CPSC 414
Assignment 1

Due 4pm, Friday February 6, 2004
(handin box in the CICSR basement)

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Name: ____________________________________________

Student Number: __________________________________

<table>
<thead>
<tr>
<th>Question 1</th>
<th>/ 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 2</td>
<td>/ 4</td>
</tr>
<tr>
<td>Question 3</td>
<td>/ 3</td>
</tr>
<tr>
<td>Question 4</td>
<td>/ 6</td>
</tr>
<tr>
<td>Question 5</td>
<td>/ 2</td>
</tr>
<tr>
<td>Question 6</td>
<td>/ 2</td>
</tr>
<tr>
<td>Question 7</td>
<td>/ 20</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>/ 46</td>
</tr>
</tbody>
</table>
1. Transformation as a Change of Coordinate Frame

(a) (4 points) Specify the coordinates of point $P$ with respect to coordinate frames $A$, $B$, $C$, and $D$.

(b) (2 points) Derive a transformation that takes a point expressed with respect to frame $D$ and re-expresses it in terms of frame $B$, i.e., determine $M_{D\rightarrow B}$, where $P_B = M_{D\rightarrow B}P_D$. Verify your solution using your answer to part (a).

(c) (3 points) Derive a transformation that takes a point from frame $C$ to frame $B$, i.e., determine $M_{C\rightarrow B}$, where $P_B = M_{C\rightarrow B}P_C$. Verify your solution using your answer to part (a).
2. Composition of Transformations

(a) (3 points) The following figure shows a house drawn with an identity transformation. Sketch the result of performing the sequence of OpenGL transformations given below. Give the corresponding $3 \times 3$ transformation matrix, and describe what each of the 3 columns in the transformation matrix represents.

```c
glTranslatef(4,1,0);
glRotatef(-90,0,0,1);
drawHouse();
```

(b) (1 point) Determine how the same transformation can be accomplished by a rotation followed by a translation, e.g.,

```c
glRotatef(theta,0,0,1);
glTranslatef(x,y,0);
```
3. (3 points) Give the code for a function \texttt{DrawArrow}(X,Y,W,L,theta) which draws an arrow that points to location \((X,Y)\), has a width \(W\), a length \(L\), and the whole arrow is at an angle \texttt{theta}. Assume OpenGL is being used as the rendering API. The function call should leave the current transformation matrix unchanged upon exit. You should use a generic version of the arrow which you then scale appropriately. The result of \texttt{DrawArrow}(1,2,2,4,90) would look as shown below.
4. Projections

(a) (2 points) Sketch a row of three adjacent cubes in two-point perspective.

(b) (2 points) An example of a cabinet projection of a cube having dimensions $D \times D \times D$ is shown below. All edges parallel to the $x$ or $y$ axes are drawn to scale, while edges parallel to the $z$ axis are drawn at half scale, i.e., half of their true length, as indicated by the $D/2$ screen-space measurement shown below. Develop a transformation matrix that can be used to produce the cabinet projection shown below. Points on the back of the cube should be left unchanged by the transformation because they already sit on the $xy$-plane.
(c) (2 points) Develop the transformation matrix that would produce the isometric rendering of the cube shown below. Thus, given a point on the cube, say (1,1,1), the matrix will compute the corresponding image-space point (x',y'). Show your work. The point (0,0,0) on the cube should be unaltered by the transformation to screen space, as shown in the figure. You may choose to express your answer as a sequence of rotations.
5. (2 points) Hierarchical Scene Diagrams

The following scene graph illustrates what parts are “attached to” what other parts; the transformations to move between the various coordinate systems are labelled. Thus $M_{rh}$ takes a point from the coordinate system of the right hand to the coordinate system of the right upper arm.

Give the compound transformation matrix that is necessary to take a point expressed in the “right hand” coordinate system to the camera coordinate system.

Also given the compound transformation matrix to express a point given in the “right hand” coordinate system in terms of the “left hand” coordinate system.

![Hierarchical Scene Diagram]

6. (2 points) Rotation Matrix

Show that the axis of rotation is an eigenvector of the transformation matrix that implements the corresponding rotation.
7. Coding

In this question, you will be animating a bird that will walk across the screen and jump around in any way you like. The following is a suggested ordering of steps.

(a) (0 points) Download and compile the template code:

```
http://www.ugrad.cs.ubc.ca/~cs414/Vjan2004/a1/a1.tar.gz
mkdir assn1
cd assn1
gunzip a1.tar.gz
tar xvf a1.tar
make
```

(b) (1 point) Add glut key bindings such that the ESC key causes the program to exit.

(c) (3 points) Create a function that draws a “tapered cylinder” as a function of the 4 parameters shown below. `nslices` will be the number of polygons used to represent the tapered cylinder, which can be considered to be open on either end. You will be using it as a modeling primitive for some of the body parts of your bird.

You will be computing all the vertex locations. First use `GL_WIRE_LOOP` in order to draw the tapered cylinder in wire frame. Once this works, compute a surface normal for each vertex and call `glNormal3f()` before each `glVertex3f()` call in order to assign a correct surface normal to each vertex, and change to using `GL_POLYGON` or `GL_QUADS` thus producing a solid-shaded tapered cylinder.

(d) (3 points) Build your articulated figure. You should make use of your tapered cylinder at least once in your character. Use an appropriate hierarchy of transformations. Implement it using appropriately structured code. You may want to define separate geometry for each type of link, or you may want to use scaled versions of the various `glutSolidX` primitives, where X can be any of `Sphere`, `Cube`, `Cylinder`, `Pyramid`, etc.
Cone, Torus, or Teapot. Draw your bird in an appropriate “rest pose” — typically this is a simple standing pose. Make your bird colourful.

(e) (2 points) Add the ability of your character to assume different poses by adding joints to all the links. To keep things simple to start with, assume that each joint only has one degree of freedom, so that you can see all of the possible joint motions in a side view. Then add additional joints that rotate about other axes if you like.

(f) (3 points) Add the capability to have your character be displayed in different poses, each pose being defined in a file, keyframe.txt. This file will consist of a specification of all of the “degrees of freedom of the character”. Each line in this file should have a format something like the following — this is just an example:

keyframe 6.34 2.0 1.0 10 20 10 10 -20 10 10 40 50 -40 50 10 20

where:

parameter 1:  time stamp (ignore this for now)
parameter 2:  body location, x
parameter 3:  body location, y
parameter 4:  body lean
parameter 5:  right hip angle
parameter 6:  right knee angle
parameter 7:  right ankle angle
parameter 8:  left hip angle
parameter 9:  left knee angle
parameter 10: left ankle angle
parameter 11: left wing angle
parameter 12: right wing angle
parameter 13: neck forward bend
parameter 14: head forward bend
parameter 15: head twist
parameter 16: head roll

Add a keybinding such that hitting the 'k' key reads in the next line of the keyframe file and displays the next pose. Once all poses are displayed, cycle back to the first pose again.

(g) (3 points) Now you will be animating the figure by using linear interpolation between the keyframes in order to produce ‘in-between’ poses of your character. Choose a fixed time step and use the glutIdle callback to advance to the next frame. At first, debug this by simply continuing to display the current pose until it is time to display the next pose. Next, implement the linear interpolation in order to compute an updated pose for each time step. Create a keybinding that lets the spacebar start and stop the animation. When you get to the end of the animation, make it loop back to the beginning. You will need to experiment in order to set reasonable keyframes (animation isn’t easy!). Set up a key binding so that the spacebar starts and stops your animation.

(h) (5 points) To improve your animation you can have your character perform a variety
of motions, populate the environment with interesting objects, generate an improved model for the body, etc. The marking here will be necessarily subjective. There will be prizes for the best three animations and they will be entered into the 414 hall of fame.

Hand-in Instructions

- Hand in a printed copy of your code and a README file.
- Create a root directory for our course in your account, called cs414. Later all the assignment handin files should be put in this directory.
- For assignment 1, create a folder called assn1 under cs414 and put all the source files that you want to handin in it, including your "makefile" and your README file. Don’t use subdirectories – these will be deleted. NOTE: we only accept README, makefile and files ending in cpp, c, and txt.
- In your README file, please describe how to run your program and what functionalities you have implemented, as well as any kind of information you would like to give us for getting credit for partial implementation. If you don’t complete all the requirements, please state clearly to what you have tried, what problems you are having and what you think might be promising solutions.
- The assignment should be handed in with the exact command:

  handin cs414 assn1

This will handin your entire assn1 directory tree by making a copy of your assn1 directory, and deleting all subdirectories! (If you want to know more about this handin command, use: man handin)