POLYTECH
NICE-SOPHIA

`

TP « Derived Class »

Objectifs de ce TP :

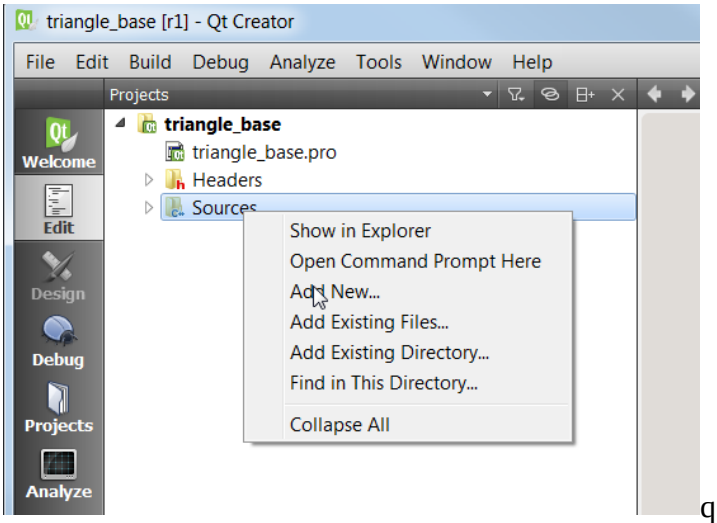
- Codage d'une classe dérivée
- Ajout de modularité dans votre code avec plusieurs fichiers et gestion des noms hiérarchiques.
- Découverte d'une fonctionnalité du C++ : les foncteurs (qui ne sera pas couverte en cours faute de temps).

1 - Une Classe Dérivée

1.1 Intégration avec QT

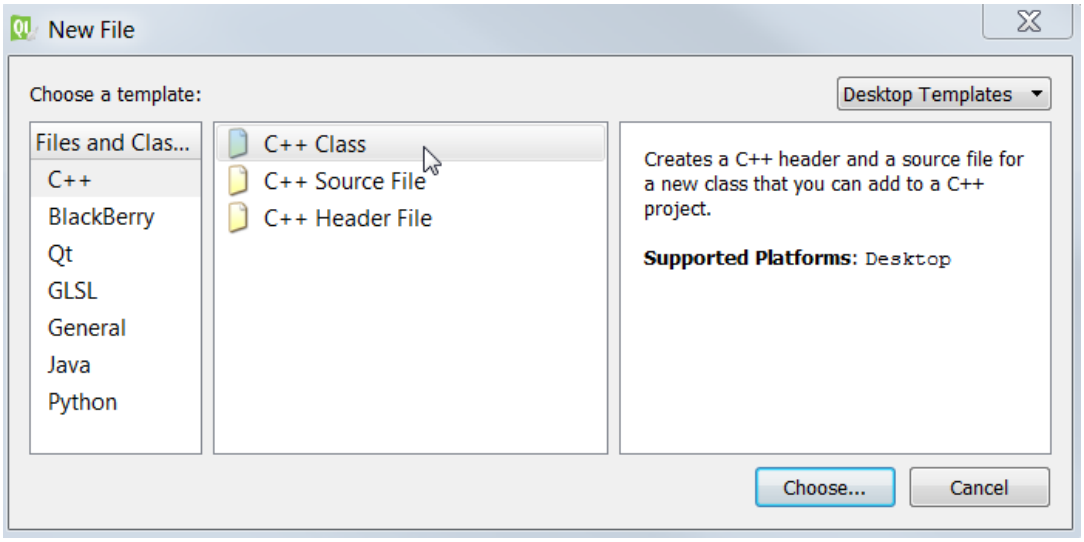
Prendre comme point de départ le projet triangle-base.
Il vous faut ajouter une classe `SplineImage` dérivée de `QImage`.
Suivre la manip suivante pas à pas :

1) En cliquant bouton droit sur « Sources », un menu contextuel apparait, il faut sélectionner alors « Add New.. »

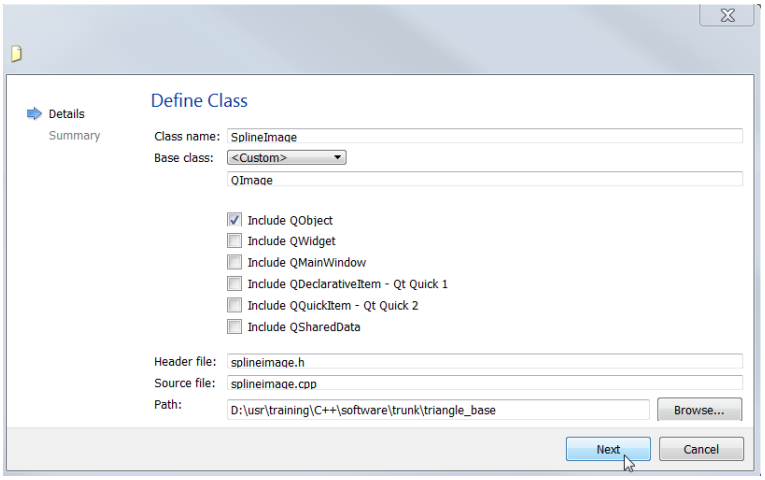


q

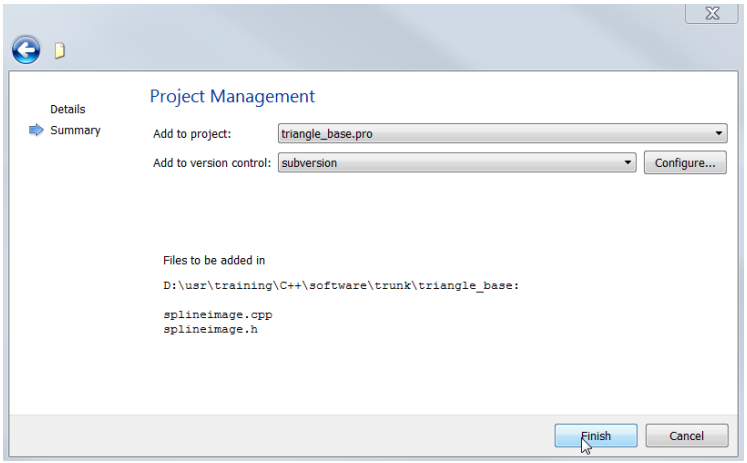
2) Sélectionner « C++ Class » puis cliquer sur « Choose… »



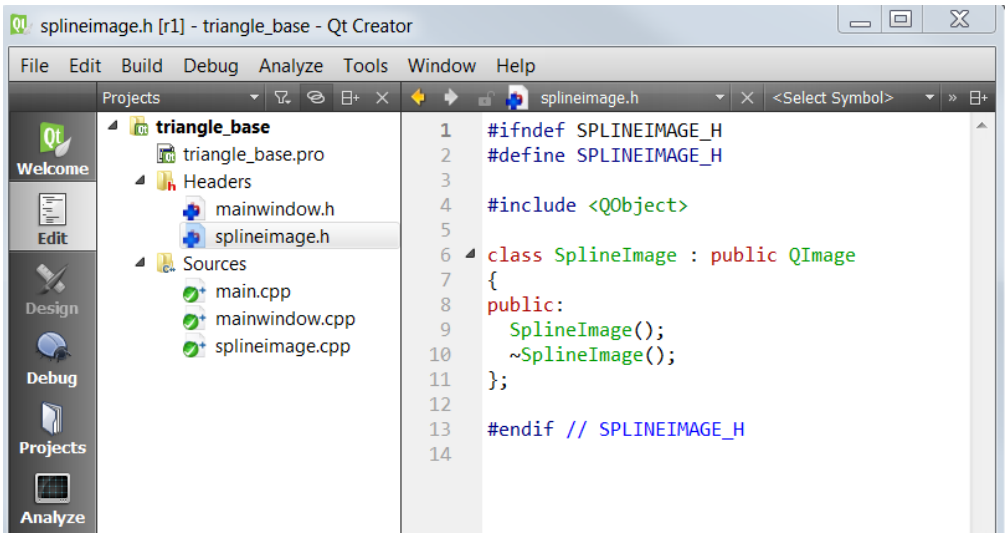
3) Remplir les champs « Class name » et « Base class » du panneau suivant et cliquez sur « Next ».



4) Le panneau final apparait enfin, cliquez sur « Finish »

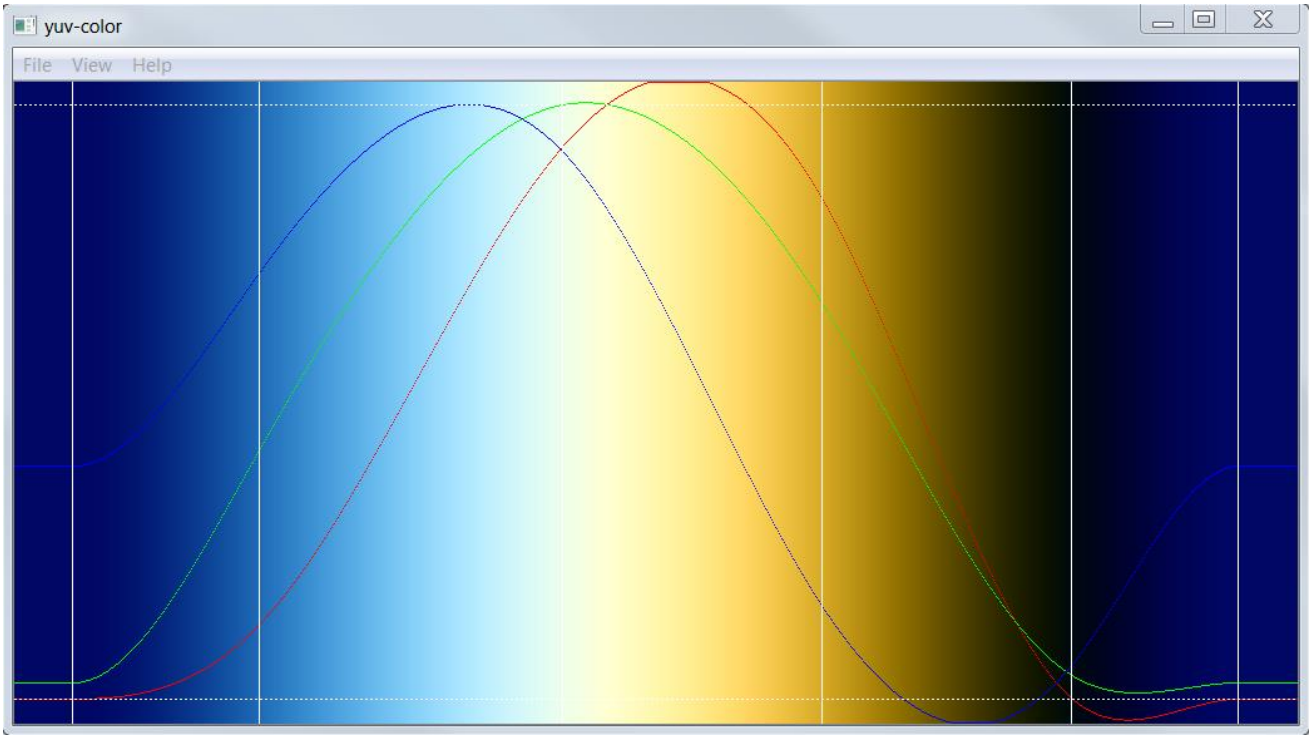


5) QT a créé 2 fichiers : `splineimage.h` et `splineimage.cpp` avec un canvas de la classe `SplineImage`.



1.2 La class SplineImage

La classe `SplineImage` est une classe dérivée de `QImage`, de dimension 1024x512 qui devra produire une image identique à l'image ci-dessous.



On remarquera les 3 courbes (rouge, verte et bleue) nommées $y_r(x), y_g(x), y_b(x)$ sont des splines construites avec les formules de la section précédente et définies par les points suivants

```
vector<double> xs{   0.,   0.16,   0.42,   0.6425,  0.8575};

vector<double> yr{   0.,  32.  , 237.  , 215.    ,  0.    };
vector<double> yg{   7., 107.  , 255.  , 170.    , 10.    };
vector<double> yb{ 100., 183.  , 235.  ,  40.    , 15.    };
```

Les courbes seront tracées pour les valeurs de x telles que $-0.05 \leq x \leq 1.05$.
Vous devrez convertir x vers/depuis une position pixel horizontal comprises entre 0 et 1023.
Vous devrez convertir les valeurs $y_r(x)$ dans l'intervalle $-10 \leq y_r(x) \leq 265$ vers/depuis une position pixel vertical comprises entre 0 et 511, idem pour $y_g(x)$ et $y_b(x)$. Il faudra écrêter si nécessaire pour ne jamais écrire des pixels, avec la fonction `QImage::setPixel`, à l'extérieur de l'image. Voir courbe rouge en haut et courbe bleue en bas.

Les axes verticaux sont donnés par le vecteur `xs` ainsi que la coordonnée $x = 1$ et seront tracés en blanc.

La couleur de l'image dépend uniquement de x dans l'intervalle $-0.05 < x < 1.05$ et cette couleur est donnée (en RGB) par $y_r(x), y_g(x), y_b(x)$ avec une fonction d'écrêtage simple pour s'assurer que chaque composante est une valeur entière entre 0 et 255.
Les axes horizontaux sont donnés par les coordonnées $y = 0$ et $y = 255$ seront tracés en pointillé blanc.

Ne pas hésiter à reprendre une partie du code de la solution du triangle pour vous simplifier le travail.

Voir le TP « spline » pour référence sur les splines. Vous pouvez utiliser soit l'interpolation linéaire soit l'interpolation spline.

Si vous utilisez les splines, rappelez-vous que QT génère des fichiers de type `Makefile` à partir du fichier projet (`.pro`). Vous devez donc vous assurer que les fichiers Eigen soient « visibles » dans votre `.pro`. A vous de trouver comment avec ce lien http://doc.qt.io/qt-5/qmake-variable-reference.html

Si vous utilisez les splines, vous devez ajouter, la ligne suivante dans votre fichier .pro, avant de lancer la compilation

```
QMAKE_CXXFLAGS  += -std=c++14 -mavx -Wno-attributes
```

1.2.1 Livrables pour la section 2

Aucun, voir section 3

POLYTECH NICE-SOPHIA

2 - Vers un Code Pro (deuxième partie)

2.1 Les Foncteurs

A Lire : le chapitre sur les foncteurs en page numérotée 550 du document « programmez-avec-le-langage-c++.pdf ».
Pour info : foncteurs en anglais : *function objects* ou *functors*

Vous avez sans doute écrit plusieurs fonctions d'écrêtage comme ci-dessous avec un copier-coller des fonctions de fichier en fichier

```
static int clamp_to_rgb(double v) {
  return (v <= 0.0) ? 0 : ((v >= 255.0) ? 255 : static_cast<int>(v));
}
static int clamp3(int v, int v_min, int v_max) {
  return (v < v_min) ? 0 : ((v > v_max) ? v_max : v);
}
```

On souhaite donc mutualiser ces fonctions dans un seul fichier : `elec4_util.h`
Faire les copier-coller nécessaire.
Compiler, avez-vous warning ?
Expliquez le rôle du mot clef `static` ?

Pour remplacer ces fonctions et obtenir un code réutilisable on se propose de créer un « *functor* » avec template c'est-à-dire une classe comme ci-dessous

```
template<typename tpl_t>
class Clamp {
private:
  tpl_t min_;
  tpl_t max_;
public:
  int operator()( …à vous de completer cette classe…

};
```

Modifiez votre code pour utiliser votre foncteur, par exemple, je trouve dans mon code après modification :

```
ELEC4::Clamp<double> clamp_to_rgb(0., 255.);
QColor vertical_color(clamp_to_rgb(yr), clamp_to_rgb(yg), clamp_to_rgb(yb));
..
ELEC4::Clamp<int> clamp_to_height(0, height - 1);
setPixel(xp, clamp_to_height(yp_r), qRgb(255, 0, 0));
```

2.2 D'autres foncteurs

Il est aussi intéressant de construire un foncteur pour une interpolation linéaire, par exemple

```
double x_min = -0.05;
double x_max =  1.05;
ELEC4::LinearInterpolate h_interpolator(0.0, static_cast<double>(width - 1),
                                        x_min, x_max);

for (int xp = 0; xp < width; ++xp) {
  double  x = h_interpolator(xp);
  …
```

2.3 Livrables pour la section 2 & 3

Aucun, ce TP n'est pas noté

~~1) Une archive `.zip` avec l'ensemble de votre projet qui doit compiler sans erreur dans l'environnement QT et donne le résultat attendu.~~

~~2) Un document `.pdf` décrivant les points clé suivants~~

- ~~Les warning résiduels de `cpplint` (version 141) seront expliqués (uniquement les warning sur les fichiers que vous avez codés).~~
- ~~La classe `SplineImage` : constructeur, destructeur…~~
- ~~Passage des coordonnées pixel aux coordonnées x et y~~
- ~~Les fonctions d'écrêtage~~
- ~~Explication des warning vu en 3.1.3~~
- ~~Le rôle des instructions préprocesseurs~~
- ~~Les classes du fichier `elec4_util.h`~~
- ~~Le ou les foncteurs utilisés, les fonctions d'écrêtage utilisées.~~