

# CPSC 314

## Assignment 1

Due 4pm, Friday October, 14 2004  
(handin box in the CICSR basement)

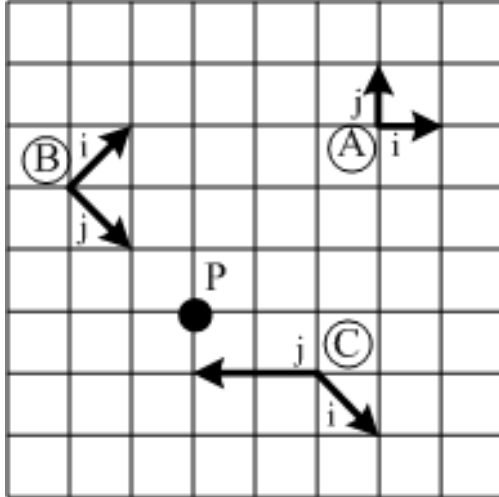
Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Name: \_\_\_\_\_

Student Number: \_\_\_\_\_

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

1. (3 points) Transformation as a Change of Coordinate Frame



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 the coordinates of  $P$  with respect to the different frames (see your answers in assignment 0).

2. (3 points) Given a line segment  $S = (P_0, P_1)$  in 2D and a point  $P$ , write an algorithm to find if the point is on the line segment. Note that your algorithm should operate in the Euclidean (not discrete) space.

3. (4 points) Decompose the following complex transformations in homogeneous coordinates into a product of simple transformations (scaling, rotation, translation, shear). Pay attention to the order of transformations.

(a) (1 point)

$$\begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

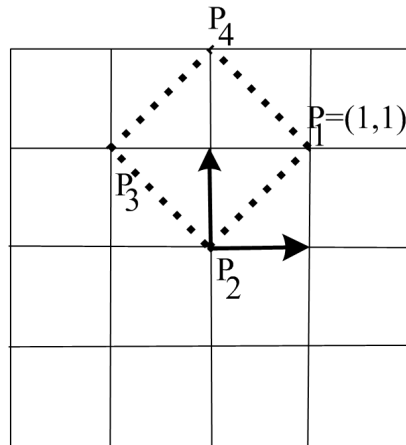
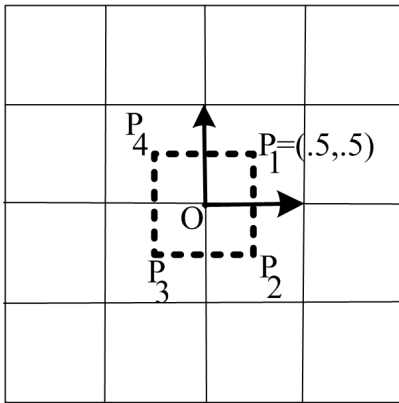
(b) (1 point)

$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(c) (1 point) What is the inverse of the transformation matrix in part (b) of this question?

(d) (1 point) Give the sequence of OpenGL transformations that would produce the same transformation matrix as in part (a) of this question.

4. (3 points) Write down the 2D transformation matrix that maps the unit square centered at the origin as shown on the left to the square on the right in the figure below. Show your work.



5. (4 points) Answer yes/no and provide a short explanation. All the transformations are in 3D.

(a) (1 point) Does perspective transformation preserve parallel lines?

(b) (1 point) Is shear \* translate = translate \* shear?

(c) (1 point) Is  $\text{shear1} * \text{shear2} = \text{shear2} * \text{shear1}$ ?

(d) (1 point) Does shear preserve lengths?

6. (3 points) Given the triangle  $T$  with vertices  $P_1 = (0, 0, 0)$ ,  $P_2 = (2, 0, 0)$ ,  $P_3 = (1, 1, 0)$  and the transformation

$$S = \begin{pmatrix} 1 & -0.5 & 0 & 0 \\ 0.5 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(a) (2 points) Compute the vertices of the triangle after applying the transformation  $S$  to it.

(b) (1 point) Compute the normal of the triangle before and after applying the transformation  $S$  to it.

7. (4 points) BONUS: Given a non-convex planar polygon  $P = P_1, P_2, \dots, P_n$  and a point  $P$  in 2D describe an algorithm for testing if the point is inside/outside the polygon. Try to make your algorithm as efficient as possible.



Figure 1: Dinosaurs.

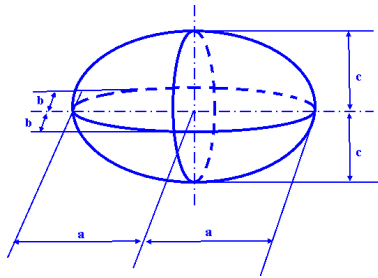
## 8. Coding

In this programming assignment you will be modeling and animating a dinosaur (Figure 1). You should use hierarchical transformations as described in class when modeling and animating your dinosaur. You will animate your dinosaur, making it turn its head, move its tail, and do other motions. The following is a suggested ordering of steps.

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

```
http://www.ugrad.cs.ubc.ca/~cs314/Vsep2005/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 (f) of your implementation. Hence you need to pass it to the appropriate method or class.
- (d) (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.



- (e) (20 points) Build your articulated figure. You should create the entire model from ellipsoids constructed using your `drawEllipsoid` function. Do not use any `glut` or `GL` geometric primitives. Use an appropriate hierarchy of transformations. Implement it using appropriately structured code. Draw your dino in a “rest pose” — typically this is a simple pose of standing on all four legs with the head up and the tail down (Figure 1, dinos at the bottom). Start with a simple model with a small number of links for basic components (head, neck, body, tail, legs). Use multiple ellipsoids for neck and tail and two or more for each leg. Add extra links as you wish to make your dino more interesting.
- (f) (15 points) Add joints to all the links to enable your dinosaur 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.
- (g) (9 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 2.0 1.0 10 20 10 10 -20 10 10 10 20 0 20 10 -10 50 -40 0 0 50 10
```

where:

```
parameter 1:  body location, x
parameter 2:  body location, y
parameter 3:  body lean
parameter 4:  right front hip angle
parameter 5:  right front knee angle
parameter 6:  right front ankle angle
parameter 7:  left front hip angle
parameter 8:  left front knee angle
parameter 9:  left front ankle angle
parameter 10: right back hip angle
parameter 11: right back knee angle
parameter 12: right back ankle angle
parameter 13: left back hip angle
```

```
parameter 14: left back knee angle
parameter 15: left back ankle angle
parameter 16: neck1 forward bend
parameter 17: neck1 twist
parameter 18: neck2 forward bend
parameter 19: neck2 twist
parameter 20: tail link 1 angle
parameter 21: 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.

- (h) (6 points) Create script files for the following poses (each file containing only one pose):
- (2 points) turn head and tail to the right
  - (4 points) stand on back legs (Figure 1, yellow and pink dinos at the back).
- (i) (15 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.
- (j) (8 points) Create keyframe animation (appropriate script files) for the following motions:
- (4 points) Swipe the tail from side to side (curl it to one side then the other) and bend the neck up and down.
  - (4 points) Stand-up: go from "rest" (standing on 4 legs) pose to stand (standing on 2 back legs and the tail).
- (k) (5 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.
- Use the root directory cs314 that you created for assignment 0. 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 on Friday Oct 14, Monday Oct 17, and Tuesday Oct 18 (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 2 required animation scenarios:

- Move the tail and the head  
`a1 move.txt`
- Stand-up go from “rest” pose to standing one.  
`a1 standup.txt`

If you are shooting for the bonus points, show the grader the extra features and/or scenarios you created.