UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl Computergrafik
Dennis Bukenberger (dennis.bukenberger@uni-tuebingen.de)
Zabiha Kehribar (zabiha.kehribar@student.uni-tuebingen.de)

14. December 2020

Graphische Datenverarbeitung
Assignment 6

Submission deadline for the exercises: 21. December 2020 6.00 am

Source Code Solutions

- Upload **only** the source code files listed in the description in Ilias.
- Upload them one by one and **don't** zip them.

The source code must run on `cgpool120[0-7].informatik.uni-tuebingen.de` by extracting your submitted `.tar.gz` file to a certain folder and running `scons`. You can log onto this machine via ssh by using your WSI account. From outside the WSI network, you may have to use a ssh gateway, e.g. `cgcontact.informatik.uni-tuebingen.de`
Attention: Do not use `cgcontact` for working, but only for ssh-ing to the `cgpool` machines.

The framework you get for completion already compiles and runs as requested above and you only have to modify source and header files - no files have to be created.

Written Solutions

Written solutions have to be submitted digitally as one PDF file via Ilias.

6.1 Duality of Multiplication and Convolution (written, 20 Points)

The convolution of a function $f(t)$ with a second function $g(t)$ is defined as:

$$(f \otimes g)(t) = \int_{-\infty}^{+\infty} f(\tau) \cdot g(t-\tau) d\tau$$

The multiplication of two functions is defined as the point-wise multiplication:

$$(f \cdot g)(t) = f(t) \cdot g(t)$$

The transformation of a signal $f(x)$ to Fourier space is given by:

$$F(k) = \int_{-\infty}^{+\infty} f(x) \cdot e^{-2\pi i k x} dx$$

We call $\mathcal{F}$ the operator mapping f to Fourier space: $\mathcal{F}f = F$. Show that convolving in signal space is the same as multiplication in Fourier space:

$$\mathcal{F}[f \otimes g] = \mathcal{F}[f] \cdot \mathcal{F}[g]$$

6.2 Fourier Transformation (written, 20 Points)

Show that the Fourier transformation of the box function $B(x)$ is a *sinc* type function. The sinc function is defined as $sinc(x) = \frac{sin(\pi x)}{\pi x}$ and a definition of the Fourier transform can be found in exercise 6.1.

$$B(x) = \begin{cases} 0 & \texttt{for } x \leq -1 \\ 1 & \texttt{for } -1 < x < 1 \\ 0 & \texttt{for } 1 \leq x \end{cases}$$

6.3 2D Discrete Fourier Transformation (60 Points)

Fourier transformations can be used to transform periodical signals to the frequency domain where the frequency distribution of the signal can be seen easily.
The Discrete Fourier Transformation (DFT) can be used to transform images (which are discretized 2D functions) to Fourier space. DFTed images reveal the distributions and directions of primary frequencies in images and are therefore an important tool in image analysis.

The task of this exercise is to implement the DFT and the Inverse Discrete Fourier Transformation (IDFT) as parts of a pipeline which transforms an image from image space to Fourier space to polar coordinate (amplitude/phase) representation and back to the (hopefully) original image.

The supplied framework supports the `debug=1` and `openmp=1` scons flags. The compiled program has to be called with the filename of the image to be processed as first parameter, e.g. `./fourier image.ppm`. It will save several images showing intermediate results of the processing pipeline.
Don't choose images larger than 128x128 pixels since the naive DFT algorithm with its $O(n^4)$ running time complexity is really slow!

a) Implement the 2D DFT. It is given by

$$X_{k_1,k_2} = \frac{1}{\sqrt{N_1 * N_2}} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cdot e^{-i2\pi \frac{k_1}{N_1} n_1} \cdot e^{-i2\pi \frac{k_2}{N_2} n_2}. \quad (1)$$

X are complex Fourier coefficients, x are image pixel values and N_1 and N_2 are the image width and height. (n_1, n_2) is an image position while $k_1, k_2)$ indexes one Fourier coefficient.
You may want to `cout<<` some status info in the most outer loop of your code to see if it proceeds at all.

b) Implement the inverse 2D DFT. It is given by

$$x_{n_1,n_2} = \frac{1}{\sqrt{N_1 * N_2}} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X_{k_1,k_2} \cdot e^{i2\pi \frac{k_1}{N_1} n_1} \cdot e^{i2\pi \frac{k_2}{N_2} n_2}. \quad (2)$$

c) Implement the conversion from Cartesian complex number representation to polar coordinate (amplitude and phase) representation. This step is needed, since only the amplitude information of the Fourier coefficients can be interpreted.

d) Implement the conversion from the polar coordinate representation of complex numbers back to Cartesian representation.

Hint: Have a look at the documentation (www.cplusplus.com) for the following C++ standard library components:

- Template class `complex` for complex numbers.
- 2-parameter arctangent function `atan2f(y, x)` which handles all cases appearing during the calculation of an inverse tangent properly.

6.4 Supersampling (Bonus, 40 Points)

A pixel actually corresponds to a square area. To calculate its value means to integrate over this area. It is not possible to solve these integrals analyticaly, so they have to be approximated numericaly. This is done by computing weighted means of samples of the integrand. The actual image is reconstructed by sampling the integrand on selected points.
The placement of the sample points grealty influences the result. The approximation error may result in random noise or in aliasing artifacts in the image.

For this exercise, a slightly modified framework from assignment 2 is supplied. It presents upscaled versions of the rendered images and the sampling pattern that was used. Your can use the keys 1-7 to switch between different sampling strategies. This modified framework version doesn't update the image continuously anymore but re-renders the image only if any key from 1-7 is pressed.

The task in this exercise is to implement the different supersampling strategies.
Use four samples per pixel always!

a) **Key 2:** Regular grid sampling

b) **Key 3:** Rotated grid sampling
Choose the samples according to en.wikipedia.org/wiki/Supersampling#Rotated_grid.

c) **Key 4:** Random sampling

d) **Key 5:** Poisson disk sampling
Use `diskRadius=0.25f` (variable already set) as disk radius!

e) **Key 6:** Jittered sampling
Jitter the regular grid sampling positions by up to 0.2 Pixels.

f) **Key 7:** Stratified sampling

Key 1 triggers rendering with one single, centered sample per pixel.

As in previous assignments, you may use the preinitialized `mtrand` object to obtain random numbers.