Advanced Topics in Computer Graphics I

Universität Bonn
Institut für Informatik II
June 22, 2017

Summer term 2016
Prof. Dr. Reinhard Klein
Alexander Dieckmann, Sebastian Merzbach

Sheet R00 - Introduction to Python

This is a voluntary exercise sheet dedicated to get a basic knowledge of Python. No points will be given, and there is no correction.

In this lecture you will have the chance to learn many interesting theoretical as well as practical topics. In any case you have a problem understanding, please always feel free to write to the mailing list. This should be a place where you students can talk freely about the lecture, so do not hesitate to ask *and* reply! Of course the tutors are on the mailing list as well and will reply as well.

Assignment 1) Installation *(0Pts)*

In this exercise we will use the latest python 2.x version, because at this time not all tools support python 3.x well yet. The distribution we use is `miniconda`, that you can download here: `http://conda.pydata.org/miniconda.html`. Later exercise sheets will contain compiled libraries for 64-bit windows. If you decide to work with a different configuration, you might have to compile those libraries yourself.

After installing you have a bare python version. Assuming that you installed into `c:\miniconda` you will have a set of scripts in `c:\miniconda\Scripts` that you probably best put in your path. The command `conda` is a simple package manager. Please install additional libraries:

```
conda install spyder numpy ipython matplotlib mayavi pep8 pillow pyface pyflakes
pyside rope sphinx spyder sympy pylint sphinx scipy tornado
```

Now you have a quite usable python installation. Execute each of `python`, `ipython`, `ipython notebook`, `ipython qtconsole`, `spyder` to see multiple different methods to interact with python. Test some examples of `http://matplotlib.org/gallery.html` on each of those command lines.

Congratulations, your python should now be ready !

Assignment 2) Python Commands *(0Pts)*

a) Create a comment.
b) Create a 1×4 row vector.
c) Create a 5×1 column vector.
d) Create a zero matrix using a function provided by numpy.
e) Print the second row of an 4×3 array.
f) Print the third column of an 4×4 array.
g) Transpose an array.
h) Create two arrays of equal size (m $\times$ m). Multiply them once using conventional matrix multiplication and once using element-wise multiplication.
i) Concatenate two arrays vertically, as well as horizontally.

j) Print the size of an array.

k) Change the structure of a 8×7 array to 14×4.

l) Replicate a 3×1 vector to an array of size 3×1000.

m) Replace all matrix entries less than 0 by 0.

n) Create a vector containing numbers from 1 to 100 with a gap of 7 between the numbers.

o) Create a vector with 100 entries. Set every second element to 0.

p) Create a vector with 100 entries. Delete every second element.

q) Create 2 arrays a, b of size 1000×3 containing random numbers.

r) You can interpret the rows of such a arrays as vectors of size 1×3. Compute the dot product of those vectors using loops. This means you have to iterate over the rows of the 1000×3 array and compute the dot product between the vectors represented by the current array row.

s) Now, try to compute the dot product without using loops. Compare the runtimes of your implementation (Hint: To measure the runtime, you can use the code snipped below or in `ipython` `%timeit`. Did you recognize something, while comparing the times (loops vs. no loops)?

```
import time
start_time = time.time()
do_some_thing()
end_time = time.time()
print('\%f seconds passed' \%(end_time-start_time))
```

t) The following scenario is given. We want to invert 1000 2×2 matrices.

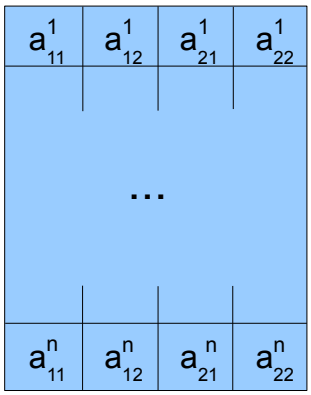

Figure 1: Memory layout of the 2×2 matrices

The 2×2 matrices are represented by a row (see Figure 1). The numbers written as subscripts represent the position of the original matrix entry of the 2×2 matrices. The superscript denotes the current array $n \in [1 \dots 1000]$. Now, create an array with a size of 1000×4 using the command `rand`. Every row in that matrix represents a 2×2 matrix (compare Figure 1). Note, that the memory layout of the 2×2 matrices must not be changed. That means we don't want to change the current structure of the 2×2 matrices represented as a row. Don't change the 1×4 matrices to 2×2 matrices and don't change the *inv* command. We can now use Cramer's Rule to compute the inverse of our 1000, as row represented, 2×2 matrices. Compute the inverses of the created 2×2 matrices without using any loops.

u) Write a function, which accepts 2 $m \times m$ matrices. The function should be able to compute the sum as well as the product of both matrices.

Good luck !