

CSCI 491 Project 1

Do you want to parent a snowman's head to its body?

Emily Palmieri

November 21, 2014

1 Introduction

The goal of this research is to create an animation in three.js in which the parent of an object seamlessly changes. To demonstrate this and to prove that a change of parent is occurring, a sufficiently complex animation featuring a snowman with a removable head is created. The tween.js library is used to create the animations. Built-in three.js methods are used to change the parent of the snowman's head at several points in the animation. The side effects of not using these methods properly or not using them at all are also explored and demonstrated in the program created.

In a 3D animation, the parent of an object can change in three ways. Suppose a scene contains a stick lying on the ground and a humanoid character, Anna. The stick initially doesn't have a parent, but it gains one when Anna picks it up. The stick could change parents as well. Suppose Anna hands the stick to another character, Kristoff. The stick has transitioned from being parented to Anna's hand to being parented to Kristoff's. Finally, the stick can be detached from its parent, when for example Kristoff drops it on the ground.

Changing a 3D object's parent is often done in video games and films to create a convincing image of an object following a complex movement. The scene described above wouldn't be very convincing if the stick didn't follow the translations and rotations of Anna and Kristoff's hands. While the animation created for this research isn't as complicated as passing a stick between two animated characters, the animation is complex enough to demonstrate that a change of parent is occurring, or would be difficult to do if it were not.

Calling methods built into three.js isn't all that is required to change an object's parent. The child and the parent's global matrix must be periodically updated or strange anomalies will occur. Demonstrations of these and similar problems with improperly parenting objects have been incorporated into the program as boolean variables that can be toggled on and off.

2 Methods

The first step of this project was to create a snowman that could be animated. The snowman created is built of five parts: a lower body, an upper body, a head, and two arms. The lower body, upper body, and head are cubes, or spheres if the option is toggled on. The upper body can be rotated about the lower body. The arms are attached to the upper body and are rotated about their point of attachment. To manage the snowman's body parts, dimensions, and pivot objects, a class called Snowman, located in javascript/snowman.js, was created.

Next, a looping animation was created. The snowman's head is initialized on the ground in front of the rest of the snowman. The animation created has six major steps.

Step 1 The head floats to the snowman's neck and then to the right hand where it is parented.

Step 2 The snowman's arms are rotated so that its hands and head are directly above the body.

Step 3 The snowman's head moves from the right to the left hand, the arms are lowered to their starting positions, and the snowman's head slides down the left arm to the left shoulder. An alternative implementation of this step breaks these actions, specifically the detachment and attachment of the parents, into two steps.

Step 3A The head is detached from the right hand and moved to the left hand.

Step 3B The head is attached to the left hand and the arms and head are moved as specified.

Step 4 The snowman's head rolls onto the neck and is attached.

Step 5 The snowman's upper body rotates 90 degrees.

Step 6 The snowman's head and body return to their starting positions.

Each major step of the animation is implemented as a method. With the exception of the first method, which only animates the snowman's head, each method begins by attaching and/or detaching the snowman's head from an object. The `createPartTween` method is then called for each snowman part that is animated in the step. This method returns a `TWEEN` object that animates the given part from its current position and rotation to a given final state. Finally, these methods define what to do after all animations in the step have finished. The following example is a simplified version of the second animation step:

```
function animationStep2() {
    var rightArmPivot = snowman.rightArmPivot;
    var leftArmPivot = snowman.leftArmPivot;

    // Parent the head to the right hand
    parenter.parent(rightArmPivot, snowman.head, scene);

    var tweenRightArm = createPartTween(
        undefined, undefined, undefined,
        undefined, 90 * Math.PI / 180, undefined,
        rightArmPivot);

    var tweenLeftArm = createPartTween(
        undefined, undefined, undefined,
        undefined, -90 * Math.PI / 180, undefined,
        snowman.leftArmPivot);
    // Begin the next step when the left arm is finished animating
    tweenLeftArm.onComplete(animationStep3);

    // Begin animating the left and right arm simultaneously
    tweenRightArm.start();
    tweenLeftArm.start();
}
```

The arguments passed to `createPartTween` are the final x-, y-, and z-positions; the final x-, y-, and z-rotations; and the snowman part to animate. Passing `undefined` for any of the first six arguments indicates that the corresponding dimension shouldn't be animated. For simplicity, all animations returned from `createPartTween` are set to last one second. See `javascript/animators.js` to see this method's implementation.

The final step of development was to properly attach and detach the snowman's head to and from other objects. The following code snippets were determined to give the correct results:

```
// Attach child to parent
parent.updateMatrixWorld(true);
child.updateMatrixWorld(true);
THREE.SceneUtils.attach(child, scene, parent);

// Detach child from parent
parent.updateMatrixWorld(true);
child.updateMatrixWorld(true);
THREE.SceneUtils.detach(child, parent, scene);
```

Position description	Parent object	Absolute coordinates	Local coordinates (Relative to parent)
In right hand	Right arm	(500, 500, 0)	(0, -400, 100)
Held above body	Right arm	(0, 800, 0)	(0, -400, 100)
Held above body	Left arm	(0, 800, 0)	(0, 400, 100)
On left shoulder	Left arm	(0, -150, 500, 0)	(0, 50, 100)
On neck	Upper body	(0, 550, 0)	(0, 400, 0)
On neck (after body pivot)	Upper body	(0, 150, -400)	(0, 400, 0)

Table 1: Expected coordinates of the head at every point where its parent changes

Only calling the `attach` and `detach` methods without updating the global matrix of both the parent and the child resulted in the snowman’s head becoming lopsided over the course of the animation. The change of parent when the head was parented from the left hand to the right hand was also jarring and unnatural. To demonstrate the effects of a “naive” implementation, adding and removing a parent was also implemented as follows:

```
// Attach child to parent
parent.add(child);
scene.remove(child);

// Detach child from parent
parent.remove(child);
scene.add(child);
```

To easily toggle between these two methods of adding and removing parents or demonstrate the effects of not updating the object transform matrices, the methods used to attach or detach objects are encapsulated in a class called `Parenter`, located in `javascript/parenter.js`. Two parameters are passed to the `Parenter` class constructor at the start of the program. From these parameters, the `Parenter` object determines what methods to use throughout the program.

3 Testing and Verification

Table 1 shows the expected relative and absolute coordinates of the snowman’s head everywhere its parent changes. These values assume that the snowman’s dimensions are at their default values, the head is parented directly from the right to the left hand, and the `THREE.SceneUtils` methods are used correctly. Calculations were made by examining the snowman’s default dimensions, the positions the head was moved to at different steps in the animation, and the snowman’s hierarchy of objects. For example, when the head is on the neck and parented to the upper body’s pivot point, its relative coordinates are (0, 400, 0), because the head’s center is 400 units above the center of the lower body where the upper body pivot is located, and its absolute coordinates are (0, 550, 0), because its center is 50 units above the top of the upper body. See figures 1 and 2 for diagrams of the snowman’s object hierarchy and default dimensions.

These expected values were compared to the program by using Javascript’s `console.log` function to print the head’s position before and after it was attached to or detached from a parent object. The expected values match those found in the application if with very small precision errors. The y coordinate whenever the head was parented to the arms was also found to be negative. This is most likely an artifact of how the arms were positioned with their y-axes facing down when the object hierarchy was created.

4 Analysis

The `three.js` source code for `THREE.SceneUtils.attach` and `THREE.SceneUtils.detach` listed below is revealing both as to why the global transform matrix for the parent and the child must be updated and why using the `Object3D` class’ `add` and `remove` methods alone don’t produce correct results.

```
detach: function ( child, parent, scene ) {
```

```

    child.applyMatrix( parent.matrixWorld );
    parent.remove( child );
    scene.add( child );
}

attach: function ( child, scene, parent ) {
    var matrixWorldInverse = new THREE.Matrix4();
    matrixWorldInverse.getInverse( parent.matrixWorld );
    child.applyMatrix( matrixWorldInverse );

    scene.remove( child );
    parent.add( child );
}

```

These methods are similar to the "naive" implementation, but a transform is also applied to the child to ensure that it remains at its current location after being re-parented. When the child is detached from a parent, the child's local transform matrix is multiplied by the parent's global transform. The child's new local state will be in relation to the world's origin instead of the parent, but because the parent's transforms have been applied to it, its position doesn't change. Before the child is attached to a parent, its local transform is multiplied by the inverse of the parent's global transform. This undoes any transformations that the parent will apply to the child before it is parented, so the child appears to have not been transformed.

The accuracy of the parent's global transform is imperative in both cases. This matrix is calculated by multiplying the global transform of the object's parent by the object's local transform, or if the object doesn't have a parent, the global transform is the object's local transform. Therefore, if the parent's global transform isn't correct, then the child can't calculate a correct global transform either. If the child never updates its global transform or can't calculate an accurate one before being detached from its parent, then its global state will be some unexpected value.

It isn't clear from the source code when or how often the global transform is updated. It's possible that the anomalies observed when this matrix wasn't updated were the result of it being set to some previous state of the object such as its state in the previous frame.

5 Interpretation

The goal of this research was reached. An object, a snowman's head, was attached to a parent, was directly attached from one parent to another, and was detached from a parent. Though the animation produced isn't physically realistic, the transition between parented states is smooth and unnoticeable. Incorrect methods of changing the parent of an object were also explored.

Future development on this program could include creating a more complete and flexible rig for the snowman and creating a GUI where users can more easily explore the model's hierarchy and methods of attaching and detaching its parts. The simple object hierarchy, or rig, created for this demonstration has two major problems: possible animations are limited and no default positions are defined. The snowman's head, for example, when it is attached to the body, can only be rotated around the lower body and around its own center. A more complex rig could allow the head to rotate around its center, to rotate around the center of the upper body, and to be detached and reattached while maintaining these two rotation methods. These abilities could also be applied to the snowman's arms and upper body.

Because the current implementation doesn't define default positions for each of the snowman's body parts, there isn't a way to easily reattach the snowman's head, or any other body part, at the correct position on the body. An animator wishing to reattach the head, for example, must estimate or calculate the head's correct position on the neck and then parent it to the upper body at that point. Ideally, the animator should be able to parent the head to the upper body at any position in the scene and then zero out all the head's positions and rotations to return it to the neck.

Finally, implementing a GUI where a user can manipulate each body part would make it easier to see how objects behave when they are attached to and detached from parent objects, moved, and rotated.

Upper Body Object Hierarchy

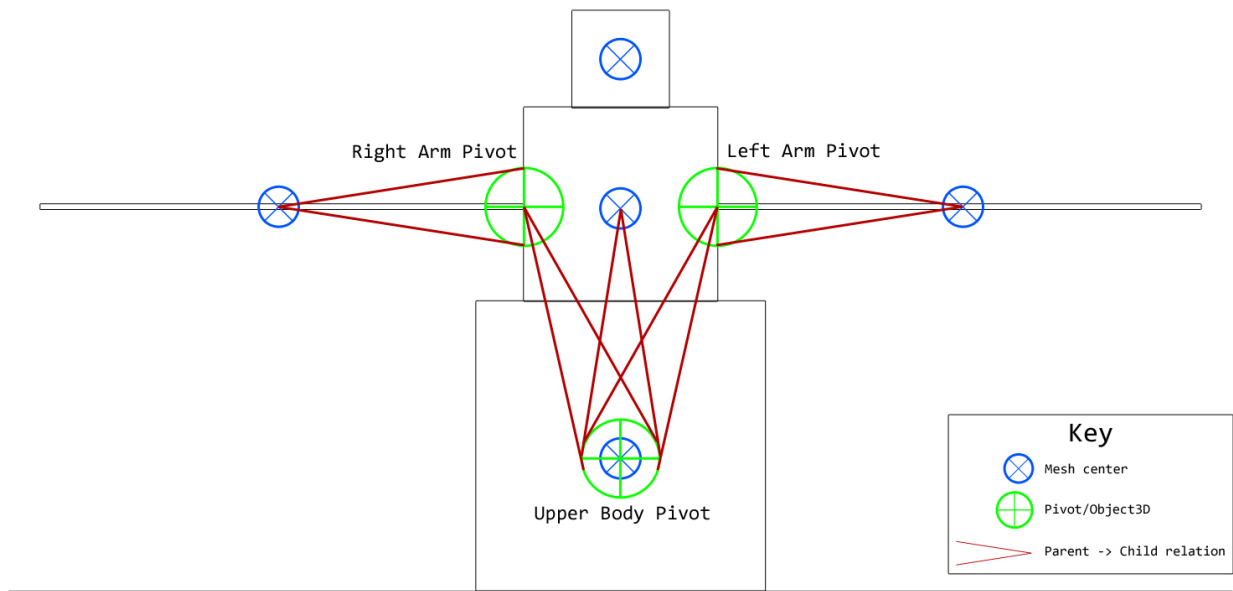


Figure 1: This diagram shows the snowman's upper body hierarchy of objects.

Default Object Dimensions

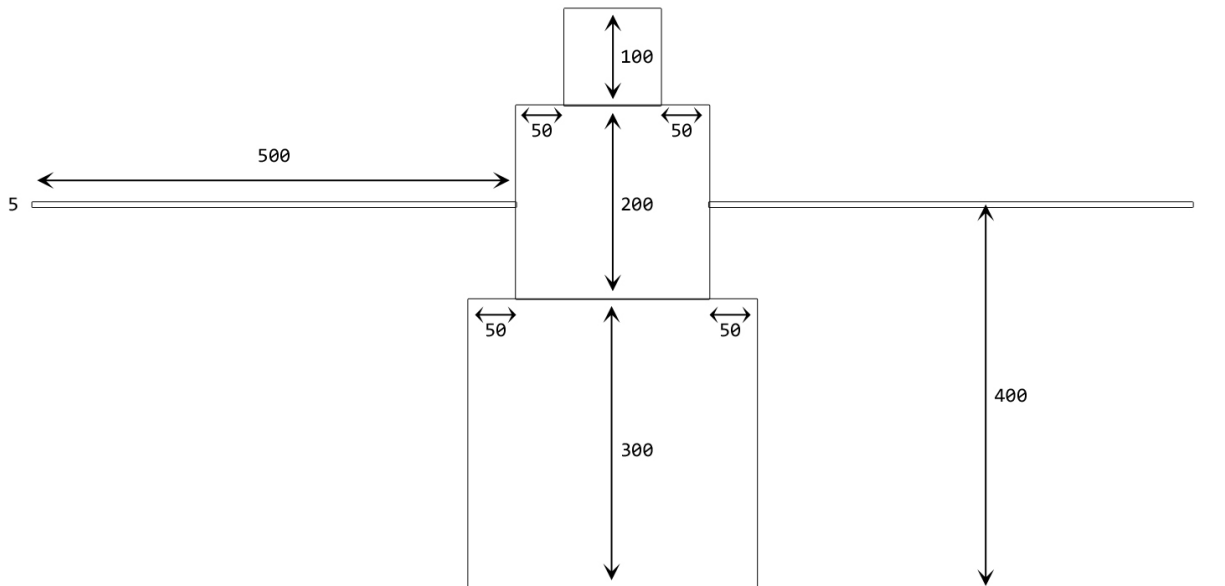


Figure 2: This diagram shows a front view of the snowman's default dimensions. The head and body parts are cubes centered over one another. The arms are cylinders with bases of radius 5.