# TIEA311 Tietokonegrafiikan perusteet

kevät 2018

(“Principles of Computer Graphics” – Spring 2018)

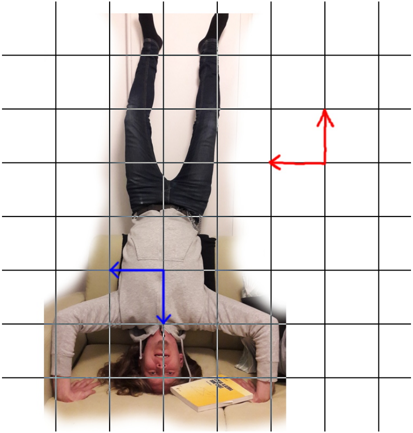
**Copyright and Fair Use Notice:**

# TIEA311 Tietokonegrafiikan perusteet – kevät 2018
# (“Principles of Computer Graphics” – Spring 2018)

Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/.

Frontpage of the local course version, held during Spring 2018 at the Faculty of Information technology, University of Jyväskylä:
`http://users.jyu.fi/~nieminen/tgp18/`

# TIEA311



Midterm

Using righthanded 2D coordinate system ($y$ opens “left” of $x$) and visual inspection of red frame $\vec{\mathbf{a}}$ and blue frame $\vec{\mathbf{b}}$, fill in the matrix:

$$\vec{\mathbf{b}}^T = \vec{\mathbf{a}}^T \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

# TIEA311 - Today in Jyväskylä

Today (if Visual Studio allows):

- Assignment 2 and 4 live. Some C++ language features weren't used in the earlier ones. Also, Assignment 4 is a bit larger code with less functionality implemented in the starter pack. Warm ups are done. Now we start working!
- C++ static member functions (i.e., "static methods")
- C++ object instantiation using constructors, operator overloading, temporary objects, pass-by-value vs. pass-by-reference
- C++ (and C) pass-by-pointer
- C++ pointer types and inheritance
- Dots, asterisks, ampersands, and arrows in C++ (and C)

# C++

- 3 ways to pass arguments to a function
  - by value, e.g. float f(float x)
  - by reference, e.g. float f(float &x)
    - f can modify the value of x
  - by pointer, e.g. float f(float *x)
    - x here is a just a memory address
    - motivations:
      less memory than a full data structure if x has a complex type
      dirty hacks (pointer arithmetic),but just do not do it
    - clean languages do not use pointers
    - kind of redundant with reference
    - arrays are pointers

# Pointers

- Can get it from a variable using &
  - often a BAD idea. see next slide
- Can be dereferenced with *
  - float *px=new float; // px is a memory address to a float
  - *px=5.0; //modify the value at the address px
- Should be instantiated with new. See next slide

# Pointers, Heap, Stack

- Two ways to create objects
  - The BAD way, on the stack
    - myObject *f() {
      - myObject x;
      - ...
      - return &x
    - will crash because x is defined only locally and the memory gets de-allocated when you leave function f
  - The GOOD way, on the heap
    - myObject *f() {
      - myObject *x=new myObject;
      - ...
      - return x
    - but then you will probably eventually need to delete it

# Segmentation Fault

- When you read or, worse, write at an invalid address
- Easiest segmentation fault:
  - float *px; // px is a memory address to a float
  - *px=5.0; //modify the value at the address px
  - Not 100% guaranteed, but you haven't instantiated px, it could have any random memory address.
- 2nd easiest seg fault
  - Vector<float> vx(3);
  - vx[9]=0;

# Segmentation Fault

- TERRIBLE thing about segfault: the program does not necessarily crash where you caused the problem
- You might write at an address that is inappropriate but that exists
- You corrupt data or code at that location
- Next time you get there, crash

- When a segmentation fault occurs, always look for pointer or array operations before the crash, but not necessarily at the crash

# Debugging

- Display as much information as you can
  - image maps (e.g. per-pixel depth, normal)
  - OpenGL 3D display (e.g. vectors, etc.)
  - cerr<< or cout<< (with intermediate values, a message when you hit a given if statement, etc.)
- Doubt everything
  - Yes, you are sure this part of the code works, but test it nonetheless
- Use simple cases
  - e.g. plane z=0, ray with direction (1, 0, 0)
  - and display all intermediate computation

# Questions?

# TIEA311 - Today in Jyväskylä (in Finnish)

The “steps of Jarno” (Ajattelumallia tehtävien ratkaisuun):

1. Luentomateriaali
2. Tehtävänanto (muista mitä aiemmissa tehtävissä on tehty/annettu)
3. Hae lähdekoodi ja testaa sen toiminta
4. Yhdistä teoria tehtävään ja lähdekoodiin, ymmärrä kokonaisuus
5. Hahmottele kevyt ”speksi” esim. paperille UML, prosessikaavio, ...

———————————————-

6. Tee osatehtävä 1
7. Päivitä ”speksi”
8. Tee osatehtävä 2
9. Päivitä ”speksi”

...

# MIT EECS 6.837 Computer Graphics

# Part 2 – Rendering

## Today: Intro to Rendering, Ray Casting

NVIDIA