

3 Environnement de développement

3.1 Environnement d'intégration continue

L'environnement d'intégration continue (jenkins) est optionnel car compliqué à mettre en place et coûteux. Néanmoins si le groupe possède un serveur personnel, cela constituerait une bonne idée. Jenkins permet de faire des tâches automatiques telles que la compilation du code ou l'exécution de tests. Il suffit de lancer une fois par semaine les tâches nécessaires au projet.

September 9, 2013

Moteur 3D

Hugo Balacey

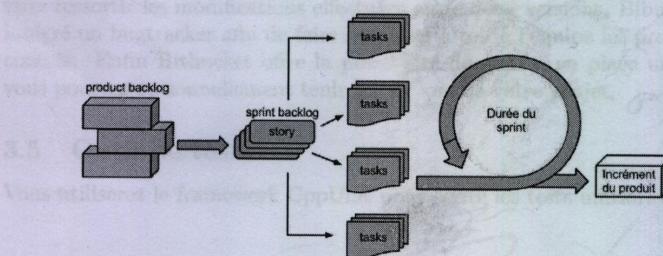
1 Description du projet

Vous disposerez d'un moteur de traitements sur des images 2D 3D et vous devrez mettre en place un moteur de visualisation 2D et 3D [C++/Qt].

2 Méthode de développement - Scrum

La méthode de développement utilisée sera Scrum via un outils en ligne gratuit Youkan. Le groupe pourra cependant proposer d'autre outils plus performant.

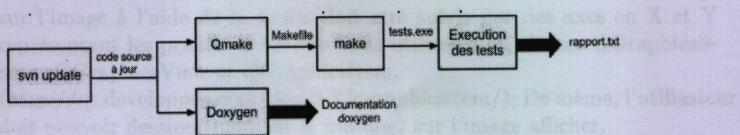
Rappel : Scrum est une méthode agile dédiée à la gestion de projets. La méthode s'appuie sur le découpage d'un projet en incrément, nommés sprint, ainsi que l'auto-organisation de l'équipe de développement. Scrum utilise une approche fonctionnelle pour récolter les besoins des utilisateurs. L'objectif est d'établir une liste de fonctionnalités à réaliser, que l'on appelle carnet du produit ou Product Backlog. Le sprint est une période d'une à quatre semaines maximum, au bout de laquelle l'équipe délivre un incrément du produit, potentiellement livrable. Une fois la durée choisie, elle reste constante pendant toute la durée du développement. Un nouveau sprint démarre dès la fin du précédent. Avant de démarrer un nouveau sprint, l'équipe doit réaliser une rétrospective, c'est à dire qu'elle analyse ce qu'il s'est passé durant ce sprint, dans le but d'effectuer une meilleure gestion du suivant. Un sprint est constitué d'une liste de story que l'on définit lors de la rédaction du sprint backlog, au départ de chaque nouveau sprint. Chaque story est ensuite divisé en plusieurs task afin de permettre une meilleure répartition de la charge de travail à effectuer durant le sprint.



3 Environnement de développement

3.1 Environnement d'intégration continue

L'environnement d'intégration continue (serveurs Jenkins) est optionnel car compliqué à mettre en oeuvre. Cependant si le groupe possède un serveur personnel, cela constituerai un plus pour le groupe en tant qu'expérience personnel et professionnel. Jenkins permet de pouvoir programmer des tâches automatisées telles que la compilation ou l'exécution des tests. L'ensemble de ces tâches constitue le build du projet, que nous avons décidé de lancer une fois par semaine. Le processus de build de notre serveur Jenkins peut être représenté par le schéma suivant :



3.2 Environnement de développement intégré

Nous vous conseillons d'utiliser QtCreator puisque le projet sera en c++/Qt. Mais un fois encore vous êtes libres pour ce point d'utiliser n'importe quel environnement de développement intégré.

3.3 Gestionnaire de versions

Vous mettrez en place un protocole de travail et d'utilisation de votre gestionnaire de versions. Vous pourrez utiliser Git ou SVN mais si vous avez déjà utiliser l'un des deux nous vous conseillons d'utiliser l'autre afin d'enrichir votre expérience personnelle.

3.4 Dépot

L'accès au code sur le net devra être restreint. Bitbucket a l'avantage d'être gratuit et privé. De plus, il permet la visualisation de l'ensemble des changements effectués avec l'auteur, la possibilité de comparer deux fichiers afin de faire ressortir les modifications effectuées entre deux versions. Bitbucket a aussi intégré un bugtracker afin de faire remonter à toute l'équipe les problèmes rencontrés. Enfin Bitbucket offre la possibilité de mettre en place un wiki. que vous pourrez optionnellement tenir à jour lors de votre projet.

3.5 Outil de tests

Vous utiliserez le framework CppUnit pour écrire les tests unitaires.

4 Interface graphique

4.1 Moteur 2D

reprendre le code existant et rajouter ou refaire certaine fonctionnalités :

- visualisation multi-Bureau : le client peut vouloir choisir de travailler sur un ou plusieurs écrans lorsque c'est possible. Pour cela vous utiliserez la classe QDesktopWidget.
(<http://qt.developpez.com/doc/4.7/qdesktopwidget/>);
- visualisateur d'image : le client doit pouvoir visualiser son image et le résultat de ses traitements sur des axes milimétrés. De même le passage sur l'image à l'aide de la souris doit être suivi par des axes en X et Y représentant les profils. Pour cela vous utiliserez les classes QGraphicsScene, QGraphicsView et QGraphicsItem.
(<http://qt.developpez.com/doc/4.7/qgraphicsitem/>); De même, l'utilisateur doit pouvoir dessiner(modifier le masque) sur l'image afficher.
- visualisateur d'histogramme : le client doit pouvoir visualiser l'histogramme de l'image en cours et modifier le contrast de l'image en manipulant l'histogramme. vous utiliserez encore une fois les classes QGraphicsScene, QGraphicsView et QGraphicsItem
- resize widget : vous apporterez un soin particulier aux méthodes de resize de vos différents widgets.

4.2 Moteur 3D

4.2.1 Objectifs

Le but est d'implémenter un moteur 3D composée de différentes fonctionnalités importantes parmi lesquelles :

- graphe de scène(<http://doc-snapshot.qt-project.org/qt5-dev/qtquick/qtquick-visualcanvas-scenegraph-renderer.html>)
- utilisations de shaders
- Vertex buffer objet (VBO)
- raffinement de maillage (http://http.developer.nvidia.com/GPUGems3/gpugems3_ch05.html) (http://theses.ulb.ac.be/ETD-db/collection/submitted/ULB-td-07112005-114016/unrestricted/06_chapitre6.PDF)

Ce moteur devra permettre de visualiser et sélectionner différent type d'objets (des points, des triangles, des polygones...). De même il devra pouvoir gérer OpenGL 1.4 et 3.3. Vos objets (primitives) devront être au centre de l'espace de visualisation (trackball), et votre projet devra être multi-plateformes (Windows linux mac).

3 possibilités s'offrent à vous :

- Soit vous êtes *puissant* et vous codez presque tout (en sachant que du code source vous est fourni pour la partie OpenGL 1.4 3.3);

- Soit vous êtes *moyen-chaud* et vous codez une partie en utilisant des bibliothèques extérieures et en reprenant le code fournis;
- Soit vous êtes *vraiment pas chaud* et vous partez d'un moteur 3D existant (Ogre ou IRlicht ou d'autre tant qu'ils sont libre). (<http://advancingusability.wordpress.com/2013/to-integrate-ogre3d-into-a-qt5-qml-scene/>)

4.2.2 Généralités

Vous implémenterez les primitives suivante pour la partie OpenGL 1.4 et 3.3 :

- dessin de textures [selectionnable]:

- void DrawTextureTransverse(*Point3D_t* <float> *min, *Point3D_t* <float> *max);
- void DrawTextureFrontale(*Point3D_t* <float> *min, *Point3D_t* <float> *max);
- void DrawTextureSagittale(*Point3D_t* <float> *min, *Point3D_t* <float> *max);

- dessin de triangles [selectionnable](pour les maillages triangulaires) :

- void DrawTriangle(float x1,float y1,float z1, float x2,float y2,float z2, float x3,float y3,float z3);
- void DrawLstTriangle();
- void DrawLstTriangleFilaire();
- void DrawLstTrianglePlein();

- dessin de Polygones [selectionnable](pour les maillages polygonaux):

- void DrawPolygone();
- void DrawLstPolygone();

- dessin de points (pour les points d'origine du maillage) :

- void DrawPoint();
- void DrawLstPoint();

- dessin de squelete [selectionnable](arbre (branche noeud)) :

- void DrawNode();
- void DrawBranch();[selectionnable]
- void DrawSkeleton();
- void DrawLstSkeleton();

- dessin de Texte [selectionnable] :

- void DrawText(); [selectionnable] pour modification

- dessin de généralités :

- void DrawRepere();

- void DrawBackground();
- void DrawBoundingBox();
- void DrawProfils();[selectionnable]

Chaque primitive (point, triangle, polygone, squelette, textures, ligne) pourra être visualisé sous la forme :

- forme d'origine;
- forme vectorielle;
- forme de tenseur (Cylindrique, sphérique). maillée.

Pour la partie openGl 3.3 vous apporterez un soins particulier à la conception et l'utilisation du moteur de shaders qu'il soient (géométrique, vertex, Fragment); Sous openGl 3.3 la forme de vos primitive sera géré par le geometry shader.

De même vous pourrez trouver et réimplémenter des shader de génération de maillage (marching cube shader)
[\(\[http://www.icare3d.org/codes-and-projects/codes/opengl_geometry_shader_marchingcubes.html\]\(http://www.icare3d.org/codes-and-projects/codes/opengl_geometry_shader_marchingcubes.html\)\)](http://www.icare3d.org/codes-and-projects/codes/opengl_geometry_shader_marchingcubes.html)
et pourquoi pas sur le net :

- Delaunay/Voronoi 3D;
- Poisson;
- autre...

4.2.3 Utilisation de la bibliothèque CGAL

Afin de générer vos maillages, vous utiliserez la bibliothèque CGAL(<http://www.cgal.org/>)

- Delaunay/Voronoi 3D;
- Poisson 3D;
- autre

Vous encapsulerez l'appelle de ses fonction dans une classe déjà fournis *CGALMapper*. un fichier d'installation vous est fournis.

4.2.4 Optimisation mémoire et raffinement de maillage (pour les puissants)

Afin d'alléger l'affichage la représentation des primitives se fera autant que possible via les VBOs (vertex buffers object). De plus, il faudra implémenter un système de raffinement de maillage permettant au moteur lors du zoom de raffiner le maillage ou au contraire de l'alléger lors du dézoom.

(http://http.developer.nvidia.com/GPUGems3/gpugems3_ch05.html)
(http://theses.ulb.ac.be/ETD-db/collection/submitted/ULBtd-07112005-114016/unrestricted/06_chapitre6.PDF)