Debugging strategies

University of California Riverside

Challenges

- Is it correct?
- How do I find the problem?

First steps

- Start simple
    - One object
    - Square domain
    - Zero velocity
    - No forces

First steps

- Start simple
  - One object
  - Square domain
  - Zero velocity
  - No forces
- Catch simple stuff
  - Crashes
  - Out of bounds
  - NaN
  - Assertions

Fail hard

- Easy to track down:
  - Compile errors
  - Segfault
  - Memory leaks
  - Assertions
  - Out-of-bounds

Fail hard

- Easy to track down:
  - Compile errors
  - Segfault
  - Memory leaks
  - Assertions
  - Out-of-bounds
- Take advantage of it

Compile errors

- Compiler is your friend
- Don't ignore warnings
- `-Wall -Werror`

  `warning: unused variable 'z' [-Wunused-variable]`

  `warning: 'y' may be used uninitialized in this function`
- Messy code is buggy code

Don't let mistakes compile

- $\vec{u} \times \vec{v}$ with $4D$ vectors?
- $\mathbf{A}\vec{u}$ with mismatched sizes?
- $\mathbf{A}^{-1}$ for non-square?

Type safety

- `int body_index;`
  - Which array?
  - Bad bug: indexing wrong array

Type safety

- `int body_index;`
  - Which array?
  - Bad bug: indexing wrong array

Type safety

- `int body_index;`
  - Which array?
  - Bad bug: indexing wrong array
- `rigid_body* body;`
  - Type safe
  - `nullptr`
  - Harder to misuse

Debugger

- Debugger quickly tells you where
  - Segmentation faults
  - Runtime exceptions

Debugger

- Debugger quickly tells you where
  - Segmentation faults
  - Runtime exceptions
- Hardware watchpoint
  - Who changed that?

Debugger

- Debugger quickly tells you where
  - Segmentation faults
  - Runtime exceptions
- Hardware watchpoint
  - Who changed that?
- Look around
  - array.size() == 0… Oops!

Valgrind

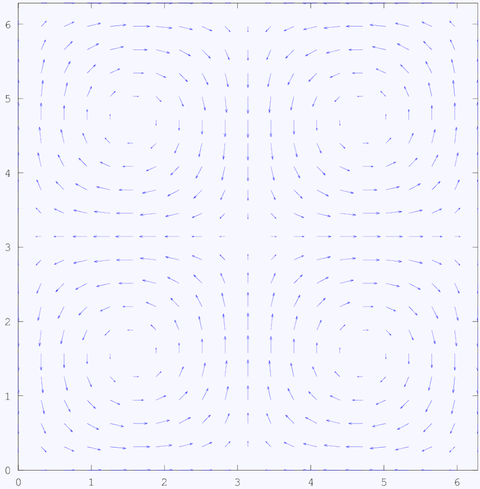
- Memory errors
- Out-of-bounds
- Memory leaks
- Double free
- Uninitialized data
- Dangling pointers

Valgrind

- Memory errors
- Out-of-bounds
- Memory leaks
- Double free
- Uninitialized data
- Dangling pointers
- Linux only (also Mac?)

Analytic solutions

- Translation
- Rotation
- Couette flow
- Taylor-Green vortex



Using analytic solutions

- Convergence study
  - $\Delta t \to 0$, $\Delta x \to 0$
- Isolating parts
  - Advection-only
  - Zero viscosity

Discretizations are sometimes exact

- Linear interpolation exact on $ax + b$
  - Is yours?

Discretizations are sometimes exact

- Linear interpolation exact on $ax + b$
  - Is yours?
- Constant $\vec{u}, p$ (translation)

Discretizations are sometimes exact

- Linear interpolation exact on $ax + b$
  - Is yours?
- Constant $\vec{u}, p$ (translation)
- Very easy to track down
  - No discretization error
  - Know what intermediates should be

Method of manufactured solutions

- Original PDE: $\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \nabla p = 0$

Method of manufactured solutions

- Original PDE: $\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \nabla p = 0$
- Add forcing: $\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \nabla p = f$
  - Must discretize the f
  - More "general" but *easier* to debug

Method of manufactured solutions

- Choose *arbitrary functions* $\hat{\vec{u}}, \hat{p}$

Method of manufactured solutions

- Choose *arbitrary functions* $\hat{\vec{u}}, \hat{p}$
- Calculate forcing term:
  $\hat{f} \leftarrow \frac{\partial \hat{\vec{u}}}{\partial t} + (\hat{\vec{u}} \cdot \nabla)\hat{\vec{u}} + \nabla \hat{p}$

Method of manufactured solutions

- Choose *arbitrary functions* $\hat{\vec{u}}, \hat{p}$
- Calculate forcing term:
  $\hat{f} \leftarrow \frac{\partial \hat{\vec{u}}}{\partial t} + (\hat{\vec{u}} \cdot \nabla)\hat{\vec{u}} + \nabla \hat{p}$
- Solve numerically: $\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \nabla p = \hat{f}$

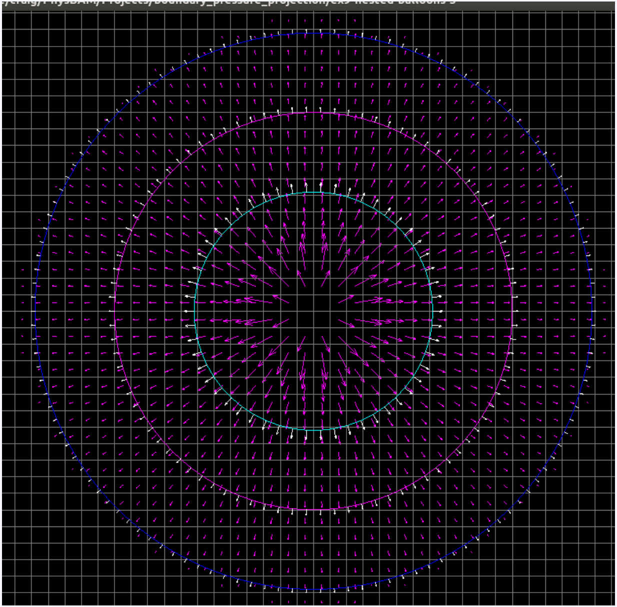
Method of manufactured solutions

- Choose *arbitrary functions* $\hat{\vec{u}}, \hat{p}$
- Calculate forcing term:
  $\hat{f} \leftarrow \frac{\partial \hat{\vec{u}}}{\partial t} + (\hat{\vec{u}} \cdot \nabla)\hat{\vec{u}} + \nabla \hat{p}$
- Solve numerically: $\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \nabla p = \hat{f}$
- Compare numercial $\vec{u}, p$ with analytic $\hat{\vec{u}}, \hat{p}$

Avoiding boundary conditions

- Periodic boundary conditions
- Analytic solution that is zero at boundary

Visual debugging



Dimensional analysis

- Physical quantities have units
  - E.g., $kg\,m\,s^{-2}$

Dimensional analysis

- Physical quantities have units
  - E.g., $kg\,m\,s^{-2}$
- Which of these is right? (c has units m/s)

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_{i+1}^n - u_i^n}{\Delta t} = 0$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta x} + c\frac{u_{i+1}^n - u_i^n}{\Delta t} = 0$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta x} + c\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0$$

Software engineering practices

- Version control

Software engineering practices

- Version control
- Testing suite
    - I thought that was working last week?

Software engineering practices

- Version control
- Testing suite
  - I thought that was working last week?
- Design before you code

Software engineering practices

- Version control
- Testing suite
    - I thought that was working last week?
- Design before you code
- Plan ahead for debugging

Software engineering practices

- Version control
- Testing suite
  - I thought that was working last week?
- Design before you code
- Plan ahead for debugging
- If you cannot debug it, don't write it

Avoid misusing indices

```
template<int d>
struct index_type
{
  int value;

  explicit index_type(int i) {value=i;}
};

int value(int i){return i;}
template<int d> int value(index_type<d> i){return i;}

template<class T, class I>
struct array
{
private:
  std::vector<T> data;

public:
  T& operator[](I i){return data[value(i)];}
  const T& operator[](I i) const {return data[value(i)];}
  void resize(I n);
  I size(){return I(data.size())}
};
```

Avoid misusing indices - usage

```
typedef index_type<0> triangle_id;
typedef index_type<1> vertex_id;
typedef index_type<2> rigid_body_id;

array<rigid_body*,rigid_body_id> rigid_bodies;
array<vec3,vertex_id> vertices;
array<ivec3,triangle_id> triangles;

// Are these per-triangle or per-vertex colors?
array<vec3,vertex_id> colors;

// Need operator++, operator<, ...
for(rigid_body_id i(0); i<rigid_bodies.size(); i++)
  rigid_bodies[i]->update();
```