Concours externe Inria 2014

Arrêté du 18 avril 2014

Poste « DI 3 : Chef de projet / expert en traitement d'images »

Accès au corps des ingénieurs de recherche

Epreuve du 11 juin 2014

Note sur 20 – Coefficient 3 – Durée 3 heures

La notation prendra en compte la qualité des réponses, mais aussi la rédaction, la présentation, le style et l'orthographe.

Veuillez respecter l'anonymat dans les réponses.

Ne pas omettre de noter votre numéro d'ordre sur les feuilles intercalaires.

MODULE A : 7 points au total

Exercice A1 : (1,5 points)

A1.1 : En imagerie, un pixel est parfois codé sur 32 bits, 3x8 bits servant à coder l'intensité des couleurs Rouge, Verte et Bleue. Lorsque les 8 bits restants sont utilisés, comment s'appelle cette valeur et quel est son rôle ?

A1.2 : Le codage de la couleur d'un pixel sous la forme de 3 composantes indiquant l'intensité en Rouge, Vert et Bleu est appelée codage RVB ; le codage TSL (ou HSL en anglais) est un autre modèle de représentation de la couleur. Expliquer ce qu'est ce second codage et pourquoi il a été introduit ?

A1.3 : Même question pour le codage YUV.

Exercice A2 : (1,5 points)

A2.1 : Qu'est-ce qu'une opération de compression ?

A2.2 : Quel est l'avantage principal de la compression sans perte ? L'inconvénient principal ?

A2.3 : Quelle(s) technique(s) de compression est (sont) utilisée(s) dans le format Joint Photographic Experts Group (JPEG) ?

A2.4 : La norme JPEG spécifie-t-elle le format de fichier ? L'algorithme de compression ? L'algorithme de décompression ?

A2.5 : Pourquoi lors de la compression JPEG, une transformation de l'espace RGB vers l'espace YUV est-elle utilisée ?

Exercice A3 : (4 points)

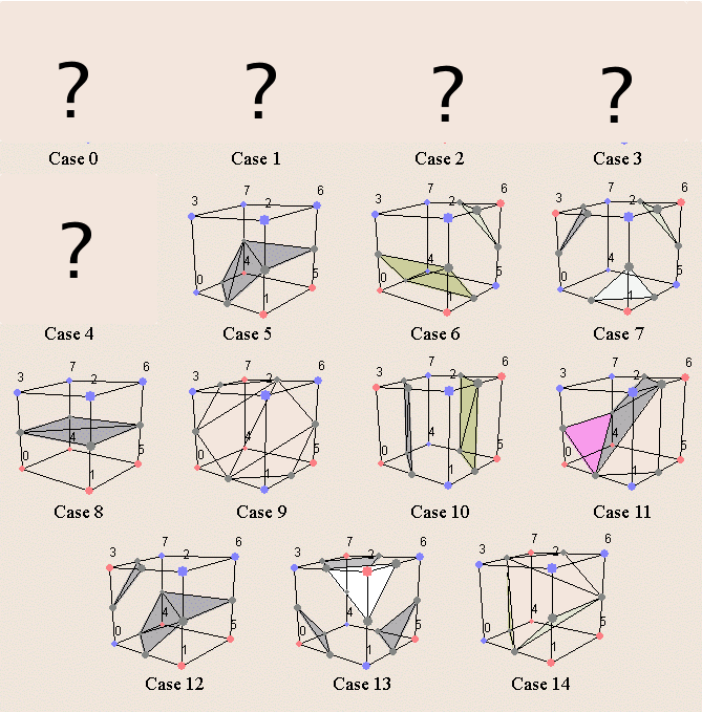
Soit un maillage structuré et uniforme 3D. Nous souhaitons calculer les différentes configurations que peut prendre l'iso-surface dans un élément de volume, selon la répartition des intersections de l'iso-surface sur les arrêtes de cet élément de volume.

A3.1 : L'élément de volume étant un cube, et sachant que chaque sommet de ce cube peut prendre 2 états (-1 = sous la surface, 1 = au-dessus de la surface), combien y a-t-il de configurations possibles ? Proposez une représentation informatique de ces cas.

A3.2 : Ces cas peuvent être généralisés, par rotations et symétries, en 15 cas. Pouvez-vous dessiner une représentation de la surface pour chacun des 5 cas où strictement moins de trois sommets sont sous la surface (donc ont la valeur -1) ? Nous attendons cinq cubes, avec les valeurs (-1 ou 1) indiquées sur les sommets, et la surface dessinée.

A3.3 : A chaque cube, correspond un cube complémentaire, avec les valeurs opposées des sommets. Comment change la surface suite à cette transformation ? Pourquoi ne pas considérer que ces cubes sont identiques ?

A3.4 : Nous avons maillé l'espace avec le maillage structuré et uniforme ; chaque sommet a reçu la valeur -1 ou 1 selon sa position par rapport à l'iso-surface que nous souhaitons représenter. Pour chaque cube, nous dessinons la surface en choisissant l'une des 15 configurations correspondantes :



Nous constatons alors des erreurs dans le modèle 3D obtenu, avec notamment des trous dans la surface. Pourquoi ? Que faut-il faire pour résoudre ce problème ?

A3.5 (bonus) : Connaissez-vous le nom de la méthode présentée dans cet exercice ?

MODULE B : 8 points au total

Exercice B1: (2 points)

Une des caractéristiques remarquables du code écrit dans les langages objets est, comparativement à d'autres langages, la faible utilisation d'instructions de type « aiguillage » telles que *switch* ou *case*. Considérer le code suivant (c'est une méthode dans une classe pour les oiseaux) :

```
double getSpeed() {
    switch (_type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException ("Unknown Bird type, this should be unreachable");
  }
```

B1.1 : Refaire (*Refactor)* le code ci-dessus dans un style « objets » en donnant les classes principales en pseudo code.

B1.2 : Pourquoi ces instructions "switch" pourraient-elles être considérées comme problématiques au cours de la vie d'un logiciel?

Exercice B2 : (2 points)

B2.1 : En 10 lignes maximum, décrire les principales différences entre une classe abstraite et une interface.

Exercice B3 : (2 points)

On vous demande de développer un démonstrateur avec une interface graphique pour une simulation numérique, simulation numérique pour laquelle le code est développé en Fortran90. Après étude des besoins de l'interface, vous optez pour une solution utilisant le langage Java.

B3.1 : Dans l'optique où on laisse le code Fortran en l'état, expliquez quelles sont les possibilités de faire cohabiter les 2 codes. Pouvez-vous citer un outil qui aidera dans ce cas ?

B3.2 : Quel est le nom usuel pour la mise en concordance de codes écrits dans des langages différents ?

Exercice B4 : (2 points)

B4.1 : Décrivez brièvement 2 scénarios ***différents*** pour mettre en œuvre les tests au cours des cycles de développement, en vous appuyant sur les 2 cas suivants :

- *release* annuel du code propriétaire (non diffusé)
- code disponible en open source avec des collaborateurs externes.

B4.2 : Donnez les avantages et inconvénients de chaque approche (scénario), en 10 lignes maximum.

MODULE C : 5 points au total

Exercice C.1 :

Attention : pour cet exercice, **vous devez utiliser les documents de l'annexe**. Les seuls documents à votre disposition sont ceux fournis dans l'annexe.
Attention de toujours respecter l'anonymat de votre copie

Le principal client de l'éditeur logiciel pour laquelle vous travaillez, souhaite développer une version mobile de son application la plus connue pour les plateformes iOS et Android.

Le développement mobile n'est pas chose courante dans votre entreprise ; les équipes sont plus à l'aise avec des langages classiques tels que C++ ou JavaScript.

Avant de commencer la mission pour le client, le chef de projet souhaite donc passer en revue un certain nombre d'environnements de développement pour mobile, parmi lesquels :

- les solutions natives iOS et Android
- les solutions basées sur HTML5
- le *framework* Qt Quick proposé par Qt.

Il confie à trois membres de l'équipe, dont vous, le soin d'examiner un de ces environnements.
Vous vous retrouvez alors en charge d'examiner la solution Qt Quick. Votre manager attend de vous trois que vous vous fassiez l'avocat de la solution que vous devez étudier. Son choix se portera sur celui qui aura apporté le plus d'éléments probants.

Dans cette optique, à partir des documents dans l'annexe, vous devez écrire **une synthèse convaincante en langue anglaise d'une demi page maximum** de la solution Qt Quick en exposant notamment :

- au moins deux points forts de l'approche ;
- au moins deux conséquences qu'un tel choix aura sur les équipes de développement et de *designers* ;
- les problèmes ou difficultés potentiels de l'approche.

DOCUMENT 1 : Extract from http://developer.ubuntu.com/apps/qml/

UBUNTU QML overview

1 - A language to write compelling UIs

QML, the Qt Meta-Object Language, is a programming language with similarities in syntax to JavaScript. Being a modern language designed with simplicity and extensibility in mind it has borrowed syntax and paradigms from ubiquitous web technologies, such as CSS and JavaScript. It is built upon and constitutes a core component of the cross-platform Qt framework.

In essence, QML coupled with the underlying Qt libraries is a high-level UI technology that enables developers to create animated, touch-enabled UIs and light-weight applications. It differs from traditional procedural languages such as C or Python in the sense that it is declarative at its core, that is, it is used to describe UI elements and their interaction. As such, traditionally complex visual motion transitions become a natural part of QML and are extremely easy to implement.

```
// Simple QML example
import QtQuick 2.0

Rectangle {
    width: 200 // this is a property
    height: 200
    color: "blue"

    Image {
        source: "pics/logo.png"
        anchors.centerIn: parent
    }
}
```

QML is also a scripting language, which does not require compilation or cross-compilation and can instantly be run on any device. This provides a very rapid development and testing workflow, where prototyping user interfaces in QML becomes a very simple and agile process. Despite it being a scripting language, QML can leverage the OpenGL support to provide near-game-performance with gourgeous movements and transitions.

2 - Components at the core

Components and their elements are the main building blocks of a QML application. Basic UI elements such as rectangles and circles can be used to create the basic UX of an application, but become more useful when they are grouped together and encapsulated into modular components to provide enhanced functionality.

QML element or components usually have various properties that help define them, which are declared with a similar syntax as CSS. As an example, a rectangle element would have width and height properties that would define its size.

3 - Deep JavaScript integration

One of the features that makes the language so powerful and flexible is the ability to embed JavaScript to implement the high-level user interface logic. Integrating JavaScript into a QML app can be done in a number of ways:
- Embedding JavaScript expressions as property values in QML elements
- Adding JavaScript functions in QML
- Using external JavaScript files and importing them into a QML app

This enables developers leverage their existing JavaScript knowledge they have of JavaScript to rapidly develop both user-interfaces and application logic.

4 - Extending QML with C++

While for most cases QML and JavaScript is all you need to write an app, there are cases where computing-intensive tasks are required (e.g. complex image manipulation, a physics engine) and you'll need to squeeze all available performance from the processor.

In these situations, QML lends itself to be extended with C++ extensions to perform the resource-intensive tasks in the backend, while still providing a clear separation between UI and application logic.

Since QML is built upon Qt, it inherits most of the functionality from the Qt framework, particularly the signals and slots mechanism and the meta-object system. Data created using C++ is directly accessible from QML, and QML objects are also accessible from C++ code.

DOCUMENT 2 : Extract from http://qt-project.org/wiki/Introduction_to_Qt_Quick

Introduction to Qt Quick

Today's consumers and enterprise users are tough to please. They grew up using slick UIs on their game consoles and seeing even fancier UIs at the movies. Specifications and feature lists alone are no longer selling points; they are means to an end. Visual impact and experience WOW are the selling points that matter today, and the consumer expects this visual delight whether they are using a powerful corporate notebook computer, a mobile device, a sleek tablet, or a set-top box.

Delivering this experience requires designers and developers to work together like never before. Gone are the days where designers could throw pixel maps over the wall and expect developers to implement their vision. So too are the days when developers could code purely for performance without regard to visual appeal. Design/development/test must become an iterative cycle, not a linear path. The Qt framework is known for high runtime performance and small footprint, making it ideal for mobile, embedded, and netbook applications. The Qt 4.7 release extends Qt with QML, a declarative language that codes the way designers think, and Qt Quick elements that are the building blocks for applications, Each frame of a story board is declared as a branch in an element tree; each visual aspect of a frame is declared as a property of elements on the branch; each transition between frames can be decorated with a variety of animations and effects.

The Qt Quick runtime implements the UI and provides direct access to native APIs and performance boosts with C++ extensions where appropriate. And because the runtime is implemented in Qt, it delivers cross-platform reach and flexibility.

Qt Creator, a development environment built for collaboration between designers and developers, supports Qt Quick. Designers work in a visual environment, developers work in a full-featured IDE, and Qt Creator supports round-trip iteration from design, to code, to test, and back to design.

Qt is built for the way product teams work today. Core business logic is coded by developers and optimized for performance, the interface is crafted by designers working with visual tools, and integrated tooling supports round-trip iteration between the disciplines.

Qt Quick Module

The Qt Quick module is the standard library for writing QML applications. While the Qt QML module provides the QML engine and language infrastructure, the Qt Quick module provides all the basic types necessary for creating user interfaces with QML. It provides a visual canvas and includes types for creating and animating visual components, receiving user input, creating data models and views and delayed object instantiation.

The Qt Quick module provides both a QML API which supplies QML types for creating user interfaces with the QML language, and a C++ API for extending QML applications with C++ code.

Note: From Qt 5.1, a set of Qt Quick based UI controls is available to create user interfaces. Please see Qt Quick Controls for more information.

For those new to QML and Qt Quick, please see QML Applications for an introduction to writing QML applications.

DOCUMENT 3 : Extract from http://qt-project.org/wiki/Introduction_to_Qt_Quick

Qt Creator

Qt Creator is a development environment built for collaboration between designers and developers. Designers work in a visual environment, developers work in a full-featured IDE, and Qt Creator supports round-trip iteration from design, to code, to test, and back to design.

Qt Creator for Designers

For designers, Qt Creator includes a visual editor for QML files called Qt Quick Designer. It allows you to rapidly design and build Qt Quick applications and components from scratch. It is completely integrated into Qt Creator and works seamlessly with the QML code editor.
With Qt Quick Designer you can:

- Design Qt Quick user interfaces with drag and drop functionality
- Customize components or choose from library of standard QML elements
- Preview QML files instantly without compilation

Exporting Designs from Adobe Photoshop and other Graphics Software

The art embedded in QML projects can be imported from popular graphics software including Adobe Photoshop and GIMP. Each scene is converted into a single QML file with an Image or a Text element for each layer and saved on the development PC. Top-level layer groups are converted into merged QML Image elements. You can open the QML file in Qt Creator for editing. If you edit the file in Adobe Photoshop and export it to the same directory again, any changes you made in Qt Creator are overwritten. However, you can re-export graphical assets without recreating the QML code.

Qt Creator for Developers

Qt Creator makes it easy to switch from a visual editor when working with designers and a code editor when fine tuning design or building application logic. In Projects, double-click a .qml file to open it in the code editor. Snippet 17 shows the QML code used to place rectangle between two adjacent elements.

```
Rectangle {
    id: rectangle2
    anchors.right: rectangle3.left
    anchors.rightMargin: 15
    anchors.left: rectangle1.right
    anchors.leftMargin: 15
    anchors.bottom: parent.bottom
    anchors.bottomMargin: 15
    }
```

DOCUMENT 4 : Extract from http://qt-project.org/doc/qt-5/qtquick-usecase-integratingjs.html

Use Case - Integrating JavaScript in QML

JavaScript code can be easily integrated into QML to provide UI logic, imperative control, or other benefits.

Using JavaScript Expressions for Property Values

JavaScript expressions can be used in QML as bindings. For example:

```
Item {
    width: Math.random()
    height: width < 100 ? 100 : (width + 50) /  2
}
```

Note that function calls, like Math.random(), will not be revaluated unless their arguments change. So binding to Math.random() will be one random number and not revaluated, but if the width is changed in some other manner, the height binding will be reevaluated to take that into account.
Adding JavaScript Functions in QML

JavaScript functions can be declared on QML items, like in the below example. This allows you to call the method using the item id.

```
import QtQuick 2.0
Item {
    id: container
    width: 320
    height: 480

    function randomNumber() {
        return Math.random() * 360;
    }

    function getNumber() {
        return container.randomNumber();
    }

    MouseArea {
        anchors.fill: parent
        // This line uses the JS function from the item
        onClicked: rectangle.rotation = container.getNumber();
    }

    Rectangle {
        color: "#272822"
        width: 320
        height: 480
    }

    Rectangle {
        id: rectangle
        anchors.centerIn: parent
        width: 160
        height: 160
        color: "green"
        Behavior on rotation { RotationAnimation { direction: RotationAnimation.Clockwise } }
    }
}
```

Using JavaScript files

JavaScript files can be used for abstracting out logic from QML files. To do this, first place your functions inside a .js file like in the example shown.

```
// myscript.js
function getRandom(previousValue) {
    return Math.floor(previousValue + Math.random() * 90) % 360;
}
```

Then import the file into any .qml file that needs to use the functions, like the example QML file below.

```
import QtQuick 2.0
import "myscript.js" as Logic

Item {
    width: 320
    height: 480

    Rectangle {
        color: "#272822"
        width: 320
        height: 480
    }

    MouseArea {
        anchors.fill: parent
        // This line uses the JS function from the separate JS file
        onClicked: rectangle.rotation = Logic.getRandom(rectangle.rotation);
    }

    Rectangle {
        id: rectangle
        anchors.centerIn: parent
        width: 160
        height: 160
        color: "green"
        Behavior on rotation { RotationAnimation { direction: RotationAnimation.Clockwise } }
    }

}
```

For further details on the JavaScript engine used by QML, as well as the difference from browser JS, see the full documentation on Using JavaScript Expressions with QML.

DOCUMENTS 5 & 6 : Extract from http://qt-project.org/doc/qt-5/supported-platforms.html

Supported Platforms

Qt is a cross-platform application and UI framework. Using Qt, you can write GUI applications once and deploy them across desktop, mobile and embedded operating systems without rewriting the source code.In Qt 5 all platforms are created using the new Qt Platform Abstraction (QPA), which makes it easier than before to port Qt into a new operating system.

Qt is supported on a variety of 32-bit and 64-bit platforms, and can usually be built on each platform with GCC, a vendor-supplied compiler, or a third party compiler. Open GL (ES) 2.0 or DirectX 9 (with ANGLE) is required for Qt Quick 2. Widgets and Qt Quick 1 can be used also without hardware acceleration.

Desktop Platforms

You can develop with Qt on the following desktop platforms:

- Windows
- Linux/X11
- Mac OS X

Embedded Platforms

You can develop with Qt for the following embedded platforms:

- Embedded Android
- Embedded Linux
- Windows Embedded (Compact and Standard)
- Real-Time Operating Systems, such as QNX, VxWorks and INTEGRITY

Mobile Platforms

You can develop with Qt for the following mobile platforms supported by Digia:

- Androitd
- iOS

The following platforms also support Qt:

- BlackBerry 10
- Sailfish OS

Work in progress ports of Qt exist for:

- WinRT (including Windows Phone)
- Tizen

Qt Licensing **extracted from http://qt-project.org/doc/qt-5/licensing.html**

Qt is available under three different licensing options designed to accommodate the needs of our various users.

Qt licensed under our commercial license agreement is appropriate for development of proprietary/commercial software where you do not want to share any source code with third parties or otherwise cannot comply with the terms of the GNU LGPL version 2.1 or GNU GPL version 3.0.

Qt licensed under the GNU Lesser General Public License (LGPL) version 2.1 is appropriate for the development of Qt applications provided you can comply with the terms and conditions of the GNU LGPL version 2.1.

Qt licensed under the GNU General Public License (GPL) version 3.0 is appropriate for the development of Qt applications where you wish to use such applications in combination with software subject to the terms of the GNU GPL version 3.0 or where you are otherwise willing to comply with the terms of the GNU GPL version 3.0.

Qt documentation is licensed under the terms of the GNU Free Documentation License (FDL) version 1.3, as published by the Free Software Foundation. Alternatively, you may use the documentation in accordance with the terms contained in a written agreement between you and Digia. Please see http://qt.digia.com/licensing for an overview of Qt licensing.