใหม่ (Refresh All) วินโดว์หลักจะส่งข้อมูลต่างๆ ที่เกี่ยวข้องไปยังวินโดว์บอกสถานะทันที

ในการทดสอบประสิทธิภาพเริ่มต้นที่วินโดว์หลักโดยเลือกเมนูต่างๆดังนี้

เมนู File มีเมนูย่อยดังนี้

File : 1 Image = เลือกสไปรต์ชนิด 1 ภาพ หลายขนาด

N Image = เลือกสไปรต์ชนิดหลายภาพ ขนาดเดียว (กว้าง 50 สูง 50 จุดภาพ)

BroadCast Message = เลือกการควบคุมสไปรต์แบบส่งข้อความ

Direct Call = เลือกการควบคุมสไปรต์แบบโดยตรง

No Priority	= ยกเลิกลำดับความสำคัญของสไปรต์ทุกตัว
With Priority	= กำหนดลำดับความสำคัญให้สไปรต์ทุกตัว
Exit	= จบโปรแกรม

เมนู Edit มีเมนูย่อยสำหรับเลือกจำนวนสไปรต์ ตั้งแต่ 1 ถึง 300 ตัว และสุดท้าย Maximum เป็นจำนวนมากที่สุดสำหรับโปรแกรมนี้คือ 500 ตัว

เมนู View มีเมนูย่อยสำหรับเลือกขนาดของสไปรต์

เมนู Timer มีเมนูย่อยสำหรับกำหนดช่วงเวลาการส่งข้อความ WM_TIMER ของตัวจับเวลา

เมนู Run มีเมนูย่อยดังนี้

Run : Go	= ดำเนินโปรแกรมตามข้อกำหนดต่างๆที่เลือกไว้
Stop	= หยุดดำเนินโปรแกรมชั่วคราว
Continue	= ดำเนินโปรแกรมต่อไป
The best	= ดำเนินโปรแกรมในกรณีที่ดีที่สุด แล้วหยุดดำเนินโปรแกรม
Refresh All	= วาดภาพพื้นหลังและสไปรต์ทุกตัว
Frame Output	= ใช้ตัวจักรทำภาพเคลื่อนไหวชนิดวาดใหม่ทั้งวินโดว์
Freeze	= จัดเรียงสไปรต์ในวินโดว์จากซ้ายไปขวา

เมนู Info : ใช้เรียกวินโดว์ "Status Window" หรือ ใช้ลดขนาดของวินโดว์นี้ให้เป็นสัญรูป

เมนู About : ใช้เรียกดูคำอธิบายเกี่ยวกับ TAM

การทดสอบประสิทธิภาพจะเลือกข้อกำหนดดังนี้

1. สไปรต์ชนิด 1 ภาพ จำนวน 1 ตัว

   ขนาดกว้างxสูงดังนี้ 10x10, 20x20, 30x30, 40x40, 50x50, 60x60, 70x70, 80x80, 90x90, 100x100, 150x150 และ 200x200 จุดภาพ (pixels)

2. สไปรต์ชนิดหลายภาพ จำนวน 1 ถึง 50 ตัว ขนาดกว้างxสูง 50x50 จุดภาพ (pixels)
3. สไปรต์ชนิด 1 ภาพ ขนาดกว้างxสูง 50x50 จำนวน 1 ถึง 50 ตัว
4. ขนาดความกว้างxสูง ของพื้นที่ทำงานของวินโดว์ที่ทดสอบคือ 443x405 จุดภาพ (pixels)
5. กำหนดให้สไปรต์เคลื่อนที่ตามแนวนอนทีละ 1 จุดภาพและทางแนวตั้งทีละ 1 จุดภาพ ถ้ามี สไปรต์มากกว่า 1 ตัว ให้แต่ละตัวอยู่ห่างกัน 10 จุดภาพ (เพื่อให้สไปรต์ไม่มีการทับกันเลย)

6. ทดสอบบนเครื่องไมโครคอมพิวเตอร์ยี่ห้อ MITAC รุ่น NoteBook 4020 ใช้ซีพียูเบอร์ 80486 SX-25 หน่วยความจำหลัก 4 เมกะไบต์ การ์ดควบคุมการแสดงผลชนิด VGA 16 สี จอภาพชนิด แอลซีดีโมโนโครม

7. รายการทดสอบได้แก่

- เวลาที่ใช้ในการเคลื่อนไหวของสไปรต์ เมื่อใช้ฟังก์ชัน amAnimate()
- เวลาที่ใช้ในการวาดใหม่ทั้งวินโดว์แสดงผล เมื่อใช้ฟังก์ชัน amRefreshAll()
- กรณีที่สไปรต์ทุกตัวเคลื่อนที่มาอยู่ในตำแหน่งเดียวกันหมด
- เวลาที่ใช้ในการส่งข้อความไปยังสไปรต์ทุกตัว เมื่อใช้ฟังก์ชัน amBroadCastMessage()
- เวลาที่ใช้ในการควบคุมสไปรต์ เมื่อเรียกฟังก์ชันควบคุมสไปรต์แต่ละตัวโดยตรง (ไม่ใช้ฟังก์ชัน amBroadCastMessage())

เมื่อนำข้อกำหนดดังกล่าวไปทดสอบปรากฏว่าได้ผลดังตารางต่อไปนี้

ตารางที่ 4.1 บันทึกข้อมูลประสิทธิภาพเมื่อใช้สไปรต์แบบ 1 ภาพ ขนาดกว้างxยาว 50x50 จุดภาพ

จำนวนสไปรต์	เ ว ล า (มิลลิวินาที)				
	เวลารวม	วาดใหม่ทั้งจอ	กรณีดีที่สุด	การส่งข้อความ	ควบคุมโดยตรง
1	17	96	17	น้อยกว่า 1	น้อยกว่า 1
2	25	98	18	น้อยกว่า 1	น้อยกว่า 1
3	33	105	22	น้อยกว่า 1	น้อยกว่า 1
4	42	109	26	น้อยกว่า 1	น้อยกว่า 1
5	51	113	29	1	น้อยกว่า 1
6	60	118	31	1	น้อยกว่า 1
7	69	121	35	1	น้อยกว่า 1
8	76	126	38	2	1
9	84	131	42	2	1
10	93	135	45	2	1
15	137	159	62	2	1
20	182	182	77	3	2
30	269	224	114	5	3
50	459	311	181	7	5

ตารางที่ 4.2 บันทึกข้อมูลประสิทธิภาพเมื่อใช้สไปรต์แบบหลายภาพ ขนาดกว้างxยาว 50x50 จุดภาพ

จำนวนสไปรต์	เ ว ล า (มิลลิวินาที)				
	เวลารวม	วาดใหม่ทั้งจอ	กรณีดีที่สุด	การส่งข้อความ	ควบคุมโดยตรง
1	16	95	16	น้อยกว่า 1	น้อยกว่า 1
2	24	99	18	น้อยกว่า 1	น้อยกว่า 1
3	33	103	22	น้อยกว่า 1	น้อยกว่า 1
4	42	107	26	น้อยกว่า 1	น้อยกว่า 1
5	50	113	28	1	น้อยกว่า 1
6	58	118	35	1	น้อยกว่า 1
7	66	122	38	1	น้อยกว่า 1
8	76	126	41	2	1
9	84	130	43	2	1
10	93	135	46	2	1
15	136	156	63	2	1
20	179	179	78	3	2
30	269	222	112	5	3
50	459	308	178	7	5

ตารางที่ 4.3 บันทึกข้อมูลประสิทธิภาพเมื่อใช้สไปรต์แบบ 1 ภาพ ขนาดต่างกัน จำนวน 1 ตัว

ขนาดของสไปรต์ กว้างxสูง (pixels)	เ ว ล า (มิลลิวินาที)		
	เวลารวม	วาดใหม่ทั้งจอ	ตัวจักรแบบเต็มจอ
10x10	10	92	93
20x20	11	93	93
30x30	12	94	94
40x40	14	95	95
50x50	16	96	96
60x60	19	98	98
70x70	22	99	99
80x80	25	100	100
90x90	28	103	104
100x100	32	105	105
150x150	54	116	116
200x200	86	134	134

ตารางที่ 4.4 บันทึกข้อมูลประสิทธิภาพเมื่อใช้สไปรต์แบบ 1 ภาพ ขนาดกว้างxยาว 10x10 จุด

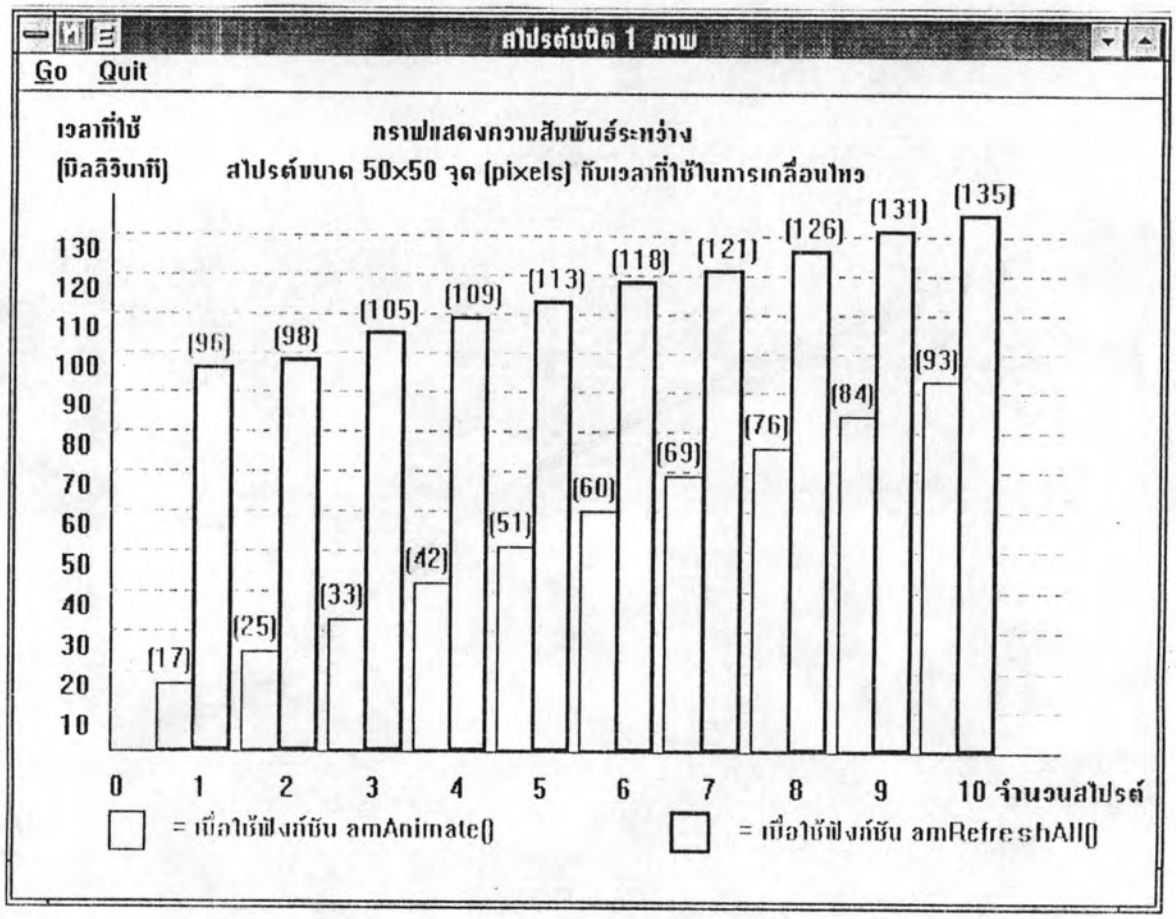
จำนวนสไปรต์	เ ว ล า (มิลลิวินาที)				
	เวลารวม	วาดใหม่ทั้งจอ	ตัวจักรแบบเต็มจอ	ส่งข้อความ	เช็คการชนทับ
1	10	92	92	น้อยกว่า 1	น้อยกว่า 1
2	12	94	94	น้อยกว่า 1	น้อยกว่า 1
3	14	95	95	น้อยกว่า 1	น้อยกว่า 1
4	17	96	97	น้อยกว่า 1	น้อยกว่า 1
5	19	97	98	1	น้อยกว่า 1
6	21	99	100	1	น้อยกว่า 1
7	24	101	103	1	น้อยกว่า 1
8	26	103	104	2	1
9	28	107	107	2	1
10	31	108	109	2	2
15	43	108	111	2	3
20	55	113	115	3	6
30	82	125	125	5	10
50	145	145	149	7	27
100	349	195	207	13	113
200	937	298	320	27	427
300	1775	412	450	47	969
500	4227	630	669	65	2555

ตารางที่ 4.5 เปรียบเทียบข้อมูลประสิทธิภาพเมื่อใช้สไปรต์แบบ 1 ภาพ ขนาดกว้างxยาว 10x10 จุด ภาพ เมื่อใช้ขั้นตอนวิธีแบบเดิมกับแบบใหม่

จำนวนสไปรต์	มิลลิวินาที			
	เวลาทั้งหมด		เช็คการซ้อนทับกัน	
	วิธีเดิม	วิธีใหม่	วิธีเดิม	วิธีใหม่
30	82	47	10	2
50	145	77	27	5
100	349	159	113	23
200	937	370	427	100
300	1775	654	969	225
500	4227	1331	2555	642

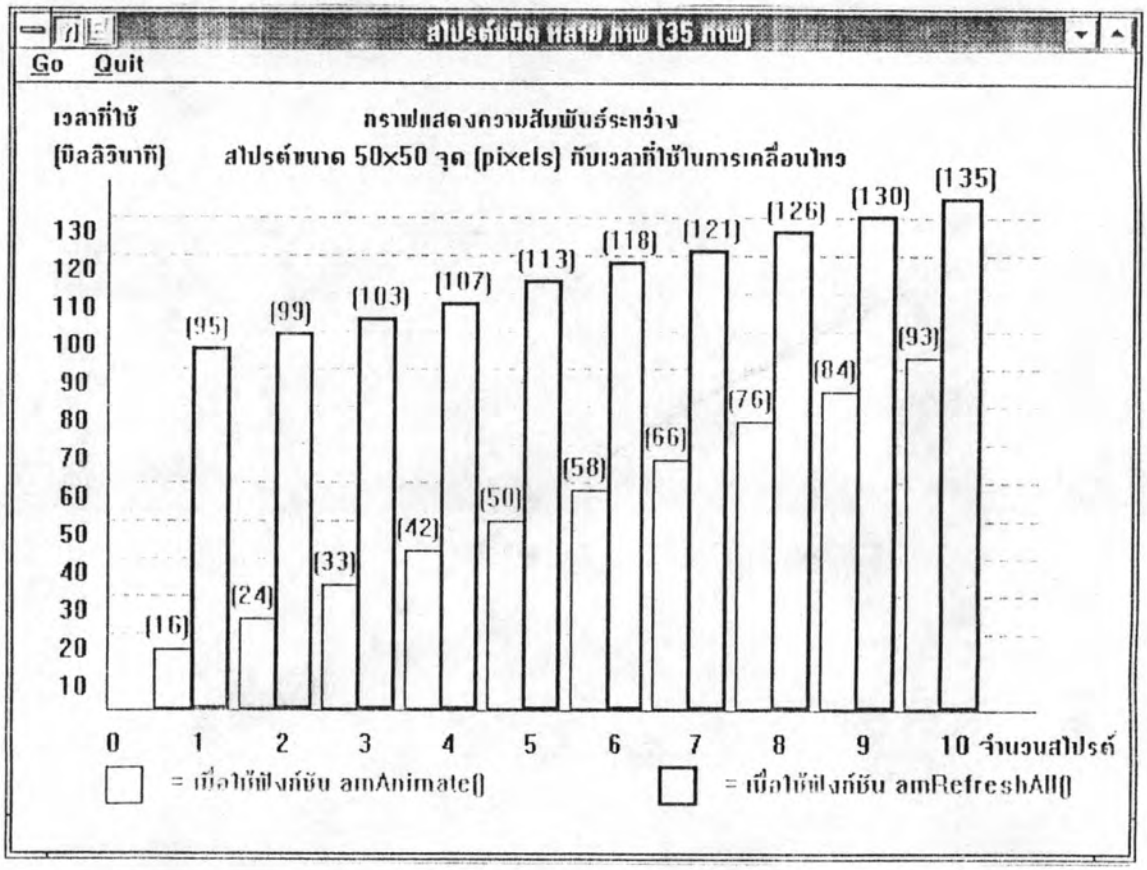
จากข้อมูลที่ได้ตามตารางที่ 4.1, 4.2, 4.3 สามารถนำมาเขียนเป็นกราฟได้ดังนี้

นำข้อมูลจากตารางที่ 4.1 มาเขียนเป็นกราฟแท่งแสดงความสัมพันธ์ ระหว่างจำนวนสไปรต์ชนิด 1 ภาพขนาดกว้าง 50 จุดภาพ สูง 50 จุดภาพ กับเวลาที่ตัวจักรทำภาพเคลื่อนไหวใช้จัดการกับสไปรต์ในแต่ละครั้ง



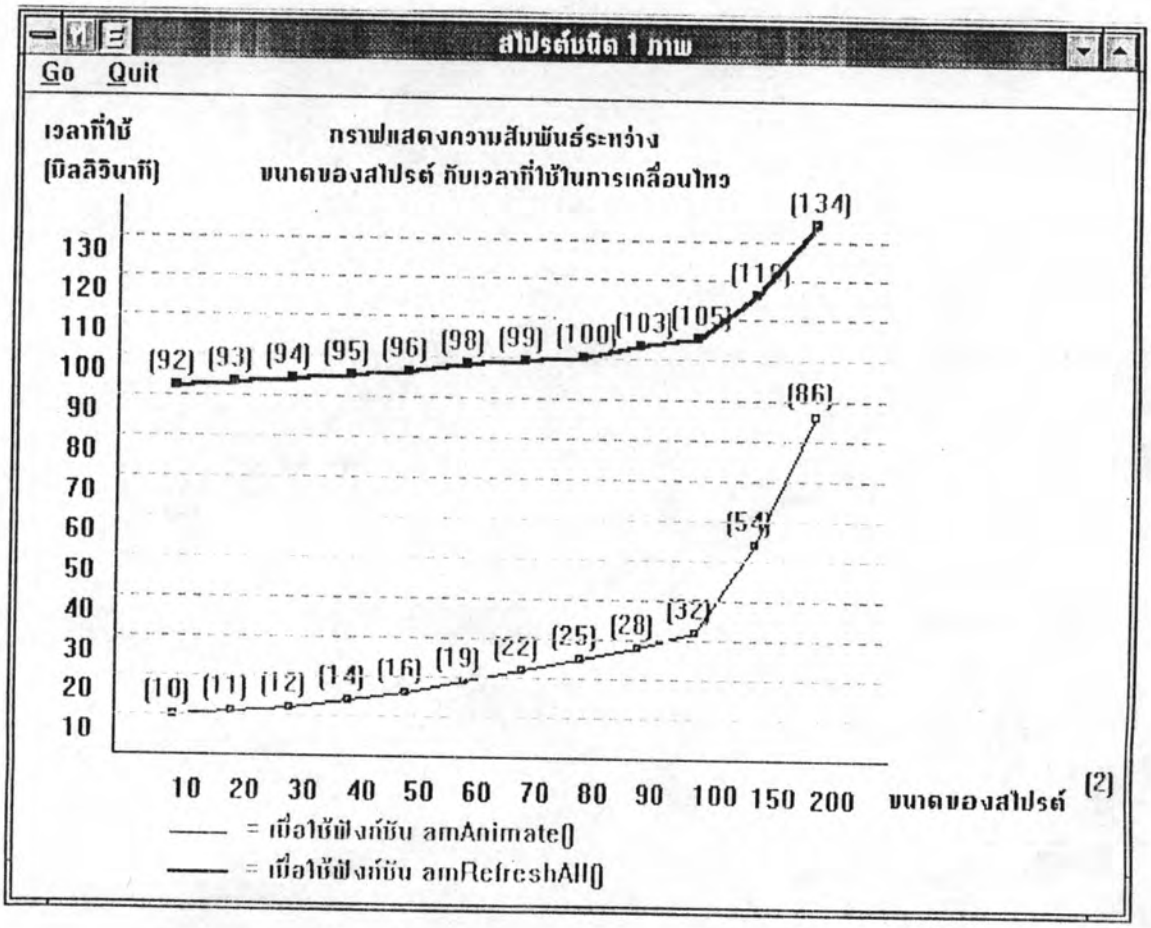
รูปที่ 4.2 กราฟแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate() และ amRefreshAll() เมื่อใช้สไปรต์ชนิด 1 ภาพขนาด 50x50 จุดภาพ

นำข้อมูลจากตารางที่ 4.2 มาเขียนเป็นกราฟแท่งแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์ชนิดหลายภาพ (ในการทดสอบใช้ 35 ภาพ) ขนาดกว้าง 50 จุดภาพ สูง 50 จุดภาพ กับเวลาที่ตัวจักรทำภาพเคลื่อนไหวใช้จัดการกับสไปรต์ในแต่ละครั้ง



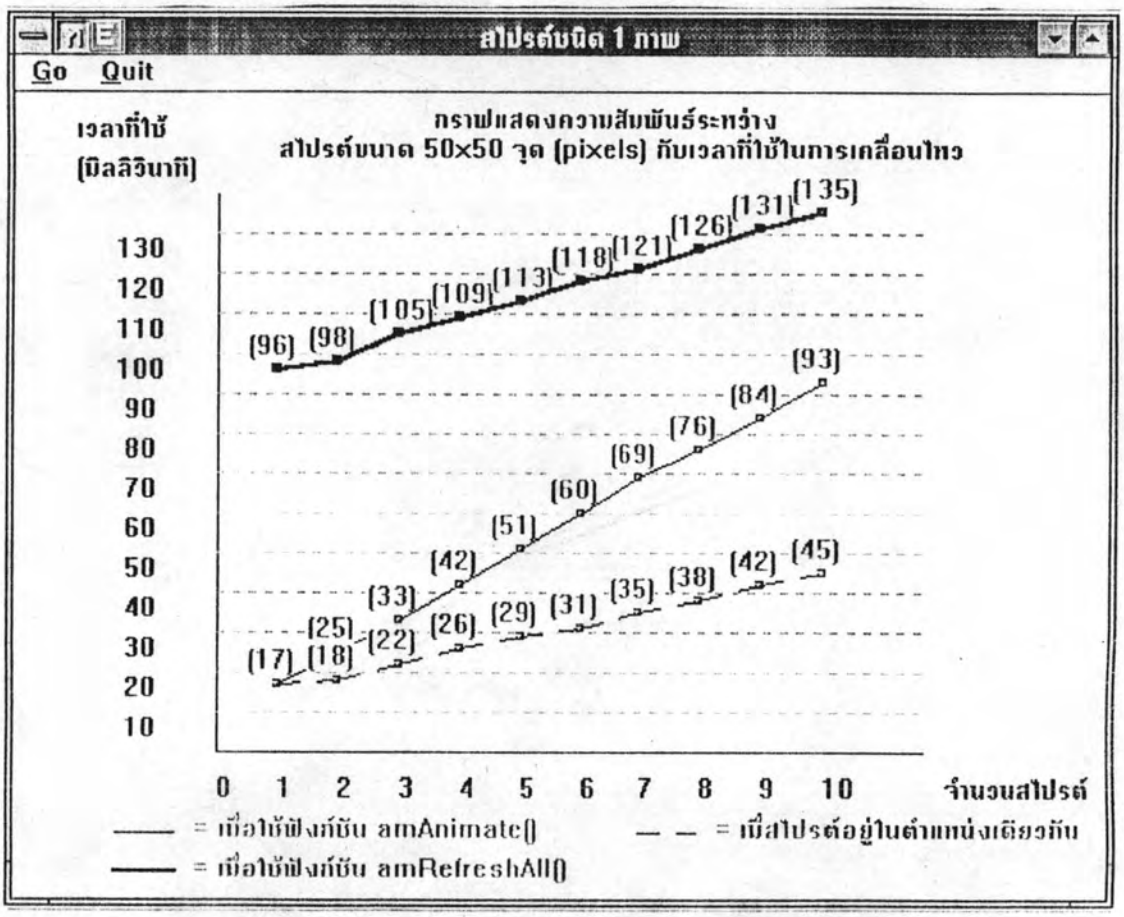
รูปที่ 4.3 กราฟแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate() และ amRefreshAll() เมื่อใช้สไปรต์ชนิด 35 ภาพ ขนาด 50x50 จุดภาพ

นำข้อมูลจากตารางที่ 4.3 มาเขียนเป็นกราฟเส้นแสดงความสัมพันธ์ระหว่างขนาดของสไปรต์กับเวลาที่ตัวจักรทำภาพเคลื่อนไหวใช้



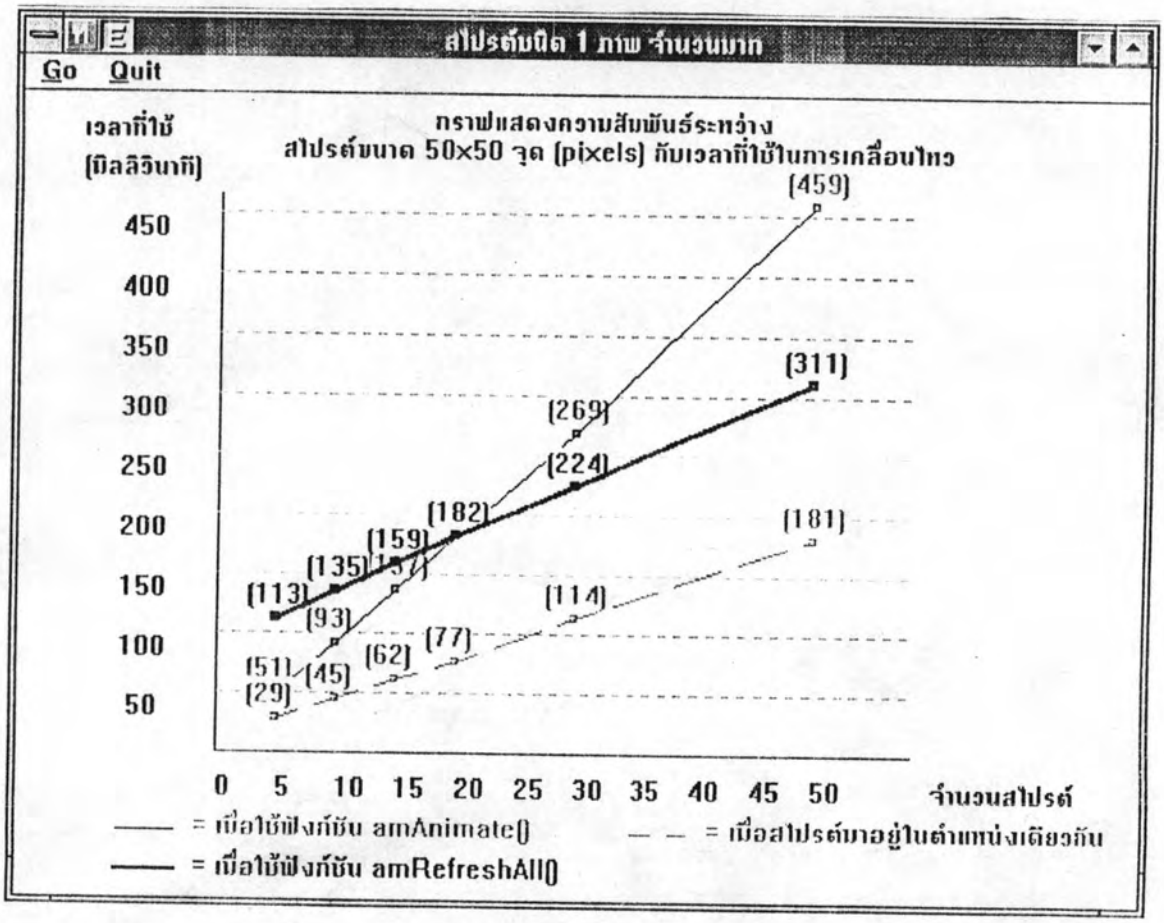
รูปที่ 4.4 กราฟแสดงความสัมพันธ์ระหว่างขนาดของสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate() และ amRefreshAll() เมื่อใช้สไปรต์ชนิด 1 ภาพ

เมื่อนำข้อมูลจากตารางที่ 4.1 มาเขียนเป็นกราฟเส้นแสดงความสัมพันธ์ระหว่างสไปรต์ขนาดกว้าง 50 จุดภาพ สูง 50 จุดภาพ กับเวลาที่ใช้ เมื่อมีสไปรต์จำนวนไม่เกิน 10 ตัวจะได้ดังนี้



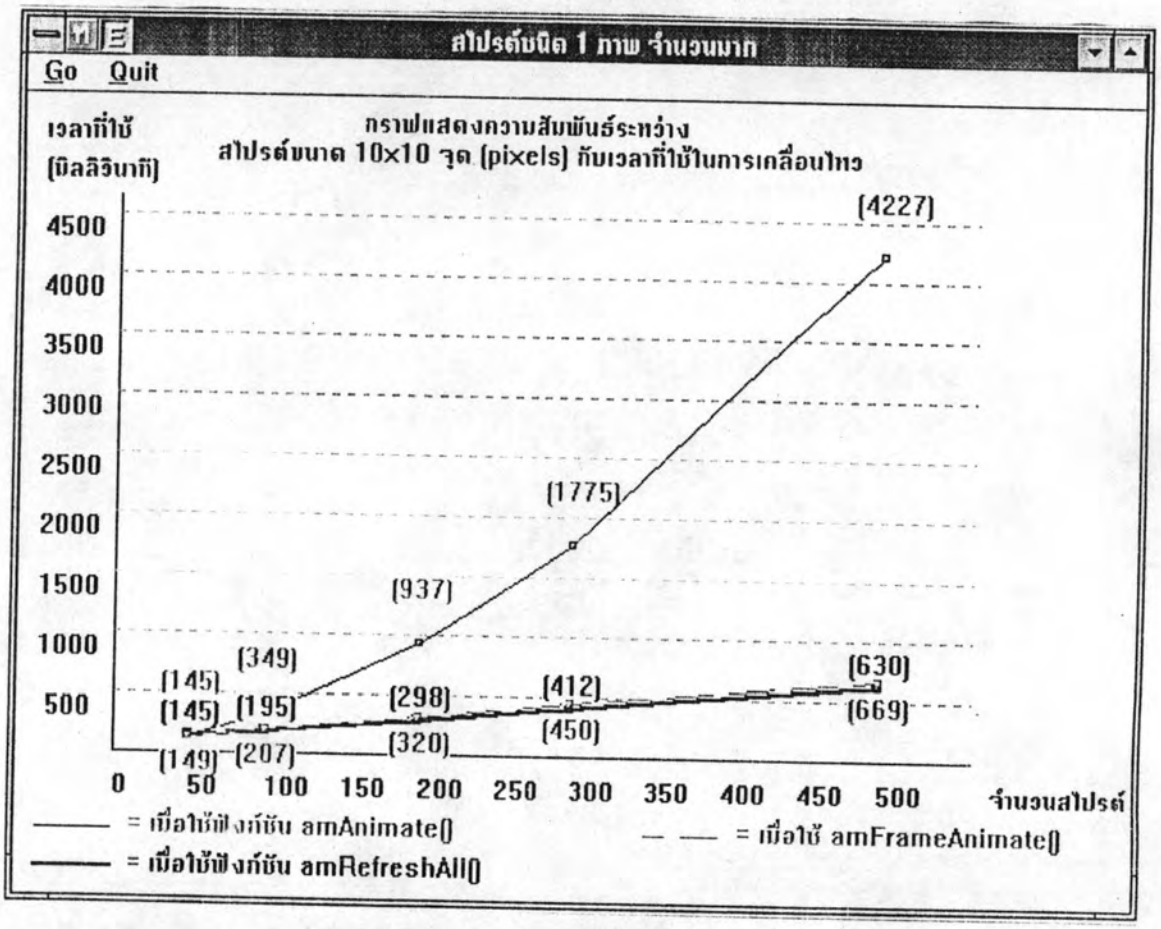
รูปที่ 4.5 กราฟแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate(), amRefreshAll() เปรียบเทียบกับกรณีดีที่สุดโดยใช้สไปรต์ชนิด 1 ภาพขนาด 50x50 จุดภาพ

เมื่อนำข้อมูลจากตารางที่ 4.1 มาเขียนเป็นกราฟเส้นแสดงความสัมพันธ์ระหว่างสไปรต์ขนาดกว้าง 50 จุดภาพ สูง 50 จุดภาพ กับเวลาที่ใช้ เมื่อมีสไปรต์จำนวนไม่เกิน 50 ตัวจะได้ดังนี้



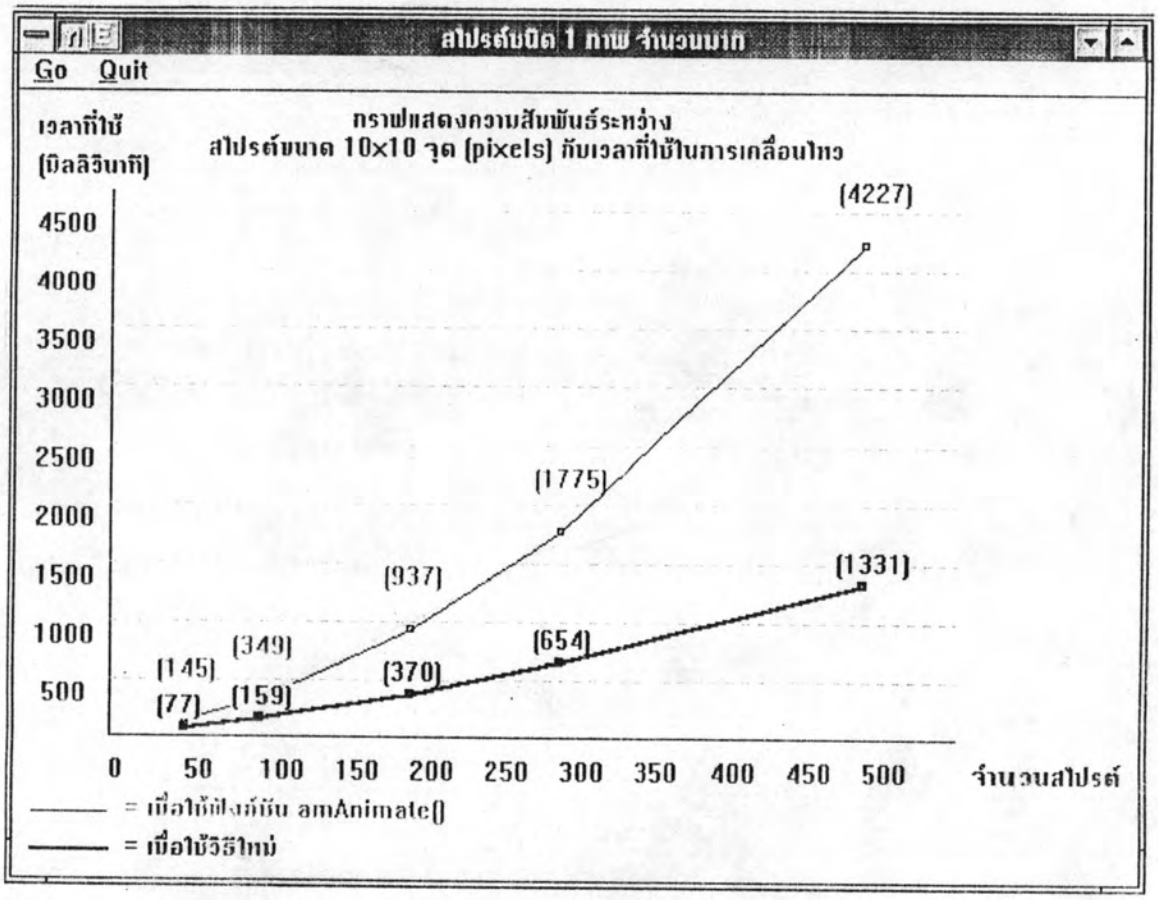
รูปที่ 4.6 กราฟแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหว
ด้วยฟังก์ชัน amAnimate(), amRefreshAll() เปรียบเทียบกับกรณีดีที่สุด
โดยใช้สไปรต์ชนิด 1 ภาพขนาด 50x50 จุดภาพ
เมื่อมีสไปรต์จำนวนมากขึ้น

นำข้อมูลจากตารางที่ 4.4 มาเขียนเป็นกราฟเส้นแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ตัวจักรทำภาพเคลื่อนไหวใช้ เมื่อสไปรต์มีขนาดกว้าง 10 จุดภาพ สูง 10 จุดภาพ ได้ดังนี้



รูปที่ 4.7 กราฟแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหว
ด้วยฟังก์ชัน amAnimate(), amRefreshAll() และ amFrameAnimate()
โดยใช้สไปรต์ชนิด 1 ภาพขนาด 10x10 จุดภาพ
เมื่อมีสไปรต์จำนวนมากๆ

นำข้อมูลจากตารางที่ 4.5 มาเขียนเป็นกราฟเส้นแสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ใช้ เมื่อสไปรต์มีขนาดกว้าง 10 จุดภาพ สูง 10 จุดภาพ โดยใช้ขั้นตอนวิธีแบบเดิมและแบบใหม่ ได้ดังนี้



รูปที่ 4.8 เปรียบเทียบข้อมูลประสิทธิภาพเมื่อใช้สไปรต์แบบ 1 ภาพ ขนาดกว้างxยาว 10x10 จุดภาพ เมื่อใช้ขั้นตอนวิธีแบบเดิมกับแบบใหม่

จากข้อมูลที่ได้จากการทดสอบทั้งสี่ตารางและนำมาเขียนกราฟแสดงความสัมพันธ์ดังที่ผ่านมาสามารถสรุปได้ดังนี้

1. จากกราฟแท่งในรูปที่ 4.2 แสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate() และ amRefreshAll() เมื่อใช้สไปรต์ชนิด 1 ภาพ ขนาด 50x50 จุดภาพ เปรียบเทียบกับกราฟในรูปที่ 4.3 แสดงความสัมพันธ์เช่นเดียวกับกราฟในรูปที่ 4.2 แต่เป็นสไปรต์ชนิดหลายภาพ (ในการทดสอบใช้ 35 ภาพ) ปรากฏว่าเวลาที่ใช้ฟังก์ชัน amAnimate() และการวาดใหม่หมดของกราฟทั้งสองรูปไม่ต่างกันเท่าไรดังนั้นสรุปได้ว่า จำนวนภาพของสไปรต์ไม่มีผลกระทบต่อความเร็วในการทำงานของตัวจักรทำภาพเคลื่อนไหว แต่เวลาในการทำภาพเคลื่อนไหวจะแปรตามจำนวนสไปรต์

2. จากกราฟเส้นในรูปที่ 4.4 แสดงความสัมพันธ์ระหว่างขนาดของสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate() และ amRefreshAll() เมื่อใช้สไปรต์ชนิด 1 ภาพ จำนวน 1 ตัว ปรากฏว่าเมื่อสไปรต์มีขนาดใหญ่ขึ้น (ในการทดสอบนี้ขนาดของสไปรต์คือขนาดพื้นที่ของกรอบภาพของสไปรต์) เวลาที่ใช้ในการทำภาพเคลื่อนไหวแต่ละครั้งก็จะมากขึ้นด้วย ดังนั้นสรุปได้ว่า เวลาในการทำภาพเคลื่อนไหวจะแปรตามขนาดพื้นที่ของสไปรต์

3. จากกราฟเส้นในรูปที่ 4.5 แสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาทีทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate() และ amRefreshAll() เปรียบเทียบกับกรณีดีที่สุด โดยใช้สไปรต์ชนิด 1 ภาพขนาด 50x50 จุดภาพ (กรณีดีที่สุดคือเมื่อสไปรต์ทุกตัวอยู่ในตำแหน่งเดียวกันและไม่เคลื่อนที่ไปไหน) ปรากฏว่าเมื่อสไปรต์ไม่เคลื่อนที่โดยอยู่ทับกันหมดจะใช้เวลาน้อยกว่าสไปรต์ที่มีการเคลื่อนที่ (เมื่อมี สไปรต์มากกว่า 1 ตัว) ดังนั้นสรุปได้ว่า ระยะทางในการเคลื่อนที่ของสไปรต์มีผลกระทบต่อการทำภาพเคลื่อนไหวเช่นกัน

4. จากกราฟเส้นในรูปที่ 4.6 แสดงความสัมพันธ์ระหว่าง จำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate() และ amRefreshAll() เปรียบเทียบกับกรณีดีที่สุดโดยใช้สไปรต์ชนิด 1 ภาพขนาด 50x50 จุดภาพเมื่อมีสไปรต์จำนวนมากปรากฏว่าเมื่อสไปรต์มีจำนวนมากขึ้นฟังก์ชัน amAnimate() จะใช้เวลามากกว่าฟังก์ชัน amRefreshAll() สรุปได้ว่า เมื่อมีสไปรต์จำนวนมากๆการวาดใหม่ทั้งจอภาพน่าจะให้ผลงานที่ดีกว่า

5. จากกราฟเส้นในรูปที่ 4.7 แสดงความสัมพันธ์ระหว่างจำนวนสไปรต์กับเวลาที่ทำการเคลื่อนไหวด้วยฟังก์ชัน amAnimate(), amRefreshAll() และ amFrameAnimate() โดยใช้สไปรต์ชนิด 1 ภาพขนาด 10x10 จุดภาพ เมื่อมีสไปรต์จำนวนมากๆ (ในการทดสอบนี้ได้เพิ่มการเปรียบเทียบกับฟังก์ชัน amFrameAnimate() ซึ่งเป็นตัวจักรที่มีกลไกการทำงานตามวิธีการวาดใหม่ทั้งจอภาพ คล้ายกับฟังก์ชัน amRefreshAll) ปรากฏว่า เมื่อมีสไปรต์จำนวนมากๆ การทำงานของฟังก์ชัน amAnimate() จะใช้เวลามากขึ้น เมื่อถึงจำนวนหนึ่ง ปรากฏว่าใช้เวลามากกว่าวิธีการวาดใหม่ทั้งจอภาพ (ในที่นี้คือพื้นที่ของ

วินโดว์ที่แสดงผล) ดังนั้นจึงเป็นการสนับสนุนการสรุปในข้อ 4 คือเมื่อมีจำนวนสไปรต์มากๆการวาดใหม่ทั้งจอภาพน่าจะให้ผลที่ดีกว่า เมื่อย้อนกลับไปดูที่ตารางที่ 4.4 จะพบว่าเมื่อมีสไปรต์จำนวนมากขึ้นการส่งข้อความควบคุมไปให้สไปรต์เหล่านั้นก็จะใช้เวลามากขึ้นตามไปด้วยแต่จะมีผลกระทบมากเมื่อมีการส่งข้อความให้สไปรต์จำนวนมากกว่าหนึ่งร้อยตัวพร้อมกัน ในตารางที่ 4.4 ยังมีข้อมูลรายงานถึงผลกระทบที่สำคัญอีกอย่างคือขั้นตอนการตรวจสอบการทับกันของสไปรต์ในตัวจักรทำภาพเคลื่อนไหว ซึ่งใช้ในการคำนวณหาพื้นที่ที่เป็น offscreen สำหรับการ update ข้อมูลบนจอภาพ ปรากฏว่าถ้าสไปรต์ไม่มีการทับกันเลย (ดังโปรแกรมที่ใช้ทดสอบนี้) และมีจำนวนมากๆ ก็จะมีผลกระทบต่อการทำภาพเคลื่อนไหวด้วย

6. ผลที่ได้ในข้อที่ 5 จะเห็นว่าเมื่อสไปรต์มากขึ้นเวลาที่ใช้ทำการเคลื่อนไหวจะมากขึ้นด้วย เมื่อพิจารณาจากตารางที่ 4.4 จะพบว่าการตรวจสอบการซ้อนทับกันเพื่อลดจำนวนการ update ภาพบนจอเป็นสาเหตุสำคัญอย่างหนึ่ง ดังนั้นผู้วิจัยจึงได้นำขั้นตอนวิธีในการตรวจสอบการซ้อนทับกันของสไปรต์ที่ได้ออกแบบใหม่คือตรวจสอบเฉพาะตัวที่เป็นไปได้ว่าจะทับกันมาทดสอบ ปรากฏผลดังข้อมูลในตารางที่ 4.5 และเมื่อนำมาวาดเป็นกราฟเส้นดังรูปที่ 4.8 จะเห็นว่า ขั้นตอนวิธีแบบใหม่ สามารถลดเวลาการตรวจสอบการซ้อนทับกันได้ทำให้ระบบมีประสิทธิภาพมากขึ้นประมาณ 42 - 68 % เมื่อเทียบกับวิธีเดิม