Προγραμματισμός Υπολογιστών με C++

2η Εργασία – Ακαδημαϊκό Έτος 2016-17

Ημερομηνία Παράδοσης Εργασίας: **7 Ιανουαρίου 2018**, ομάδες 1-2 ατόμων

1. Εκφώνηση

Χρησιμοποιώντας ως βάση τον κώδικα που αναπτύξατε στην πρώτη εργασία, υλοποιείστε τα ακόλουθα:

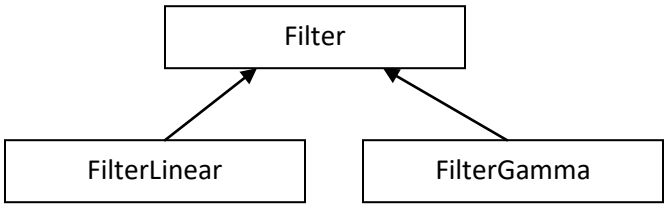
1. Μεταφέρετε τη λειτουργικότητα και υλοποίηση της κλάσης Image σε μια templated κλάση math::Array<T> (που ανήκει επομένως στο namespace math)και προσαρμόστε την κατάλληλα ώστε όπου πριν είχατε τον τύπο Color τώρα να χρησιμοποιείται ο άγνωστος τύπος T:
    - Αλλάξτε στη νέα κλάση Array την εσωτερική αναπαράσταση των δεδομένων (buffer) ώστε αντί για T * buffer να είναι std::vector<T> buffer.
    - Αλλάξτε στη νέα κλάση όσα ονόματα χρησιμοποιούσαν το συστατικό «Pixel» ώστε στην κλάση Array να είναι πιο αφηρημένα.
    - Καταργήστε τις μεθόδους load και save από τη νέα κλάση Array.
    - Προσθέστε έναν νέο τελεστή () έτσι ώστε αν έχετε ένα στιγμιότυπο της Array arr, η arr(i,j) να επιστρέφει αναφορά στο στοιχείο τύπου T στη γραμμή j και στήλη i.

2. Δηλώστε την κλάση Image ως απόγονο της κλάσης Array<T>, με τον τύπο T να είναι math::Vec3<float> [1]. Η κλάση Vec3 ορίζεται στο αρχείο Vec3.h. Η κλάση Image έχει, επιπλέον των μεθόδων της Array<T>, τις μεθόδους load() και save() που είχε και στην 1η εργασία. Οι δύο αυτές μέθοδοι εξακολουθούν να πρέπει να χρησιμοποιούν τις συναρτήσεις ReadPPM και WritePPM της βιβλιοθήκης ppm.lib.

3. Δηλώστε και υλοποιήστε μια βασική κλάση Filter της οποίας ο ρόλος είναι να υλοποιεί ένα φίλτρο επεξεργασίας εικόνας. Η μόνη μέθοδος που διαθέτει η κλάση Filter είναι η pure virtual μέθοδος:
   `Image operator << (const Image & image)`
   Η μέθοδος αυτή επενεργεί πάνω σε μια εικόνα (βλ. παρακάτω) και να την αλλοιώνει, παράγοντας μια νέα και θα πρέπει να μπορεί να κληθεί πολυμορφικά για τους απογόνους της κλάσης Filter. Οι απόγονοι αυτής της κλάσης θα υλοποιούν διαφορετικά ο κάθε ένας αυτή τη μέθοδο, ανάλογα με το είδος του φίλτρου. Η κλάση θα πρέπει να διαθέτει επίσης τους κατάλληλους constructors (default, copy).

4. Δηλώστε και υλοποιήστε δύο εξειδικευμένα φίλτρα ως απογόνους της βασικής κλάσης Filter, σύμφωνα με το παρακάτω διάγραμμα κληρονομικότητας:

[1] Για ευκολία, μπορείτε να δημιουργήσετε ένα ψευδώνυμο Color για τον τύπο math::Vec3<float> και να χρησιμοποιείτε αυτό στη δήλωση και υλοποίηση της Image και της Filter.



Η κάθε μία υλοποιεί με διαφορετικό τρόπο τον τελεστή << που κληρονομεί. Συγκεκριμένα, η FilterLinear εφαρμόζει σε κάθε Pixel $\mathbf{p}(x, y)$ της εικόνας εισόδου (δεξιός τελεστέος) τον τύπο:

$$\mathbf{p}'(x, y) = \mathbf{a} \cdot \mathbf{p}(x, y) + \mathbf{c}$$

Όπου $\mathbf{p}'(x, y)$ το εικονοστοιχείο στη συντεταγμένη (x,y) της εικόνας που δημιουργεί και επιστρέφει ο τελεστής και **a**, **c** δύο σταθερές τιμές math::Vec3<float>. Για παράδειγμα, αν $\mathbf{a} = (-1, -1, -1)$ και $\mathbf{c} = (1,1,1)$ το φίλτρο θα πρέπει να παράγει το αρνητικό της πρωτότυπης εικόνας, όπως και στην πρώτη εργασία. Σημείωση: όλα τα αποτελέσματα της παραπάνω πράξης θα πρέπει να τα περιορίζετε στο διάστημα [0,1] για κάθε κανάλι χρώματος.
Ο τελεστής << της FilterGamma υλοποιεί την πράξη:

$$\mathbf{p}'(x, y) = \mathbf{p}(x, y)^{\gamma}$$

όπου κάθε κανάλι χρώματος ενός εικονοστοιχείου υψώνεται στον εκθέτη *γ*. Τυπικές τιμές για το *γ* μεταξύ του 0.5 και του 2.0.
Οι κλάσεις FilterLinear και FilterGamma πρέπει να διαθέτουν επιπρόσθετα πεδία από τη βασική κλάση για να φυλάνε τις παραμέτρους τους, καθώς και τους αντίστοιχους κατασκευαστές.

5. Υλοποιήστε μια κύρια εφαρμογή η οποία θα παράγει ένα εκτελέσιμο με όνομα filter.exe για τη Release έκδοση και filterd.exe για τη Debug έκδοση. Για τη δημιουργία του εκτελέσιμου **θα πρέπει να συνδέσετε τη βιβλιοθήκη ppm.lib** (την αντίστοιχη Debug ή Release έκδοσή της) που φτιάξατε στην πρώτη εργασία. Αν δεν την έχετε φτιάξει ήδη, θα πρέπει να την υλοποιήσετε στα πλαίσια της δεύτερης εργασίας.
   Η εφαρμογή, αφού φορτώσει μια εικόνα ppm που προσδιορίζουμε από την κονσόλα, θα εφαρμόζει πάνω σε αυτή μια σειρά από φίλτρα επεξεργασίας εικόνας των παραπάνω 2 τύπων. Ο τύπος, ο αριθμός και η σειρά των φίλτρων εικόνας που θα επενεργούν πάνω στην εικόνα προσδιορίζονται κι αυτά από τα ορίσματα της εφαρμογής. Η τροποποιημένη εικόνα θα πρέπει να αποθηκεύεται σε αρχείο ppm κι αυτή, αφού πρώτα αλλαχθεί το όνομα αρχείου προσθέτοντας το πρόθεμα “filtered_” στην αρχή του ονόματος της εικόνας εισόδου.
   Ο τρόπος εκτέλεσης του προγράμματος ακολουθεί το παρακάτω μοτίβο:

   filter -f [φίλτρο 1] [παράμετροι] … -f [φιλτρο k] [παράμετροι] [όνομα αρχείου εικόνας]

Μετά τον προσδιοριστή –f ακολουθεί το όνομα του φίλτρου, που μπορεί να είναι είτε linear είτε gamma και αμέσως μετά μια σειρά από ορίσματα που αντιστοιχούν στις εξειδικευμένες παραμέτρους του κάθε φίλτρου. Το φίλτρο linear θα πρέπει να ακολουθείται από 6 αριθμούς float που αντιστοιχούν στις συντεταγμένες των και **a**, **c** διανυσμάτων. Το φίλτρο gamma ακολουθείται από 1 παράμετρο που αντιστοιχεί στον εκθέτη *γ*. Συνοπτικά, η παραμετροποίηση φαίνεται στον ακόλουθο πίνακα:

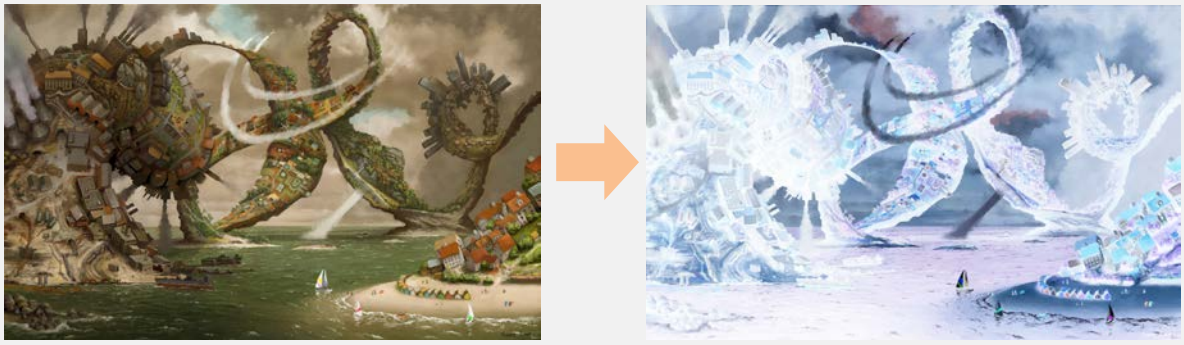
Κλάση που υλοποιεί το φίλτρο	Προσδιοριστής φίλτρου	Παράμετροι
FilterLinear	linear	aR aG aB cR cG cB
FilterGamma	gamma	γ

Το αρχείο εικόνας εισόδου δίνεται πάντα τελευταίο. Αν δε δοθεί φίλτρο, η εφαρμογή πρέπει να τερματίζει χωρίς να κάνει τίποτα. Αν δεν προσδιοριστεί αρχείο ή δε δοθεί το όνομα κάποιου φίλτρου σωστά (ή καθόλου), η εφαρμογή πρέπει να βγάζει σχετικό μήνυμα λάθους και να τερματίζει.

Ο αριθμός των φίλτρων και το είδος τους προσδιορίζεται κατά την εκτέλεση με βάση τις παραμέτρους που παίρνει η εφαρμογή από τη γραμμή εντολών. Επίσης, ένα φίλτρο ενδέχεται να εφαρμόζεται πολλαπλές φορές και με διαφορετικές παραμέτρους κάθε φορά. Η σειρά εφαρμογής των φίλτρων είναι από αριστερά προς δεξιά. Παραδείγματα:

```
> filter -f gamma 2.0 -f linear -1 -1 -1 1 1 1 image01.ppm
```

Εφαρμόζει πρώτα ένα FilterGamma φίλτρο με γ=2, και στη συνέχεια αντιστρέφει την εικόνα (αρνητικό).

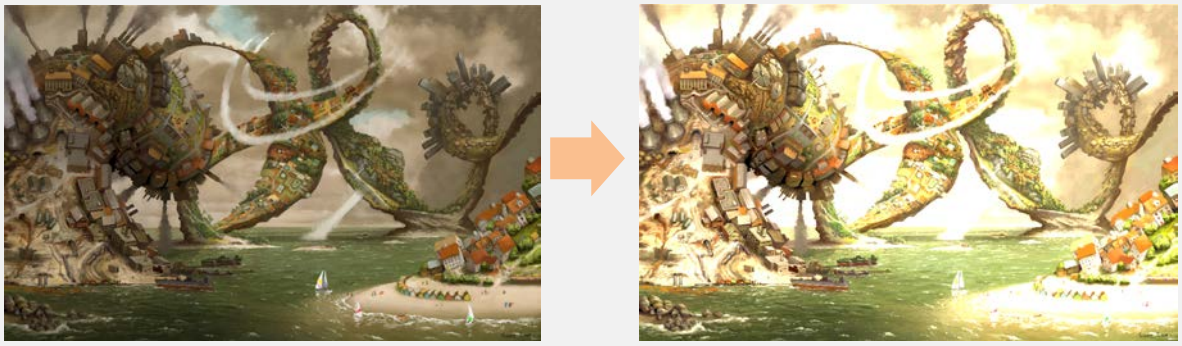


```
> filter -f gamma 2.0 -f gamma 0.5 image01.ppm
```

Εφαρμόζει 2 διαδοχικά φίλτρα FilterGamma τα οποία αλληλοαναιρούνται.

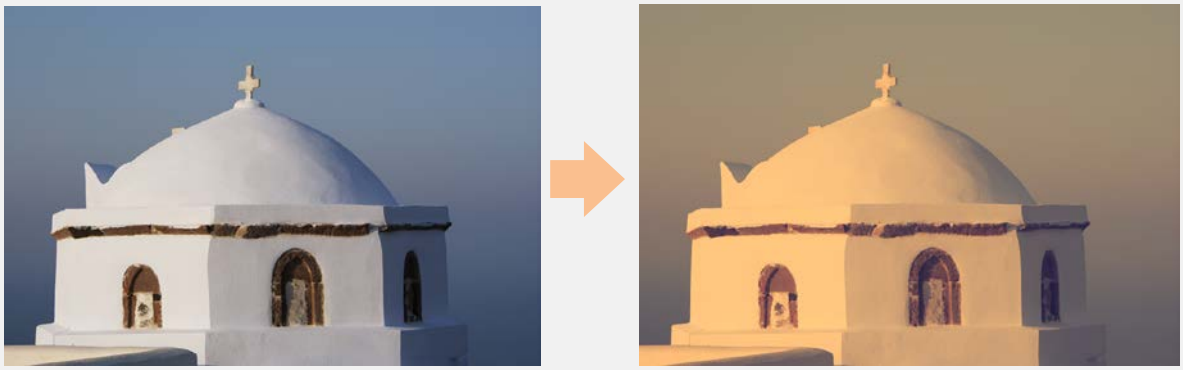
```
> filter -f linear 2 2 2 0 0 0 image01.ppm
```

Διπλασιάζει τη φωτεινότητα της εικόνας. Τιμές πάνω από την τιμή κορεσμού (1.0) αποκόπτονται στο 1.0 (βλ. παραπάνω οδηγίες για την υλοποίηση του FilterLinear << operator).



```
> filter –f gamma 0.7 -f linear 1 0.8 0.3 0.1 0.1 0.3 –f gamma 1.2 image02.ppm
```

Εφαρμόζει 3 φίλτρα που αλλοιώνουν μη ομοιόμορφα τα χρώματα της εικόνας.



6. (**bonus +0.3**) Δηλώστε και υλοποιήστε μια επιπρόσθετη κλάση απόγονο της Filter, την FilterBlur που υλοποιεί τον τελεστή << έτσι ώστε για κάθε pixel (i, j) της εικόνας προορισμού, διαβάζει μια περιοχή NXN pixels της εικόνας εισόδου με κέντρο το (i, j) και θέτει ως χρώμα του pixel (i, j) το βεβαρυμμένο άθροισμα των (το πολύ NXN) έγκυρων pixels της NXN γειτονιάς με βάρος $1/N^2$. Ο υπολογισμός μπορεί εύκολα να προκύψει από τον τύπο:

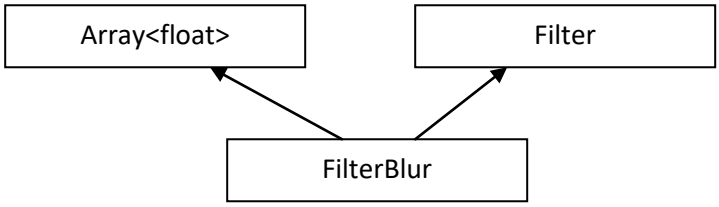
$$\mathbf{p}'(i,j) = \sum_{m=-N/2}^{N/2} \sum_{n=-N/2}^{N/2} \mathbf{p}(i+m, j+n) f(m+\frac{N}{2}, n+\frac{N}{2})$$

Το άθροισμα «τρέχει» για όλα τα εικονοστοιχεία $\mathbf{p}(i+m, j+n)$, για τα οποία $0 \leq i+m < w$, $0 \leq j+n < h$ (εντός ορίων), όπου w, h το πλάτος και ύψος της εικόνας. Πολλαπλασιάζει δε το εικονοστοιχείο με το αντίστοιχο σημείο του πίνακα του φίλτρου *f*.

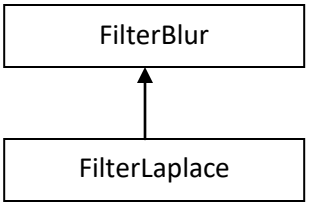
Η αντίστοιχη με τα άλλα φίλτρα παραμετροποίηση από τη γραμμή εντολών είναι:

Κλάση που υλοποιεί το φίλτρο	Προσδιοριστής φίλτρου	Παράμετροι
FilterBlur	blur	N

Για την υλοποίηση του φίλτρου αυτού, η FilterBlur κληρονομεί τόσο από την Filter, όσο και από την Array<float> η οποία και προσφέρει την εσωτερική αναπαράσταση των τιμών του πίνακα *f.* Στο συγκεκριμένο φίλτρο, όλα τα κελιά έχουν την τιμή $1/N^2$ και προσδίδονται στον πίνακα κατά την αρχικοποίηση αντικειμένου τύπου FilterBlur (για συγκεκριμένη παράμετρο κατασκευής N).



7. (**bonus +0.3**) Δηλώστε και υλοποιήστε μια επιπρόσθετη κλάση απόγονο της FilterBlur, την FilterLaplace.



Η FilterLaplace που υλοποιεί τον τελεστή << έτσι ώστε για κάθε pixel (i, j) της εικόνας προορισμού, διαβάζει μια περιοχή 3X3 pixels της εικόνας εισόδου με κέντρο το (i, j) και θέτει ως χρώμα του pixel (i, j) της εικόνας εξόδου το βεβαρυμμένο άθροισμα των (το πολύ 3X3) έγκυρων pixels της 3X3 γειτονιάς με βάρος $f(k, l)$, όπου το $f(k, l)$ προκύπτει από τον ακόλουθο πίνακα 3X3

που αντιστοιχεί κελί προς κελί με τη γειτονιά 3X3 των Pixels της εικόνας πάνω στην οποία εφαρμόζεται το φίλτρο:

$$f = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{p}'(i,j) = \mathbf{max}\left(\mathbf{0}, \mathbf{min}\left(\mathbf{1}, \sum_{m=-1}^{1}\sum_{n=-1}^{1} \mathbf{p}(i+m, j+n) f(m+1, n+1)\right)\right)$$

Για τις πράξεις min και max μπορείτε να χρησιμοποιήσετε τις μεθόδους clampToLowerBound(val) και clampToUpperBound(val) του Vec3.

Στη γραμμή εντολών, δηλώνουμε τη χρήση του φίλτρου Laplace με το λεκτικό laplace και καμία παράμετρο, π.χ.:

```
> filter -f laplace –f linear -2 -2 -2 1 1 1 image01.ppm
```

Βρίσκει τις ακμές της εικόνας, τις ενισχύει X2 και τις αντιστρέφει.

2. Γενικές οδηγίες

Για την υλοποίηση των προγραμμάτων, μπορείτε να χρησιμοποιήσετε όποιο περιβάλλον θέλετε αλλά ο κώδικας που θα παραδώσετε θα πρέπει να μπορεί να γίνει compile και link είτε με τον gcc σε περιβάλλον Windows, είτε με το Visual Studio 2015, πάλι προφανώς σε περιβάλλον Windows. Παραδίδετε έτοιμο Visual Studio project ή αντίστοιχα makefile (για τον gcc) και όχι μόνο τα .cpp και .h αρχεία και έχετε επιβεβαιώσει ότι γίνονται σωστά build. Πριν ανεβάσετε την εργασία σας, καθαρίστε τα παραγόμενα από το Visual Studio αρχεία (Build→Clean Solution και διαγραφή του αρχείου .sdf από το solution directory).

Μπορείτε να ελέγξετε το αποτέλεσμα με το λογισμικό προεπισκόπισης που σας δίνεται στο eclass ή με οποιοδήποτε άλλο πρόγραμμα που αναγνωρίζει το μορφώτυπο PPM.

Προσοχή: Σε οποιαδήποτε περίπτωση διαπιστωθεί **αντιγραφή** κώδικα ή αδυναμία του εξεταζομένου φοιτητή να **εξηγήσει την υλοποίησή του**, η εργασία **μηδενίζεται** αυτομάτως. Επίσης, το εκτελέσιμο της εργασίας **πρέπει να δουλεύει** για να βαθμολογηθεί η τελευταία.