NAME_____

MIDTERM EXAM

15-462 Computer Graphics

Spring 1998

grade value: 13 points

SOLUTIONS

Q1:	/ 5%
Q2:	/15%
Q3:	/ 5%
Q4	/25%
Q5:	/25%
Q6	/25%
TOTAL:	/100%

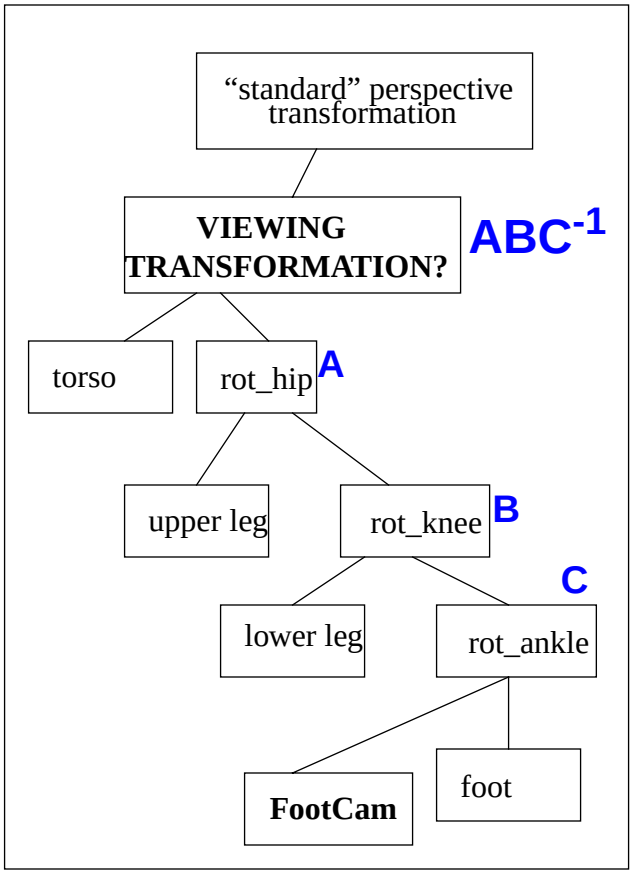
Please write your name on *every* page.

FINAL SCORE (= 13*TOTAL% / 100)

NAME_____

1. Remedial CoasterCam *(5%)*

OK. We're giving you another chance on this. In the hierarchy below, what should go in the "viewing transformation" box so that the scene ends up being drawn from the camera's viewpoint, given its place in the hiearchy?



NAME_____

2. Linearity, *again*(15%)

As long as we're giving second chances…

(a) Suppose **F**(**X**) is a linear function that maps vectors to vectors. What conditions must **F** satisfy in order to be linear?

F(k X) = k F(x), F(X+Y) = F(X) + F(Y), for all scalars k and vectors X,Y

(b) Is translation linear? Prove it.

Trans(X) = X + T, where T is the translation vector

Trans(kX) = kX + T, but k Trans(X) = kX + kT

or Trans(X+Y) = X + Y + T, but Trans(X) + Trans(Y) = X + Y + 2T

(c) Is scale linear? Prove it. (Just show for uniform scale, i.e. equal scaling in all directions.)

Scale(X) = sX, where s is the (scalar) amount to scale by.

Scale(kX) = k Scale(X) = ksX

Scale(X+Y) = Scale(X) + Scale(Y) ks(X+Y)

(d) What kind of transformation did we learn about, **other than translation**, that can be coerced into the standard matrix transformation framework through the use of homogoeneous transformations? Explain in a sentence or so (no equations needed) how the use of homogeneous transformations helps.

Perspective projection. Perspective projection entails a divide by z. We construct a matrix that puts z into the "w" coordinate of the output vector. Then when we divide by w (to convert from homogeneous to regular coordinates) we're dividing by z.

(e) Give an example of a simple transformation that **isn't** linear, not otherwise referred to on this page. Describe what it does, and give the three scalar equations for the transformed coordinates x', y', z' in terms of the original x, y, z.

x' = kxz

y' = kyz

z' = z

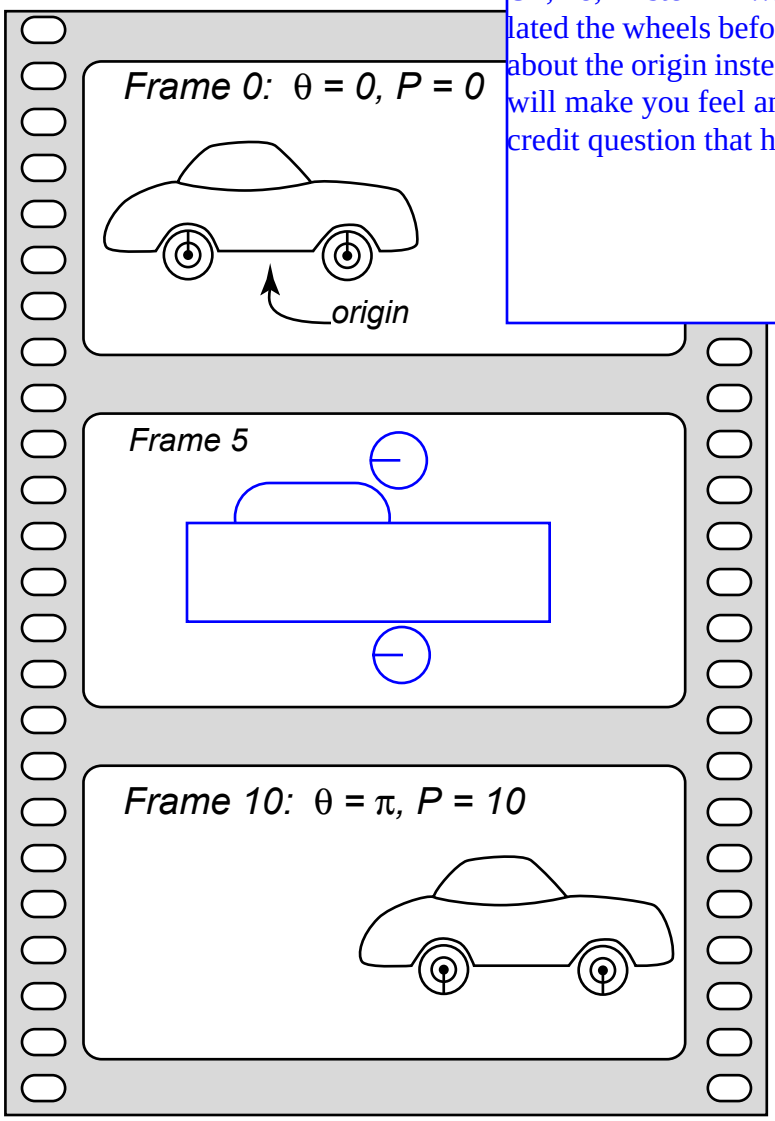
This does a "z taper", scaling in the xy plane as a function of position along the z axis. Turns a z-axis aligned cylinder into a cone.

NAME_____

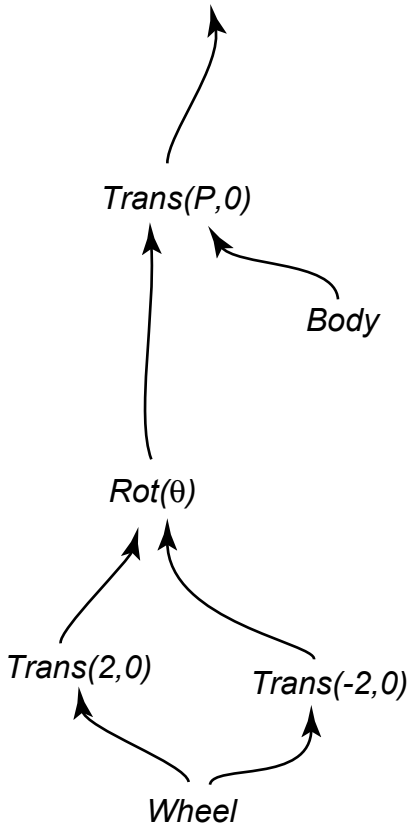
3. Fun with cars *(5%)*

In the car animation depicted below, Theta and P are varying linearly over time, so at Frame 5 theta = pi/2, and P = 5. Make a *rough* sketch of the car at Frame 5. The hierarchy graph for the car model is shown below on the right.

Notes: Theta is in radians. Positive angles denote counterclockwise rotation. Professional drivers; do not attempt these maneuvers.



Oh, no, Mister Bill!!! This hierarhcy is broken! We translated the wheels before we rotated them, so the rotation is about the origin instead of the wheel centers! Gotcha. If it will make you feel any better, think of this as an extra-credit question that hardly anybody got.



NAME_____

4. Near and Far *(25%)*

a) The class notes showed a homogeneous perspective transformation that (once we'd done the divide) mapped (x, y, z, 1) to (x/z, y/z, 0, 1). Say we wish to scale z such that instead of being always mapped to zero, it varies over the range [0, 1] (again, once we've divided), where 0 corresponds to the near plane (z = -n), and 1 to the far plane (z = -f).

i) Why might this be useful?

To expedite clipping at the near and far planes, and to scale z into a known range for a fixed-precision z-buffer.

ii) Write out a formula for z that would do the job. To be more precise, specify in terms of n and f values for *a*, *b*, *c* and *d* in in the matrix below such that the resulting matrix will map z as described.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & b & c & d \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

a = 0

b = 0

c = f/(f-n)

d = fn/(f-n)

Hint: Note that the matrix multiply produces x' = x, y' = y , z' = ax + by + cz + d, w' = z

Well, let's see. No reason z should depend on x and y, so a and b must be zero. And we're going to do a homogeneous divide-by-w after the matrix multiply, with w' = z, which means that z'(z) = c + d/z. We want

(1) z'(-n) = c - d/n = 0 and

(2) z'(-f) = c - d/f = 1.

So we have two linear equations to solve for c and d. Hmmm.

Subtracting (1) from (2) gives

d/n - d/f = 1, so d = 1/(1/n - 1/f) = fn/(f - n)

NAME_____

5. Clipping *(25%)*

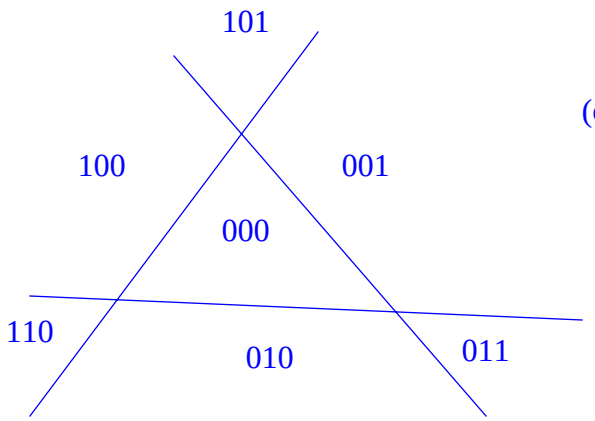
In the Cohen-Sutherland edge-clipping algorithm, we can trivially accept an edge if both its endpoints have an outcode of 0, where the outcode of a point is classified as in the diagram below. It is obvious why this works: it means both endpoints lie inside the clipping region.

1001	1000	1010
0001	0000	0010
0101	0100	0110

f) We can also trivially reject an edge if the bitwise and of its endpoint outcodes is non-zero. Why is this?

Each outcode bit encodes the results of a halfplane test, corresponding to one of the edges of the clipping window. A non-zero bitwise and means that both endpoints of the line being clipped failed *the same* half-plane test. If both endpoints are on the wrong side of the same clipping edge, the whole line must be.

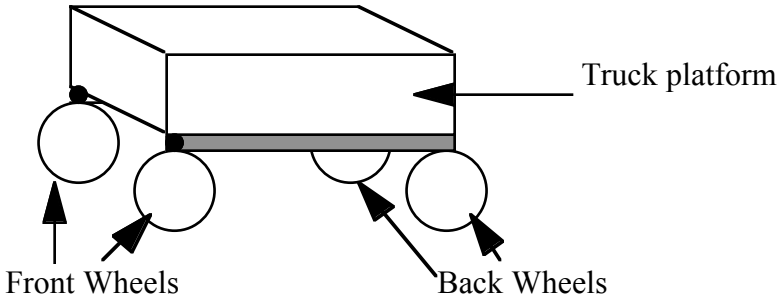
g) The Cohen-Sutherland algorithm can be extended to clip to any convex polygon, not just rectangles. Draw the equivalent of the above diagram for a C-S clipping routine that works on triangles.



(or equivalent)

NAME_____

6. Keep on Truckin' *(25%)*



A hierarchy doesn't have to be a tree. It can be a DAG in which some nodes have more than one parent (such as the wheel in Problem 3.) When you recursively descend the hierarchy in the usual way, such multi-parented objects are hit more than once, so you get multiple copies of the object, differently transformed. In this problem, we'll try to exploit this capability to build an efficient hierarchy that models a really stupid truck.

The diagram above shows a simple truck with four wheels. This particular truck has four-wheel steering, so that when the steering wheel is rotated, all four wheels turn together to the left or right, by the same amount. In addition, of course, the whole truck can translate and rotate as a unit.

Draw a hierarchy that captures this behavior **using as few nodes as possible.** You only need to represent the four wheels (not the truck body), and the transformations that apply to them. Your hierarchy may contain these three node types:

1. Wheel primitive
2. Translate
3. Rotate

Label each node by type, and identify its role in the model (e.g. *Translate: move the left front wheel to Saturn*). You do **not** need to give numerical values for transformation parameters. You can make any assumptions you want about where the origin is.

[HINT: In case we didn't make it obvious, the hierarchy with the fewest nodes is NOT a tree]