

CMSI 371-01

COMPUTER GRAPHICS

Spring 2016

Assignment 0204

This initial assignment is meant to get you into a development groove with 2D canvas graphics, as well as mark your first step toward an animated 2D scene. Note that for outcomes that cover proficiency in both 2D and 3D, we stick to a maximum of | for now.

Outcomes

This assignment will affect your proficiencies for outcomes *1a*, *2a* (max |), *3a* (max |), and *4a–4f*.

Not for Submission

Don't limit your canvas exposure solely to the functions/properties that you specifically use in your drawings—spend some “big picture” time with the MDN canvas tutorial and reference pages (links available at the course website). You might be pleasantly surprised by how much you can do.

For Submission

First, envision an animated 2D scene that you'd like to render. Storyboard or script it so that you can get a concrete idea of what you'd like to see.

Then, write at least three (3) canvas functions that draw three distinct objects in your scene. Known as *sprites*, they can be things, characters, vehicles, buildings, whatever you like. We say “at least” because, really, the more you do, the better you'll get, and we don't want to artificially limit your practice time (or your creativity). Give your sprites some internally movable characteristics like limbs or facial expressions (see the next section).

Use standard control structures like loops and conditionals as needed; don't feel limited to just canvas functions and properties. We are assuming that you can figure these out in JavaScript on your own, but if you're really stuck then ask me.

Consult the MDN canvas website extensively so that you don't miss out on its full feature set.

Parameterize the Sprites

Your functions' drawing commands should be *parameterized by a separate model object*. That is, your functions should accept an object parameter whose properties affect how something gets drawn. For example, if you decide to draw a cartoon character,

you can supply an object with a property that states how open or closed its eyes are. Or, you might draw a box with a hinged lid, and its object might state how open, in degrees, the lid should be.

Draw Around the Origin, Transform to Test

One set of characteristics that you *shouldn't* parameterize is whole-object movement, rotation, or scaling. These can be done independently by using the `translate`, `scale`, and `rotate` functions.

Center your drawing code around the origin. You can invoke `translate` before calling your sprite function to draw it at the desired location. Similarly, `scale` and `rotate` will resize and turn your entire sprite, respectively. You may play with these in your demonstration pages (described next).

Standalone and Combo Demonstrations

Define your sprite functions in separate JavaScript files (e.g., *lion.js*, *clown.js*, and *circus.js*). Because these functions are intended for reuse, yes, this time you are allowed to define them within a top-level-scope container object. For encapsulation, it remains recommended that you define the functions inside an anonymous function that is called immediately.

Test/demonstrate your functions by (1) creating one HTML file per sprite, using an inline `script` element to call your sprite functions, and (2) creating a *combo.html* file which loads all of your sprite scripts and draws them all on a single canvas.

In case you're keeping score, this all means that, if you define n sprites, you should end up with n JavaScript files and $n + 1$ HTML files.

How to Turn It In

Commit your work under *sprites*. Remember that “committing” doesn't just mean “submitting,” but progressively saving what you do so that you can recover prior code as needed! And, don't forget those descriptive commit messages!