CMSI 371-01

COMPUTER GRAPHICS

Spring 2015

Assignment 0326b Feedback

Outcome *3a* now covers enough of the overall graphics library to merit a full proficiency range. With instance transforms, outcome *3d* now covers the full envisioned vertex shader, and also drops the proficiency cap even with the fragment shader remaining.

Daniel Levine *dannymlevine / dannymlevine@gmail.com*

1. *** Shader shows no sign of matrix library use. (*2a*, *2b*, *3a*, *3d*, *4a*)
2. We'll see what the rest of the code reveals, but in general, this function should no longer be here. (*4b*)
3. When adapting sample code, do give the whole thing a once-over. That way you don't miss really simple tweaks like the page's title and headings. (*4b*)
4. There is some ambiguity here…do these functions *perform* the transform on the given matrix, or do they *set* the matrix to that transform? I will guess the former, because otherwise it would not be meaningful to do a translate on the matrix followed by a scale. But still…the resulting multiplication does not look right. I have a suspicion about how you implemented `translation` and `scale`…I'll have to see if my suspicion is correct when I look at *matrix.js*. (*2a*)
5. Suspicions in note #4 aside (and that will be cleared up shortly), the coverage here is decent but needs a few more cases for generality. (*4a*)
6. *** Alas, as I suspected, the `translation` and `scale` functions are being done incorrectly. What these functions are doing are *replacing* the supposedly "significant" elements of the target matrix. However, that is incomplete. Just because the translation and scale parameters in the basic matrix go directly into specific cells *does not mean that the other cells aren't affected*. If, for example, you perform `scale` on a matrix that already has a translation in it (i.e., the 4th column first row is not zero), *your code will fail to scale that translation term*. The result will be a hodgepodge matrix that ultimately will not perform the operation that is expected had a *full matrix multiplication* been done. That full matrix multiplication is the *correct* way to apply these transforms; you don't just plug in the values. (*2a*, *3a*, *4a*)
7. See, you did port the rotation function to your matrix library. Thus it should have been removed from the scene code, as indicated in note #2. (*4b*)
8. Also, why do these function declarations have a name? Those are completely superfluous and potentially confusing. If you are assigning functions to variables or properties, don't include a name. (*4b*, *4c*)
9. OK, so you built the projection matrices. You should have used them in your scene. (*2b*, *3d*, *4a*)
10. *** And here we have another major issue (and a likely reason for why you might have hit a dead end): this code isn't defining a JavaScript object with methods properly. How to do this is too long to explain here; check the JavaScript textbook chapter on objects for that, or just look in the web. The short version is this: creating an object involves (a) defining a function that acts as a constructor; calling that function with the `new` keyword will create the new object. (b) Functions that are intended to be methods (i.e., to be called using dot/period notation on an object expression) should be assigned to the constructor function's *prototype*. This is a core mechanism of JavaScript object behavior; if you have not learned about it, now is the time to do so. (c) To finally create an object, invoke the constructor function with `new` (e.g., `var m = new Matrix(4);`), then perform computations on it using its methods or other functions. For closure, `Matrix` methods should also return a `Matrix` result. (*3a*, *4a*, *4d*)
11. *** Another major missing feature from the assignment is the implementation of instance transformations. This will allow you meshes to be moved around easily. There is no sign of code for this functionality in the submission. (*2a*, *3d*, *4a*)

CMSI 371-01
Computer Graphics
Spring 2015

Assignment 0326b Feedback

Outcome *3a* now covers enough of the overall graphics library to merit a full proficiency range. With instance transforms, outcome *3d* now covers the full envisioned vertex shader, and also drops the proficiency cap even with the fragment shader remaining.

2a — / …Transforms are created, but not correctly, and they are not used.

2b — | …The projection matrices look better, but are still not used.

3a — / …The core functionality is in there, but the object structure is missed.

3d — / …Despite the object issues, the existing code still could have been used effectively in the 3D scene, but it is not.

4a — / …The incorrect implementation of the transformation matrices is the greater issue here than the incorrectly-defined `Matrix` "object"—it belies a fundamental misunderstanding of how matrices work. The missing object is instead reflected in…

4b — / …Yes this is mainly the mis-structured `Matrix` object.

4c — +

4d — | …The `Vector` example could have been referenced as a model for making JavaScript objects, in addition to the JavaScript textbook and many sources on the web.

4e — +

4f — / …Matrix work was done around two weeks after the due date.

Updated feedback based on commits up to 2015-04-30 *(no written notification was given on this resubmission, but as a courtesy I will throw these in because I happened to notice them when grading Assignment 0430a and 0430b)*:

Raw matrix code is certainly present now, and some of it is used in the scene code; that is a core improvement. Beyond that, however, gaps remain, and they tie in with the Assignment 0326a issues. Overall there is a disconnect with how objects are used in general and how they should be implemented in JavaScript in particular. This is clearest in the existing unit test failures—the core issue here is that the code confuses `matrix` *instances* with the central `matrix` *object* that effectively represents the "class" of all matrices. It is best to point out the issue by example: the `multiplication` function does it right; the other functions do not. That is the core issue, and is at the heart of the unit test failures and other problems in the code itself. When you get the chance, you should definitely review object-oriented programming in JavaScript. Go back to the `vector` example if needed, in case you learn better from sample code rather than conceptual material.

Other lingering issues include problems with the instance matrix implementation: sure, there is something there, but it does not work correctly; one example is the `translation` portion, which (as you probably know because your code does not use it at all) distorts the object instead of moving it. Another issue is the unusual global-level `rotationMatrix`, `scaleMatrix`, and `translationMatrix` variables—those are redundant to the instance transform (once that works correctly) and even then, they are applied incorrectly in the shader.

Still, this is a decent move up, and worth upgrades in some outcomes:

2a, *3a*, *3d*, *4a* — | …Acknowledging the progress seen while noting lingering issues.

2b — + …This is focused on projection, and that is sufficiently addressed.

4b, *4d* — /, | …These remain where they are because the remaining issues pertain to them the most.