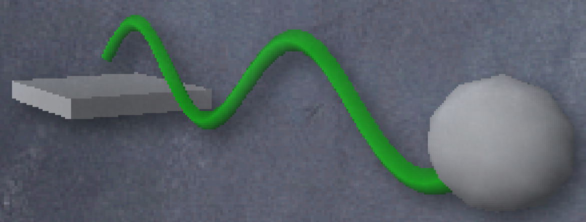
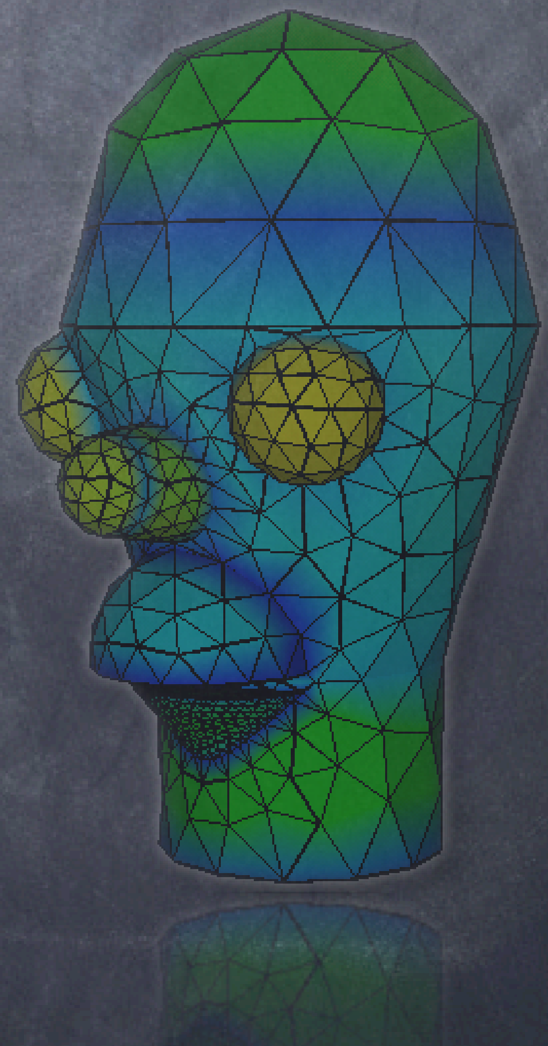
Large class learning and teaching of computing skills



Hans Fangohr
School of Engineering Sciences
University of Southampton
16 March 2006



Overview of talk

1. Motivation (objectives & constraints)
2. Implementation (course structure & tools)
3. Content (choice of language & examples)
    1. Language (Python)
    2. Examples (Visual Python)
4. Summary

1. Motivation: method

- Objectives:
  - learning of numerical methods, programming, problem solving
  - make topic more “attractive”
- Constraints
  - large student groups (100 and 200 students), 12 lectures

How do we learn to solve a problem?

- By trying to solve many problems:
  - succeed -> done
  - fail -> need to find better way, try again
- ... trial and error (gathering experience)
- lectures not the best medium for problem solving -> emphasis on practicals

Motivation: Content

Observation: Computing at home is very attractive - computing at uni is not.

- Computer games:
  - are interactive
  - require creativity
  - provide realistic graphics
- Computer education:
  - exploit interactivity
  - emphasise creativity (problem solving)
  - choose attractive content (for example using 3d graphics where possible)

2. Implementation: Course structure

- 12 lectures (one per week)
- 6 practical sessions for each student ("labs") each lasting 3 hours
- 5 assignments (associated with labs)
- marking of assignments and feedback in labs
- help session

Lectures

- Content chosen to provide relevant knowledge and examples for assignments
- Provide worked examples. Once understood, the concepts provided can be transferred to assignments
- Use to resolve problems coming up in practicals

Practicals (Labs)

- every student has 1 lab every 2 weeks
- work throught self-paced assignment in presence of demonstrators and lecturer (approx. one demonstrator for 10 students)
- self-paced work consists of sequence of problem solving exercises
- students can seek help (from demonstrators and friends)
- approximately 50% of class finish assignment in lab session

Practicals (Labs)

- students can carry on working on assignment outside the lab hours
- once the assignment is completed:
  - student submits work electronically
  - demonstrator conducts mini-viva with student
  - demonstrator determines mark

Help session

- Once a week a help session is offered
- Students can drop in to
  - catch up with assignments
  - clarify and improve knowledge and understanding
  - get support if they like to take the course material further

Learning process

- use lectures to provide examples and highlight relevant parts of the lecture notes
- most of learning takes place in labs (problem solving, hurdle, solution). Refer back to lecture notes
- regular one-to-one viva
  - ensures students' understanding
    allows to challenge more advanced students,
    provides feedback (on progress) to lecturer

Group learning

- We encourage discussions between students in labs
  - the 'teaching students' gains deeper insight
  - the 'learning student' can catch up
- Problem: need to mark understanding of each individual
  - ➡ one-to-one viva

Plagiarism

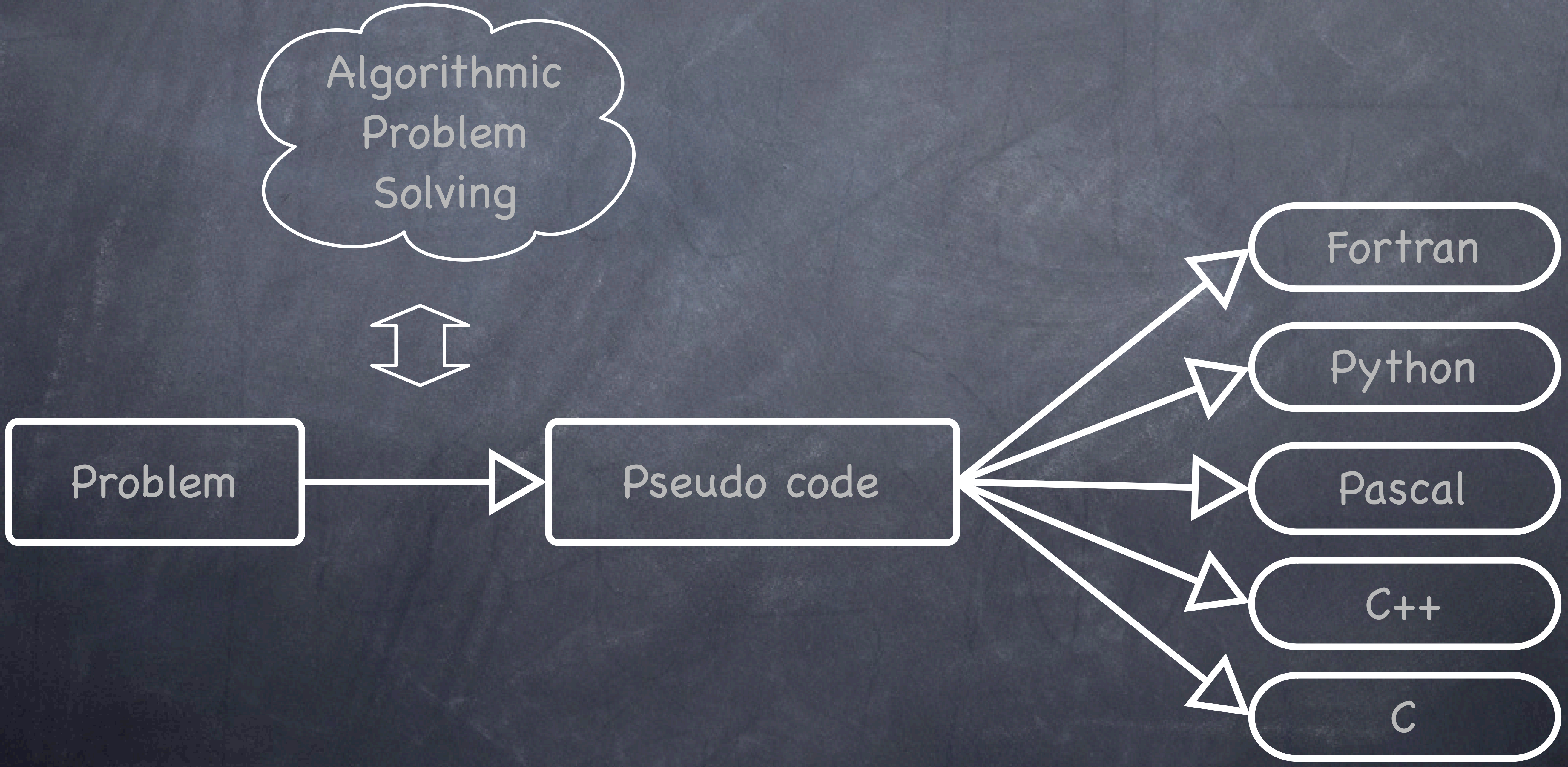
- Mark electronic work (i.e. computer programs)
- No printed proof of work -> possibility for complaints
  - email-submission system
- electronic copies allow plagiarism detection

3. Content & Examples

- What programming language (-> Python)
- Overview of content of module
- Problem solving examples
- Visualisation

Programming

= algorithmic problem solving & implementation



What programming language to use?

- Points to consider:
  - trends in industry and academia
  - power and flexibility, global use
  - availability (free? different OS?)
  - ease of use
- Problem solving process in language independent
  - -> choose language that is beginner friendly

What programming language to use? Comparison & Results

- Have compared
  - C
  - Matlab
  - Python
- in undergraduate and postgraduate learning.
- Result*: Python preferred

*H. Fangohr. A Comparison of C, Matlab and Python as Teaching Languages in Engineering. Lecture Notes on Computational Science **3039**, 1210-1217 (2004)

Modules taught with the structure presented here

- SESG1009 Modelling and Computing
  approximately 200 students

  -> Matlab

- SESA2006 Computing
  approximately 100 students

  -> Python




Python - the language


- (The author of Python is fan of Monte Python)
- interpreted
- platform independent
- procedural, object oriented, functional
- large libraries, good glueing language
- clear syntax
- large and increasing user community

Python for Scientific Computation

- Python is general purpose language
- Need extensions (packages or modules) for numeric work:
  - Numeric (fast matricies [LAPACK])
  - SciPy (Scientific Python)
  - pylab (plotting like matlab)
  - Visual Python (3d programming for ordinary mortals)

Overview of material in teaching module

- introduction to programming
- using Numeric
- usinc scipy
- using visual python
- (LaTeX)

Introduction to (scientific) Python

Lecture	Lab.	Content
1 & 2		Introduction & formalities, Using IDLE, basic data types: strings, floats, ints, boolean, lists, type conversion, `range`, for-loop, if-then, importing modules, the `math` module, the `pylab` module, plotting simple functions $y = f(x)$, defining python functions, basic printing, importing python files as modules.
	1	Programs to write: 1. computer chooses random integer, user has to guess 2. finding the plural of (regular) English nouns automatically 3. plotting mathematical functions $y = f(x)$ 4. retrieve current weather conditions in Southampton from Internet (i.e. processing of text file)
3 & 4		Ordinary Differential Equations (ODEs), Euler's method in Python, Use of `Numeric` and `scipy`, use of `scipy.integrate.odeint` to solve ODEs
	2	Programs to write: 1. proving that $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ for $n = 1000$ 2. currency conversion (exercise functions) 3. implement composite trapezoidal rule for integration of $f(x)$ and evaluate convergence properties empirically 4. use of `scipy`'s `quad` for integration 5. automatic integration of function and plotting of integrand

Temperature from the internet

http://weather.noaa.gov/pub/data/observations/metar/decoded/EGHI.TXT

```
Southampton / Weather Centre, United Kingdom (EGHI) 50-54N 001-24W 0M
Mar 12, 2006 - 10:20 AM EST / 2006.03.12 1520 UTC
Wind: from the SSE (160 degrees) at 8 MPH (7 KT) gusting to 21 MPH (18 KT) (direction
variable):0
Visibility: greater than 7 mile(s):0
Sky conditions: partly cloudy
Temperature: 37 F (3 C)
Dew Point: 21 F (-6 C)
Relative Humidity: 51%
Pressure (altimeter): 30.42 in. Hg (1030 hPa)
ob: EGHI 121520Z 16007G18KT 120V200 9999 SCT034 03/M06 Q1030
cycle: 15
```

The current temperature

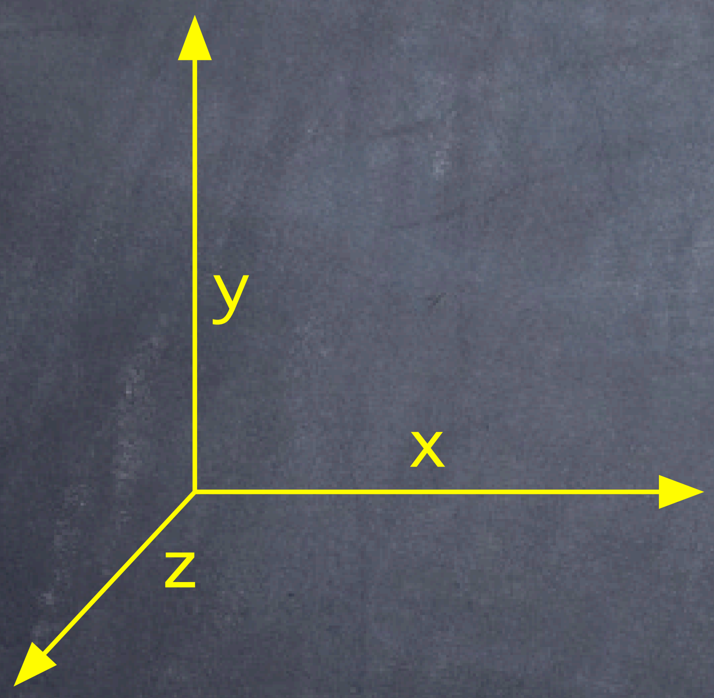
```
import urllib

#define URL location
datalocation = "http://weather.noaa.gov/pub/data/observations/
metar/decoded/EGHI.TXT"

#retrieve data
datalines = urllib.urlopen(datalocation).readlines()

#iterate over lines in weather data
for line in datalines:
    #split lines into a list of strings
    bits = line.split()

    #if the first word is 'Temperature'
    if bits[0] == 'Temperature:':
        #extract degree in C ([3]),
        #ignoring the opening parenthesis ([1:])
        temperature = float(bits[3][1:])
        #note that the conversion to float is not
        #strictly neccessary here

print "The temperature in Southampton is",temperature,"C."
```

Program output: "The temperature in Southampton is 3.0 C."

Visual Python and time dependent processess

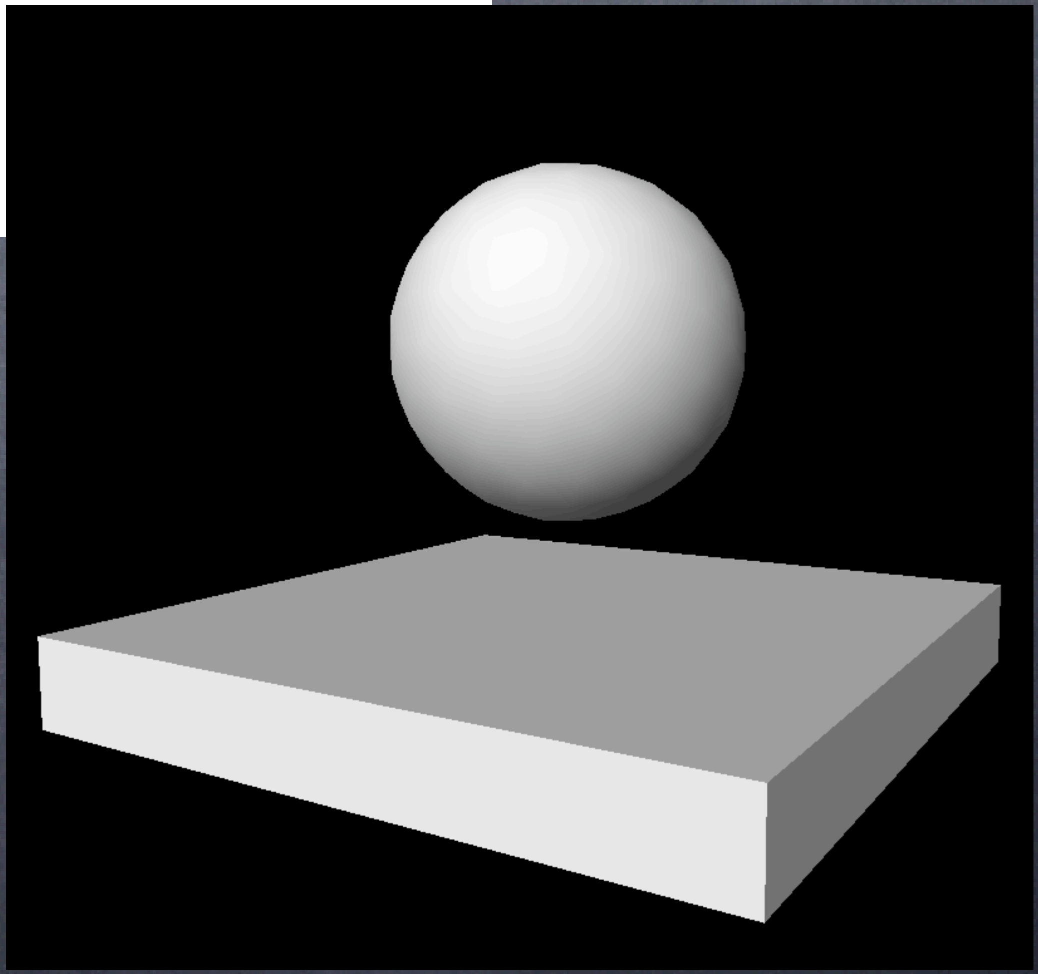
5 & 6		Introduction to Visual Python, finite differences for differentiation, Newton method for root finding. Calling Python functions with keyword arguments, name spaces, exceptions. Example code for dealing with 3d vectors and scalars.
	3	Programs to write: 1. implement a 2nd order Runge Kutta integrator for ODEs 2. solve given 1d ODE using `scipy.integrate.odeint` 3. visualise $\mathbf{r}(t) \in \mathbb{R}^3$ in real-time using Visual Python 4. compute and visualise solution to 2nd order ODE with two degrees of freedom using Visual Python
7 & 8		Finding ODEs to describe a given system. Example code dealing with time dependent 3d problems and visualisation.
	4	Programs to write: 1. Use `scipy`'s root finding tools (`bisect`) to find root of $f(x)$ 2. Use root finding and integration of ODE to solve boundary value problem ("shooting method") visualised with Visual Python 3. (Exercise on LaTeX– therefore only 2 other tasks.)

Visual Python

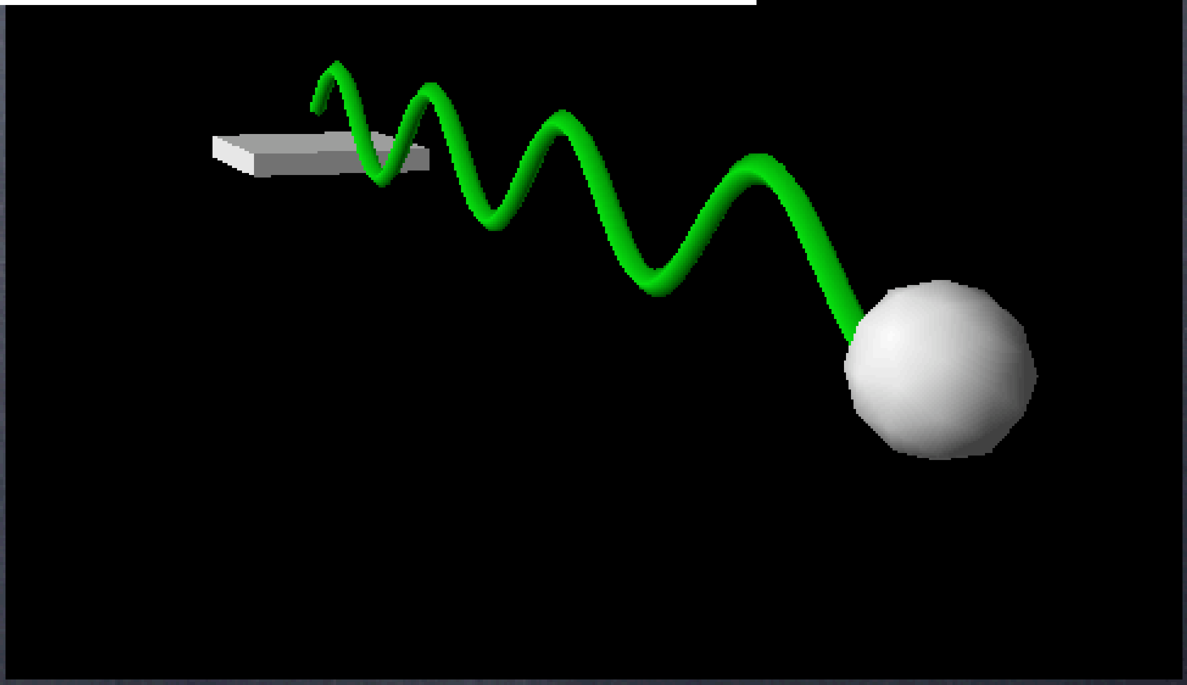


- set of 3d objects `(sphere, box, cone, spring, ...) in 3d space
- allows rotation and zoom of scene (default)
- can modify attributs of objects such as position, colour, size
- can force a certain 'frame rate'
- examples

```python
import visual, math

sphere = visual.sphere()
box = visual.box( pos=[0,-1,0], width=4, length=4, height=0.5 )

#tell visual not to automatically scale the image
visual.scene.autoscale = False

for i in range(1000):
    t = i*0.1
    y = math.sin(t)

    #update the sphere's position
    sphere.pos = [0, y, 0]

    #ensure we have only 24 frames per second
    visual.rate(24)
```

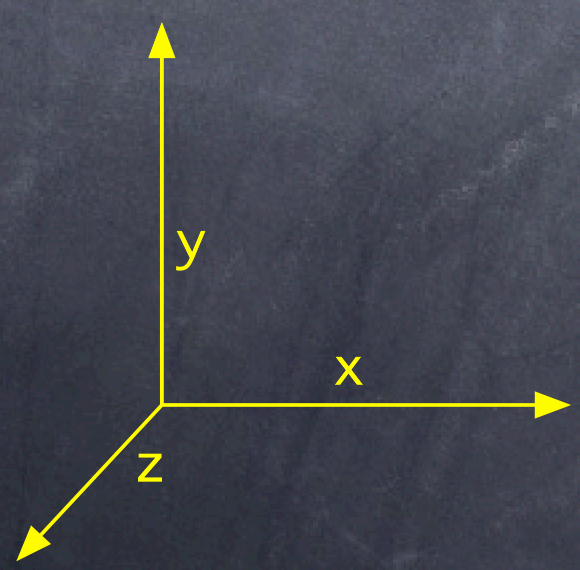


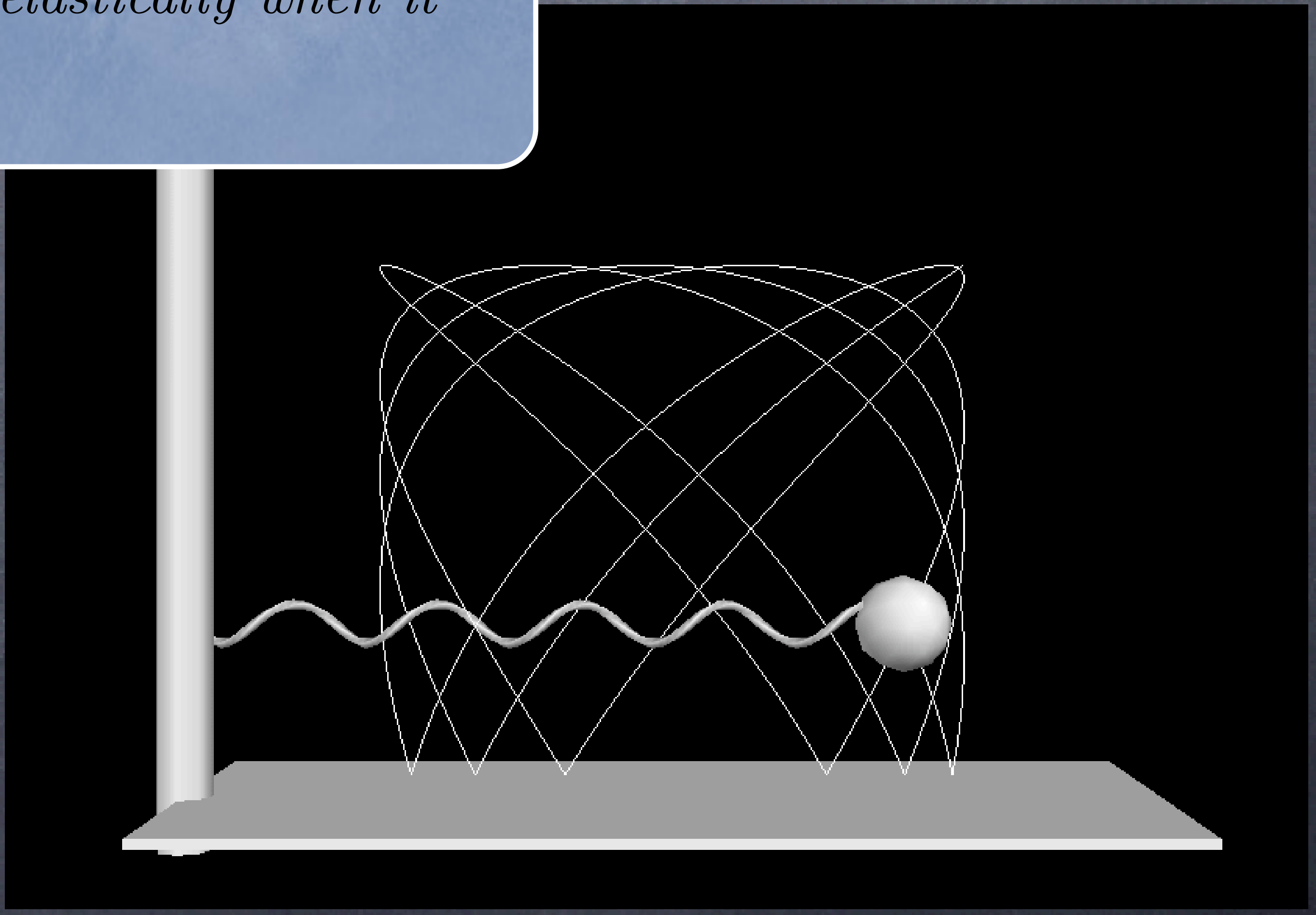
```python
import visual, math

sphere= visual.sphere()
box   = visual.box  (pos=[0,-1,0], width=4, length=4, height=0.5)
trace = visual.curve(radius=0.2, color=visual.color.green)

for i in range(1000):
    t = i*0.1
    y = math.sin(t)

    #update the sphere's position
    sphere.pos = [t, y, 0]

    trace.append( sphere.pos )

    #ensure we have only 24 frames per second
    visual.rate(24)
```



Bouncing mass on spring

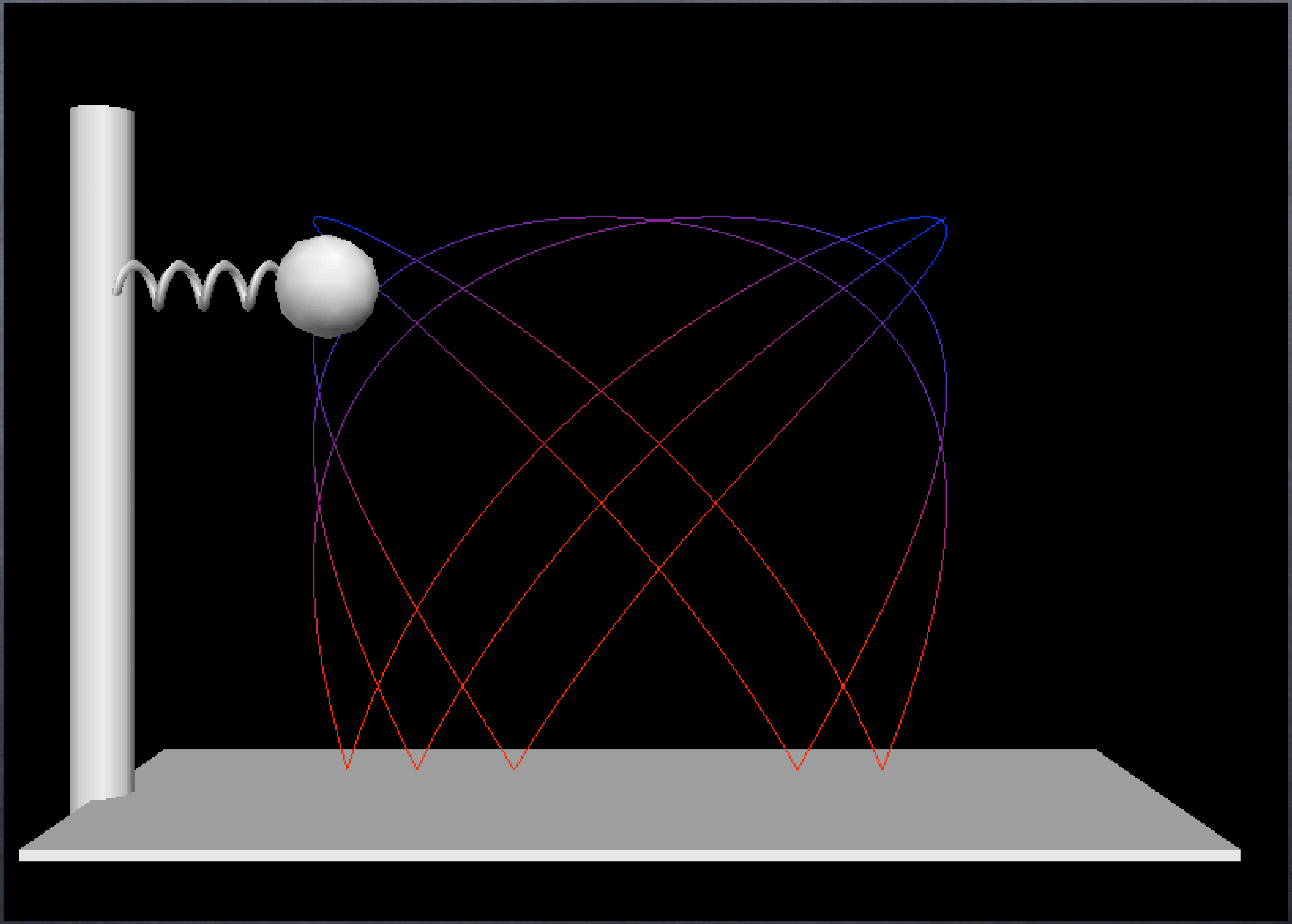
A sphere at position $\mathbf{r} = (r_\mathrm{x}, r_\mathrm{y}, r_\mathrm{z})$ *of mass* $m = 1\mathrm{kg}$ *is subject to a horizontal force* $\mathbf{F}_\mathrm{spring} = (-kr_\mathrm{x}, 0, 0)$ *and to a vertical force due to gravity* $\mathbf{F}_\mathrm{grav} = (0, -mg, 0)$*. The initial position is* $\mathbf{r}(t_0) = (3, 5, 0)\mathrm{m}$*, initial velocity* $\mathbf{v}(t_0) = (0, 0, 0)\mathrm{m/s}$ *and* $k = 5\mathrm{N/m}$*. Compute the time development of the system, assuming that the sphere will bounce elastically when it touches the ground at* $r_\mathrm{y} = 0$*.*



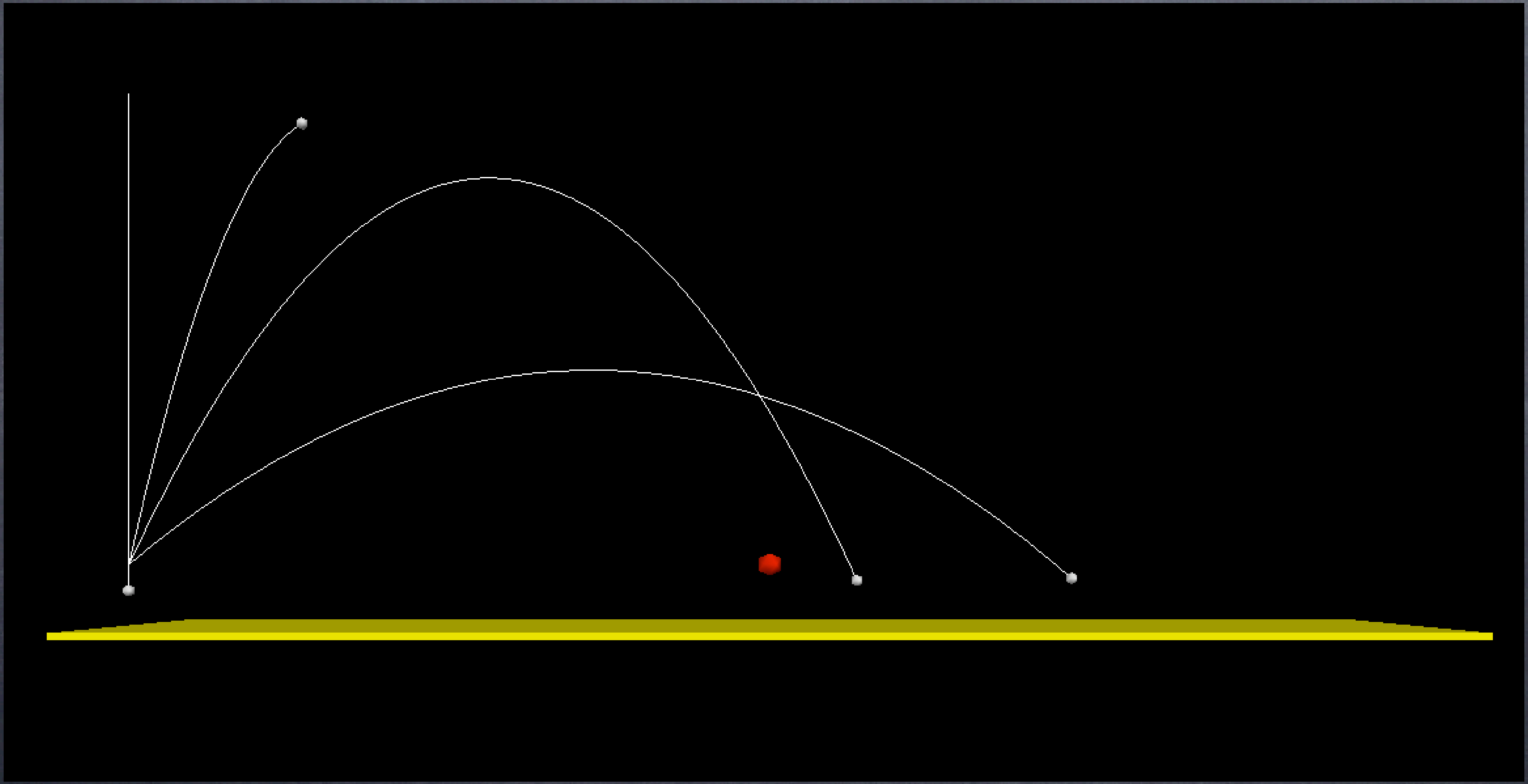


visualise kinetic energy

- red: kinetic energy high
- blue: kinetic energy low

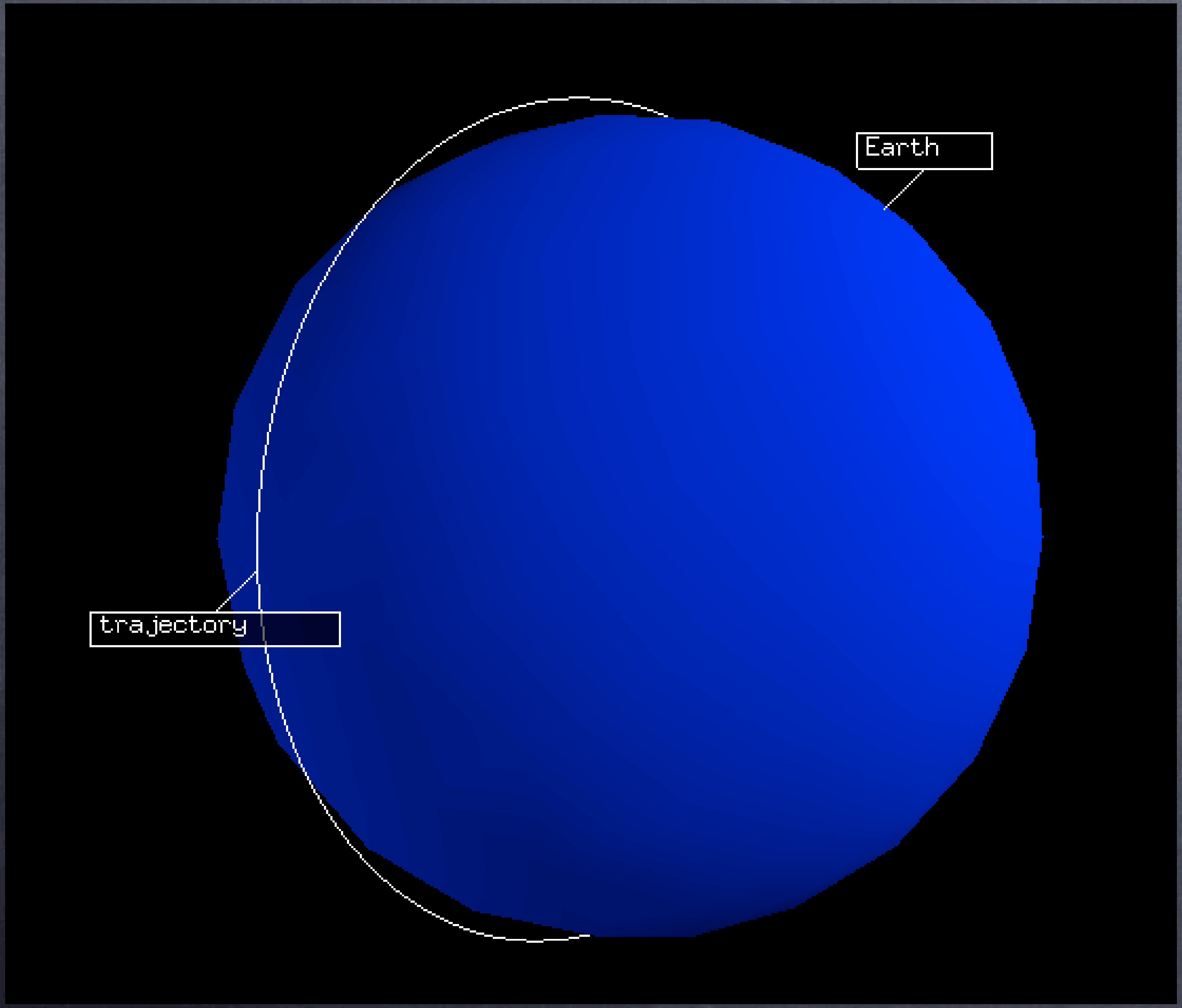


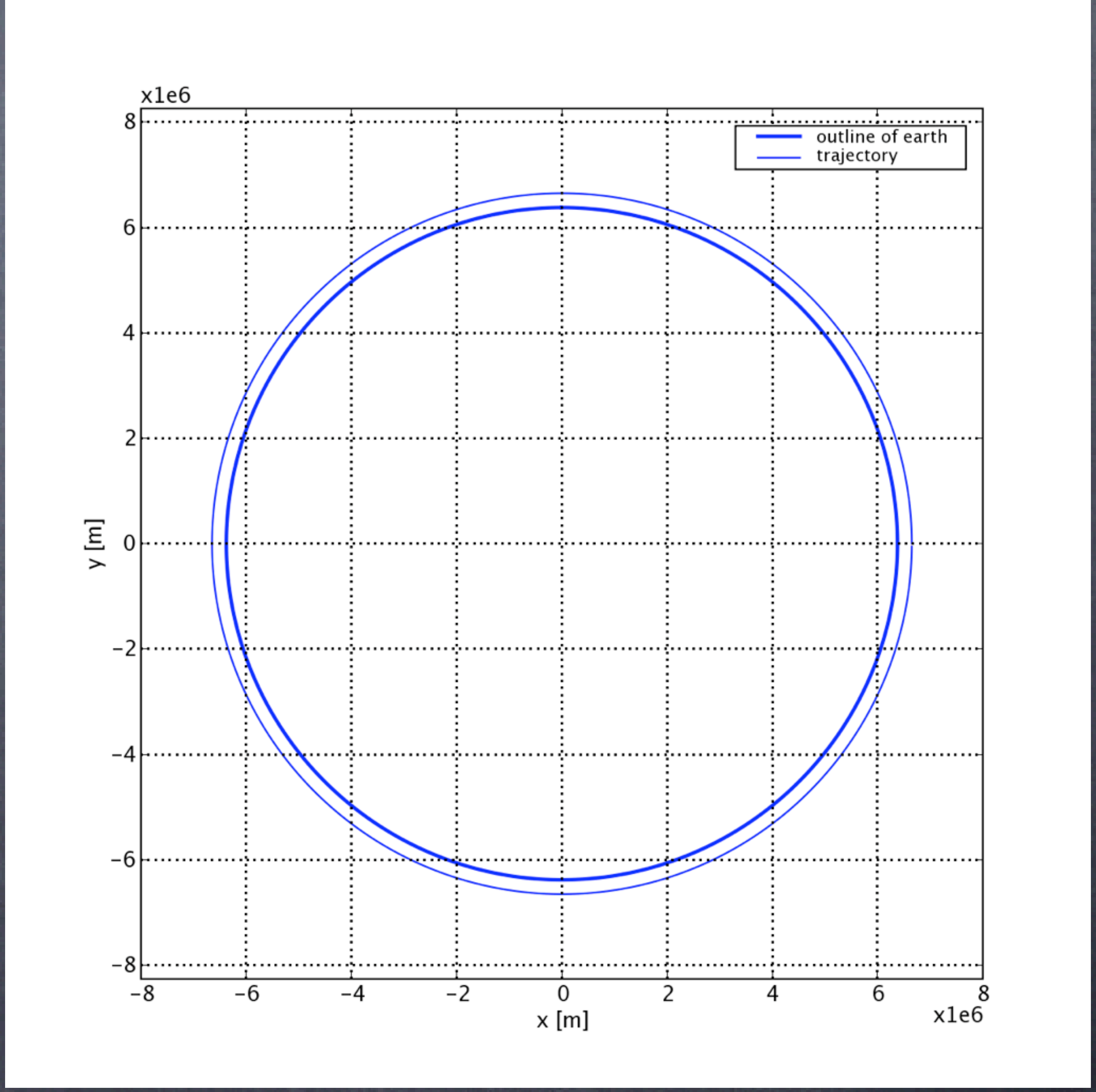
Shooting method



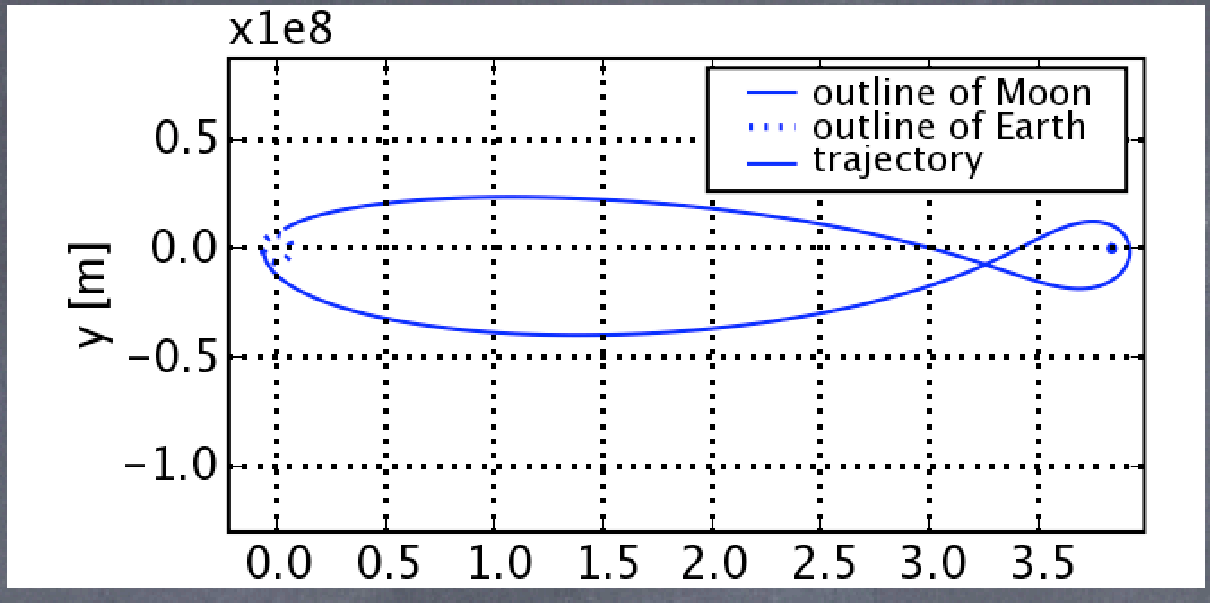
Apollo 13 mission

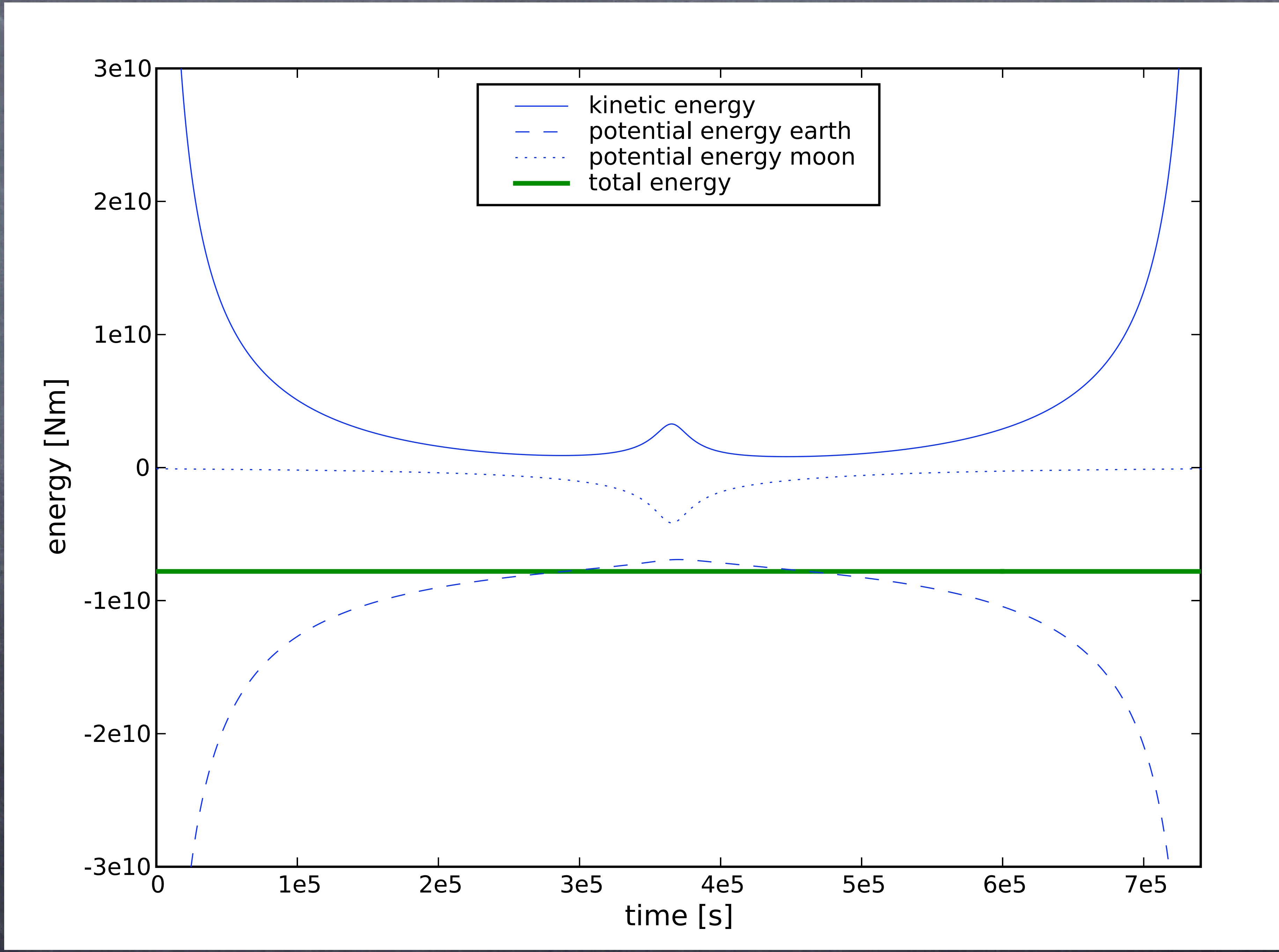
"Parking orbit"



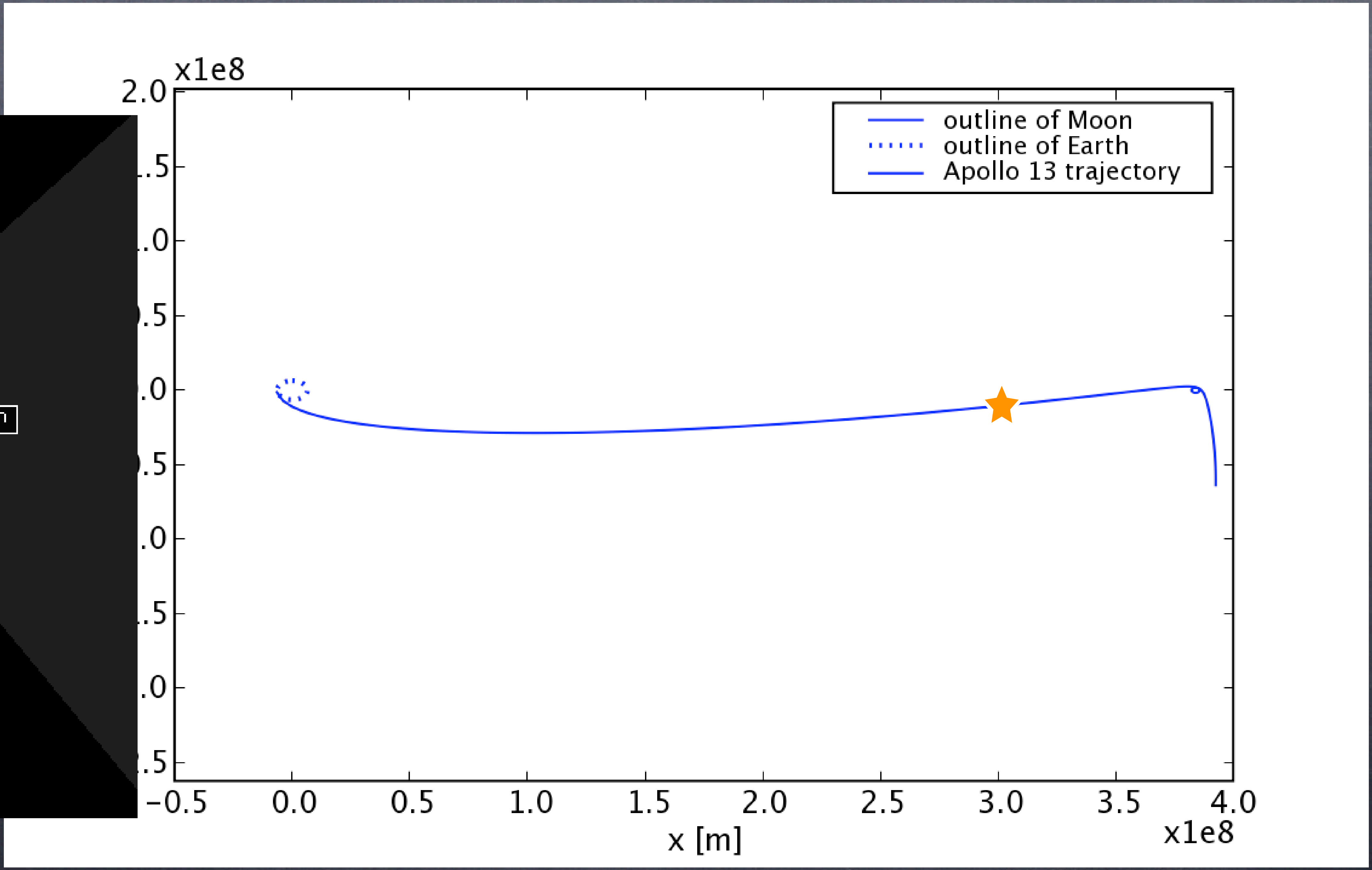


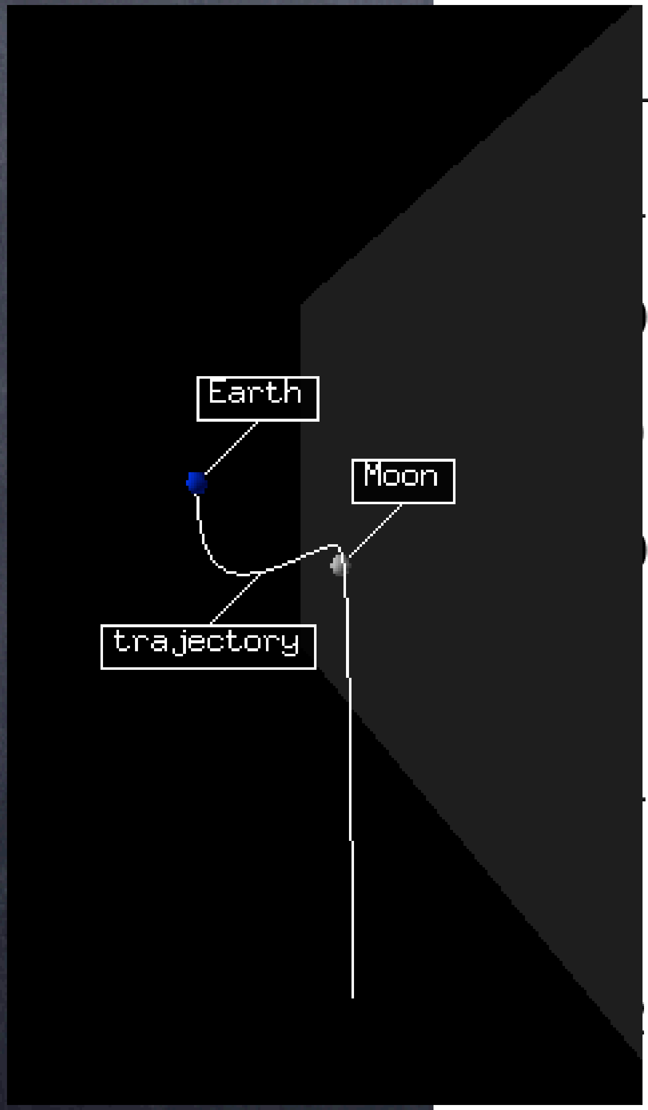
Apollo 13: free return orbit



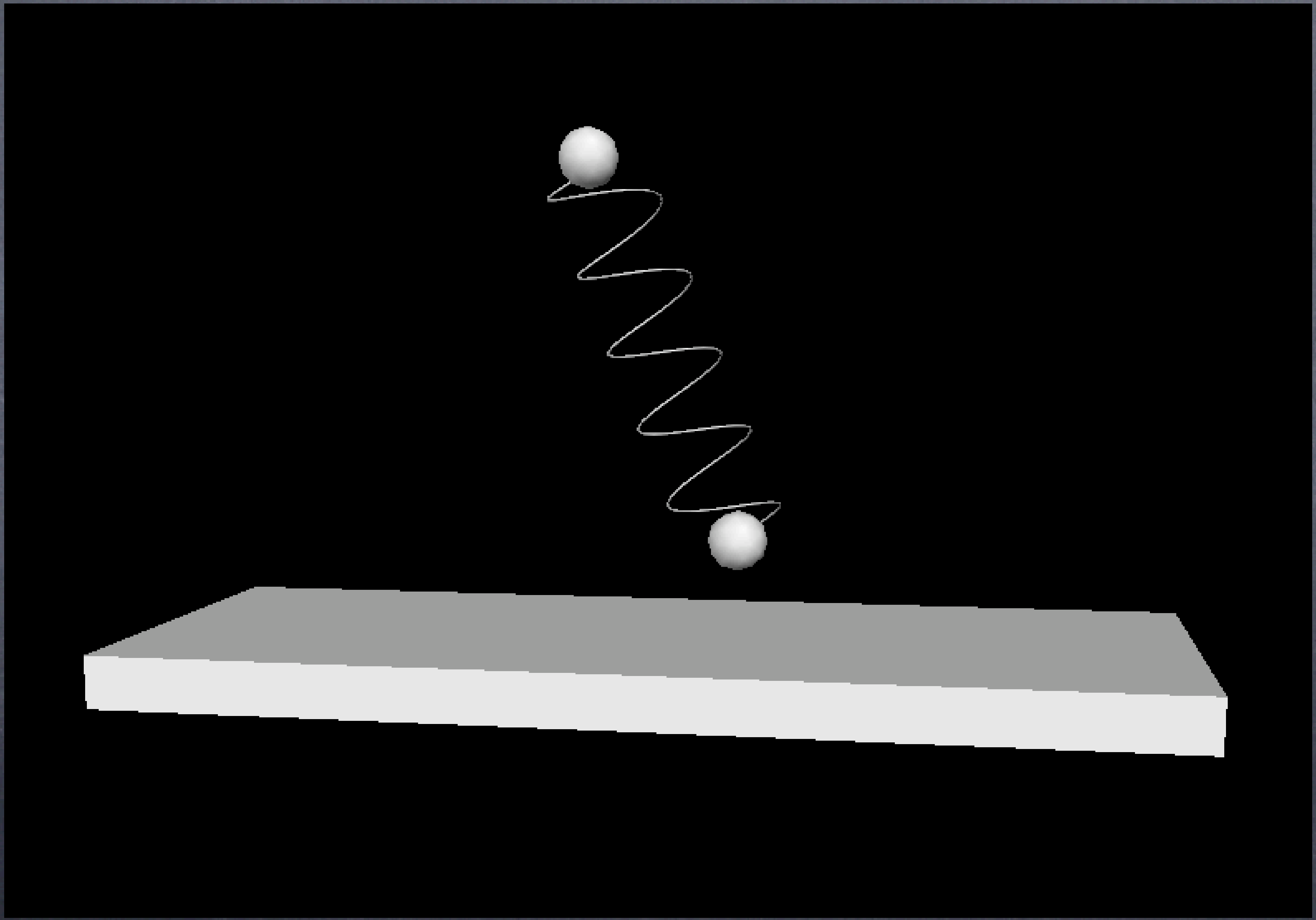


Apollo 13: Accident



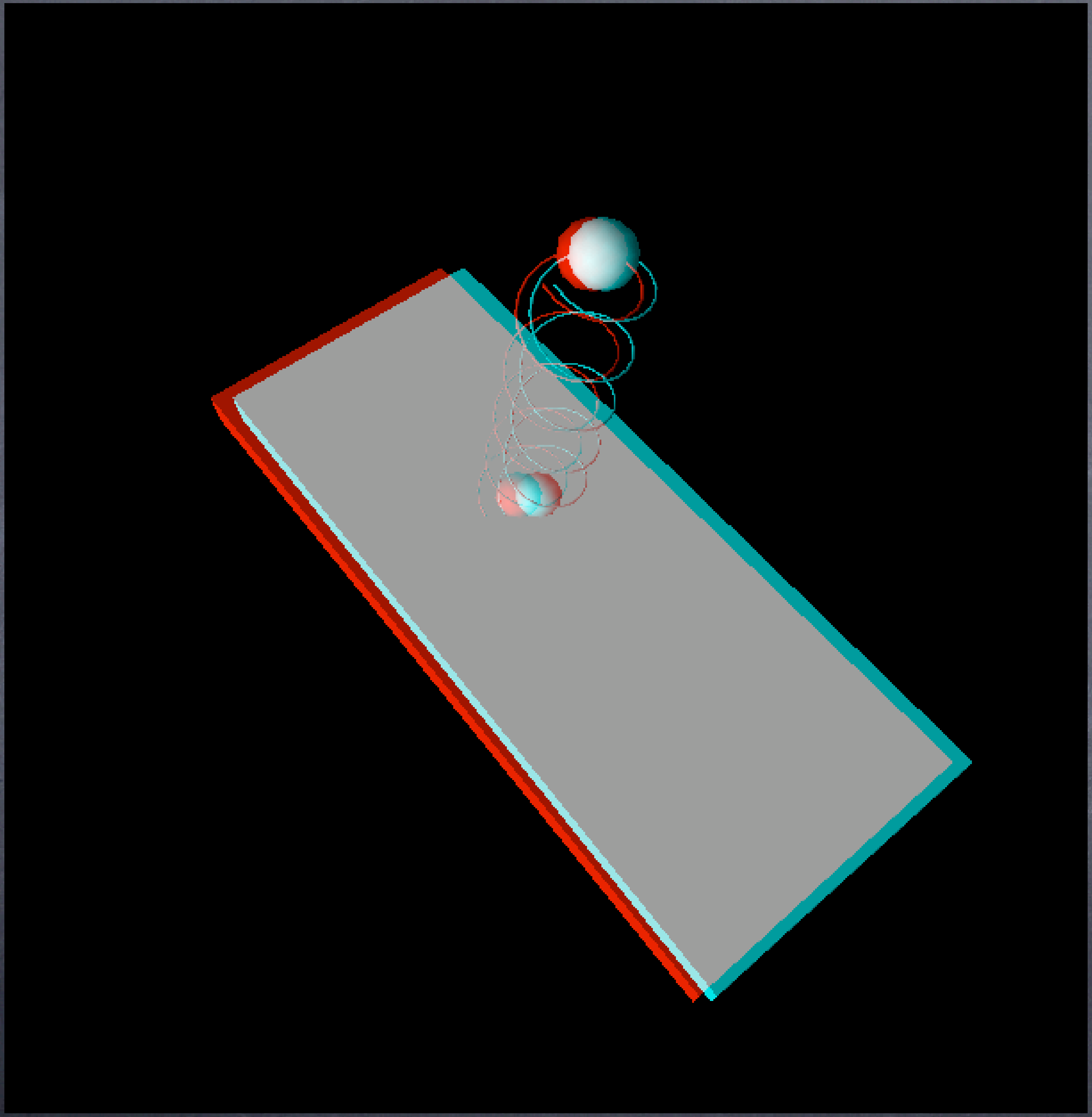


Two connected masses



3d vision

- Visual Python supports anaglyphic glasses (i.e. red-blue, red-green or red-cyan)
- just need to add
  `visual.scene.stereo=”redcyan”`
  in the beginning of the program



Summary

- Demonstrated effective teaching of large classes (in computing)
- Structure (practical laboratories)
  - each student receives regular feedback, use of demonstrators
  - problem solving exercises
- Content
  - relevant to degree
  - fun (where possible)

Summary

- Perception of computing has improved a lot (good ratings, interest in further modules)
- Did Visual Python improve the learning process: 4.2 (plus minus 0.68) (1-not at all, 5-Very much)
- Students like to have software at home (and in future)
- From student feedback questionnaires:
  "This module assesses what you can do rather than what you can remember."

*H. Fangohr. "A Comparison of C, Matlab and Python as Teaching Languages in Engineering". Lecture Notes on Computational Science **3039**, 1210-1217 (2004)
*H. Fangohr. "Exploiting real-time 3d visualisation to enthuse students: a case study of using Visual Python in Engineering" (in print) (2006)

Thank you