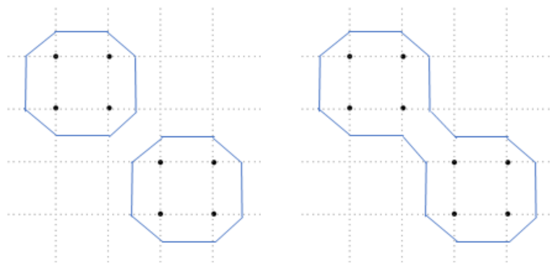
1. Marching Cubes. Sketch an evaluation grid with ambiguous cases such that two or more different solutions are equally possible representations of the boundary of the implicit curve being evaluated by the algorithm.

   Solution:

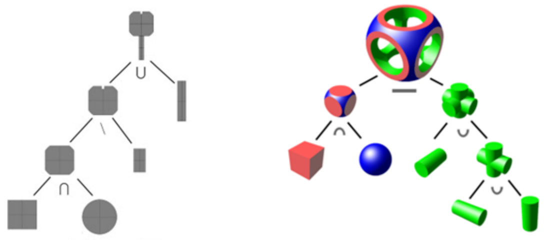
   Here is an example for 2 possible interpretations. Each grid node marked with a black point was evaluated inside the implicit equation, all the others were evaluated outside. (I will let you come up with examples with more than 2 interpretations.)



□

2. Constructive Solid Geometry. CSG can be used to create more complex objects out of simple primitives using union, intersection and difference, among other operations. Sketch a CSG object of your own design that combines union, intersection and difference operations. Draw the object at each tree node, including the primitive objects at the leaf, and the final root object. Your object can be in 2D or 3D.

   Solution: Multiple solutions are possible, here is one example in 2D and another in 3D:



□

3. CSG point classification. Most of the algorithms in CSG are better implemented recursively. Implement the point classification procedure below for a generic tree, according to the given Object and Node structures. Consider that the method "Object::classify(Point p)" is available and complete the function "classify(Node*,Point)."

```
// The CSG primitive object
class Object
{ ...
bool classify ( Point p ); // true if the object contains p, false otherwise
...
};

// A node of the CSG tree.  It can represent a primitive object or a node
// performing an operation between the two children:  na op nb
struct Node
{ Object* obj; // the primitive object if this node is a leaf, or null otherwise
Node* left; // the A node or null if this node is a leaf
Node* right; // the B node or null if this node is a leaf
Operation op; // enumerator that can be:  Union, Intersection, or Difference
};

// Returns true if the tree with given root contains point p, and false otherwise
bool classify ( Node* root, Point p )
{
}
```

Solution:

```
bool classify ( n, p )
{
If ( n->obj ) return n->obj->classify(p);
If ( n->op()==Union) return ( Classify(n->left,p) or Classify(n->right,p) );
If ( n->op()==Inter.)  return ( Classify(n->left,p) and Classify(n->right,p) );
If ( n->op()==Diff.)  return ( Classify(n->left,p) and !Classify(n->right,p) );
}
```

□

4. Euler Formula. The Euler equation is V-E+F. Give the Euler characteristic and the V, E, and F numbers for:
a) a cube model represented with 6 faces, and
b) a cone with 280 vertices and 280 faces.

Solution:

a) V=8, E=12, F=6; and since a cube has 1 shell and 0 holes/genus, then the Euler characteristic has to be 2.
b) A cone must have characteristic 2, so we have: $280 - E + 280 = 2 => E = 558$.

□

5. The genus is a topologically invariant property of a surface defined as the largest number of non-intersecting simple closed curves that can be drawn on the surface to cut it without separating it. Roughly speaking, it is the number of holes in a surface.
a) Describe the Euler equation and how the Euler characteristic encodes the genus number and the number of shells of a 2-manifold.
b) How can we determine the genus number of a 2-manifold mathematically?

Solution:

a) V-E+F = 2(s-h).

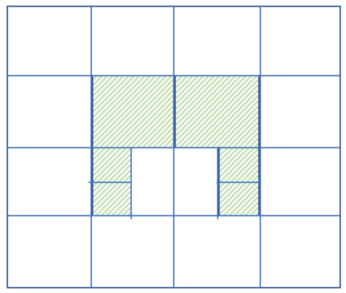
b) just isolate h in the equation above.

□

6. Quadtrees. Consider the quadrants of a quadtree representation to be encoded in the following order: bottom-left, botom-right, top-left, top-right. Now consider a quadtree string representation where characters in the string can be: B (black) for fully-filled quadrants, W (White) for empty quadrants,

or G (gray) for partially-filled quadrants. The string is encoded according to the following steps:
a) The first 4 elements of the string describe the state of the four quadrants according to the established ordering.
b) The string is then scanned, from left to right, and for each G character encountered: the respective quadrant is divided in its 4 sub-quadrants and the corresponding four new characters are appended at the end of the string.
Sketch the object described by the string below:
GGGG WWWG WWGW WBWW BWWW BWBW WBWB

Solution:



□

7. Answer the questions:

   a) What is the main difference between using tetrahedra or cubes in the Marching Cubes algorithm?

   b) How would you compare the possible ambiguous cases in marching squares (2D) versus marching cubes (3D)? Explain. What can be done to solve ambiguous cases?

   Solution:

   a) A tetrahedral subdivision will have less rules to be implemented, since only 4 vertices (instead of 8) are evaluated for defining how the surface intersects with one cell.

   b) In 3D there are more possible ambiguous cases. This happens because there are more evaluation points to consider for each volumetric cell than the 2D version of the algorithm. One possible way to solve such cases is to further subdivide the evaluation grid. Adaptive grids (like quadtrees/octrees) are always preferable.

□