CSCI 520 - Assignment 3

NAME: Shreya Bhaumik
USC ID: 6796453186

Operating System: macOS 10.15
Source Code Editor: Sublime Text, Version 3.2.2, Build 3211
Programming Language: C++
API: OpenGL(Compatibility Profile)
Libraries Used: ADOL-C, Eigen, Vega FEM

How to execute the code (For macOS):

1. To set up ADOL-C and Eigen, follow steps mentioned in
   http://barbic.usc.edu/cs520-s20/assign3/
   To set up OpenGL as well, follow the steps in the aforementioned site.

   CAUTION:
   However, for macOS 10.15, "`brew install freeglut`" did not work.
   I had to do "`brew cask install xquartz`" and then "`brew install freeglut`" instead.
   Also, instead of installing freeglut-3.0.0, it installed freeglut-3.2.1. Therefore, after uncommenting "`OPENGL_LIBS=-framework OpenGL /usr/local/Cellar/freeglut/3.0.0/lib/libglut.dylib`", I had to convert the 3.0.0 to 3.2.1 i.e. "`OPENGL_LIBS=-framework OpenGL /usr/local/Cellar/freeglut/3.2.1/lib/libglut.dylib`".
   This should be enough to be able to compile the code.

2. Compile using command: `make`
   Note: To delete all the object files and executables: `make clean`

   CAUTION:
   Again, for macOS 10.15, after running "`make`", I got the error:
   "`freeglut (../driver): failed to open display ''`".
   To resolve this, give Full Disk Access to both *XQuartz* and *Terminal* on *System Preferences -> Security & Privacy -> Privacy -> Full Disk Access*. Then do the golden step of restarting the laptop.

3. To run the executable: `./run.sh`
   Editing the run.sh file:
   - You can change between models by uncommenting the "`cd <model-name-folder>`" you want to render and commenting the one which was in use.

- In "`../driver skin.config 0 0`", after filename-*'skin.config'*, next argument is for choice in IK algorithm, and the one after that is for choice in skinning method to be used.
  IK algorithm - 0(Tikhonov IK method), 1(Pseudoinverse method), 2(Transpose method)
  Skinning method - 0(Linear Blend Skinning), 1(Dual Quaternion Skinning)
  By default, i.e. if you keep it as "`../driver skin.config`", both are set to 0.

- Instead of "`../driver skin.config 0 0`", you can execute "`../driver changedskin.config 0 0`" to run IK with a set of different IK-Joint-IDs. This can be implemented for all the 4 models – armadillo, hand, dragon and star.

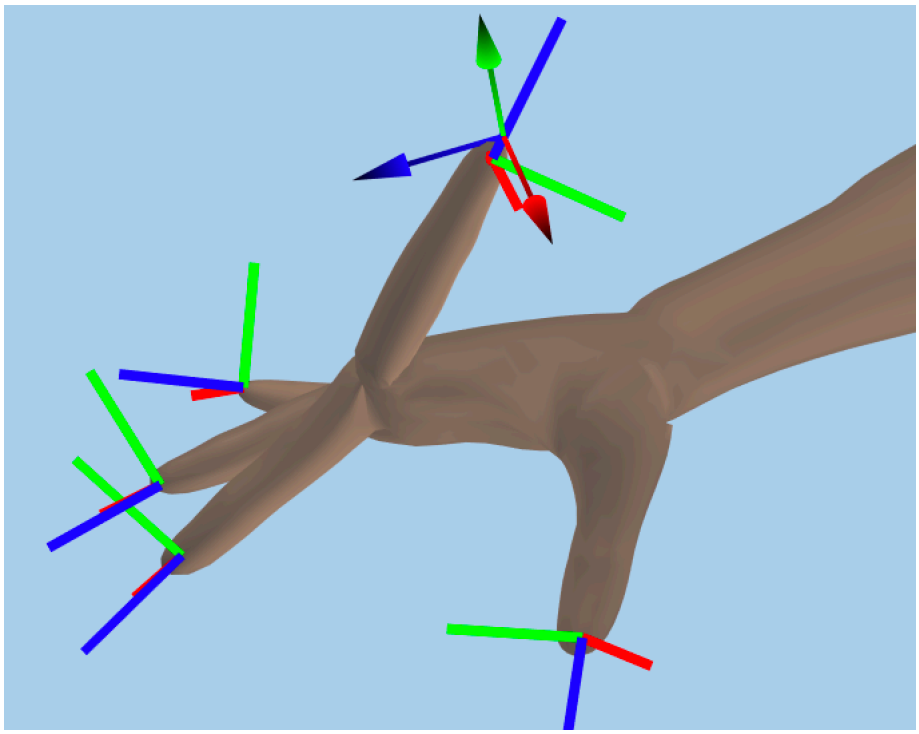
4. Controls:
   - 'r' – run automation of IK-handle drag (can be toggled *on/off*)
   - 'c' – toggle visibility of skybox
   - 'w' – toggle visibility of wireframe
   - 'e' – toggle visibility of model mesh
   - 's' – toggle visibility of skeleton
   - 'x' – take 550 screenshots in sequence and store as *.ppm* files
   - '=' – show hierarchy of each joint on the skeleton one by one
   - '\' – reset camera position to initial position
   - '0' – reset model mesh and skeleton to rest position
   - 'ESC' - exit

Accomplishments:

1. Implemented Forward Kinematics (*FK.cpp*).
2. Computed skinning transformations for all the joints (*FK.cpp*).
3. Coded Forward Kinematics again for help in computing Jacobian matrix using ADOL-C (*IK.cpp*).
4. Set up ADOL-C by using the above-mentioned Forward Kinematics function as the function to be computed using ADOL-C (*IK.cpp*).
5. Implemented the Tikhonov IK method (*IK.cpp*).
6. For skinning, used Linear Blend Skinning (*skinning.cpp*).

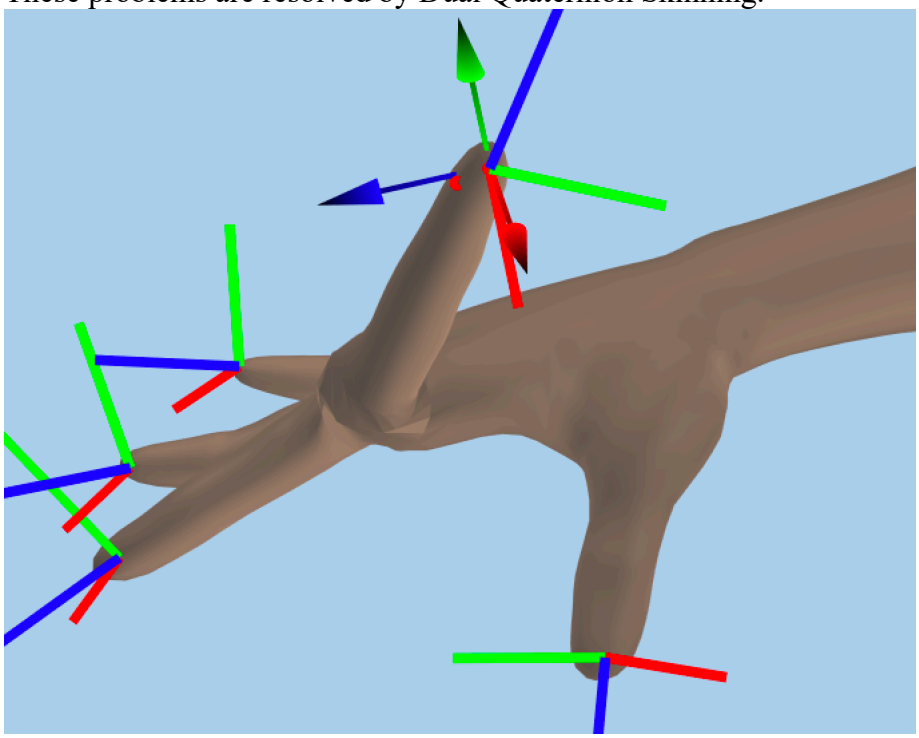
Extra Credits:

1. **Dual Quaternion Skinning:**
   Linear Blend Skinning is a very popular skinning algorithm, but it produces some horrible artifacts. It causes "candy-wrapper" and "collapsing-joint" artifacts.



(This image was taken using Linear Blend Skinning in the Assignment)

These problems are resolved by Dual Quaternion Skinning.



(This image was taken using Dual Quaternion Skinning in the Assignment)

However, Dual Quaternion Skinning has its own artifacts known as “bulging-joints” and “distorted normal”.

2. **Pseudoinverse method for Inverse Kinematics:**
Pseudoinverse, also known as Moore-Penrose inverse, sets $\Delta\theta$ to:

$$\Delta\theta = J\dagger\ \Delta P \quad \text{-------} \ (1)$$

where n x m matrix $J\dagger$ is the pseudoinverse of J.
$\Delta\theta$ turns out to be the unique vector of smallest magnitude which minimizes $||J\Delta\theta - \Delta P||$.
Pseudoinverse method has problems near singularity. When the model is in an extended posture, the Jacobian matrix is on the verge of losing full row rank. This causes large angular changes in the model and the model starts moving without control.

The algorithm (1) is derived as:

$$J\Delta\theta = \Delta P$$
$$J^TJ\Delta\theta = J^T\Delta P$$
$$\Delta\theta = (J^TJ)^{-1}\ J^T\Delta P$$

J^TJ is an n x n matrix whose inverse would be very expensive to calculate as n here is corresponding to the total number of joints in the model (the degree of freedom of the model). If J^TJ is invertible, then $J\dagger = (J^TJ)^{-1}\ J^T$. If JJ^T is invertible, then $J\dagger = J^T\ (JJ^T)^{-1}$. JJ^T will be an m x m matrix and m<=n. So, computing its inverse will be less expensive.

Using *Damped Least Square method* (or as we used Tikhonov regularization), we can remove and reduce near singularities in the Jacobian matrix and thereby stabilize $\Delta\theta$.
The algorithm we use for Damped Least Square method is:

$$(J^TJ + \alpha I)\ \Delta\theta = J^T\Delta P \quad \text{-------} \ (2)$$

By adding the orthonormal set I, we can guarantee that an inverse will exist. However, we have to choose α very carefully. If it is too small, it will still result in jerky movements when the model is in an extended posture – which is the same problem as with pseudoinverse method.

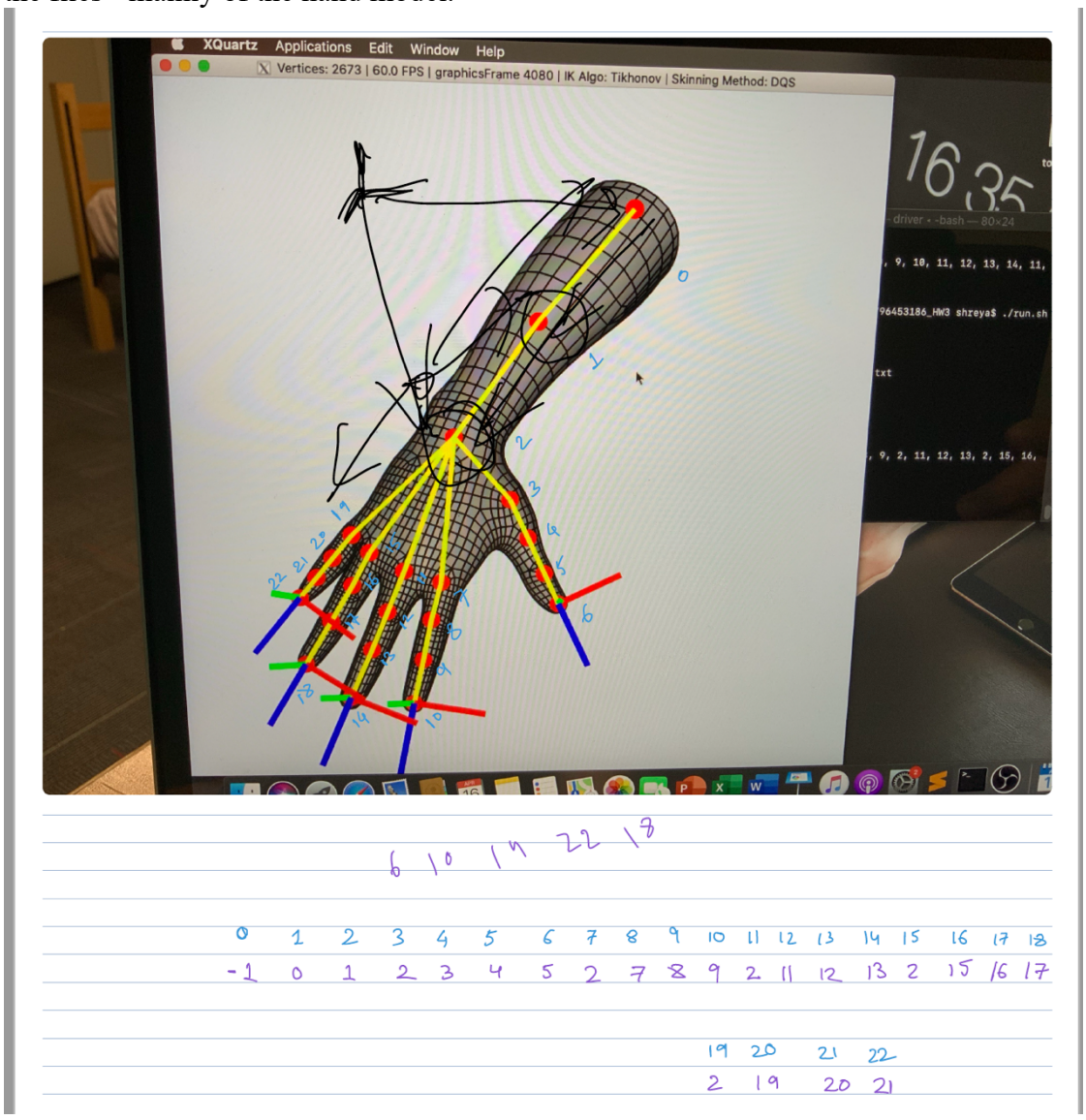
3. **Transpose method for Inverse Kinematics:**
This method just uses the transpose of the Jacobian matrix.
The algorithm goes as:

$$\Delta\theta = \alpha J^T\Delta P \quad \text{-------} \ (3)$$

Once again, α needs to be chosen very carefully. It should be chosen as close as possible to ΔP.

4. **Hardcoded a new model – STAR:**
   Manually hardcoded all the files necessary for the model to be rendered and be usable for Forward Kinematics, Inverse Kinematics and Skinning.
   The files are all the six files (*star.obj, star.mtl, jointHierarchy.txt, jointWeights.txt, jointRestTransforms.txt, skin.config*) in the folder '***star***'.
   The folder has been setup exactly as the folders for the other models provided with the starter code. It can be implemented similarly in the *run.sh*.

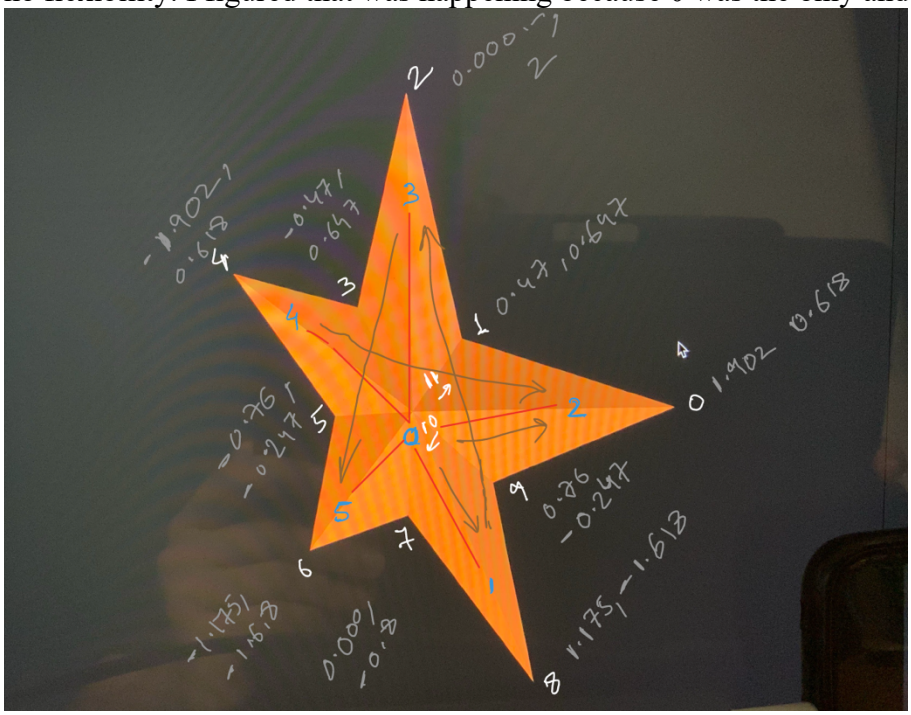
   THOUGHTS:
   For better understanding of the joint hierarchies, skeleton structure, weight distributions and meshes, I thought of creating all the files for a model. It's a behemoth task to take up, if the model has too many vertices. So as to make an informed decision about the model, I studied the files - mainly of the hand model.



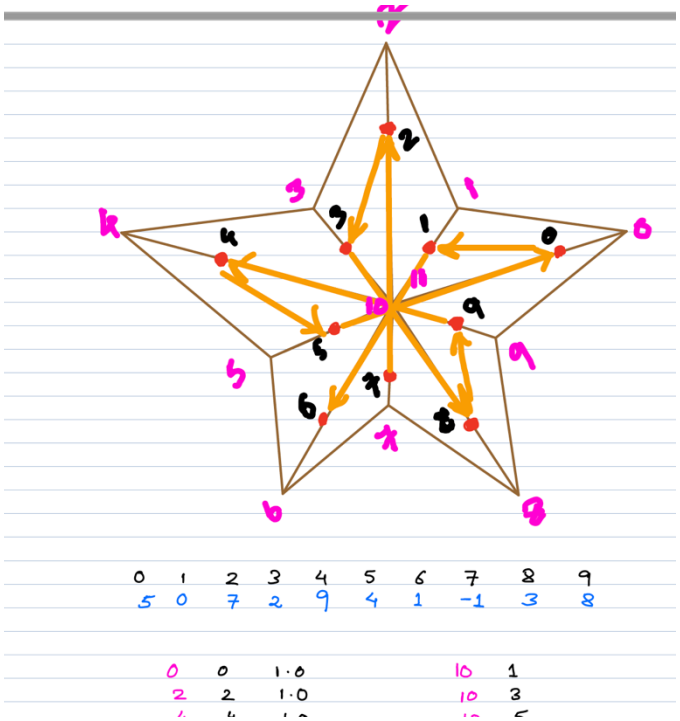
   After much thought, I decided upon making a simple star with 12 vertices. After having

made the *star.obj* and the *star.mtl* file, I had a hard time deciding on the structure and hierarchy of the skeleton – How many joints should the skeleton have? How should the hierarchies be arranged for best expression during IK?

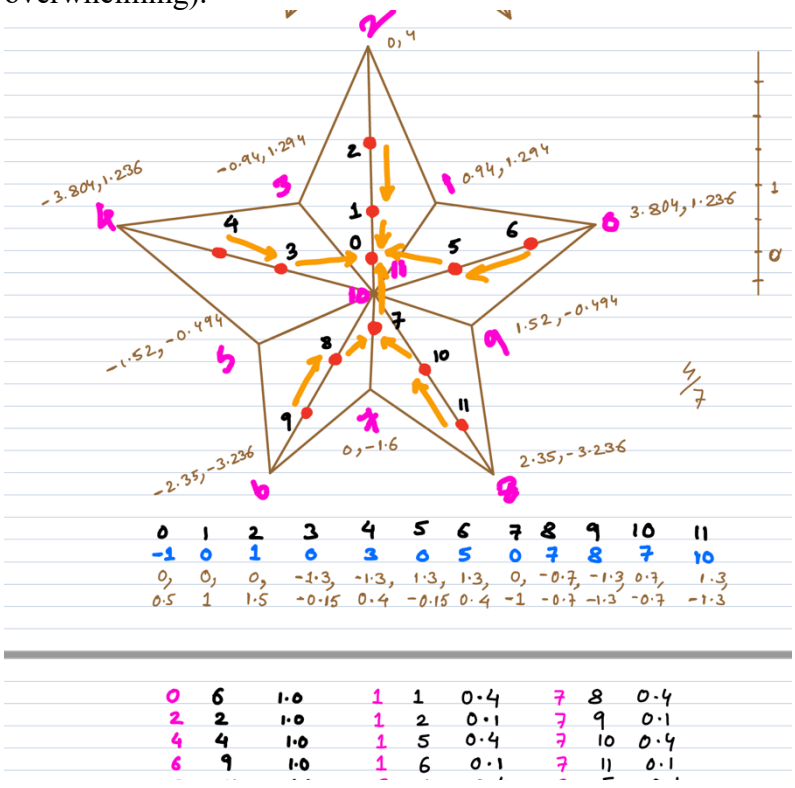
My first attempt was a skeleton with 6 joints and the red lines (in the image below) indicating the structure of the skeleton - meaning 0 was the parent of all the other joints. I made all the files for *jointHierarchy.txt, jointWeights.txt, jointRestTransforms.txt, skin.config* and after running the code on the model I found that the model moved like a stiff star. Pulling one handle at any point always made the whole model move like a stiff metallic piece. There was no flexibility. I figured that was happening because 0 was the only and universal parent.



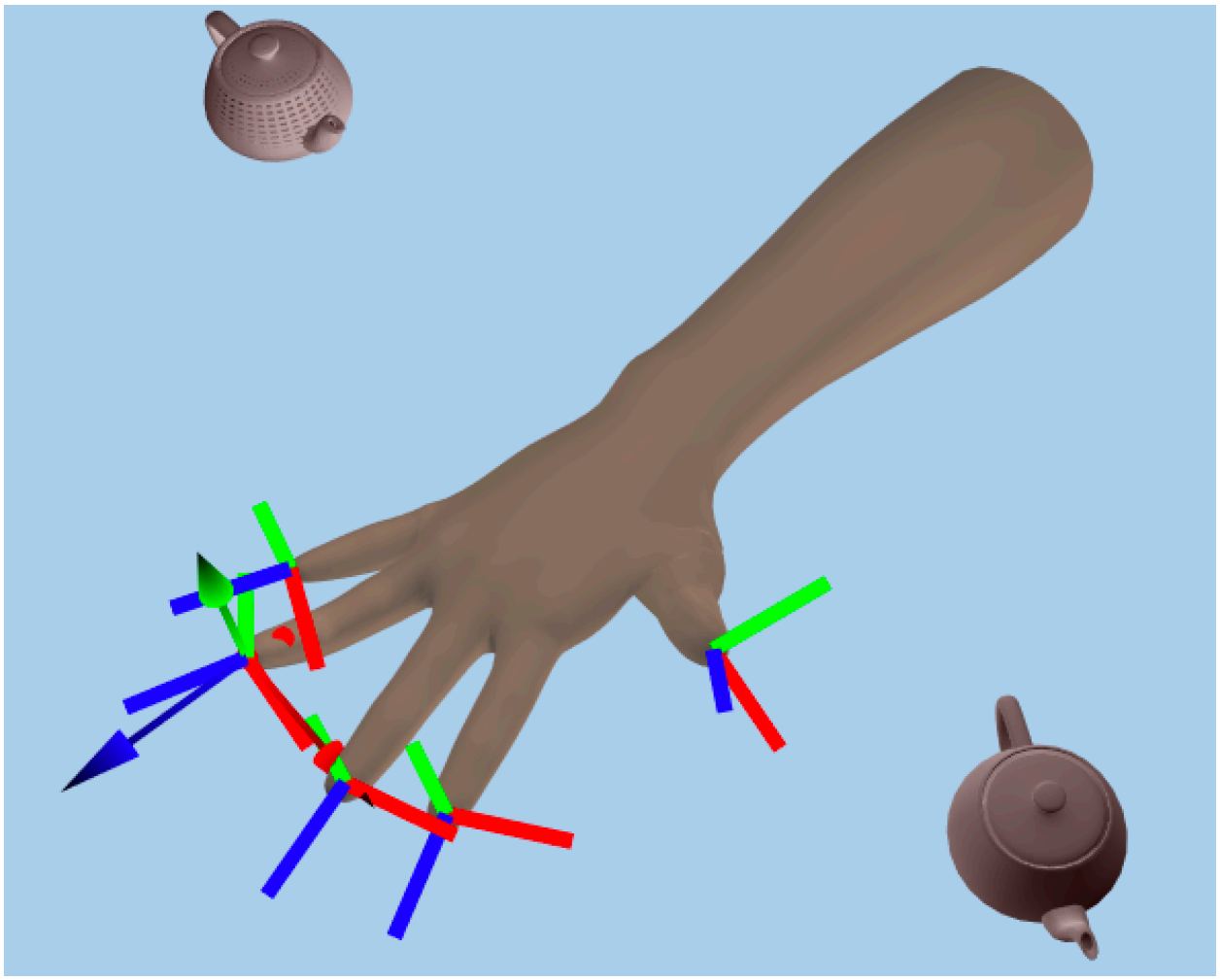
Next, I attempted two other worthless sets of skeleton structures before I realized that I might have to increase the number of joints for it to work better. This skeleton structure also failed because of how unnatural it is. The bones or bars between two joints crossed each other and also if pulled too much, the structure would collapse because of huge mismatch between bone geometry & hierarchy and skinning weight distribution.

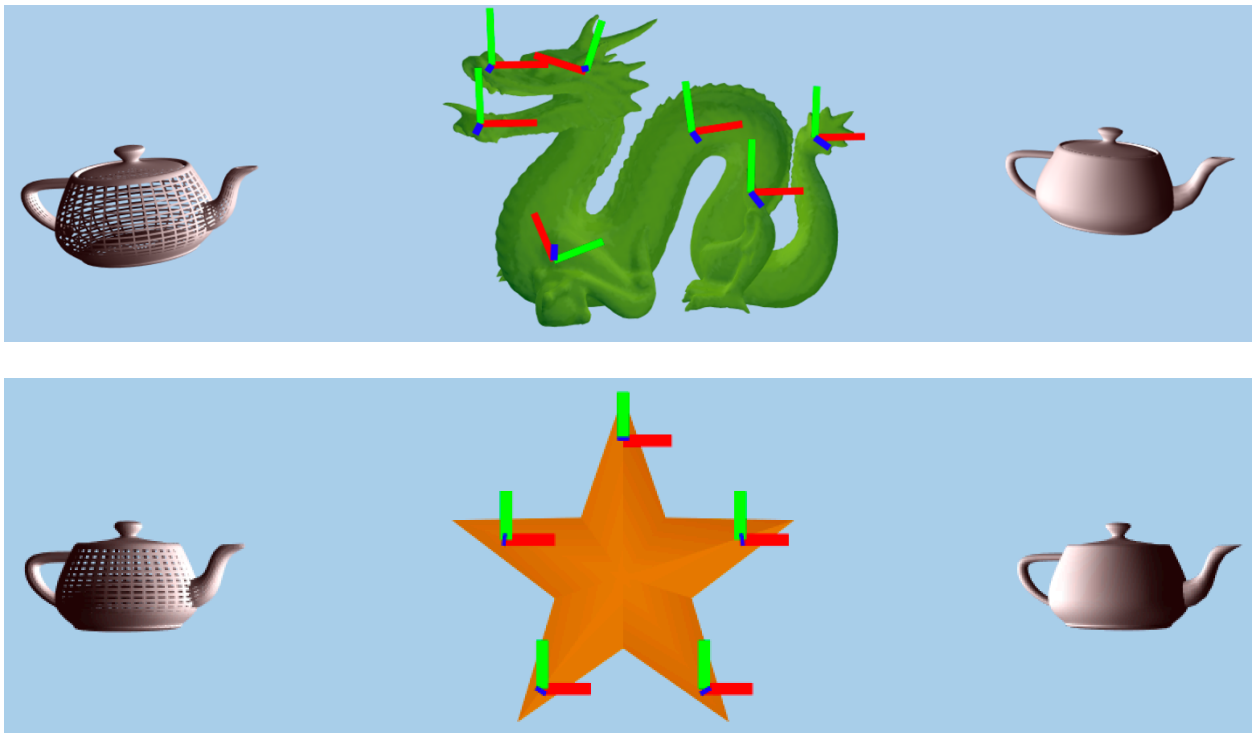


The final skeleton structure that I settled for is as below. This skeleton structure seemed well balanced even though I ended up having 12 joints for a model with 12 vertices (which is not a good choice for a model – but manually hardcoding for more vertices seemed overwhelming).



5. **Fixed size Utah Teapots in the scene for model size comparison:**
While making the star model, I was worried about how the arrows for different axes and the skeleton bones and joints would size up relative to the model. Then I realized that the starter code intelligently calculates a model-radius which it gets from creating a bounding box about the model. It then uses this radius to set the camera at a distance such that the size of the model doesn't matter. The arrows and everything else about the skeleton are made in such a manner that their thickness remains constant even when camera zooms out from the model. This made me wonder about how to visualize the relative sizes of the different models. So, I placed two fixed size Utah teapots (one solid and the other wireframed) in the scene.





Who would have guessed the armadillo was this small and the hand was this big!
Even my star of 12 vertices is bigger than the armadillo of 5002 vertices!

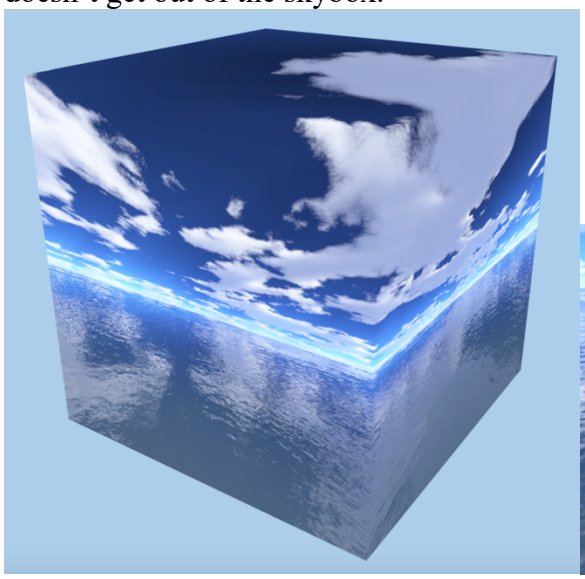
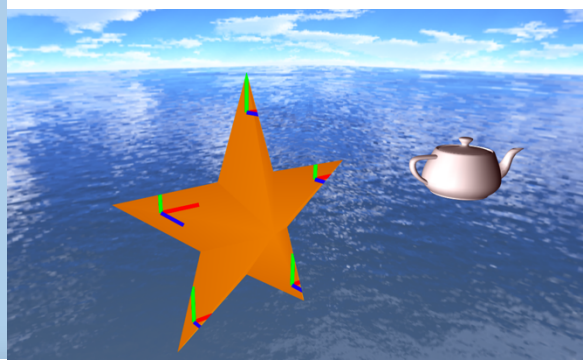
6. **Automated IK-handle drag (can be toggled *on/off* – 'r'):**
I was having a lot of trouble in recording the animation image sequence for the homework submission. I patched up the screenshot code from HW1 into this HW3, but as soon as I started recording, the fps dropped and made it hard for me to drag the IK-handles. So, I decided to automate the IK-handle drag.

   One trouble I could think of was, since the objects are so varied in sizes, how do I set a value for the drag so that it is not too much for armadillo and not too little for the hand.
The realization (from the previous extra credit point) of how the model-radius keeps a measure of the model's size, helped me use the radius to set a suitable drag distance that would auto-adjust itself for every model.
Another problem I faced was about how to show which handle is being dragged, since I am not actually clicking the mouse. I found the solution in professor's *Vega* library, which I used, to call the function that can display the arrows that show up, when the mouse is actually clicked for dragging that particular IK-handle.

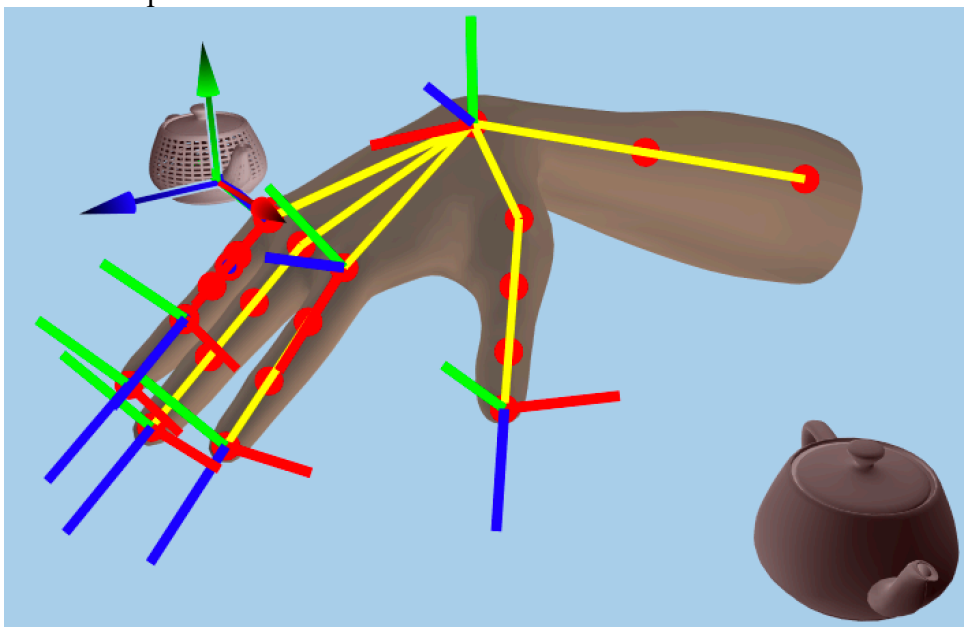
   After the model has moved following the IK-handle drag sequence, I have set the model to go back to its rest position.

   The IK-handle drag automation is a toggle option on keyboard which can be turned *on/off* by pressing 'r'. If 'r' is toggled *off* before the drag sequence is over, the automation immediately stops, and the model goes back to rest position.

7. **Skybox (can be toggled *on/off* – 'c'):**
   Implemented a skybox. However, loading the pictures was proving to be a toll on the fps – which was again bad for taking screenshots. Therefore, I kept the skybox as a toggle option on keyboard. It can be turned *on/off* by pressing 'c'.
   The skybox is also scaled based on the dimension of the model; so that, the camera (which is also positioned in the starter code using the scale factor) does not go out of the skybox. Care has also been taken so as to ensure that the two Utah teapots (whose size doesn't change), doesn't get out of the skybox.

8. **Tweaked IK Joints IDs:**
   I created a copy each, by the name "*changedskin.config*", of the config files for all the 4 folders of the 4 models - armadillo, dragon, hand and star. In those files I tweaked the IK Joint IDs. This enabled me to easily make the models move in ways previous IK Joints were not able to provide with.



With the "*changedskin.config*" file for the hand, it can be positioned as shown in the above picture, which was very challenging with handles previously being just at the finger tips.

I also observed that, IK algorithms - mainly pseudoinverse, become more unstable with increase in the number of IK joints. Also, their instability increases more if the IK joints are higher in the joint hierarchy (i.e. if the joint is a parent of many other joints directly or indirectly).

9. **Manipulated lighting and model colors and background:**
   Changed the lighting setup of the scene.
   Changed the colors of the models by tweaking the respective **.mtl* files (*as is visible in the screenshots used previously*).
   Also changed the background color of the screen.

10. **IK algorithm and Skinning method used displayed on window title:**
    The IK algorithm and the Skinning method currently being used, can be seen on the window title bar alongside previously present vertices count, fps and graphics frame.

Folders:

- The animation screenshots in numbered order can be found in the folder "*Animations*" in the root.
- I have also provided some other screenshots shown here on the readme file, in a folder within the Animations folder by the name "*InterestingShots*" – these are not meant as part of animation and are very low resolution.
- The images used for the skybox are stored in the folder "*Skybox"* in the root.

Thank you!