CSC 418/2504 Computer Graphics, Fall 2015
Assignment 1

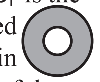
Due online by midnight Wed, Oct 14, 2015

Part A [50 marks in total]

Below are 4 exercises for you to work through, covering different topics from the first weeks of class. Some of these problems will require considerable thought. You are also advised to consult the relevant sections of the course textbook as well as your notes from class. Your proofs and derivations should be carefuly written, mathematically correct, concise and clear. In many of these problems it is necessary to show steps toward the solution. Show your work, or you will not be eligible for partial marks. Electronically submit both part A (in PDF format) and part B.

1. The revolution of a planet around its star is proposed to be a wiggly-ellipse in 2D with the following parametric form:
   $$x(t) = 4\cos(2\pi t) + 1/16 * \cos(32\pi t) \quad , \quad y(t) = 2\sin(2\pi t) + 1/16 * \sin(32\pi t)$$
   Find the tangent vector [2 marks] and a normal vector [2 marks] to the wiggly-ellipse as a function of the time. Is the curve symmetric around the X-axis? Y-axis? (proof/counterexample) [2 marks]? What is the formula to compute this curve's perimeter [2 mark]? How can one piecewise approximate this perimeter [3 marks]?

2. Suppose you wish to find the intersection(s) of a 2D line and a circle. Let $\bar{p}(\lambda) = \bar{p}_0 + \lambda\vec{d}$ be a line in 2D, where $\bar{p}_0$ is a 2D point, and $\vec{d}$ is a 2D vector. Let $||\bar{q} - \bar{p}_1||^2 = r^2$ be the implicit formula of the circle, where $\bar{p}_1$ is the center of the circle, r is the radius of the circle, and $\bar{q}$ is a point on the circle. A 2D donut can be defined as the grey region between two concentric circles of radius r_1 and r_2 ($r_1 < r_2$). What is the area (shown in grey) of the donut [1 mark]? How many intersections can there be between the line and the boundary of the donut [1 mark]? Describe an algorithm to compute the number [2 marks], location(s) [4 marks] of intersection(s). If the line and donut are both transformed by a non-uniform scale (s_x, s_y) around the origin, how do the number of intersection(s) and their location(s) change [2 marks]? How do the number of intersection(s) and their location(s) change if this transformation is only applied to the donut [3 marks]?



3. Two transformations f_1 and f_2 commute when $f_1 \boxtimes f_2 = f_2 \boxtimes f_1$. A point $\bar{p}$ is a fixed point of a transformation f if and only if $f(\bar{p}) = \bar{p}$. For each pair of transformations below, specify whether or not they commute in general. Moreover, if you conclude that they commute, provide a proof, and if you claim the converse, provide a counterexample as proof [2 marks for each part].

   (a) translation and uniform scaling

   (b) translation and non-uniform scaling

   (c) scaling and rotation, both having the same fixed points

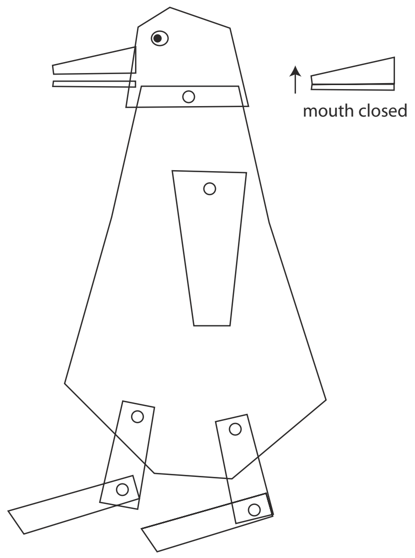
   (d) scaling and scaling, having different fixed points

   (e) translation and shear

4. Given an arbitrary, non-degenerate 2D triangle with vertices $\bar{v}_0$, $\bar{v}_1$, and $\bar{v}_2$, write a procedure for determining if a point $\bar{q}$ is inside / outside the triangle [4 marks], or on an edge of the triangle [2 marks]. The procedure can be written in English sentences or in pseudocode, as long as the steps are clear. How can one triangulate a quadrilateral such that it is the union of two triangles [3 marks]? Give a procedure that can triangulate any n-sided convex polygon [2 marks]? If the procedure will not work in general for concave polygons provide a counterexample [1 mark]. How can one use the point in/out/on a triangle procedure (or the idea behind it) to perform a point in/out/on a convex polygon test [4 marks]?

Part B: Animating a Hierarchical 2D Object [50 marks in total]

The figure below shows an articulated nine-part, nine-degree-of-freedom (DOF) planar robot penguin:



It has six joints (depicted by circles), each with a rotational degree of freedom. The beak has a single translational degree of freedom: the beak can only move up or down (in the coordinate frame of the head). The penguin can translate in X and Y. Your task will be to render and animate the penguin 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 jaw 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 device 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 device 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 device coordinates. For efficiency, we 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 penguin using OpenGL. When the program is run, the robot should move (i.e., animate) in order to help test that the rendering is done correctly.

To accomplish this, you must perform the following basic tasks:

- **(a)** [5 marks] Design the parts in terms of suitable generic shapes and deformations, and draw them using OpenGL. i.e. define shapes like (drawSquare, drawCircle, drawJoint...).
- **(b)** [10 marks] Design and implement suitable transformations that map each part's local coordinate frame to the coordinate frame of its predecessor in the kinematic tree. Extend the GUI with additional spinners to control each of the 9 degrees of freedom (DOFs). i.e. add GUI controls for each joint's rotation/translation and an animation checkbox; ensure these values are used when rendering the joints. Hint: The interactive DOF controls will be useful in debugging the transform hierarchy you build.
- **(c)** [10 marks] Design and implement a set of functions that will control the animation, i.e., will control the state of each joint in each frame. You can use simple functions such as sinusoids to define 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-framing to specify a few key poses for the penguin (in terms of the joint angles) and linearly interpolate between them for smooth animation. *H*int: You can use the GUI built for (b) to choose the set of key-frames or help you specify the values for the joint angles that produce the desired animation.
- **(d)** [15 marks] Put it all 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 device. i.e. display the parts of the penguin by calling draw for the different shapes in (a) after transforming them according to the values defined in (b).
- **(e)** [2 marks] Be sure to also draw the small circles which depict the locations of the rotary joints.
- **(f)** [8 marks] Write a 1-page report explaining how the penguin works (animate, transform,draw...).

Helper Code

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

This simple demo program opens a window, and animates two squares connected by a hinge. To unpack, compile and run this demo on CDF, download the file *a1.tgz* and use the following commands

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

Compilation

To compile programs easily we have set up a simple makefile for you, that builds the executable using the *make* command you issued above. *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 *penguin* to the name of the f ile 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 = penguin*.

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/penguin. 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 on your assignment*.

The *REPORT* f le should be a well-structured written (or diagramatic) 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., *a1/penguin*):

```
cd ../../
tar cvfz a1-solution.tgz a1
submit -c csc418h -a A1b a1-solution.tgz   (if registered for CSC418)
submit -c csc2504h -a A1b a1-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). This sample code is designed to work on Linux and Mac OS X machines, but should be portable to other Unix platforms (including 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).

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/penguin* directory. You will then need to add the skeleton code to your Visual C++ project.

If you develop on a different platform, 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 configuration. Several marks will be deducted if your code does not compile and/or run without modification. If the marker cannot easily figure out how to compile and execute the code, it will most likely receive zero marks. If you choose to develop the assignment in MacOSX or Windows, test porting your code to 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. We will not be sympathetic to porting problems that you have at the last minute.