# Frequently Asked Questions

**Animation & CGI Motion**

## General:

### Common Errors:

- Empty (black) frames from generating pngs:
  - do not pass in the `-d 0` flag option when running simulation.
- Tiny residuals: Use `scalar` (A wrapper for double, defined in `MathDefs.h`) rather than `floats`.
- `NaN` errors:
  - 0 or infinity somewhere, check for divergence.
  - If in creative scene, then make sure no two particles are overlapping
- Visual desktop not working:
  - Try opening the virtual desktop in a separate window. You can also access it using https://pagoda-cigar-3000.codio.io/ where you replace pagoda-cigar with your boxes domain name, which you can find in the menu at Project->Box Info (scroll to the section Web: Static content). Codio Documentation
- How does the oracle measure error:
  - For each time step, the oracle *assumes* that your start of time step position and velocity is correct. It then compares your end of time step position and velocity to its own computation of position and velocity, based on the aforementioned assumption. So, if your code updates position correctly, but updates velocity incorrectly, you would get zero residual for position. Note that zero residual for position does not imply that the entire trajectory over all time is tracking the correct position. It just means that every single one of your position updates was correct (even if it may have been computed based on an incorrect velocity).
- If you are getting different simulation results after each run it is possible that you didn't initialize some vectors or matrices properly. Check that you zeroed them out before use.

### Debugging Errors:

- Oracle in release vs debug mode:
  - Debug mode does some safe-guarding when it comes to memory (such as extra allocation and default initializations) which can break your code when gone in Release mode. Additionally, we perform some extra checks at each time step, like correct vector sizes and indices within bounds.
  - Changing between release and debug mode:
    - Inside the build directory:
      Run `ccmake .`
      Press "Enter" to edit the option `CMAKE_BUILD_TYPE`
      Type "`Release`" and hit enter.
      Press "`c`" to configure (apply the change)
      Press "`q`" to generate and quit (save the change and quit)
      `make` your project

`build` directory again.

- Interpreting residuals.txt:
  - The first column signals if the test passed, 1 being "Passed" and 0 being "Failed." The second is the residual for that tests. If all your tests passed, the first column is all 1s and the second is all 0s.
- Segmentation fault:
  - There are a number of reasons why you could end up having a segmentation fault:
    - null pointer dereference / invalid memory access
    - use or mishandling of uninitialized vectors/matrices
    - use of already freed objects
  - In order to debug a segfault, you should set up breakpoints / printing statements, and run your scenes step by step (by pressing "s"). This way you might be able to pin down exactly which line of your code causes the segfault.
  - What do those tests have in common (in their setup xml), and as a group how do they differ from the tests that are not seg faulting?

**Helpful Tips:**

- Use Git, and version control your code (locally, or in github in a private repo)
- If you are falling all the test scenes, try making your own test scenes with very simple scenes that has feature nodes of what you are failing, or take the scene you're failing and sequentially remove elements until you pass again
- Run your test scene against the oracle with the display on. The oracle will show you where your code's output differs from the oracle's.
- There ARE hidden tests. This is why even if you pass all given tests, you may not get 100% when you submit. Please generalize!
- Read through the .h files, as they contain methods that are already implemented that you will need (especially `MathDefs.h`)
- If you don't like using Codio's interface you can SSH into your Codio box as described here.
- GDB is a good debugger that codio already comes with.
- When running assignments locally on a Ubuntu virtual machine, make sure you install the same version of everything as in Codio (i.e: `g++ 4.8`)
- For equation derivations in general:
  - Use symbolic differentiation (`python`, `R`, `MatLab` or Mathematica).
    - Some helpful python libraries you might want to use are `sympy`(helpful article on sympy), `numpy`, and `linalg from scipy`, and one super helpful function you will want to use is `printing.cxxcode()`
    - In Matlab, checkout `collect()` and `ccode()`
    - When symbolic derivations is still giving errors, trying simplifying the output
  - When deriving by hand, keep the scalars and vectors separate and keep combining them (e.g. if there is a long expression and you know it is a scalar overall, then just call it "a" and so on) And if there is a vector in that expression somewhere, do not combine it. Vectors will finally combine themselves into scalars by the dot products
    - Source
  - Lastly, there is a c++ auto differentiation library, which we've never used but may be useful to some of you, called FadBad++

# Theme 1

### Common Errors:

### Helpful Tips:

- Force is the negative gradient of Potential Energy, because of that output of `accumulateGradU()` needs to be negated.

**Conventions:**

- Functions:
  - `void accumulateGradU( VectorXs& F, const VectorXs& dx = VectorXs(), const VectorXs& dv = VectorXs() );`
    - Input: nothing
    - Output: `F`, `dx` (optional) , `dv` (optional) (passed by reference as arguments)
    - Explanation: `F` is a zeroed `VectorXs` (see `VectorXs::Zero()` in Eigen documentation) of size `2N`, where `N` is the number of particles in the scene
    - `accumulateGradU(...)` calls `addGradEToTotal(... )` and `F` holds the accumulated forces
  - `void accumulateddUdxdx( MatrixXs& A, const VectorXs& dx = VectorXs(), const VectorXs& dv = VectorXs() );` and `void accumulateddUdxdv( MatrixXs& A, const VectorXs& dx = VectorXs(), const VectorXs& dv = VectorXs() );`
    - Input: nothing
    - Output: `A`, `dx` (optional) , `dv` (optional) (passed by reference as arguments)
    - Explanation: `A` is a zeroed `MatrixXs` (see `MatrixXs::Zero()` in Eigen documentation) of size `2Nx2N`, where `N` is the number of particles in the scene
  - `const std::vector<scalar>& getRadii() const;`
    - Input: nothing
    - Output: A `vector` of `scalar` of size `N` that contains all the radii of the particles in the scene
  - `const std::vector<scalar>& getEdgeRadii() const;`
    - Input: nothing
    - Output: A `vector` of `scalar` of size `M` (the number of edges in the scene) that contains all the radii of the edges in the scene
- Variables:
  - `m_gravity` in `SimpleGravityForce.h`
    - Value of `g`
  - `h` is the time step size (with the unit of second) and `F` is the force vector (with the unit of Newton) that is dependent on position and velocity.

**Milestone 1:**

-

**Milestone 2:**

-

**Milestone 3:**

- Common Errors:
  - Implicit Euler doesn't converge:
    - Don't forget to zero out the Force vector and the dF/dq and dF/dqdot matrices before each iteration

# Theme 2

### Common Errors:

- Wrong polynomial degree
    - Double check variable names. It might be helpful to use unique variable names for each set of coefficients.
    - Make sure you push them in the correct order
- Wrong Number of Polynomials: `detectCollisions()` have been called the wrong number of times. Reread the write up and comments given in the method and make sure to follow the exact same routine outlined.
- Infinite Loop: Print out the impact zones `Z` and `Zprime`, compare the vertices in each zone manually (Check the definition of `ImpactZone.h`)
- For general derivation tips, refer to the point about derivation in General/Helpful Tips.

### Helpful Tips:

- Supplementary material, a summary of collision detection algorithms: http://www.cs.cmu.edu/~jbruce/thesis/chapters/thesis-ch03.pdf

### Conventions:

- In `gradn = ( -I I )`, `I` is a 2x2 Identity Matrix

### Milestone 1:

-

### Milestone 2:

- Derivation
    - It could be helpful to let `B(t)=(x3(t)-x2(t))` and `C(t)=(x2(t)-x1(t))`. Then just find the product of the polynomials in this manner because it helps cancel things out. It makes the derivation still long, but reduces a number of calculations.
    - Do variable substitution as early as possible, and get rid of the squared-norm term first. Do not expand α until you really think it is the time.
- HybridCollisionDetector

### Milestone 3:

# Theme 3

### Common Errors:

### Helpful Tips:

### Conventions:

### Milestone 1:

### Milestone 2:

- LCP internal errors happens when the iterative solver falls into an infinite loop. Most likely, the derived A and b values are incorrect.

# Theme 4

### Common Errors:

- For Derivation tips, refer to the point about derivation in General/Helpful Tips.
- In `BendForce`, when two endpoints are in exact same position, like in `test02`, `test10`, and `test12`, `theta` physically has two valid values, `PI` and `-PI`, but `atan2` does not always return what the oracle expects. If `(abs(e01_x_e12) <= 0) && (e01_dot_e12 < 0)`, then set `theta` to `PI`, which is the one that the oracle accepts.
- `e01_x_e12` and `e01_dot_e12` may be zero, so don't leave these in denominator alone. Derive by hand!

### Helpful Tips:

### Conventions:

# Theme 5

### Common Errors:

### Helpful Tips:

### Conventions:

- `d`, `v` and `u` are protected member variables in `StableFluidsSim`, and can be accessed through protected member methods. However, in the functions in T5M1, you are implementing `diffuse()`, `advect()` and `project()`, and are updating `d`, `u` and `v` in these methods using its old values. Thus, you need to use `x` and `x0` as the array containing the new values and the old values, respectively, of `d`, `v` and `u`.