CPSC 314, Midterm Exam 2

21 Mar 2005

Closed book, one single-sided sheet of handwritten notes allowed. Answer the questions in the space provided. Do not open until told to do so. Place your ID face up in front of you.

Name: _____

Student Number: _____

Question	Points Earned	Points Possible
1		4
2		6
3		6
4		4
5		4
6		6
7		2
8		4
9		10
10		10
11		10
12		10
13		12
14		12
Total		100

1. (4 pts) Multiple choice: for which of the following objects would backface culling gain efficiency while still yielding correct results?

   - glass balls
   - mirrored balls [*]
   - pieces of paper
   - opaque cubes [*]

   Both the mirrored balls and the opaque cubes are closed surfaces, so it's safe to cull the backfacing polygons. Glass balls are closed but transparent, so the backfacing polygons might contribute to the final image. Pieces of paper are not closed manifold surfaces.

2. (6 pts) Multiple choice: which techniques are view-dependent?

   - ray tracing [*]
   - backface culling [*]
   - BSP tree traversal [*]
   - radiosity
   - z-buffer [*]
   - BSP tree creation

   Radiosity and BSP tree creation are view-independent.

3. (6 pts) True/false: mark the following statements as true or false

   - the human eye is good at absolute color judgements [F]
   - a pure red light shining on a pure blue object will look purple [F]
   - metamerism occurs in the HSV and RGB but not the YIQ color spaces [F]
   - ray tracing handles shadows easily [T]
   - radiosity handles marble and milk easily [F]

   The eye is good at relative judgements and bad at absolute judgements. A red light on a blue object yields black, because color is multiplied component-wise: (1,0,0)*(0,0,1) = (1*0,0*0,0*1) = (0,0,0). Metamerism is the phenomenon that very different spectra look like the same color to people because they create the same output from the three kinds of sensors in our eyes (cones): it holds for all possible color spaces. It is very easy to incorporate shadows into ray tracing by checking whether the ray between the point on an object and the light intersects another object; in contrast, is it difficult to do so in the hardware rendering pipeline. Radiosity does not handle subsurface scattering effects by default; marble and milk do not look realistic without this.

4. (4 pts) Give two examples of problems that can occur with motion capture data.

   foot skate: feet don't stay in solid contact with floor surface, either hovering above it or penetrating the surface

   retargeting motion: difficult to adapt motion to figure with different shape than original actor

   noisy data: targets can be occluded from all cameras

5. (4 pts) Give two examples of problems that can occur with flood fill scan conversion.

   inefficient in time because pixels can be visited several times (up to 4x)

   inefficient in memory because need storage to keep track of whether pixel has been visited.

   Answers that infinite loops are a danger are incorrect. The fact that any algorithm could have a bug in the implementation is not an interesting observation. A previous midterm asked a question about finding the bug in a proposed flood-fill algorithm, perhaps this caused some confusion.

   Some people answered with problems that are generic to scan conversion, not specific to flood filling; for instance, that small polygons may be missed completely, or that the color of shared edges can depend on the order of operations. Although it's not what I had in mind, it was counted as correct.

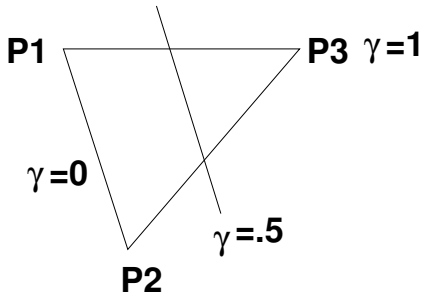
6. (6 pts) In Situation A, your scene has 100 polygons and your framebuffer size is 1920x1200 pixels. In Situation B, your scene has 100,000 polygons and your framebuffer is 320x240 pixels. Briefly discuss which hidden surface removal methods would be efficient for each scene, justifying your decision in terms of the distinction between image space and object space.

   For A, there are a lot of pixels and few objects. For B, there are relatively few pixels and a lot of objects. For A, an object-space algorithm like BSP trees would be a good choice for A since marching through the list of objects will be feasible because there are so few, and a Z-buffer for a display that big would require a great deal of memory. Although Warnock's algorithm operates in image space, it also might be a good choice because the scene probably has low depth complexity so the one-pixel expensive case would not occur often. For B, an image-space algorithm like Z-buffering would make sense because the list of objects is quite long. Resolving visibility at the object level would be expensive, and moreover data structures like the BSP tree would need to be recomputed if objects move. Warnock's algorithm would be a poor choice because the scene has high depth complexity.

   Some people had different arguments for and against these methods. For instance, that memory is so cheap that Z-buffering is OK to use in case A and the low depth complexity of the scene makes it a competitive choice.

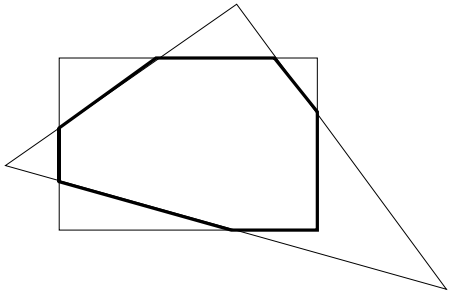
   As long as the answers demonstrated a fundamental understanding of the method characteristics and the concepts of object and image space, credit was given. Reasons for losing points included answers that showed confusion on the concepts, not discussing object and image space, or not discussing specific hidden surface removal algorithms.

7. (2 pts) Sketch the line $\gamma = .5$ in the triangle below, where α, β, and γ are barycentric coordinates defined according to $P = \alpha P1 + \beta P2 + \gamma P3$.

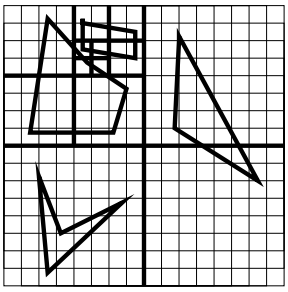


8. (4 pts) What is the maximum number of sides that a triangle might have when clipped to a rectangular viewport? Sketch an example.

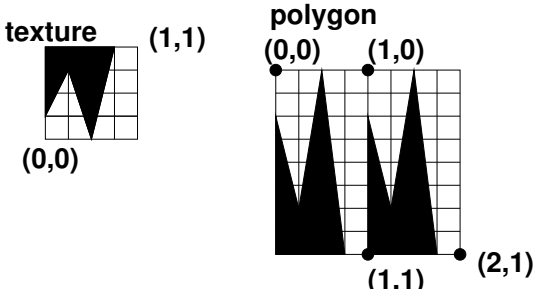
   7



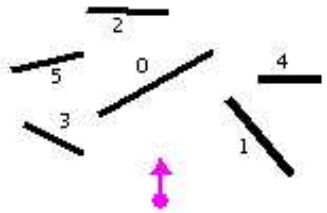
9. (10 pts) Sketch the viewport subdivisions that would be required for Warnock's algorithm in the picture below, where the background grid shows the size of a single pixel.

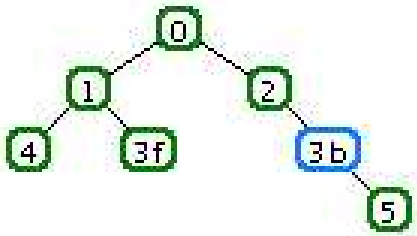


10. (10 pts) Sketch the texture on the polygon with the given (s,t) texture coordinates:



11. (10 pts) Construct the BSP tree for the line segments shown below, adding the objects to your tree in numerical order. Use the convention that the right child of a node is the labelled side, and the left child is the unlabelled child.





12. (10 pts) Show the traversal order of the BSP tree given the eye point shown with a dot and arrow in the picture above. Show intermediate stages of computation.

    (0L) = subtree with root 2.

    (0U) = subtree with root 1

    Recursively divide the root into the subtree that is furthest from the viewpoint, the root itself (the polygon to draw) and the subtree that is closest to the viewpoint. Continue until you reach all leaf nodes of the tree.

    (0L) 0 **(0U)**

    (2U) 2 **(2L)** 0 **(1U)** 1 **(1L)**

    2 (**3bL**) 3b (**3bU**) 0 4 1 3f

    2 5 3b 0 4 1 3f

    Alternative, more detailed, method:

    #L = labelled side (right child) #U = unlabelled side (left child)

    0L 0 OU: start at root node, labelled side is far for 0
    2 0 0U: expand 0L
    2U 2 2L 0 0U: unlabelled side is far for 2
    2 2L 0 0U: 2U→null (nothing on the far unlabelled side of 2)
    (2) 3b 0 0U: draw segment at 2, expand 2L
    (2) 3bL 3b 3bU 0 0U: labelled side is far for 3b
    (2) 5 3b 3bU 0 0U: expand labelled side of 3b
    (2) 5U 5 5L 3b 3bU 0 0U: unlabelled side is far for 5
    (2) 5 5L 3b 3bU 0 0U: 5U→null (nothing on the far unlabelled side of 5)
    (2 5) 3b 3bU 0 0U: draw 5, 5L→null (nothing on near labelled side of 5)
    (2 5 3b) 0 0U: draw 3b, 3bU→null (nothing on near unlabelled side of 3b)
    (2 5 3b 0) 1: draw 0, expand 0U
    (2 5 3b 0) 1U 1 1L: unlabelled side is far for 1
    (2 5 3b 0) 4 1 1L: expand unlabelled far side of 1
    (2 5 3b 0) 4U 4 4L 1 1L: labelled side is far for 4
    (2 5 3b 0 4) 4L 1 1L: 4U→null, draw 4
    (2 5 3b 0 4 1) 1L: 4L→null, draw 1
    (2 5 3b 0 4 1) 3f: expand 1L
    (2 5 3b 0 4 1) 3fL 3f 3fU: labelled side far for 3f
    (2 5 3b 0 4 1 3f) 3fU: 3fL→null, draw 3f
    (2 5 3b 0 4 1 3f): 3fU→null

13. (12 pts) Write an algorithm (pseudo-code) for clipping a line L with endpoints P_1 and P_2 against a triangle V with vertices V_1, V_2, and V_3. Follow the framework of the Cohen-Sutherland algorithm for clipping a line against a window.

Note: The algorithm is almost identical to the one shown in class. The only differences are the number of edges and the table, and how we check if a point is inside/outside w.r.t a given edge.
Assume the triangle edges are given in CCW order $V_1V_2 \;\; V_2V_3 \;\; V_3V_1$

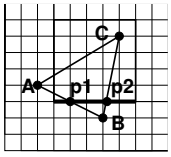
Calculate a 3 digit binary code for each of the line end points $d_j = d_j^1 d_j^2 d_j^2$, such that $d_j^i = 0$ if P_j is to the left of line V_iV_{i+1} and $d_i^j = 1$ if P_j is to the right of line V_iV_{i+1} for $j = 1, 2 \;\; i = 1, 2, 3$.
Use the implicit line equation $F_{V_1V_2}$ to determine where the end points are.
If $F_{V_1V_2}$ at p and $F_{V_1V_2}$ at V_3 share the same sign, then both points are on the same side of the line (in this case the inside).

If $(d_1 \;\; OR \;\; d_2 = 0)$ then both end points are in the triangle, thus the line is inside.
If $(d_1 \;\; AND \;\; d_2 \neq 0)$ then both end points are to the right of the same edge, thus the line does not intersect the triangle, and the whole line is clipped.

Otherwise, find $\min_{i,j} \;\; s.t. \;\; d_{j_i} \neq 0$ and find $\bar{P}_j =$ the intersection between P_1P_2 and the edge V_iV_{i+1}, recursively call the clipping algorithm with the new line $\bar{P}_jP_{j+1}$

```
edge[1] = v1,v2
edge[2] = v2,v3
edge[3] = v3,v1

clip(p,q) {
        oc_p = getOutCode(p);
        oc_q = getOutCode(q);
        if (oc_p || oc_q == 0)
                return (p,q); //trival - line inside

        if (oc_p && oc_q != 0)
                return; //trivial - line outside

        if (oc_p != 0) {
                index = firstNonZeroBit(oc_p); //returns index of non zero bit
                new_p = intersect(p,q,edge[index]); //intersect two lines, returns point
                return clip(new_p, q);
        } else { //oc_q != 0
                index = firstNonZeroBit(oc_q);
                new_q = intersect(p,q,edge[index]);
                return clip(p, new_q);
        }

}

getOutCode(p) {
        assert F_v1_v2(v3) < 0; //make sure that line is oriented s.t. negative side is i
        assert F_v2_v3(v1) < 0;
        assert F_v3_v1(v2) < 0;

        if F_v1_v2(p) < 0 d1 = 0 else d1 = 1;
        if F_v2_v3(p) < 0 d2 = 0 else d2 = 1;
        if F_v3_v1(p) < 0 d3 = 0 else d3 = 1;

        return (d1,d2,d3);
}
```

14. (12 pts) Give the vertex list for the triangle with endpoints A=(-1,1), B=(3,-1), C=(4,4) after clipping against the line (0,0), (5,0) while using the Sutherland-Hodgeman algorithm. Clarification: the rectangular viewport stretches from (0,0) to (5,5). Compute the coordinates of any new points that you add to the list. Show your work.



Input vertex list: A=(-1,1), B=(3,-1), C=(4,4).

Clipping against line from (0,0) to (5,0) (bottom edge of viewport)

First consider segment AB. Point A is above the clipping line (inside), point B is below the clipping line (outside). Need to compute new intersection point p1 and add it to the output vertex list.

$x_{int} = x_1 + (y_bot - y_1)/m$
$m = (y_2 - y_1)/(x_2 - x_1) = (-1 - 1)/(3 - (-1)) = -2/4 = -1/2$
$x_{int} = -1 + (0 - 1)/m = -1 - 1/m = -1 - (-2/1)) = 1$
$p1 = (1, 0)$

Then consider segment BC. Point B is below the clipping line (outside). Point C is above the clipping line (inside). Need to compute new intersection point p2 and add both p2 and C to the output vertex list.

$x_{int} = x_1 + (y_bot - y_1)/m$
$m = (y_2 - y_1)/(x_2 - x_1) = (4 - (-1))/(4 - 3) = 5/1 = 5$
$x_{int} = 3 + (0 - (-1))/m = 3 + 1/m = 3 + 1/5 = 3.2$
$p2 = (3.2, 0)$

Then consider segment CA. Point C is above the clipping line (inside). Point A is above the clipping line (inside). Add A to the output vertex list.

Final vertex list: [p1, p2, C, A]