Chef d'œuvre
Equipe OSIRIX

Rapport de conception générale

09 décembre 2011

Table des matières

Glossaire

- **Shaders**
  Suite d'instructions réalisée sur le GPU permettant de paramétrer une partie du processus de rendu telles que les réflexions, les textures, etc.

- **CPU**
  Central Processing Unit, composant de l'ordinateur qui exécute les programmes informatiques.

- **GPU**
  Graphics Processing Unit, circuit intégré présent dans la carte graphique et assurant les fonctions de calcul de l'affichage.

- **Texture**
  Ensemble de données associant des caractéristiques quelconques à des coordonnées géométriques. Par exemple, une texture peut définir, en chaque point d'un objet, sa couleur.

Bibliographie

- **« Interactive Reflection Editing »**
  (SIGGRAPH Asia 2009), par Tobias Ritschel, Makoto Okabe, Thorsten Thormählen et Hans-Peter Seidel

  http://www.mpi-inf.mpg.de/resources/ReflectionEditing/

- **« Interactive On-Surface Signal Deformation »**
  (SIGGRAPH 2010), par Tobias Ritschel, Thorsten Thormählen, Carsten Dachsbacher, Jan Kautz et Hans-Peter Seidel

  http://www.mpi-inf.mpg.de/resources/OnSurfaceDeform/

I - Introduction

Le but de ce chef d'œuvre est de permettre l'édition interactive d'images de synthèses, en agissant sur les différents aspects visuels qui la composent. Pour cela, il faudra implémenter les différents algorithmes et techniques présentés dans les sources suivantes : « Interactive Reflection Editing » et « Interactive On-Surface Signal Deformation ».

Le rapport de conception permet de décomposer le logiciel en composants hiérarchisés : sous-systèmes, modules et composants de base. Il précise pour chaque composant du logiciel : sa fonction et ses interfaces avec les autres composants.

Ce travail permet à l'équipe de développement d'avoir une première vue d'ensemble du projet, qui sera affinée dans le prochain document : le rapport de conception détaillée.

II - Architecture matérielle du système

Le système devra être exécuté sur un ordinateur tournant sous le système d'exploitation Fedora, qui est une distribution Linux. De plus, il devra tourner grâce à un microprocesseur de type x86.

Le logiciel développé devra utiliser le moteur de rendu 3D mis à disposition par le client et écrit en C++. De ce fait, le logiciel sera développé dans ce langage. De plus, OpenGL Shading Language (GLSL) sera utilisé pour la programmation des shaders. Ainsi, une partie de l'exécution sera assurée par le CPU, et l'autre par le GPU.

Le CPU prendra en charge toute la partie concernant les interactions homme-machine telles que la gestion de l'interface d'édition, ou l'ajout des contraintes sur la scène. Il devra également initialiser différents paramètres nécessaires aux traitements sur GPU.

Le GPU, quant à lui, aura à gérer l'essentiel des instructions. En effet, il devra effectuer les différents calculs permettant de modifier le rendu comme l'a spécifié l'utilisateur.

Différentes bibliothèques logicielles seront utilisées afin de simplifier le développement, et d'assurer une cohésion au sein du logiciel grâce à leur aspect multiplateforme.

Ainsi, les interactions homme-machine seront gérées grâce au framework Qt, permettant de créer une interface facilement, et d'établir des liens entre l'interface et les différents modules du système.

Une autre bibliothèque logicielle sera également utilisée pour le module du solveur, afin de pouvoir calculer le poids des différentes contraintes.

III - Diagramme de modélisation du système logiciel

Afin de bien représenter l'organisation du système logiciel, nous nous sommes appuyés sur un diagramme de classes, issu de la méthode UML.

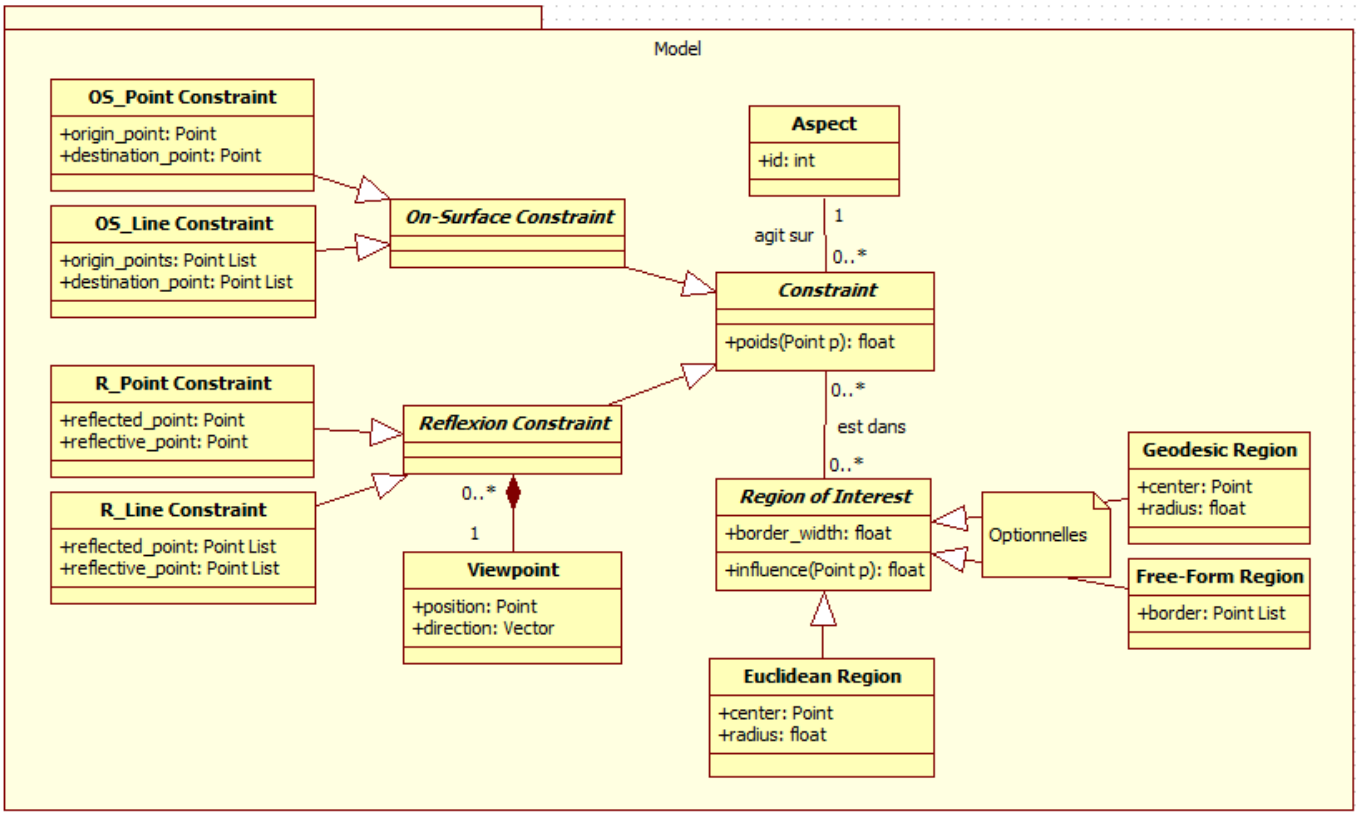

Figure 1 : Diagramme de classes – Partie Model

Ce diagramme représente les différentes données avec lesquelles on devra interagir.

Pour la partie interface graphique, on utilisera les classes présentes dans Qt, de la même manière pour la partie « Sur la surface » et pour la partie « Réflexion ».

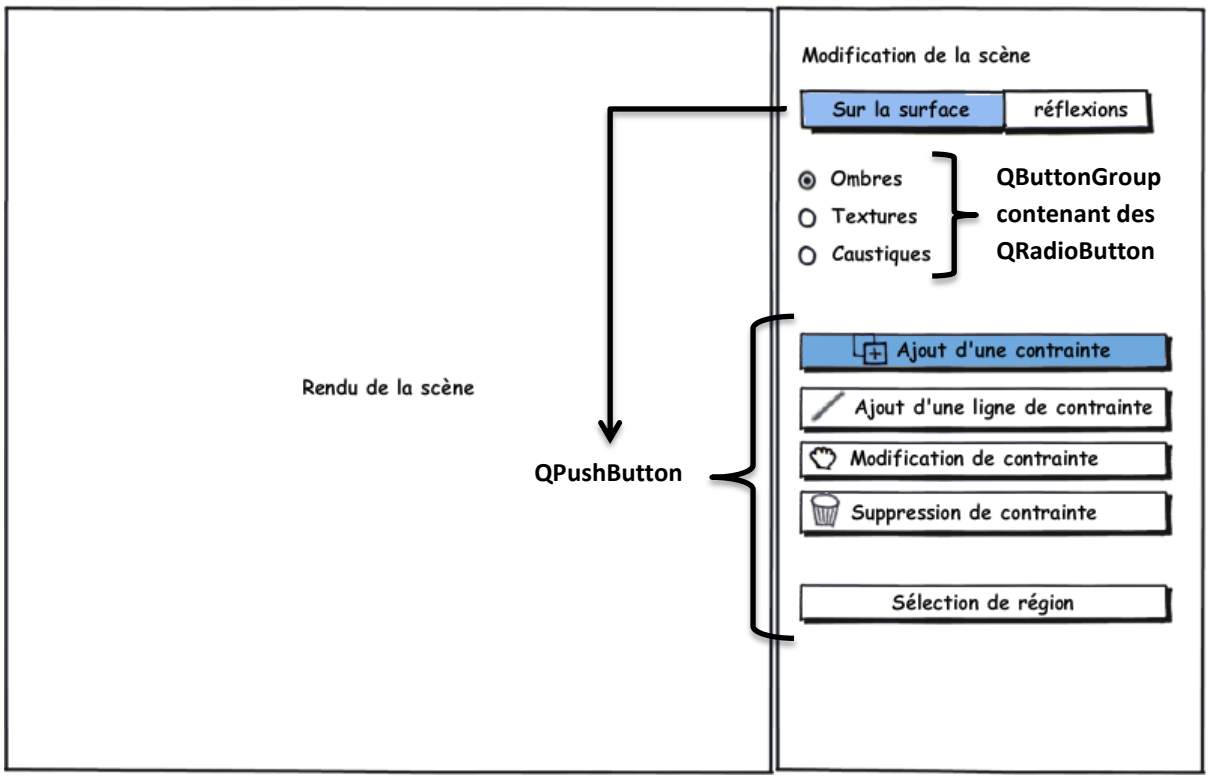

Figure 2 : Maquette de l'interface - Partie Sur la surface

IV - Décomposition en modules logiciels

Afin de séparer les différentes tâches que le programme doit effectuer, le logiciel sera décomposé en plusieurs modules.

Tout d'abord, on séparera l'interface homme-machine du moteur. Ainsi, l'interface sera composée d'un ensemble de classes qui n'auront aucun traitement calculatoire à faire et qui se contenteront de relayer les actions de l'utilisateur au moteur.

Le moteur lui sera décomposé en plusieurs parties.

- Une partie sur CPU qui fonctionnera comme une machine à états (cf. figure 2) et qui aura pour but d'assurer la cohérence entre l'interface, les données et les calculs. Même si plusieurs traitements seront communs, la partie CPU sera répartie en deux sous parties : Edition des réflexions et Edition sur la surface.
- Une partie sur GPU qui exécutera les phases de calculs numériques et de rendu, qui sera composée de plusieurs shaders. Lors des phases de rendu, les shaders seront exécutés selon les traitements demandés par l'utilisateur.

V - Comportement du système

Au travers d'un diagramme de cas d'utilisation et d'un diagramme d'états/transitions, nous allons représenter le comportement du système selon les actions de l'utilisateur.

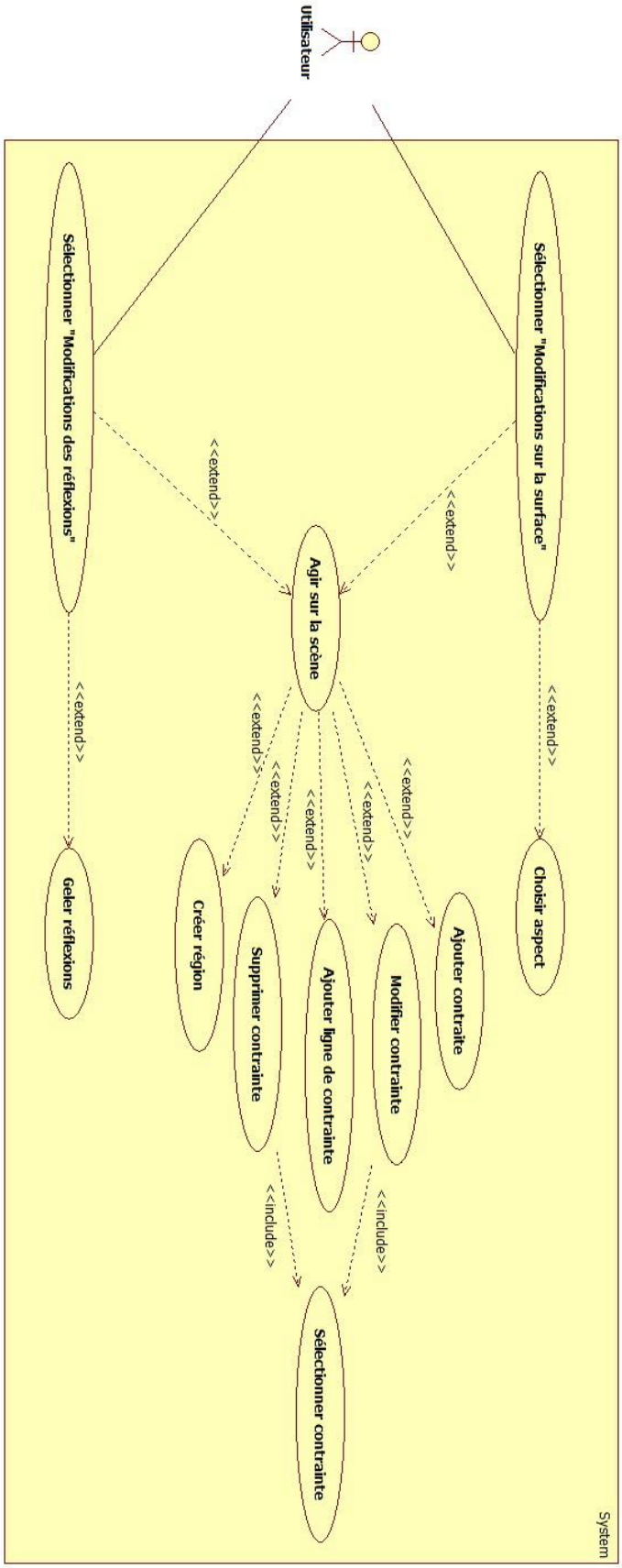

Figure 3 : Diagramme de cas d'utilisation

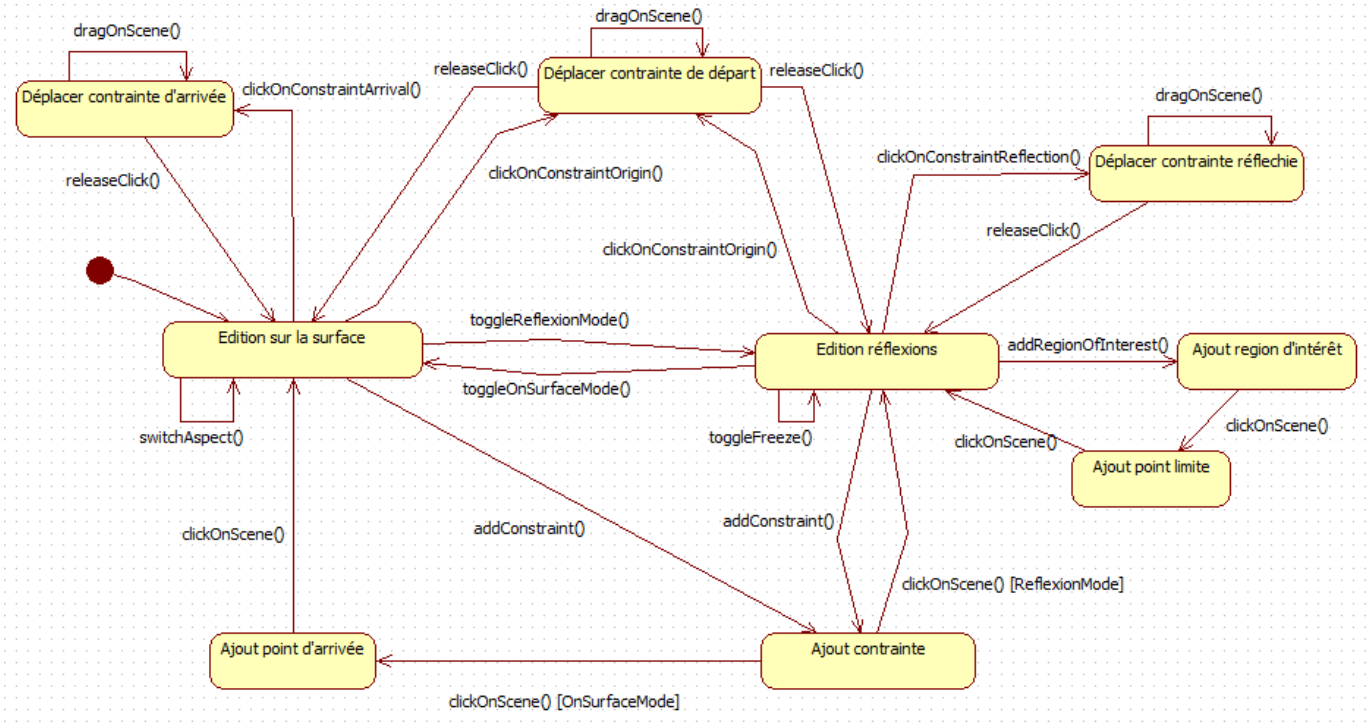

Figure 4 : Diagramme d'Etats/Transitions

Ces diagrammes montrent les différentes actions que l'utilisateur peut faire, ainsi que comment doit réagir le système. On représente ainsi le programme comme un automate déterministe, ce qui aidera l'équipe de programmation pour l'implémenter. Le cas d'utilisation « Agir sur la scène » est fictif et amène un peu plus de lisibilité au diagramme.

Afin de réaliser les traitements qu'il souhaite, l'utilisateur pourra interagir avec le logiciel au travers de plusieurs fonctions, modélisables grâce à des diagrammes de séquences.

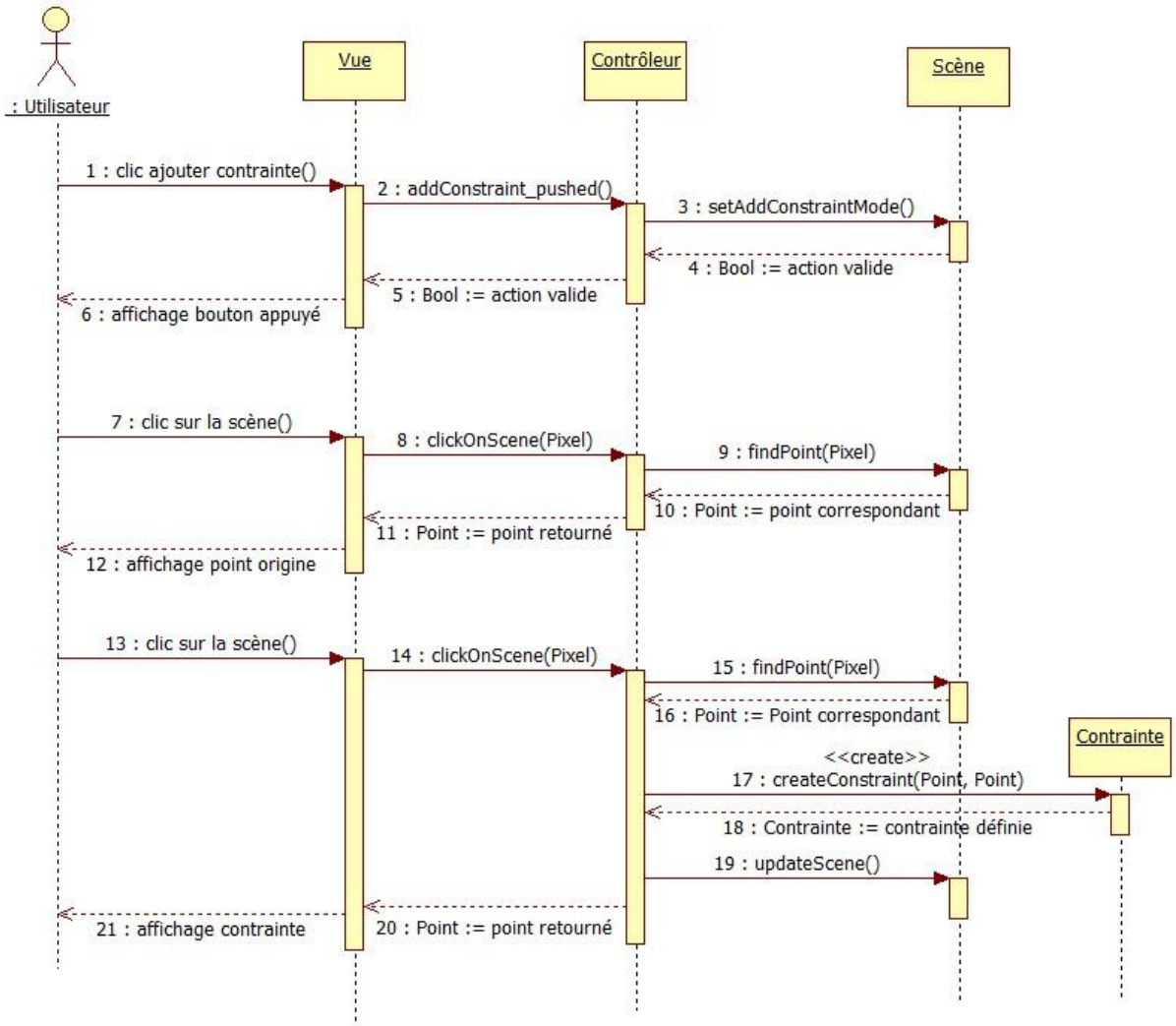

Figure 5 : Diagramme de séquence - Ajouter une contrainte

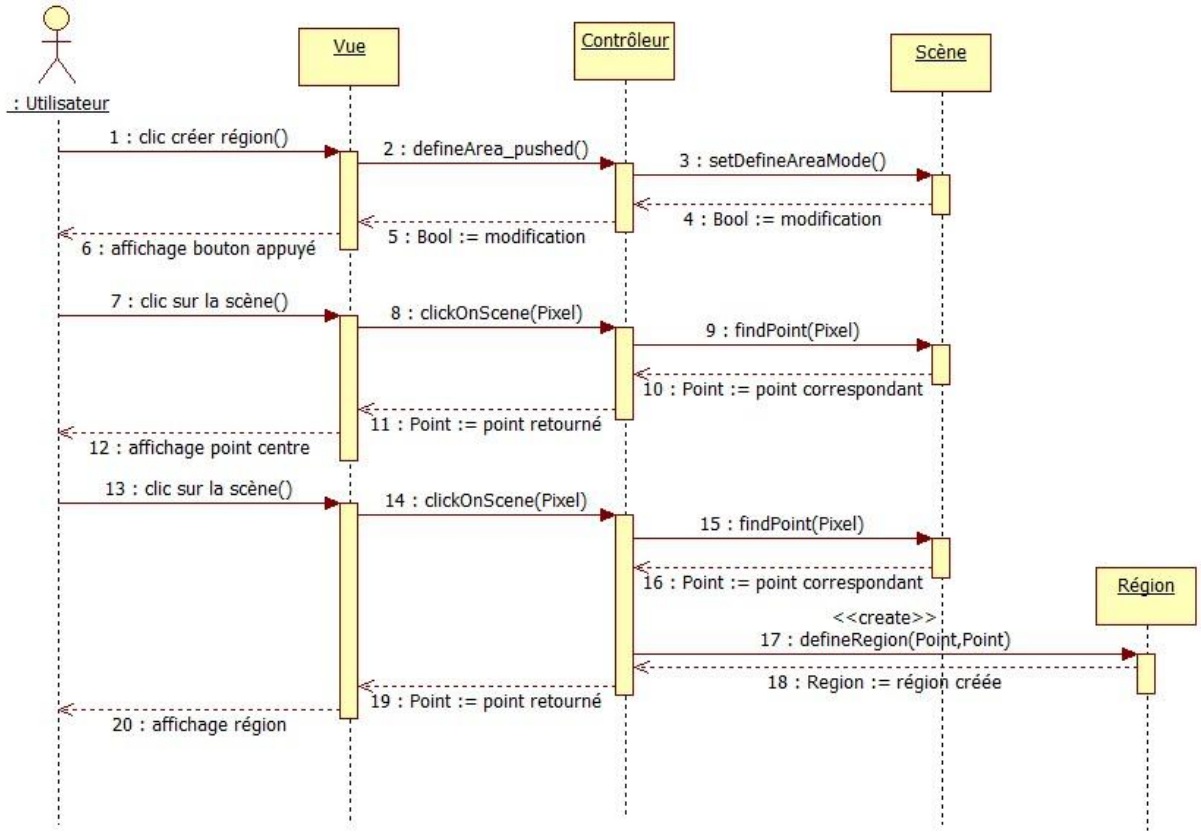

Figure 6 : Diagramme de séquence - Créer une région d'intérêt

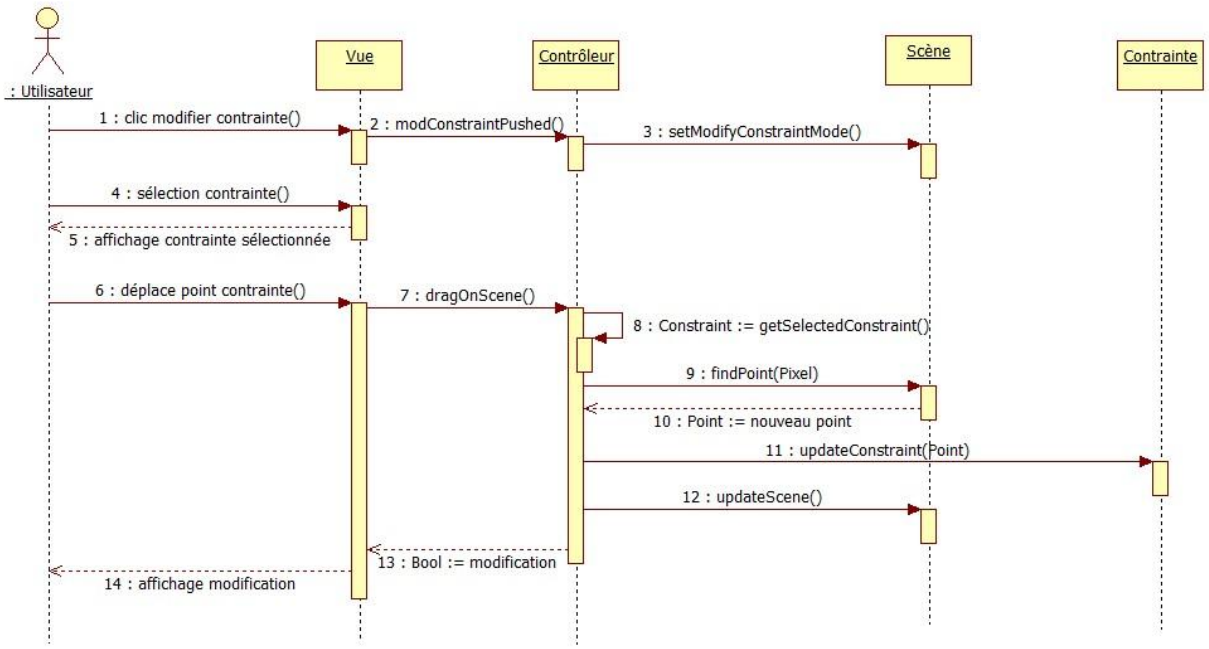

Figure 7 : Diagramme de séquence - Modifier une contrainte

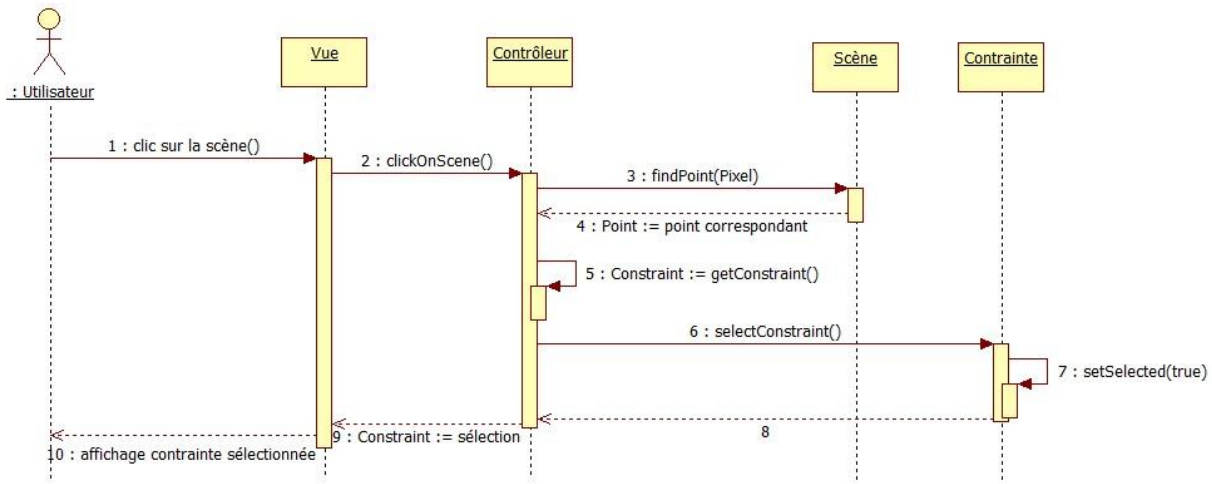

Figure 8 : Diagramme de séquence - Sélectionner une contrainte

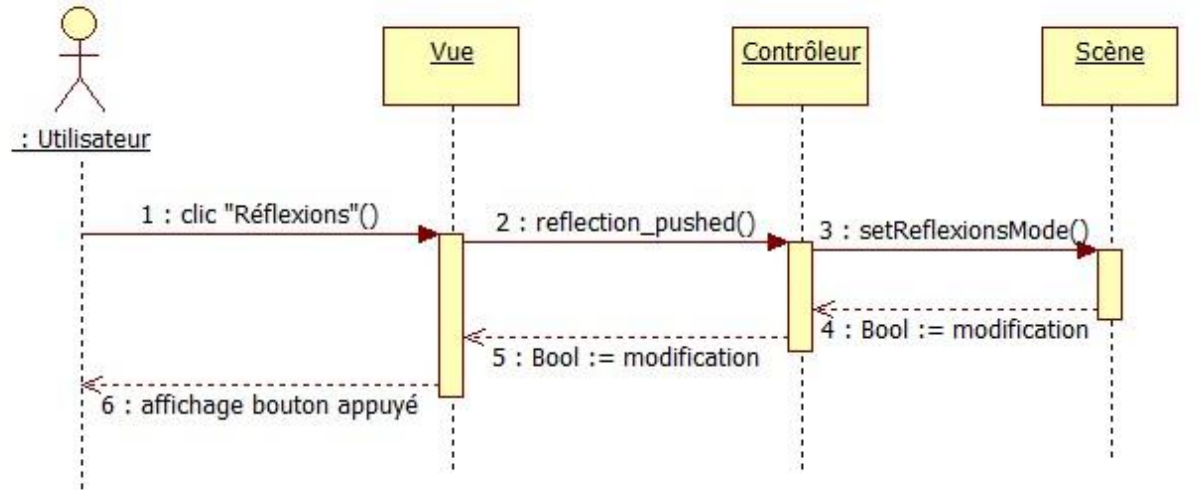

Figure 9 : Diagramme de séquence - Choisir mode Réflexion

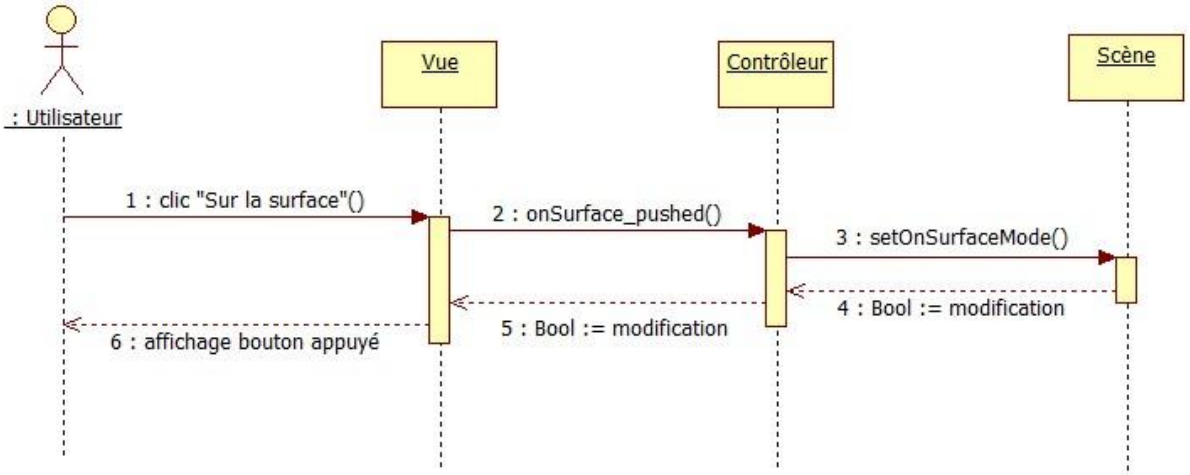

Figure 10 : Diagramme de séquence - Choisir mode Sur la surface

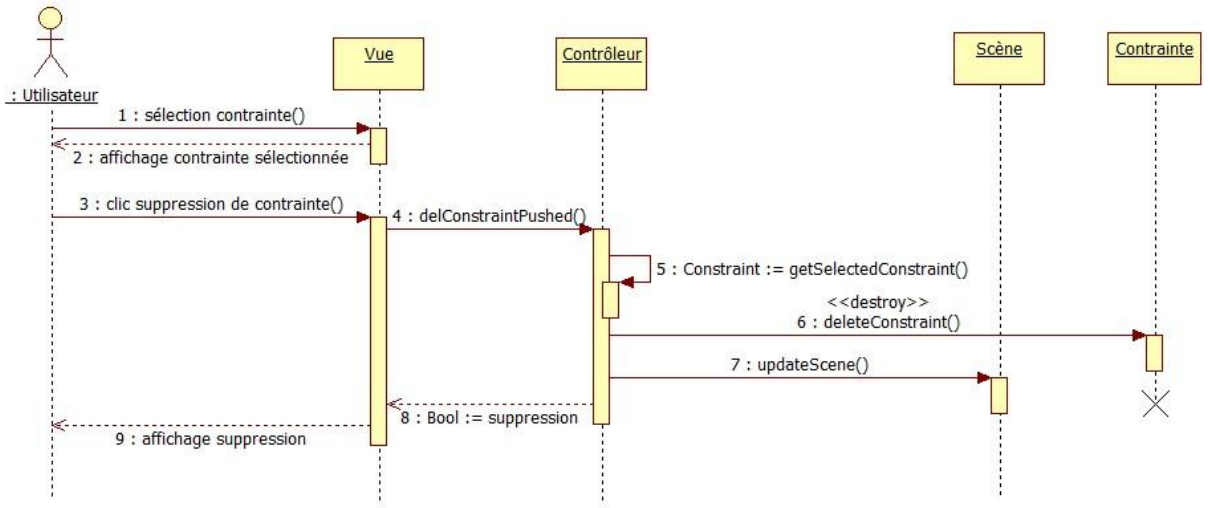

Figure 11 : Diagramme de séquence - Supprimer une contrainte

VI - Détails des fonctions système

- ## Fonctions obligatoires

1. La fonction **Aspect selectedAspect(Panel select)** prend en entrée le panel de sélection d'aspect, si l'onglet « réflexions » est sélectionné, l'aspect « reflection » est retourné. Si l'onglet « Sur la surface » est sélectionné, on retourne l'aspect correspondant au radio bouton sélectionné, c'est-à-dire « textures » ou « ombres ».
2. La fonction **TextureWeight defineWeight(RegionOfInterest region)** prend en entrée une zone d'effet et retourne une texture contenant le poids de la contrainte en chaque point.
3. La fonction **EuclideanRegion defineEuclideanRegion(Point center, Point limit)** prend en entrée le point d'origine de la région et le point permettant de définir le rayon de la zone d'intérêt.
4. La fonction **Point findPoint(Pixel p)** prend en entrée un pixel sélectionné de l'écran et retourne le point de la scène correspondant.
5. La fonction **Constraint createConstraint(Point origin, Point destination)** prend en entrée un point d'origine et un point de destination et retourne la contrainte définie par ces deux points.
6. Le produit disposera d'une option permettant de figer les réflexions obtenues d'un certain point de vue et de les garder identiques lors du déplacement de ce dernier, pour effectuer des modifications d'éditions (cf. figure 2-b, « Rapport de spécification »). Cette fonction **Freeze()** ne recalculera pas les vecteurs réfléchis mais les laissera calculés selon le point de vue duquel ils ont été définis.

- ## Fonctions optionnelles

1. Une fonction **RegionOfInterest defineRegion(Point center, Point limit, Panel selectDistance)** prend en entrée le point d'origine de la région, le point permettant de définir le rayon de la zone d'intérêt et le panel de sélection de distance et retourne la région d'intérêt définie par la fonction **EuclideanRegion defineEuclideanRegion(Point center, Point limit)** si le bouton distance euclidienne est sélectionné et la fonction **GeodesicRegion defineGeodesicRegion(Point center, Point limit)** si le bouton distance géodésique est sélectionné.

2. La fonction **RegionOfInterest defineRegion(Vector<Point> limits)** prend en entrée une liste de points décrivant le contour de la région définie par l'utilisateur et retourne la région d'intérêt correspondante.
3. On ajoutera au panel de sélection d'aspect un bouton permettant de sélectionner l'aspect « caustiques ».
4. La fonction **ConstraintLine createConstraint(Vector<Point> origin, Vector<Point> destination)** prendra une liste de points définissant une courbe d'origine et une liste de points définissant une courbe de destination et retourne la contrainte définie par ces deux lignes.

VII - Communication inter modules/sous-systèmes

Les deux sous-systèmes du logiciel (traitement des réflexions et traitement sur la surface) ne communiqueront pas directement, mais l'interface graphique permettra de basculer d'un sous-système à l'autre.

La communication entre l'interface graphique et les différents modules du logiciel se fera grâce à la bibliothèque logicielle Qt comme dit précédemment. Cette liaison sera donc faite grâce au système de signaux et slots. En effet, chaque objet peut se connecter à un autre objet par un envoi de signal. Ce signal est alors reçu par le slot prévu à cet effet, et exécute alors le code correspondant. Par exemple, lorsque l'on clique sur un bouton de l'interface graphique, le bouton va émettre un signal et un code correspondant à cet appui pourra alors être exécuté.

Une communication sera également faite entre le CPU et le GPU. En effet, le CPU devra gérer l'appel à différents shaders selon l'utilisation qui est faite. La gestion de la mémoire partagée pour le GPU sera assurée par les différents paramètres des shaders.

Index des illustrations :