PA3 – Math Quaternions (3.0)

Student Information

Integrity Policy: All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies: (•) Yes ( ) No

Name: Joey Domino

Date: 10/22/2022

Submission Details

Final ***Changelist*** number: 157696

Verified build: (•) Yes ( ) No

Number Tests Passed: 503

Required Configurations: DEBUG only

Discussion (What did you learn):

I learned math. a LOT of math.

Verify Builds

- Follow the Piazza procedure on submission
  - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
  - No – Generated files
    - *.pdb, *.suo, *.sdf, *.user, *.obj, *.exe, *.log, *.pdb, *.db, *.user
    - Anything that is generated by the compiler should not be included
  - No – Generated directories
    - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
  - *.sln, *.cpp, *.h
  - *.vcxproj, *.vcxproj.filters, CleanMe.bat

Standard Rules

Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
  - As soon as you get something working, submit to perforce
  - Have reasonable check-in comments
    - Points will be deducted if minimum is not reached

Write all programs in cross-platform C++

- Optimize for execution speed and robustness
- Working code doesn't mean full credit

Submission Report

- Fill out the submission Report
  - No report, no grade

Code and project needs to compile and run

- Make sure that your program compiles and runs
  - Warning level ALL …
  - NO Warnings or ERRORS
    - Your code should be squeaky clean.
  - Code needs to work "as-is".
    - No modifications to files or deleting files necessary to compile or run.
  - All your code must compile from perforce with no modifications.
    - Otherwise it's a 0, no exceptions

Project needs to run to completion

- If it crashes for any reason…
  - It will not be graded and you get a 0

No Containers

- NO STL allowed {Vector, Lists, Sets, etc...}
  - No automatic containers or arrays
  - You need to do this the old fashion way - ***YOU EARNED IT***

Leave Project Settings

- Do NOT change the project or warning level
  - Any changing of level or suppression of warnings is an integrity issue

Simple C++

- No modern C++
  - No Lambdas, Autos, templates, etc…
  - No Boost
- NO Streams
  - Used fopen, fread, fwrite…
- No code in MACROS
  - Code needs to be in cpp files to see and debug it easy
- ***Exception:***
  - implicit problem needs templates

Leaking Memory

- If the program leaks memory
  - There is a deduction of 20% of grade
- If a class creates an object using new/malloc
  - It is responsible for its deletion
- Any ***MEMORY*** dynamically allocated that isn't freed up is ***LEAKING***
  - Leaking is ***HORRIBLE***, so you lose points

No Debug code or files disabled

- Make sure the program is returned to the original state
  - If you added debug code, please return to original state
- If you disabled file, you need to re-enable the files
  - All files must be active to get credit.
  - Better to lose points for unit tests than to disable and lose all points

No Adding files to this project

- This project will work “as-is” do not add files…
- Grading system will overwrite project settings and will ignore any student’s added files and will returned program to the original state

UnitTestConfiguration file (if provided) needs to be set by user

- Grading will be on the UnitTestConfiguration settings
  - Please explicitly set which tests you want graded… no regrading if set incorrectly

Due Dates

- See Piazza for due date and time
- Submit program perforce in your student directory assignment supplied.
- Fill out your this ***Submission Report*** and commit to perforce
    - ***ONLY*** use Adobe Reader to fill out form, all others will be rejected.
    - Fill out the form and discussion for full credit.

Goals

- Math Quaternions
    - Add Quaternions to the existing Math library
- Create the Quaternion, Vector and Matrix classes for the math engine
    - Use a Unit Test environment
- Reverse engineering
    - Debug and test your programming skills

Assignments

1. Create a stand-alone static Math Library for Game Engine
    - Write a Vector Class
    - Write a Matrix Class
    - Write a Quaternion Class
    - Create any necessary classes and code to complete the library
    - Document the code
    - Code should be warning free
    - 2 configurations:
        - x86 Debug/Release
2. Unit tests is large… don't be intimidated
    - Over 500 unit tests
    - Over 10000 individual checks
    - The unit tests are in Beta
        - There are missing tests, a correct library will pass all tests
            - Unfortunately there may be false positives
            - Just because they all pass, the library may not be behaving 100% correctly.
                - You are graded on the existing unit tests
        - Additional unit tests will evolve as we discover more issues

3. Implement the functions and API
    - You have freedom to implement the code in any way or style you desire
    - Keep it clean and neat, with reasonable documentation
    - You need to use the new Vect, Matrix, Quaternion data definitions
    - Data must remain in the private section

4. Unit test framework is provided
    - All tests should run for full credit
    - You cannot modify the tests given

5. Const correctness and prototypes will be reviewed
    - Try to use references
    - Try to use const as much as possible

Validation

Simple checklist to make sure that everything is submitted correctly

- Make sure program build without errors or warnings
- Project should be able to run without crashing
- Set the _UnitTestConfiguration.h to the tests you want graded
    - DO NOT change the project setting to exclude any files
- If all of the tests run and are correct, you should be in great shape
    - Over 500 tests

Hints

Most assignments will have hints in a section like this.

- Do this assignment by iterating and slowly growing your project.
    - Use the test code for your use cases
- This is a fairly easy assignment, but tedious.
    - Look at the lecture slide for the math.
        - There are websites with additional math or ask to the forum
- The hardest part might be figuring out the API
    - Reverse engineer the API from the test code.
        - ***consts*** might give you troubles, so be aware of the different ***consts***
- Use *Keenan's modification to Quaternions*…
    - The tests are expecting that implementation

Troubleshooting

- Your tests will not run at first since you haven't implement code yet.
    - Focus on the first tests, getting the constructor and bracket operator working.
    - Copy your old implementation from Math 2.0 into this project
- I would comment in one test group, then focus on the one test at the time
    - Baby steps - Incremental development!
- You are learning 2 things, test-driven development and reverse engineering.
    - Both are very useful, but require a little thinking.
- Please
    - Step through the code to help you understand
    - Use Piazza