CPSC 314, Programming Project 1: Articulated Armadillo

Out: Mon 22 Jan 2007. Due: Fri 2 Feb 2007, 5:59pm. Value: 8% of final grade

In this project, you will create an articulated armadillo. 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, and shown in class.

Modelling: [35 points total] Model a armadillo 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 armadillo out of transformed cubes. Remember that you can do nonuniform scaling to create long skinny boxes. You should orient your armadillo so that you see a side view from the default camera position. You should create your armadillo 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 body coordinate system and the ear should be placed relative to the head. This hierarchical structure is absolutely critical in the long run for both modelling and animation, even if it might seem like extra work at first! Your armadillo should have at least the following parts:
  - (2 pts each, 6 total) Front Body, Middle Body, Rear Body
  - (2 pts each, 6 total) Head, Snout, Ears
  - (3 pts each, 12 total) 4 Legs: Leg, Paw, 5 Claws
  - (10 pts) Tail: should make out of several segments, so it can curl.
  - (extra credit: up to 2 pts) Add color, and/or add more geometric detail to your armadillo, such as eyes, or more bands in the body middle. And/or make an interesting environment for your armadillo, again using only cubes and 4x4 matrices.

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

- (4 pts) Head nod: Move the head down vertically with respect to the upper body, the ears and snout should of course move with it.
- (4 pts) Rear leg raise: Rotate the rear leg up and forward with respect to the body, the paw and claws should of course move with it.
- (8 pts) Front leg curl: Rotate the front leg back toward the inside of the body. Rotate the paw with respect to the leg even more toward the body center.
- (4 pts) Upper body turn: Turn front body with respect to the middle body so armadillo looks sideways.
- (12 pts) Tail curl: Curl the tail down, with each segment moving incrementally with respect to the previous one.
- (8 pts) Body curl: the front and rear body segments should rotate inwards with respect to the middle body segment. The head nods down. The tail curls down. The rear legs raise. The front legs curl.
- (15 pts) Smooth transitions: Show all 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. You must also ensure that if your program does not redraw unnecessarily and burn CPU cycles when transitions are not 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 have your armadillo jump in the air, or walk forward, or dance, or do kung-fu! Or new camera positions or trajectories. Whatever strikes your fancy.

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

- (10 pts) Keys: Add the following GLUT key bindings for the animation: 'h' for head nod, 't' for tail curl, 'c' for body curl, 'p' for upper body turn, '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. 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 README writeup.

Suggested Strategy

- You should build your armadillo in a 'rest' standing pose. Consider the rest pose a starting place where you define the rotation angle of each joint to be 0, and to move the joints for the animation 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 three 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.

Documentation

- **README (required)**: Your README file should include your name, student number, and username. Your README must also include this statement:

  > By submitting this file, I hereby declare that I worked individually on this assignment and that I am the only author of this code. I have listed all external resoures (web pages, books) used below. I have listed all people with whom I have had significant discussions about the project below.

  You do not need to list the course web pages, textbooks, the template code from the course web site, or discussions with the TAs. Do list everything else, as directed at in the collaboration/citation policy for this course at

  `http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007/policies.html#plag`

  Including this statement in your README is your official declaration that you have read and understood this policy.

  In your README, state what functionality you have successfully implemented. 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. 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.
- **2 images (required)**: You must include at least two images of your armadillo, front and side views are a good choice. We'll post the best of these armadillo images in the Hall of Fame. The template code allows you to save frames as PPM image files. You can browse the images that you created with the `display` command. You may submit some extra images.

- **movie (optional)**: You may include a movie, especially nice if you'd like to show off a cool extra-credit animation of your armadillo in action in the Hall of Fame.

  You can make animated GIFs easily using the `convert` command: `convert -delay 20 -loop 0 img*.ppm anim.gif`

  You can make an MPEG animation file (uses much less space, but won't play directly on a web page) using the `ffmpeg` command: `ffmpeg -i img%03d.ppm -r 24 a1.mp4`

  and play your movie using `ffplay a1.mp4`

  If you will dump a large number of images, consuming a lot of disk space, create a temporary directory `/tmp/foo` where `foo` is your user name. Edit the appropriate line in the dumpPPM() function call in order to ensure that image files get written to this directory. Hitting d when running your code in animate mode will results in a large number of PPM files being written to this directory of the form imgNNN.ppm, where NNN is the frame number. Dont forget to delete all your PPM files once your movie has been created to save some disk space. If you want to keep your movies reasonably small in size, use a small window when dumping your frames. For example, a 500 x 300 video will use only 25% the disk space of a 1000 x 600 video.

Template Download from `http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007/proj1.tar` and use this command to unpack it: `tar xvf proj1.tar` You will see two files, `p1.cpp` and `Makefile`.
The template code allows you to change the viewpoint to look at the central object from any of six directions. Consider the armadillo 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 armadillo from any of the other 5 faces of the cube: other side, front, back, over, under. Trigger this action with the 's', 'f', 'b', 'a', 'u', respectively. The 'r' key resets to the original view. You should construct your armadillo so that the default view is indeed the side view. The 'q' key exits the program. Each time you hit the 'i' key a new image is saved in the ppm directory. When you hit 'd', the image counter will be reset and program will dump out a new image at each redraw until you hit 'd' again to stop the dump. Hitting any key will trigger a redraw.

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 `p1` and copy to there all the files you want to hand in: README, makefile, source files ending in `.cpp` and `.h`, image files ending in `.png` or `.jpg` or `.gif`, and movie files ending in `.mp4`. Do not use subdirectories, these will be deleted. The assignment should be handed in with the exact command: `handin cs314 p1`. For more information about the `handin` command, see `man handin`.
You do not need to hand in any hardcopy printouts.

Grading All of the above breakdown of marks was based on correct performance. You also need to produce clean code, you will lose marks for poor style up to a maximum of 15% of the assignment grade. The most important style issue is to have reasonable modular structure. For example, parameterized functions should do similar things rather than a lot of cut-and-paste code with slight alterations to handle different cases. Note that global variables are necessary in event-driven programming, we do not consider them to be a style problem. Your code should be readable, with well-chosen variable and function names and enough comments to explain what is happening. Your rule of thumb for comments should be: what somebody has to fix a bug in this code two years from now? Your comments should help that person understand the structure of the code quickly, and explain anything tricky or non-obvious.
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 a 10-minute demo slot in class. You should be logged into a 011 lab machine at least 10 minutes before your scheduled session. For the first part of your slot, you will list the files, compile, and demo your program for 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 before submitting the assignment 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. Plan ahead: ensure your code runs correctly on the lab machines before submitting it, both in terms of compile/run, and parameter settings for animations so that your transitions look good (the lab computers may be slower or faster than your home machine). The face to face grading time slots are short, you will not have time to do any quick fixes. If, nevertheless, you somehow discover some critical problem at the last minute, do **not** just edit the original file! Instead, copy the submitted code to a new file and change only that new file. Then the grader can quickly verify that you only made a trivial change by running `diff` to compare the two files.