Inverse-Kinematics-In-Opengl

```
- NOTICE: This project is a read-me ONLY.
- The codebase the project was developed on is confidential and thus cannot be shared.
```

Presentation (Presents outdated codebase. Quaternion calcultions have been remedied as explained below).

I implemented inverse kinematics in OpenGl

Development Process:

- Find a way to represent and update the joint chain
    - Ended up finding all the bones with no children, and those endpoints would be the end effectors
    - The chain would be the bone with the end effector and all its parents
    - Because IK deals with joints, but we only have bones with A4, I had to add an extra layer of abstraction when updating joint positions:
        -

```
private setJointPosition(index: number, position: Vec3) {
  if (index == 0) {
    this.jointChain[0].position = position.copy();
  } else if (index == this.jointChain.length) {
    this.jointChain[index - 1].endpoint = position.copy();
  } else {
    this.jointChain[index - 1].endpoint = position.copy();
    this.jointChain[index].position = position.copy();
  }
}
```

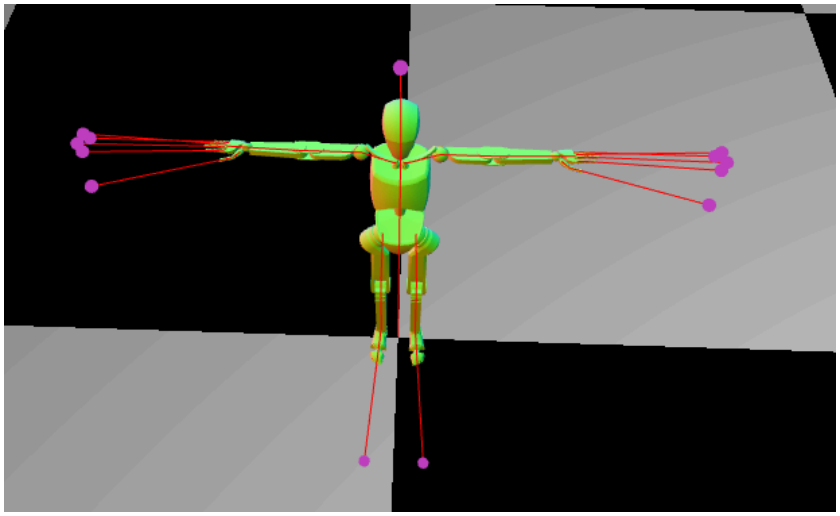
        -

```
private getJointPosition(index: number) {
  if (index == 0) {
    return this.jointChain[0].position.copy();
  } else if (index == this.jointChain.length) {
    return this.jointChain[index - 1].endpoint.copy()
  } else {
    return this.jointChain[index - 1].endpoint.copy();
  }
}
```

- Find a way to visualize and intersect the end effectors
    - Created and rendered a unit sphere (pain), and put it at all the end effector positions
        - Used http://www.songho.ca/opengl/

gl_sphere.html#:~:text=In%20order%20to%20draw%20the,triangle%20strip%20cannot%20be%20used as reference.

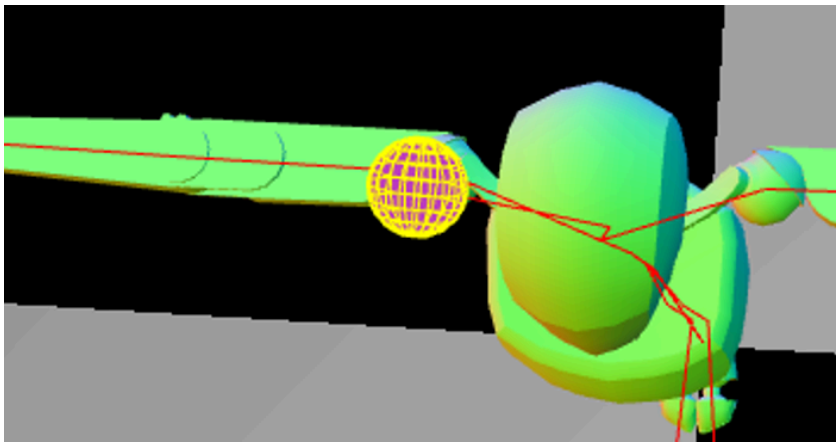
- results:



- Find when mouse intersects end effector
  - Sphere intersection logic

```
// returns t value
private intersectSphere(rayOrigin: Vec3, rayDirection: Vec3, sphereCenter: Vec3, sphereRadius = GUI.kSphereRadius): number {
  //used https://iquilezles.org/articles/intersectors/ as resource
  let rayToCenter = Vec3.difference(rayOrigin, sphereCenter);
  let b = Vec3.dot(rayToCenter, rayDirection);
  let c = Vec3.dot(rayToCenter, rayToCenter) - sphereRadius * sphereRadius;
  let h = b * b - c;
  if (h < 0) {
    return Number.MAX_VALUE;
  }
  let t1 = -b - h < 0 ? Number.MAX_VALUE : -b - h;
  let t2 = -b + h < 0 ? Number.MAX_VALUE : -b + h;
  return Math.min(t1, t2);
}
```

- Highlight intersected end effector with gridded lines
  - Needed extra unit sphere, renderpass, shader, and logic for the gridded lines around the intersected end effector
  - results:

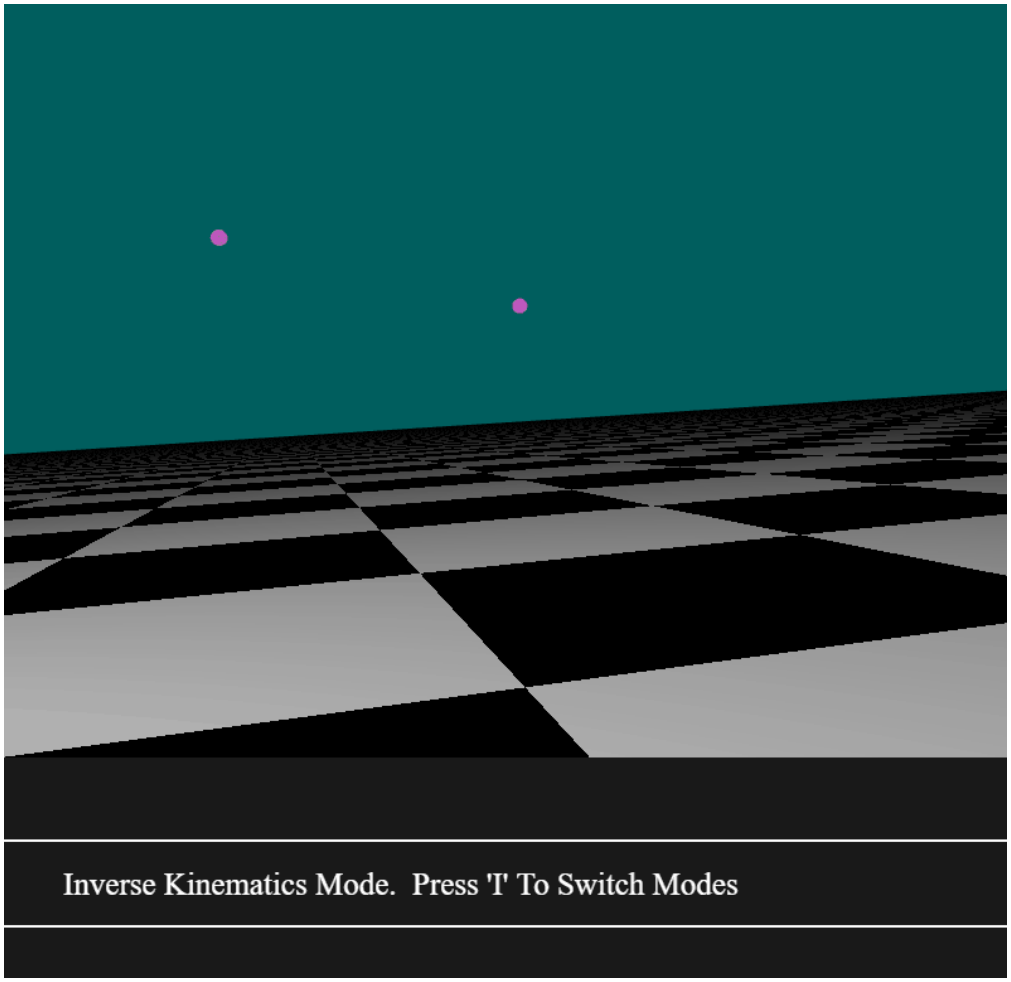


- Find way to tell end effector where I want it to go
  - When dragging an end effector, targetPos = p + v * (most recently

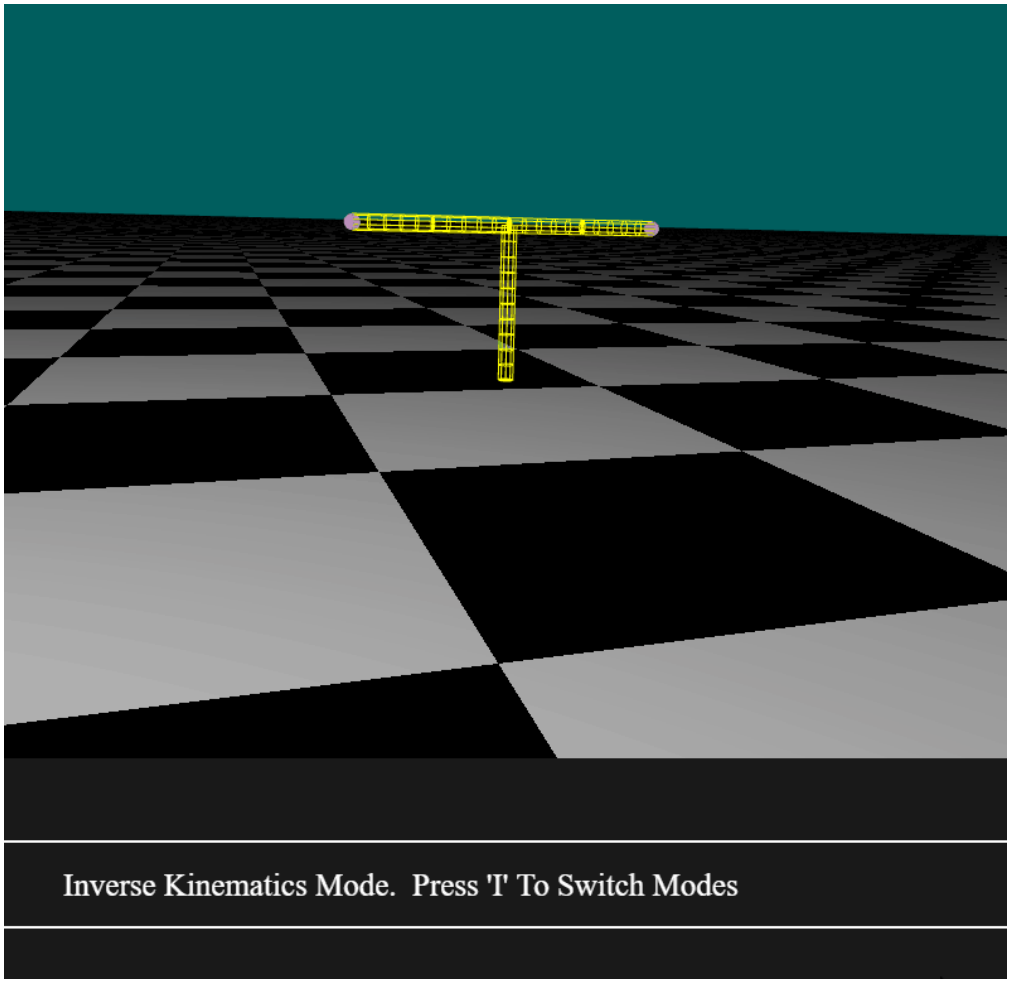
calculated t value when calculating end effector intersection)

```
private getTargetPos(mouse): Vec3 {
  var mousePosWorldCoords = this.screenToWorldCoords(mouse.offsetX, mouse.offsetY);

  let p = this.camera.pos();
  let v = Vec3.difference(new Vec3(mousePosWorldCoords.xyz), p).normalize();

  let vScaled = new Vec3();
  v.scale(this.lastEndEffectorT, vScaled);
  let targetPos = Vec3.sum(p, vScaled);
  return targetPos;

}
```

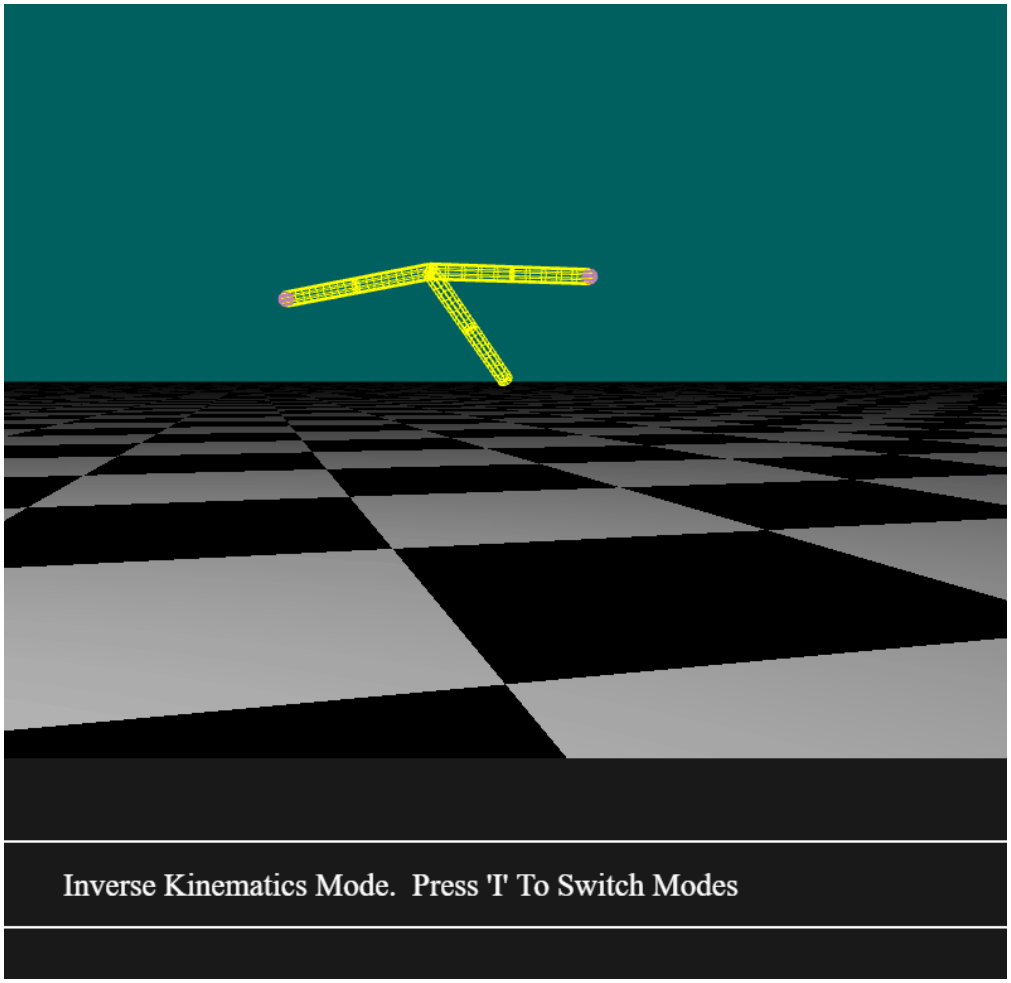
- Results:



- Implement FABRIK
  - Used http://andreasaristidou.com/publications/papers/FABRIK.pdf as main reference
  - Code is inverseKinematicsTranslation(targetPos) in Scene.ts
  - Results:



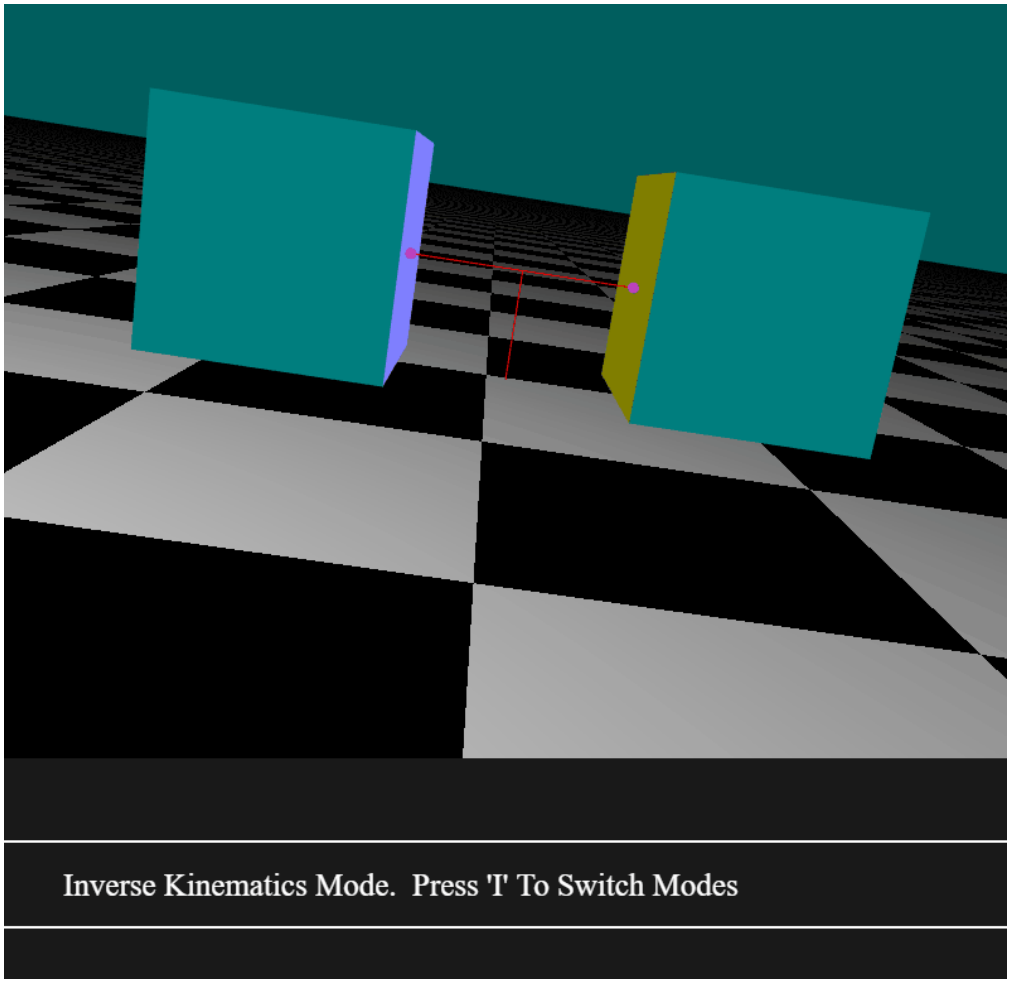
- Fix branching issue
    - FABRIK doesn't handle branching (updating positions of bones not in the joint chain)
    - I'm handling this by setting all children's positions to their parents endpoint at the end of an IK iteration
        - This works for most scenes, but breaks if the scene has disconnected bones (such as the mannequin and the head). I tried to remedy this, but didn't have enough time to find a good solution.
    - Results:



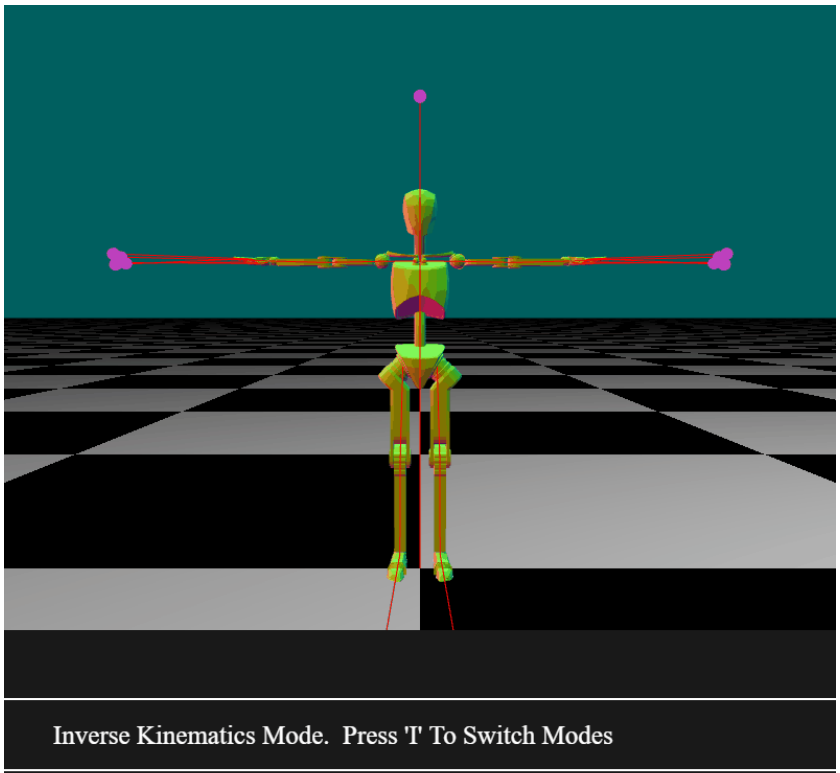
- Get Quaternion calculations working
    - This is what the professor and I were discussing for quite a while.
    - The solution ended up being quite simple:

```
private updateRotationQuat() {

  let q = new Quat();
  // initial orientation
  let v1 = Vec3.difference(this.initialEndpoint, this.initialPosition).normalize();
  // final orientation
  let v2 = Vec3.difference(this.endpoint, this.position).normalize();
  let a = Vec3.cross(v1, v2);

  q.xyz = a.xyz;
  q.w = v1.length() * v2.length() + Vec3.dot(v1, v2);
  q.normalize();
  this.rotation = q;

}
```

      Used https://stackoverflow.com/questions/1171849/finding-quaternion-representing-the-rotation-from-one-vector-to-another as reference
    - Results:

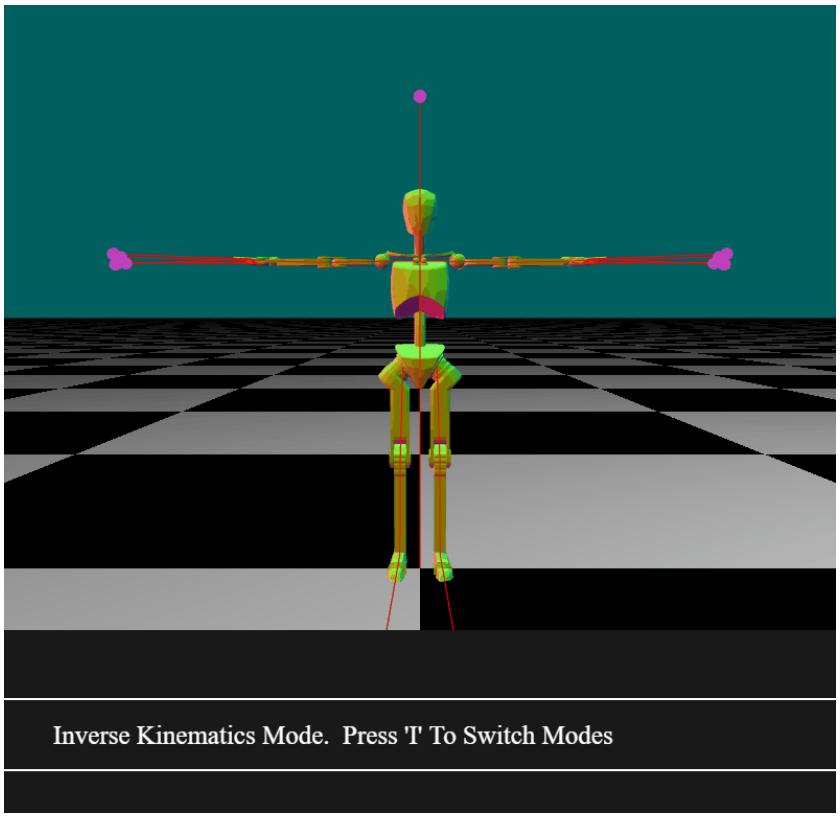


- Bugs/limitations
    - My current bone branching implementation assumes bones/joints are connected, which isnt always the case (mannequin and head scene).
        - So the bones snap together when they shouldnt

- 

- 

    - Didn’t have time to implement angle constraints (so rotations on mannequin look unrealistic)



- Codebase difficulties
    - A4 is extremely difficult to deal with due to needing render passes and shaders for everything (rendering the sphere/gridded lines around the sphere alone was quite painful)
    - A4 has bones, not joints, so had to add extra logic and abstractions to represent that
    - FABRIK expects a single chain, not branching, which is required for A4 models
    - FABRIK expects connected joints, but A4 models have separated joints (not enough time to fix this, unfortunately)
    - I hate quaternions (that issue alone took forever, yet the solution was so simple ;-☺