UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

INTRODUCTION TO MODERN C++

Lecture / Lab 6

Rémi Géraud

March 17, 2016

École Normale Supérieure de Paris

EASTER EGG!

- QEM/MMEF: We need 1 additional Lab session (because Easter)
- Everybody: We need 1 additional Lecture (because 1st session)

Lecture / Lab 6

A short review, the Path Onward, and some Machine Learning.

WHAT YOU KNOW NOW

- Basic program structure
- Variables and types
- Algorithms, branches, loops
- A few things about memory
- Object oriented design

WHAT YOU KNOW NOW

- Basic program structure
- Variables and types
- Algorithms, branches, loops
- A few things about memory
- Object oriented design

And you experimented it on

- BBP-type computation of π
- Black-Scholes European call option pricing
- Euler integration of ODEs for ballistics
- Monte-Carlo integration of the n-dimensional sphere
- Portfolio evaluation in the geometric Brownian motion model
- Markov chain evolution of credit risk rating

FROM NOW ON

From now on we will focus more on *specific applications*:

- Graphics and multimedia
- Machine learning
- Networking
- Simulation and scientific computation
- Multithreading
- User interfaces

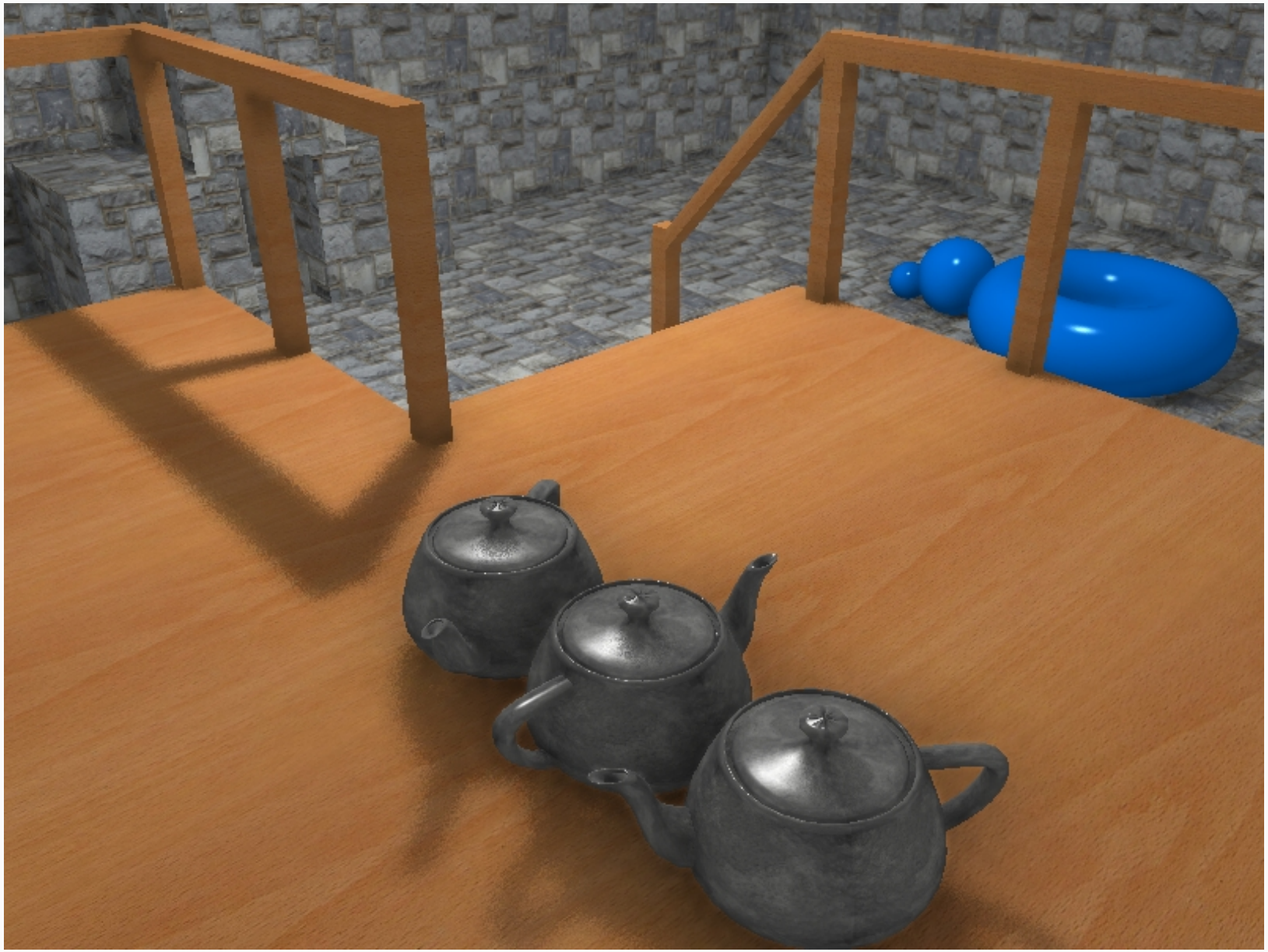
ALSO, THE FINAL PROJECT

Remember you should do a final project to validate this course

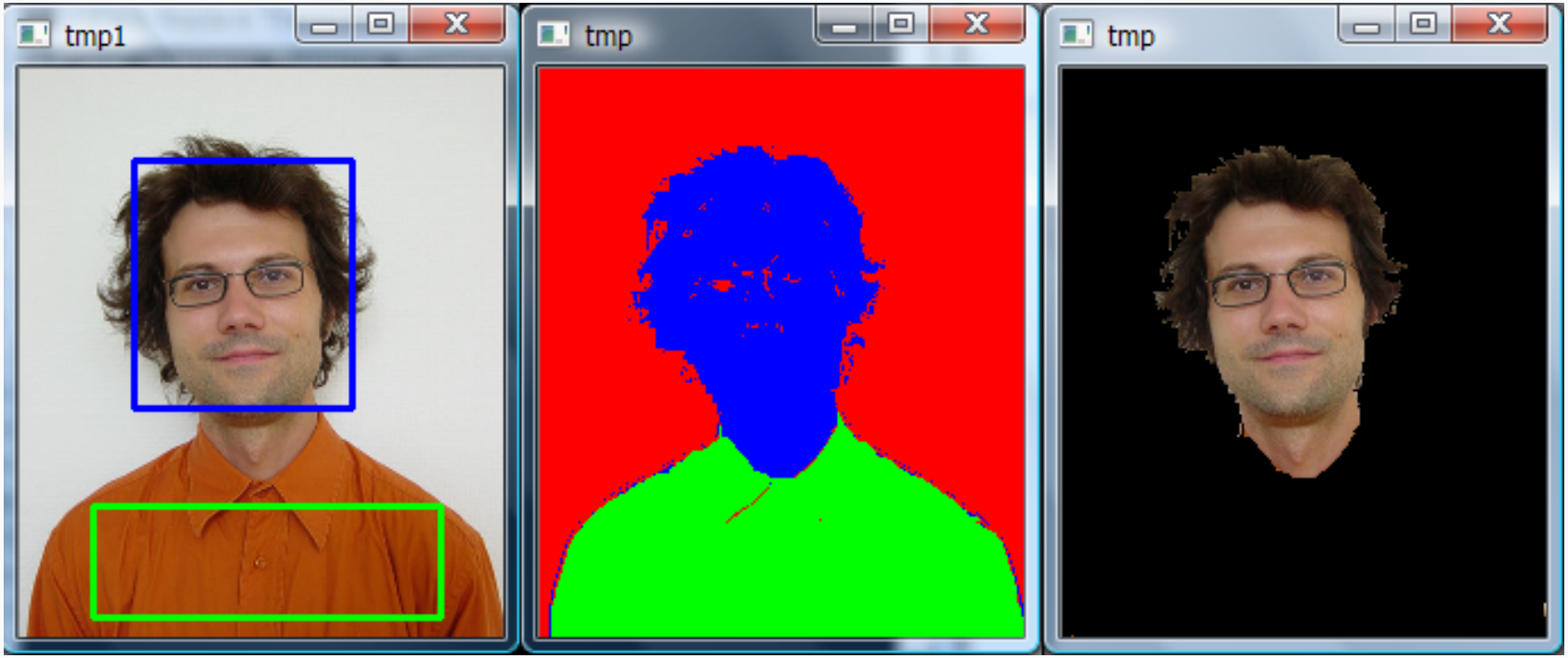
- Select a topic
- Write a brief project plan **which has to be approved**
- Do the project
- Make a presentation + demo
- Provide all source code to me.

In teams of 3-5 students.

teasers: 3d graphics



TEASERS: COMPUTER VISION



TEASERS: SCIENTIFIC SIMULATION

TEASERS: USER INTERFACES

TEASERS: HOW?

You can achieve this kind of things by using:

- Advanced C++ features (threading, sockets, STL...)
- 3rd-party libraries (SDL, Qt, OpenGL, OpenCV...)

And most probably a combination of both.

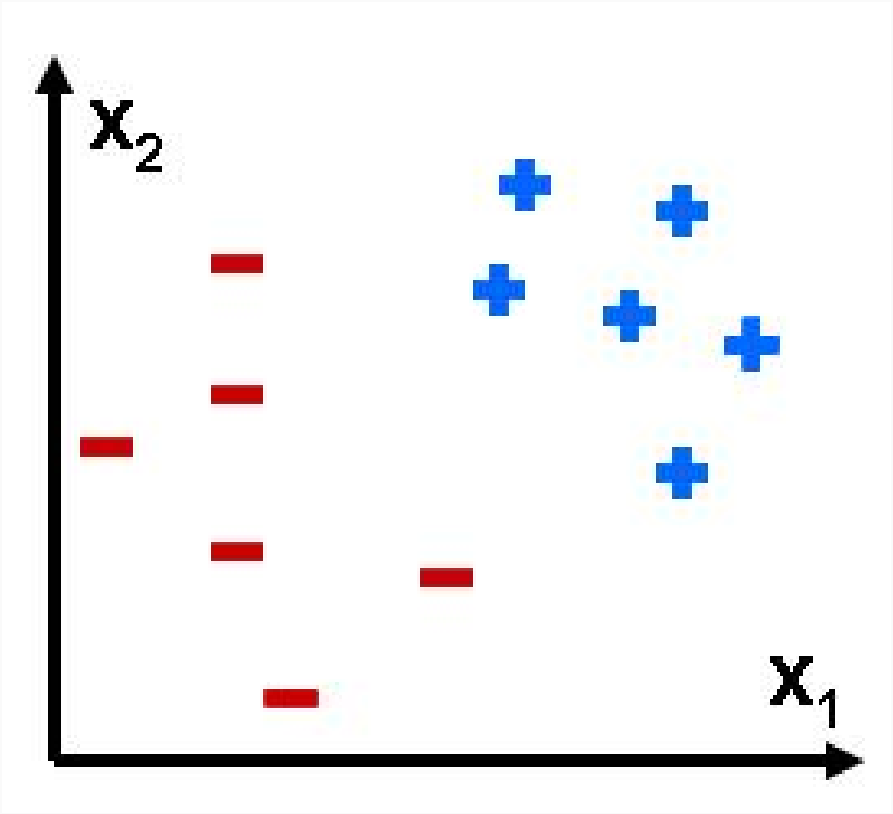
We will do slightly simpler versions of these (e.g. Angry birds) but you will have all you need.

Hint: **Good design** is key. The rest is ”just” work.

THE PERCEPTRON

THE PERCEPTRON

The perceptron is one of the first historic machine learning algorithms. It is a *classifier* (like SVMs, naïve Bayes, etc.)



Goal: Find a line that separates (+) from (−). Why?

Recall that the equation of a line is: $y = ax + b$.

- $y > ax + b$ means we are *above the line*
- $y < ax + b$ means we are *under the line*

If we are given a *new* point (x, y) we can say whether it is above or under the line, i.e. **we can classify it automatically**.

The perceptron algorithm tells us the line equation that we need.

THE PERCEPTRON ALGORITHM

We need the following ingredients:

- *Training samples* $X = \left\{ \vec{x}_1 = \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}, \ldots, \vec{x}_n \right\}$
- *Answers for them* $\vec{T} = (t_1, \ldots, t_n)$ (each t_i is -1 or +1)

We are looking for a vector of *weights* $\vec{W} = (w_1, w_2, w_3)$ such that

$$\text{For each } \vec{x}_i \text{ in } X, \ \text{sign}(\vec{W} \cdot \vec{x}_i) = t_i$$

Question: Why 3D vectors with a component set to 1?

Central idea of the perceptron: Look at samples one after the other and update weights as we go.

THE PERCEPTRON ALGORITHM

I. Learning phase:

1. Initialise $X, \vec{T}$, and set $\vec{W} = 0$.
2. For each training sample $\vec{x}_i$:
   (i) Compute the "prediction" $u = \text{sign}(\vec{W} \cdot \vec{x}_i)$
   (ii) If $u \neq t_i$, update the weights:

$$\vec{W} \leftarrow \vec{W} + \lambda(t_i - u)\vec{x}_i$$

Here, $\lambda > 0$ is a small parameter.

II. Once trained,

- Input: Some sample $\vec{x}$.
- Output: $\text{sign}(\vec{W} \cdot \vec{x})$.

THE PERCEPTRON ALGORITHM

Task:

- Create a class `Vector3D` and functions `dot`, `mul` and `add`
- (Bonus challenge: Do that with `operator+` and `operator*`)
- **Implement the Perceptron algorithm**
- Test it. What are some limitations?

Bonus question: Find the best value of λ.

Bonus question: Can you think of another approach to classify data?

Bonus question: Generalise the perceptron algorithm to n-D vectors.

Questions?

HINT

```
class Vector3D {
  public:
    double x, y, z;
    Vector3D(double x, double y, double z) :   // Constructor
       x(x), y(y), z(z)                        // Init list
       {};

    // Addition of two vectors
    Vector3D operator+(const Vector3D& other) const;
    // Multiplication of a vector by a scalar
    Vector3D operator*(const double& other) const;
    // Dot product between two vectors
    double   operator*(const Vector3D& other) const;
};
```

```
int main() {
    Vector3D v {1, 2, 3};
    // ...
}
```