CSC 418/2504 Computer Graphics, Winter 2013
Assignment 1 (10% of course grade)

Due 11:59pm on Wed., Feb. 6, 2013.

Part A [50 marks in total]

Below are different exercises covering different topics from the first weeks of class. They require thought, so you are advised to consult the relevant sections of the textbook, the online lecture notes and slides, and your notes from class well in advance of the due date. Your proofs and derivations should be clearly written, mathematically correct, and concise.

All questions require showing the steps toward the solution, and marks will be subtracted if this is not the case. Even if you cannot answer a question completely, it is very important that you show your (partial) answers and your reasoning. Otherwise your TA will not be able to award you partial marks.

Both Part A and Part B must be electronically submitted. Part A must be in PDF format, by scanning your handwritten solution or by using LaTeX/word to typeset it.

Points & Polygons [8 marks total]

1) [4 marks] Suppose a simple (i.e. non-intersecting) polygon’s points are \( p_1, p_2, \ldots, p_n \). If, when looking along the line segment from point \( p_i \) to point \( p_{i+1} \), the interior of the polygon lies to the right, the polygon’s winding order is clockwise. Otherwise, it is counterclockwise. Give an algorithm to determine a simple polygon’s winding order, regardless of whether it is concave or convex.

2) [4 marks] A polygon is defined to be convex if, for any two points inside the polygon, all points on the line between those two points lie inside the polygon.

An infinitely long line can divide the set of 2D points into two sets: those on one side of the line, and those either on the other side or on the line itself. Prove that, by using the lines corresponding to the edges of a convex polygon, you can tell if a given point is outside the polygon.

2D Curves [12 marks total]

2) A spiral-shaped curve can be defined parametrically by the following:

\[
\begin{align*}
x(t) &= t \cos(t) \\
y(t) &= t \sin(t) \\
0 &\leq t \leq 3\pi
\end{align*}
\]

a) [3 marks] Show that this parametric curve is a section of the curve defined by the implicit function:

\[
f(x, y) = \tan\left(\sqrt{x^2 + y^2}\right) - \frac{y}{x} = 0
\]
b) [2 marks] Give an example of a point that satisfies the implicit equation but does not lie on the parametric curve defined above.

c) [4 marks] Use the parametric representation to give equations for the tangent of the curve and the unit normal at a given value of $t$. Choose the normal that faces outward from the spiral.

d) [3 marks] Compute the intersection of the spiral with a circle of radius $r$ centered at the origin.

**Transformations** [30 marks total]

3) [10 marks] Let $f$ be an arbitrary homography, and let $p_1, p_2, p_3$ be the vertices of a triangle. Prove that if the triangle contains a point $q$, the homography-transformed triangle defined by $f(p_1), f(p_2), f(p_3)$ may not contain point $f(q)$. In other words, the inclusion relationship between points and triangles is not preserved under arbitrary homographies.

4) [8 marks] Two transformations $f_1$ and $f_2$ commute when $f_1 \circ f_2 = f_2 \circ f_1$. A point $p$ is a fixed point of transformation $f$ if and only if $f(p) = p$. For each pair of transformations below, specify whether or not they commute. In each case, your solution can either be a derivation that proves/disproves commutativity, or if $f$ and $g$ do not commute, a specific counter-example.

(a) Two different translations
(b) A rotation about the origin and a uniform scaling
(c) A rotation about the origin and a non-uniform scaling
(d) A shear with respect to the x-axis and a non-uniform scale about the origin

5) (a) [6 marks] Prove that a shear in $y$ can be created with a combination of rotations and shears in $x$.

(b) [6 marks] Show that any 2D rotation can be achieved as a series of 3 consecutive shears.
**PartB: Programming (50 marks)**

The figure below shows an articulated planar robot with ten parts and ten degrees of freedom.

![Articulated Robot Diagram](image)

It has seven rotational joints, depicted with small circles; each has one rotational degree of freedom. The outer limbs of each arm scale non-uniformly outward along the given arrows, providing two more degrees of freedom. The entire character moves up and down as indicated by the arrow near the base, providing one translational degree of freedom. Your task is to render and animate such a robot using OpenGL.

Hierarchical objects like this are often defined by specifying each part in a natural, part-based coordinate frame along with transformations that specify the relative position and orientation of one part with respect to another. These transformations are often organized into a kinematic tree (e.g., with the torso as the root and the head as a leaf). In addition to the kinematic tree, one must also specify the transformation from the root (e.g., the torso) to the world coordinate frame. Then, for example, to draw the torso you transform the points that define the torso from the torso's coordinate frame to the world coordinate frame, and then from the world coordinate frame into display coordinates. Then to draw an arm, you must transform the points that define the arm in the arm coordinate frame to the torso's coordinate frame, and then from the torso's coordinate frame to the world coordinate frame, and then into display coordinates. And so on down the tree.

Rendering articulated objects is easiest if a current part-to-device mapping is accumulated as you traverse the object/part hierarchy. You maintain a stack of coordinate transformations that represents a sequence of transformations from the current part coordinates, up through the part hierarchy to world coordinates, and finally to display coordinates. For efficiency, do not apply each of the transformations on the stack in succession. Rather, the top of the stack always represents the composition of the preceding transformations. OpenGL provides mechanisms to help maintain and apply these transformations.

**Your programming task**

Your task is to design and render the articulated robot using OpenGL. When the program is run, the robot should move (i.e. animate) in order to test that the rendering is done correctly. All parts of the robot should be visible for the duration of the animation.
To accomplish this, you must perform the following tasks:

a: [5 marks] Design the parts in terms of suitable geometric shapes and deformations, and draw them using OpenGL. Use a minimal number of shapes to create the needed parts.

b: [10 marks] Design and implement transformations that map each part’s local coordinate frame to the coordinate frame of its parent in the kinematic tree. Extend the GUI with additional spinners to control each of the 10 degrees of freedom. Hint: The interactive degree of freedom controls will be useful when debugging the transform hierarchy that you build.

c: [10 marks] Design and implement a set of functions that will control the animation; i.e., they will control the state of each joint in each frame. You can use simple functions such as sinusoids to control the way in which parts move with respect to one another. Or, if you wish, you could specify a sequence of specific joint angles that the rendering will loop through. You can also use key frames to specify a few key poses for the robot (in terms of the joint angles) and linearly interpolate between them for smooth animation. Hint: You can use the GUI built for part b to choose the set of key frame values or help you specify the values for the joint angles that produce the desired animation.

d: [15 marks] Put the above together to generate your animation by drawing each part in turn as you descend the kinematic tree (once per frame). Use the OpenGL transformation stack to control relative transformations between parts, the world, and the display. It is not necessary to write code that could be used to render arbitrary articulated objects, thereby requiring that your code be able to traverse any kinematic tree. To keep things simple, you may hardcode the sequence of parts that are drawn.

e: [2 marks] Be sure to also draw small circles which depict the locations of the rotary joints and the larger circles that represent the eyes.

f: [8 marks] Explain everything you did in a written report (described below).

**Starter Code**

To get you started, we have created a simple demo for you. This will show you how to use the basic commands of OpenGL to open a window and draw some simple shapes. It will also provide you with a template Makefile for compilation and linking of your program.

This demo program opens a window and animates two squares connected by a hinge. To unpack, compile, and run this demo on CDF, download the sample code from the Assignments section of the course website and use the following commands:

```
tar xvfz a1.tgz
cd a1/robot
make
./robot
```

**Compilation**

To compile programs easily, we have set up a simple makefile that builds the executable using the `make` command. Make searches the current directory for the file called Makefile, which contains instructions for compilation and linking with the appropriate libraries. You will find this Makefile useful when compiling programs for your later assignments. For
example, if you wish to compile a program with a different name, change all occurrences of "robot" to the name of the file you wish to compile. You can also change the Makefile to include code from several files by listing the C++ source files (i.e., the files ending in .cpp) on the line "CPPSRCS=". The name of the executable file is determined by the name you use in the Makefile on the line "PROGRAM = robot". When you run the demo program, a graphics window will appear. The size of the window is determined by parameters (xmax and ymax) to the initialization routine. Each pixel is indexed by an integer pair denoting the (x, y) pixel coordinates with (0, 0) in the top left hand corner and (xmax, ymax) in the bottom right.

**Turning in your Solution to Part B**

All your code should remain in the directory a1/robot. In addition to your code, you must complete the files CHECKLIST and REPORT contained in that directory. *Failure to complete these files will result in zero marks for your assignment.*

The REPORT file should be a well-structured written (or diagrammatic) explanation of your design, your part descriptions, and your transformations. The description should be a clear and concise guide to the concepts, not a simple documentation of the code. In addition to correctness, you will also be marked on the clarity and quality of your writing. We expect a well-written report explaining your design of parts and transformations.

Note that this file should not be thought of as a substitute for putting detailed comments in your code. Your code should be well-commented if you want to receive full (or even partial) credit for it.

To pack and submit your solution, execute the following commands from the directory containing your code (i.e. the directory containing the a1 folder):

```bash
    tar cvfz alb-solution.tgz a1
    submit -c csc418h -a Alb alb-solution.tgz (if registered for CSC418)
    submit -c csc2504h -a Alb alb-solution.tgz (if registered for CSC2504)
```

**Compatibility**

All of your assignments must run on CDF Linux. You are welcome, however, to develop on other platforms and then port to CDF for the final submission. The sample code is designed to work on CDF Linux, but should be portable to other platforms (including Mac OS X and Windows with Visual C++). If you are running your own machine, it almost certainly has OpenGL installed; if not, you can search for an rpm or go to http://www.opengl.org/ (see their getting started FAQ). Mac OS X users might need to install the glui library.

If you are planning to develop for windows with Visual C++, we have included some starter code to help you out. Unpack the starter code and place the unpacked files and the include directory in your a1/robot directory. You will then need to add the skeleton code to your Visual C++ project.

**If you develop on a platform other than CDF Linux machines, be sure you know how to compile and test your code on CDF well before the deadline.** Your assignment must run on the CDF Linux machines to receive any credit. Several marks will be deducted if your code does not compile and/or run without modification. If the marking TA cannot easily figure out how to compile and execute the code, it will most likely receive zero marks. If you choose to not develop the assignment on CDF Linux, test the porting of your code to CDF Linux long before the deadline, perhaps even before you have finished the assignment. Often bugs that are "hidden" when compiling on one platform make their presence known by crashing the application on a different platform. There will not be any special conditions allowed for problems encountered while porting at the last minute.