Name: __________ KEY _______________  
Student ID Number: ________________

Signature: __________________________  
Fire Alarm Code: ____________________

CPSC 414 1999-2000 (T1) 1st Midterm Exam
Department of Computer Science  
University of British Columbia  
K. Booth & R. Scharein

Exam Instructions (Read Carefully)

1. Sign the first page of the exam with your Signature in the space provided on the upper left immediately.

2. Continue reading the instructions, but do not open the exam booklet until you are told to do so by a proctor.

3. Print your Name and Student Identification Number on every page in the space provided at the top of each page before you start the exam.

4. Cheating is an academic offense. Your signature on the exam indicates that you understand and agree to the University’s policies regarding cheating on exams.

5. Please read the entire exam before answering any of the questions.

6. There are four questions on this exam, each worth the indicated number of marks. Answer as many questions as you can.

7. Write all of your answers on these pages. If you need more space, there is blank space at the end of the exam. Be sure to indicate when a question is continued, both on the page for that question and on the continuation page.

8. Interpret the exam questions as written. No questions will be answered by the proctor(s) during the exam period.

9. The exam is closed book. There are no aids permitted of any kind.

10. You have 70 minutes in which to work. Budget your time wisely.

11. In the event of a fire alarm during the exam, enter the four-character code provided by the proctor(s) in the space on the upper right, then gather your belongings and exit the room, handing your exam to a proctor as you exit.

12. No one will be permitted to leave the exam room during the last ten minutes of the exam.

<table>
<thead>
<tr>
<th>Question</th>
<th>Mean</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1(a)</td>
<td>11.2</td>
<td>15</td>
</tr>
<tr>
<td>1(b)</td>
<td>0.9</td>
<td>5</td>
</tr>
<tr>
<td>2(a)</td>
<td>4.5</td>
<td>8</td>
</tr>
<tr>
<td>2(b-c)</td>
<td>10.6</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>9.8</td>
<td>16</td>
</tr>
<tr>
<td>4(a-g)</td>
<td>7.8</td>
<td>14</td>
</tr>
<tr>
<td>4(h-n)</td>
<td>9.1</td>
<td>14</td>
</tr>
<tr>
<td>4(o-t)</td>
<td>5.9</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>59.8</td>
<td>100</td>
</tr>
</tbody>
</table>
Question #1 [20 marks total]

This question tests your knowledge of the hardware components of a cathode ray tube (CRT).

(a) [15 marks] Draw a schematic diagram of a monochrome (black-and-white) CRT and include a brief description of the major components and include labels in the diagram indicating the approximate locations of the components. (The diagram in the text described nine components, your diagram might have a few more or a few less and still be a satisfactory answer to the question.)

See Figure 4.5 in the text for an example of a suitable diagram. The major components are:

- **glass tube**: is evacuated to near zero pressure and contains all of the components except for the deflection system (if it is electromagnetic - an electrostatic deflection system is inside the tube).
- **electron gun**: a source of electrons comprising three sub-components,
  - **filament** or heater, which provides thermal energy as a result of a current that runs through it much light a light bulb; the filament current is kept constant to maintain a constant temperature on the cathode
  - **cathode**: which is a metal plate whose sole purpose is to heat up and emit thermal electrons into the vacuum inside the tube, regulated by a grid through which a varying voltage (the ‘‘intensity’’ signal from the display controller) varies from zero to some negative voltage. When the grid is zero, electrons are free to flow to the other end of the CRT, but when it is highly negative none can get through.
- **focusing**: is electromagnetic or electrostatic fields to shape the electrons into a beam that converges at the back surface of the glass at the far end of the CRT.
- **acceleration**: is an electrostatic (never electromagnetic) field of high voltage (10KV to 25KV, usually at the lower range for a monochrome CRT) caused by the anode or charged metallic coating on the inside of the glass just before the far end of the CRT. The purpose of the anode is to add energy to the electrons passed by the grid so that they have lots of energy when they hit the far end of the CRT.
- **deflection**: Electromagnetic coils (outside the glass tube) or electrostatic plates (inside the tube), a pair for each of x and y, provide fields that change the direction of the electron beam left or right, and up or down under control of the x and y voltage signals from the display controller.
- **phosphor**: The substance that coats the inside of the glass at the far end of the CRT which is put into an excited state when struck by electrons and then decays (emitting a photon of light), which is what we see when we look at the front of the glass.
(b) **[5 marks]** Explain briefly how a colour beam penetration CRT differs from a monochrome CRT.

**[2 marks, one for each point]** A beam penetration CRT is identical to a monochrome CRT except that it has two layers of phosphor (one red and one green) on the inside of the glass and the accelerating system has a variable voltage, rather than a constant voltage.

**[1 mark for general explanation]** At the lowest voltage setting, the electron beam is mostly absorbed as it hits the first layer of phosphor, resulting in light (photons) characteristic of that phosphor (say red, if that is the first layer). At the highest voltage setting, the electron beam mostly passes through the first layer of phosphor (thus ‘penetrating’ the phosphor) and is absorbed as it hits the second layer of phosphor, resulting light (photons) characteristic of that phosphor (green, if that is the second layer). Intermediate voltage settings give a mixture of the two colours.

**[2 marks for any two drawbacks: costs, time, bad colour, or limited colour gamut]** There are three drawbacks to the system in addition to the extra cost associated with the variable accelerating voltage and the need for two layers of phosphor.

The first drawback is that changing the voltage of the accelerating system requires a significant amount of time compared to the drawing time of a single line, point of character. For this reason the display list usually needs to be sorted by colour, to minimize the amount of time spent waiting for the voltage to change (i.e., so that each voltage setting is only used once per frame in the ideal case). This leads to overhead in earlier parts of the graphics pipeline, because the higher-level primitives must be decomposed into their various colours and the lower-level constituents sorted by colour.

The second drawback is that the electron beam always excites some of both phosphor layers, regardless of the voltage setting, so in fact the colours are not really that ‘sharp’, but instead are always a blend of the two.

The final drawback is that adding a third layer of phosphor does not provide a full range of colour because there is no way to excite just the first and last layers without also exciting the middle layer. This means the system is inherently only a two-dimensional colour space, whereas (as we will see in Chapter 11) humans see a three-dimensional colour space.
Question #2 [24 marks total]
This question tests your knowledge of the 2-D version of the Sutherland-Hodgman algorithm, as described in the text and in lecture, for clipping a polygon whose vertices are given in clockwise order in an input array whose size MAX permits up to seven elements. Below is the main body of the clipping function given in the textbook. You may assume that the clipping boundaries are the standard ones for a $2 \times 2$ square window centred on the origin, $-1.0 \leq x, y \leq 1.0$, and that clipping is performed against the four clipping boundaries in four stages using the order Left, Right, Bottom, and then Top.

```c
void SutherlandHodgmanPolygonClip(vertex *inVertexArray,
    vertex *outVertexArray, int inLength, int *outLength, vertex *clip_boundary)
{
    vertex s, p, i;
    int j;

    *outLength = 0;
    s = inVertexArray[inLength-1];
    for (j=0; j<inLength; j++) {
        p = inVertexArray[j];
        if (Inside(p,clip_boundary)) {
            if (Inside(s,clip_boundary)) {
                Output(p,outLength,outVertexArray);
                s = p;
            }
            else if (Inside(s, clip_boundary)) {
                Intersect(s, p, clip_boundary, &i);
                Output(i, outLength, outVertexArray);
                Output(p, outLength, outVertexArray);
                s = p;
            }  
        }  
    }
}
```

Note: The code above from the textbook tests the line segment between the current vertex $p$ and the previous vertex $s$ to see if it crosses the clipping boundary, computing the intersection vertex $i$ if the line segment does cross. For the first iteration of the loop (when $j=0$), the previous vertex $s$ is the last vertex in the array.

Hint: It may be useful to draw a diagram for one or more of the stages in the clipping algorithm.
In this question, the variable `inLength` has a value of three, which is the number of vertices whose coordinates are in `inVertexArray`. The unused elements of `inVertexArray` are left blank. Use this same convention when you write your answers for Parts (b)–(c) of this question.

(a) [8 marks] Determine the numeric values of the four window edge coordinates with respect to the four boundaries of the clipping region for the second vertex (0.0, +2.0) of the input triangle given above.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>−2.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>+2.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

`inLength` 3

`inVertexArray` 

Left: \[ 1 + x = +1.0 \]

Right: \[ 1 - x = +1.0 \]

Bottom: \[ 1 + y = +3.0 \]

Top: \[ 1 - y = -1.0 \]

[2 marks each] Equations not required but numeric results must be exact, otherwise partial credit of +1 if WEC equation is correct but numeric is wrong, and similarly partial credit of +1 if a plausible but incorrect WEC is given AND the numeric value is correct for that equation. Out codes are NOT acceptable, but if all four are given instead of WECs, then +2 marks total partial credit.

The window edge coordinates are the signed distances from the respective clipping boundaries. The equations are listed above, you were only expected to provide the numeric values for this example, which result from substituting \(x=0.0\) and \(y=+2.0\) into the four equations.

The numeric signs of the four window edge coordinates form the outcodes for a vertex. In this case the outcodes would be 0001 (i.e., only the outcode for Top is a one, the others are all zero).
(b) [8 marks] Assume that the values in `inLength` and `inVertexArray` are those given on the previous page. Indicate the values in `outLength` and `outVertexArray` after the clipper is called with the Left clipping boundary. Unused entries in `outVertexArray` should be left blank.

\[
\begin{array}{c|c}
\hline
x & y \\
\hline
-1.0 & 0.0 \\
-1.0 & +1.0 \\
0.0 & +2.0 \\
\hline
\end{array}
\]

\[
\text{outLength} \quad 4
\]

\[
\begin{array}{c|c}
\hline
x & y \\
\hline
0.0 & 0.0 \quad \text{outVertexArray}
\hline
\end{array}
\]

-1 for each coordinate that is incorrect.

-1 mark if the points are in a cyclic shift of the order given here.

-1 mark if the `outLength` value does not agree with the number of entries that are filled in.

(c) [8 marks] There is no clipping performed for the Right or Bottom clipping boundaries, so assume that the output from Part (b) is copied to `inLength` and `inVertexArray` and used as the input for the next clipping stage. Indicate the values in `outLength` and `outVertexArray` after the clipper is called with the Top clipping boundary. Unused entries in `outVertexArray` should be left blank.

\[
\begin{array}{c|c}
\hline
x & y \\
\hline
-1.0 & 0.0 \\
-1.0 & +1.0 \\
-1.0 & +1.0 \\
\hline
\end{array}
\]

\[
\text{outLength} \quad 5
\]

\[
\begin{array}{c|c}
\hline
x & y \\
\hline
0.0 & +1.0 \\
0.0 & 0.0 \\
\hline
\end{array}
\]

NOTE: The vertex (-1.0, +1.0) appears twice as a result of the clipping!
Question #3 [16 marks total]
This question tests your knowledge of 3-D vector algebra and the way in which polyhedral models were defined for Assignment #1.

Given four points \( a, b, c, \) and \( d \) that form a perfect square \( S \) (in that order around the perimeter), use vector algebra to find a mathematical expression for the point \( p \) that is the apex of a pyramid with base \( S \) such that all three of the following properties hold:

(a) all four triangular faces of the pyramid are congruent (they have the same size and shape);
(b) the height of the pyramid is equal to the length of the (equal) sides of \( S \); and
(c) the points \( a, b, c, \) and \( d \) appear in clockwise order when viewed from the point \( p \).

**Hint:** Explain your work if you want partial credit for incorrect answers, and consider drawing a simple diagram to illustrate how you derived your vector expressions.

\[
p = \frac{(a + b + c + d)}{4} + \frac{(b - a) \times (c - a)}{\text{norm}(b - a)}
\]

where the norm of a vector is given by

\[
\text{norm}(b - a) = \| b - a \| = \sqrt{(b - a) \cdot (b - a)}
\]

The first term is the centroid (median) of the square, which is the exact center due to symmetry. The second term is an offset from that, normal to the plane of the square \( S \) by a distance \( s \), the length of the sides of \( S \). The numerator of that term is the cross product of two sides, one from \( a \) to \( b \) and the other from \( a \) to \( c \). This corresponds to the \( P1P2 \times P1P3 \) cross product in the text, and gives a vector whose magnitude is the area of the parallelogram defined by those vectors (in this case the square \( S \), which has area \( s^2 \)) and direction normal to the plane of the vectors (in this case the plane of \( S \)). So the numerator is exactly what we want (except maybe for orientation) but it is too long by a factor of \( s \). So we divide by \( s \), which we can find many ways, but the one above works.

The orientation is correct because (as explained in the text) the cross product assumes that the endpoints 1-2-3 are in clockwise order (in our case these are \( a \rightarrow b \rightarrow c \)). We did not assume anything about the orientation of \( S \), only that the vertices occur in order along the perimeter, so \( p \) is automatically put on the side from which the corners would be seen as clockwise!

**NOTE:** On the exam, the question did not state this explicitly, but it has to be true otherwise there is no solution at all to the problem because there are six different cyclic orderings of the four vertices but only two of them could satisfy Condition (c). Our solution positions \( p \) so that it is on the appropriate side of the plane of \( S \) so that the points appear in the desired order, clockwise.

For the first term, \( (a + c)/2 \) or \( (b + d)/2 \) would work equally well (it’s a square!). For the norm in the denominator of the second term, any of the sides could have been used, and for the numerator, there are lots of possibilities so long as the convention is clockwise.
Question #4 [40 marks – 2 marks each]

Explain the following terms as they have been used so far in our course on computer graphics. If the term is an **acronym**, give the full name for the acronym **and** its meaning, usage, or how it functions (i.e., do not just provide the full name, but also explain what the acronym identifies).

(a) aliasing

Visual artifacts such as the "jagged" corners on lines or characters that result from the requirement that only pixels at integral coordinates be displayed, which is an inherent problem when continuous signals (images) are approximated by discrete samples.

(b) calligraphic display

A "line drawing" or "vector" display in which points, line segments and possibly curve segments (such as conic sections) are drawn directly by the hardware (as opposed to a raster or bitmap display).

(c) convex set

A set $S$ is **convex** if and only if for any two points $a$ and $b$ and any scalar (real number) $\lambda$ in the range $0 \leq \lambda \leq 1$ the point $(1-\lambda)a + \lambda b$ is in the set $S$. **Note:** This is equivalent to saying that every $a + \lambda(b-a)$ must be the set $S$.

(d) DDA

**Digital Differential Analyzer** – an incremental algorithm for drawing lines or more generally circles, ellipses, and other curves based on forward differencing equations derived from discretizing the differential equations of the curves. For the simple case of straight lines, there are very efficient implementations, such as the mid-point algorithm based on the work of Bresenham, Pitteway, and others. These use only integer arithmetic and mostly adds or add-by-one operations in the inner loop and are suitable for implementation directly in hardware.

(e) frame buffer

The hardware that implements a bitmap in which memory is addressed using $(x,y)$ addresses that correspond to pixel locations on the screen (this depends on the addressability of the frame buffer) and every pixel has one or more bits associated with it, sometimes considered to be multiple fields (such as the R, G, and B fields in a "full colour" 24-bit frame buffer).

(f) `glBegin()`

An OpenGL function that specifies the beginning of a group of geometric primitives that follow the call to `glBegin()`.

An OpenGL function that is used to define a graphical object. A call to `glBegin()` is followed by calls to one or more OpenGL functions that define primitives or transformations associated with the object.
(g) **gluLookAt()**

A GLUT function that defines a viewing transformation given an eyepoint, a center of view (the point being "looked at"'), and a vector indicating the "up direction" for the viewer. The function computes appropriate rotation and translation matrices that are applied to the current viewing transformation. The up direction must not be collinear (in the same direction as) the line of sight between the eyepoint and the point being looked at, because there is no unique viewing transformation in that case.

(h) **GLUT**

The "Graphics Library Utility Toolkit" provides a set of "widgets" and auxiliary functions for quickly building user interfaces using lower-level functions in OpenGL and X Windows.

(i) **graphics pipeline**

The name given to the total process through which an abstract representation of a scene or object (usually a data structure that is designed specific to a particular application) is sequentially transformed into various intermediate representations that capture the geometry and other properties, ultimately into a bitmap representation in the frame buffer, then to analog voltages to the CRT and finally light sensed by the eye and processed by the human visual system.

The sequence of operations (many of them geometric transformations) that turn a high-level description of a scene into an image on a display screen. Some of the steps (in order) are a model (data structure), modeling transformations to a transformed model (display list), viewing and windowing transformations to a transformed model in window coordinates, clipping and perspective transformations in some order, a viewport transformation, scan conversion, and finally digital-to-analog conversion to form an electronic or hardcopy image. [A diagram, or even the general idea of a sequence of well-defined steps from a model or data structure to a bitmap or electronic image is all that is required for full marks.]

(j) **outcode**

One-bit quantities that specify whether a point is "in" (value=0) or "out" (value=1) with respect to each of the boundaries of a clipping region. There are four outcodes for standard 2-D clipping (left, right, bottom and top boundaries) and six outcodes for standard 3-D clipping (left, right, bottom, top, far, and near boundaries). Often the outcodes are thought of as a set, an represented as a four-bit (or six-bit) quantity. The outcodes can be used to quickly determine all trivial accept cases (no clipping required) and some trivial reject cases where clipping would result in nothing being visible.
(k) pixel
A "picture element" corresponding to specific \((x,y)\) position within a raster image. Each pixel has an intensity level that is on/off in the simplest case of a bi-level display, anywhere from 2-16 bits of grey-scale intensity for a monochrome display, or 24 or more bits of RGB (red-green-blue) intensities for a colour display. Also associated with a pixel may be a z-depth, transparency or coverage information (the "alpha" channel), and other information used by the display hardware and software.

(l) scan conversion
The process of converting analytic graphics primitives such as points, lines, polygons, text, and higher-order curves or surfaces to their bitmap or pixel representation in the frame buffer. This is also known as "rasterization".

(m) SRGP
The "Simple Raster Graphics Package" is a complete set of, low-level 2-D drawing and input primitives designed for a raster display system. The X Window System offers a similar functionality through a different set of primitives.

(n) wire-frame rendering
A drawing style in which only the edges of polygons or other primitives are drawn (not the interior areas). Line drawing or calligraphic displays always do wireframe rendering; raster displays can do either wireframe or filled rendering.
Each term below is a **person’s name**. Enter **all** of the letters of the important things that the person contributed to the field of computer graphics from the following list.

| A | co-inventor of electricity and magnetism |
| B | co-inventor of hypertext |
| C | co-inventor of the mouse |
| D | co-inventor of outcodes |
| E | co-inventor of polygon clipping |
| F | co-inventor of fast scan conversion algorithms |
| G | co-author of our textbook |
| H | generally credited with founding interactive computer graphics |

**Note:** Some people worked on the same contributions (so some of the letters are used more than once) and some people worked on more than one contribution (so they have more than one letter).

(o) Jack Bresenham  
(p) Daniel Cohen  
(q) Douglas Engelbart  
(r) James Foley  
(s) Michael Pitteway  
(t) Ivan Sutherland

+2 if exactly correct  
+1 mark if some correct but some missing or some added  
0 otherwise