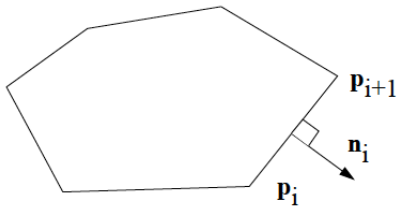
CSC 418/2504 Computer Graphics - Fall 2011: Assignment 1

Part A due in class on Wednesday, October 5, 2011 (7:10 PM)
Part B due online by 11:59 PM on Wednesday, October 5, 2011

Part A: Written (50 marks)

This section consists of 5 exercises covering topics from the lectures of the first weeks. They require thought; you are advised to consult relevant sections of the textbook and lecture notes well in advance of the deadline. Proofs and derivations should be clearly written, mathematically correct, and concise. Show steps toward each solution; marks will be subtracted if your derivations or proofs do not include such steps. Partial credit will be awarded, including partial credit for solutions that are only partially complete; it is very important to show your partial answers and reasoning to get such credit.

1: (11 marks) You are given the vertices of a convex polygon in the 2D plane in counter-clockwise order as ($\mathbf{p}_1$, ..., $\mathbf{p}_n$). The coordinates of vertex $\mathbf{p}_i$ are (x_i, y_i).



1a: (3 marks) Give an expression for the coordinates of the outward-facing normal $\mathbf{n}_i$ of the edge connecting $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$.

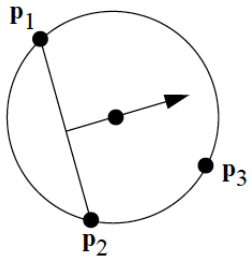
1b: (3 marks) Let $\mathbf{q}$ = (x_q, y_q) be an arbitrary point on the plane containing the given polygon. Let $\mathbf{l}$ be the line containing $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$, and let $\mathbf{n}_i$ be the outward-facing normal vector as given in part a.

What is a test that determines whether or not $\mathbf{q}$ lies on the outward side of $\mathbf{l}$ (the side toward which $\mathbf{n}_i$ points)?

1c: (5 marks) Provide an algorithm that determines whether a 2D point $\mathbf{q}$ is inside, outside, or on the boundary of the given polygon. *Hint*: Each edge is contained in an infinite line. Each infinite line divides the 2D plane into two half-planes: the "left" half-plane and the "right" half-plane (left and right are defined with respect to a counter-clockwise direction of traversal of the vertices). The key insight you should use is that the interior of a convex polygon is the intersection of the left half-planes of each edge of the polygon.

2: (7 marks) You are given a polygon with vertices ($\mathbf{p}_1$, ..., $\mathbf{p}_n$), and you know that the edges of the polygon are of equal length. Write an algorithm that determines if the polygon is regular without using trigonometric functions. Explain any tests you use in the algorithm.

3: (10 marks) A circle *circumscribes* a set of points if the points lie on the circle:



3a: (3 marks) The perpendicular bisector of a line segment is the line perpendicular to the line segment that passes through its midpoint. A ray that lies along the perpendicular bisector whose origin is located at the midpoint is shown in the figure above. Derive equations for the perpendicular bisectors of the two segments ($\mathbf{p}_1$, $\mathbf{p}_2$) and ($\mathbf{p}_1$, $\mathbf{p}_3$).

3b: (4 marks) The center of the circle circumscribing the three points lies on the perpendicular bisector of *any* segment connecting two points on the circle. Use the equations derived in part a to find an expression for the center of the circle in terms of $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$. Assume these points are not collinear.

3c: (3 marks) Write the implicit equation for the circle that circumscribes $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$.

4: (12 marks) Transformations f and g are said to commute if and only if $f(g(\mathbf{p})) = g(f(\mathbf{p}))$ for all $\mathbf{p}$. For each of the four cases given below, prove whether or not f and g commute, where f and g are 2D affine transformations. You may prove non-commutativity by giving a counterexample.

4a: (3 marks) Both f and g are arbitrary rotations about the same pivot (also known as a fixed point).

4b: (3 marks) f and g are arbitrary rotations about different pivots.

4c: (3 marks) One is an arbitrary rotation about the origin and one is a uniform scaling.

4d: (3 marks) One is a translation and one is a non-uniform scaling.

5: (10 marks) A projectile launched from a cannon located at the origin follows a trajectory given by the parametric equations:

$$x(t) = at$$
$$y(t) = (-1/2)gt^2 + bt$$

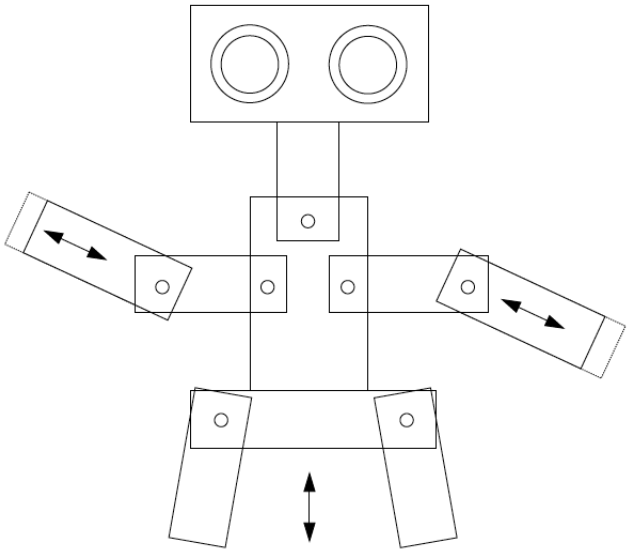
where the initial velocity is $\mathbf{v} = (a, b)$ and g is the gravitational constant. $t \in [0, t_i]$ is the free parameter corresponding to time and t_i is the time of impact with the ground, which corresponds to $y = 0$.

5a: (4 marks) Give expressions for the tangent and normal vectors of the projectile trajectory as a function of time t.

5b: (6 marks) Give an expression for the time of impact t_i. Also give expressions for the location and velocity at the time of impact.

Part B: Programming (50 marks)

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



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 part. 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. This generalizes to any part down the kinematic 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

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 generic 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 controls to modify 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 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 frames to specify a few key poses for the robot (in terms of the joint angles) and linearly interpolate between them for animation. Hint: You can use the GUI built for part b to choose the set of key frames 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 among 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. However, you are more than welcome to write more general code if you desire. To receive credit, any more general code must be capable of drawing the robot.

e: (2 marks) Include 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).

Feel free to be creative in adding detail to the shapes of the parts, but make sure they have the approximate shapes as depicted in the diagram with the appropriate degrees of freedom located in the correct places. You are also welcome to vary the color of the parts. These details will not affect your marks, however.

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 programs. 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 report and name it "report.txt", which you should place in the a1/robot directory. The report.txt file should be a well-structured written explanation of your design, your part descriptions, and your transformations. In particular, describe 1) how you represent your model and where in your code the representation is located 2) how you traverse the parts of your model to render it, and where in your code this is located 3) how you specify the motion of each degree of freedom and where in your code this motion is specified. The description should be a clear and concise guide to the concepts, not simply a 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.

Specific submission commands will be announced on the course web page prior to the submission deadline.

Compatability

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 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.