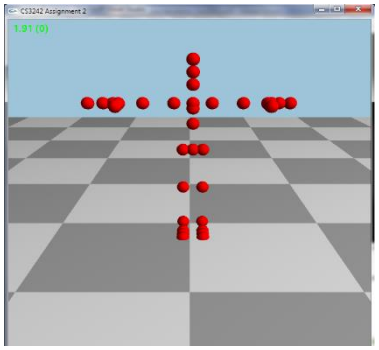
CS3242 3D Modeling and Animation

Assignment 2: Forward and Inverse Kinematics

In this assignment, you are going to implement forward and inverse kinematics to simulate movement of a human character. You can start with the provided code template.

The code package contains a simple BVH player and a set of .bvh motion data (in data/BVH_NUS005 folder). The player uses the GLM library for vector and matrix calculation, and the GLUT library to support OpenGL rendering.

You can build the code in Visual Studio 2015. Below is a screenshot of the program. In the viewport, you can rotate with left mouse, and zoom with right mouse.

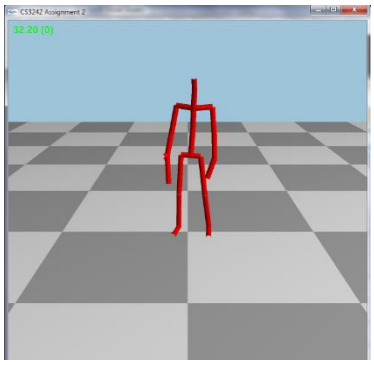


Press 'L' key to load a .bvh file.

The renderer then displays a character in T-pose. There are five display modes: joint – OpenGL matrix, joint – GLM matrix, joint – GLM quaternion, skeleton, and mannequin. The default mode displays joints transformed by OpenGL. You can also press 'a' or 'A' key to scroll to other display modes.

Part 1: Animation basics [1%]

After a BVH file is loaded, the top left corner of the window displays the time elapsed since the BVH file is opened, and the frame that is currently played. For now, it only draws frame zero.



Your task is to modify the idle() function in BVHPlayer.cpp so that the animation can be iterated from the first to the last frame. Remember to loop when the animation ends.

[Hint: you may use getFrameTime() to get the time per frame and getNumFrames() of BVH class to get the total number of frames in the BVH file.]

Part 2: Forward kinematics [1%+4%+4% = 9%]

You may notice that the joints are still not drawn with correct transformations. Your task now is to animate the character using forward kinematics so that the joints are transformed properly.

a. Transform via OpenGL matrix stacks. [1%]

Edit renderJointsGL(const JOINT*, const float*, float) in BVHAnimator.cpp to make correct joint transformation.

[Hint: you can study how the skeleton is drawn and do similarly for the joints.]

b. Transform via quaternion algebra. [4%]

In this part, your job is to edit the quaternionMoveJoint() method in BVH.cpp to make correct joint transformation by **using quaternions**. In particular, transformation is represented by a translation vector and a quaternion (for rotation). You need to maintain the translation and the quaternion properly for each joint.
Note that no marks will be given for this part if quaternion is not used for rotation.

To view the figure rendered by quaternion, switch to display mode Joint – quaternion (by pressing 'a' three times). The joints are colored in blue.

[Hint: study how matrixMoveJoint() is implemented. In fact, each 4x4 matrix contains a 3x3 rotation matrix and a 1x3 translation vector. Learn how the rotation matrix and the translation vector are maintained after each transformation. Replace the rotation matrix with quaternions.]

c. Skinning [4%]
It might be dull to just draw joints or bones. Complete the method BVHAnimator::renderMannequin(int frame, float scale) to draw a mannequin by rigidly skinning the character. You can simply use primitive geometries (sphere, cube, cylinder etc.) in glu or glut. Once this method is complete, press A to toggle among [JOINT]-[SKELETON]-[MANNEQUIN] to enjoy the animated motion.

Part 3: Inverse kinematics [10%]

For this part, we will be using only *static poses*, i.e., only to draw the character at one particular frame).

Uncomment the following line (line #7, right below the header) in BVHPlayer.cpp:
#define TEST_IK
This enables the program to draw a sphere at a specified target location and pose the character's left arm to reach the target with his left hand joint.

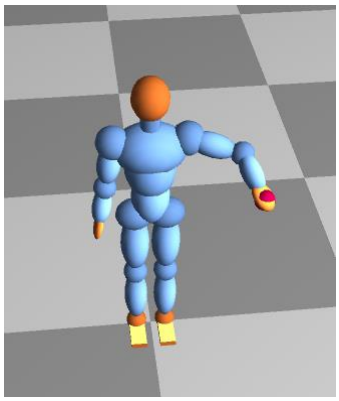
Your task is to complete the method BVHAnimator::solveLeftArm(int frame_no, float scale, float x, float y, float z) by implementing an IK solver to compute the joint angles of left shoulder joint and left elbow joint. You can choose either the CCD or the Jacobian Inverse method. You may assume 3DOF (X-, Y-, Z- rotation) for shoulder joint and 1DOF for elbow joint, i.e., 4 variables and 3 constraints.

Use left, right, up, down arrow keys, pageUp and pageDown to move the target (you may refer to the skeyboard function in BVHPlayer.cpp). Once you have implemented the solver correctly, the program shall perform similar to this:
https://www.youtube.com/watch?v=n4imnXV5IWE

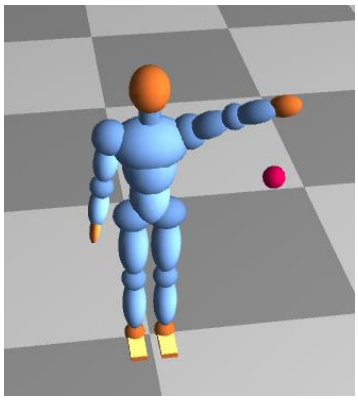
[Hint: you don't have to deal with the range of motion of joints. That is, an elbow can bend backward and we will still give you full marks if the hand reaches the target.

Try to understand the meaning of offset for each joint. Be cautious of using "*" operator to multiply matrices in glm library. Do check the source code of "*" operator in glm library.]

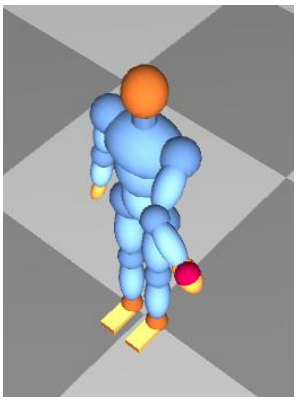
Test with the following examples.



(a) A valid solution



(b) An invalid solution



(c) Not valid in real life but acceptable for this assignment.

Submission

Submission deadline is **Tuesday, April 12, 2016, at 11:59AM (noon)**. Please zip your BVH.cpp, BVHAnimator.cpp, and BVHPlayer.cpp and name the file as <YOURNAME>_Asn2.zip before uploading to IVLE.

Please note that for this assignment, you are required to edit ONLY the specified methods/functions. You may add extra functions, but please avoid changing any other existing functions. There will be a 50% penalty on uncompilable program, and a 20% penalty if program crashes at runtime.

References

1. A tutorial on matrices with GLM: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/
2. GLM code samples http://glm.g-truc.net/0.9.5/code.html
3. GLM matrix functions: http://glm.g-truc.net/0.9.4/api/a00133.html