6.837 — FALL '98

Midterm Quiz

Time allotted: 1 hour 15 minutes

Open book, open notes
(100 points)

Instructions: Answer all questions using the space provided. If you need more paper, raise your hand, and it will be provided for you. If you have a question during the test, do not leave your seat. Instead, raise your hand until someone comes to assist you. If any question seems unclear or appears to lack sufficient information, state a reasonable assumption (in writing on your quiz) and proceed. The point value of each problem is indicated in parenthesis. Make sure that you answer each part of each question, even if you believe you have already answered the question in an earlier part.

Solutions

Name_____
(Print)

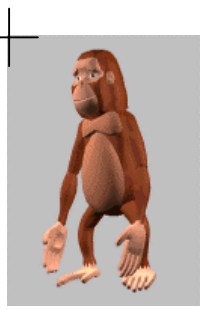
I understand that this quiz is subject to the
conditions and procedures stated in
MIT Policies and Procedures handbook in Section 10.2.
Furthermore, I neither gave, nor received, any
unsanctioned assistance during this examination.

Signed _____ Date _____

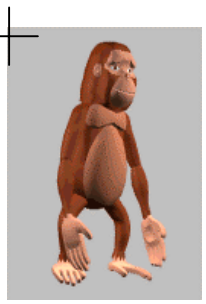
RASTER ADDRESSING: You are given the following familiar objects and interfaces.

```
public class Raster {
    protected int width,  height;          // dimensions of Raster
    public int pixel[];                    // width * height array of pixels

    public Raster( );
    public Raster(int width, int height);
    public Raster(Image img);
    public int size( ) ;
    public int getWidth() ;
    public int getHeight() ;
    public int [] getPixels() ;
    public void fill(Color c);
    public Image toImage();
    public int getPixel(int x, int y);
    public Color getColor(int x, int y);
    public boolean  setPixel(int pixvalue, int x, int y);
    public boolean  setColor(Color c, int x, int y);
}

public class Sprite extends Raster {
    public int x, y;                   // upper-left coordinate of sprite on playfield

    public Sprite(Image img, int x, int y);
    public void draw(Raster playfield);
}
```

flipped = false



flipped = true



Implement the draw() method of a FlipSprite object with the following interface:

```
public class FlipSprite extends Sprite {
    public boolean flipped;

    public FlipSprite(Image img, int x, int y) {
            super(img, x, y);
            flipped = false;
    }

    public void draw(Raster playfield) {
            // YOUR CODE HERE
    }
}
```

When "flipped" is false the sprite should be drawn normally. When "flipped" is true the sprite should be drawn mirrored from left to right as shown.

Part A: (10 points) Write a code fragment to implement the draw() method with proper clipping and transparency. You should not need to alter the other methods. Assume the default Java color model, also assume (pixelValue >> 24) == 0 indicates a transparent pixel, and any other value is opaque.

(additional room for Part A)

```
public void draw(Raster playfield) {
    if (flipped) {
        for (int j = 0; j < height;  j++) {
            if ((y+j >= 0) && (y+j) < playfield.height) {
                for (int i = 0;  i < width; i++)
                    if ((x+i >= 0) && (x+i) < playfield.width) {
                        int pix = getPixel(width-i-1, j);
                        if ((pix >> 24) != 0)
                            playfield.setPixel(pix, x+i, y+j);
                    }
            }
        }
    } else
        super.draw(playfield);
}
```

Part B: (5 points) Suppose you are asked to debug someone else's FlipSprite object. Discuss the symptoms of an incorrect implementation of x-clipping and y-clipping. Consider both minimum and maximum extents. Assume the Raster object implementation given in class.

Without proper clipping in x against the play field's width the sprite appears to wrap around to the left side of the play field and it is shifted down by one pixel position. Similarly, if the sprite's x coordinate is not properly clipped against 0 then the sprite appears to wrap around to the right side of the play field, but shifted up one pixel. Improper clipping in y generates an ArrayIndexOutOfBoundsException for both y values less than zero and those greater than or equal to the play field's height.

Part C: (5 points) Write an expression for the index where the (i, j) pixel of a sprite will be written in the pixel[ ] array of the playfield raster. Assume flipped is false, $0 \le i <$ playfield.width, and $0 \le j <$ playfield.height.

```
playfield.pixel[(y+j)*playfield.width + (x+i)]
```

LINE DRAWING ALGORITHMS: Movies developed for theaters are usually shot at 24 frames per second, while video on television is displayed at 30 frames per second. Thus, when motion pictures are transferred to video formats they must undergo a process called "pull-down", where every fourth frame is repeated thus matching the frame rates of the two media, as depicted below.

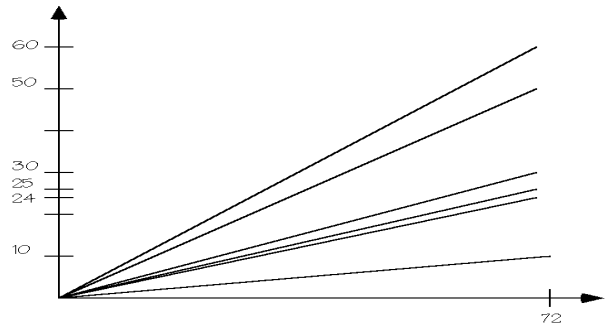
Motion Picture	1	2	3	4	4	5	6	7	8	8	9	10	11	12	12	13	14	15	16	16	17	18	19	20	20	21	22	23	24	24
Video	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Once, I was employed by a company with a similar, yet more complicated problem. They had developed a multimedia movie-player technology that they wished to display on a workstation's CRT screen having a 72 frames-per-second update rate. They planned to support a wide range of source materials including, but not limited to:

Source	Frame rate
Motion Pictures	24 fps
US Video (frames)	30 fps
US Video (fields)	60 fps
European Video (frames)	25 fps
European Video (fields)	50 fps
Common Multimedia	10 fps

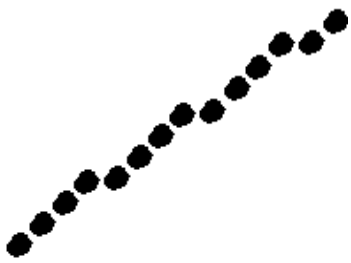
These source materials were to be displayed as close as possible to the correct frame rate. We were not expected to support frame rates higher than the CRT's update rate. Upon having the problem explained, someone commented, "It's just a line-drawing algorithm!"

Part A: (5 points) Sketch the generalized pull-down problem as a series of ideal lines. Draw the CRT's update rate along the x-axis and the various source material frame rates along the y-axis.



Part B: (10 points) Explain how this generalized version of the "pull-down" problem is similar to a discrete approximation to these ideal lines. Explain the significance of a discrete line with multiple pixels set in the same row. Relate this to the motion picture/video example given earlier.

An "ideal" pull-down process would need to start source frames at arbitrary points in the middle of update frames. But, update frames are discrete things. Thus we must settle on a discrete approximation that is as close as possible to the ideal pull-down. In a discrete pull-down algorithm the case when a frame is repeated is like the case when two pixels are drawn on the same row of a raster by a discrete line algorithm. The discrete line that matches the motion picture/video example given looks something like:



Part C: (7 points) Suggest the most appropriate line-drawing algorithm for this problem, choosing from those that we discussed in class. Justify your answer.

Bresenham's algorithm seems like the best choice here. It requires no divides and no floating point calculations. It gives exactly the same answer as a DDA when the line's end points fall on integer boundaries, as in the discrete pull-down algorithm. Breseneham's algorithm is better suited than the two-step because we'll probably want to evaluate it one update frame at a time to decide whether to use the old source frame again or step to the next one.

Part D: (8 points) What optimizations might be applied to the generalized pull-down problem beyond those discussed in class (Hint: I can think of at least two).

1) All the lines start at (0, 0) and progress toward (72, X) so we can avoid the subtraction to compute the slope.

2) Step values are always +1 in x and y.

3) All lines have slopes less than one and greater than zero, and we don't have to compare dx to dy to decide whether to step in x or y.

SCAN-CONVERTING FILLED PRIMITIVES: A disc-filling algorithm has been developed that closely resembles the edge-equation-based triangle-drawing algorithm that we covered in class. The key insight was that by using a look-up table containing the squares of the positive integers up to the greater of the width or height of a raster, the process of filling a circular conic could be reduced to the evaluation of a linear expression. In this problem you will derive this algorithm.

Part A: (10 points) Find the discriminating function, which is also an edge equation in this case, for a circular disc with a center at (x_0, y_0) and a radius of r. Reminder: General conics have the form

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0.$$

I am asking for the expressions for A, B, C, D, E, and F in terms of x_0, y_0 and r.

The equation of a circle is

$$(x - x_O)^2 + (y - y_O)^2 = r^2$$

this implies

$$x^2 + y^2 - 2x_Ox - 2y_Oy + x_O^2 + y_O^2 - r^2 = 0$$

thus

$$f(x, y) = x^2 + y^2 - 2x_Ox - 2y_Oy + x_O^2 + y_O^2 - r^2$$

and

$A=1$, $B=0$, $C=1$, $D=-2x_O$, $E=-2y_O$, and $F= x_O^2 + y_O^2 - r^2$.

Part B: (5 points) Is the sign of your edge equation positive or negative in the interior of the circle?

We know that (xO, yO) is an interior point. At that point $f(x_O, y_O) = -r^2$, therefore the edge equation is negative inside of the disc.

Part C: (10 points) Give expressions for computing the bounding box of the disc. Compute the integer valued extents x_{min}, x_{max}, y_{min}, and y_{max}. Assume that x_0, y_0 and r are given as floating point numbers.

```
xmin = (int) (xO - r)
xmax = (int) (xO + r + 1)
ymin = (int) (yO - r)
ymax = (int) (yO + r + 1)

if (xmin < 0) xmin = 0;
if (xmax >= raster.width) xmax = raster.width - 1;
if (ymin < 0) ymin = 0;
if (ymax >= raster.height) ymax = raster.height - 1;
```

Part D: (10 points) Consider the following Java implementation of the circular-disk algorithm.

```
public class FlatDisc implements Drawable {
        float x0, y0, r;
        int color;

        public FlatDisc(float x, float y, float radius, int fillColor)
        {
                x0 = x;  y0 = y;  r = radius;
                color = fillColor;
        }

        // discSetup( ) initializes all of the following variables;
        protected float A, B, C, D, E, F;
        protected int xmin, xmax, ymin, ymax;
        protected static float Square[ ];

        protected void discSetup( Raster raster)
        {
                // some code based on your answers for parts B and C
                // You don't need to write it to answer this part
        }

        public void draw(Raster raster)
        {
                discSetup(raster);
                // Your code goes here
        }
}
```

Write the missing code fragment of the draw() method which scan converts a disk by filling its interior with the value given by the variable "color". You can assume that the discSetup( ) method initializes the values of *A, B, C, D, E, F*, x_{min}, x_{max}, y_{min}, and y_{max} in the manner you specified in parts B and C. Note also that all coefficients are floats, and **I recommend that you evaluate your edge expression using floating point rather than using integers as I did in class** (You are going to be graded on correctness, not speed). You can also assume that the table *Square[]*, has been pre-computed, and it contains all the squares of integers from 0 to the greater of the width and height minus 1. You should ignore edge-equation coefficients that, because of their values, do not require a multiplication.

```
public raster draw(Raster r) {
    discSetup(raster);
    float t = D*xmin + E*ymin + F;

    for (int y = ymin; y <= ymax; y++) {
        float edge = Square[y] + t;
        boolean beenInside = false;
        for (int x = xmin; x <= xmax; x++) {
            if (Square[x] + edge <= 0) {
                raster.setPixel(color, x, y);
                beenInside = true;
            } else if (beenInside) break;
            edge += D;
        }
        t += E;
    }
}
```

(additional room for Part D)

Part E: (10 points) Write a difference equation that computes the successive entries of the Square[ ] lookup table. Be sure to specify the initial table entry. Assume the variable n has been computed as follows:

```
n = (raster.width >  raster.height) ? raster.width - 1 : raster.height - 1;
```

$f(x) = x^2$ and $f(x+1) = x^2 + 2x + 1$

thus

$f(x+1) = f(x) + 2x + 1$

The following loop would fill the array:

```
Square[0] = 0;
for (int i = 0;  i < n-1; i++)
        Square[x+1] = Square[x] + x + x + 1;
```

Part F: (5 points) Do you consider the use of the difference equation found in Part E worthwhile for this application? Discuss its advantages and disadvantages.

It's probably not worthwhile to use the difference equation from Part E, because the table only needs to be generated once, and it is computed outside of the loop. The advantage of the difference equation is that it is fast since it has no multiplies. The disadvantage is that it is not self-explanatory what the loop is computing, especially when compared to the line:

```
Square[x] = x * x;
```