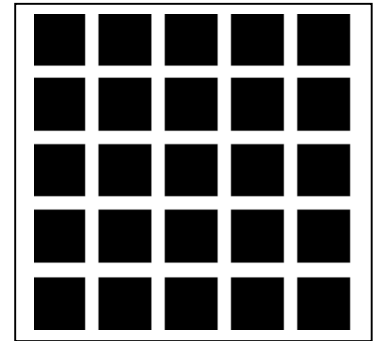


## 3451 final, summer 2006

**A) Perception:** Explain in details why this image appears to be blinking (support your explanation with facts about the anatomy of the human vision system). Make sure that you have a logically correct explanation. Be specific: Where, when, and why do spots appear? When and why do they disappear?



*This is the Hermann grid illusion. In a portion of the image seen by a ganglion cell in the peripheral part of the retina, the streets appear brighter than the crossings by contrast to their neighborhood. The inhibitory neighborhood of a street point (for which 6 out of 8 squares are dark) is darker than the inhibitory neighborhood of a crossing (for which only 4 out of 8 squares are dark). The effect vanishes when we look directly at an area, because our fovea has higher resolution and hence a smaller inhibitory neighborhood which is white in both cases.*

**B) Geometry:** Set B is defined as  $\{P: (P.z > 0) \ \&\& \ ((P.x)^2 + (P.y)^2 < r^2)\}$  in a local coordinate system  $C_1 = [O_1, I_1, J_1, K_1]$ . Point Q is defined by its coordinates  $(Q.x, Q.y, Q.z)$  in a second local coordinate system  $C_2 = [O_2, I_2, J_2, K_2]$ . How would you test whether  $Q \in B$ ? Provide a brief outline of the overall strategy and then the detailed geometric constructions or computations for each step, indicating exactly what is computed.

*Express Q in the global system, compute its coordinates in  $C_1$ , plug them in the inequality defining B.*

*$P = O_2 + Q.x I_2 + Q.y J_2 + Q.z K_2$ ; // the coordinates of P are the global coordinates of Q*

*$x = O_1 P \cdot I_1$ ;  $y = O_1 P \cdot J_1$ ;  $z = O_1 P \cdot K_1$ ; //  $(x, y, z)$  are the local coordinates of Q in  $C_1$*

*return  $(z > 0) \ \&\& \ (x^2 + y^2 < r^2)$ ; // plugging them in the definition of B*

**C) Topology:** Consider 3 solids: A, B, and C, that are each closed, finite, and connected. They form a valid electronic assembly if the following three conditions are satisfied: (1) A, B, and C do not interfere, (2) A and C do not touch, (3) B touches both A and C. Use logic, set, and topological operators to express these 3 conditions.

*(1):  $(A.i \cap B.i = \emptyset) \ \&\& \ (B.i \cap C.i = \emptyset) \ \&\& \ (C.i \cap A.i = \emptyset)$*

*(2):  $(A \cap C = \emptyset)$*

*(3):  $(A \cap B \neq \emptyset) \ \&\& \ (B \cap C \neq \emptyset)$*

**D) Curves:** We want a multi-resolution format for rendering closed-loop curves in the plane. The curve should be a coarse polyloop when it is small and should be iteratively refined as we zoom in. We want to explore the visual impact of the popping effect when increasing the resolution. (1) Provide the details of the cubic B-spline subdivision scheme that would refine a polyloop A into a smoother polyloop B. (2) Suggest a geometric error for measuring the popping effect and provide a precise definition for it. Explain why such a measure makes sense. (3) Explain how to compute a simple conservative upper bound on this error and explain why it works. (4) Suggest a subdivision scheme that would reduce the popping effect, provide the subdivision steps and explain why it will have less of a popping effect than the cubic B-spline.

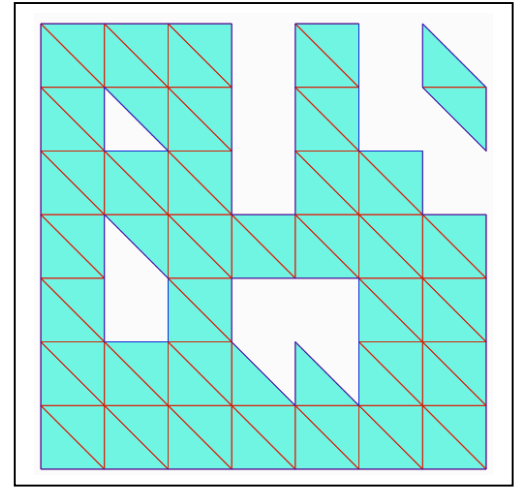
*(1) First, refine the polyloop by introducing a new vertex in the middle of each edge. Then, for each vertex v, compute the Laplace vector  $L(v)$  between v and the average of its 2 neighbors. Finally, move each vertex v to  $v + L(v)/2$ . The last two steps amount to a tuck(1/2) of the old vertices.*

*(2) Use the Hausdorff distance,  $H(A, B)$  between a curve A at one resolution and its refined version B obtained by subdivision.  $H(A, B)$  measures the discrepancy between A and B, which is the maximum distance between a point on any one of these curves and the other curve. It makes sense because it is 0 when the two curves are identical and measures the deviation of B in the normal direction from A (which is what would be perceived as popping).*

*(3)  $H(A, B)$  will not exceed  $M/2$ , where  $M = \max(L(v).norm())$  for all vertices v of A, because no point of A has traveled more than  $M/2$ . Indeed, the refine step does not change the shape of the polyloop. Then, the midpoints of the edges of A remain fixed and the old vertices of A move, each by no more than  $M/2$ . A point between a mid-edge point m and a vertex v would move by a fraction of  $L(v)/2$ .*

*(4) The cubic B-spline is actually the scheme  $J(0)$  in the family of Jarek subdivision schemes:  $J(s) = \{\text{refine}; \text{tuck}(1/2); \text{tuck}(-s/8)\}$ . We can lower  $H(A, B)$  if instead of  $J(0)$ , we use  $J(s)$  with  $0 < s < 8$ , because the tuck by a negative value of  $-s/8$  will bring the refined curve B closer to A. In fact,  $J(5)$  seems to yield the lowest error.*

E) **Border Loops:** Assume that you have a corner table representation of an oriented triangle mesh  $M$  with  $t$  triangles. The mesh needs not be connected. Assume that  $(c.o == -1)$  for all corners  $c$  that have no opposite. Use the corner operators to formulate an algorithm that will count the number of border loops in  $M$ . (For example, the mesh below has 5 loops.) If you wish, for simplicity, you may assume that the border is manifold. Provide a complete and detailed algorithm in Processing. Include comments to explain what the variables mean and what each statement does.



```
int countloops() {
  boolean M[] = new boolean [3*nt];           // flag visited corners
  for (int c=0; c<3*nt; c++) M[c]=false;      // all corners not visited
  int loops=0;                                 // counter of loops
  for (int b=0; b<3*nt; b++) {               // look for not marked border corners
    int c=b;                                  // use c to trace loop
    if ( (!M[c]) && (o(c) == -1) ) {          // found not marked and border
      loops++;                                // new loop
      while (!M[c]) {                         // while not finished tracing
        M[c]=true;                            // mark it
        c=n(c); while ( o(c) != -1 ) c=n(o(c)); // start with the next corner keep turning until a border corner is reached
      }; };
    return(loops); }
}
```

F) **Solids:** You are given a corner table representation of a manifold triangle mesh that has no self-intersections, but may have more than one component (**shell**). Provide a high-level description and pseudo-code of an algorithm that will compute how many **connected solids** are bounded by the mesh.

**We first identify the connected components of the mesh and label each triangle with its component ID.**

```
for (int t=0; t<nt; t++) M[t]=0;
int component = 0; for (int t=0; t<nt; t++) if (!M[t]) visitComponent(3*t,component++);
void visitComponent(int c, int n) { M[t(c)]=true;
  if ((o(c)!=-1)&&(!M[t(o(c))])) visitComponent(c.o,n);
  c=n(c); if ((o(c)!=-1)&&(!M[t(o(c))])) visitComponent(c.o,n);
  c=n(c); if ((o(c)!=-1)&&(!M[t(o(c))])) visitComponent(c.o,n); }
```

**Then, we make all components active and count outer shells as follows:**

```
Int solids=0; // will contain the final number of solids
Int processed=0; // keeps track of the number of deactivated components
while (processed!=component) { // some active component remains
  for each active component C { // deactivate outer shells of solids
    if (C is not contained in any active component) {solids++; processed++; deactivate C;};
  }
  for each active component C { // deactivate shells around holes
    if (C is not contained in any active component) { processed++; deactivate C;};
  }
}
```

**To know whether a component C is contained in a component B, throw a ray from a the midpoint of a triangle of C and use the parity of the intersections with B (even means C is in B).**

G) **NPR**: Assume that you have a detailed mesh made of 20,000 triangles for each animal in a virtual farm. Given a **fixed** viewpoint E, you wish to compute 20 edges per animal, which (we hope) when drawn would let the viewer recognize the animal and its pose. Suggest an approach for computing such 20 edges. Be specific. When you describe a property of an edge, clearly indicate how that property is to be tested or computed. Note that this is a hard problem, so you are not expected to produce the best set of edges, but a reasonable guess, which may not in fact accomplish what is needed. Nevertheless, propose an approach.

*Start with the set S of silhouette edges which are those bounding a front and a back facing triangle. A triangle t is front facing if  $((g(p(c)) - g(c)) \times (g(n(c)) - g(c))) \bullet (E - g(c)) > 0$ , where g(c) is the location of the vertex of corner c. We assume that the animals have smooth shapes, and hence ignore creases.*

*Then, split the silhouette edges into short segments and discard segments whose midpoint M is hidden. This may be tested by casting a ray from M to E.*

*Now, chain the segments into runs (connected components).*

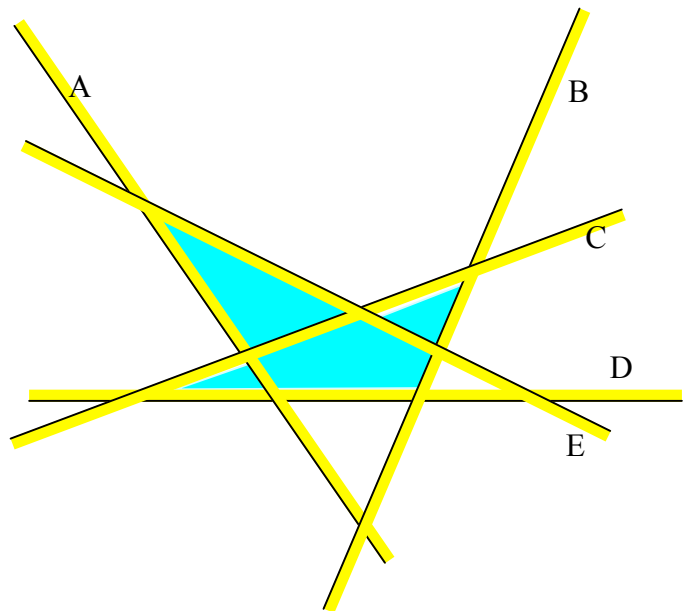
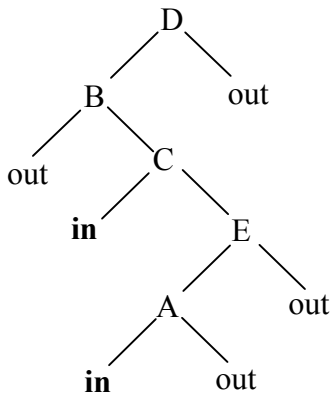
*Selectively collapse edges in these runs so as to simplify them. Always collapse next the edge that will produce the smallest error. If a run becomes a single edge E and if half the length of this edge is less than the error that would be produced by collapsing any other edge, remove E.*

*Repeat this process until no more than 20 edges are left.*

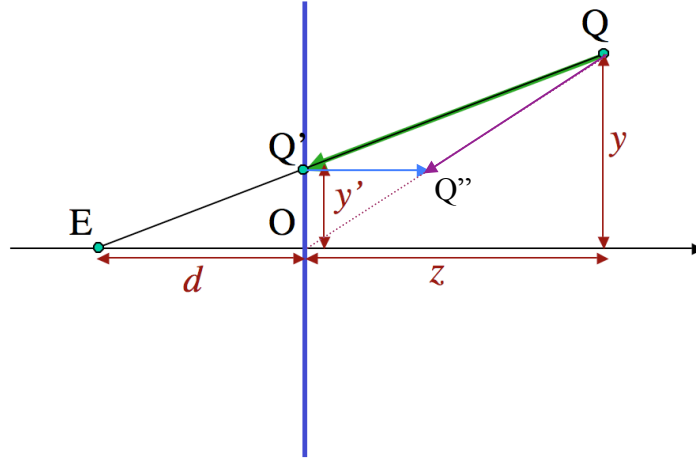
H) **BSP and CSG**: Draw a BSP tree and provide a CSG expression for the 2D set shaded in the figure using the half-spaces as marked.

**CSG** =  $CD \cap B \cup AE \cap C$

**The BSP tree is:**



I) **Perspective:** The eye E is at distance  $d$  from the screen. (1) Explain why we do not use a perspective projection during rasterization. (2) Write the perspective transform  $Q''=(x'',y'',z'')$  of a point  $Q=(x,y,z)$ . (3) Draw the construction of  $Q''$  in the figure below and explain it. (4) Assume that you are given a pixel  $(u,v)$  at which a visible point  $Q$  was projected and the corresponding  $z$ -buffer value  $w=Z[u,v]$ . Provide the detailed construction for recovering the coordinates  $(x,y,z)$  of  $Q$  in the screen coordinate system from only  $u$ ,  $v$ , and  $w$ . (Ignore the quantization and  $z$ -scaling that maps the  $z$ -buffer content to  $[-1,1]$  for all points between near and far.)



(1) Projection would lose depth information which is stored in the  $z$ -buffer for selecting the visible (front-most) surfaces.

(2)  $Q'' = dQ/(d+z)$

(3)  $Q''$  must be on the horizontal of  $Q'$  so that it projects onto the same pixel. It also is a scaling, so it lies on the line from  $Q$  to  $O$ . Hence, it is the intersection of both.

(4) We invert the perspective. Hence:  $Q = d(u,v,w)/(d-w)$

J) **Questions:** Points for the questions that you have submitted before the final.

*You get 10 points if you have provided at least 10 reasonable questions.*