

## Particles – Project

### 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:

Date:

### Submission Details

Final **Changelist** number:

Verified build:                      Yes                      No

(everyone)      Logs.pdf:

(grads) Analysis.pdf:

Discussion (What did you learn):

## 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

### Containers - ALLOWED

- You can have STL containers and arrays ...
  - Be careful – might be a performance slowdown

### Project Settings Modifications - ALLOWED

- You are permitted to change settings... Keep Level 4, be careful on SIMD level

### 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

### Adding files to this project - ALLOWED

- Just be careful to add properly and submit the correct files
  - NO extra files

### UnitTestFixture file (if provided) needs to be set by user

- Grading will be on the UnitTestFixture 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 performance in your student directory assignment supplied.
- Fill out your this **Submission Report** and commit to performance
  - **ONLY** use Adobe Reader to fill out form, all others will be rejected.
  - Fill out the form and discussion for full credit.

## Goals

- Optimized a real world project
  - Refactor and improve an existing project
- Do not change any behavior
  - Number of particles, timing, randomness

## Assignments

Please **VERIFY** the correct builds for each project

- **Optimize for execution speed.**

### Description

- Given a particle system, that dynamically updates several particles. Every particle and controlling object is dynamically allocated, often in very large bloated data structures. Each particle is controlled per frame by its own unique math transformations that are not optimized. The memory allocations in the system are slow and fragments memory as the number of particles increase. Some of the systems uses STL in a very inefficient manner. Many C++ classes are inefficient and naïve in nature.

### Students refactor the project:

- Maximize the number of particles to be processed with the given memory and performance constraint
  - Try to keep the frame rate (update) constant
  - Try to keep the memory within a fixed specified size limit
- Real working system
  - Program needs to launch and close cleanly
  - Error free
    - Warning level 4 free or higher
- No memory Leaking allowed
  - If you leak 20% deduction
- It's a contest
  - See how you can improve to original system.

**Allowed to change compiler settings:**

- Do not exceed SSE4.1
- Only use MMX, SSE
  - no AVX, VEX instructions

**Submission date**

- ONLINE students pay attention to the dates
  - The particle system is due before by 5:00pm of the “official in-class” final exam day schedule.

**Particle system**

- I will be running on the teaching machine at school
  - I will run only in Release mode
- I am ignoring warnings...
  - so you can disable treat warnings as errors compiler flag if you want
  - Keep warning level at 4... but you can run with warnings
- Once the class is given the exam, I will download that class' Particle system and grade
  - Your particle system will be graded together (online and inline students)

**Required:**

- **Graduate Students - Analysis Paper**
  - Graduate students must perform an optimization analysis for the particle system before they work on the system.
  - Write a 2-3 pdf paper detailing the current performance and a plan to improve the performance plan white paper.
    - Name the paper:
      - Analysis.pdf
      - Submit in the same directory as the solution file (GameParticles.sln)
    - Highlight your PLAN with expected improvement
- **Log paper (Everyone)**
  - Everyone needs to submit your log to perform:
    - Submit as a PDF file.
  - Defining each major optimization and it relative performance
    - Include Dates and Changelist number
    - Expected length 2-3 pages in pdf format.
  - Name the file:
    - Logs.pdf
    - Submit in the same directory as the solution file (GameParticles.sln)
- **Use the Submission report to help with logs....**
  - See next page

**Sample log:**

- Changelist: 90863 ( Saturday Nov 5 ) date stored in changelist automatically
  - Added const to Vect4D
    - Saved 0.25 ms in release
  - Reworked Matrix to use references instead of pointer
    - Saved 0.05 ms in release
  - Added += operator in Vect4D
  - Reworked code to use this everywhere
    - Save 0.1 ms in release
- Changelist: 90872 ( Monday Nov 12 )
  - Added SIMD to Vect4D
    - Saved 2.5 ms
  - Cursed Keenan's Name out loud
    - His comments in code were wrong
    - Chased a red herring for 4 hours
  - Sutter come-on give a little more help.

**Grading**

- Project MUST build and Run
  - You cannot change the OpenGL code ( gl\*\*\*\*\* functions)
  - Exception: allowed to change data format types on GL calls
    - double to float for example
- Ignoring warnings... if it builds (in Release mode) that's good enough
  - You can change compiler settings... SSE4.1 or lower is allowed
- Grading Ratio
  - I will run the original and each students particle system in release
  - I will grade on performance ratio:
    - For example
      - if Original = 66ms, student's = 33ms
      - Ratio:  $66/33 = 2.0x$  faster
- The original behavior needs to be preserved
  - Number of particles and lifetime needs to be set to the original for grading.

Grading TABLE:

**Performance Ratio:**

> 4.00 :	10.0 pts
> 3.75:	9.5 pts
> 3.50:	9.0 pts
> 3.25:	8.5 pts
> 3.00:	8.0 pts
> 2.75:	7.5 pts
> 2.50:	7.0 pts
> 2.25:	6.5 pts
> 2.00:	6.0 pts
> 1.75:	5.5 pts
> 1.50:	5.0 pts
0 - 1.50:	0.0 pts ←----- NOTE you have to 50% improvement for any points

Validation

*Simple checklist to make sure that everything is submitted correctly*

- Is the project compiling and running without any errors or warnings?
- Does the project run **ALL** the unit tests execute without crashing?
- Is the submission report filled in and submitted to performe?
- Follow the verification process for performe
  - Is all the code there and compiles “as-is”?
  - No extra files
- Is the project leaking memory?

Hints

Most assignments will have hints in a section like this.

- Do many little check-ins
  - Iteration is easy and it helps.
  - Performe is good at it.
- Use piazza forums
  - Share ideas... but not that many
    - You want to win the prize!