Chef d'œuvre
Equipe OSIRIX

Rapport de conception détaillée

09 Janvier 2012

Table des matières

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 ».

L'objectif de la phase de conception détaillée est de produire tous les éléments nécessaires au codage du prototype il détaille complètement le rapport de conception générale.

Ce travail permet à l'équipe de développement de savoir précisément comment coder toutes les fonctionnalités du logiciel à livrer au client.

II - Implémentation de l'interface

Ce diagramme représente l'instantiation des différents objets de la vue. Ce sont des objets Qt qui permettent de créer l'interface visuelle de l'application.

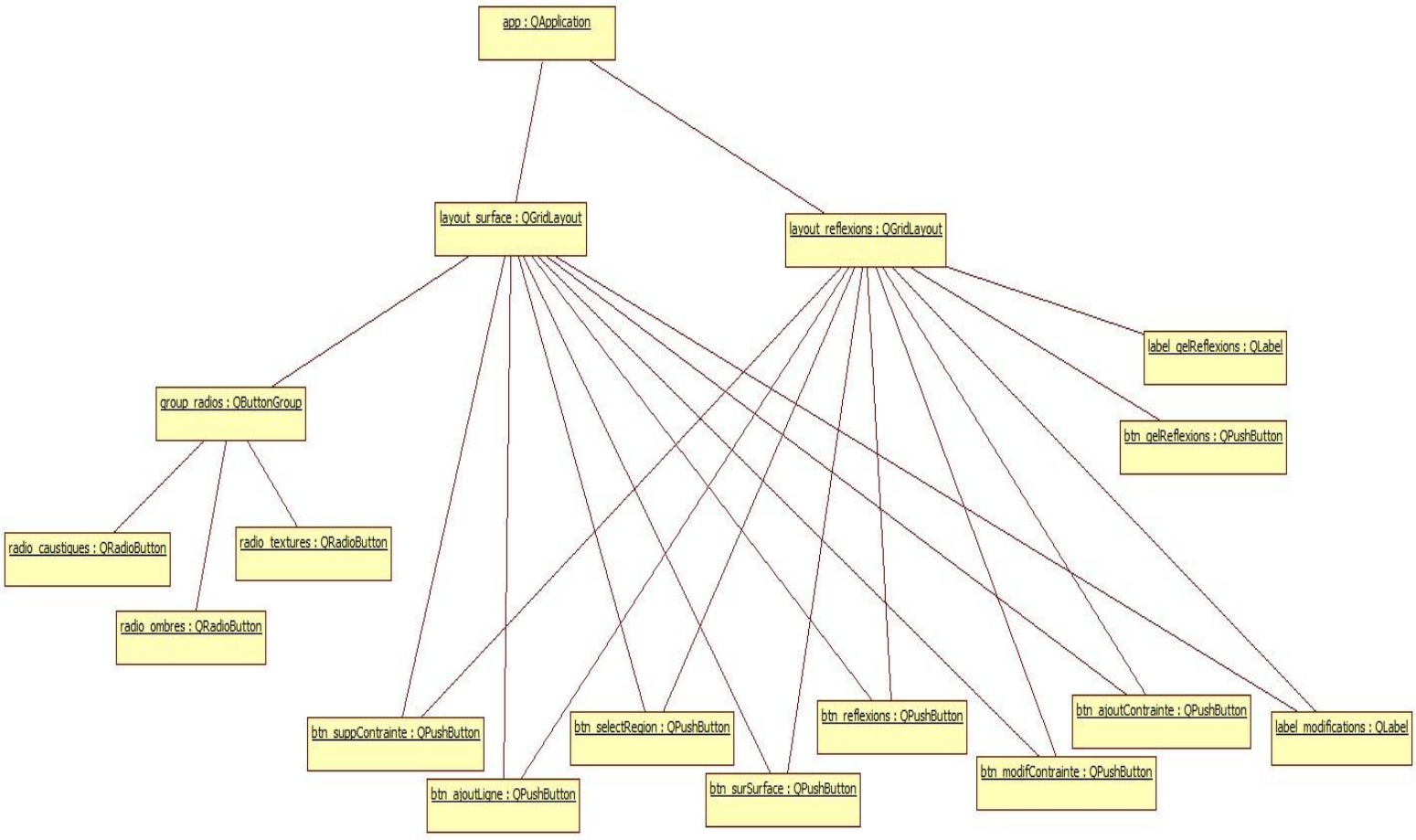


Figure 1 Diagramme d'objets

II - Ajouter Contrainte

Cette fonctionnalité permet de créer une contrainte dans la scène. Cette contrainte sera représentée par deux points : un rouge et un vert correspondant au point d'origine et au point de destination de la contrainte.

Lorsque l'utilisateur clique sur le bouton "Ajouter contrainte" sur l'interface, le système se charge de désactiver tous les modes, et d'activer le mode d'ajout de contrainte "AddConstraintMode".

L'aspect du bouton est donc changé pour montrer qu'il est sélectionné, et le curseur sur la scène également afin de montrer à l'utilisateur qu'il peut ajouter une contrainte.

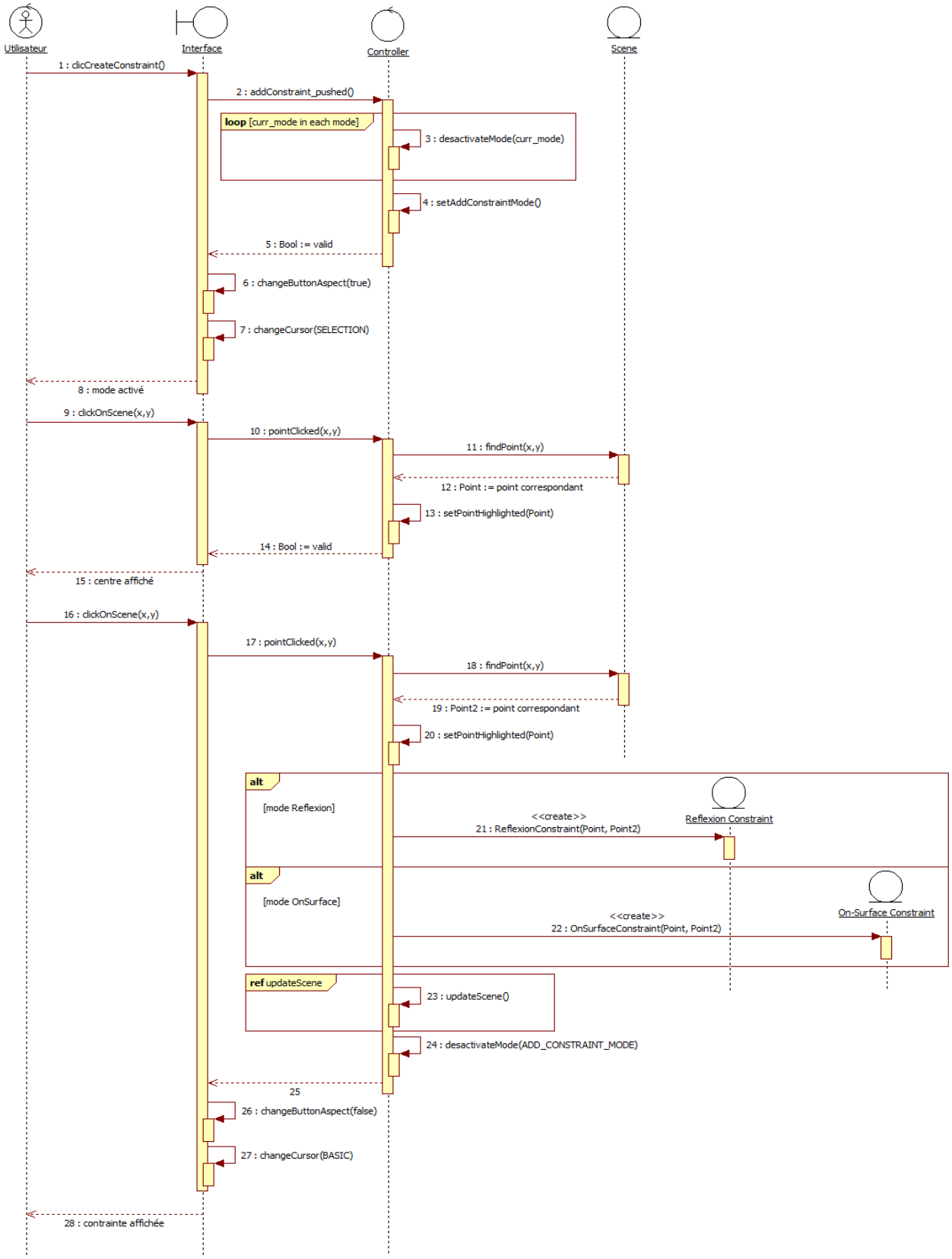
Lorsque l'utilisateur clique une première fois sur la représentation de scène, il va définir le point d'origine de la contrainte. Le système récupère le point 3D correspondant au clic de l'utilisateur, et l'affecter comme point d'origine de la contrainte.

Sur le même principe, l'utilisateur peut définir le point de destination de la contrainte.

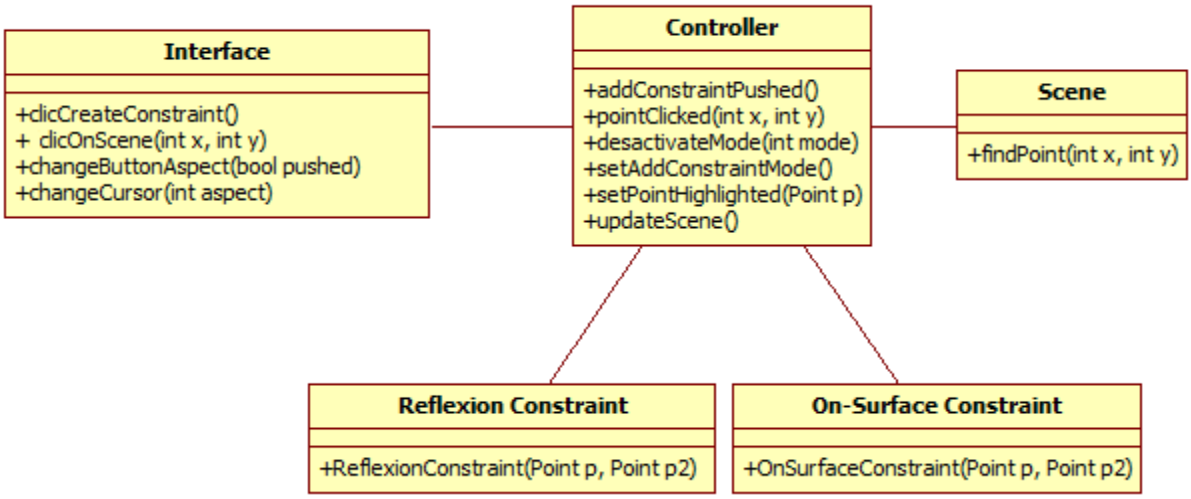
Une contrainte est alors créée avec ces deux points, et on met à jour la scène afin d'effectuer les modifications nécessaires.

Les modifications visuelles retournent ensuite à la normale (bouton non appuyé, curseur standard).

1) Diagramme de séquence détaillé



1) Diagramme de classes participantes



IV – Modifier Contrainte

Cette fonctionnalité permet de modifier une contrainte existante sur la scène. On peut alors déplacer le point d'origine, ou le point de destination de la contrainte, ce qui influera sur la scène.

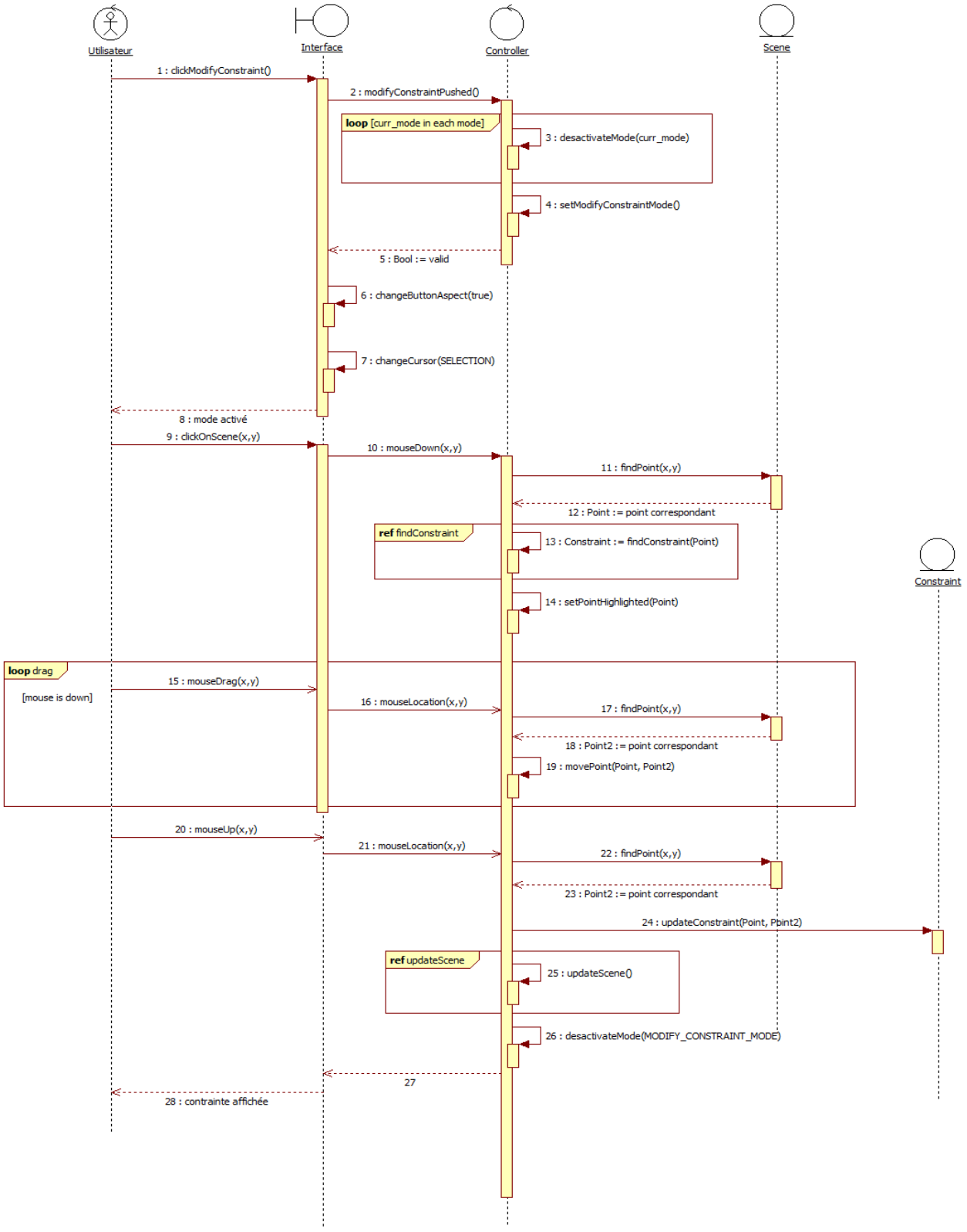
Comme précédemment, lorsque l'utilisateur clique sur le bouton de modification de la contrainte, le système se charge de rendre un feedback visuel au niveau du bouton et du curseur de la souris.

Ensuite, le système va chercher, parmi toutes les contraintes présentes dans la scène, la contrainte dont le point correspond à l'endroit où a cliqué l'utilisateur.

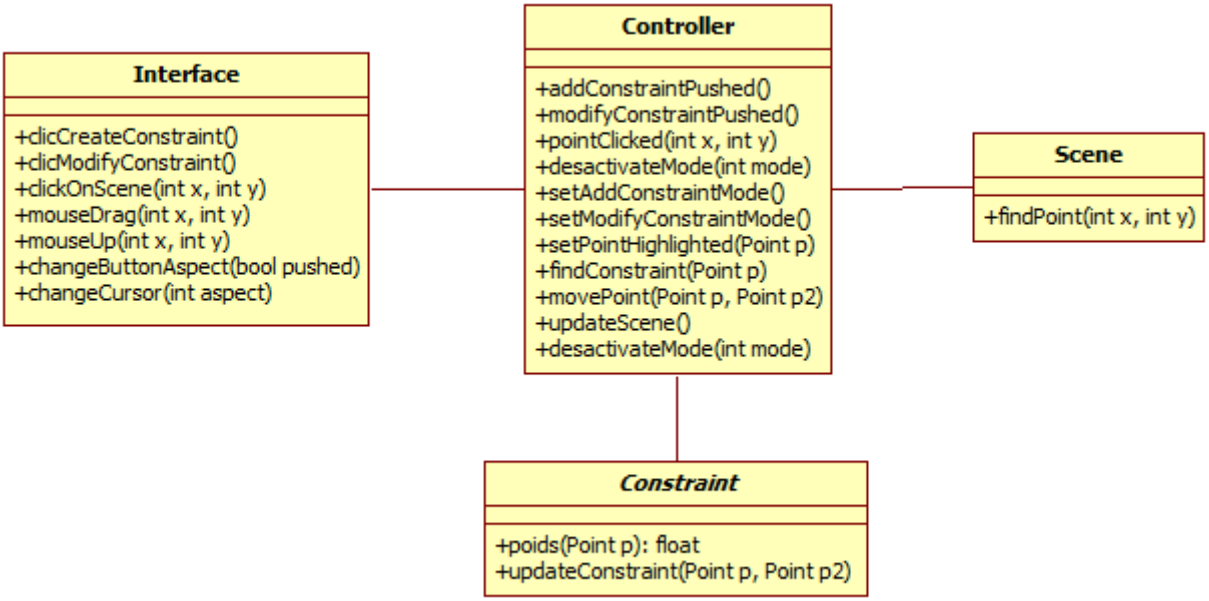
L'utilisateur peut alors glisser-déposer un point de la contrainte. Une fois le point deposé, le système met à jour la scène pour prendre en compte la modification faite.

Le système retourne ensuite à la normale, au niveau du visuel et au niveau du mode.

1) Diagramme de séquence détaillé



2) Diagramme de classes participantes


Interface
+clicCreateConstraint()
+clicModifyConstraint()
+clickOnScene(int x, int y)
+mouseDrag(int x, int y)
+mouseUp(int x, int y)
+changeButtonAspect(bool pushed)
+changeCursor(int aspect)
Controller
+addConstraintPushed()
+modifyConstraintPushed()
+pointClicked(int x, int y)
+desactivateMode(int mode)
+setAddConstraintMode()
+setModifyConstraintMode()
+setPointHighlighted(Point p)
+findConstraint(Point p)
+movePoint(Point p, Point p2)
+updateScene()
+desactivateMode(int mode)
Scene
+findPoint(int x, int y)
Constraint
+poids(Point p): float
+updateConstraint(Point p, Point p2)

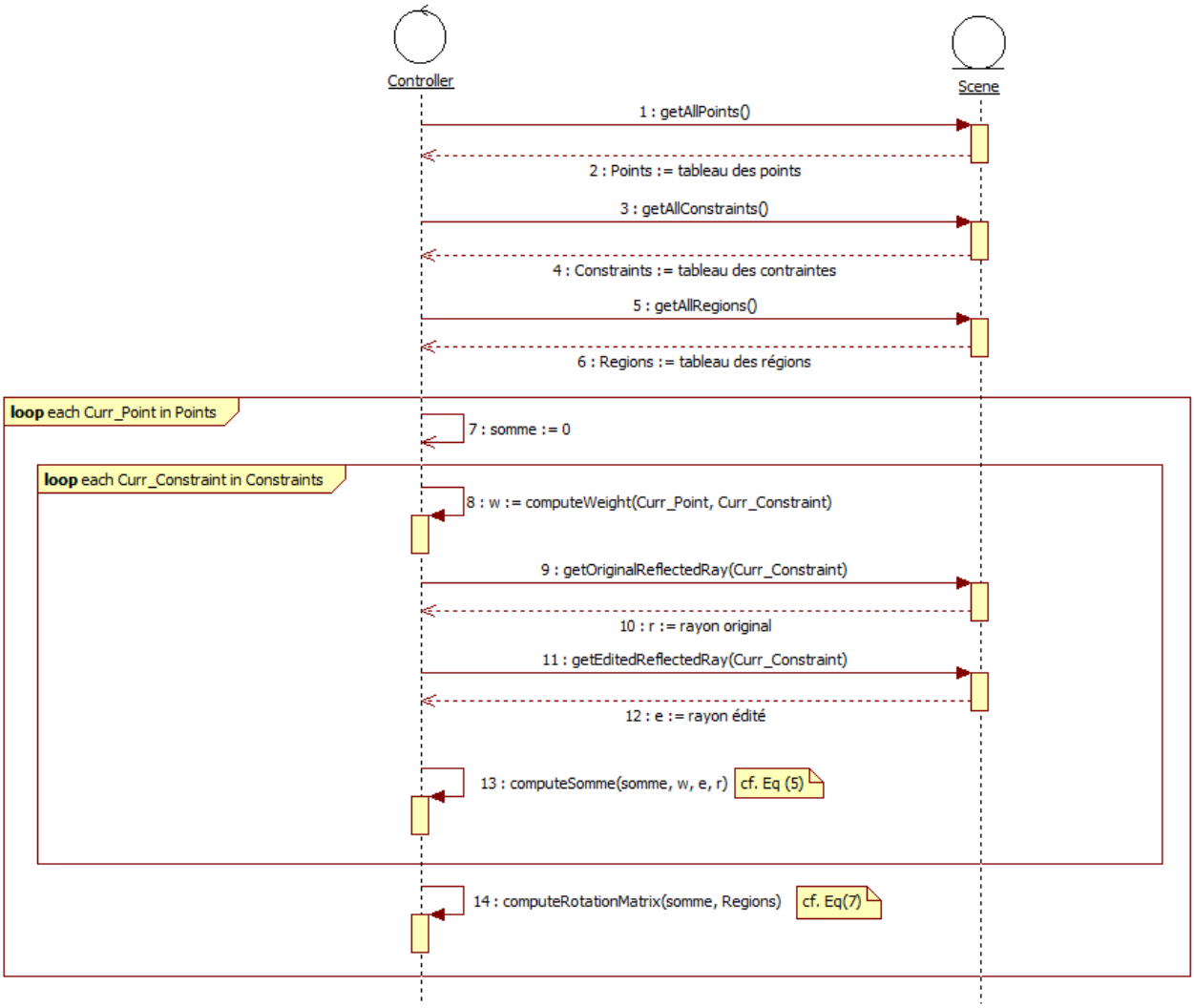
V – Update Scene

Cette fonction va se charger de prendre en compte les modifications spécifiées par l'utilisateur à travers l'IHM afin de les appliquer sur la scène.

Le système récupère donc tous les points de la scène, ainsi que tout ce qui a été défini par l'utilisateur (contraintes et régions). Ensuite, le système est chargé de calculer le rayon édité a partir de ces données et du rayon d'origine.

Pour cela, le système calcule les poids appliqués sur chaque point de la scène, pour chaque contrainte spécifiée. Une matrice de rotation est ensuite calculée, permettant de définir le rayon édité.

Diagramme de séquence détaillé



VI – Créer une Région d'intérêt

Cette fonction permet à l'utilisateur de définir une région d'intérêt sur la scène afin de délimiter la zone d'influence des contraintes.

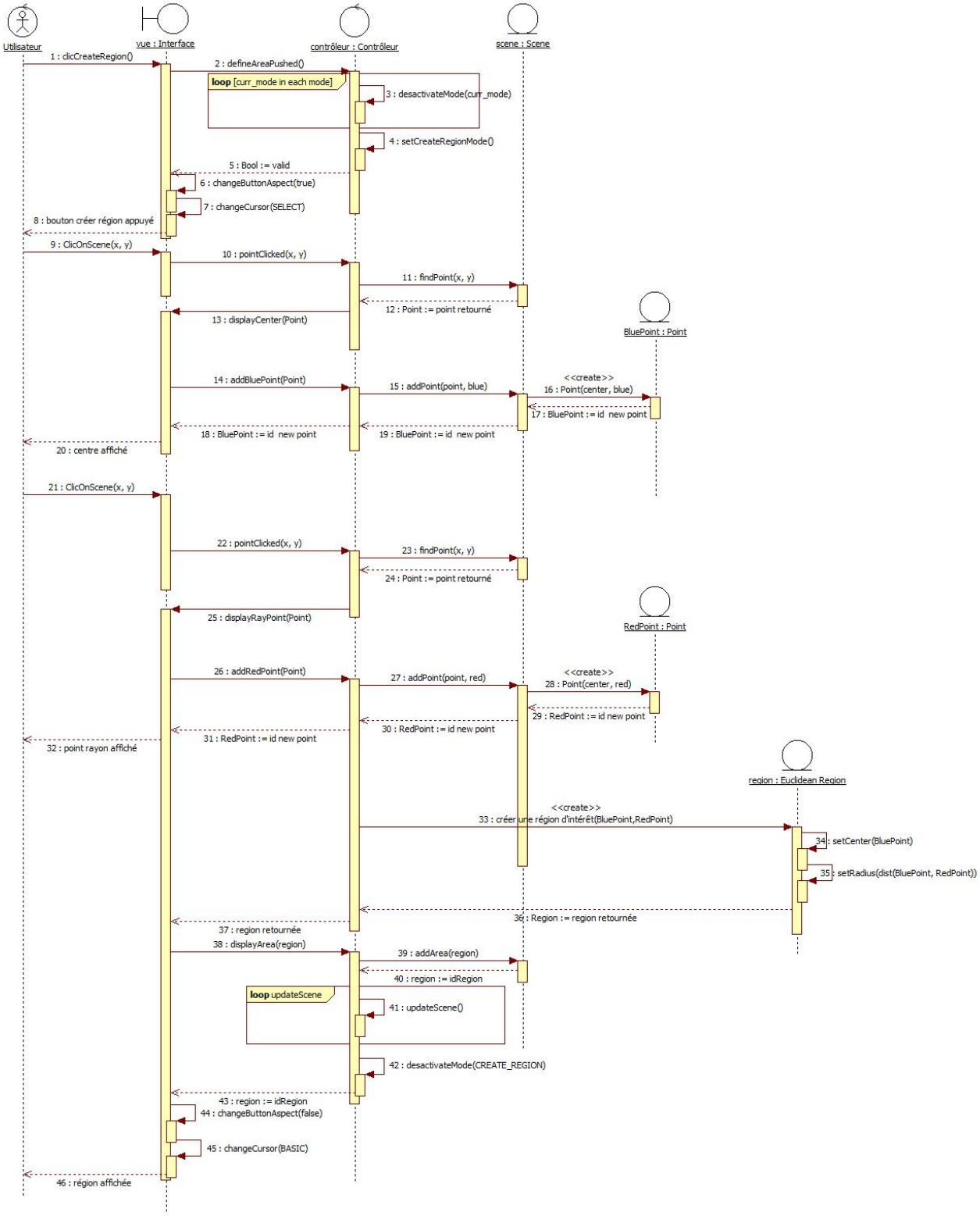
Lors de l'appui du bouton d'ajout de région, le système va modifier le mode, ainsi que l'aspect visuel (bouton, curseur).

Ensuite, lorsque l'utilisateur fait un premier clic, le système va rechercher le point 3D correspondant afin de le définir comme centre de la région d'intérêt (point bleu affiché sur la scène).
Au deuxième clic, l'utilisateur définir le rayon de la zone d'intérêt. De la même façon que précédemment, le système récupère le point 3D correspondant (point rouge sur la scène).

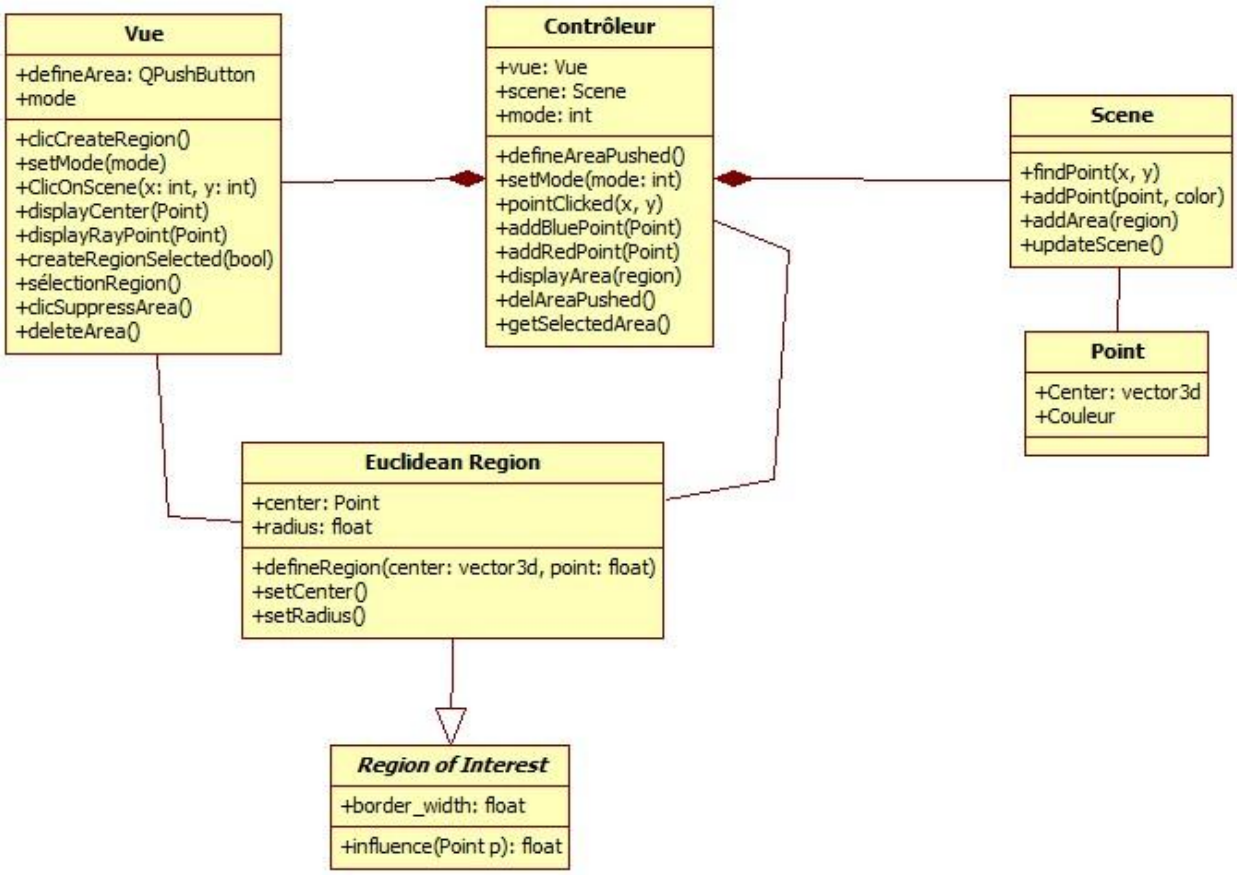
La région est ensuite créée avec ces deux points, et affichée sur la scène. La scène est ensuite mise à jour afin de prendre en compte la nouvelle région.

Le système retourne ensuite à la normale.

1) Diagramme de séquence détaillé



2) Diagramme de classes participantes



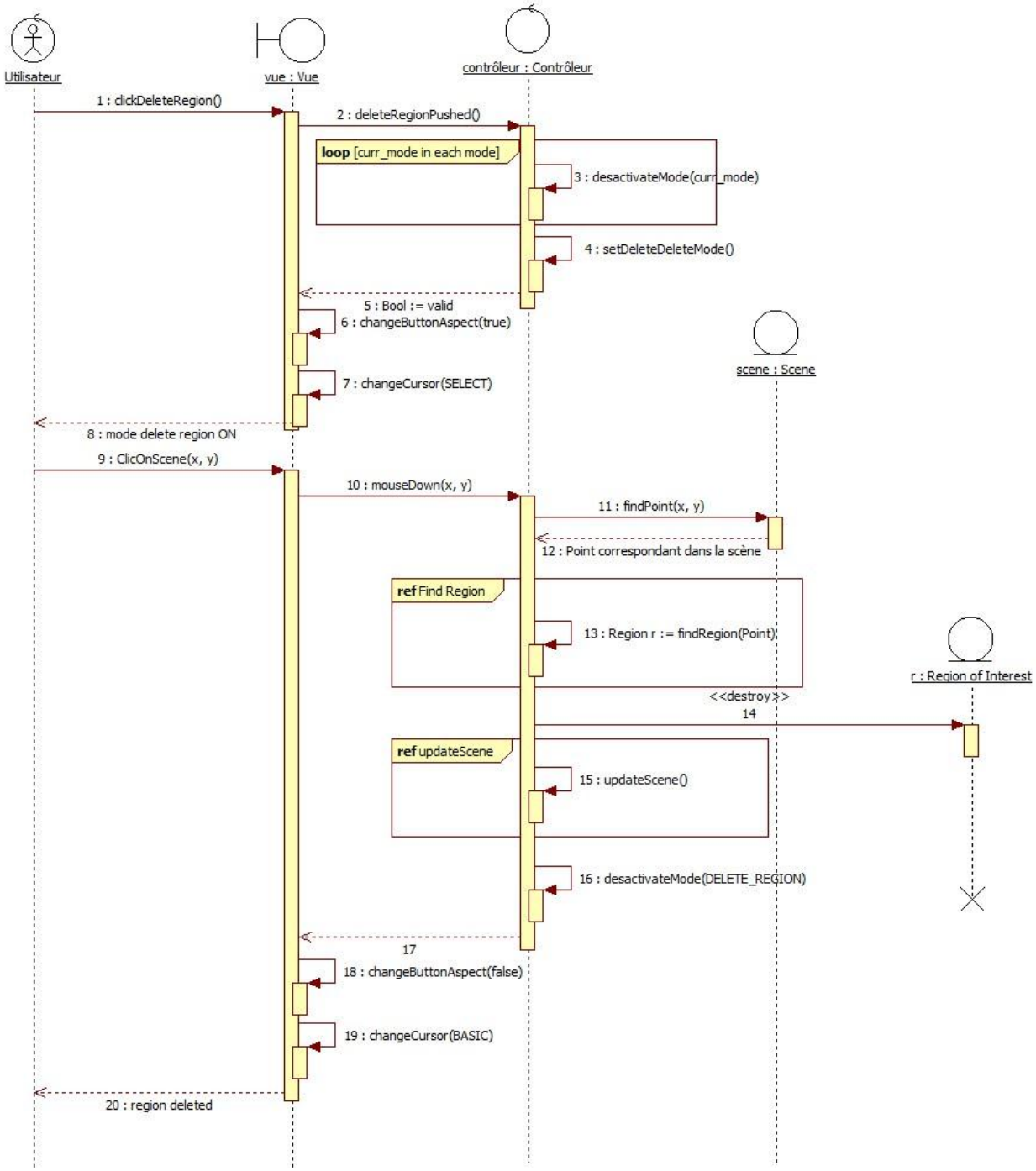
VII – Supprimer une Région

La fonction supprimer une Région intervient lorsque l'utilisateur clique sur le bouton correspondant sur l'interface.

Le système doit annuler toutes les tâches actuelles et modifier l'aspect et le mode au clic sur du bouton.

L'utilisateur clique ensuite dans la scène pour sélectionner la Région qu'il souhaite supprimer. Le système doit effectuer un certain nombre d'opérations suite à ce clic, récupérer les coordonnées de ce point dans la scène (en 3D) puis trouver la Région d'Intérêt la plus proche de ce point (référence au diagramme de classe findRegion présenté en VIII). Le contrôleur pourra alors supprimer la Région d'Intérêt correspondante et faire la mise à jour de la scène, les points qui étaient sous l'influence de contraintes dans cette Région retrouvent leurs réflexions physiques.

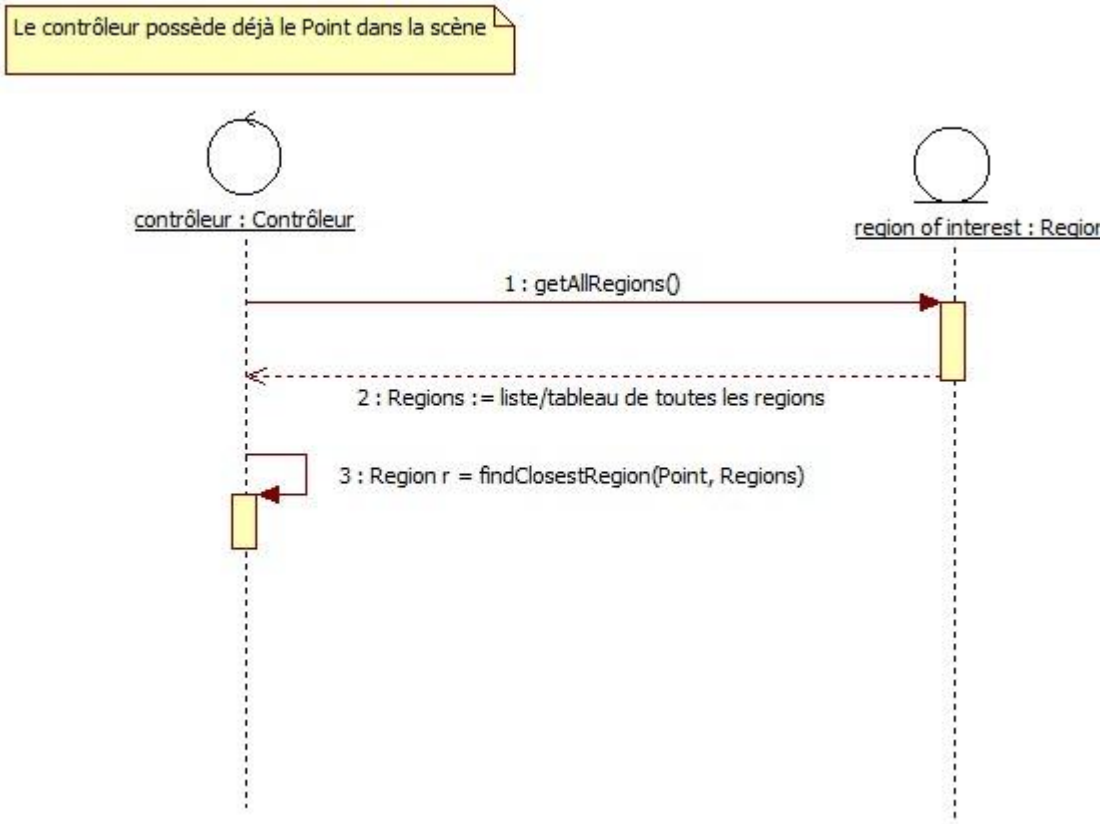
Le système retourne ensuite à la normale, au niveau visuel et au niveau du mode.


Utilisateur
vue : Vue
contrôleur : Contrôleur
1 : clickDeleteRegion()
2 : deleteRegionPushed()
loop [curr_mode in each mode]
3 : desactivateMode(curr_mode)
4 : setDeleteDeleteMode()
5 : Bool := valid
6 : changeButtonAspect(true)
scene : Scene
7 : changeCursor(SELECT)
8 : mode delete region ON
9 : ClicOnScene(x, y)
10 : mouseDown(x, y)
11 : findPoint(x, y)
12 : Point correspondant dans la scène
ref Find Region
13 : Region r := findRegion(Point);
r : Region of Interest
<<destroy>>
14
ref updateScene
15 : updateScene()
16 : desactivateMode(DELETE_REGION)
17
18 : changeButtonAspect(false)
19 : changeCursor(BASIC)
20 : region deleted

VIII – Trouver une Région

Cette fonctionnalité permet de trouver une région, selon un point cliqué par l'utilisateur.

Pour cela, le système récupère toutes les régions de la scène, et cherche parmi chacun des centres, celui dont les coordonnées sont les plus proches du point cliqué. Cette région est alors retournée.



IX – Supprimer une contrainte

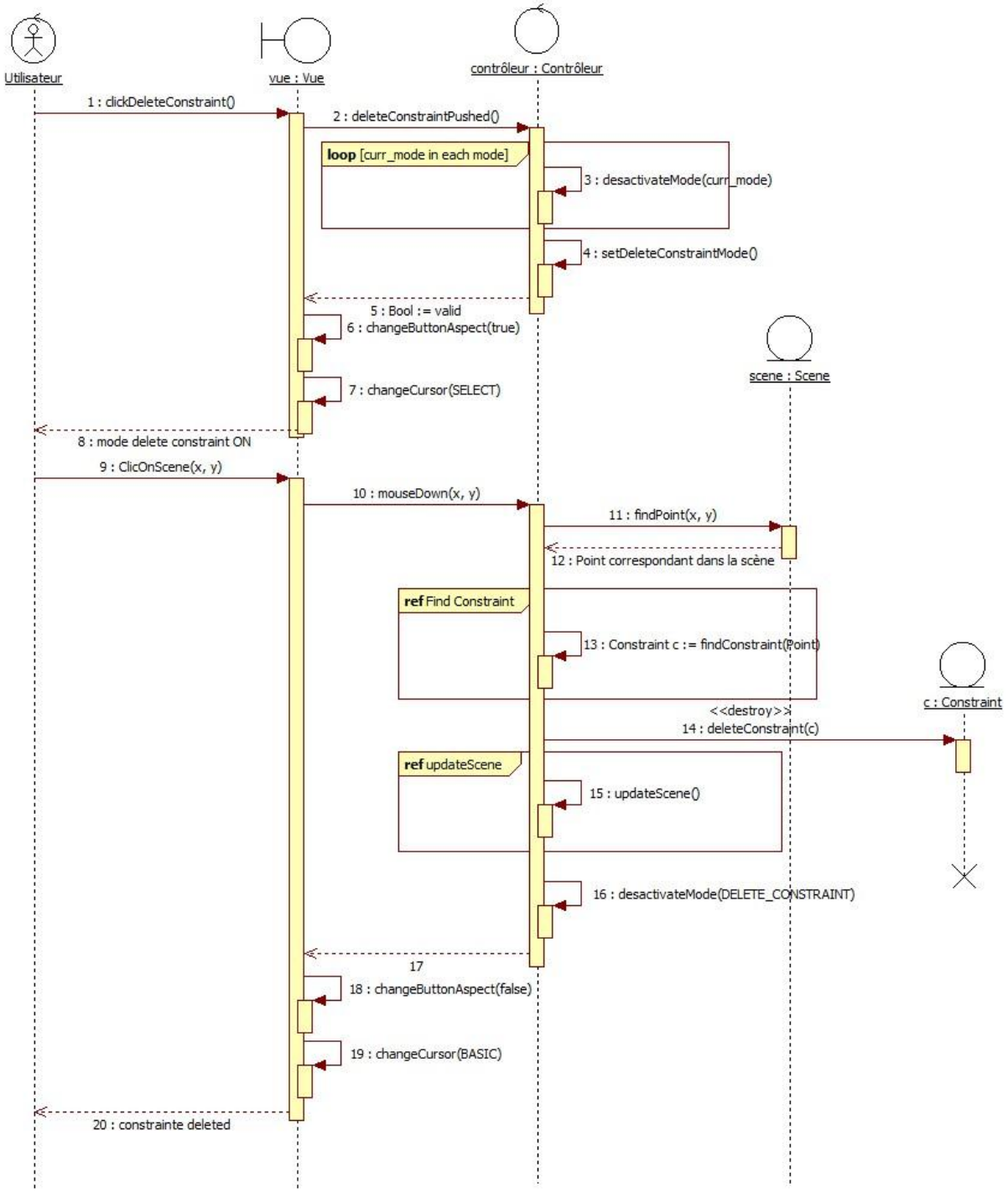
Cette fonction permet à l'utilisateur de supprimer une contrainte de la scène, et donc d'enlever les modifications qu'elle engendrait.

Pour cela, le système modifie l'aspect et le mode au clic sur le bouton "Supprimer contrainte".

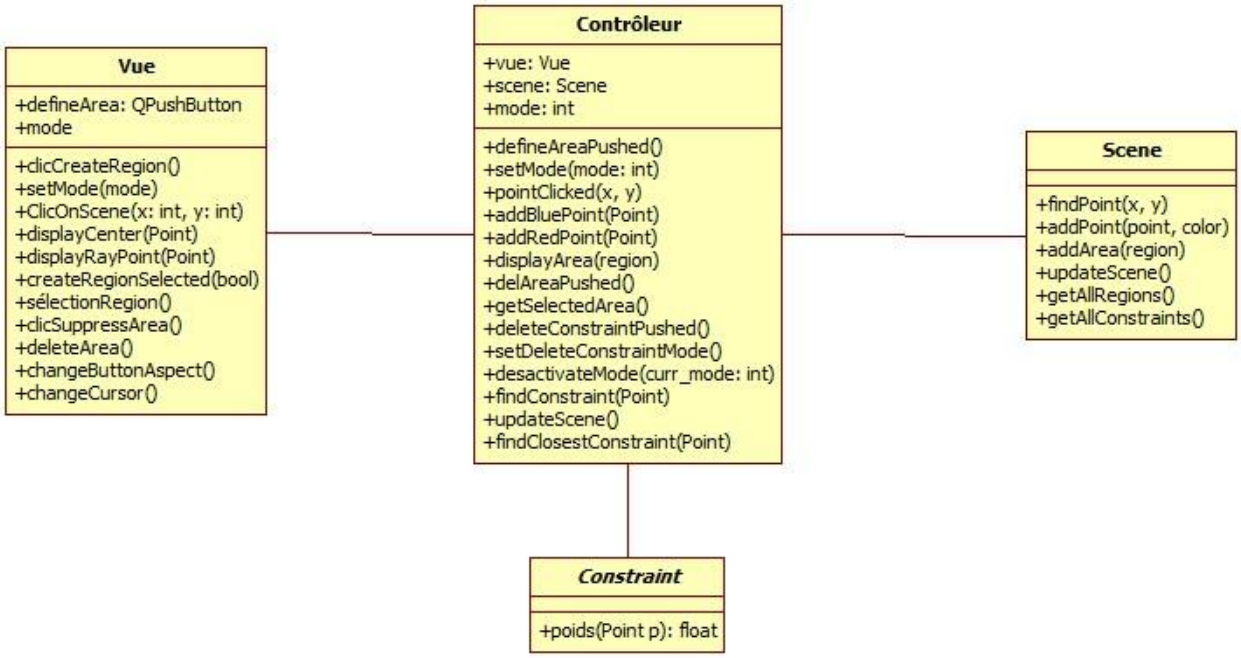
Ensuite, lorsque l'utilisateur clique sur un point de la scène, le système va rechercher la contrainte la plus proche du point cliqué. Cette contrainte est alors supprimée et les deux points (origine et destination) la représentant sont également supprimés.

Le système retourne ensuite à la normale, au niveau du visuel et au niveau du mode.

1) Diagramme de séquence détaillé



2) Diagramme de classes participantes

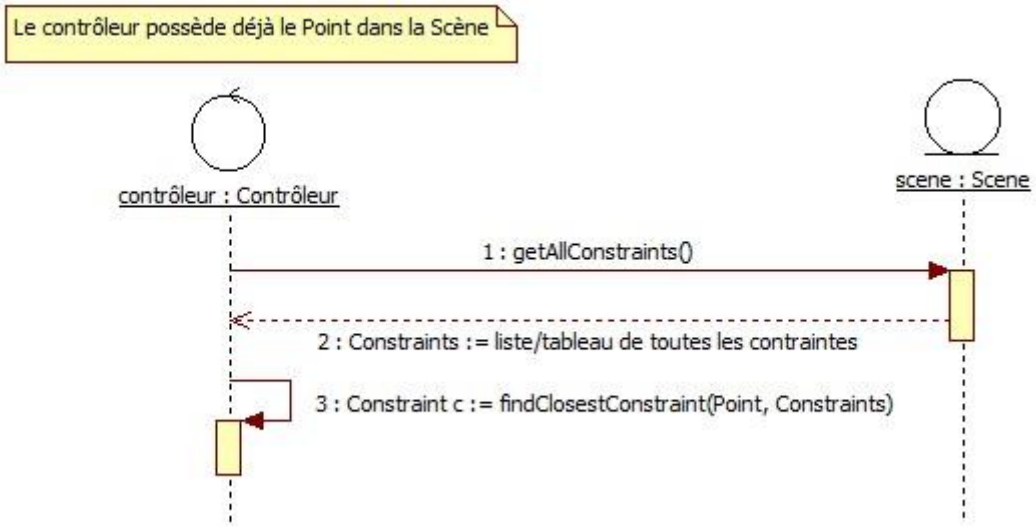


X – Trouver une contrainte

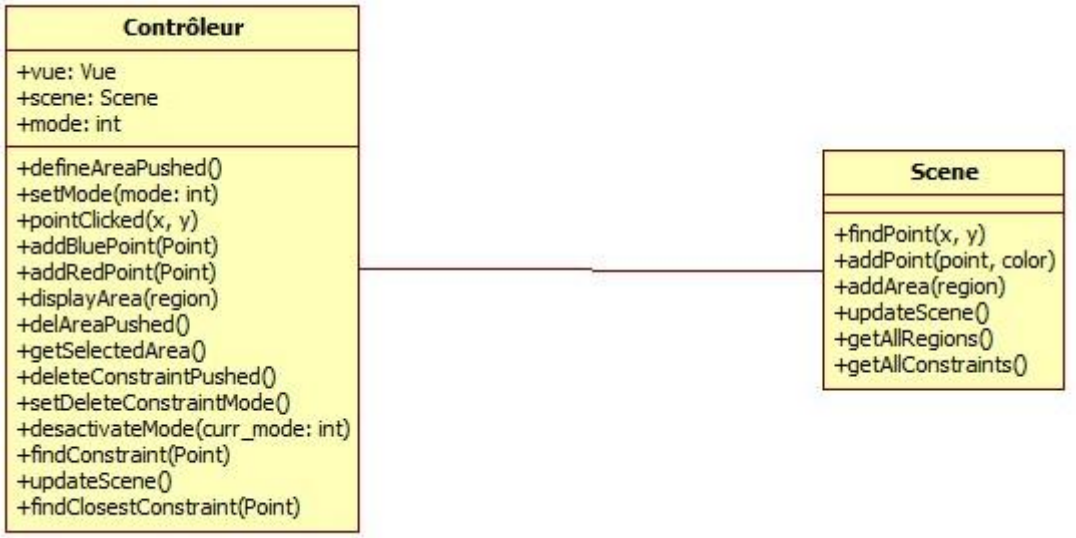
Cette fonctionnalité permet de trouver une contrainte, selon un point cliqué par l'utilisateur.

Pour cela, le système récupère toutes les contraintes de la scène, et cherche parmi chacun des deux points (origine, destination), celui dont les coordonnées sont les plus proches du point cliqué. Cette contrainte est donc retournée.

1) Diagramme de séquence détaillé



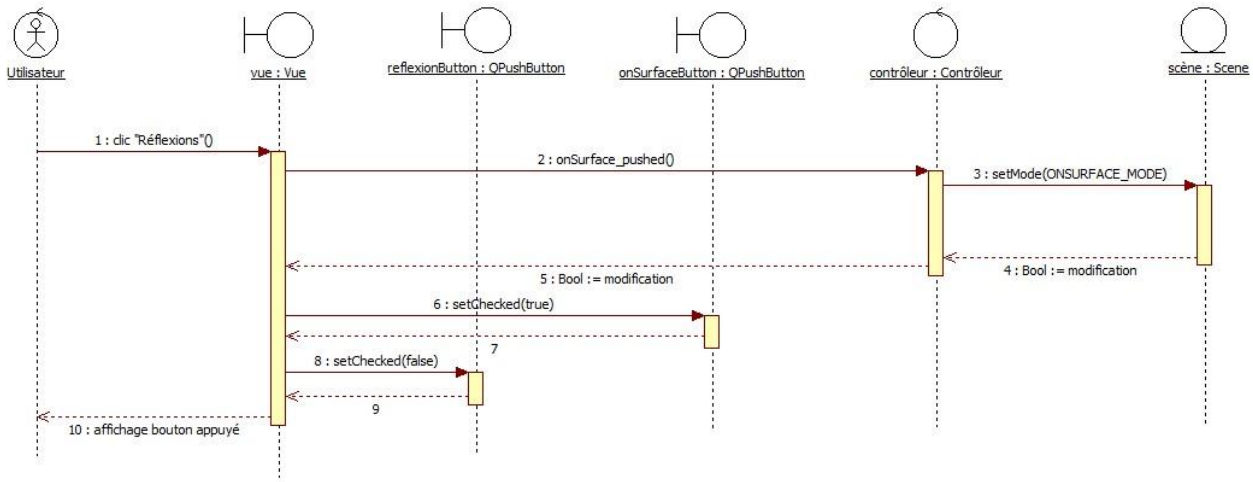
2) Diagramme de classes participantes



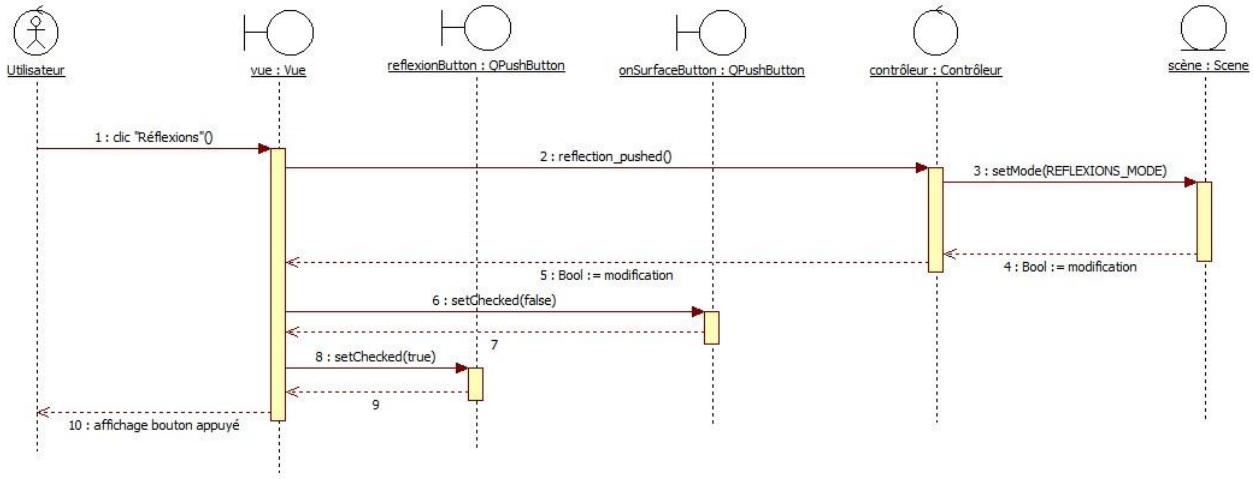
XI – Sélectionner le mode « Sur la surface »

Ces diagrammes montrent les interactions des différents objets de la vue qui appartiennent à Qt (les boutons qui définissent le mode) afin que leur aspect soit modifié.

Ce changement de mode est ensuite répercuté dans le modèle par une modification de variable.



XII – Sélectionner le mode « Réflexion »



XIII – Solveur

Nous avons retenu la librairie C++ Boost uBlas afin de nous aider à résoudre des systèmes dans certains calculs. En effet, nous aurons par exemple besoin de calculer le résultat d'une équation découlant d'un arg min afin de trouver les poids interpolés lorsque plusieurs contraintes ont été spécifiées par l'utilisateur.

Cette librairie permet de résoudre des systèmes linéaires, et dispose de fonctions complètes qui nous faciliterons les traitements mathématiques de notre projet.

De plus, cette librairie est une de mieux documentées, ce qui nous permettra un gain de temps dans son utilisation.

XIV – Tests Unitaires

Afin de vérifier que notre système fera ce que l'on souhaite, nous avons défini différents tests unitaires. En effet, ces tests permettront de tester les différentes fonctionnalités du logiciel tout au long du développement.

On s'assurera ainsi qu'une fonctionnalité ne régresse jamais au fur et à mesure des améliorations faites sur le système en faisant des tests réguliers.

De plus, le suivi constant des fonctionnalités par des tests permet d'éviter de détecter des comportements non souhaitables tardivement dans le processus de développement. Ainsi, on évite de trop longue phases de correction de code, et de recherche d'erreurs.

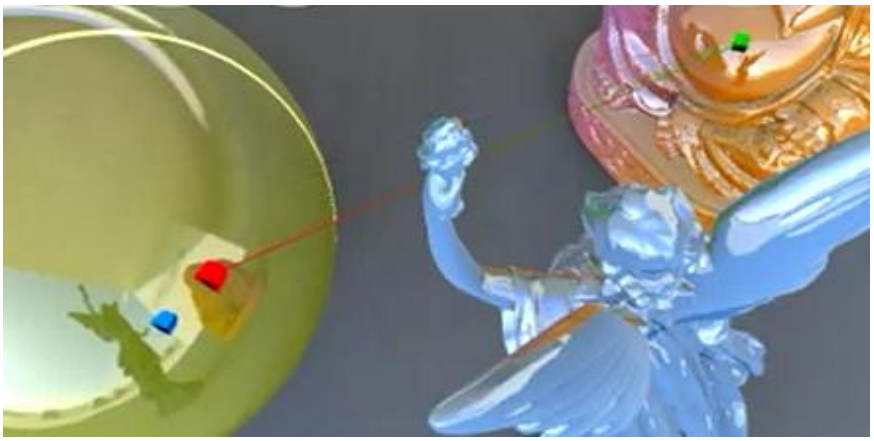
1) Tests sur les contraintes

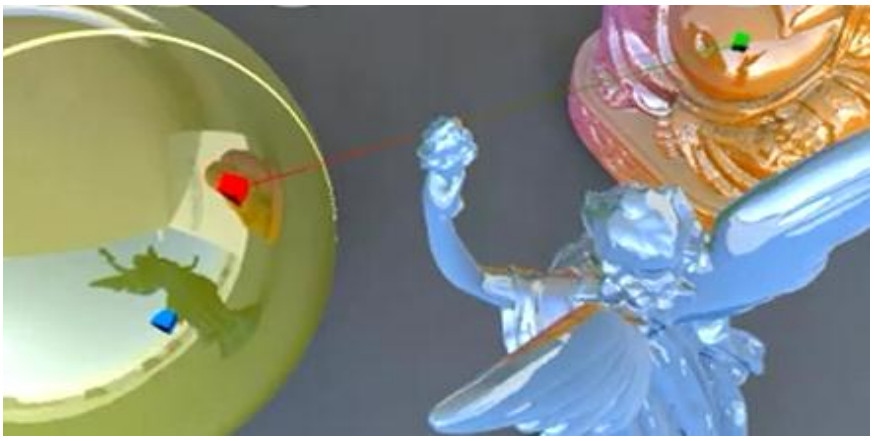
L'ajout d'une contrainte utilisera une fonctionnalité du moteur existant qui permet de trouver le point de la scène 3D correspondant au point cliqué sur la vue 2D. Les tests sur cette fonctionnalité consisteront donc à s'assurer que le point d'origine de la contrainte soit bien initialisé au point qui a été cliqué et dont les coordonnées sont retournées par la fonction existante du moteur.

La partie principale des tests consistera à vérifier la cohérence des modifications des points d'origine et de destination de la contrainte avec le résultat obtenu après l'application des algorithmes.

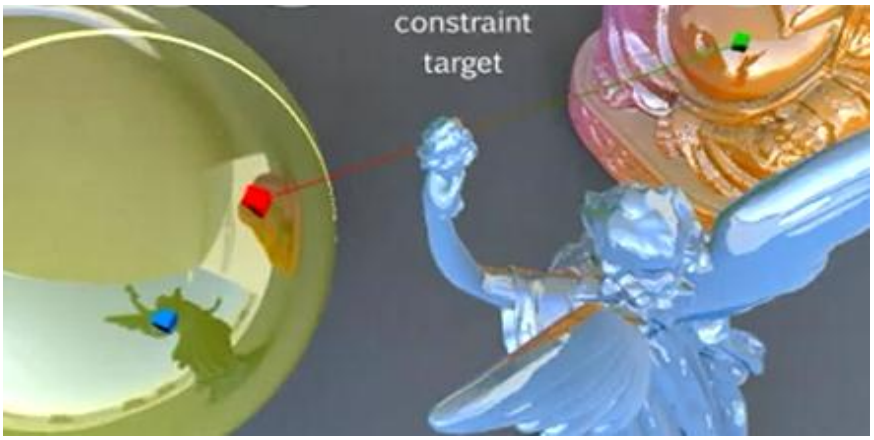
D'un point de vue général, il faudra vérifier qu'une scène qui a été modifiée avec divers traitements appliqués redevienne exactement comme la scène d'origine après que les différentes contraintes aient été supprimées.

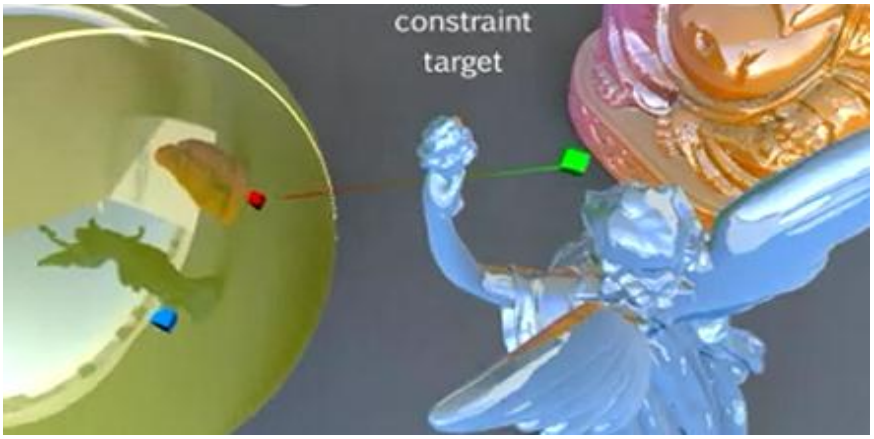
Plus spécifiquement, lors du déplacement du point d'origine d'une contrainte de réflexion (point rouge), le point correspondant sur la scène doit être déplacé exactement au même endroit (point vert). En effet, la destination de la contrainte ne changeant pas pendant le traitement, la source doit rester la même.





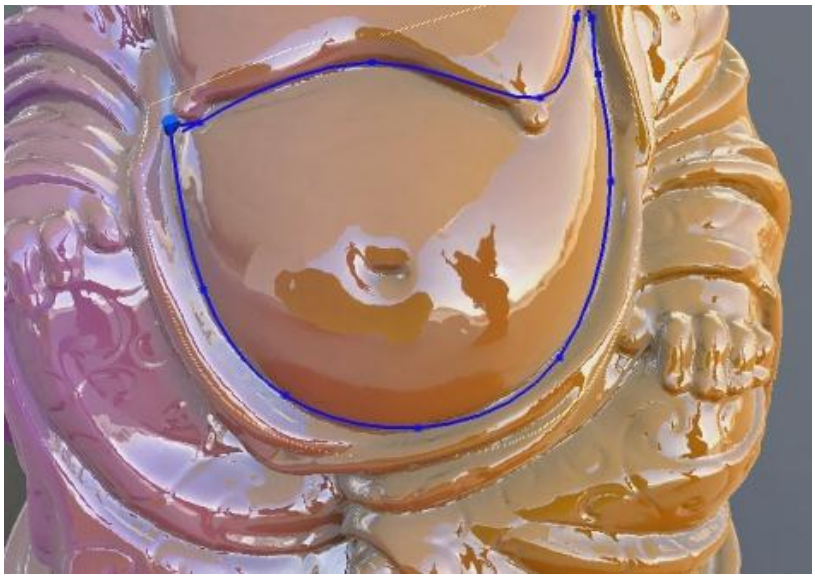
A contrario, lors du déplacement du point de destination de la contrainte de réflexion (point vert), le point sur lequel est la contrainte d'origine (point rouge) doit correspondre à ce déplacement.





2) Tests sur les régions d'intérêt

Les régions d'intérêt permettent de délimiter l'influence de la modification de contraintes. Ainsi, on doit vérifier que tous les points de la scène qui ne se trouvent dans aucune région d'intérêt ne soient jamais déplacés lors de différentes modifications de contraintes.



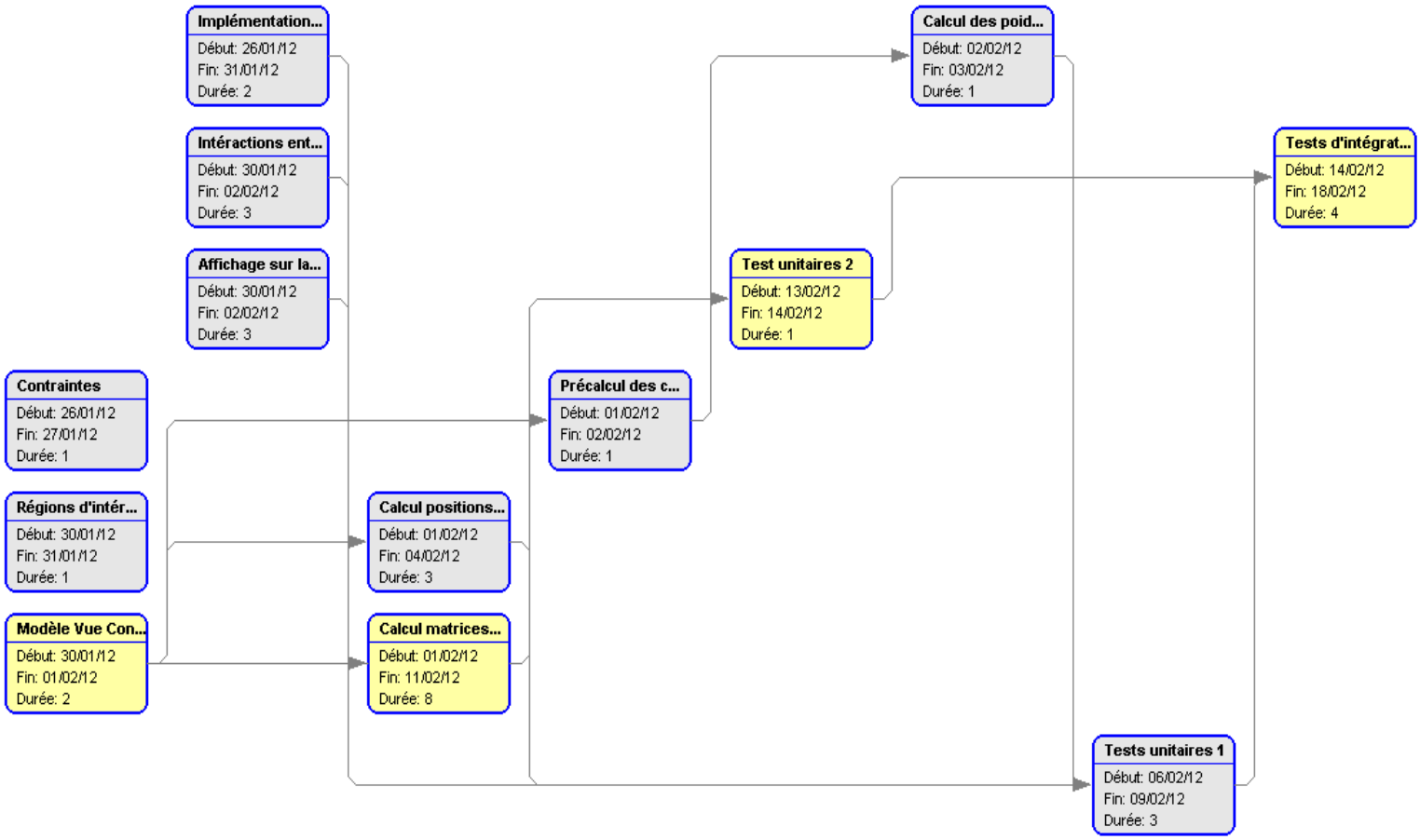
XV – Eléments fournis par le Client

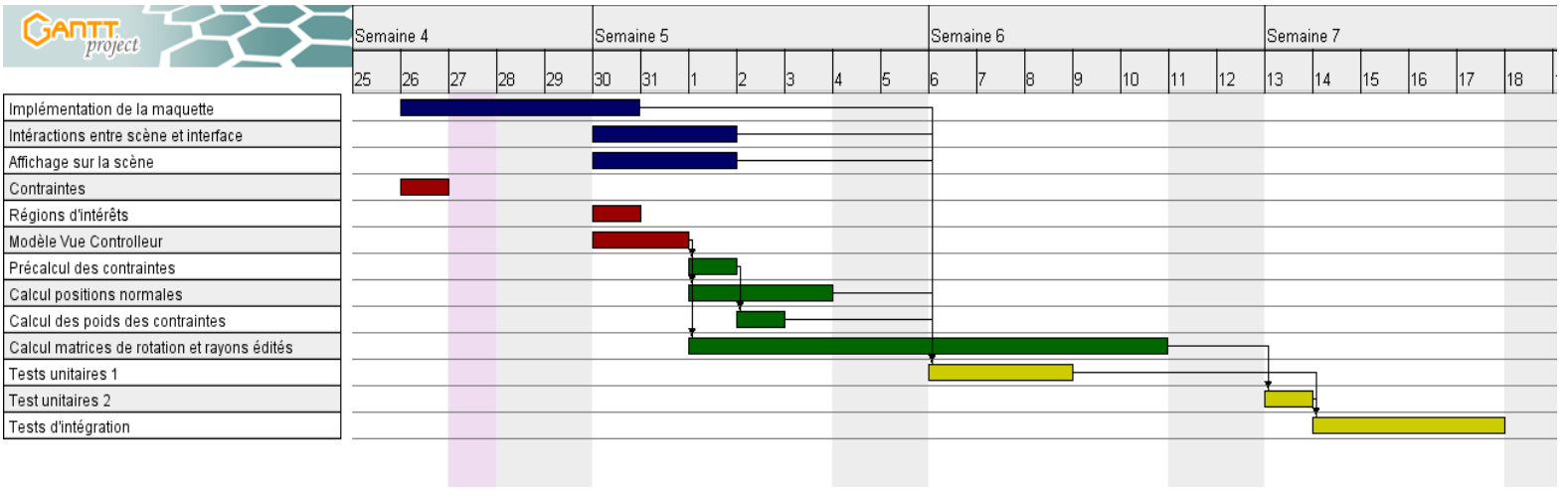
Le Client s'est engagé à nous fournir les fonctionnalités suivantes avant le début du mois de Février :

- Un moteur permettant le rendu d'une scène 3D gérant les réflexions.
- Une fonction *getNormal* qui permet de connaitre la normale aux pixels.
- *findPoint*(int x, int y) qui retourne un Point, qu'il appellera sans doute *UnProject*().

XVI – Planning

Dans le but de connaître précisement les dépendances entre les tâches de notre projet nous avons effectué des diagrammes de GANT et de PERT.




GANTT project
Semaine 4
Semaine 5
Semaine 6
Semaine 7
25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Implémentation de la maquette
Intéractions entre scène et interface
Affichage sur la scène
Contraintes
Régions d'intérêts
Modèle Vue Controlleur
Précalcul des contraintes
Calcul positions normales
Calcul des poids des contraintes
Calcul matrices de rotation et rayons édités
Tests unitaires 1
Test unitaires 2
Tests d'intégration