Assignment 6: Trajectory Optimization

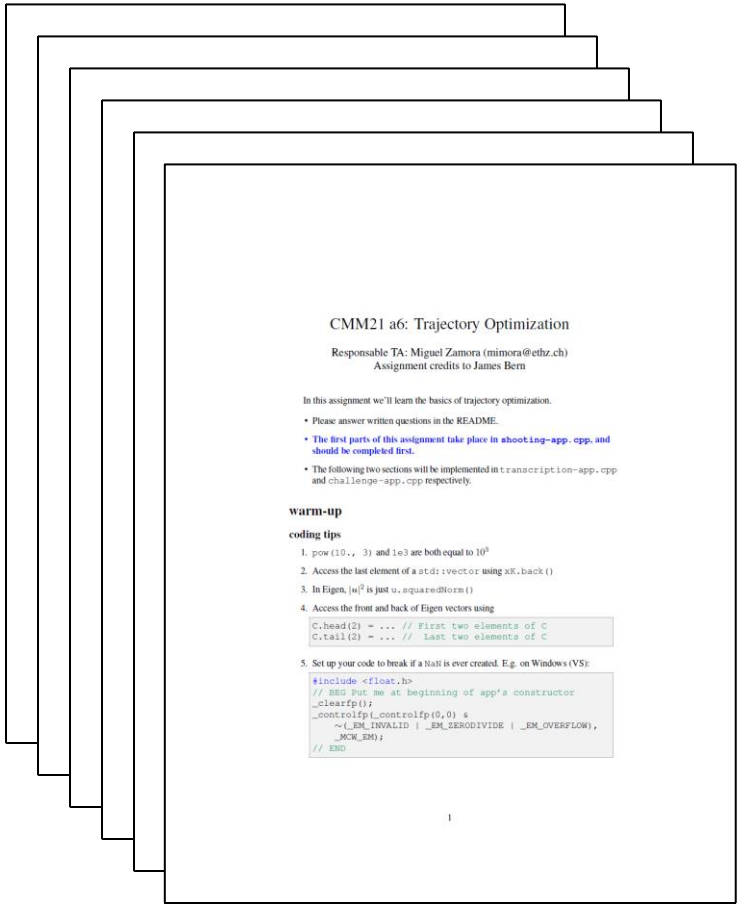
TA: Miguel Zamora (mimora@ethz.ch)

Assignment 6

- Assignment credits to James Bern!
- Use Github Issues to raise questions
- Use **Release** or **RelWithDebInfo** mode otherwise too slow!

~~a4~~ a6

By James Bern



CMM21 a6: Trajectory Optimization

Responsable TA: Miguel Zamora (mimora@ethz.ch)
Assignment credits to James Bern

In this assignment we'll learn the basics of trajectory optimization.

- Please answer written questions in the README.
- **The first parts of this assignment take place in `shooting-app.cpp`, and should be completed first.**
- The following two sections will be implemented in `transcription-app.cpp` and `challenge-app.cpp` respectively.

warm-up

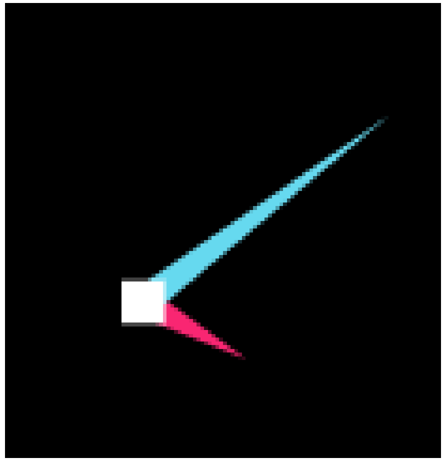
coding tips

1. `pow(10., 3)` and `1e3` are both equal to 10^3
2. Access the last element of a `std::vector` using `xK.back()`
3. In Eigen, $|u|^2$ is just `u.squaredNorm()`
4. Access the front and back of Eigen vectors using

```
C.head(2) = ... // First two elements of C
C.tail(2) = ... //  Last two elements of C
```

5. Set up your code to break if a `NaN` is ever created. E.g. on Windows (VS):

```
#include <float.h>
// BEG Put me at beginning of app's constructor
_clearfp();
_controlfp(_controlfp(0,0) &
    ~(_EM_INVALID | _EM_ZERODIVIDE | _EM_OVERFLOW),
    _MCW_EM);
// END
```



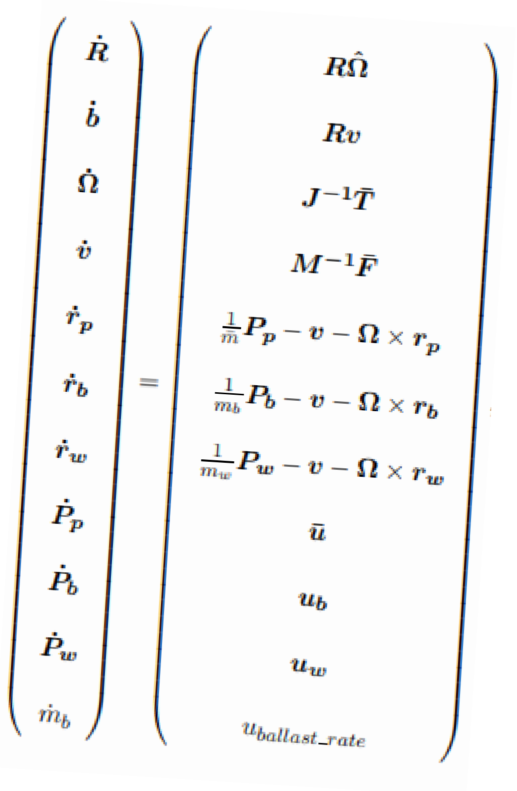
disclaimer

Terminology for trajectory optimization is very confusing and not particularly consistent across sources.

Good resource: http://underactuated.mit.edu/trajopt.html

state variables $\boldsymbol{\xi}$

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{bmatrix}$$



$$\begin{pmatrix} \dot{\boldsymbol{R}} \\ \dot{\boldsymbol{b}} \\ \dot{\boldsymbol{\Omega}} \\ \dot{\boldsymbol{v}} \\ \dot{\boldsymbol{r}}_{\boldsymbol{p}} \\ \dot{\boldsymbol{r}}_{\boldsymbol{b}} \\ \dot{\boldsymbol{r}}_{\boldsymbol{w}} \\ \dot{\boldsymbol{P}}_{\boldsymbol{p}} \\ \dot{\boldsymbol{P}}_{\boldsymbol{b}} \\ \dot{\boldsymbol{P}}_{\boldsymbol{w}} \\ \dot{m}_b \end{pmatrix} = \begin{pmatrix} \boldsymbol{R}\hat{\boldsymbol{\Omega}} \\ \boldsymbol{R}\boldsymbol{v} \\ \boldsymbol{J}^{-1}\bar{\boldsymbol{T}} \\ \boldsymbol{M}^{-1}\bar{\boldsymbol{F}} \\ \frac{1}{\bar{m}}\boldsymbol{P}_{\boldsymbol{p}} - \boldsymbol{v} - \boldsymbol{\Omega} \times \boldsymbol{r}_{\boldsymbol{p}} \\ \frac{1}{m_b}\boldsymbol{P}_{\boldsymbol{b}} - \boldsymbol{v} - \boldsymbol{\Omega} \times \boldsymbol{r}_{\boldsymbol{b}} \\ \frac{1}{m_w}\boldsymbol{P}_{\boldsymbol{w}} - \boldsymbol{v} - \boldsymbol{\Omega} \times \boldsymbol{r}_{\boldsymbol{w}} \\ \bar{\boldsymbol{u}} \\ \boldsymbol{u}_{\boldsymbol{b}} \\ \boldsymbol{u}_{\boldsymbol{w}} \\ u_{ballast\_rate} \end{pmatrix}$$

Note: More complicated systems possible...

first order dynamics $\dot{\boldsymbol{\xi}}$

$$\boldsymbol{F} = m\boldsymbol{a}$$

$$\frac{1}{m}\boldsymbol{F} = \boldsymbol{a}$$

$$\dot{\boldsymbol{\xi}} = \begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{v}} \end{bmatrix}$$

integrating forward in time

input

- control trajectory $\boldsymbol{u} = (\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{K-1})$
- initial conditions $\boldsymbol{\xi}_0$

algorithm-ish

- apply physics ($\boldsymbol{F}_k = m\boldsymbol{a}_k$) for $k = 0, \ldots, K-1$

output

- physically-valid state trajectory $\boldsymbol{\xi}(\boldsymbol{u}) = (\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_K)$

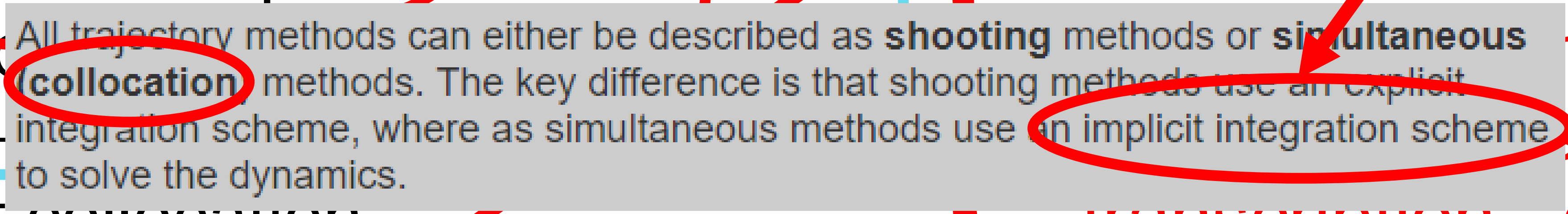
what is trajectory optimization?

Find the right control trajectory $\boldsymbol{u}$ to achieve a given task.

some approaches

Trajectory Optimization Terminology

- Direct s
  - Op
- Direct transcription
  - C
    -



All trajectory methods can either be described as **shooting** methods or **simultaneous** **(collocation)** methods. The key difference is that shooting methods use an explicit integration scheme, where as simultaneous methods use an implicit integration scheme to solve the dynamics.

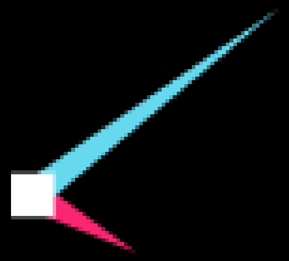
- Direct collocation
  - Represent $\boldsymbol{u}, \xi$ as splines and optimize over spline knots
    - Physics imposed as constraints

NOTE: Some sources
hese
s “direct
transcription,” even
though one of them is
already called direct
transcription.

other approaches

- Direct multiple shooting
- Orthogonal collocation
- Differential Dynamic Programming (DDP)
- ...

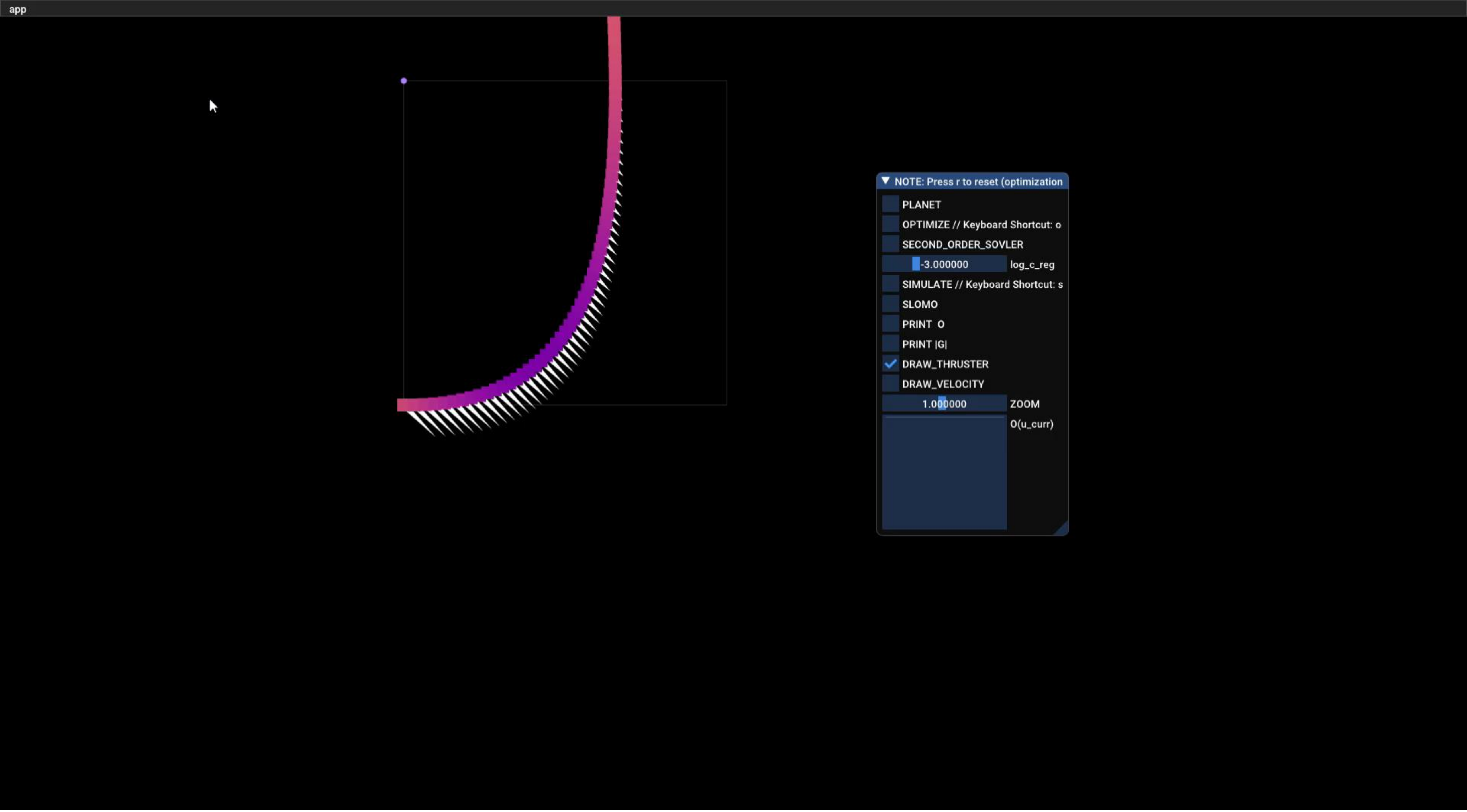
spaceship (no orientation)



shooting-app.cpp

Direct shooting

- Optimize over $\boldsymbol{u}$
  - $\boldsymbol{\xi}(\boldsymbol{u})$ found by solving physics



app

▼ NOTE: Press r to reset (optimization

□ PLANET

□ OPTIMIZE // Keyboard Shortcut: o

□ SECOND_ORDER_SOVLER

-3.000000 log_c_reg

□ SIMULATE // Keyboard Shortcut: s

□ SLOMO

□ PRINT O

□ PRINT |G|

☑ DRAW_THRUSTER

□ DRAW_VELOCITY

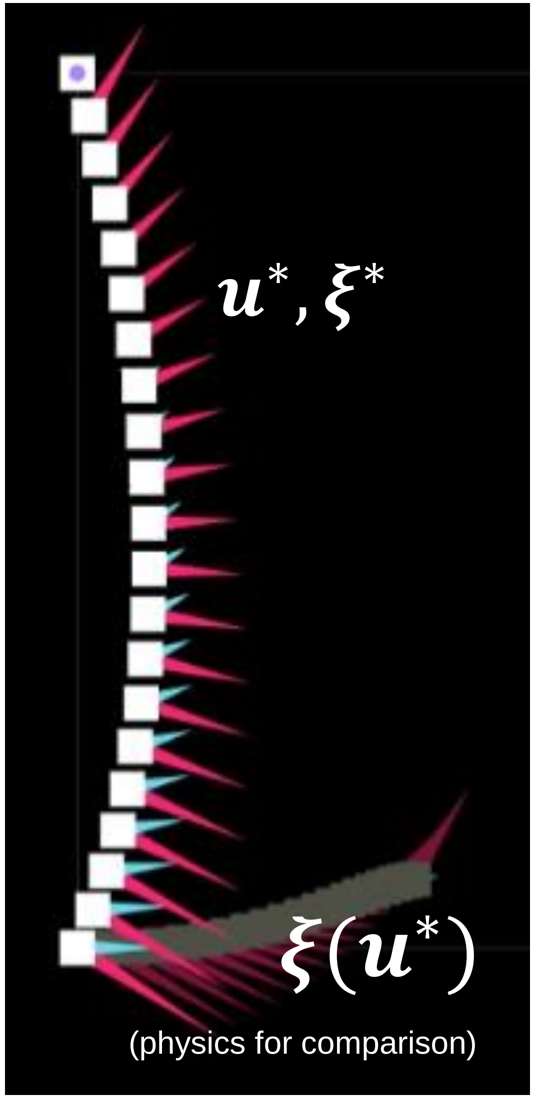
1.000000 ZOOM

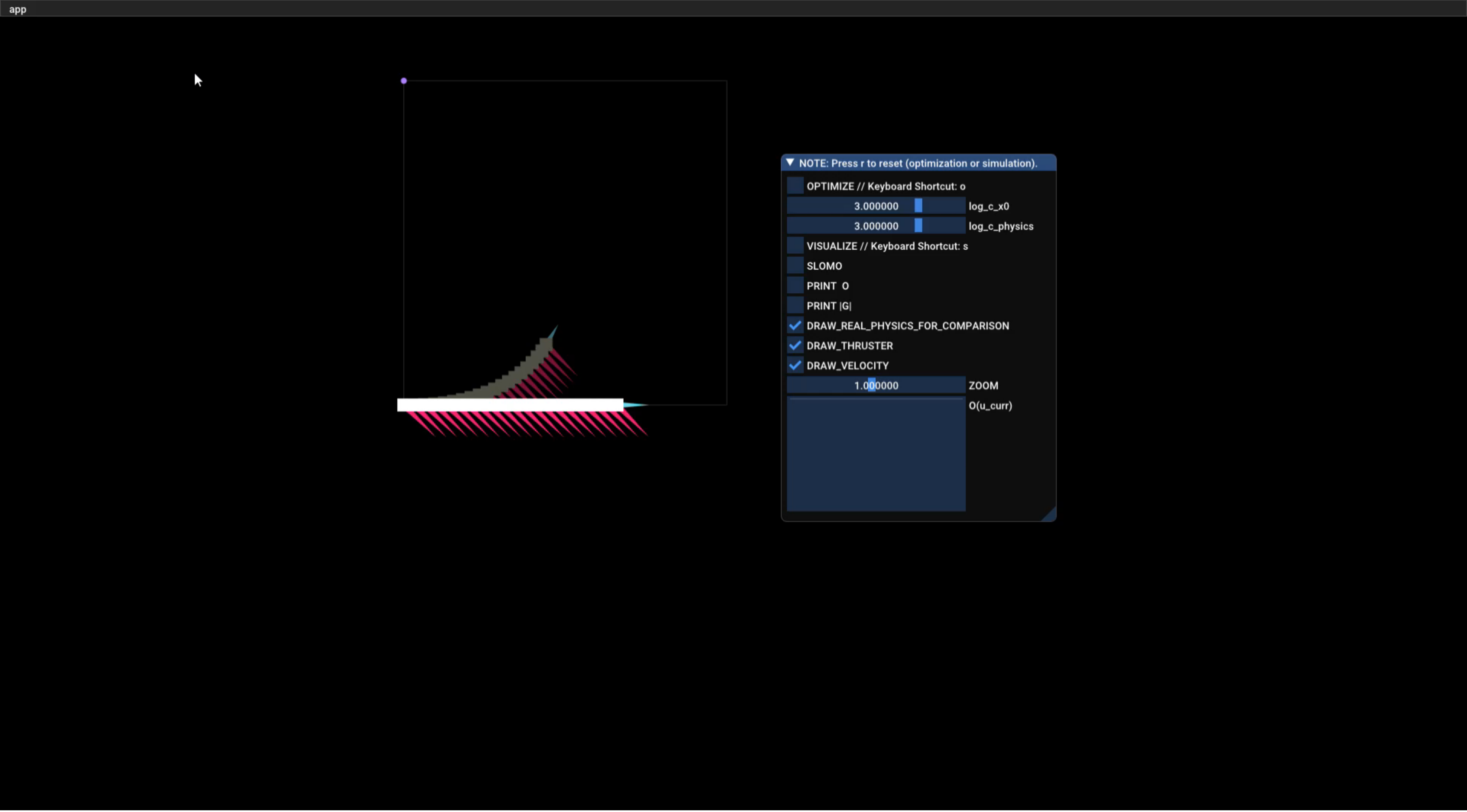
O(u_curr)

transcription-app.cpp

Direct transcription

- Optimize over $\boldsymbol{u}, \boldsymbol{\xi}$
  - Physics imposed as constraints


$\boldsymbol{u}^*, \boldsymbol{\xi}^*$
$\boldsymbol{\xi}(\boldsymbol{u}^*)$
(physics for comparison)



▼ NOTE: Press r to reset (optimization or simulation).

□ OPTIMIZE // Keyboard Shortcut: o

3.000000 log_c_x0

3.000000 log_c_physics

□ VISUALIZE // Keyboard Shortcut: s

□ SLOMO

□ PRINT O

□ PRINT |G|

☑ DRAW_REAL_PHYSICS_FOR_COMPARISON

☑ DRAW_THRUSTER

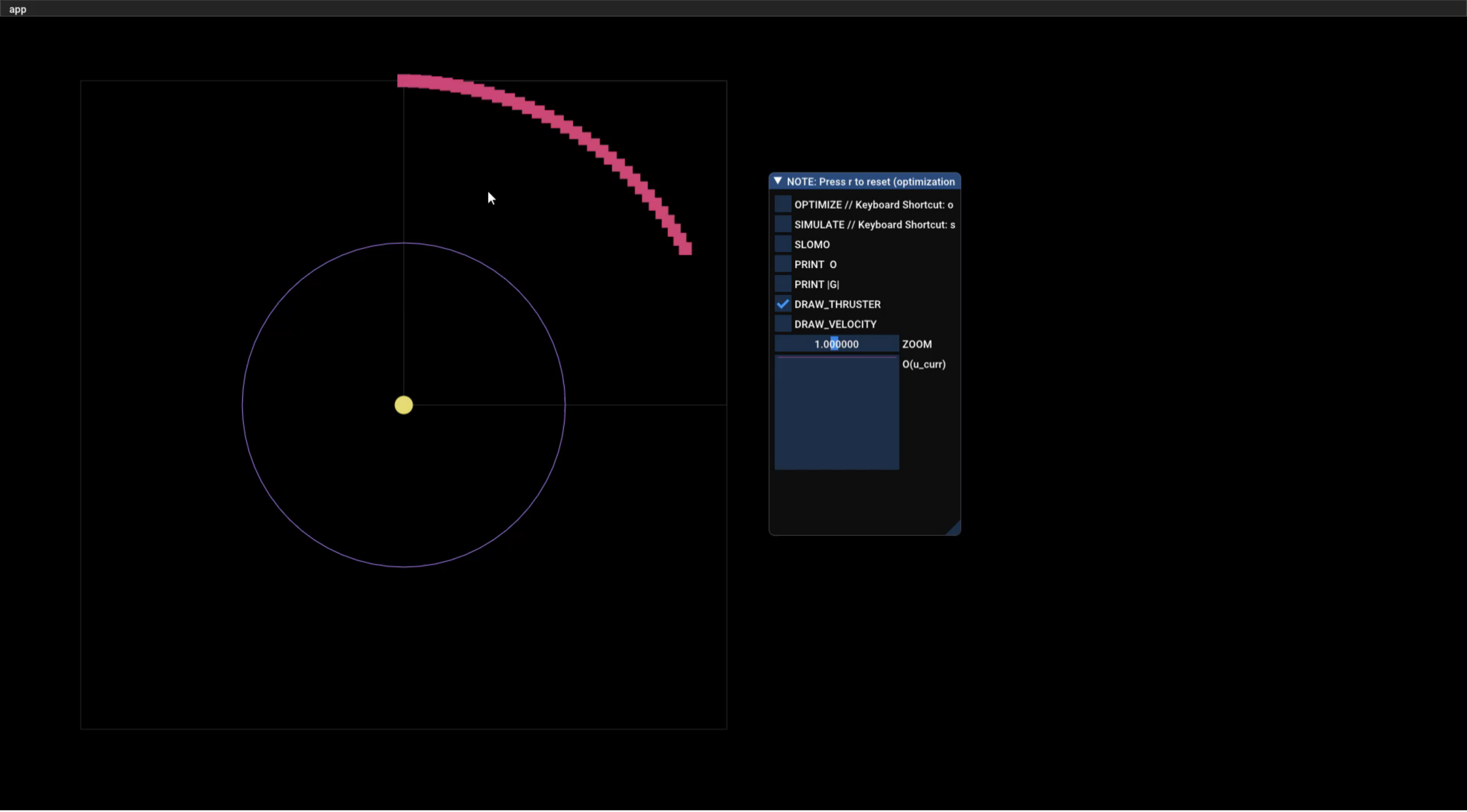
☑ DRAW_VELOCITY

1.000000 ZOOM

O(u_curr)

challenge-app.cpp

Use direct shooting to put spaceship into orbit.



NOTE: Press r to reset (optimization

□ OPTIMIZE // Keyboard Shortcut: o

□ SIMULATE // Keyboard Shortcut: s

□ SLOMO

□ PRINT O

□ PRINT |G|

☑ DRAW_THRUSTER

□ DRAW_VELOCITY

1.000000 ZOOM

O(u_curr)

Thanks!