Computer Graphics	Prof. Brian Curless
CSE 457	Autumn 2000

Midterm Study Guide

Hierarchical Modeling, Projections, and Hidden Surfaces

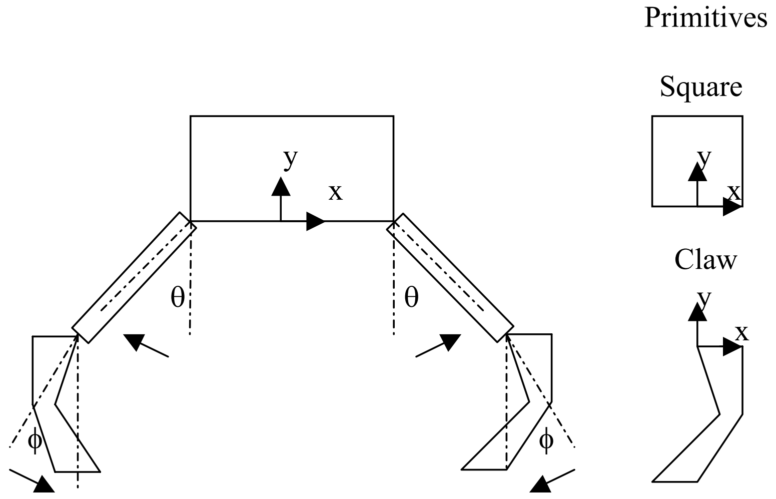
The midterm will be on Wednesday, November 8. It will cover all material up through hidden surface algorithms. The mideterm will last 50 minutes and will be close book. These problems are intended to get you to think in some depth about the material that was not covered by homework #1.

Problem 1

You are modeling a hanging pincer as shown below. There are two primitives available to you, **Square** and **Claw**. The local coordinates for each primitive are shown. The transformations available are

- $R(\theta)$ – rotate by θ degrees (counter-clockwise)
- $T(t_x, t_y)$ – translate by t_x, t_y
- $S(s_x, s_y)$ – scale by s_x, s_y
- R_y – reflect through the y axis

The arrows in the figure indicate the direction of rotation that θ and ϕ refer to for each pincer. The base of the figure is twice as wide (along the x axis) as the square. Each finger is 1/3 as wide as the square, and 3/2 as long.



a. Give names to the different pieces of the pincer. For each piece, define a primitive to draw it if necessary, using, for example, a function definition with OpenGL code. Now draw a directed acyclic graph (DAG) to specify the pincer. For each of the edges in your DAG, write the expression for that transformation using only the symbols given on the previous page. This transformation should be the transformation required to position the piece that is the next node. Each node in the DAG should be one of the primitives given to you, or one of the primitives you have defined. Leave θ and ϕ as symbols, these are the parameters you intend to animate. Assume **Square** is one unit in size.

b. What is the entire sequence of transformations applied to the claw drawn on the far left side of the figure?

Problem 2

a. Perspective Projections

True or False

1. Size varies inversely with distance

2. Distance and angles are preserved

3. Parallel lines do not remain parallel

4. Perspective projections make z-buffers more imprecise

How many different vanishing points can a perspective drawing have?

How many different principal vanishing points can a perspective drawing have?

b. Parallel Projections

True or False

1. Parallel projections are more realistic looking than perspective projections

2. Parallel projections are good for exact measurements

3. Parallel lines do not remain parallel

4. Angles (in general) are not preserved

5. Lengths vary with distance to the eye

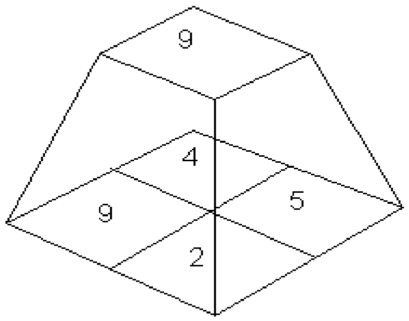
Parallel projections can be further broken down into two types: “orthographic projections”, where the direction of projection is perpendicular to the projection plane, and “oblique projections”, where the direction of projection is not perpendicular to the projection plane.

Two common types of oblique projections are the “cavalier projection” and the “cabinet projection”. In a cavalier projection the foreshortening factors for all three principal directions are equal, whereas in a cabinet projection the edges perpendicular to the plane of projection are foreshortened by one half.

Suppose you wanted to use an oblique projection that foreshortened the edges perpendicular to a plane of projection by one third instead of one half. What angle between the direction of projection and the projection plane is required?

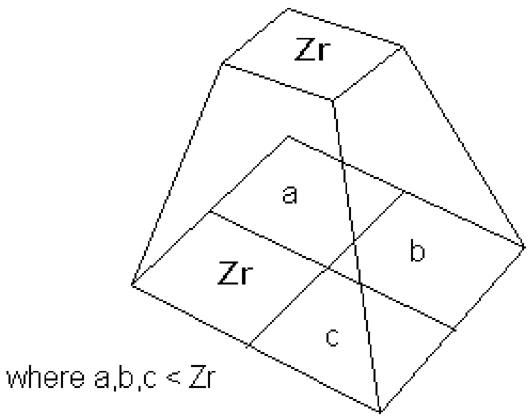
Problem 3

The Z-buffer algorithm can be improved by using an image space “Z-pyramid.” The basic idea of the Z-pyramid is to use the original Z-buffer as the finest level in the pyramid, and then combine four Z-values at each level into one Z-value at the next coarser level by choosing the farthest (largest) Z from the observer. Every entry in the pyramid therefore represents the farthest (largest) Z for a square area of the Z-buffer. A Z-pyramid for a single 2x2 image is shown below:



a) (3 Points) At the coarsest level of the pyramid there is just a single Z value. What does that Z value represent?

Suppose we wish to test the visibility of a polygon **P**. Let **Zp** be the nearest (smallest) Z value of polygon **P**. Let **R** be the smallest region in the Z-pyramid that completely covers polygon **P**, and let **Zr** be the Z value that is associated with region **R** in the Z-pyramid.



Problem 3 - continued.

b) (5 Points) What can we conclude if **Zr** < **Zp**?

c) (5 Points) What can we conclude if **Zp** < **Zr**?

If the visibility test is inconclusive, then the algorithm applies the same test recursively: it goes to the next finer level of the pyramid, where the region **R** is divided into four quadrants, and attempts to prove that polygon **P** is hidden in each of the quadrants **R** of that **P** intersects. Since it is expensive to compute the closest Z value of **P** within each quadrant, the algorithm just uses the same **Zp** (the nearest Z of the *entire* polygon) in making the comparison in every quadrant. If at the bottom of the pyramid the test is still inconclusive, the algorithm resorts to ordinary Z-buffered scan conversion to resolve visibility.

d) (7 Points) Suppose that, instead of using the above algorithm, we decided to go to the expense of computing the closest Z value of **P** within each quadrant. Would it then be possible to always make a definitive conclusion about the visibility **P** of within each pixel, without resorting to scan conversion? Why or why not?

Another type of coherence that one might like to exploit in a hidden surface algorithm is *temporal coherence* - the fact that the same polygons are likely to remain visible from one frame of an animation to the next.

5e) Suppose that we kept track of the polygons that were at least partially visible for some frame *t*. How might we use this information in frame *t* + 1 to exploit temporal coherence with a Z-pyramid? How much better is using a Z-pyramid in this case than using a simple Z-buffer?

Problem 4

Answer the following questions regarding BSP trees

a. Is it sort-first, sort-last, or both?

b. Is it image-precision, object-precision, or both?

c. Is it image-order, object-order, or both?

d. Do all polygons need to be rendered?

e. Is it an online algorithm?

f. Can it handle transparency?

g. Can it handle refraction?

h. Can shading be done efficiently?

i. What are the memory requirements proportional to?

j. What types of 3d graphics programs are well suited to BSP trees?

Answers

1a.

```
Base()
{
    glPushMatrix();
    glScalef(2,1,0);
    Square();
    glPopMatrix();
}

Finger()
{
    glPushMatrix();
    glScalef(1/3,3/2,0);
    Square();
    glPopMatrix();
}
```

$$\begin{tikzcd}
& \boxed{\text{Base}} \arrow[dl, "M_{FL} = R_y M_{FR}"'] \arrow[dr, "M_{FR} = T(1/2,0)\ R(\theta)\ T(0,-3/2)"] & \\
\boxed{\text{Left Finger}} \arrow[d, "M_{CL} = R(-\phi)"'] & & \boxed{\text{Right Finger}} \arrow[d, "M_{CR} = R(-\phi)"] \\
\boxed{\text{Left Claw}} & & \boxed{\text{Right Claw}}
\end{tikzcd}$$

b. $R_y\ T(1/2,0)\ R(\theta)\ T(0,-3/2)\ R(-\phi)$

2a.

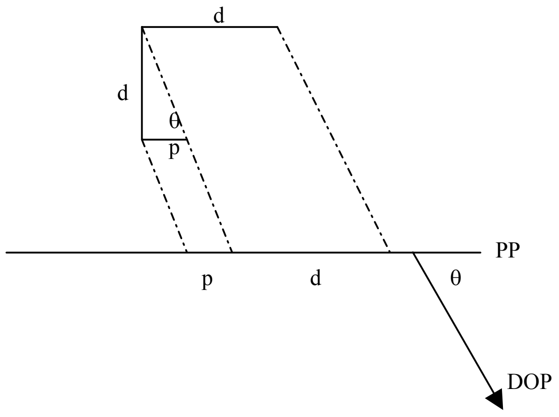
1. True – as distance increases size decreases.
2. False – parallel lines go to a vanishing point and distances are compressed.
3. True – see above.
4. True – the compression of distant z values has an effect on depth ordering and intersections

A perspective drawing can have infinitely many vanishing points.

However, it will only have 3 principal vanishing points – along the 3 axes.

b.

1. False – we are accustomed to seeing things in a perspective projection
2. True – distances are not compressed and angles remain the same.
3. False – parallel lines remain parallel.
4. False – angles are preserved.
5. False – distances are preserved.



Here, we see a parallel projection of two lines, each of length d. The DOP makes an angle θ with the projection plane. One line is parallel to the PP, so the length of its projection is d. The other line is perpendicular to the PP, so it gets foreshortened to length p. Lengths p and d are related by d/p = tan θ. For this problem, we want p to be 2/3 of d, so:

$$d/p = d/(2/3d) = 3/2 = \tan\theta$$
$$\theta = 56.3°$$

3.

a. The z-value of the farthest pixel.
b. Polygon P is completely obscured within region R.
c. We can't conclude anything – we must subdivide region R.
d. Yes. Eventually the subdivision would reach a single pixel region, so we could determine for certain that P is hidden, or not, within the pixel.
e. Suppose that we mark all the polygons that are visible in frame t, and insert those into the z-pyramid first for frame t+1. Since these will tend to be low z-values in the new frame the hidden geometry will tend to be discarded sooner than it would otherwise. In this case, the z-pyramid is a big win over the z-buffer.

4.

a. Sort first. Objects are sorted when the tree is built in the preprocessing step.
b. Object precision in that objects are split and the tree constructed at the object level, but a painter's algorithm really determines visibility at the pixel level so you can argue it is both.
c. Object order. The algorithm renders objects one after another. Visibility is not determined pixel by pixel.
d. Yes, in the basic algorithm objects are drawn from back to front, we can't skip any polygons because we don't know if they will be occluded until everything is drawn.
e. No, all the object data must be available for the tree to be constructed and rendering to begin.
f. Yes, it is easy to handle transparency with a painter's algorithm. Transparent objects just alpha blend with objects behind them which have already been drawn.
g. No, refractive objects don't simply blend their color with the color further along the line of sight. They may potentially let you see around objects. Raytracing is the simplest way to handle refraction.
h. No, because all polygons must be rendered. (On the other hand, because whole polygons are rendered you can make use of efficient incremental rendering algorithms.
i. N, where N is the number of polygons, in the best case and 2^N-1 in the worst case (every polygon split at every step in construction.)
j. Walkthroughs where the viewpoint is changing but the scene is static – architectural visualization and first-person games.