

# Programming Assignment #2

---

**Due** Oct 1 by 11:59pm      **Points** 4      **Submitting** a website url or a file upload  
**Available** until Oct 4 at 11:59pm

---

This assignment was locked Oct 4 at 11:59pm.

**Due:** Saturday, October 1st (at midnight).

**Late Policy:** Assignment submission will remain open for 3 additional calendar days after the formal deadline. Late submissions will be penalized by a maximum of 1 point on our 4-point scale (i.e. if you submit 3 days late and your assignment was otherwise deserving a "3", you might get a grade as low as a "2" due to this penalty). Some leniency might be exercised on this general rule based on how late the assignment was, and whether turning in an assignment was a repeating occurrence or a one-time event.


**Synopsis:** You will make a program with an object (or more) that uses the concept of hierarchical modeling and have it be animated.

**Learning Objectives:** To see how transformations in 2D and hierarchical objects are useful in modeling and animation, obtain exposure to the implementation of these concepts in the HTML Canvas and the transform stack it implements, and to experiment with using them in web programming.

**Evaluation:** Based on our 4-point grading scheme, as discussed in our introductory lecture. You get a check ("3") if you turn in a viable, and complete submission (even if it just draws a rectangle like the example in the tutorial). "Above and beyond" grades (i.e. a "4") will be awarded for people who have crafted something particularly cool. As a general rule, no more than 1/3 of all assignments turned in (the very best ones, that is) will be considered for a "4" grade.

**Collaboration policy:** This is an assignment to be done individually. Code not written by you needs to include proper attribution (see [this post](#) here). It is always ok to use code provided in our in-class examples as a starting point, but you need to add your own effort to raise those examples (or other sources) to what is asked by the programming assignment (i.e. finding some code on some online forum that does all the job for you that is needed to satisfy the assignment is not the intent, if you haven't added any of your own effort to it). If you use

somebody else's code (other than our GitHub examples), make sure to clarify in your submission notes what **you** did, and what you repurposed from the external source.


**Hand-in:** Electronic turn-in on Canvas. Make sure that you turn in all files needed for your program to run. It is acceptable to turn in a single HTML file with your program, but even preferable to separate your code into an .html file and a separate .js file containing the JavaScript code, similar to the examples in our [GitHub repository](https://github.com/sifakis/CS559F22_Demos)  ([https://github.com/sifakis/CS559F22\\_Demos](https://github.com/sifakis/CS559F22_Demos)) (see, e.g. Demos 0-2 from Weeks 2 & 3). If you submit anything else than a single HTML file, please put everything in a single ZIP archive. ***It is not acceptable to submit a link to JSbin for this assignment!*** For this assignment, we discourage you from using any libraries. But if you do use a library, make clear to mention it in the comment box.

## Description

In class, we've been learning about transformations (in 2D), create composite transforms by combining elementary ones, and the Canvas transform stack. Now is your chance to try these concepts out!

Like the previous assignment, you must make a web page with an HTML5 Canvas on it. You must do all the drawing with Canvas. In fact, you must use the canvas transformation commands (i.e. the **translate()**, **scale()**, **rotate()** methods of the drawing context) as opposed to implementing these transforms via some other process (e.g. adding coordinates together by hand to implement a translation, or using a linear algebra library to implement such operations; the latter is a fine way to implement transforms as a general practice, but for this assignment we want you to use the canvas commands for elementary transforms as much as possible).

Transformations are helpful all of the time. However, they are really useful when you want to make objects that are hierarchical (have parts that move relative to other parts). They are also really useful for moving things around.

An example of a hierarchical model might be a model of a car with spinning wheels: the wheels of the car rotate, but they stay attached to the car as it moves around. Another example would be a quadcopter as demonstrated in the lecture – with 4 propellers that spin while the copter flies around. Since we have only worked in 2D so far, consider how this would look in a top view ([link](http://graphics.cs.wisc.edu/Courses/559-f2015/Examples/QuadCopter/quad.html)  (<http://graphics.cs.wisc.edu/Courses/559-f2015/Examples/QuadCopter/quad.html>)). Note: we are not hiding the source code of this

implementation from you, but you must make your own object and motion. *This particular example is **not** intended to be an example of how to code up such models in JavaScript/Canvas, and you are strongly recommended to not use it as a starting point for your implementation. This is merely supposed to be a visual illustration of how a moving hierarchical object might look like.* We also gave examples of hierarchically modeled objects in class, such as a simple articulated robotic chain ([link](https://jsbin.com/wovupusife) ⇨ <https://jsbin.com/wovupusife>). It is **ok** to use these code samples (the ones where we explicitly described the code during class; many/most of these will also be duplicated in our GitHub repository of demos) as starting points in your implementation, or to get some inspiration to get started. Again, you should deviate from these examples by adding your own shapes, connectivity of pieces, and/or motion.

Your implementation *must* demonstrate the use of the HTML Canvas transform stack, in service of the hierarchical modeling concept. If you're not using the `save()`/`restore()` commands in a way similar to how we did in class, you are probably doing this wrong.

We would encourage you to be creative! Pick something interesting. It just has to have parts that move relative to each other (e.g. be hierarchical). And it should have at least one part that has two children (like the quadcopter has 4 propellers, or a car has 2 wheels). And it has to move by itself to show this off.

You do need to animate your object (or objects) in a way that shows off the hierarchy.

The [QuadCopter](http://graphics.cs.wisc.edu/Courses/559-f2015/Examples/QuadCopter/quad.html) ⇨ <http://graphics.cs.wisc.edu/Courses/559-f2015/Examples/QuadCopter/quad.html> demo flies around and its 4 propellers spin. One possibility is to use “`requestAnimationFrame`” to do the animation (or `setTimeout`). See the [tutorial](http://graphics.cs.wisc.edu/WP/tutorials/when-do-i-draw-some-comments-on-code-organization/) ⇨ <http://graphics.cs.wisc.edu/WP/tutorials/when-do-i-draw-some-comments-on-code-organization/>, especially the last example. It is also OK to use sliders to control the animation, as we have seen in class (instead of the auto-update via the `requestAnimationFrame` mechanism) if that works best with your particular hierarchical model.

Some ideas (but you should be creative!)

- You could make a car (side view) where the wheels spin while it drives. Or make it a dump truck and the back can tilt up and down. Or a firetruck where the ladder goes up and down.
- You could make a tree where the branches wave in the wind.
- How about a clock with two hands (minute/hour), where the clock itself is moving. Maybe have the clock mounted on some other object (a character's hand?), or have several of

them in your scene.

- A model of the solar system, with planets and their moons?

To compete for a “4” (above-and-beyond) grade we suggest that you model objects with interesting motion, have some added complexity to the hierarchical structure of your object (say, something more than a Y-shaped robotic arm with 2 fingers ...), or use components in your model that are particularly creative.

## Readings (new!)

Although we will try to make the class lectures and slides be as descriptive as possible, and have them be your primary reference for many of the concepts discussed in class, we will also provide supplemental reading materials that help build more sound foundations for the theoretical concepts discussed in class. Although reading these in great detail is not an absolute necessity for completing the programming assignments, we believe you would benefit from doing so; some of these will also prepare you for the lectures ahead of us!

At this point, we recommend that you review the following:

- Readings about the basic math (Linear Algebra and coordinate systems)
  - [Tutorial: Points and Vectors](http://graphics.cs.wisc.edu/WP/tutorials/points-vectors-and-coordinate-systems-why-are-points-and-vectors-different/) ⇨ [\(http://graphics.cs.wisc.edu/WP/tutorials/points-vectors-and-coordinate-systems-why-are-points-and-vectors-different/\)](http://graphics.cs.wisc.edu/WP/tutorials/points-vectors-and-coordinate-systems-why-are-points-and-vectors-different/)
  - Chapter 1 of Hart's Big Fun Computer Graphics Book ([Link](#))
  - Chapter 5 of Hart's Big Fun Computer Graphics Book ([Link](#))
- If you need to brush up on linear algebra, you can try chapters 1,2 and 4 or Practical Linear Algebra ([Link](#), [Link](#), [Link](#))
- Foundations of Computer Graphics (Chapter 6, section 1 [Link](#)) discusses transformations in a manner similar to Hart, if you want to see it another way.
- Reading ahead to Chapter 7 of Hart's Big Fun Computer Graphics Book ([Link](#)) will give you a sense of why we're so interested in transformations.
- There is a highly recommended [tutorial about types of graphics \(image vs. objects\)](http://graphics.cs.wisc.edu/WP/tutorials/image-based-graphics-vs-object-based-graphics/) ⇨ [\(http://graphics.cs.wisc.edu/WP/tutorials/image-based-graphics-vs-object-based-graphics/\)](http://graphics.cs.wisc.edu/WP/tutorials/image-based-graphics-vs-object-based-graphics/) so you know why we're starting where we are.