CS 184, Fall 1996
Midterm #1
Professor: unknown

Problem #1, Transformations (8pts)

All questions assume a **right handed coordinate system**.

Circle the correct answer: (2 pts each)

a) In 3 space, two rotations about arbitrary axes can alaways be applied in either order to get the same result.

True / False

b) In 3-space, two rotations about principal axes can always be applied in either order to get the same result.

True / False

c) In 3-space, two uniform scalings (scaling with different scale factors in x, y, and z) can always be applied in either order to get the same result.

True / False

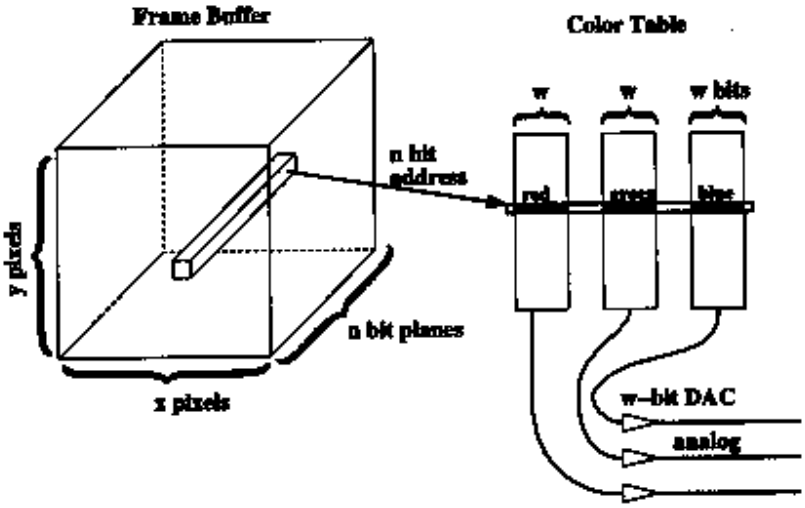
Short answer: (2 pts each)

d) In a right handed coordinate system, in what direction does the positive y-axis point after a 90 degree rotation around the positive x-axis?

Problem #2, Display Hardware (12 pts)

Suppose we have a frame buffer with

- *x* x *y* pixels
- *n* bit planes for an index into the color table
- *w* bits for each of red, green, and blue in the color table



MAKE *NO ASSUMPTIONS* ABOUT THE RELATIVE MAGNITUDES OF *x*, *y*, *n*, AND *w*.

a) How many colors can we choose from for our color table?

b) What is the maximum number of different colors we can have in our color table at one time?

c) What is the maximum number of different colors that can be displayed on our screem at one time?

Problem #3, Vanishing Points (18 pts)

In this problem, we use this perspective projection:

- The eye is at the point z=1 on the z-axis: COP = (0,0,1).
- The projection plane is at z=2: VRP = (0,0,2).
- The up vector is along the y-axis: VUP = (0,1,0).
- The view window is from (-1,-1) to (1,1).

---

a) What is the vanishing point for the family of lines: $p_0 + t(1,1,1)$?

b) Describe the set of lines whose vanishing point is (*x*,*y*,2.

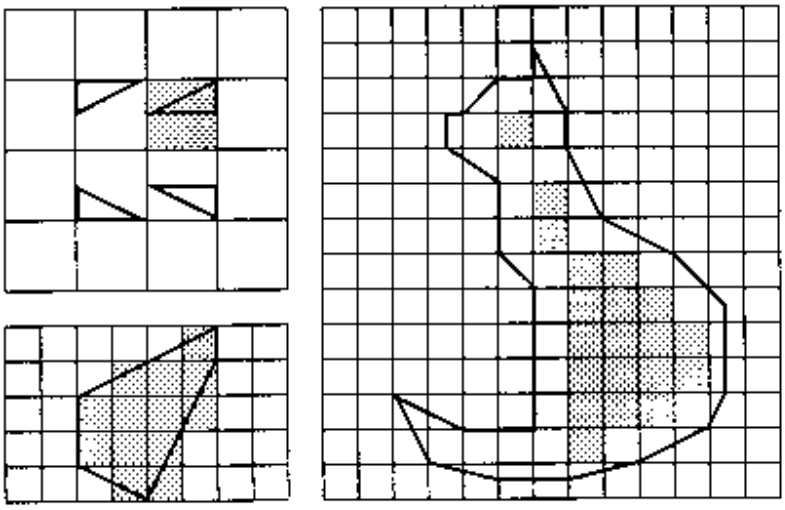
c) what is the 4x4 matrix that will compute this projection? (You may express this as a product of 4x4 matrices if you prefer.)

Problem #4, is nonexistant

Problem #5, Sample Point Algorithms for Scan Conversion (12 pts)

The images on the left of this figure illustrate a sample point scan conversion algorithm that uses a sample point somewhere on the edge of a pixel. The thick lines indicate polygon boundaries. The shaded regions indicate pixels that were determined to be interior based on the algorithm.

All of the vertices of polygons in this figure lie exactly on pixel grid lines or exactly halfway between pixel grid lines.

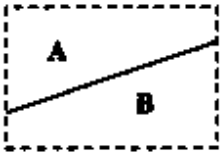


a) What is the sample point for this algorithm? Draw a pixel and indicate the sample point.

b) Scan convert the cat. Shade in all of the pixels that should be interior to the cat polygon on the right according to this scan conversion algorithm. (Some pixels have been pre-shaded for your convenience.)

Problem #6, Interior and Exterior of Polygons (16 pts)

a) In this figure, we see a window divided into two labeled regions by an edge of a polygon. We do not know the orientation of the edge between region A and B.



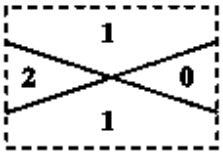
In each row of this table, one entry has been filled in to indicate that we know whether the given region is interior to the polygon by the given rule. Your task is to fill in the remaining table entries. (Each ros represents a different geometry for the polygon outside the window.) Each entry in the table should be one of these three values:

- **in** if the region is interior to the polygon according to the rule
- **out** if the region is exterior to the polygon according to the rule
- "**?**" if the answer cannot be determined from the given information

Region A			Region B		
Winding Rule in/out/?	Parity Rule in/out/?	Non exterior in/out/?	Winding Rule in/out/?	Parity Rule in/out/?	
		in			

		out		
in				
out				
	in			
	out			

In this figure, we see a window divided into four regions labeled with their winding numbers. Draw in the rest of the contour outside the window in a way consistent with the given winding numbers.



Here, the numbering on the test stops. For simplicity, I will just give questions without numbers the numbers they presumably should have.

Problem #7, Scan Converting Lines, Circles, and Ellipses and misc Hardware

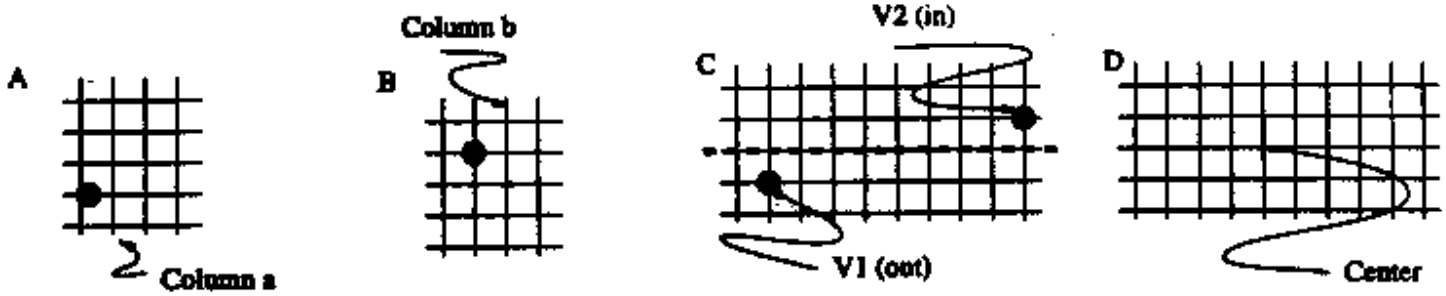


Figure 1: Line scan conversion question

Figure 1: Line scan conversion question

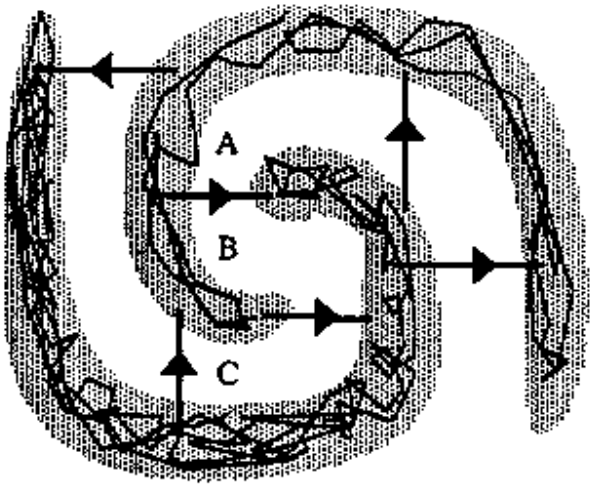
Scan Converting Lines, Circles, and Ellipses and misc Hardware

- Figure 1 (A) shows a pixel grid, with a pixel marked; this pixel lies on a line with slope 0 and 1. Which pixel in column (a) could lie on this line? (1)
- In a line scan conversion algorithm, how does one choose which of these pixels to mark? (2)
- While it is usual to look ahead one pixel in scan conversion, it is not possible to look ahead by two pixels; this exploits the fact that a pattern of three consecutive pixels does not occur in a line with slope between 0 and 1/2. Mark this pattern on the grid. (2)
- Name one advantage and one disadvantage of looking ahead two pixels. (2)

- Pigure 1 (B) shows a pixel grid, with a pixel marked. This pixel lies in the octant 0 <=$_{\text{less than or equal to}}$ x <= R and x < y of a circle of radius R; which pixels in column (b) could lie in this octant of this circle? (2)
- Figure 1 (C) shows a pixel grid, with two vertices marked; there is a clip boundary given in dashed lines. Mark the line endpoint obtained by clipping the line from v1 to v2 against this clip boundary, and assuming that the pixels lie on the intersection of coordinate lines, mark the pixels that would be set on by scan-converting the line from this endpoint to v2. (2)
- Recall that pixels that lie on a clip boundary are, by convention, inside the clip region. Compare the set of pixels obtained when the line is clipped, then scan-converted with those obtained when the line is scan-converted, and then the pixels outside the clip boundary thrown away -- is there a problem? How could this be avoided? (3)
- Mark all the pixels in circle of radius two pixels, whose center lies on the pixel shown in figure 1 (D). (3)
- Scan converting an axis aligned ellipse is less efficient that scan converting a circle; why? (2)
- Scan converting a general ellipse is more difficult that scan converting an axis aligned ellipse; why? (2)
- Commonly, 8 bit per pixel framebuffers have color tables, but 24 bit per pixel framebuffers do not. Why is this? (2)
- Honest Sid's Pre-owned Computers offers a workstation that has 1000 by 1000 pixels, and its monitor is refreshed 50 times a second. A salesman assures you that the display memory is conventional static Ram (access times for RAM range from 50-80 ns) and that pixels are read out of memory three pixels at a time by the display hardware. What is the disadvantage of buying this computer? (2)

Problem #8, Scan Conversion (25 points)

In the figure below, the gray areas represent a dense concentration of edges of a single, very degenerate polygon. You **cannot** tell how many edges there are within that region, nor the direction of them. There are six edges we **are** sure of -- these are labeled in bold with the directions specified. There are three topologically distinct regions we care about, A, B, and C. Each label in question 7.2 and 7.3 is worth 2 points for a correct answer, -1 point for a wrong answer (we'll mark "2/-1 points"), and 0 points if left blank. This is so that a random guessing strategy will, on average, yield 0 overall points.



(7.2) (2/-1 points) Using the **PARITY** rule, mark regions A, B, and C on the diagram with one of these three labels: **PI, PO, P?** (for "Parity IN", "Parity OUT", or "Parity Depends-what-is-in-the-gray-region")

(7.3) (2/-1 points) Using the **WINDING** rule, mark regions A, B, and C on the diagram with one of three labels: **WI, WO, W?** (for "Winding IN", Winding OUT", or "Winding Depends-what-is-in-the-gray-region")

(7.4) (5 points) There are 6 bold edges with directions specified. If we **remove** the directional label on **two** of them (but don't tell you which two), draw a circle around the label of the region(s) whose **WINDING** rule label from (7.3) **could change**.

Problem #9, Transformations (25 points)

(9.1) (5pts) Each of the following 4 x 4 matrices can be best described as performing one of the following transformations: translation, rotation, scaling, shearing, reflection, or perspective projection. The matrices are designed to operate on *column* vectors. For each matrix, write next to it which sort of transformation it is.

```
|   2   0   0   0   |
|   0  3/2  0   0   |
|   0   0   3   0   |
|   0   0   0   1   |

|  1  0  0  0  |
|  0  1  0  0  |
|  0  0  1  0  |
|  0  0 -1  1  |

| 1 0 1 0 |
| 0 1 2 0 |
| 0 0 1 0 |
| 0 0 0 1 |

rt(x) means square root of x

|   rt(2)/2      0       rt(2)/2      0       |
|      0         1          0         0       |
|  -rt(2)/2      0       rt(2)/2      0       |
|      0         0          0         1       |

|    1     0     0    rt(3)   |
|    0     1     1     1/2    |
|    0     0     1    rt(2)   |
|    0     0     0      1     |
```

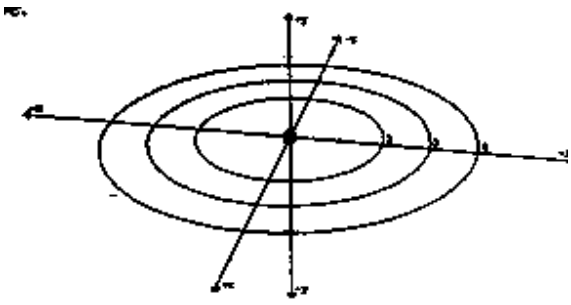
(9.2) (5 pts) Some of the transformations on #D points that are useful in graphics can be represented as simple 3 x 3 matrix transformations in R^3, that is, y=Ax where x and y are in R^3, and A is in R^{3x3}. Some other useful transformations are not linear operations in R^3, and thus cannot be represented by a 3x3 matrix. However, by using homogeneous coordinates, everal of these non-linear transformations can be represented by 4x4 matrices. For each type of transformation below, write whether it is **linear**, and can be represented as a 3x3 matrix, or **homogeneous** if it requires a 4 x 4 matrix and homogeneous coordinates.

- Translation
- Rotation about the z axis
- Perspective projection
- Scaling, leaving the origin fixed
- Rotation about an axis not passing through the origin

(9.3) (7 pts) Derive transformation that rotates points about the axis going from (1,0,0) to (2,1,0) by angle ALPHA. You may write your answer either as a 4x4 *column* vector homogeneous transformation matrix, or in terms of the transformations T(x,y,z), R_x(THETA), R_y(THETA), and R_z(THETA) as discussed in the book.

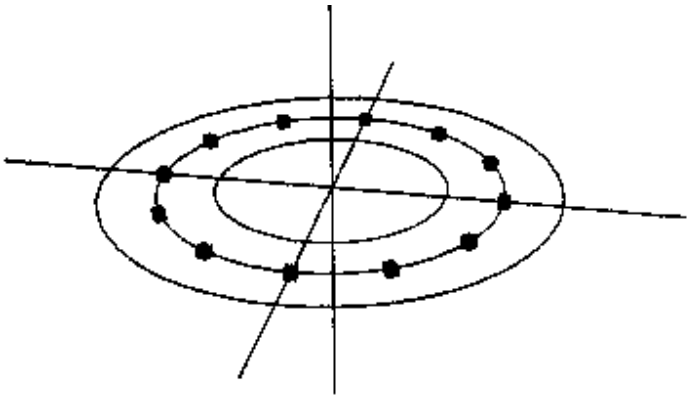
(9.4) (8 pts) You are writing a simple GL program to experiment with its viewing transformation commands, translate(x,y,z) and rotate(angle, axis). The program you begin with includes a C function, drawcube(), that draws a tiny cube, with sides of length 0.2, centered about the origin.

The program also draws the coordinate axes and three circles in the x-z plane, with radii 2, 3, and 4. The drawcube() function produces the following small cube:



```
drawcube();
```

The program also includes a funtion called drawcubecircle() that draws twelve cubes in a circle using the GL translate() and rotate() commands as shown below. Note that the rotate(int angle, char axis) command expectsits angle in **tenths** of degrees, so rotate(300,'y') rotates thirty degrees about the y axis.



```
void drawcubecircle() {
  int i;

  for (1=o;i<12;i++) {
    translate(3,0,0);
    drawcube();
    translate(-3,0,0);
    rotate(300,'y');
  }
}
```

Write a new function, called drawcubetorus(), that uses the same sorts of techniques as in drawcubecircle(), to produce the following figure. The inenr radius of the torus is 2, the outer radius is 4, and the torus consists of twelve rings of radius 1 with eighteen cubes each.

```
void drawcubetorus() {
  /* write your code here */

}
```

Problem #10, Projection and Perspective

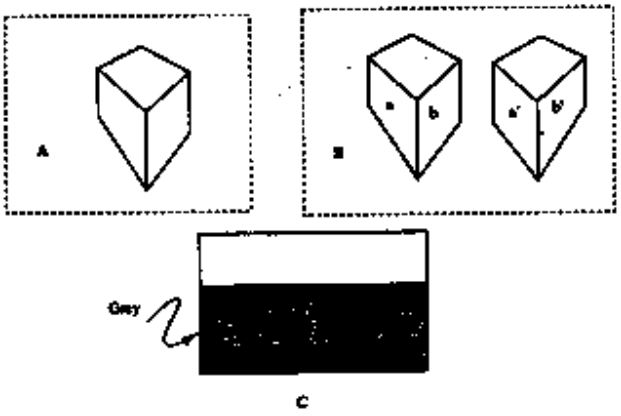


Figure 2: Perspective question

Projection and perspective

1. True or False: In an isomeric projection of a cube containing three vertical lines, the two lines at the base must make angles of 30 degrees with respect to the horizontal (1).
2. True or False: In a perspective projection, there are at most three vanishing points (1).
3. True or False: In an orthographic projection, a sphere can have an outline shaped like an ellipse (1).
4. True or False: In a perspective projection, a sphere can have an outline shaped like an allipse (1).
5. True or False: In a perspective projection, all the lines parallel in space to a given line l have the same vanishing point in the picture (1).
6. True or False: In a parallel projection, a cube can have three vanishing points (1).
7. Figure 2 (c) shows a projection of an infinite grey plane, after clipping to a viewing volume; is the projection parallel or perspective? (1)

- Figure 2 (a) shows a perspective projection of a cube sitting on an infinite ground plane. Construct the horizon, showing the construction lines you used. (4)

- Figure 2 (b) shows a perspective projection of two buildings. Both are cubes lying on a fixed ground plane, and faces a' and b and b' are parallel. Show two reasons why this could not ba a real perspective drawing of the geometry described, marking your construction lines as necessary. (4)

- Give criterion for the sets of parallel lines that, viewed in a perspective camera with focal point (center of projection) the origin and image plane z= -1, have no vanishing points (3).

- A camera has a focal point (center of projection) the origin, and viewing plane z= -1. Give the vanishing point on the camera plane of the lines given parametrically by (1,1,0) + t(0,0,1) and (-1,-1,0) + t(0,0,1). (3)

- For the same camera, describe the family of lines whose vanishing point is (2,2,-1) (i.e., the point with x-coordinate 2 and y-coordinate 2 on the image plane) (4).
