6.S096 Lecture 8 – Project Environments

Iterators, N-Body Problem, Setup

Andre Kessler

Outline

Sample Solution

```
class List {
  size_t _length;
  ListNode *_begin, *_back;
public:
  class iterator {
    friend class List;
    ListNode *_node;
  public:
    iterator( ListNode *theNode );
    iterator& operator++();
    int& operator*();
    bool operator==( const iterator &rhs );
    bool operator!=( const iterator &rhs );
  }; // ...etc
```

Sample Solution

```
iterator begin();
iterator back();
iterator end();
```

Iterators allow us to write a fast reduce function like this:

```
int ReduceFunction::reduce( const List &lis ) const {
  int result = identity();
  for( auto it = lis.begin(); it != lis.end(); ++it ) {
    result = function( result, *it );
  }
  return result;
}
```

Iterator Implementation

```
List::iterator::iterator( ListNode *theNode ) :
  _node{theNode} {}

List::iterator& List::iterator::operator++() {
  _node = _node->next();
  return *this;
}

int& List::iterator::operator*() {
  return _node->value();
}
```

Const Iterator Implementation

```
List::const_iterator::const_iterator( ListNode *p ) :
  _node{p} {}

List::const_iterator&
  List::const_iterator::operator++() {
    _node = _node->next();
    return *this;
}

const int& List::const_iterator::operator*() {
  return _node->value();
}
```

More in the code...

Let's look into the code...

Final Project

Groups of 2-4 people; 3 recommended

N-Body Gravity Simulation

The Problem

Have N point masses with initial positions r_i, velocities v_i, accelerations a_i, and masses m_i. Compute all-pairs forces

$$F_{ij} = -\frac{Gm_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|^3}(\mathbf{r}_i - \mathbf{r}_j)$$

and update the locations.

Most basic integrator

```
void System::integrateSystem( float dt ) {
  Vector3f r, v, a;
  for( size_t i = 0; i < _nBodies; ++i ) {
    r = _body[i].position();
    v = _body[i].velocity();
    a = _body[i].acc();

    v = v + ( a * dt );
    r = r + v * dt;

    _body[i].position() = r;
    _body[i].velocity() = v;
  }
}
```

Components

Requirements

25% **Physics Engine** - quality and extensibility of simulation code

25% **Visualization** - OpenGL; getting a good visualization working

15% **Unit testing** - gtest, quality and coverage of tests

15% **Software Process** - code reviews, overall integration of project

10% **Interactive** - user interactivity with simulation (keyboard, mouse, etc)

10% **Do something cool** - make it look cool, add a useful feature, do something interesting!

Extra 5% available in all areas for exceptional effort.

"Do Something Cool"

Just a few examples of potential areas:

- Advanced OpenGL
- Threading with `<thread>`
- Parallelize with OpenMP
- More interactive (other forms of input)

Physics Engine

You should be producing a library called `libnbody.a`. The only strict requirement is that you will wrap all of your code in a `namespace nbody` and provide the following interface:

```
namespace nbody {
  class Simulation {
    // ...
  public:
    void loadRun( std::istream &in );
    void evolveSystemFor( float time );
    void saveRun( std::ostream &out ) const;
    // ...
  };
} // namespace nbody
```

Physics Engine

```
void nbody::Simulation::loadRun( std::istream &in );
```

Constructor to read in a common “state” file.

Physics Engine

```
void nbody::Simulation::evolveSystemFor( float time );
```

Evolve the system forward in time by time (in seconds).

Physics Engine

```
void nbody::Simulation::saveRun( std::ostream &out )
  const;
```

Function to write out a common “state” file.

Physics Engine

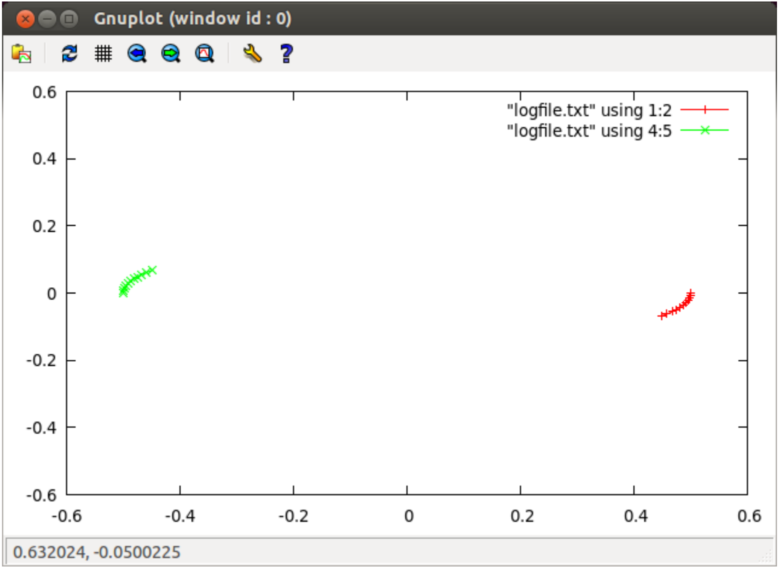
Why so little specification?

Physics Engine

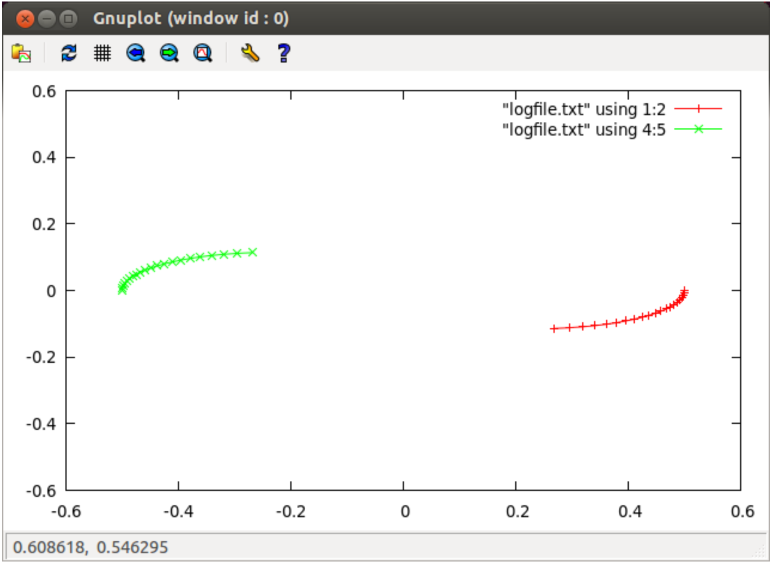
Why so little specification?

Your team should take the initiative! Look up various ways of do the integration to improve accuracy, different size time steps, unit systems, and so forth. I'll be adding hints throughout the week.

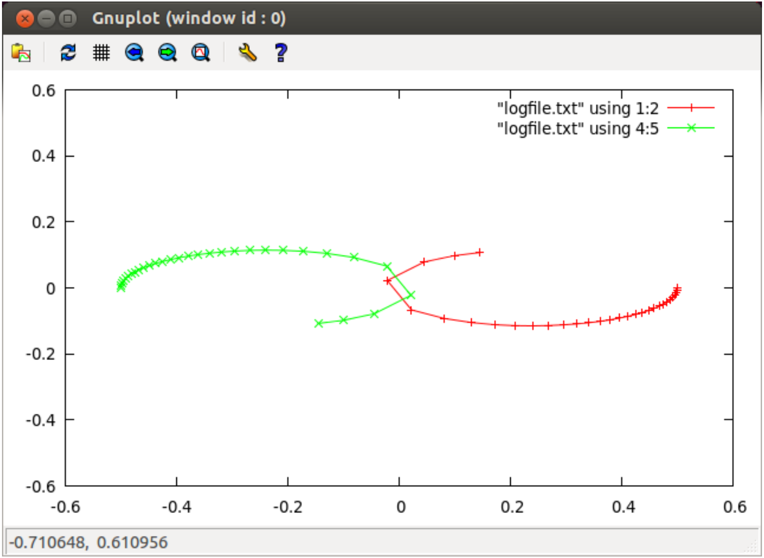
Binary Star System Example 1



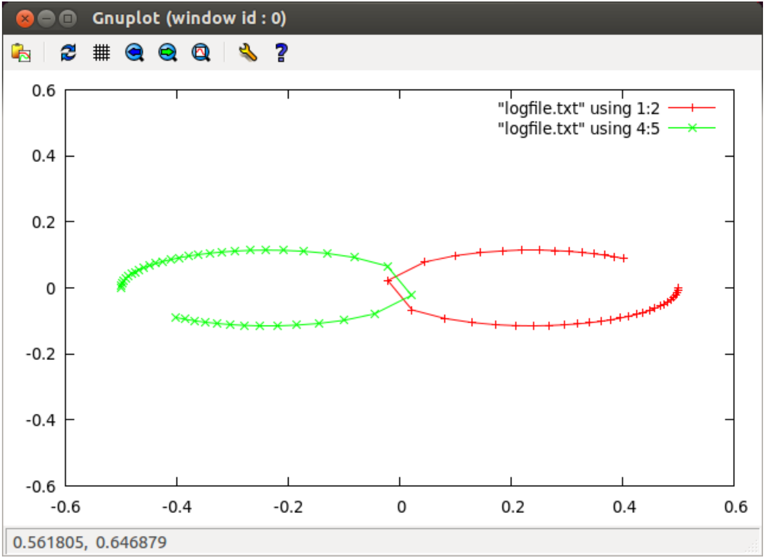
Binary Star System Example 2



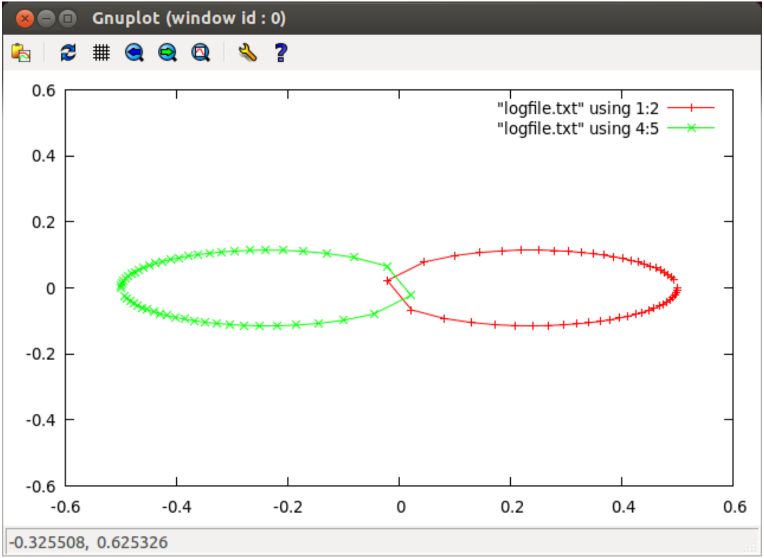
Binary Star System Example 3



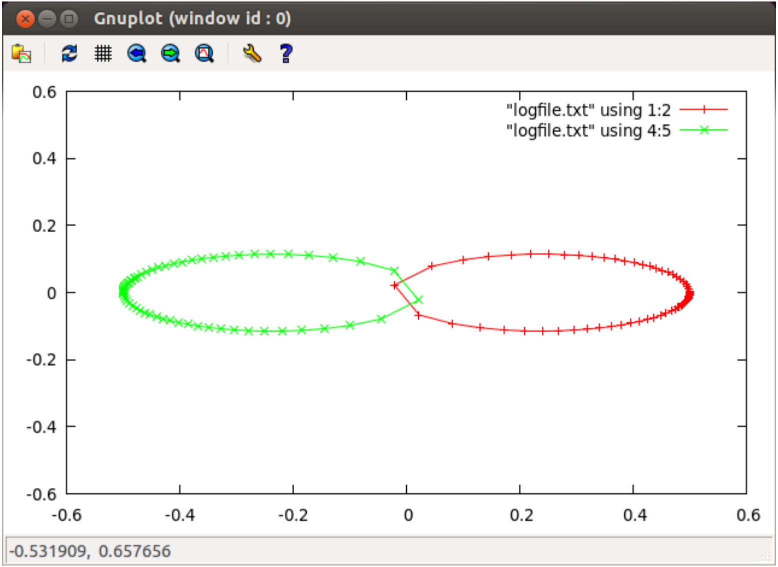
Binary Star System Example 4



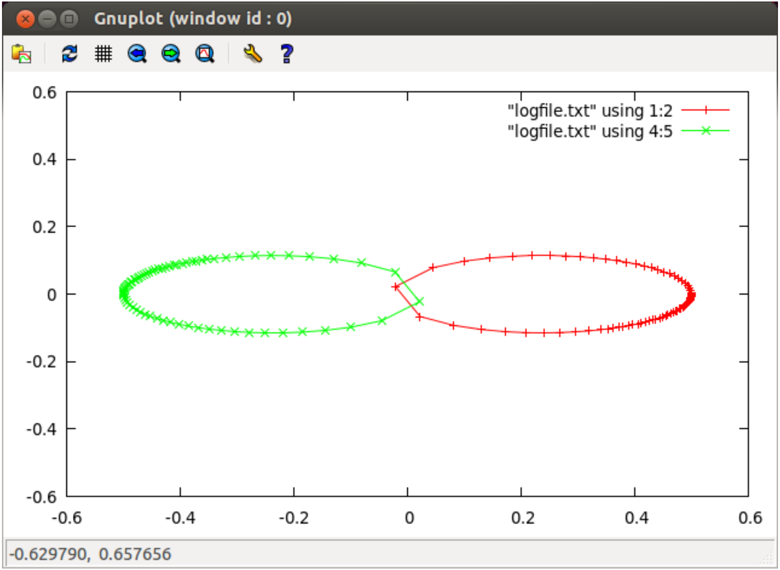
Binary Star System Example 5



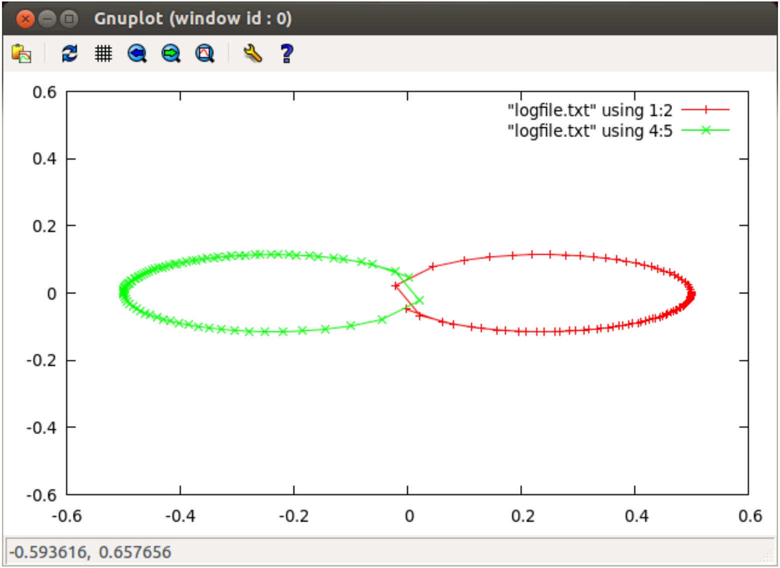
Binary Star System Example 6



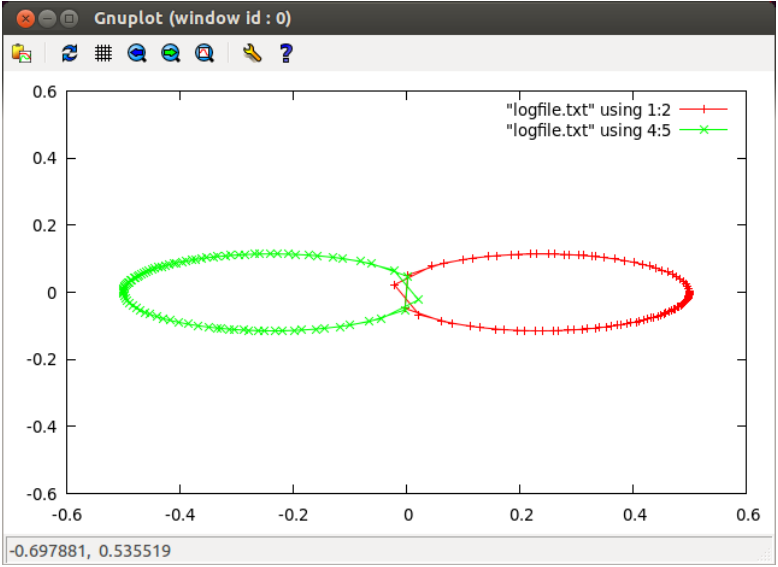
Binary Star System Example 7



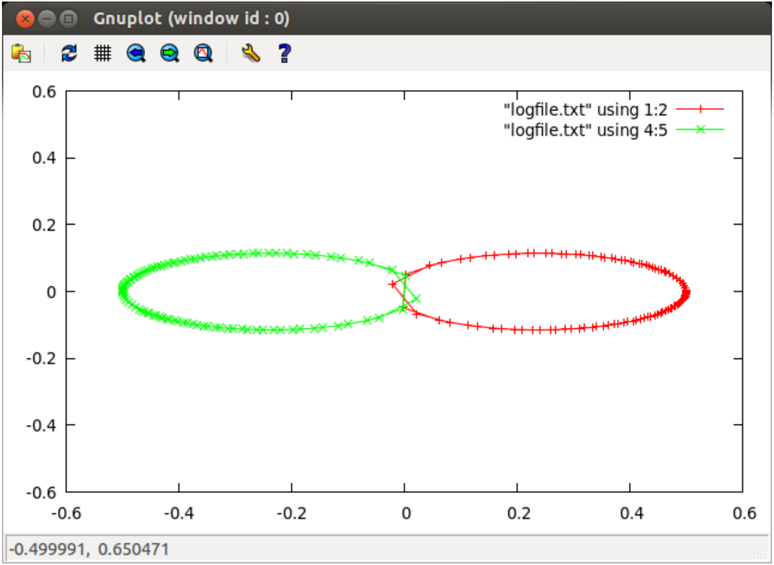
Binary Star System Example 8



Binary Star System Example 9

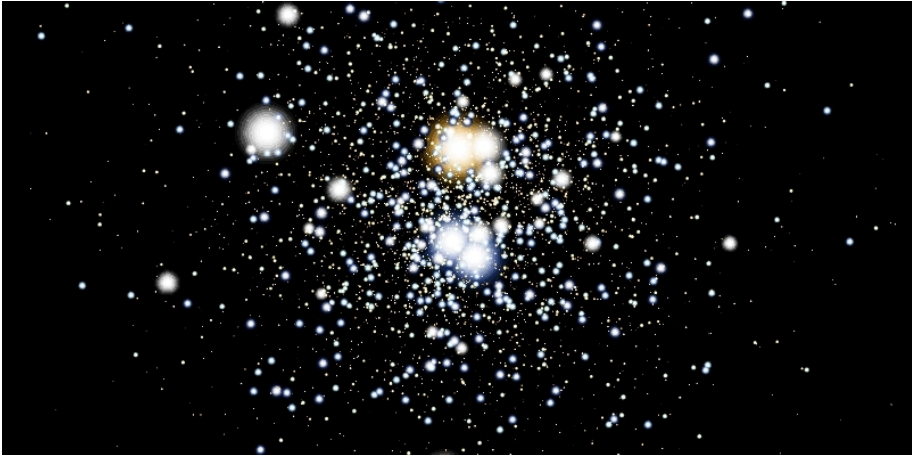


Binary Star System Example 10



Visualization

OpenGL; sample code provided tomorrow and Wednesday.



Courtesy of Aaron M. Geller. Used with permission.

Interactive

Your final product should be easy to use.

- Mouse integration (moving the view)

- Keyboard controls

- Command line arguments

Content Provided

`Vector3.h`

So that you don't have to write (all) of your own vector math, feel free to use the header available.

It's a templated 3-d vector class that can be widely useful and is guaranteed fast ("plain old data type")

Content Provided - Vector3.h

```
template<typename T>
class Vector3 {
  T _x, _y, _z;
public:
  Vector3() : _x{}, _y{}, _z{} {}
  Vector3( T x_, T y_, T z_ ) :
    _x{x_}, _y{y_}, _z{z_} {}
  inline T x() const { return _x; }
  inline T y() const { return _y; }
  inline T z() const { return _z; }
  T norm() const;
  T normsq() const;
};
```

Content Provided - Vector3.h

All the overloads and helpful functions you could want:

```
template<typename T> inline
const Vector3<T> operator+( const Vector3<T> &a,
                            const Vector3<T> &b ) {
  return Vector3<T>{ a.x() + b.x(),
                     a.y() + b.y(),
                     a.z() + b.z() };
}
//..etc
template<typename T> inline
T dot( const Vector3<T> &a, const Vector3<T> &b ) {
  return a.x() * b.x() + a.y() * b.y() + a.z() * b.z();
}
```

Code Reviews

- You should be doing reviews of all committed code within your group.

- Each group member should send me one such review by Wednesday.

- There will also be an inter-group review that I will organize.

What you send to me

- Your name and the name of the person whose code you are reviewing.
- The snippet of code you are reviewing: more than 30 lines, less than 100.
- Your comments interspersed in their code.
- A summary of main points relating to the review (what they did well, major areas for improvement, common issues, general observations).

What you send to me

- Your name and the name of the person whose code you are reviewing.
- The snippet of code you are reviewing: more than 30 lines, less than 100.
- Your comments interspersed in their code.
- A summary of main points relating to the review (what they did well, major areas for improvement, common issues, general observations).

You should choose a bite-sized chunk that will take you 45 mins to 1 hour to fully review.

Tips for effective code review

- Most important features for the code: correctness, maintainability, reliability, and performance. Consistent style is good, but those other points come first!
- Keep your review short and to the point.
- Check the code for compliance with the class coding standards.
- Take the time for a proper review, but don't spend much more than an hour; additionally, don't review much more than about 200 lines of code at once.

Structure of Final Project Source

Live demonstration of project setup

Separation of build and source

Have a clean build.

- Since this is definitely under revision control, we want to keep our directories free from clutter
- Hence, all object (.o) files will go in the bin/ directory.
- Third-party libraries live in their own directory `third_party/gtest` or whatever.
- Headers for our project named "project" are deployed to the install directory.

Be able to build in one step

- We have an upper-level Makefile so that we can still just `make` our project.
- However, that's been split up into more modular sub-makefiles (`make/*.mk`).

Unit Testing and Test-Driven Development

Testing your source code, one function or “unit” of code at a time.

- Test-driven development: write the tests first and then write code which makes the tests pass
- Decide how you want the interface to work in general, write some tests, and go develop the specifics.

gtest: the Google C++ Testing Framework

Highly cross-platform, available from here.

- Runs your full suite of tests (potentially each time you compile)
- Tests are very portable and reusable across multiple architectures
- Powerful, but very few dependencies.

Example from their primer:

```
ASSERT_EQ(x.size(), y.size()) << "unequal length";

for (int i = 0; i < x.size(); ++i) {
  EXPECT_EQ(x[i], y[i]) << "differ at index " << i;
}
```

Examples

Let's see some examples...

Wrap-up & Wednesday

Final project due Saturday.

You need to begin work on it *now*!

Class on Wed.

- OpenGL, templates, more on large projects

Questions?

- Office hours Mon, Tues

MIT OpenCourseWare
http://ocw.mit.edu

6.S096 Effective Programming in C and C++
IAP 2014

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.