

CPSC 314

Assignment 1

Due 4pm, Friday October, 15 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: _____

Theory	/ 20
Question 1	/ 0
Question 2	/ 4
Question 3	/ 2
Question 4	/ 2
Question 5	/ 2
Question 6	/ 4
Question 7	/ 6
Question 8 (bonus)	/ 3
Programming Assignment	/ 80
TOTAL	/ 100

3. (2 points) Given a triangle $T = (P_0, P_1, P_2)$ in 2D and a point P , write an algorithm to find if the point is inside or outside the triangle.

4. (2 points) What will the following transformations in 3D homogenous coordinates do to a unit cube centered at the origin (use row vectors)? Sketch your answers.

(a) (1 point)

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

(b) (1 point)

$$\begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. (2 points) Write a transformation that mirrors through arbitrary line $Ax + By + C = 0$ in 2D. (Hint: Break it down to sub problems.)
6. (a) (2 points) Given a convex planar polygon $P = P_1, P_2, \dots, P_n$, describe an algorithm for triangulating the polygon (triangulate = split into triangles) without adding any extra vertices.
- (b) (2 points) Will your algorithm work for non-convex polygons? If yes, prove. If not, bring a counter-example.

7. (6 points) Answer yes/no (no explanation). All the transformations are in 3D.

(a) (1 point) Does Perspective preserves parallel lines?

(b) (1 point) Does Perspective Warp preserves angles?

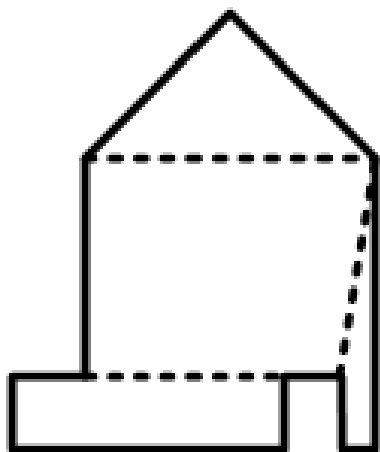
(c) (1 point) Is there an α for which the Perspective Warp becomes a Perspective Projection?

(d) (1 point) Is shear * rotate = rotate * shear?

(e) (1 point) Is rotate1 * rotate2 = rotate2 * rotate1?

(f) (1 point) Does shear preserve parallel lines?

8. (3 points) BONUS: Given a non-convex planar polygon $P = P_1, P_2, \dots, P_n$, describe an algorithm for splitting the polygon into convex pieces, without adding any extra vertices. Example:



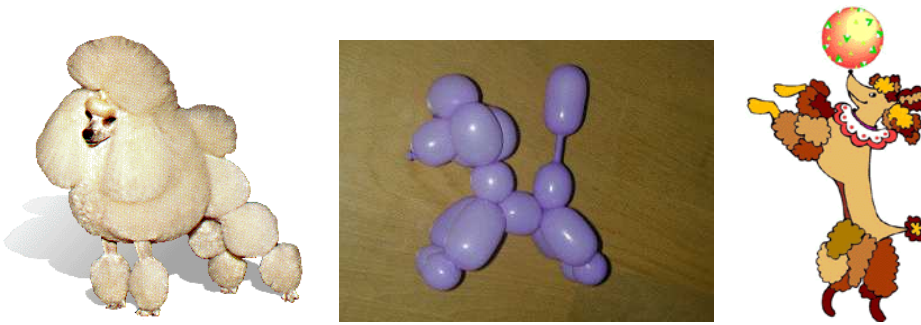


Figure 1: Poodles.

9. Coding

In this programming assignment, you will be animating a poodle dog (Figure 1). Your poodle will stand, sit, wag its tail, beg and jump/dance 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/~cs314/Vsep2004/a1/a1.tar.gz
mkdir assn1
cd assn1
gunzip a1.tar.gz
tar xvf a1.tar
make

```

- (b) (1 point) Add glut key binding such that the ESC key causes the program to exit.
- (c) (1 point) Modify the template main to accept 0 or 1 command line arguments. You will be using the command line argument starting from step (h) of your implementation. Hence you need to pass it to the appropriate method or class.
- (d) (8 points) Create a function that draws a “tapered cylinder” as a function of 4 parameters: two radii, height, and the number of polygons used to represent the tapered cylinder. The cylinder should be closed at both ends. You will be using it as a modeling primitive for some of the body parts of your dog.

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.

- (e) (5 points) Create a `drawEllipsoid()` functions that draws an ellipsoid as a function of 3 radii. Remember that you can do nonuniform scaling to create ellipsoids from spheres.
- (f) (12 points) Build your articulated figure. You should create the entire model from ellipsoids and tapered cylinders (you must use each primitive at least once). Use your own functions to model those. Do not use any glut or GL geometric primitives.

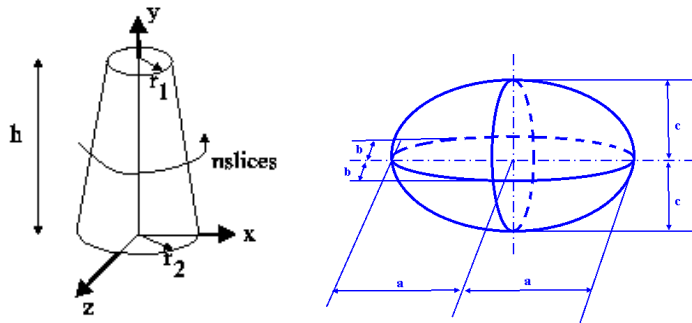


Figure 2: Tapered cylinder and ellipsoid.

Use an appropriate hierarchy of transformations. Implement it using appropriately structured code. Draw your dog in an appropriate “rest pose” — typically this is a simple standing pose. Make your dog colorful.

- (g) (10 points) Add joints to all the links to enable your dog to assume different poses. 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.
- (h) (12 points) Add the capability to define the poses of your character using a script file. The name of the script file (e.g. “stand.txt”) to use should be passed as an argument to your program. If no file is given, the model should be displayed in a rest pose. This file will consist of several pose specifications (keyframes). Each specification should contain all of the “degrees of freedom of the character”. Each line in this file should represent a pose (keyframe). A line format should look similar to the example below (the actual parameters depend on your particular model structure):

```
keyframe 6.34 2.0 1.0 10 20 10 10 -20 10 10 10 20 0 20 10 -10 50 -40 50 10
```

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 front hip angle
parameter 6:  right front knee angle
parameter 7:  right front ankle angle
parameter 8:  left front hip angle
parameter 9:  left front knee angle
parameter 10: left front ankle angle
parameter 11: right back hip angle
parameter 12: right back knee angle
parameter 13: right back ankle angle
parameter 14: left back hip angle
```

```
parameter 15: left back knee angle
parameter 16: left back ankle angle
parameter 17: head forward bend
parameter 18: head twist
parameter 19: tail link 1 angle
parameter 20: tail link 2 angle
```

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

- (i) (6 points) Create script files for the following poses (each file containing only one pose):
 - (3 points) sit,
 - (3 points) stand on back legs.
- (j) (16 points) Now you will be animating the figure by using linear interpolation between the keyframes (poses) 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. 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!). Setup a key binding so that the spacebar starts and stops your animation.
- (k) (9 points) Create keyframe animation (appropriate script files) for the following motions:
 - (3 points) Wag the dog's tail and shake the head.
 - (3 points) Sit-down: go from "rest" (standing) pose to sitting one.
 - (3 points) "Beg": Jump or dance on back legs.
- (l) (4 points) Bonus: 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.
Extra BONUS points The best three animations will get extra bonus points (10 for first, 8 for second, and 5 for third) and they will be entered into the 314 hall of fame.

Hand-in Instructions

- Hand in a printed copy of your code and a README file.
- Hand in all the keyframe files you generated. Document in the README file what each keyframe file does. Note that all the keyframe files MUST have a .txt ending.
- Create a root directory for our course in your account, called cs314. Later all the assignment handin files should be put in this directory.
- For assignment 1, create a folder called assn1 under cs314 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, h, 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 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 cs314 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)

Programming Assignment Grading

The programming part of this assignment will be graded with face-to-face demos: you will demo your program for the TA. If need be, you can concisely summarize the arguments in your README about incomplete work. You can also explain any extra credit features you implemented.

- We will circulate a signup sheet for demo slots in class. Each slot will be 10 minutes. The demo sessions will be Friday Oct 15, Monday Oct 18, and Tuesday Oct 19 (except during scheduled lab/lecture times).
- You must ensure that your program compiles and runs on the lab machines. If you worked on this assignment elsewhere, it is your responsibility to test it in the lab. The face to face grading time slots are short, you will not have time to do any 'quick fixes'! If your code as handed in does not run during the grading session, you will fail the assignment.
- Bring a printed copy of your code and README file to the face-to-face grading session to give to the TA. This printout must match exactly what you submitted electronically. The code that you demo must match exactly what you submitted electronically: you will show the TA a long listing of the files that you're using, so that he can quickly verify that the file timestamps are before the submission deadline.

- Arrive at CICSR 011 at least 10 minutes before your scheduled session. Log into a machine, and double-check that your code compiles and runs properly. Then delete the executable.
- When the TA comes to your computer. you will type the following:

```
ls -l  
make
```

You will then run your assignment with the 3 required animation scenarios:

- Wag the dog's tail and shake the head.
`a1 wag.txt`
- Sit-down: go from "rest" (standing) pose to sitting one.
`a1 sitdown.txt`
- "Beg": Jump or dance on back legs.
`a1 beg.txt`