COMP 557 - Fall 2020 - Assignment 1
Transform Hierarchy for an Animated Character

Available Tuesday 15 September

Due 23:30 Tuesday 6 October

Weight 15%

Getting Started

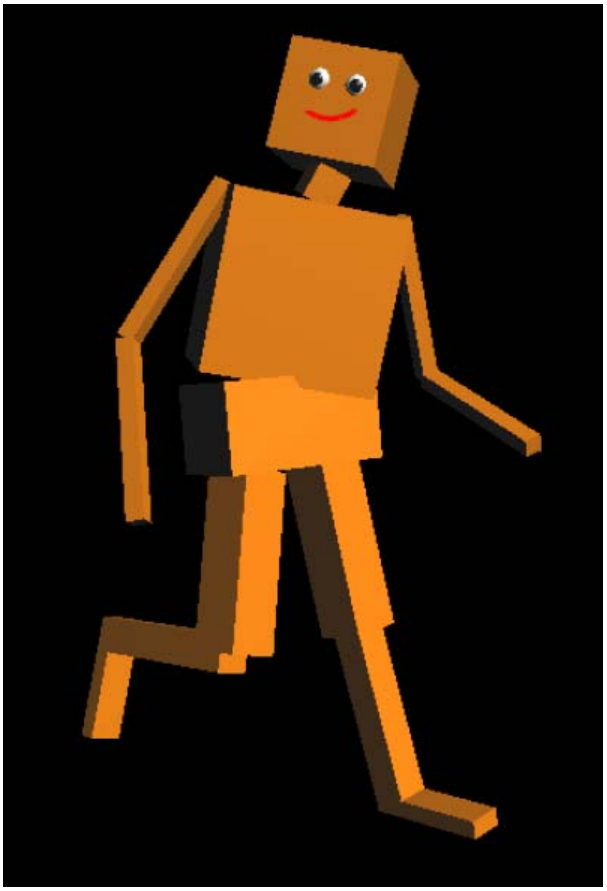
Download the provided code from McCourses and import it as a new java project into your workspace. You should have everything in the provided jars folder in the build path, that is, the jogl fat jar as well as the *vecmath* and *mintools* jars. Recall, the vecmath jar is useful for simple vector mathematics, and the mintools jar is a collection of classes useful for building simple swing interfaces. There are three main parts of the mintools package:

- parameters contains boolean and double parameter classes for quickly setting values with swing interface controls.
- swing contains layout helpers and panels for quickly throwing together a Java swing interface.
- viewer currently only contains a helper class for drawing bitmap fonts into your opengl window.

The source for the vecmath and mintools is provided for your convenience, and for the javadoc.

Do not change any of the package names as this will interfere with marking.

Objectives



The purpose of this assignment is to use a hierarchy of transformations to make an articulated character. You must develop a hierarchy which is useful for posing the character, and ultimately create a short key frame animation. Other objectives include creating simple geometry, implmenting a basic matrix stack, and GLSL lighting of your geometry.

Note that the BasicPipeline only has the most basic viewing and projection transformations set to start. If your character is roughly a unit or so tall, and centered at the origin, it should be visible.

Your character should be collection of rigid solid objects "connected" by various parametric joints, such as single axis rotary joints, or 3 axis spherical joints. There is a lot of room for creativity in making a unique character with a given personality or appearance. Have fun! Note that to receive full marks your character should have a minimum level of complexity with appropriate degrees of freedom and limits!

It will be useful to plan ahead by sketching on paper the skeleton of your character, to think about what kinds of transforms and geometry you will need.

Some geometry drawing code is provided to help you draw the different body parts of your character, but others must be created as objectives. While the basic geometry drawing code draws, for instance, a unit radius sphere, you can use non-uniform scaling and translations to make shapes appropriate for your character.

Because you control of the GLSL lighting computations, you will be able to set different material colours for different objects. See the basic vertex and fragment programs that are provided as stubs and inspiration for

creating your scene.

1. ***Matrix Stack (1 marks)***

   The BasicPipeline class sets up a basic vertex and fragment program, and looks up the IDs of various input parameters. The class also defines push and pop methods for a matrix stack. Create a matrix stack implementation, and consider making it efficient (i.e., preallocating matrices for a given stack depth, e.g., 32 or 64 so that it is fast and sufficient for most characters).

2. ***Matrix Transformations (1 marks)***

   The BasicPipeline class also has convenience methods called translate, rotate, and scale to allow you to modify the modeling matrix at the top of the stack. The normal action is to concatenate the the transform to the current transform chain by right multiplication. See rotate as an example, and note that you must implement translate and scale.

3. ***FreeJoint DAG Node (1 marks)***

   Finish implementing the *FreeJoint* class extending GraphNode. This will be for the root of your character allowing you can modify both translation and rotation of the root. You are extending the abstract *GraphNode* class from which you can create additional nodes in your scene graph. Examine the code in the provided abstract class, and note that you are extending this class to implement this objective as well as several other objectives below. In particular, note calling super.display from the FreeJoint will run a for loop that will call the display of all children nodes.

   There are 6 DoubleParameter member variables in the class for controlling the 6 degrees of rigid motion. You may need to adjust the min and max values for each parameter (e.g., the translation parameters should have reasonable limits based on the size of your character. Note that these parameters are exposed in the key frame posing interface by adding them to the GraphNode.dofs collection.

   In the display method of your class you can apply the transformations to the modeling matrix in the BasicPipeline using the functions you implemented in the previous objective. Note that applying the translation first is best such that translations are specified in directions aligned with the world/canonical coordinate system. If you apply rotation first (e.g., Euler angles, the representation is your choice), then translation will instead happen in the local system. Try it both ways to convince yourself that this is the right choice for your character!

   Note the use of the push and pop methods you implemented for the basic pipeline to save and restore the state of the modeling transformation matrix, and that the call to super.display() should come after your transformations so that the children will draw with the correct transformation.

   Test your new class with the single FreeJoint created in the CharacterMaker.create() method. Run the code and see how changing the parameters moves the node by clicking the "debug frames" boolean parameter checkbox in the controls window.

   Note that you have a button to re-create the character, which is useful for running your code in debug as you can quickly test small adjustments to your code without restarting your program.

4. ***RotaryJoint Graph Node (1 marks)***

   Create a *rotary* joint node. The joint should perform a translation within a parent frame to the location of the rotation axis of the joint, and a rotation about a single fixed axis, by an amount specified in a *DoubleParameter*. Your constructor or access methods should let you specify the translation in the parent frame, the axis of rotation, and also let you set reasonable min and max values for the angle parameter (see the setMinium and setMaximum methods of DoubleParameter). For instance, when creating an elbow joint, you'll be able to set the minimum and maximum elbow bend.

   Test your new class by adding a RotaryJoint to your character as a child of your FreeJoint in the CharacterMaker.create method, and of course, click the debugFrames checkbox when you run.

5. ***SphericalJoint Graph Node (1 marks)***

   Create a *spherical* joint node that uses 3 Euler angles to specify the pose (i.e., orientation). Again, like in the RotaryJoint, apply this orientation change at some fixed position within the parent frame. Expose the three rotation parameters by adding them to the GraphNode.dofs collection. Note that it may be hard to set reasonable limits different spherical joints! Choose whatever you think is best.

   Test your new class by adding a SphericalJoint to your character as a child node to something in the CharacterMaker.create() method.

6. ***Geometry Graph Nodes (1 marks)***

You must also create a geometry node for the leaves of your scene graph. Use the Cube and Sphere classes in the geom package, or define others (see optional objectives).

You will likely find it most useful to specify a translation and uniform scale to apply to the geometry to have it take an appropriate shape and position for your desired character. If it makes sense, you may even want to specify a rotation! You will likely also want to specify material parameters for the lighting computations of the geometry. See the kdID member variable in the BasicPipeline and note that you can use this (and other GLSL shader inputs) with glUniform calls to set the material parameters.

These geometry nodes will not likely have any parameters to interactively adjust their appearance, but feel free to expose whatever you like, e.g., you could let the material colour change to let your character blush. There is an opportunity to be creative by designing something special and exposing the parameters so that you can animate it.

Test your geometry nodes by adding them to any test nodes you've created so far, but note that the real test for your geometry nodes will be in the next objectives!

7. ***GLSL lighting (2 marks)***

   The basic fragment program does not implement any lighting computation. Add basic lighting for an aesthetically pleasing 3 direction lighting model. You will need to add uniforms for the 3 light directions, and their intensities (which you may even specify as a colour).

   Implement both diffuse Lambertian and Blinn-Phong lighting computations in the glsl/basicLighting.fp fragment program. Note that you will need to create uniforms beyond the kd in the provided code. Take care in adding code to the BasicPipeline to get the IDs of these uniforms so that you can set them properly (e.g., ks and the specular exponent).

8. ***Load Graph Nodes from XML (1 mark)***

   The provided code includes an xml parser. When the load from file checkbox is selected, pushing the recreate character button in the interface will rebuild your scene graph by loading the character from an xml file.

   Adapt the commented code for loading joints and geometry classes that you wrote in previous objectives. See the example stub character.xml in the provided code. Feel free to add additional attributes that you need to properly define your character!

9. ***Create a Character (2 marks)***

   Your program now has all the nodes needed to build a character, and the ability to create the scene graph by loading from and xml file.

   Create your character by editing the character.xml file, and make any adjustments to your parser and your GraphNode classes as needed.

   Note that your character must have reasonable complexity (e.g., at least arms and legs with knees and elbows) and reasonable behaviour to receive full marks for this objectives. For instance, elbows like joints should be rotary joints rather than sphereical joints and have reasonable limits. Also, geometry should be appropriately positioned such that the joints connect the ends of bodies and move in a reasonable manner (e.g., joint motions should not have the character tearing apart at the joints). Please discuss in the forums if you have questions about what is reasonable.

   Note that the constructors of your GraphNodes must add double parameters to the GraphNode.dofs collection. In doing so, the parameters for posing your character will automatically appear in the user interface.

10. ***Sample Pose (1 mark)***

    Having finished creating your character, create a pose to show your character to the world. Take a screen capture and save to an image file (png or jpg) and submit this along with your other materials. Your filename should have the form "lastname-description.png" where lastname is your last name and description is the name or description of your character.

11. ***Key Frame Animation (1 marks)***

    You must create a simple key-frame animation to bring your character to life. Perhaps have your character wave hello, or try to communicate with gestures, or something mod difficult such as a walk cycle! Note that choices you make in the parameters of joints might make some poses challenging (i.e., gimbal lock).

Simple key framing code is already written for you. Use the controls to set, delete, copy, and paste key poses. If you copy the pose at the beginning of the animation to the end then you can create an animation which loops. Use the save (and load buttons) to save your animation to edit again later.

One strategy for creating an animation is to create a number of keyframes with the basic motion of just a few joints, and then revisit each frame making additional adjustments and updating the keyframes with the new poses. Use the load and save buttons to save to the file a1data/keyposes.java but note that you may first need to create the directory! Also note that if you make changes to your character, your old serialized keyframe animation will very likely become useless.

12. ***Movie (1 mark)***

    Create a video of your keyframe animation.

    One way to do this is to click the "animate" checkbox on the keyframe controls, and then use the "Canvas Recorder Controls" record button to save the animating canvas to png files (but you must create a **stills** folder in your project directory first (DO NOT SUBMIT THIS FOLDER WITH YOUR ASSIGNMENT)). Then use ffmpeg to create an mp4 from the still images!

    Alternatively you can use a video capture tool such as fraps.

    Your movie filename should have the form "lastname-description.mp4", like above, though your file extension may change depending on the type of movie you create. Whatever format you use, it should be playable in VLC player.

13. ***Readme File (1 mark)***

    Create a *readme.txt* or *readme.pdf* file to submit with your assignment. Use the readme to explain any notable decisions you made in writing the code. You can also mention anything notable with respect to the creation of your character, such as which node you choose to be the root, or any notable features you have in your character. Be brief!

Optional

Up to two bonus marks will be given if you add extra polish to your assignment:

1. The viewing and projection matrices are set to be rather poor yet functional defaults. Use what you've learned in class to set a more appropriate camera fro your character
2. Within the geom package create and use more geometry in your character: flat capped cylinders, capsules (i.e., cylinders capped by half spheres), and cones.
3. Consider using procedural generation for a non-xml loaded character (e.g., to make a long tail from many segments), or add tools to let you easily control many joints at once (e.g., map a single double parameter to many joints along a tail or tentacle... see Figure 4 of Min et al. 2019 SoftCon for inspiration).
4. If you have other ideas, please just have fun and don't ask the profs or TA to pre-approve any bonus marks. Use your readme file to convince the TA that something you have done is excellent and deserves some extra recognition!

Finished?

Great! Be sure your name and student number appears is in the window title, in your readme, and in the top comments section of each of your source files.

Prepare a zip file with your source code folder, your sample pose image, your character file(s) and keyframe file(s), your optional video, and readme file. Only use a *zip* archive! Submit it via MyCourses. **DOUBLE CHECK** your submitted files by downloading them. You cannot receive any marks for assignments with missing or corrupt files!!

Note that you are encouraged to discuss assignments with your classmates, but not to the point of sharing code or answers. All work must be your own. Please talk to the TAs or the prof if the academic integrity policies are not clear.