CPSC 314, Programming Project 1: Articulated Giraffe

Out: Thu 12 May 2005. Due: Wed 18 May 2005, 11:59pm. Value: 9% of final grade

In this project, you will create an articulated giraffe. There are three required parts (100 points), and up to 5 extra credit points can be earned. The best work will be posted on the course web site in the Hall of Fame.

Modelling: [35 points total] Model a giraffe out of transformed cubes, using only $4x4$ matrices to position and deform them.

- (1 pt) Create a drawCube() function that draws a unit cube. This is the only place in your program that you should call OpenGL geometry commands. You may use either glutSolidCube or create your own out of six square faces.
- (34 pts) Model your giraffe out of transformed cubes. Remember that you can do nonuniform scaling to create long skinny boxes. You should orient your giraffe so that you see a side view from the default camera position. You should create your giraffe using a hierarchical scene graph structure. That is, instead of just using an absolute tranformation from the world origin for each individual part, you should use a relative transformation between an object and its parent in the hierarchy. For example, the head should be placed relative to the neck coordinate system. This hierarchical structure will make both modelling and animation much easier in the long run, even if it might seem like extra work at first! Your giraffe should have at least the following parts:
  - (3 pts each, 12 total) Body, Tail, Head, Ears.
  - (3 pts each, 12 total) 4 Legs: upper leg, lower leg, hoof
  - (10 pts) Neck: should make out of several segments, so it can curl.
  - (extra credit: up to 2 pts) Add color, and/or add more detail to your giraffe, such as eyes or horns, etc. And/or make an interesting environment for your giraffe, again using only cubes and $4x4$ matrices.

Animation: [55 points total] Animate the joints of your giraffe. Specifically

- (5 pts) Head/neck turn: Turn neck/head section wrt body so giraffe looks sideways.
- (5 pts) Tail wag: Move the tail vertically from down to up.
- (8 pts) Single leg raise: Rotate the upper leg up with respect to the body (the lower leg and hoof should of course move with it), rotate the lower leg with respect to the upper leg.
- (12 pts) Neck curl: Curl down the neck, with each segment moving incrementally with respect to the previous one.
- (10 pts) Rear: Have the giraffe stand on its hind legs; that is, rotate the hind legs in the opposite direction from the single leg raise. At the same time, have both the front legs raised up using the 'single leg raise' functionality that you implemented above. When you hit the 'rear' toggle key again in jumpcut mode, the giraffe should return to the rest position. If the user hits the key to make the giraffe rear when it is not in the rest position: if the front legs are already raised, they should stay raised; if the rear legs are already raised, they should return to the rest position.
- (15 pts) Smooth transitions: Show the motion as smoothly animated transition instead of a jumpcut. That is, draw many frames in succession where each movement is small. You should linearly interpolate between the old joint angle and the new one. You should request a redisplay event from GLUT between each step rather than taking control away from the main event-handler by doing the transition in your own loop. For full credit, you should properly handle the case where a second transition begins while another transition is already happening.
- (extra credit: up to 3 pts) Add more motions. For instance, an ear wiggle where the ears do some interesting and vaguely physically plausible rotation(s), or a more complex tail wag, or having your giraffe drop down to be on all fours, or having your giraffe jump forward, or walk forward. Or having the giraffe strike a kung-fu pose. Or new camera positions or trajectories. Or whatever strikes your fancy.

Interaction: [10 points total] Interactively control your giraffe.

- (10 pts) Keys: Add the following GLUT key bindings for the animation: 'h' for head turn, 't' for tail wag, 'c' for neck curl, 'r' for rear, 'l' for front left leg raise, 'm' for front right leg raise, 'n' for rear left leg raise, 'o' for rear right leg raise. These keys should act as toggles: when the user hits the key, move from rest position to new position, or from the new position back to the rest position. Add the binding of the 'q' key for a clean exit of the program. Have the spacebar toggle between jumpcut mode and smooth transition mode. In transition mode, hitting a key should cause a single smooth transition: either from the rest position to the new position, or from the new position to the rest position the next time the key is hit. If you create extra credit motions, pick unused letters to trigger them and document these in your writeup.

Suggested Strategy You should build your giraffe in a 'rest' pose, because you will be moving the joints as below. Consider the rest pose a starting place where you define the rotation angle of each joint to be 0, and for the animations below you will be changing that angle. You might find it easier to debug your code if you use a separate transformation for the joint animation than the one you use for the modelling.
Clearly, there are dependencies here: if you don't model a tail, you can't get credit for tail animation. You should definitely interleave the modelling, animation, and interaction. If you don't interleave the modelling and animation, you might spend a lot of time creating an animal with the wrong kind of hierarchical structure to move correctly. Start by placing an upper leg segment with respect to the body, and then immediately implement the jump-cut animation of that leg. After the upper leg is working correctly, then model the lower leg placement, and then do the lower leg animation. Finally, add the hoof. After all that seems to work correctly, then instantiate the other three legs. Then move on to do other body parts.
While you're debugging, don't forget to try moving the camera as described below to check whether things are placed correctly. Sometimes a view from one side can look right, but you can see from the top or front that it's in the wrong spot along one of the other axes. You can get far with 3 camera placements: default side, top, and front.
Do smooth transitions after you've done all the body parts. Do not do transitions with a loop where you take control away from the standard event-handling code; instead, use GLUT idle functionality to request that the display function is called to work within the event handling architecture. Your program should redraw exactly and only when necessary: when nothing is moving, the view should not be continually redrawing. In response to a keystroke trigger a jumpcut change, the program should redraw once. When a smooth transition is triggered, the view should redraw many times until the transition has ended, and then redrawing should stop.
Finish doing the entire required functionality before starting on any extra credit.

Template Download from `http://www.ugrad.cs.ubc.ca/~cs314/Vmay2005/proj1.tar`, which contains the two files `p1.cpp` and `Makefile`. To unpack the file, type `tar xvf proj1.tar`
The template code allows you to change the viewpoint to look at the central object from any of six directions. Consider the giraffe to be at the center of a cube. In the default you're looking at it from one face of the cube, for a side view. You can move the camera so that you can see the giraffe from any of the other 5 faces of the cube: other side, front, back, over, under. Trigger this action with the 's', 'f', 'b', 'o', 'u', respectively. The 'r' key resets to the original view. You should construct your giraffe so that the default view is indeed the side view.

Documentation Your README file should include your name, student number, and username. State what functionality you have successfully implemented, as well as any information you would like to give us for getting credit for partial implementation. If you did extra credit work, say what you did and how many extra credit points you think the work is worth. Please be clear and concise.
You must include at least two images of your giraffe, front and side views are a good choice. We'll post the best of these giraffe images in the Hall of Fame. On the Linux boxes in the lab, the easiest way to take snapshots is with the ImageMagick `import` utility. Just type `import myfile.png` and click on your program's window with the resulting X-shaped cursor. You can look at your images with the `display` command. If you want to show off your giraffe in action through a loop of several images, you can make animated GIFs easily using the `convert` command, as described at `http://www.tjhsst.edu/~dhyatt/supercomp/n401a.html`.

Handin Create a root directory for our course in your account, called cs314. All the assignment handin subdirectories should be put in this directory. For project 1, create a subdirectory of cs314 called proj1 and copy to there all the files you want to hand in, including your source, makefile, README. Do not use subdirectories, these will be deleted. We only accept README, makefile, source files ending in .cpp and .h, and image files ending in .png or .jpg or .gif. The assignment should be handed in with the exact command: `handin cs314 proj1`. This will handin your entire proj1 directory tree by making a copy of your proj1 directory. Note that any subdirectories in that directory will be deleted! For more information about the `handin` command, see `man handin`.

Grading Your grade will be partially based on correct performance and partially on clean coding practices (probably a 70%/30% split). Note that global variables are necessary in event-driven programming, we do not consider that to be a problem. We are looking for reasonable modular structure so that parameterized functions do similar things rather than a lot of a lot of cut-and-paste code with slight alterations to handle different cases.
This project will be graded with face-to-face demos: you will demo your program for the grader. We will circulate a signup sheet for demo slots in class. Each slot will be 10 minutes. For the first part of your slot, you will demo your program to the grader. Then, the grader will briefly discuss the code with you. Finally, the grader will spend some time alone, looking at the code further and writing up notes.
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 README file to the face-to-face grading session to give to the grader. The code that you demo match exactly what you submitted electronically: you will show the grader a long listing of the files that you're using, to 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 grader comes to your computer, type `ls -l; make; p1`.
You should not change your code after submitting it, you should have ensured your code runs correctly on the lab machines before submitting it. If you did not plan ahead and suddenly discover some critical reason to change your code, for instance that you need to change a paramter to get your smooth transitions looking reasonable because the lab computer is much faster or slower than your home machine, then copy the submitted code to a new file and change only that new file. Then the TA can quickly verify that you only made a trivial change by running "diff" between the two files. Do not just edit the original file!