

# CSC 418/2504 Computer Graphics, Fall 2007

## Assignment 1

**Part A Written: Due in class Wed , Oct 3, 2007 [50 marks]**

**Part B Programming: Due online on Wed, Oct 3, 2007 at midnight [50 marks]**

### Part A [50 marks in total]

Below are 5 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 carefully written, mathematically correct, concise and clear. In many of these problems it is necessary to show steps toward the solution. Show your work.

1. The revolution of a planet around its sun can be approximated with an ellipse in 2D having the following parametric form:

$$x(t) = a \cos(2\pi t) \quad , \quad y(t) = b \sin(2\pi t)$$

Find the tangent vector and a normal vector to this ellipse as a function of the time  $t$ . (A normal vector is any vector perpendicular to the tangent). Write the transformation matrix that makes the curve a circle of radius 1.

2. Suppose you wish to find the intersection(s) of a 2D line and a circle. Let  $\vec{p}(\lambda) = \vec{p}_0 + \lambda \vec{d}$  be a line in 2D, where  $\vec{p}_0$  is a 2D point, and  $\vec{d}$  is a 2D vector. Let  $\|\vec{q} - \vec{p}_1\|^2 = r^2$  be the implicit formula of the circle, where  $\vec{p}_1$  is the center of the circle,  $r$  is the radius of the circle, and  $\vec{q}$  is a point on the circle. Derive mathematical expressions and a simple algorithm that you might use to compute the number and the location(s) of the intersection(s).
3. Two transformations  $f_1$  and  $f_2$  commute when  $f_1 \circ f_2 = f_2 \circ f_1$ . A point  $\vec{p}$  is a fixed point of a transformation  $f$  if and only if  $f(\vec{p}) = \vec{p}$ . 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.
  - (a) translation and uniform scaling
  - (b) translation and non-uniform scaling
  - (c) scaling and rotation, both having the same fixed points
  - (d) rotation and another rotation, having different fixed points
4. Given a simple polygon with vertices  $\vec{v}_0, \vec{v}_1, \dots, \vec{v}_n$  write a procedure to determine if the polygon is convex. Write a procedure to triangulate this polygon. The algorithm should be more efficient if the polygon is convex.
5. Given an arbitrary, non-degenerate 2D triangle with vertices  $\vec{v}_0, \vec{v}_1$ , and  $\vec{v}_2$ , write a procedure for determining if a point  $\vec{q}$  is inside the triangle, outside the triangle, or on an edge of the triangle. This procedure can be written in English sentences or in pseudocode, as long as the steps are clear. *Hints:* Review the procedure for clipping a line to a viewport (e.g., Section 12.1 of the textbook). How can you determine if two points are on the same side of a line?

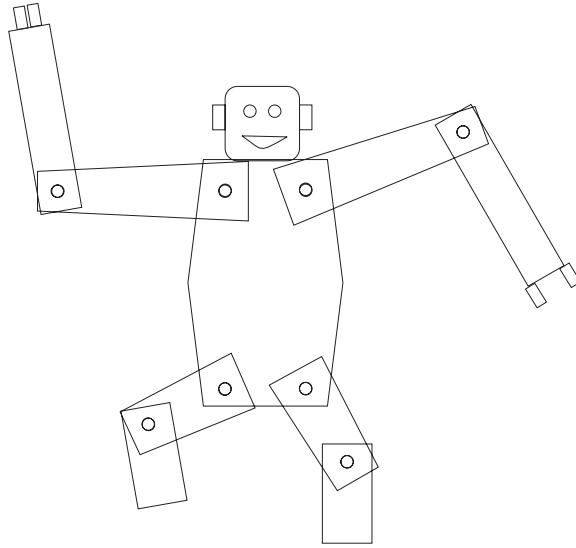


Figure 1: A simple 2D hierarchical object

## Part B [50 marks in total]

### Background

Figure 1 shows an articulated 10 part, 10 degree-of-freedom (DOF) planar robot monkey. It has eight rotational joints (depicted by circles), each with 1 rotational degree of freedom. The hands are grippers, each with a single translational degree of freedom; the grippers move *only* together or apart along the ends of monkey's lower arms.

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 don't 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 the transformations.

### Programming Problem

Your task is to design and render the articulated robot monkey in Figure 1 using OpenGL. When the program is run the robot should move (animate) in order to help test that the rendering is done correctly. To design the object you will need to

1. design and generate the part descriptions in terms of suitable generic shapes and deformations, and then
2. design and generate suitable transformations that map each part's local coordinate frame to the coordinate frame of its predecessor in the kinematic tree.

Render the object by drawing each part in turn as you descend the kinematic tree. Be sure to also draw the small circles which depict the locations of the rotary joints. Use the OpenGL transformation stack to control relative transformations between parts, the world and the display device. It is not necessary to write code that could be used to render arbitrary articulated objects, thereby requiring that your code can traverse any kinematic tree. You may *hardcode* the sequence of parts that are drawn.

For the animation you can use simple functions such as sinusoids to define the way in which parts move with respect to one another. You could also use randomized motion like a Gaussian random walk from one frame to the next. Or if you wish you could specify a sequence of specific joint angles that the rendering will loop through.

## Marking:

The work you do on this assignment should be your own. The course policy concerning extensions and late assignments are given on the course web site.

**Part A:** Hand in your written solutions for Part A on paper by the due date and time in the drop box.

**Part B:** Hand in a paper listing for the program along with a concise report in the drop box. The report should be a well-structured written/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.

For the electronic version of your solution, submit the assignment on CDF using one of the following command-line options:

- `submit -N alb csc418h filename1 filename2 ...`
- `submit -N alb csc2504h filename1 filename2 ...`

Please do not tar or compress your files.

Your assignment must run on the CDF Linux configuration. Several marks will be deducted if your program does not compile and run without modification.