Déroulé du cours :

8h00-9h15 9h30-10h45 11h00-12h15 13h30-14h45 15h00-16h15 16h30-17h45

Penser à proposer de terminer à midi et reprendre à 13h15 pour compenser.

Attention : certains fichiers sont à épurer avant distribution aux étudiants car une proposition de solution est déjà présente dans le fichier (penser à la supprimer avant affichage puis annuler la modif pour faire revenir le code ensuite) : 06

Jour 1 :

- se présenter : ESEO, développeur soft bas niveau et architecte logiciel. Machines du siècle dernier peu puissantes → besoin optimisation (exemple du JumboScan, de la gestion de flux en audio). Donner activités présentes en soulignant le 60-70 % de presta → discours pratique
- présenter les enjeux de la matière : économie d'énergie, de la batterie, temps de réaction d'un système, etc

- Imagerie numérique :
- présenter le principe d'une représentation matricielle d'une image
- distribuer juste 00_base.c, voir à ce que tout le monde puisse l'exécuter
- insister sur l'importance de la précision sur les types (exemple : bitmap déclarée en uint16 alors que uint8 suffit => empreinte mémoire doublée pour rien, parcours de la mémoire aussi)
- voir pourquoi les valeurs ne sont pas forcément à 0, expliquer la faille potentielle qu'il y a derrière
- passer à 01_base_initialisee.c expliquer le principe de la boucle while pour parcourir un buffer avec tableau : Combien de fois / point de départ / action / passage à la prochaine itération
- construire collectivement l'affichage sur ce principe
- proposer de faire un peu mieux avec 01_ter (on troque le if pour un while, intérêt à pondérer)
- TD : faire un tracé d'une ligne horizontale, puis verticale et enfin oblique
- faire changer le rapport de forme de l'image à ceux qui ont terminé, parler des invariants de boucle et veiller à ce que le test se fasse sur l'initialisation de counter

- Exploitation des fichiers :
- Présenter le code 05_mires_raw_gris.c : expliquer le principe de poser le buffer dans un fichier, ouvrir le résultat avec TheGimp, montrer l'aspect décodage des données en faisant varier les paramètres de l'importation
- Introduire la notion de projet dans les IDEs : fichier qui contient la configuration dans l'éditeur mais surtout l'ensemble des fichiers et règles données au compilateur.

Dans CodeBlocks :
- créer un workspace, puis un projet Console et enfin ajouter les fichiers au projet – montrer l'exemple avec le 06 des fichiers qui sont dans common, on les ajoute tous et on les voit bien. Ceci dit ça ne compile pas : les fichiers de header ne sont pas trouvés car ils ne sont pas dans les chemins de compilation. Clic droit sur le projet → Build Options, onglet Search ditectories, sélectionner le projet en haut à gauche et ajoute le chemin vers common et là c'est bon.
- pour donner un chemin à l'exécution : menu Project → set program's arguments

Dans Xcode :
- ajout des fichiers par drag'n'drop, décocher la copie
- arguments : choisir la bonne target et faire Edit Scheme dans le menu des targets

- Présenter le code 06_BMP_Read_Write.c – attention il nécessite ce qu'il y a dans common sans time_measure.c
- Dans ce code proposer de faire des lignes pour montrer qu'on peut bien y modifier l'image
- Toujours dans ce code demander de faire un miroir suivant un axe vertical (commencer par le tableau puis le code) Horizontal, puis
- Puis demander de faire une transformation en négatif d'une image
- Expliquer l'invariant algorithmique et le principe de la LUT
- Demander de faire l'implémentation

Jour 2 :

- Mise en place de la LUT de façon collective
- Expliquer le principe de correction lumière/contraste (capteur non linéaire en fonction du flux lumineux), proposer l'approximation avec une fonction affine, demander d'implémenter puis optimiser. Problèmes standards : boucle infinie pour la LUT à cause d'une variable de boucle en uint8_t, type double pas ou mal utilisé, oubli du test sur la valeur
- Montrer puis faire circuler le code 06bis BMP Lumiere contraste.c, expliquer le principe, insister sur le fait que la LUT doit être dans la heap et pas dans la pile et que dans ce cas la LUT a aussi remplacé du code

- Grosse partie descendante sur la mesure du temps d'exécution :
FAIRE DES CLASSES POUR LA MESURE ET BLINDER LE PROJET CODE BLOCKS
- 07 : Time measure.c puis .h puis 07_Mesure...
- montrer le principe de mesure utilisé, les pointeurs de fonction et la souplesse que ça permet
- expliquer le cahier des charges du code
- demander les améliorations possibles dans le basic_processing
- parler de l'accès dans le sens naturel de la mémoire, des caches et la façon d'en tirer partie
- montrer que c'est la façon de faire la chose qu'on va surtout modifier
- coup du gradient recopié plutôt que calculé : idem LUT

- 08 : lumière contraste
- gain faible entre méthode 1 et méthode 2 car le double if prend beaucoup de temps et est le goulot d'étranglement
- gros gain sur la méthode 3 (la dupliquer pour faire variante avec pointeur de LUT local) : si faible gain avec pointeur LUT local c'est que le compilo avait vu l'optimisation à faire

→ gros TD pour mise en œuvre – évaluation de l'année précédente par exemple

Jour 3 :

Mise en commun / correction du TD

09 (pas obligatoire)

Pot pourri : (voir notes)

Points à voir dans bmp.c : localiser le code le read, expliquer que c'est à cause de certaines architectures qu'il y a cette obligation d'avoir des longueur de ligne multiples de 4
- lecture jusqu'à plus de données à prendre
- traitement par paquets puis reliquat final

- arrondi au supérieur multiple de 4

Partie objets/mémoire managée

Ficelles classiques :
- méthodes utilitaires à mettre en méthodes de classe pour économiser le contexte
- invariants de boucle
- factorisation dans les calculs

Gestion de la mémoire
- réutilisation des instances, quitte à rajouter une méthode d'initialisation des instances
- StringBuilder vs String ; instance immuable vs muable
- expliquer la gestion de la mémoire automatique
→ allocations rapides
→ collectes lentes voire gelant l'interface ou les sockets, etc. Contre ça : bien avoir des temporaires vraiment temporaires et ne modifier l'état des plus persistantes que si on est absolument sûr de la nécessité, un if est moins cher qu'une racine supplémentaire lors de la collecte. Expliquer la gestion des pools d'objet
+ Libération explicite des ressources

Design :
- Bonne utilisation de l'héritage et du ploymorphisme : exemple 2 (Forme) puis 3 (GUI)
- Bonne utilisation de l'encapsulation pour éviter de se poser constamment des questions sur la validité de l'état d'une instance
- Aller encore plus loin avec lors des changement de configuration → changement de la nature concrète d'instances à qui on délègue le travail effectif : exemple du soft de dessin et des tampons

Evaluation 15h30 (pause 15h10).