Introduction to Programming Final Project

- Instructor : Andrew Yarmola andrew.yarmola@uni.lu
- Course Schedule : Wednesday 14h00 - 15h30 Campus Kirchberg B21
- Course Website : sites.google.com/site/andrewyarmola/itp-uni-lux
- Office Hours : Thursday 16h00 - 17h00 Campus Kirchberg G103 and by appointment.

The Final Project will be due February 9th and will be submitted on GitHub. You will work in teams of two. Your team can choose any one of the following projects :

1. Perfectly reflecting 2D Waves (PDE Solver)
2. A tool for audio syncing using fast Fourier transforms.
3. Fractal viewer
4. A tool for handwritten digit recognition using machine learning or a neural network
5. A tool for counting the number and size of objects in an image/video

I strongly encourage that different teams pick different projects, but I am not enforcing this for the Final Project. The descriptions below might be a little sparse, so please don't hesitate to ask for help and any clarification.

All code must be written by the two members of your team, but you can and should use examples to learn.

Suggested Exercise

If you want to have a little practice before working on the main project, I suggest you you try to make a simple calculator application.

- Learn about the `.grid` layout manager here : http://effbot.org/tkinterbook/grid.htm and here : https://infohost.nmt.edu/tcc/help/pubs/tkinter/web/grid.html
- use for-loops to draw (most of) your buttons

1. Perfectly reflecting 2D Waves

For this project, you will write a finite difference PDE solver for the 2D wave equation

$$\frac{\partial^2 u}{\partial t^2} - c^2 \nabla^2 u = \frac{\partial^2 u}{\partial t^2} - c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = 0$$

where $u(x, y, t)$ is a function of two spacial variables x, y and a time variable t. See a demo video here : https://www.youtube.com/watch?v=aGlFwKXVQfc

We will restrict ourselves to a rectangular domain and assume that our wave has perfect reflection at the boundary. The latter assumption is called the homogeneous Neumann boundary condition expressed as

$$\mathbf{n} \cdot \nabla u = 0.$$

Here $\mathbf{n}$ is the outward normal to the boundary. Our domain will just be a rectangular box $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [0, t_{max}]$ so the boundary normals will be simple.

Our the initial conditions will be given as two functions :

$$u(x,y,0) = f(x,y) \quad \text{and} \quad \frac{\partial u}{\partial t}(x,y,0) = g(x,y)$$

The full input to the problem will be

- the constant c (this is the wave propagation speed)
- $f(x,y)$ and $g(x,y)$ given as equations
- x_{min} , x_{max} , y_{min} , y_{max} , and t_{max} (we assume $t_{min} = 0$)
- n_x , n_y , and n_t (these are the number of samples)

Central finite difference and discretization

We will discretize our domain by dividing $[x_{min}, x_{max}]$ into n_x evenly spaced samples $\{x_i\}$, including endpoints. Similarly, subdivide $[y_{min}, y_{max}]$ into n_y samples $\{y_j\}$ and $[0, t_{max}]$ into n_t samples $\{t_k\}$. The solution will be the discretized 3 dimensional array $u_{i,j}^k \approx u(x_i, y_j, t_k)$.

On our domain, we want to discretize the second order derivatives as

$$\frac{\partial^2 u}{\partial t^2}(x_i, y_j, t_k) \approx \frac{u_{i,j}^{k+1} - 2\, u_{i,j}^k + u_{i,j}^{k-1}}{\Delta t^2}$$

and similarly with respect to x, y.

We also discretize the initial conditions as

$$u_{i,j}^0 = f(x_i, y_j) \quad \text{and} \quad \frac{\partial u}{\partial t}(x_i, y_j, 0) \approx \frac{u_{i,j}^1 - u_{i,j}^{-1}}{2\Delta t} = g(x_i, y_j).$$

Using two steps to approximate $\frac{\partial u}{\partial t}(x_i, y_j, 0)$ is why this is called the **central** finite difference method. Notice that $u_{i,j}^{-1}$ isn't actually defined on our domain, so we will **use** the above equation to give the value.

Boundary conditions and recursion

The boundary conditions also discretize nicely. In the x direction, one has $\mathbf{n} \cdot \nabla u = \pm \frac{\partial u}{\partial x}$. For example, on the $x = x_{\min}$ bounrady, one gets:

$$-\frac{\partial u}{\partial x}(x_0, y_j, t_k) \approx \frac{u_{1,j}^k - u_{-1,j}^k}{2\Delta x} = 0$$

In particular, this let's you define the value $u_{-1,j}^k = u_{1,j}^k$.

This, along with the $g(x,y)$ initial condition, let's you define $u_{-1,j}^k, u_{n_x,j}^k, u_{i,-1}^k, u_{i,n_y}^k, u_{i,j}^{-1}$, and $u_{-1,j}^{n_t}$ using only points on the interior of your domain.

Starting with $u_{i,j}^0 = f(x_i, y_j)$ and the discretized wave equation, **your job** is to write down a formula for $u_{i,j}^{k+1}$ in terms of $u_{i,j}^k, u_{i,j}^{k-1}, u_{i\pm1,j}^k$, and $u_{i,j \pm 1}^k

Hint : When writing the discrete version of the wave equation and solving for $u_{i,j}^{k+1}$ I suggest you use the **Courant numbers** :

$$\gamma_x = c\frac{\Delta t}{\Delta x} \quad \gamma_y = c\frac{\Delta t}{\Delta y} \quad \gamma = 2\left(1 - \gamma_x^2 - \gamma_y^2\right)$$

Programming Tasks

Your goal will be to make a user interface with the following features :

- make sure you figure our the recursion formulas above and be especially careful at the boundary
- you should write a solver class that implements the numeric solution
- your interface should have inputs for c, x_{min} , x_{max} , y_{min} , y_{max} , t_{max} , n_x , n_y , and n_t
- your interface should have $f(x, y)$ and $g(x, y)$ as inputs using python style syntax (e.g. `exp(-10*(x**2+y**2))`)
- your should have a 2D or 3D `matplotlib` animation plot of your solution embedded in your interface
    - if you choose to use a 2D plot, the colors should indicate the value of u.
- don't forget an "solve and animate" button

2. A tool for audio syncing using fast Fourier transforms.

This project is about making a somewhat useful audio editing tool. The idea is as follows : imagine you record a Skype conversation, but the audio quality is very poor. Luckily, each person recored their own voice in high quality using a nice microphone. You would like to sync all the high quality audio steams together into a version of the conversation. However, everyone records their audio slightly differently and they didn't all start recording right at the same time. The idea is to create a nice algorithm for aligning the high quality audio streams to the original (poor quality) Skype call.

The algorithm

Our input is as follows :

- one (poor quality) full conversation audio recording
- two (or more) high quality recording of each person in the conversation

Our task is to alight and merge the high quality audio using the full conversation as our "ruler".

The idea on how to do with is simple : cut each audio steam into really short segments (~10 ms, for example) and apply (windowed) fast Fourier transforms to each slice. The transforms will containing the data of the **most dominant** frequencies in each short segment. You will then alight the segments by matching dominant frequencies from the high quality audio streams to the poor quality one. This should let you compute your timing offset.

After checking that your timing offsets work everywhere, you can then shift and mix (i.e. add) the high quality audio together. Alternatively, you can try just adding the matched high quality segments one at a time to where they belong.

Recording your audio and loading it into python

I suggest you record a conversation on a laptop while each of you records your own voice on your cellphone. Alternatively, if you want, I can try to provide audio samples for you.

Use Audacity http://www.audacityteam.org to convert your audio data to WAV files.

Use `scipy.io.wavfile` to read the files as `numpy` arrays into your code. You should also use this to write your final file.

Programming Tasks

Your goal will be to make a program that can be used both as a GUI **and** at the command line :

- it should just be one program that runs a GUI if there are no command line arguments, but runs as a command line program if arguments are supplied.
- for asking users for file locations, see `tkinter.filedialog`
- on the GUI, only allow for 2 high quality streams, but allow the command line to use more.
- embed a graph of the final (high wuality) audio wave in your interface.
- use nice command line arguments to specify the file to sync to, the name of the output, and the high quality audio files.

3. Fractal viewer

The goal for this project is to make a tool for visualizing the Mandelbrot and (quadratic) Julia sets. There are slightly different dynamical objects, so let me explain each individually.

Mandelbrot set

Let $f_c(z) = z^2 + c$, then the Mandelbrot set $\mathscr{M}$ is
$$\mathscr{M} = \{c \in \mathbb{C} \mid \limsup_{n\to\infty} |f_c^n(c)| < \infty\}$$
where f^n denotes order n composition of f_c with itself. The most interesting part of $\mathscr{M}$ will line in the box $\{x + iy \mid x \in [-2, 1] \text{ and } y \in [-1.5, 1.5]\}$.

Quadratic Julia sets

For $c \in \mathbb{C}$, one defines the quadratic Julia set $\mathscr{J}_c$ as the smallest closed set containing at least three points which is completely invariant under f_c. Another definition is that $\mathscr{J}_c$ is the **set of limit points** of $\bigcup_{z\in\mathbb{C}} \bigcup_{n>0} f_c^{-n}(z)$. In fact, $\mathscr{M}$ is the set of parameters c such that $\mathscr{J}_c$ is connected.

Neither of the definitions is very good to compute with. The best method is to compute the **distance** of a pixel to $\mathscr{J}_c$.

Distance estimator and potential

The way to estimating distance to a Julia set and to the Mandelbrot set are actually quite similar. For a detailed discussion, see here. I will give a quick overview.

For Julia sets

Let $p \in \mathbb{C}$ be a "pixel." To estimate the distance $d_{\mathscr{J}_c}(p)$ of p to $\mathscr{J}_c$ we consider the sequence of points $w_k = f_c^k(p)$ with $w_0 = p$ **as functions of** p. We will also need to understand how these vary with respect to p. Let $w'_k = dw_k/dp$. The main ingredient is the Hubbary-Douady potential
$$G(p) = \lim_{k\to\infty} \frac{\log|w_k|}{2^n}.$$
The distance $d_{\mathscr{J}_c}(p)$ is then
$$d_{\mathscr{J}_c}(p) = \frac{G(p)}{|G'(p)|} = \lim_{k\to\infty} \frac{|w_k| \log|w_k|}{|w'_k|}$$
Since $w'_{k+1} = 2 w_k w'_k$ and $w'_0 = 1$ by the definition of f_c, it is easy approximate $d_{\mathscr{J}_c}(p)$ by taking k large enough.

For Mandelbrot set

This will be very similar to the Julia set setting, however, our parameter will be by a point $c \in \mathbb{C}$. Let $z_k = f_c^k(c)$ and $z_0 = c$. Also, define $z'_k = dz_k/dc$, **notice that we differentiate with respect to** c. The potential and distance functions are actually defined the same way, so
$$d_{\mathscr{M}}(c) = \frac{G(c)}{|G'(c)|} = \lim_{k\to\infty} \frac{|z_k| \log|z_k|}{|z'_k|}$$
However, $z'_{k+1} = 2 z_k z'_k + 1$ and $z'_0 = 1$. Again, taking k large enough, it is possible to approximate the

distance.

Programming Tasks

Your goals for the project will be to make a GUI for viewing these fractals in **color** and allow for **zooming**. You should

- have an area to display the image
- allow for both black/white and a color version of the image by using the distance to create gradual color gradients
    - have a switch for going from color to black/white.
- have a button to switch between showing the Mandelbrot set and Julia sets
- have an input field for specifying the c parameter for Julia sets
- allow for 3x zooming when the user clicks a point on the image
    - you don't need to have smooth zooming, just update the image to show a better quality version of the zoomed area
- you will also need a zoom-out button for going back a step and a reset button to go back to the original box.

4. A tool for handwritten digit recognition using machine learning or a neural network

In this project, you will use machine learning to recognize hand-written digits (i.e. 0,...,9). This is a very famous application of machine learning algorithms and is commonly used by the Post. All machine learning algorithms require a training data set. For this we will use the MNIST data set available at http://yann.lecun.com/exdb/mnist/

You have **two** options on how to implement you recognition algorithm and training :

- a **neural network** - a large collection of "neurons" trained to solve a problem
    - coded from scratch, the algorithm can be done in under 75 lines of code
    - very good step-by-step introduction on how to code and design your neural network available here : http://neuralnetworksanddeeplearning.com/chap1.html
- a **support vector machine** - a supervised learning model that chooses buffer regions between clusters of data
    - can use the built-in SVM as part of the module `sklearn` (called scikit-learn)
    - you can learn how to use scikit-learn and SVMs from our textbook http://www.scipy-

lectures.org/packages/scikit-learn/index.html

It's up to you to decide with method your prefer.

Programming Tasks

Your goals for the project will be to make a GUI for testing your trained recognizer by drawing digits with a mouse pointer.

- train your algorithm on the MNIST data set
- use built-in methods or devise a way to **save you trained state** (not very useful if you have to train your program every time you use it)
- make a GUI with a small square canvas on which you can **draw digits with the mouse**
- have a display area and a button that will run your algorithm on the drawn image, recognize, and then display the result
  - to get from a tkinter canvas to an image that you can feed to your recognition algorithm, use the `.postscript(file=, colormode=)`
  - you will probably have to resize your image, use `scipy.misc.imresize`
- have a clear canvas button to clear the canvas

5. A tool for counting the number and size of objects in an image/video

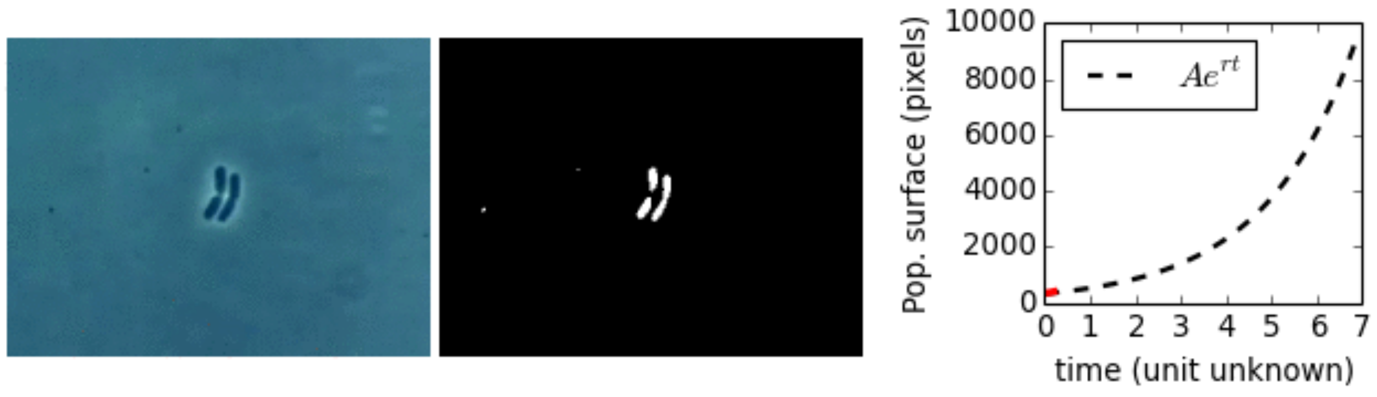
The goal for this project is to be able to take an image (or a video) and count the objects and their sizes in the image (or frames of the video). You will also display useful data about what you see.

You will have to learn from several examples. In our text, there are a lot of image processing examples

- http://www.scipy-lectures.org/intro/summary-exercises/image-processing.html#proposed-solution
- http://www.scipy-lectures.org/packages/scikit-image/index.html

One great example is counting change. See here for a detailed example : http://opensciencecafe.org/2016/01/counting-change-image-analysis-python/

For videos, you can also keep track of changes over time. Such as number of objects, average size, or total area.



You can find the example for the above GIF here : https://gist.github.com/Zulko/633c1b0807b37aa52d9c

and the bacteria video at : https://www.youtube.com/watch?v=gEwzDydciWc

Programming Tasks

You will make a graphical interface for your program which will display the image (or frame of video) and also show information from your analysis. Your goals will be as follows :

- have a way to ask the user for the file location, see `tkinter.filedialog`
- have an analyze button to process the image or video
- display the image or video **frame** in one part of your interface
- display **object count**, **average size**, and **total object area**
- display a graph with information in another part of your interface
  - you should have several types of graphs available.
  - choose at least 3 types from the suggestions below
    - histogram of sizes of objects
    - image of labeled data
    - for videos : object count vs time
    - for videos : average object size vs time