Report for A5: Keyframing and Animation

https://gitlab.com/mhartfield/skinning-and-mesh-interactivity

We, Raymond Wang and Max Hartfield, have completed all that was required for Assignment 5: Keyframing and Animation. We have correctly implemented both keyframing and animations.

Keyframes:

In Gui.ts, we created a keyframe class to store the keyframe info. The info we stored included a list of rotations, positions, endpoints, and ordering. This keyframe class is responsible for interpolating between two keyframes and resetting the bones to a specific keyframe.
The Gui has a list of keyframes which gets extended each time the user presses the ‘k’ key. Each time the user presses the ‘p’ key we reset the bones to the first keyframe to start the animation.

```
export class KeyFrame {
 private rotations: Quat[];
 private positions: Vec3[];
 private endpoints: Vec3[];
 private ordering : number[];
  constructor(bones: Bone[]) {

   this.rotations = [];
   this.positions = [];
   this.endpoints = [];

   this.ordering = [];
   let currBones : number[] = [];

   for(var i: number = 0; i < bones.length; i++) {
     this.rotations.push(bones[i].rotation);
     this.positions.push(bones[i].position);
     this.endpoints.push(bones[i].endpoint);
     currBones.push(i);
     this.ordering.push(i);
   }

   while(currBones.length > 0) {
     let next : number[] = [];
     currBones.forEach(parent => {
       let parentBone : Bone = bones[parent];
       parentBone.children.forEach(child => {
         this.ordering.push(child);
         next.push(child)
```

```
        });
      });
      currBones = next;
    }

  }
  public interpolate(kf: KeyFrame, time: number, bones: Bone[], initPos : Vec3[],
initEnd : Vec3[]): void {
    for(var count : number = 0; count < this.rotations.length; count++) {
      let i : number = this.ordering[count];

      // get slerp'd rotation quaternion
      let slerp : Quat = Quat.slerpShort(this.rotations[i], kf.rotations[i], time, new
Quat())
      bones[i].rotation = slerp;

      // case for if the bone has a parent: must calculate the new position first
      if(bones[i].parent >= 0){
        let v_bp : Vec3 = initPos[i].subtract(initPos[bones[i].parent], new Vec3());
        bones[i].position = bones[bones[i].parent].rotation.multiplyVec3(v_bp, new
Vec3()).add(bones[bones[i].parent].position);
        // bones[i].position = bones[bones[i].parent].endpoint;
      }

      // update the endpoint based on the new rotation
      let v_body : Vec3 = initEnd[i].subtract(initPos[i], new Vec3())
      bones[i].endpoint = bones[i].rotation.multiplyVec3(v_body, new
Vec3()).add(bones[i].position, new Vec3());
    }
  }
  public reset(bones: Bone[]): void {
    for (let i = 0; i < bones.length; i++) {
      bones[i].rotation = this.rotations[i];
      bones[i].position = this.positions[i];
      bones[i].endpoint = this.endpoints[i];
    }
  }
}
```

```
public keyFrames: KeyFrame[] = [];
```

```
public getNumKeyFrames(): number {
  //TODO: Fix for the status bar in the GUI
  return this.keyFrames.length;
}
```

```
public getMaxTime(): number {
  //TODO: The animation should stop after the last keyframe
  return this.getNumKeyFrames() - 1;
}
```

Playback:
In App.ts, we keep track of the elapsed time since the user presses the ‘p’ key and we interpolate between two key frames depending on the elapsed time. Going from one keyframe to another happens in one second so we decide which two keyframes we are interpolating by looking at how many seconds have elapsed, and then we use the decimal value to decide how much to interpolate by using SLERP.

The logic for interpolating between two keyframes is as follows (in the keyframe’s interpolate function). First, we call the interpolate function on the keyframe that is earlier and then pass it the next keyframe that should be next in time in animation, as well as passing the decimal portion of the time for interpolation, and finally we pass in the original positions and endpoints at the time of loading in order to rotate the bones properly. In the interpolate function, we loop through all the bones in the order calculated by the keyframe to be a level traversal, so that at a child bone, we have already calculated the parent’s rotation.

For each bone, we interpolate between the two rotations at each keyframe to get the bone’s current rotation. Then, if the bone has a parent, we calculate the endpoint of the parent bone by creating a vector of the original direction of the bone, by subtracting its original position from endpoint, and then rotating this by the parent’s rotation quaternion, and adding that to the position of the parent. Now that we have the endpoint of the parent, any bone with a parent will have their position set to this value. Then, we will rotate the current bone in a similar fashion, by calculating its original vector by subtracting initial position from initial endpoint, rotating it, and adding that to position to get the endpoint. After all bones have been updated in order, the frame will successfully be an interpolation of the two keyframes.

```
private millis: number;
 private elapsed: number;
 private initPos: Vec3[];
 private initEnd: Vec3[];
```

In init skeleton:

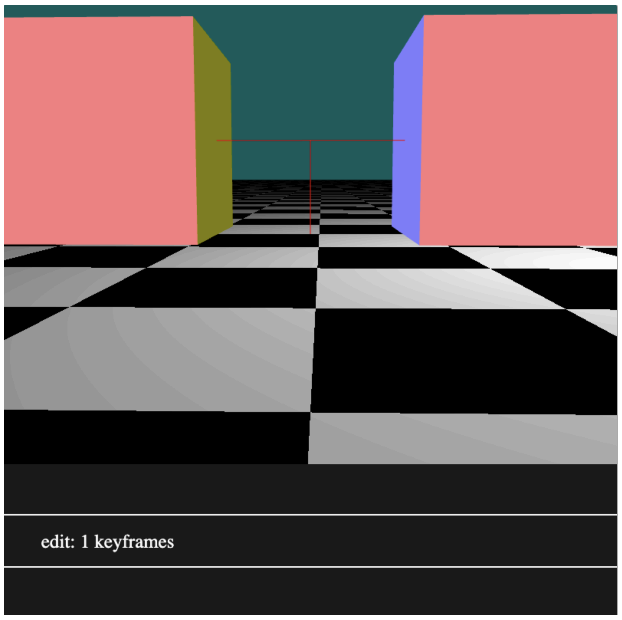
```
this.initPos = [];
   this.initEnd = [];
```

```
for(let i : number = 0; i < this.scene.meshes[0].bones.length; i++){
  this.initPos.push(this.scene.meshes[0].bones[i].position.copy());
  this.initEnd.push(this.scene.meshes[0].bones[i].endpoint.copy());
}
```

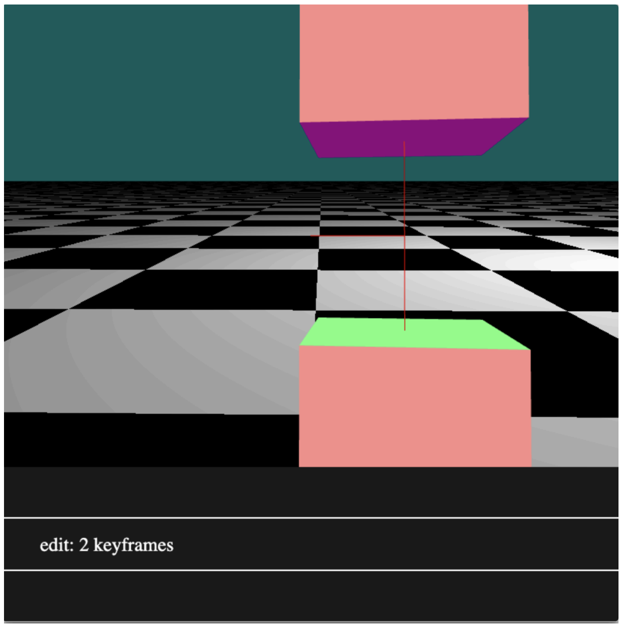
```
// Update skeleton state
   let curr = new Date().getTime();
   let deltaT = curr - this.millis;
   this.millis = curr;
   deltaT /= 1000;
   this.getGUI().incrementTime(deltaT);
   if(this.getGUI().getNumKeyFrames() >= 2 && this.getGUI().mode == Mode.playback) {
     this.elapsed += deltaT;
     let idx : number = Math.floor(this.elapsed);
     this.getGUI().keyFrames[idx].interpolate(this.getGUI().keyFrames[idx + 1],
this.elapsed - idx, this.scene.meshes[0].bones, this.initPos, this.initEnd);
   } else {
     this.elapsed = 0;
   }
```

Pictures / Videos:

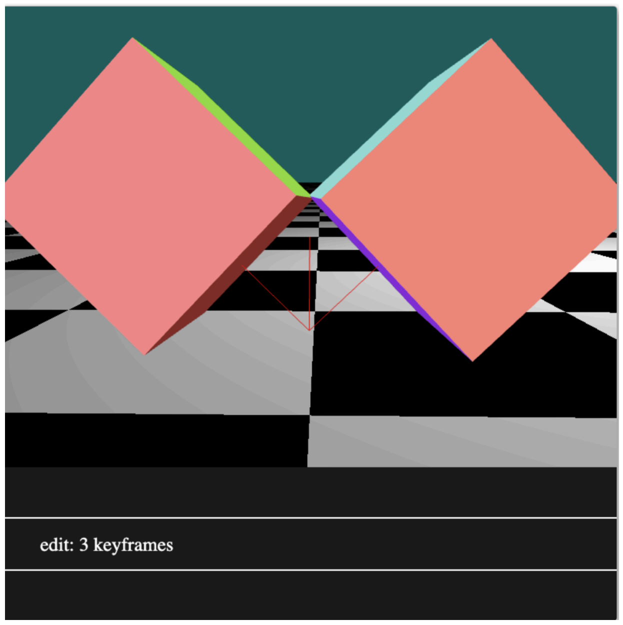
First Keyframe:



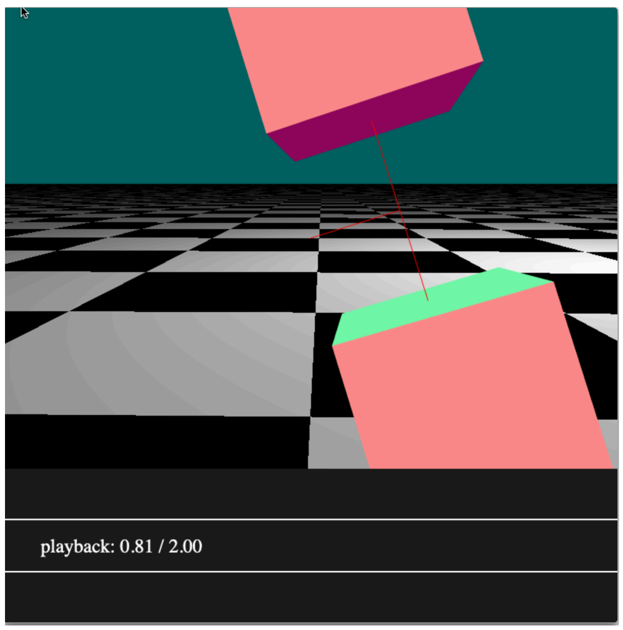
Second Keyframe:

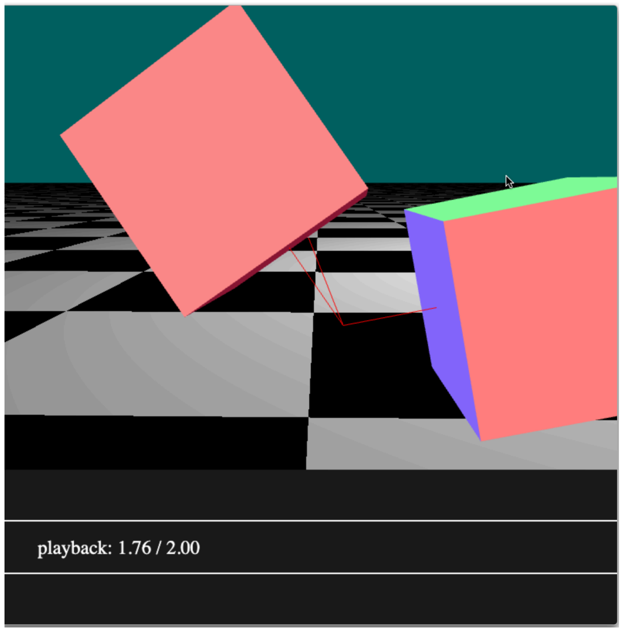


Third Keyframe:


edit: 3 keyframes

Playback with different times:


playback: 0.81 / 2.00


playback: 1.76 / 2.00

Issues we encountered and fixed:

Keyframing:

We had no issues with creating keyframing - it was pretty simple to just implement a data structure that stores the data of the model at the time the keyframe is created, and to create an array that stores all the keyframes for a scene.

Playback:

We encountered an issue where the cage was not being rotated along with the actual bone, but this was due to our approach to time causing problems with the quaternion calculations of endpoints. By changing the time given to slerp to be the decimal value of the time since the animation has started, and then calculating endpoints and positions from initial positions of bones, we fixed this. We also encountered an error where loading certain scenes would crash the program, but this was because of either the selected bone not being cleared upon new load or the traversal being incorrect by not considering multiple bones without parents.

Known Bugs:
We have no known bugs.

Future Work:
We have completed all required aspects of this project and have no future work to do.