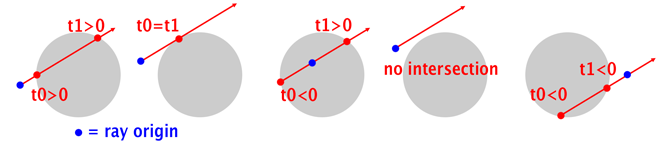
**Ray Intersections Project Guide**

****

### **0. Guidance**

#### **Learning Javascript**

The basic features of the Javascript language are relatively simple and sufficient for assignments in this course. To get a feel for the syntax, see [learnxiny](https://learnxinyminutes.com/docs/javascript/). You may also find [Javascript for People Who Know Python](https://www.youtube.com/watch?v=GAoheEUiwwY) useful.

#### **Debugging**

I highly recommend spending a little time gaining proficiency with your tools (i.e. Chrome or Firefox). Investment in debugging skills pays huge dividends… Make sure you

1. **Understand how to set breakpoints in your code**
2. **Step through code execution line by line, and**
3. **See the values of your variables at any point in time** (without “print” statements).

See [**Debugging in chrome**](https://developers.google.com/web/tools/chrome-devtools/javascript/).

#### **Getting Help**

Should you find yourself spending an excessive amount of time without progress, please utilize the following:

1. Your partner
2. The discussion boards
3. Email ([micguerrero@csumb.edu](mailto:micguerrero@csumb.edu) or the TA).

#### 

#### **Clarifying / Improving this Guide**

If something printed here is unclear to you or seemingly wrong, please leave a comment on the side of page using the commenting feature of Google Docs. It should be as simple has highlighting something, right clicking, and selecting comment. You can see a demonstration of this feature [**here**](https://www.youtube.com/watch?v=RC6LyeigAFs).

### **1. Overview**

In this programming project you will be implementing the API for an implicit sphere. The code for this will serve two functions:

1) fully define the shape in space and

2) determine the **details** of specific possible ray intersections.

**Details** include if an intersection exists (*Boolean*), the closest point at which it happens (*Vector3*), and the normal (*Vector3*) of surface at that point.

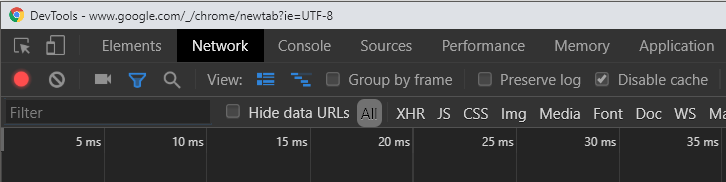
### **2. Common Mistakes**

As you proceed through this assignment, take note of the following commonly made mistakes:

* Not using the “var” keyword before declaring a variable. Javascript will allow you to get away with many things that other languages won’t. In this case, the language will treat your variable as a global which could be very confusing if not intended. For example, declaring variable foo in a function will make foo available everywhere and any change made to it will be permanent.
* Browser caching previous code. **This will happen to you!**

Sometimes you will make changes to your code and the web page will not reflect those changes on refresh due to caching. Here are a couple ways to get around this.

* Hold shift and click the refresh button (chrome, firefox?)
* Make sure the dev tools are open and “disable cache” is selected



* Other Javascript-isms. See [here](https://docs.google.com/document/d/1Ql7-U9bKUf94Wl26uXbGij0MfYvwUy_GxS9_3kWmTvw/edit?usp=sharing).

### **3. Project Files**

You should begin this project with the following files:

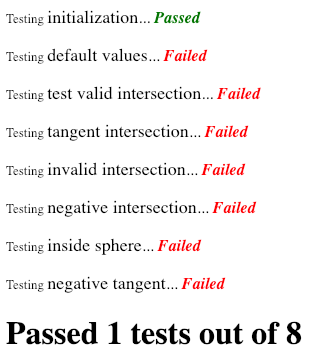
**math/ray.js**

**test/test.js**

**test/test-sphere.html**

### **4. Setup**

To start, run test-sphere.html and you should see the following:



This is the result of automated testing of your code. Initialization is the only test that currently passes and it does so by making sure that the arguments passed into the constructor of Sphere are applied to member variables.

|  |
| --- |
| var Sphere = function(origin, radius) {  this.origin = origin;  this.radius = radius; |

### **5. Implementation**

More detailed information about how to code these are found inside of the *sphere.js* file itself. For more of a conceptual understanding of the code and its conventions, read the todos below.

#### **Todo #1 Default Values**

Take a look at the code for *testDefaultValues()* in *test-sphere.html*. It creates a Sphere without passing any parameters. Make sure you handle this case by setting *origin* to the zero vector and *radius* to 1.

#### **Todo #2 Test Valid Intersection**

This is the first basic test case where 2 intersections are expected (we only care about the nearest one). The setup looks as follows:

|  |
| --- |
| var r1 = new Ray(new Vector3(0, 0, -10), new Vector3(0, 0, 1)); var s3 = new Sphere(new Vector3(), 1); |

Imagine the XZ plane as the ground and Y axis as pointing upward (doesn’t actually matter here but it helps to visualize). The ray starts along the z-axis at -10 and proceeds forward in the positive z direction. The sphere is located at the world origin and has a radius one. So we would expect the ray to intersect through the middle of the sphere (once through the front and once through the back). We get 2 answers because the math produces all valid solutions that satisfy the ray equation being equal to the sphere equation. See the test code to see further details.

#### **Todo #3 Test Tangent Intersection**

This covers the edge case where a ray skims the edge and “intersects” at precisely one location. A tangent intersection is the only time there will be only 1 intersection point.

#### **Todo #4 Test Invalid Intersection**

A common result of our intersection test is that a particular ray will not intersect the sphere at all. This function tests for this.

#### **Todo #5 Test Negative Intersection**

This test uses values that would intersect in the negative direction and ensures your code does not return a valid hit. A ray has a particular direction but the math doesn’t care, it will give you an answer for (albeit negative) intersections in the opposite direction.

#### **Todo #6 Test Inside Sphere**

Should the ray originate inside the sphere, you will get one negative and positive result. However, this case should not be treated as valid as we will not be interested in detecting the inside of a sphere. **Hint** - think about how far your ray’s origin is from the center of the sphere.

#### **Todo #7 Test Negative Tangent**

This is similar to the previous test which checked negative intersection but this one is the special case where you only have one (tangent).

#### 

#### **Todo #8 Test Normal**

Once we’ve intersected our ray with the surface, it will be useful to know a little bit more about the geometry at the point of intersection. Specifically, we want to know which direction the surface is facing at the point. This direction is known as the “normal”. You can also think of it as the direction that is tangent to the surface. The required ingredient to generate this are the intersection point and the center point of the sphere. Remember your vector maths and think it though. What operations can you do with those two ingredients to get the answer?

### **6. Grading**

You will be given credit for each function that passes its test for a total of 50 points.