BOUM

Le logiciel CyBloon

Arnaud Bardoux Quentin Bolle Camille Brasseur Rémi Nollet
Ionelia Aniela Popescu Quentin Santos Maxime Savaro Bertrand Simon

de septembre à décembre 2012

Table des matières

Première partie

Les bases

1 Présentation

BOUM (pour Balloons On Ur Monitor) est un projet né de la volonté de réaliser un logiciel lié à la sculpture de ballons. Il a été commencé dans le cadre du master 1 d'informatique fondamentale de l'ENS de Lyon. Le logiciel créé a été nommé **CyBloon** et a pour but de sculpter des ballons virtuels sur un écran. Cela implique en pratique :

- la modélisation d'une sculpture par des classes en C++ (section 3, p.9)),
- la gestion d'un affichage 3D (section 7, p.16)
- la création d'un langage apte à représenter une sculpture étape par étape (section 5, p.11)
- la possibilité pour l'utilisateur de manipuler ce langage (section 9, p.25)),
- la possibilité pour l'utilisateur de créer via une interface cliquer-glisser (section 8, p.23)
- la création de tutoriels qui reprennent les étapes de construction d'une sculpture (section 10, p.31)).

1.1 Équipe

- Arnaud Bardoux,
- Quentin Bolle,
- Camille Brasseur,
- Rémi Nollet,
- Ionelia Aniela Popescu,
- Quentin Santos,
- Maxime Savaro et
- Bertrand Simon

prennent part à ce projet dont Camille Brasseur est l'initiatrice et la coordinatrice.

1.2 Vocabulaire

Ce projet utilise un objet bien particulier qui est le ballon à sculpter. Dans un souci de clarté, nous allons ici définir les différents termes qui le caractérisent.

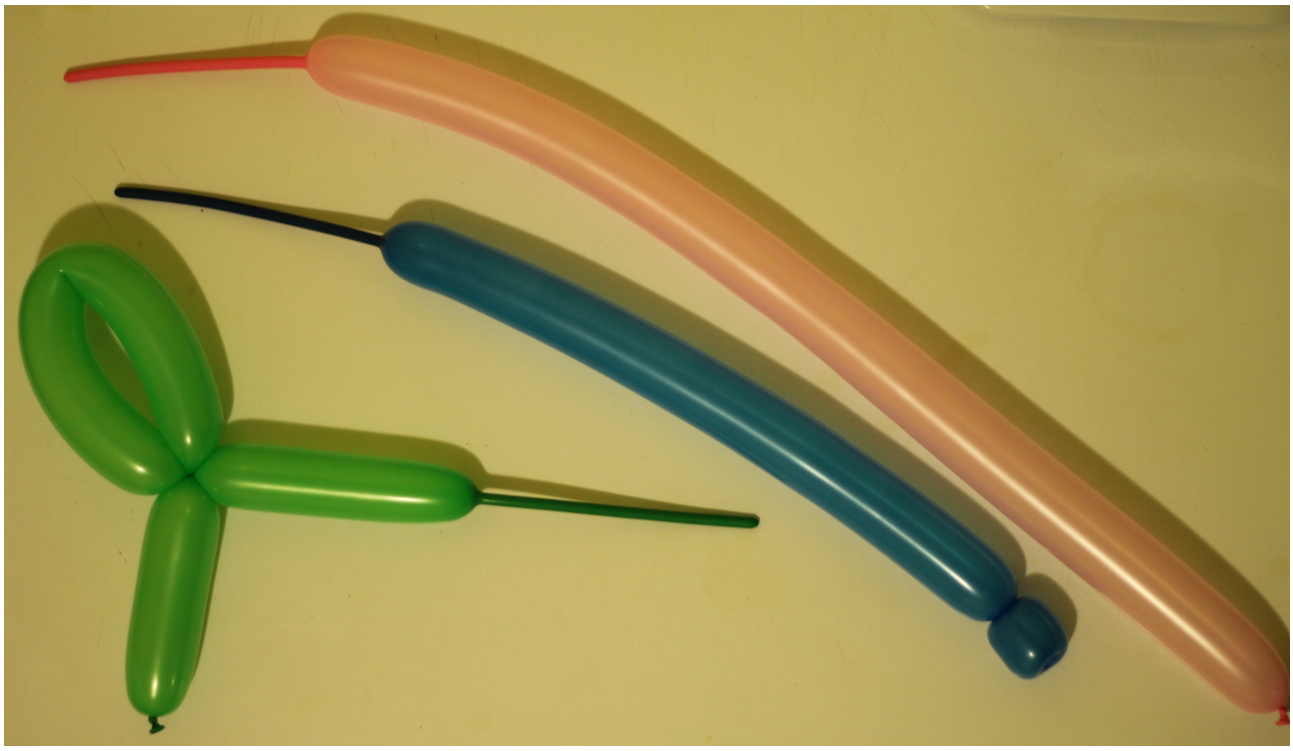


FIGURE 1 – Des ballons à sculpter

Un ballon, une fois sculpté, se compose de trois parties distinctes : le nœud du ballon, une longueur gonflée et une longueur non gonflée. La sculpture s'effectue par *torsions* successives du ballon pour le séparer en *bulles*. Le ballon se sculpte toujours du nœud jusqu'à la *réserve* non gonflée. A chaque torsion, de l'air est chassé de la partie initialement gonflée et remplit donc peu à peu la réserve non gonflée gardée au bout du ballon.

Pour réaliser une sculpture, le ballon doit donc être gonflé, au minimum, de la longueur totale des bulles qui seront utilisées et conserver une partie non gonflée au minimum équivalente aux nombre de torsions réalisées.

Dans toute la suite du document, nous appellerons *nœud* une torsion. Un *nœud* pourra également être le point de jonction entre une bulle gonflée et une bulle non gonflée (dans ce cas, il n'y a pas de torsion).Le nœud du ballon, la partie nouée qui retient l'air insufflé, sera appelée l'*embout* du ballon. Cet embout sera, dans la plupart des cas, considéré comme une bulle particulière : la seule bulle du ballon qui ait une longueur nulle et un seul nœud.

1.3 Travaux existants

En ce qui concerne le langage, les ballooneurs utilisent de manière individuelle des codes pour représenter leurs sculpture. Il est possible de trouver un certain nombre de possibilités envisagées sur `http://www.balloonhq.com/faq/notation.html` . Ces écritures sont peu utilisées mais pourront inspirer nos choix.

Des études ont déjà été menées sur les formes géométriques liées aux ballons. Ces recherches utilisent la notion de ballon mais ne la développent pas comme nous souhaitons le faire.

Enfin, et c'est ce qui nous intéresse le plus, quels logiciels existent actuellement ?

FIGURE 2 – Un exemple de réalisation possible sur `balloonmaster.com`

Nous en avons trouvés qui proposent des animations interactives (comme l'application pour iPhone « Balloon Animal »). Cependant, aucune nouvelle sculpture ne peut y être ajoutée par l'utilisateur.

Par ailleurs, le site `http://balloonmaster.com/` propose plusieurs rubriques autour des ballons. `http://balloonmaster.com/newSite/deco/decoHome.html`, notamment, permet en théorie de prévoir les couleurs d'une décoration classique comme un mur ou une colonne. Cependant, cette page semble mal fonctionner.

Sur ce site, il est également possible de télécharger un logiciel proposant de prévoir des décorations en ballons. Le principe est de récupérer la photographie d'un lieu et d'y ajouter des photos de sculptures proposées par l'auteur. Certaines couleurs peuvent être choisies et la taille et la disposition des sculptures est au choix de l'utilisateur. Aucune création de sculpture ne semble possible ; il ne s'agit que de manipuler des photos.

2 Objectifs

À terme, le but de BOUMest de développer un logiciel utilisable par n'importe quel individu capable de se servir d'une souris et d'un clavier. *A priori*, ce logiciel concerne avant tout les personnes pratiquant la sculpture sur ballon en amateur. Cependant, il s'adresse aussi aux néophytes complets. En ce qui concerne les sculpteurs de ballons professionnels, le logiciel nécessitera sans doute quelques améliorations avant de répondre à leurs attentes. Cela peut s'envisager pour l'avenir.

Plusieurs utilisations seront idéalement mises en place.

2.1 Logiciel

1. Un **affichage 3D** permettra à l'utilisateur de manipuler des sculptures sur ballons.
   Certaines seront originellement proposées par le logiciel, d'autres pourront être créées grâces aux outils décrits ci-après.

2. Au moyen d'un **langage** créé spécialement pour le projet, l'utilisateur pourra décrire ses propres sculptures de ballon.
   Le langage décrira les *étapes de construction* de la sculpture. Celle-ci sera ensuite affichable en 3D.

3. Un écran de **création interactive** sera également mis à disposition de l'utilisateur. Par ce biais, il pourra construire une sculpture en positionnant des bulles aux emplacements de son choix.
   Ce positionnement pourra se faire soit par *cliquer-glisser* soit par *identification* des nœuds reliant chaque bulle ajoutée, selon les contraintes et possibilités que les développeurs auront.

4. À partir de cette création, l'utilisateur pourra récupérer son objet 3D fini et demander un **calcul de mot** correspondant à sa sculpture, le *mot* étant un mot du langage décrit ci-avant.

La gestion du passage d'objet-ballon à mot-ballon sera entièrement géré par le logiciel.

5. Une fois un *mot-ballon* créé, et certifié correct par le logiciel, il sera possible d'obtenir un **didacticiel** montrant la sculpture correspondante étape par étape.
   Le didacticiel utilisera des affichages 3D de la sculpture aux différentes étapes, qu'il identifiera en lisant le mot bloc par bloc.

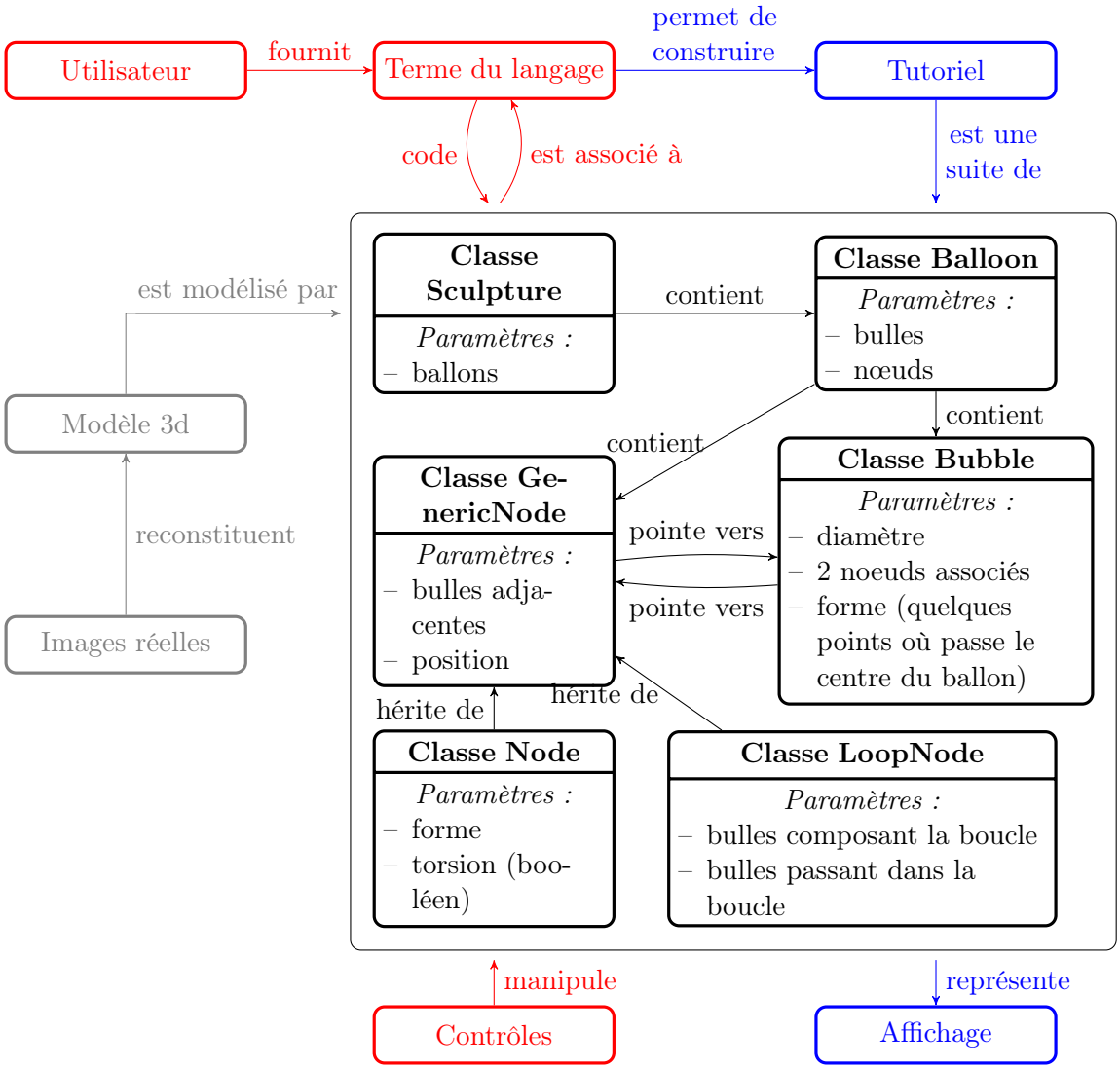


FIGURE 3 – Diagramme prévisionnel du logiciel

2.2 Calendrier

Pour mener le projet à bout, ce dernier a été décomposé en plusieurs modules de travail représentés dans la figure 4.

Chaque module est représenté sous la forme d'un rectangle de couleur (bleu pour ceux liés à la théorie, jaune pour ceux liés à l'implantation logicielle) et un identifiant correspondant à sa réalisation. Les dépendances sont indiqués au-dessus des modules, parenthésées.

En pratique, le problème du langage (qui était soit trop compliqué à manipuler pour un novice en informatique, soit trop vague pour être utilisé dans notre logiciel) nous a forcé à modifier nos prévisions.

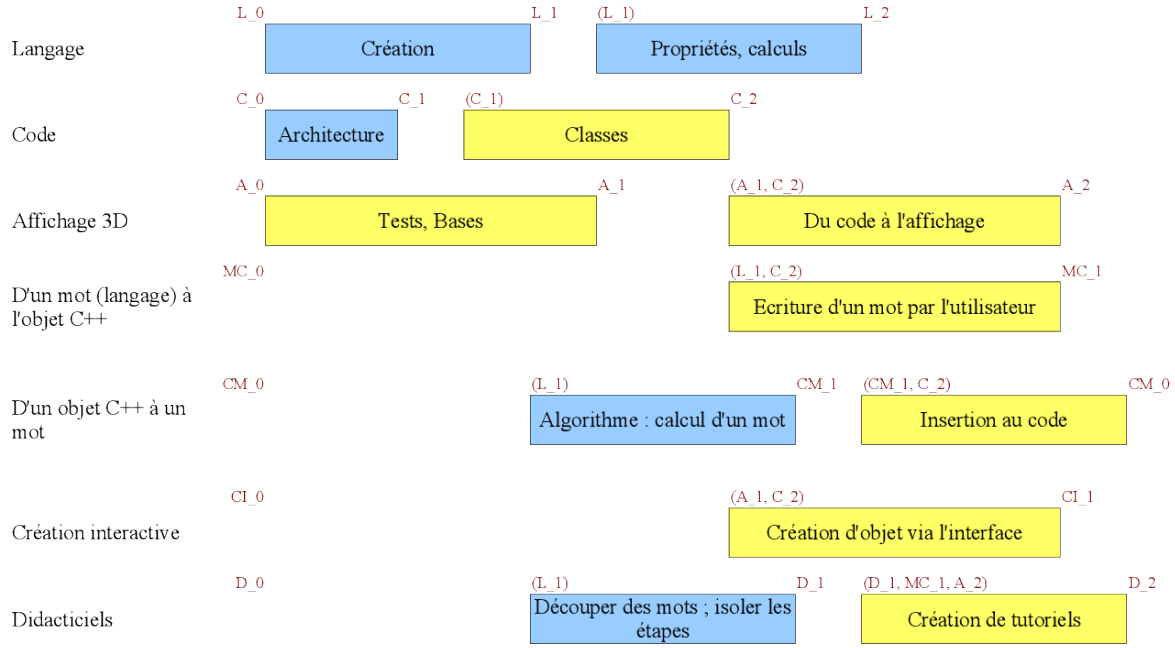


FIGURE 4 – Découpage des objectifs du projet

Un module « Création de formulaires » a du être ajouté à ceux ceux initialement prévu. Il remplace, en beaucoup plus complexe, le module « Écriture d'un mot par l'utilisateur » que nous avions imaginé à l'origine.

Des modules ont également dû être laissés de côté. Nous n'avons ainsi pas fait de calculs ni cherché à démontré des propriétés de notre langage (qui s'avère très proche d'un langage de programmation). De même, l'algorithme de calcul d'un mot n'a pas été effectivement établi.

2.3 Communication

Quentin SANTOS,
Camille BRASSEUR,
Bertrand SIMON

Nous disposons d'un site (`www.sinon.org/ballons.html`) où les informations sur le projet sont régulièrement mises à jour. Pour s'adresser à un public plus spécifiquement concerné, nous postons aussi sur un forum (`balloonforum-fr.com/`). Nous y avons présenté le projet et reçu des avis et encouragements variés. Plusieurs personnes sont volontaires et très motivées pour tester le logiciel dès qu'une version en sera disponible. A ce sujet, nous avons actuellement un souci pour la compilation sous windows. Le logiciel n'est disponible que sous linux et ne peut donc pas être tester à grande échelle.

Deuxième partie

Le cœur (non visible) du logiciel

3 Représentation orientée objet d'une sculpture de ballons

Bertrand Simon,
Arnaud Bardoux

Nous allons dans cette partie exposer les choix de représentation d'une sculpture de ballons dans le programme. Nous utilisons la programmation orientée objet, car l'utilisation des classes simplifie l'implémentation. Voici les principaux éléments pour comprendre la hiérarchie utilisée.

Comme représenté dans la figure 5, une sculpture est transcrite dans le programme par la classe `Sculpture`. Cette classe contient des `Node` et des `Balloon`. Un `Balloon` est un ensemble de `Bubble` avec certains paramètres (sa couleur, son diamètre et sa taille).

Un `Node` est un point de l'espace, et pointe vers les `Bubble` qui y passent. Il peut subir une torsion ou non (par exemple, quand un `Node` lie deux `Bubble` dont l'une est gonflée, et l'autre ne l'est pas). S'il n'en subit pas, l'affichage devra alors lier les deux diamètres des deux bulles de manière continue, et non simuler une torsion.

Une `Bubble` a plusieurs paramètres :

- son `Balloon` parent
- sa taille
- les deux `Node` auxquels elle est liée
- sa forme (droite, courbée ou « appletwist »)
- un booléen qui indique si elle est gonflée ou non
- si la bulle est courbe, des éléments permettant de calculer la trajectoire de la bulle. Il s'agit des deux vecteurs d'orientations d'entrée et de sortie, grâce auxquels, avec la taille et les positions des `Node`, on peut calculer deux points de contrôle de Bézier, qui suffisent à calculer la trajectoire de la bulle. L'algorithme est précisé dans la section 7.4.

Nous avons aussi crée la classe `LoopNode`, qui représente l'endroit où des bulles passent dans une boucle. Cette classe hérite de `Node`, et pointe en plus vers les bulles composant la boucle, et indique quelle longueur de chaque bulle doit passer à travers la boucle. Pour l'instant, on peut construire cette classe depuis un fichier texte, mais elle n'a pas de conséquence sur l'affichage. C'est-à-dire que l'on ne force pas les bulles considérées à passer dans la boucle.

Nous avons crée des fonctions qui permettent de déplacer la `Sculpture` ou un `Node` en mettant à jour les points de contrôle des bulles concernées, ce qui permet de manipuler facilement cette structure de données, car un seul appel permet de déplacer proprement ce que l'on veut.

Toutes ces classes possèdent en outre un nom, donné par l'utilisateur ou défini par défaut, ainsi qu'un identifiant unique, pour pouvoir les distinguer, par exemple dans le cliquer-glisser. La hiérarchie a été conçue dans le but d'être construite facilement (par le parser et par l'interface cliquer-glisser) et d'être utilisable efficacement (par les différentes parties du programme). Cependant, pour certaines parties du programme (le parser et les formulaires), nous avons eu besoin de manipuler une hiérarchie analogue, mais avec des paramètres différents, qui permettent par exemple de vérifier que la sculpture que l'on est en train de construire est cohérente. Pour ce faire, ces deux parties utilisent donc des hiérarchies similaires à celle-ci mais indépendantes, afin de séparer clairement les paramètres utiles à chaque point du programme.

Le point sur lequel nous avons hésité est la représentation des boucles. Nous avons d'abord pensé à créer une classe virtuelle à partir de laquelle héritent deux classes : les nœuds « normaux »et les boucles. Puis nous nous sommes aperçus que les boucles peuvent en fait être vues comme des nœuds auxquels on ajoute une liste de bulles. C'est pourquoi nous avons adopté ce schéma-ci.

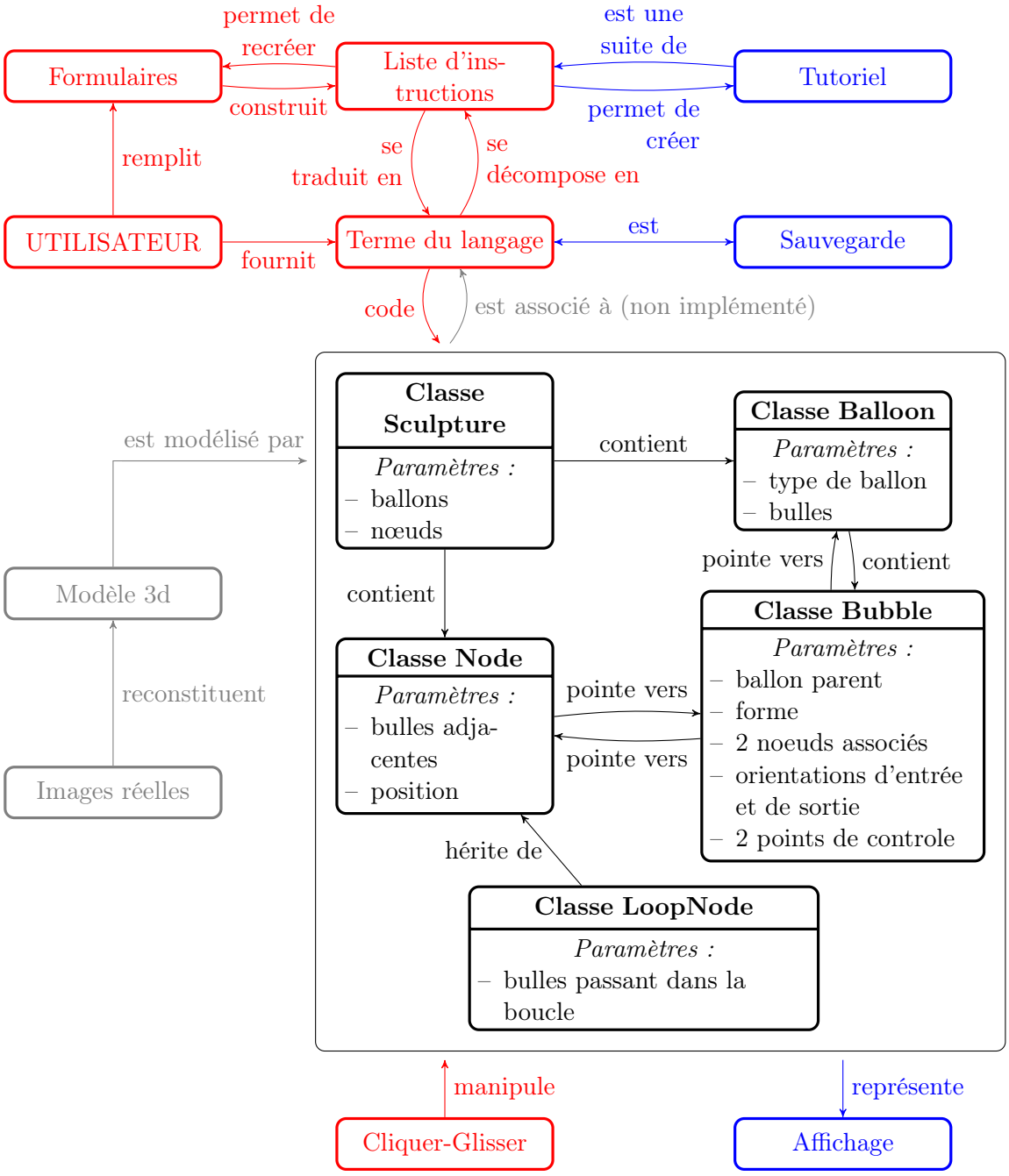


FIGURE 5 – Schéma non exhaustif résumant le principe du programme

Légende :

- **noir** : l'**UML** décrit la sculpture comme un ensemble de ballons, eux-même correspondant à une succession de nœuds et de bulles. Un nœud peut ou non correspondre à une boucle dans laquelle d'autres bulles passent (d'où l'héritage). Une bulle sera décrite par les nœuds extrêmes et la trajectoire qu'elle décrit ;
- **rouge** : les **entrées** sont fournies par l'utilisateur, soit sous la forme d'un terme représentant une sculpture à représenter, soit par des formulaires qui permettent d'indiquer étape par étape la sculpture, soit via une interface cliquer-glisser qui permet de manipuler visuellement les ballons. ;
- **bleu** : la **sortie** peut être une simple image affichée à l'écran ou bien un tutoriel, qui se présente comme une série d'images à différentes étapes de la construction accompagnée d'explications. Concrètement, le tutoriel est en fait composé de plusieurs suites d'instructions, qui passent dans les différentes parties du programme pour être finalement affichées. On peut aussi sortir directement le terme du langage, pour sauvegarder la sculpture construite.
- **gris** : Ce qui n'existe pas à l'heure actuelle est un moyen de sauvegarder la classe sculpture en un terme du langage. Il serait aussi possible de partir d'images réelles de ballons afin de créer un modèle 3d qui permettrait de construire une classe sculpture, mais ceci reste hypothétique.

4 Parser

Bertrand Simon

Le parser est la seule partie du programme qui prend en entrée un mot du langage. Comme indiqué sur la figure 5, son rôle est essentiellement de créer une classe `Sculpture` complète, prête à être utilisée par les autres éléments du programme.

Il doit donc pour cela effectuer plusieurs vérifications, pour s'assurer que le mot est correct et ne provoquera pas d'erreur des autres parties du programme. Par exemple, une bulle courbe ne peut avoir ses orientations d'entrée et de sortie identique. En effet, on suppose que les bulles ont une courbure régulière, donc deux orientations identiques donneront une bulle droite. Ainsi, la sortie du parser donne toujours une sculpture exploitable, car toutes les vérifications nécessaires sont implémentées dans cette partie.

L'implémentation du parser utilise une hiérarchie de classes analogue à celle du corps du programme. Comme cette partie nécessite des paramètres pour vérifier que la sculpture qui est en train d'être construite est cohérente, nous avons choisi de séparer la hiérarchie utilisée par le parser, et celle utilisée par le corps du programme, par souci de clarté (comme indiqué dans la section 3).

Le parser prend en entrée un flux de caractères. Il peut donc être utilisé à partir d'un fichier, de données entrées dans l'interface, ou d'une chaîne en mémoire dans le programme. Son fonctionnement repose sur la fonction `Next_Token()`, qui renvoie le prochain « token ». Il s'agit du prochain délimiteur (qui est par exemple l'un des caractères : « `(){}<>;,%#` »), ou bien de la prochaine chaîne de caractères alphanumérique. Ainsi, on peut facilement connaître le prochain élément et appeler au besoin la fonction qui parse un ballon, une bulle...

Une fois que le flux est terminé et que la sculpture construite est cohérente, il reste à traduire la sculpture construite dans l'architecture du parser en une structure utilisée par le reste du programme, celle de la figure 5.

Le parser doit aussi s'adapter pour construire les objets nécessaires aux différentes parties du programme. Par exemple, le formulaire et le tutoriel ont besoin d'une liste d'instructions. C'est donc le parser qui construit cette liste à partir du mot du langage, afin que chaque partie du programme puisse y avoir accès. Nous avons également crée une fonction qui permet de renvoyer les erreurs rencontrées afin de les afficher dans la fenêtre du programme.

Cette partie du programme doit évidemment suivre les évolutions du langage, mais aussi des données dont les autres parties du programmes ont besoin. En effet, le parser doit prendre en entrée le langage, et calculer toutes les informations utiles.

5 Langage

Camille Brasseur,
Maxime Savaro,
Bertrand Simon,
Quentin Bolle

5.1 Objectifs

Pour être utilisable, le langage doit remplir un critère essentiel. **Un mot doit représenter une et une seule sculpture de ballon.** En revanche, une même sculpture pourra être représentée par plusieurs mots distincts. Ils correspondront chacun à une manière de réaliser la sculpture.

Il va donc être possible de définir, au sein de notre langage, une relation d'*équivalence* entre plusieurs sculptures. Deux sculptures équivalentes auront le même affichage 3D mais seront caractérisées par des mots, et donc des tutoriels, distincts.

Ce critère de base peut être décomposé en plusieurs critères secondaires, plus significatifs pour sa création. Au sein du langage, il est nécessaire d'avoir :

– une identification de chaque **ballon** utilisé si la sculpture en utilise plusieurs ;

– une identification des **nœuds** (plusieurs bulles sont ainsi rattachées) ;
– la donnée du **nœud d'entrée** et du **nœud de sortie** d'une bulle ;
– une détermination de la **longueur** de chaque bulle (qui peut être arrondie au pouce près) ;
– une information sur la **position** de la bulle, soit au sein de la sculpture (horizontale, verticale, selon un axe donné...), soit par sa position par rapport à chacun de ses nœuds ;
– une identification des **boucles**, qu'elles soient composées d'une bulle ou de plusieurs ;
– une information sur l'éventuel **passage** d'une partie de la sculpture à travers une boucle donnée.

Ces critères sont nécessaires mais peuvent ne pas être suffisants. Par ailleurs, certains pourront être définis implicitement dans le langage. L'ordre dans lequel ils seront utilisés peut (et doit sans doute) différer entièrement de l'ordre dans lequel ils ont été énoncés.

Au sein du langage créé, il doit être possible de démontrer les propriétés suivantes :
– Si t_n représente la longueur de ballon utilisée pour un nœud, n_n le nombre de nœuds dans un ballon B, t_i la longueur de la i^{ieme} bulle de ce ballon, et t_B la taille totale de ce ballon, alors on doit avoir $(\sum_i t_i) + n_n * t_n \leq t_B$.
– Pour une sculpture ou une portion de sculpture n'impliquant qu'un ballon, un nombre *pair* de bulles se croise en un même nœud. L'embout compte comme une bulle.
– En approfondissant ces deux propositions, il doit être possible de calculer le nombre minimum de ballons utilisés sur une sculpture unicolore.

D'autres propriétés pourront être démontrées si le langage s'y prête.

5.2 Syntaxe choisie

Devant la difficulté de réaliser un langage à la fois complet et accessible à l'utilisateur, nous avons décidé de passer par une interface pour créer un mot du langage. L'utilisateur pourra donc créer des mots, et donc les étapes de sculpture associées, sans jamais connaître le langage.

La description d'une sculpture commence par la définition des ballons utilisés. Pour cela, on utilise le mot-clef `balloon` suivi du nom du ballon et, entre parenthèses, sa taille et sa couleur. A ma fin des définitions, un `%` clôt cette première partie. On décrit chaque ballon par une chaîne bulle-noeud-bulle ... dans des accolades précédées du nom du ballon sculpté. Il est possible de définir une boucle une fois que toutes les bulles qui en font partie ont été formées. Pour cela, on utilise le mot-clef `loop` suivi du nom de la boucle puis des bulles qui la composent entre parenthèses. On peut ensuite passer un ensemble de bulles à travers celle-ci, en précisant la longueur de bulle passée :
`add (bulle5,2)(bulle6,3) > boucle1`

Une bulle est codée par deux parenthèses entre lesquelles plusieurs informations peuvent être indiquées, séparées par des virgules. Une bulle non gonflée, ou le bout du ballon, sera codée de même mais entre chevrons : $<$ $>$.

Exemple : `NOEUD1 (bulle1) (bulle2) NOEUD1 <bulle3>`

Au sein d'une bulle, une seule donnée est obligatoire : la longueur. On peut définir d'autres paramètres pour plus de précision :
– **LG** la longueur de la bulle
– **ID** le nom de la bulle (utile pour les boucles)
– **IO** l'orientation d'« entrée »
– **OO** l'orientation de « sortie »
– **SH** « forme »de la bulle

Les différents paramètres doivent être séparés par une virgule.

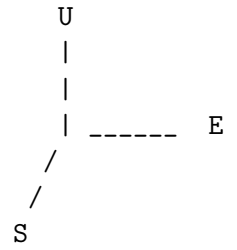
Exemples : `(LG : 1, OO : S, ID : ear, SH : BEND, IO : W)`

À noter qu'il n'est pas utile de donner toutes ces informations ; par exemple, dans le cas d'une bulle droite, une seule orientation suffit : `(LG : 1, SH : STRAIGHT, IO : N)` est probablement suffisant.

On compte la **longueur** en pouces.

Le **nom** peut être n'importe quelle suite de lettres et de chiffres sans espace. Pour une bulle nommée nez, on écrira : `ID:Nez`

On définit l'**orientation** par les lettres U (up), D (down), N (north), S (south), E (east) et W (west). Les orientations sont données dans le sens du ballon (embout vers fin). Pour l'orientation dite « d'entrée », il s'agit de dire quelle direction prend la bulle à partir du nœud qui la commence. Pour l'orientation dite « de sortie », il s'agit de dire quelle direction elle prend pour rejoindre le nœud.



La **forme** sera explicitée par un mot-clef. Pour le moment, les mots-clef utilisables sont :
- STRAIGHT (pour une bulle droite)
- BEND (pour une bulle courbe)
- @ (pour un apple-twist)

Ces mots clefs vont sans doute évoluer et d'autres compléteront la liste.

5.3 Exemples

5.3.1 Le lapin :

```
balloon ballon 260 BLUE;

%

ballon{
(LG:2,IO:N,OO:N) A (LG:6,IO:UNE,OO:UNW, SH:BEND) (LG:6,IO:DSW,OO:DSE, SH : BEND) A
                    # embout; museau; oreille gauche; oreille droite
(LG:1,IO:D,OO:D) B (LG:4,IO:DSE,ID:AVD,OO:DSW, SH:BEND) (LG:4,IO:UNW,ID:AVG,OO:UNE, SH:BEND) B
                    # cou et pattes avant
(LG:3,IO:DN,OO:DN) C (LG:10,IO:SE,ID:patte,OO:NE, SH:BEND)C
                    # corps et pattes arrieres en boucle, avec ID pour la suite
(LG:1,IO:N,OO:N)  # queue
}

Loop corps: (patte);
Add (AVD,1)(AVG,1) > corps;
;
(everything after the semi colon is ignored)
```

5.3.2 La libellule :

```
  balloon corps  260 (0 128 0 200) ;
  balloon yeux   260 (127 127 127 255) ;
  balloon aileAv 260 WHITE ;
  balloon aileAr 260 WHITE ;

  %

  yeux{ O ( LG : 2 , ID : droite , SH : BEND , IO : UUN , OO : UUS ) O
```

```
  ( LG : 2 , ID : gauche , SH : BEND , IO : UUS , OO : UUN ) O
  ( LG : 2 ,                           IO : DDE , OO : DDE ) E
}

corps{ A ( LG : 1 , SH : BEND    , IO : NNU , OO : NND           ) A
  ( LG : 8 , SH : BEND    , IO : ND  , OO : NU , ID : ann1 ) A
  ( LG : 2 , SH : STRAIGHT, IO : W   , OO : W              ) B
  ( LG : 1 , SH : BEND    , IO : NNU , OO : NND            ) B
  ( LG : 8 , SH : BEND    , IO : ND  , OO : NU , ID : ann2 ) B
  ( LG : 8 , SH : BEND    , IO : ND  , OO : NU , ID : ann3 ) B
  ( LG : 4 , SH : BEND    , IO : DW  , OO : DE , ID : tete ) E
  ( LG : 22, SH : STRAIGHT, IO : E   , OO : E  , ID : longbody )
}

loop boucle1 : (ann3);
loop boucle2 : (ann2);
loop boucle3 : (ann1);

add (longbody,0) > boucle1;
add (longbody,1) > boucle2;
add (longbody,2) > boucle3;


aileAv { B ( LG : 28 , SH : BEND , IO : NW , OO : SW) B
  ( LG : 28 , SH : BEND , IO : SW , OO : NW) B
  ( LG : 1  , IO : UE   , OO : UE)
}

aileAr { A ( LG : 28 , SH : BEND , IO : UNW , OO : DSW) A
  ( LG : 28 , SH : BEND , IO : USW , OO : DNW) A
  ( LG : 1  ,             IO : UE  , OO : UE)
}
;
```

Troisième partie

L'apparence : le logiciel tel que l'utilisateur le manipule

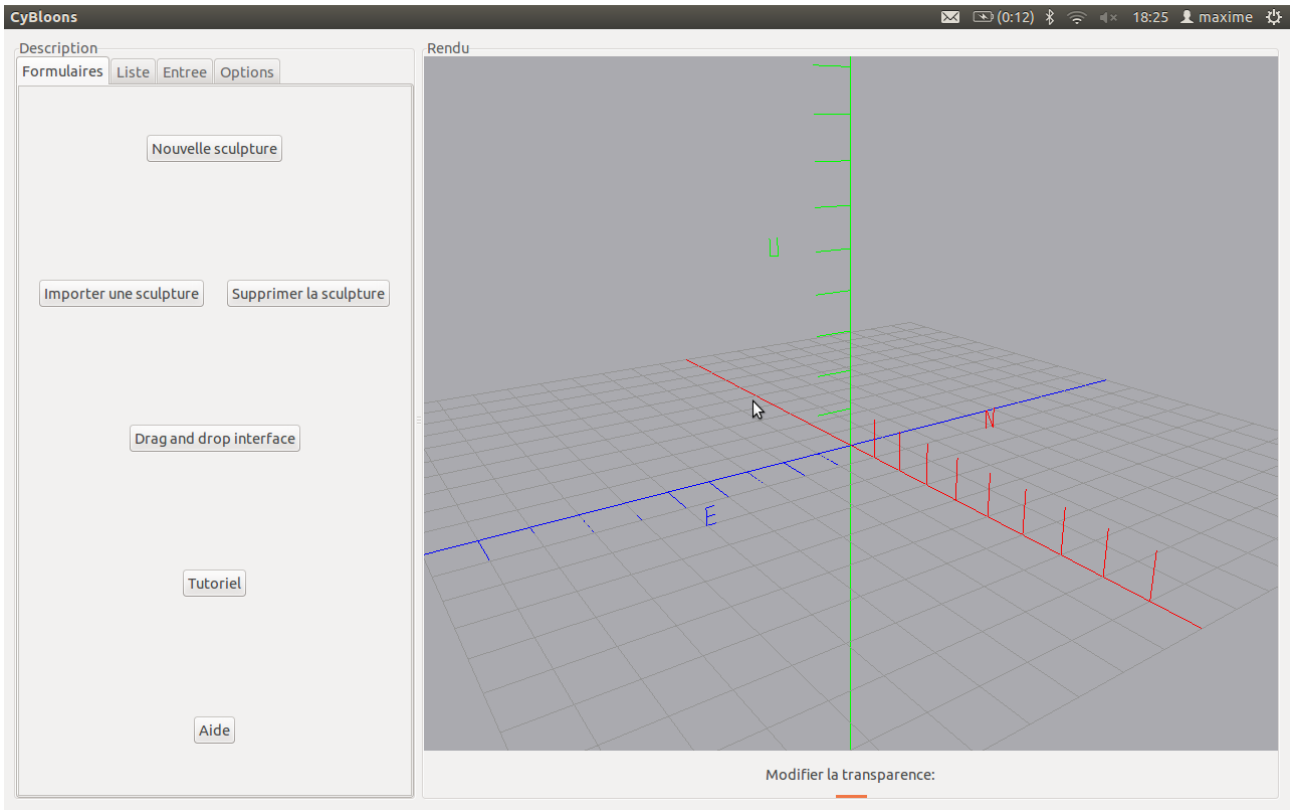


FIGURE 6 – Le logiciel à son ouverture

6 Interface du logiciel

Quentin SANTOS

6.1 Organisation générale

Le dessin des sculptures en ballons et l'interface utilisateur ont soulevé plusieurs problématiques. Les conditions que nous nous sommes fixées au départ étaient :

- avoir un module de dessin simple qui ne dépende pas de la partie interface utilisateur (i.e. GLFW, Glut, GTK) ;
- pouvoir changer facilement la sculpture dessinée à l'écran ;
- pouvoir dessiner la figure avec un code relativement simple.

6.2 Indépendance à l'interface utilisateur

Le premier point a fixé la façon dont serait organisé le code ; en trois couches :

- l'interface utilisateur ;
- le contexte OpenGL ;
- les instructions de dessin.

La dernière partie est faite pour ne regrouper que les informations permettant de dessiner un objet donné. La seconde est plutôt administrative : elle gère l'initialisation d'OpenGL et prépare la scène de

rendu. La dernière s'occupe de l'initialisation de l'interface utilisateur, des principaux événements, de la communication avec le contexte OpenGL et permet de faire le raccord avec le reste du programme.

La différence entre les parties gérant l'interface utilisateur, d'une part, et le contexte OpenGL, d'autre part, a une importance plus forte qu'entre les deux autres couches. En effet, les deux dernières parties n'utilisent qu'OpenGL (et éventuellement Glu), mais la première peut utiliser une bibliothèque haut-niveau quelconque (Glut, Gtk, Qt, etc).

En suivant cette organisation, nous avons d'abord commencé par utiliser GLFW pour sa simplicité, avant de basculer vers une bibliothèque plus puissante : Glut. Enfin, nous avons construit une véritable interface utilisateur utilisant GTK avec boutons, champs de texte, formulaires, etc. Lors de la mise en place de cette dernière version, nous avons pu basculer entre GTK — pour tester la nouvelle interface — et Glut — pour tester les autres partie de l'application.

6.3 Raccord avec le reste du programme

Un point fondamental des projets collaboratifs et que chacun puisse utiliser le code d'un autre développeur sans avoir à lire tout son code. Pour répondre à cette exigence, nous avons mis en place des interfaces (fichiers d'en-tête) intuitives et faciles à utiliser qui permettrait aux autre développeur de pouvoir utiliser le système de dessin sans être embêté par des paramètres spécifiques ou des ordres d'appel.

Afin d'y parvenir, nous avons réduit autant que possible le nombre d'appel de fonction qu'un programmeur doit effectuer pour faire fonctionner le tout. Ainsi, l'initialisation de l'interface utilisateur se fait en un appel ; le dessin d'une structure se fait en un autre appel ; et entrer dans la boucle principale ne demande qu'un appel supplémentaire.

7 Affichage 3D

Quentin Santos,
Ionelia Aniela Popescu,
Maxime Savaro

7.1 Objectifs

L'affichage du ballon doit permettre d'appréhender la sculpture le mieux possible. Plusieurs outils on donc été mis en place :

- zoom ;
- rotations de l'image ;
- déplacements autour de l'image.

L'utilisateur peut également choisir la couleur de fond de cette fenêtre et choisir d'afficher ou non les axes et le plan permettant de repérer l'orientation (figures 7 à 10). Il peut également modifier la transparence des ballons (figure 11). Nous pouvons envisager dans l'avenir de représenter leur texture et leur couleur de la manière la plus réaliste possible.

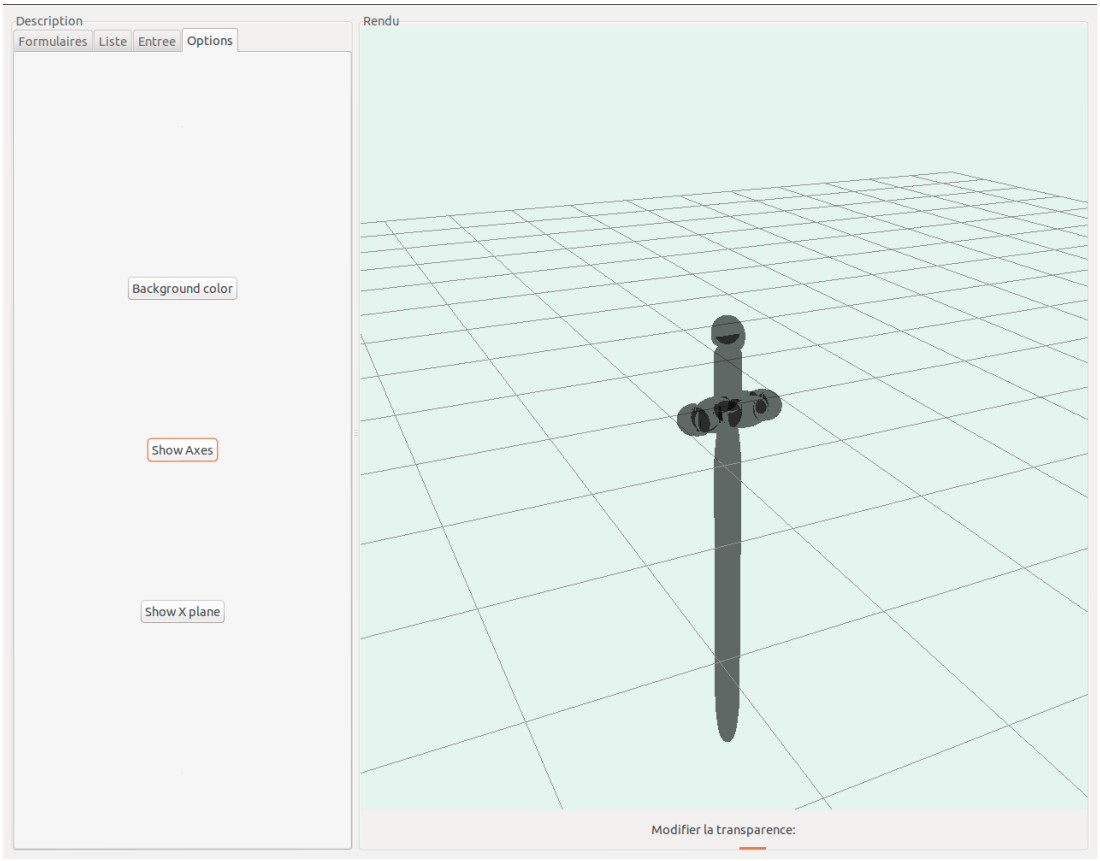


FIGURE 7 – Options d'affichage : sans axe

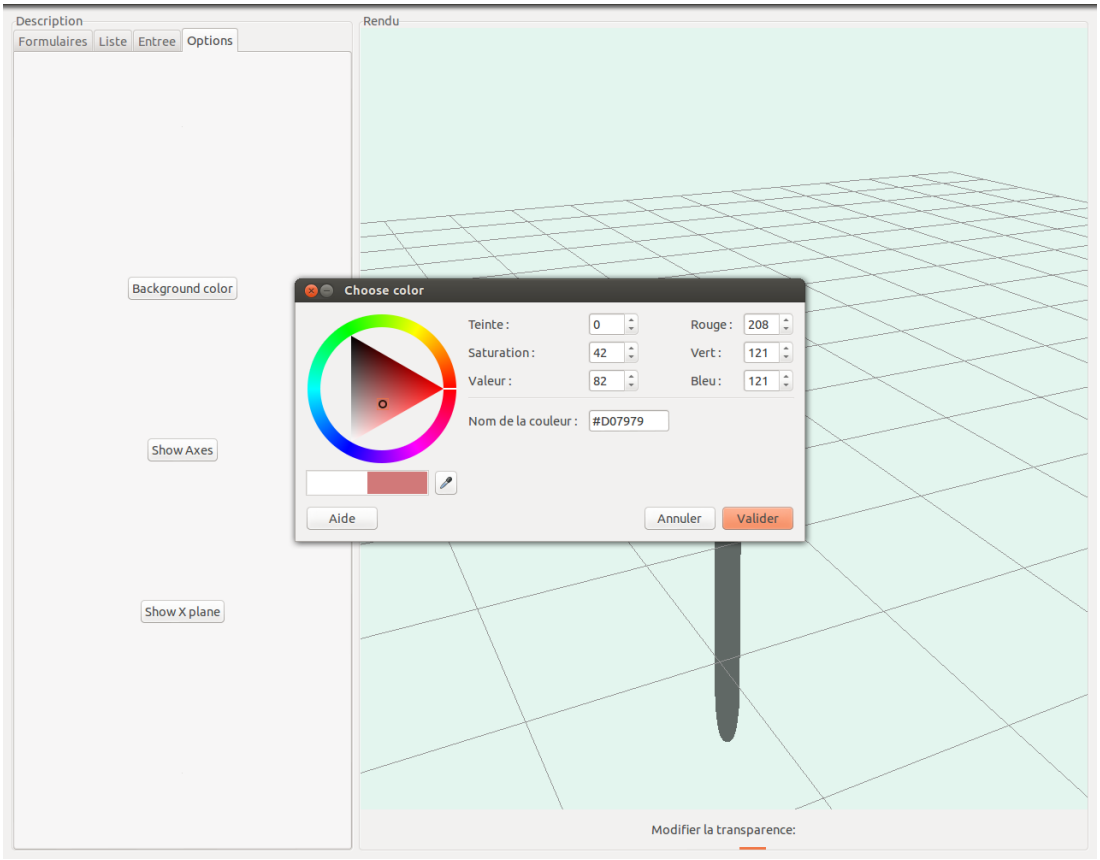


FIGURE 8 – Options d'affichage : choix de couleur

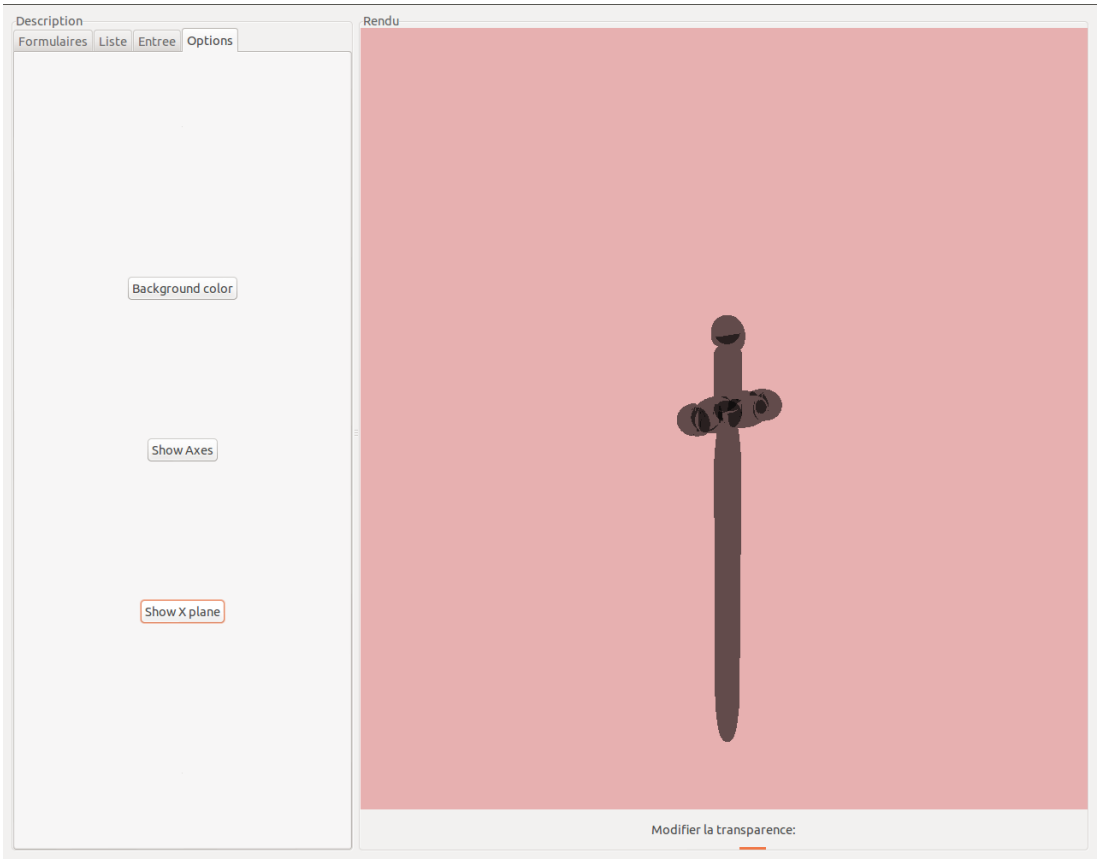


FIGURE 9 – Options d'affichage : sans plan horizontal

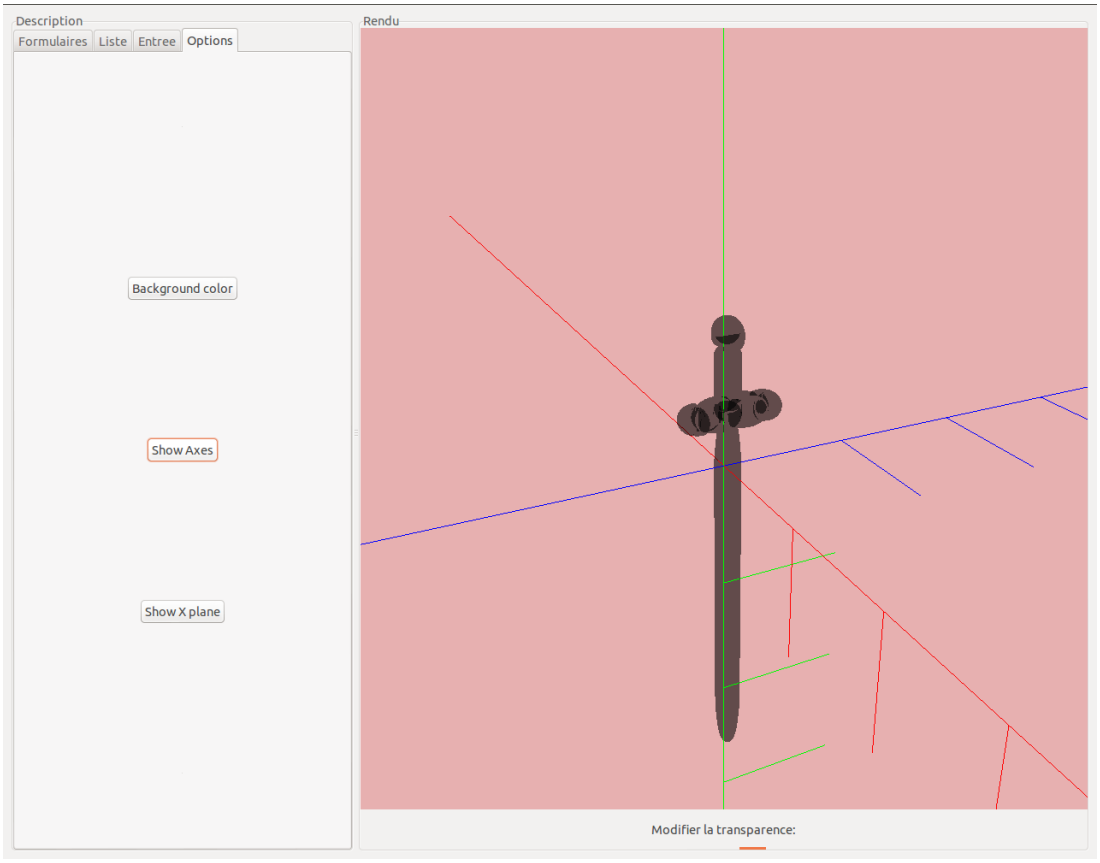
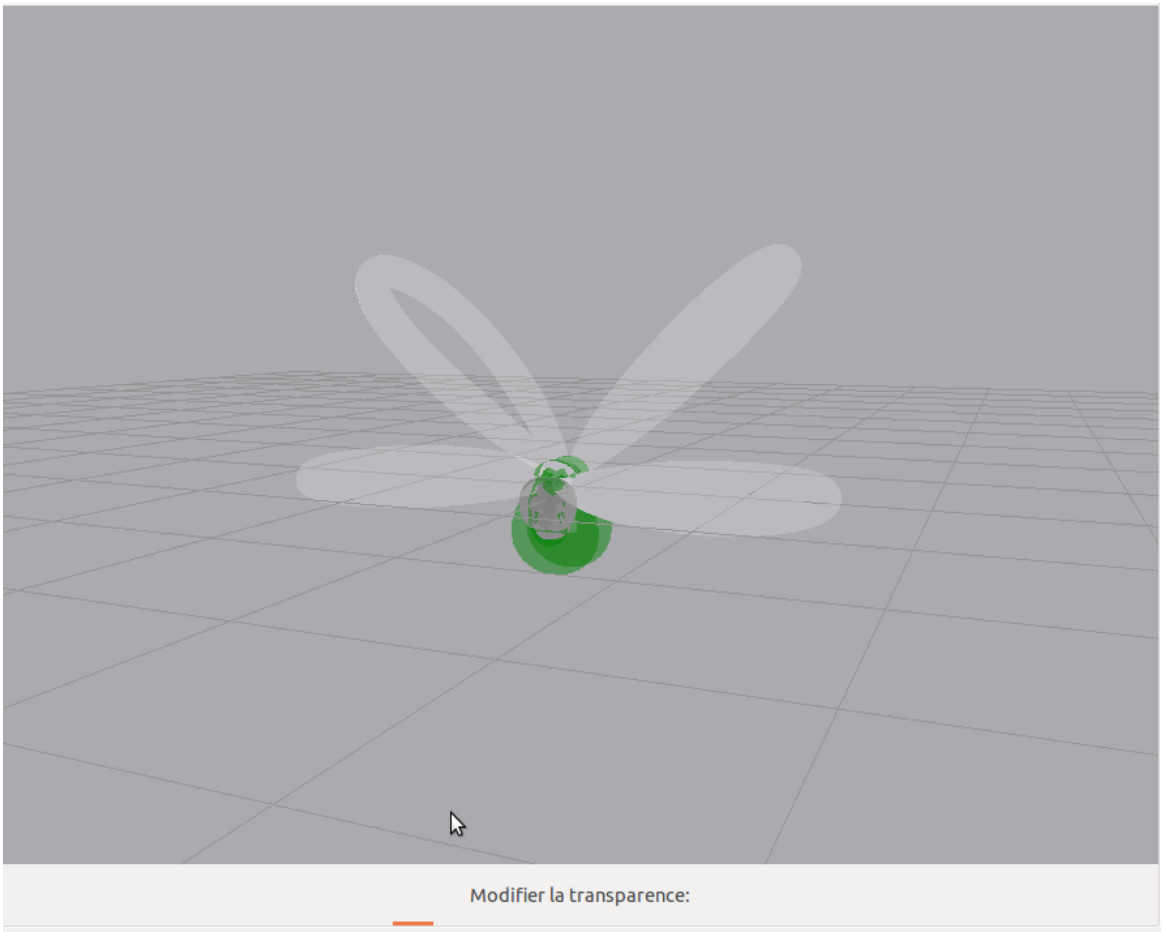


FIGURE 10 – Options d'affichage : avec axes



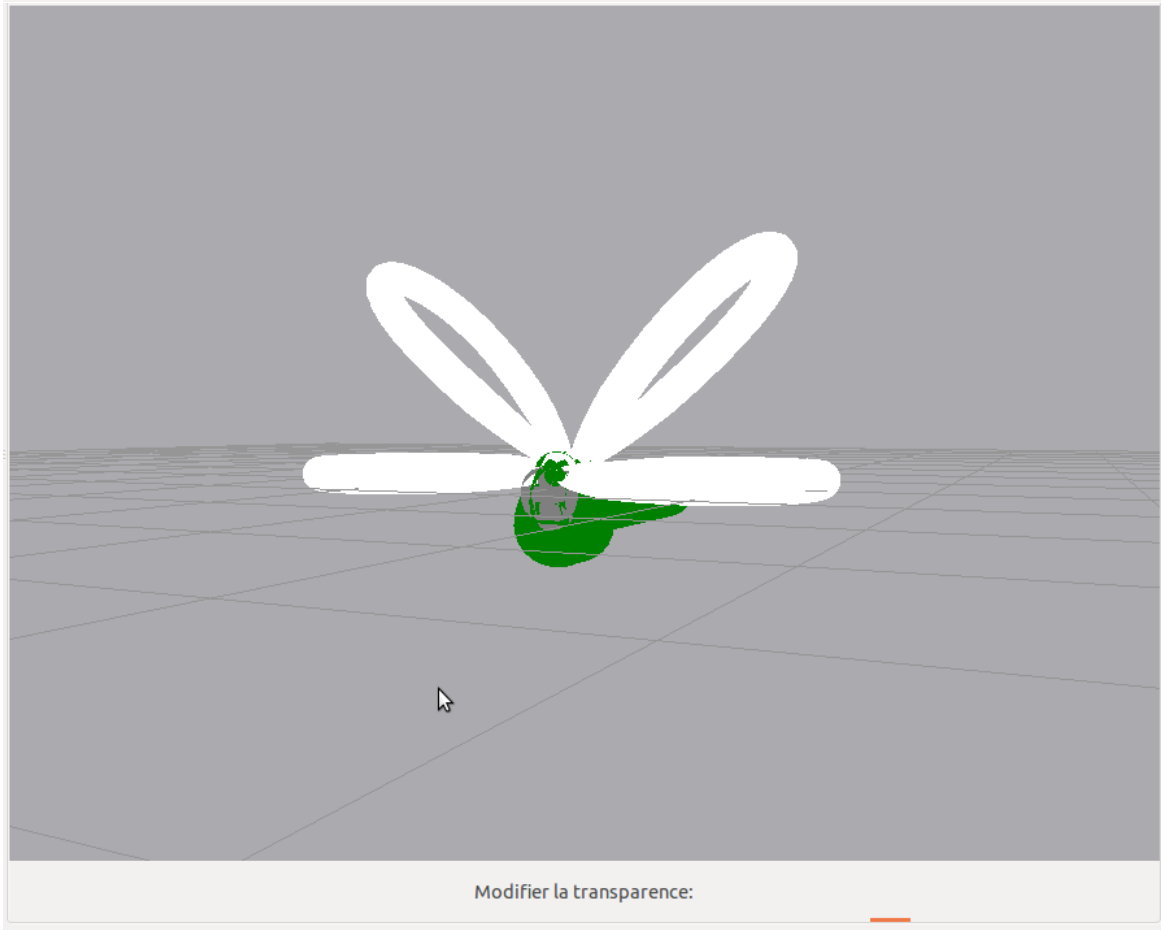


FIGURE 11 – Modification de l'opacité des ballons

7.2 Choix d'implémentations

La partie affichage prend la `Sculpture` en entrée et effectue le dessin de chacun de ses `Balloon`. Le dessin de chaque `Balloon` consiste lui-même en le dessin de chacune de ses `Bubble`.

Fondamentalement, la forme de la bulle d'un ballon à sculpter est un tube tordu. Les extrémités sont simplement des variations dans le rayon. Donc, nous devions faire quelque chose que nous pourrions tordre, et faire en sorte que cela ressemble à un tube. La première chose qui vient à l'esprit pour obtenir des courbes faciles à manipuler sont les courbes de Bezier. Nous avons donc implanté des courbes de Beziers spatiales pour répondre à notre problème.

Nous avions ainsi des lignes courbes et nous pouvions les tordre selon notre fantaisie. C'était la partie facile. Maintenant, qu'est-ce qu'un tube ? Simplement un empilement de cercles. Bien, nous n'avions plus qu'à dessiner des cercles, mais des cercles qui suivent notre courbe de Bezier.

L'astuce aura été de rendre les cercles localement perpendiculaires à la courbe de Bezier en calculant une base orthonormale avec une direction tangente à la courbe. Une fois la base donnée, dessiner un cercle devient trivial. Pour obtenir la base, nous calculons la direction locale (dérivée première de la courbe de Bezier) : cela nous donne un vecteur. Nous prenons ensuite un vecteur normal au précédent de manière arbitraire : deuxième vecteur. Le dernier est simplement calculé comme le produit vectoriel des deux autres.

7.3 Position des nœuds

Maxime Savaro

Pour s'occuper de l'affichage de sculpture entière, le gros défi est de réussir à connaître l'emplacement de chaque nœud. Il s'agit à la fois de suivre les indications de l'utilisateur et d'obtenir une sculpture physiquement possible.

Il s'avère impossible de calculer « brutalement » toutes ces valeurs, par un système d'équations par exemple. Un tel procédé serait bien trop lourd pour nos exigences, on procède donc par étapes et par approximation.

7.3.1 Calcul préliminaire

Dans un premier temps, on calcule grossièrement les positions des nœuds à partir des données utilisateur : il s'agit de suivre les bulles pour découvrir la position d'un nouveau nœud. Un premier nœud est placé arbitrairement à 0. On parcourt les nœuds de la sculpture en considérant ceux dont on n'a pas encore initialisé la position, on considère ensuite les bulles le rattachant à la partie initialisée de la sculpture (i.e. notre nœud est une et une seule des extrémités de ces bulles). Chaque bulle permet de deviner un position pertinente pour le nouveau nœud (position de l'autre extrémité de la bulle calculée à partir de la taille de la bulle et des orientations). Pour les bulles courbes, le vecteur vers le nouveau nœud est donnée par la moyenne des deux directions des extrémités de la bulle multipliée par la longueur déclarée de la bulle. Si un nœud peut être fixé à la partie de la sculpture dont on a déjà initialisé les positions par plusieurs bulles différentes, on moyenne les positions données par chacune des bulles.

On a pu constater que ceci était suffisant pour les sculptures les plus simples, qui présentent une topologie d'arbre. Pour des sculptures plus complexes, le résultat n'est pas satisfaisant ; en particulier, la taille des bulles n'est plus respectée. Il est en outre nécessaire de pouvoir afficher une sculpture non connexe pour réaliser les tutoriels. Pour ce faire, on détecte si aucun positionnement de nœud n'a été effectué lors d'un parcours, puis on positionne un nœud arbitrairement et on continue l'algorithme.

7.3.2 Raffinement des positions

La deuxième étape pour ces sculptures consiste en un algorithme connu de visualisation de grands graphes (force directed algorithm), qui applique simplement des forces aux nœuds et les déplace pas à

pas jusqu'à obtenir une position stable. Le principe de l'algorithme est de modéliser le graphe par un système physique dont on approxime le comportement : chaque bulle se comporte comme un ressort (c'est en réalité un peu plus compliqué). À chaque étape, les nœuds sont déplacés selon les forces exercées par les bulles. Un tel algorithme est sensible à la position initiale des nœuds.

Plus de détails sur cet algorithme sont accessibles sur la page `https://en.wikipedia.org/wiki/Force-based_algorithms_(graph_drawing)`.

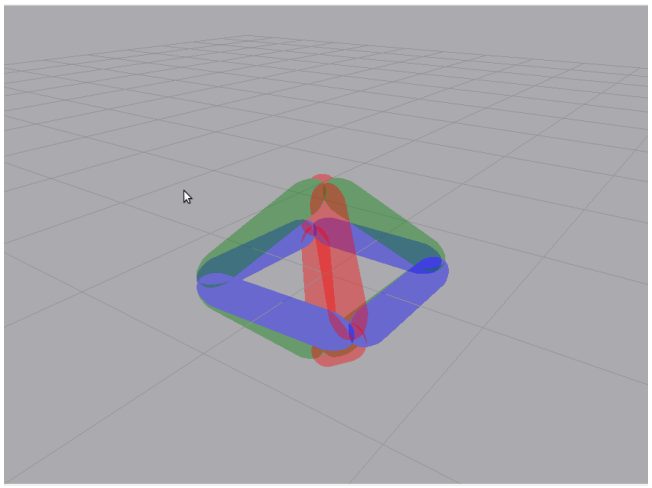


FIGURE 12 – Une sculpture...

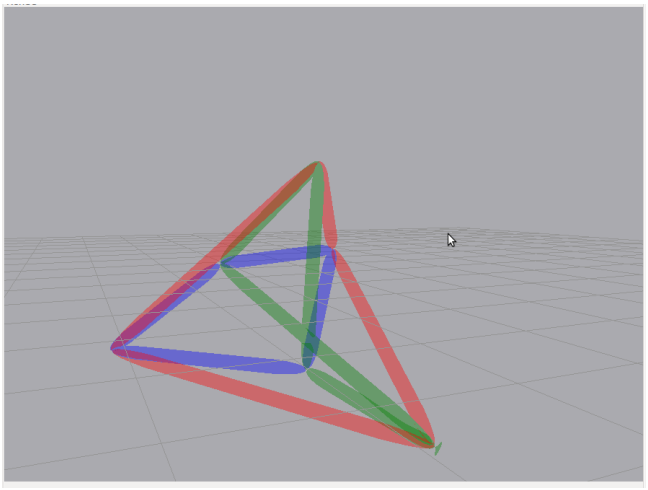


FIGURE 13 – ...déformée par l'utilisateur

Ce type d'algorithme est intéressant pour son aspect ludique (l'utilisateur voit sa sculpture bouger), et pour le fait qu'il converge vers un minimum local. En effet une même sculpture peut avoir plusieurs dispositions réalistes, mais on cherche à atteindre celle que l'utilisateur décris, i.e. celle qui est la plus proche de ce que rend l'étape d'initialisation. Pour appliquer cet algorithme dans l'interface du logiciel, il suffit d'appuyer sur la touche `a`. Les images de la figure 14 illustrent l'effet de cette action.

Il a fallu effectuer quelques adaptations pour s'approcher de nos objectifs. L'algorithme conventionnel tend à écarteler les nœuds pour rendre le graphe plus lisible, ce qui nous est nuisible car cet effet rend les bulles trop longues et la sculpture adopte une disposition trop éloignée de celle rendue à la première étape. Pour l'heure, cette étape permet seulement d'imposer la longueur des bulles, en modélisant les forces exercées par les bulles sur les nœuds. La force exercée entre deux nœuds par une bulle droite ne s'annule que si la distance entre ces nœuds est égale à la longueur déclarée de la bulle.

Il reste à prendre en compte les forces exercées par les bulles entre elles. Autour d'un même nœud, les bulles sortantes et entrantes s'orientent de façon à maximiser leur distance (quatre bulles se dirigent

spontanément dans les directions des sommets d'un tétraèdre régulier, mais il est aussi possible que ces bulles forment une croix...). Il s'agira donc d'implémenter une force de répulsion entre les bulles, agrémentée d'un frottement statique conséquent.

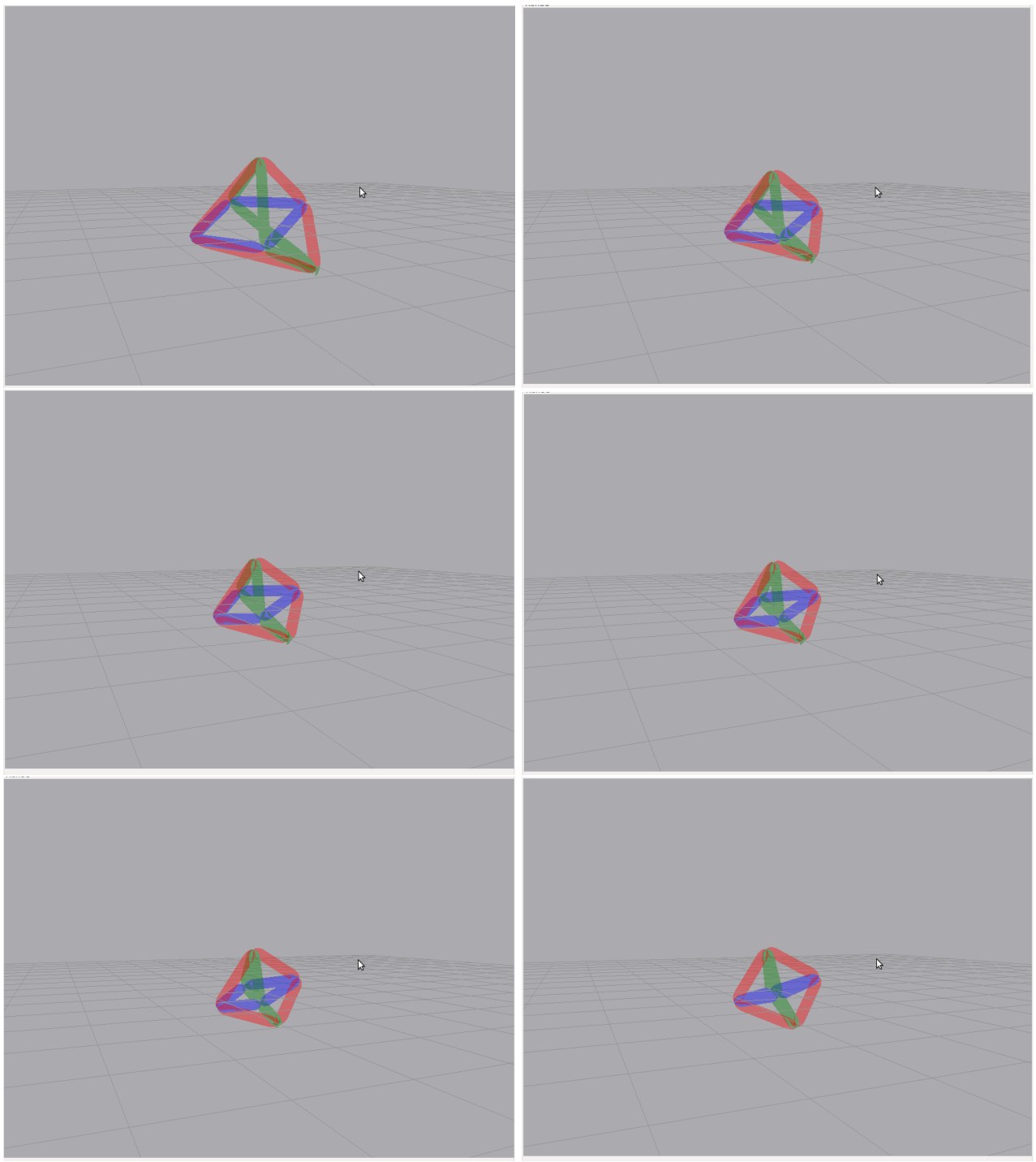


FIGURE 14 – Pressions successives de la touche « a »

7.4 Forme des bulles

7.4.1 Objectifs

Comme indiqué dans la section 3, pour calculer la trajectoire d'une bulle courbe, on a besoin de calculer 2 points de contrôle, qui dépendent de la longueur de la bulle, de ses orientations, et de la position des nœuds qui y sont rattachés. La fonction qui exécute ce calcul est `UpdateControlPoints` dans le fichier bubble.cpp.

7.4.2 Solution choisie

On commence par calculer le rapport *rate* de la distance entre les nœuds sur la taille de la bulle, qui doit être plus petit que 1, sinon la bulle est affichée droite. Ensuite, on calcule un coefficient $coeff$ qui est égal à la taille de la bulle quand $rate = 0$, et à 0 quand $rate = 1$. Pour calculer le premier point de contrôle, on commence par multiplier l'orientation d'entrée par $coeff$. Ensuite, on « éloigne »ce vecteur du vecteur formé par les deux nœuds à l'aide de produit vectoriels. Le coefficient d'éloignement *eloign* doit décroître en fonction de *rate*. Le point de contrôle est le nœud de départ auquel on ajoute ce vecteur.

7.4.3 Explication

Quand *rate* est proche de 0, on veut des points de contrôle proches de la verticale du vecteur formé par les deux nœuds. C'est pourquoi $coeff$ est proche de la taille de la bulle (ainsi on obtient une bulle qui part en direction des points de contrôles, et fait demi-tour vers la moitié du chemin : elle est donc de la bonne longueur). Quand *rate* est plus grand, on veut au contraire des points de contrôles moins éloignés. Donc on se rapproche plus des orientations données par l'utilisateur. Les valeurs adoptées sont $eloign := 1 - rate$ et $coeff := taille * sqrt(1 - rate)$.

7.4.4 Optimisations

Le calcul n'est pas parfait. En effet, l'utilisation de produit vectoriels induit des trajectoires non attendues lorsque l'on bouge les nœuds dans l'interface cliquer-glisser, car le calcul suppose que la direction des nœuds est semblable à la moyenne entre les deux orientations, ce qui n'est plus valide si l'on bouge les nœuds à la souris. Un autre problème est que l'on ne garantit pas la taille de la bulle. Mais ceci ne doit pas être primordial car l'utilisateur ne peut donner une taille et une orientation parfaitement cohérentes. Il nous a donc paru naturel de s'autoriser une marge de manœuvre afin d'adapter les valeurs. Cependant, ce calcul reste tout de même à améliorer.

8 Création par cliquer-glisser

Ionelia Aniela Popescu,
Quentin Santos

Le but du programme est de permettre à tout utilisateur de créer une sculpture, même s'il n'est pas expert en informatique. Pour cela, nous avons créé une interface cliquable qui permet de créer successivement des bulles, organisées en ballons, et ainsi de créer une sculpture sans connaître le langage, qui rebute certains.

Le processus de modélisation 3D paraît simple à première vue, puisqu'on rencontre souvent cette technique de création aujourd'hui. En réalité, c'est un peu complexe, principalement à cause de l'utilisation de dispositifs planaires pour interagir avec un environnement et des objets en trois dimensions. Dans cette situation, le développeur qui veut créer une interface cliquer-glisser doit être attentif aux petits détails qui peuvent rendre la tâche de l'utilisateur plus facile et plus intuitive. Si possible, il doit s'assurer que chacun des gestes de l'utilisateur a une signification dans l'environnement.

Pour cette interface, nous avons cherché à offrir les outils que nous considérons comme étant indispensables pour la modélisation de ballons. Une partie importante de cette interface est le déplacement d'un objet sur les 3 axes. Des améliorations sont encore possibles.

Il y a 4 parties principales dans l'implémentation de l'interface cliquer/glisser :

8.1 *Renderer*

Le lieu où « dessiner c'est gagner » ! Si l'interface cliquer-glisser est active, on donne le contrôle au *renderer* pour :

- dessiner les bulles de la sculpture modélisée ;
- avoir un retour visuel pour connaître la bulle, le nœud ou le groupe sélectionné(e)(s) ;
- déplacer les nœuds par cliquer-glisser ; les nœuds sont invisibles lorsqu'ils ne sont pas sélectionnés.

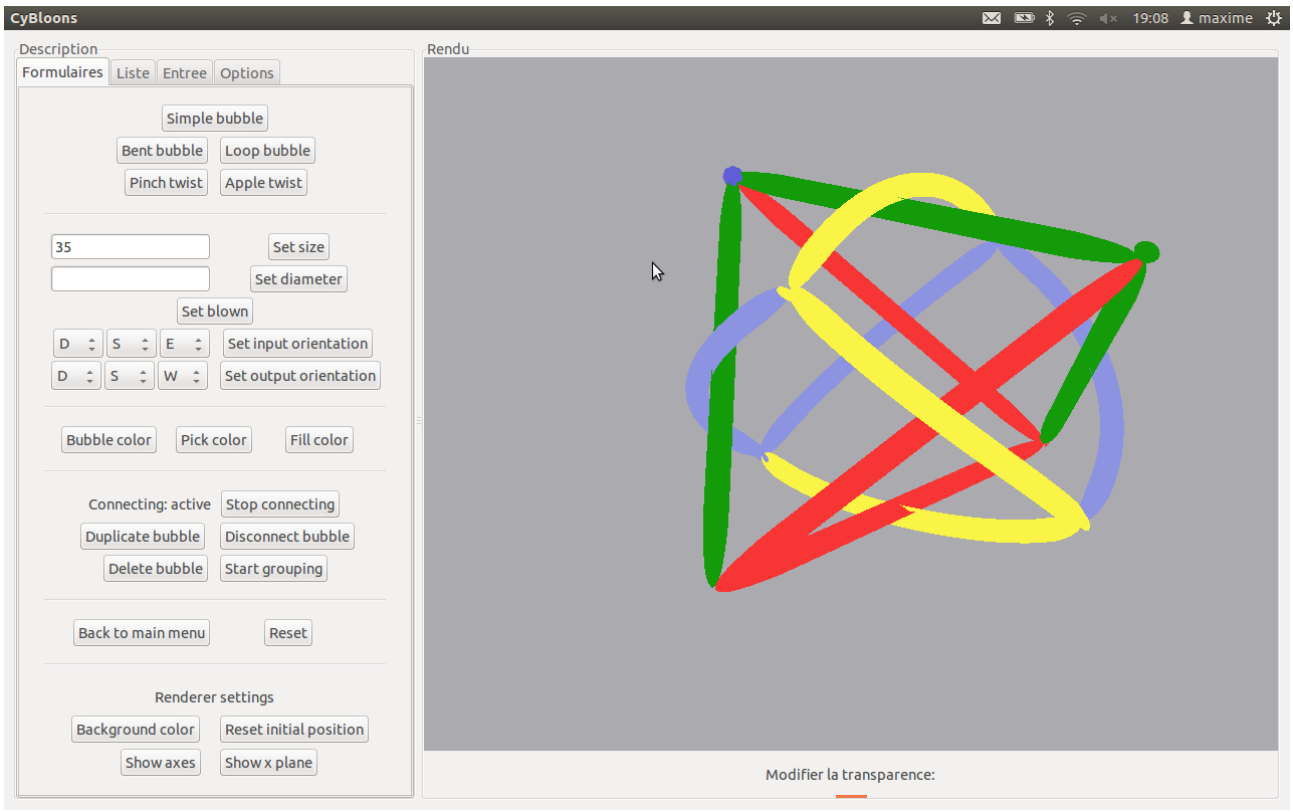


FIGURE 15 – L'interface cliquer-glisser

8.2 La fenêtre gtk+OpenGl

Entre autres choses, ici, on traite aussi l'utilisation de la souris pour l'interface cliquer-glisser.

On peut prendre une bulle ou un nœud et le déplacer à la souris : clic gauche et long glissé. Pour déplacer selon un axe précis, on combine un clic droit et les touches x, y et z.

Lorsque des évènements surviennent ici, comme la sélection d'une nouvelle bulle ou d'un nouveau nœud, et si l'interface cliquer-glisser est active, on lui donne le contrôle pour mettre à jour son état. Elle doit vérifier les collisions et lier des objets, ajouter une bulle à un groupe, *etc.*

Une amélioration ici sera l'ajout d'un clic droit pour le déplacement en profondeur. Ce n'est pas particulièrement difficile, mais le manque de temps a empêché, jusqu'ici, sa réalisation.

8.3 Le formulaire gtk

Dans cette partie nous nous sommes inspirés du code des formulaires pour créer la fenêtre **gtk** avec des boutons pour les outils nécessaires pour une interface cliquer-glisser.

Lorsque des évènements surviennent ici :

- on fait un appel aux fonctions de la classe qui gère les évènements du cliquer-glisser : `dragDropManager`.
- si nécessaire, on met à jour la configuration de la fenêtre. Par exemple, le mode « connecting » permet de connecter systématiquement deux bulles dès que vous approchez leurs extrémités. Un court message indique s'il est actif. Il s'active par un bouton « start connecting », et se désactive par le même bouton alors renommé « stop connecting ».

8.4 La classe qui gère les évènements du cliquer-glisser : `dragDropManager`

La partie « model » de l'interface cliquer-glisser.

Il y a deux fonctionnalités majeures dans cette partie :

1. Mettre à jour son état quand des actions ou événements comme « prendre », « glisser », « relâcher » se déroulent (cette partie est liée à gtk — la partie « viewer »).

La méthode principale est « update », qui va lier des bulles ou des noeuds quand ils entrent en collision, et qui va déplacer un groupe quand on déplace l'une des bulles de ce groupe.

Remarque : ici, il reste à vérifier les collisions entre les bulles.

2. Créer des classes du modèle (Balloon, Bubble, *etc.*), et mettre à jour leur état à la demande (cette partie est liée à la fenêtre gtk qui représente la partie « contrôler »).

   Avec les méthodes de cette partie, on peut créer des nouvelles bulles, changer les paramètres d'une bulle, *etc.*

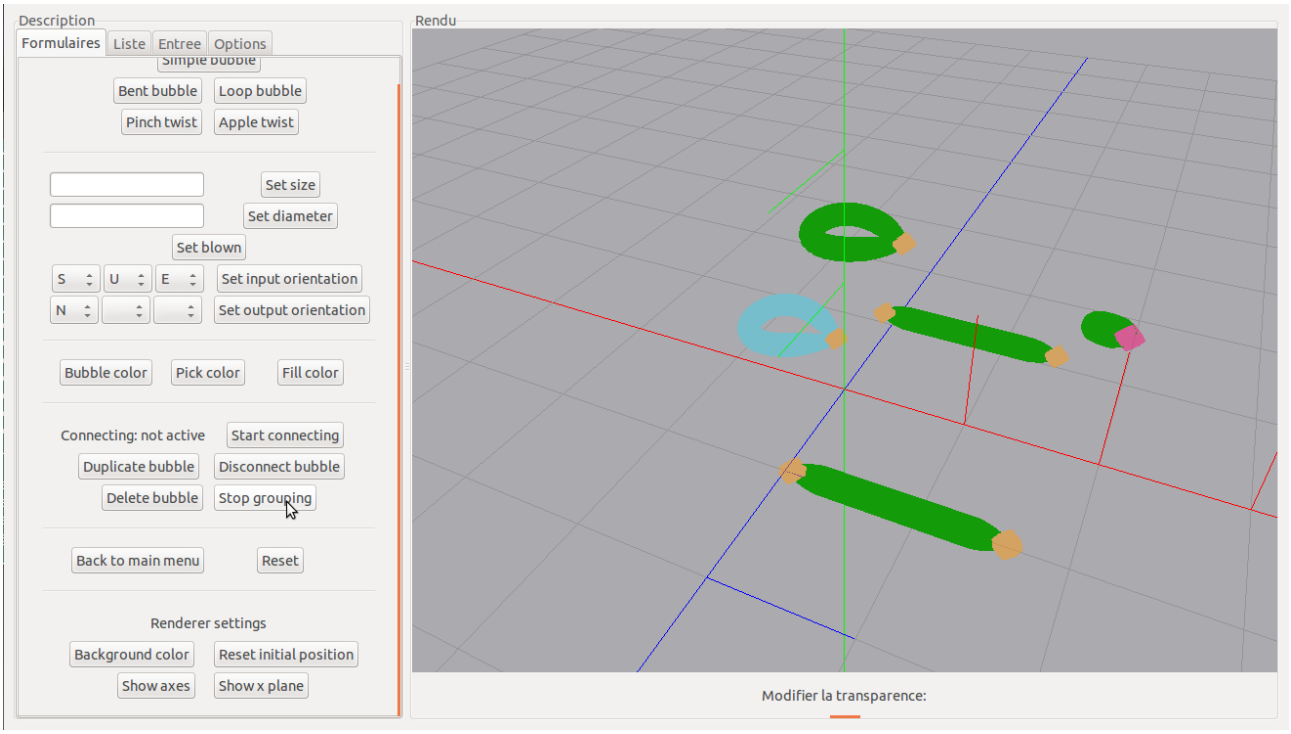


FIGURE 16 – Les différentes possibilités de bulles

Enregistrement de la création

Rémi NOLLET

L'étape suivante, en cours d'implémentation, est la sauvegarde d'une sculpture ainsi crée, par production de sa description dans le langage décrit dans la section 5. Par la suite ceci sera encore amélioré afin que le logiciel soit capable d'optimiser l'ordre de construction de la sculpture.

9 Formulaires

Arnaud BARDOUX,
Bertrand SIMON,
Maxime SAVARO

9.1 Problématique

Lors de la création du projet, notre ambition était de créer un langage compréhensible et utilisable par n'importe qui. Il fallait également que ce langage contienne toutes les informations nécessaires à la description du ballon. Les critères précis sont ceux que nous avons donnés dans la partie 5 concernant le langage (p.11). Malheureusement, cela s'est avéré impossible. Pour avoir un langage assez précis, il fallait nécessairement passer par une forme qui nécessite un minimum de pratique pour être utilisée.

C'est pour respecter notre décision de rendre la création de mot accessible à tous que nous avons mis en place des formulaires. Pour faire l'intermédiaire entre la sculpture manuelle et l'écriture d'un

mot de notre langage, ils requièrent que l'utilisateur indique une à une les étapes qu'il accomplit. En cours de création, il est possible de vérifier l'affichage (partiel) de la sculpture, le mot en cours de création ainsi qu'une liste correspondant à la liste des actions indiquées par l'utilisateur.

9.2 Utilisation

La création d'une sculpture passe par plusieurs étapes, chacune correspondant à un formulaire particulier. L'utilisateur peut utiliser successivement les formulaires de création de :

- sculpture ;
- ballon ;
- bulle ;
- boucle.

La première étape est forcément une création de ballon, dans une sculpture. Sans ballon, rien n'est possible. Ensuite, on y ajoute les bulles qui le composent, et on crée des nouveaux nœuds en même temps. A tout moment, il est possible de créer un nouveau ballon, de modifier un ballon précédent ou même une bulle appartenant à n'importe quel ballon. Il est également possible d'annuler les dernières actions une à une puis de les refaire. Chaque formulaire contient les différents paramètres nécessaires à la création de l'objet concerné. Nous allons détailler chaque formulaire.

9.2.1 La création (ou modification) d'une sculpture

Elle passe par la donnée de :

- son nom (une chaîne de caractères alphanumériques sans espace)
- les ballons qui le composent
- les boucles qui le composent

L'utilisateur peut créer et modifier les ballons et les boucles qui composent sa sculpture.

La figure 17 montre ce formulaire.

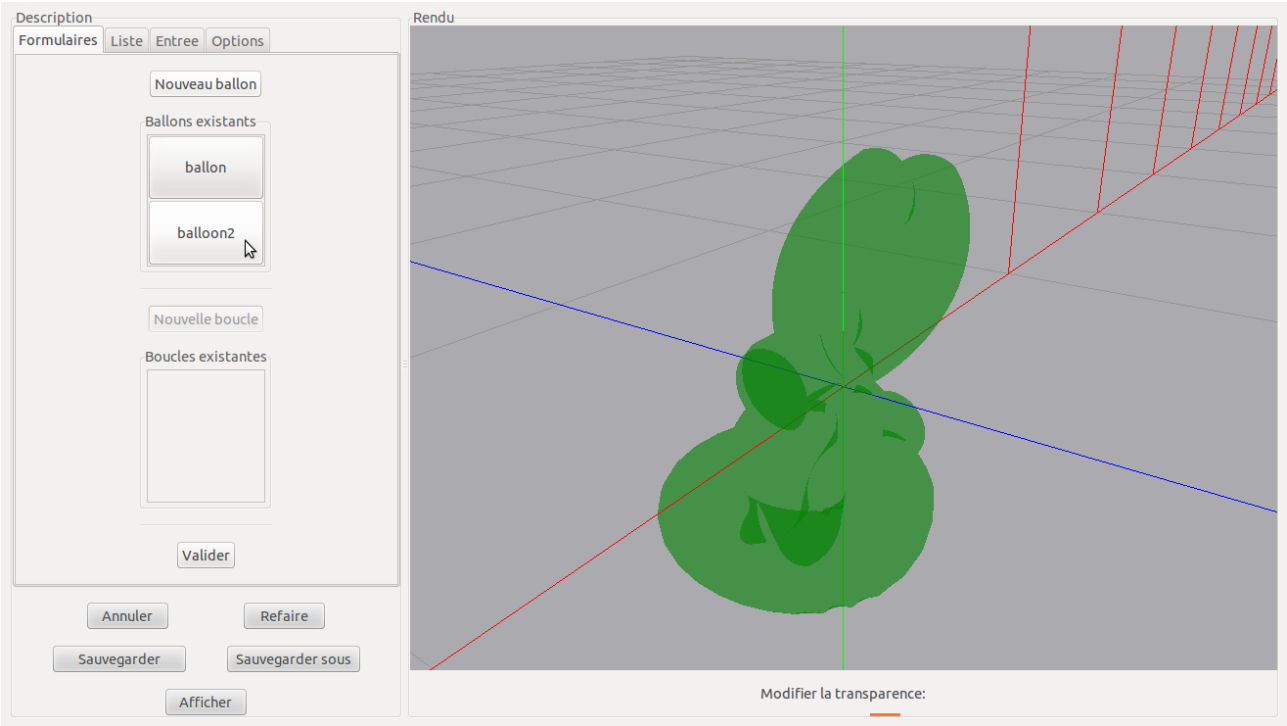


FIGURE 17 – Formulaire de création/modification d'une sculpture

9.2.2 La création (ou modification) d'un ballon

Elle passe par la donnée de :
- son nom (une chaîne de caractères alphanumériques sans espace)
- sa taille (160, 260, 350 ou 646)
- sa couleur (via la représentation hexadécimale en RGB)
- les bulles qui le composent

L'utilisateur peut créer des nouvelles bulles à partir de ce formulaire ou modifier celles qu'il a créées auparavant et qui s'affichent dans une liste.

La figure 18 montre ce formulaire.

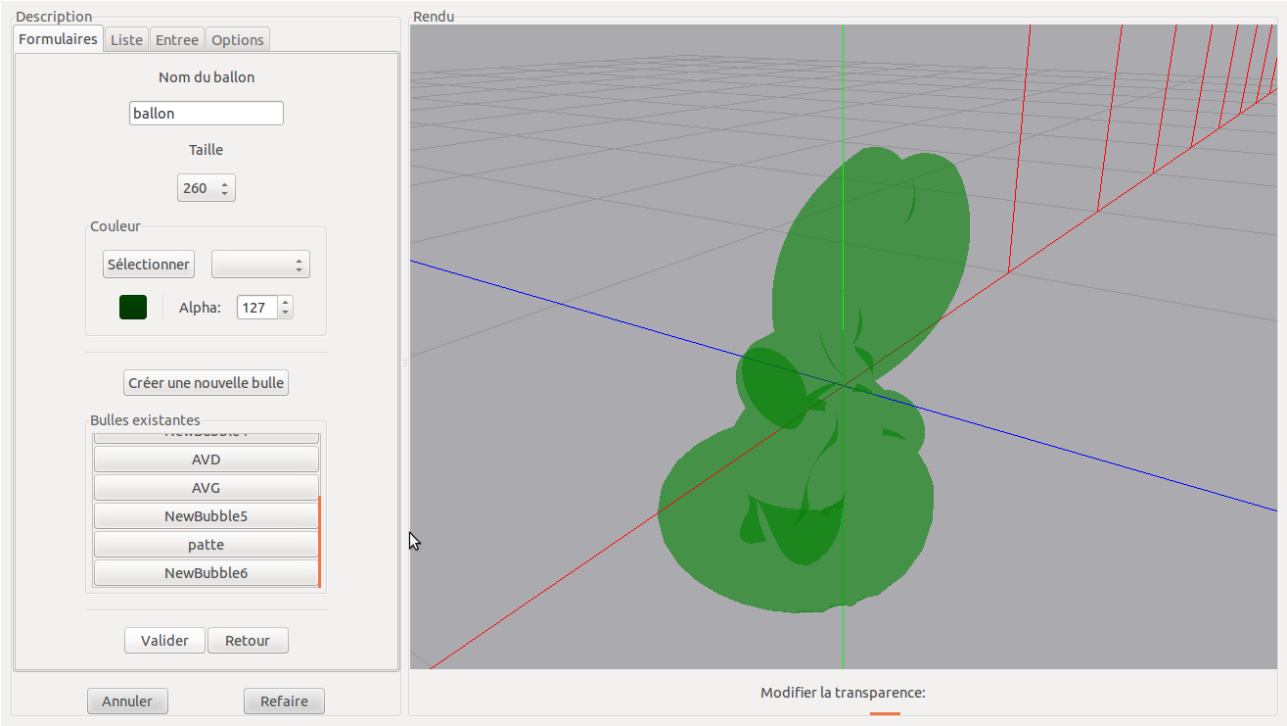


FIGURE 18 – Formulaire de création/modification d'un ballon

9.2.3 La création (ou modification) d'une bulle

Elle passe par la donnée de :
- son nom (une chaîne de caractères alphanumériques sans espace)
- sa longueur (inférieure à la longueur totale du ballon)
- si elle est gonflée ou non
- les nœuds d'entrée et de sortie
- l'orientation de la bulle par rapport à son nœud d'entrée et à son nœud de sortie (par l'utilisation des trois dimensions représentées par les lettres N,S,E,W,U,D ainsi que nous l'avons expliqué dans la section langage (p.13))
- la forme de la bulle (droite, courbe ou tulipe ; sélectionnable dans une liste)

La figure 19 montre ce formulaire.

Si aucun nœud n'existe dans la sculpture, il est possible d'en créer un dans ce formulaire. On arrive alors au formulaire concernant les nœuds.

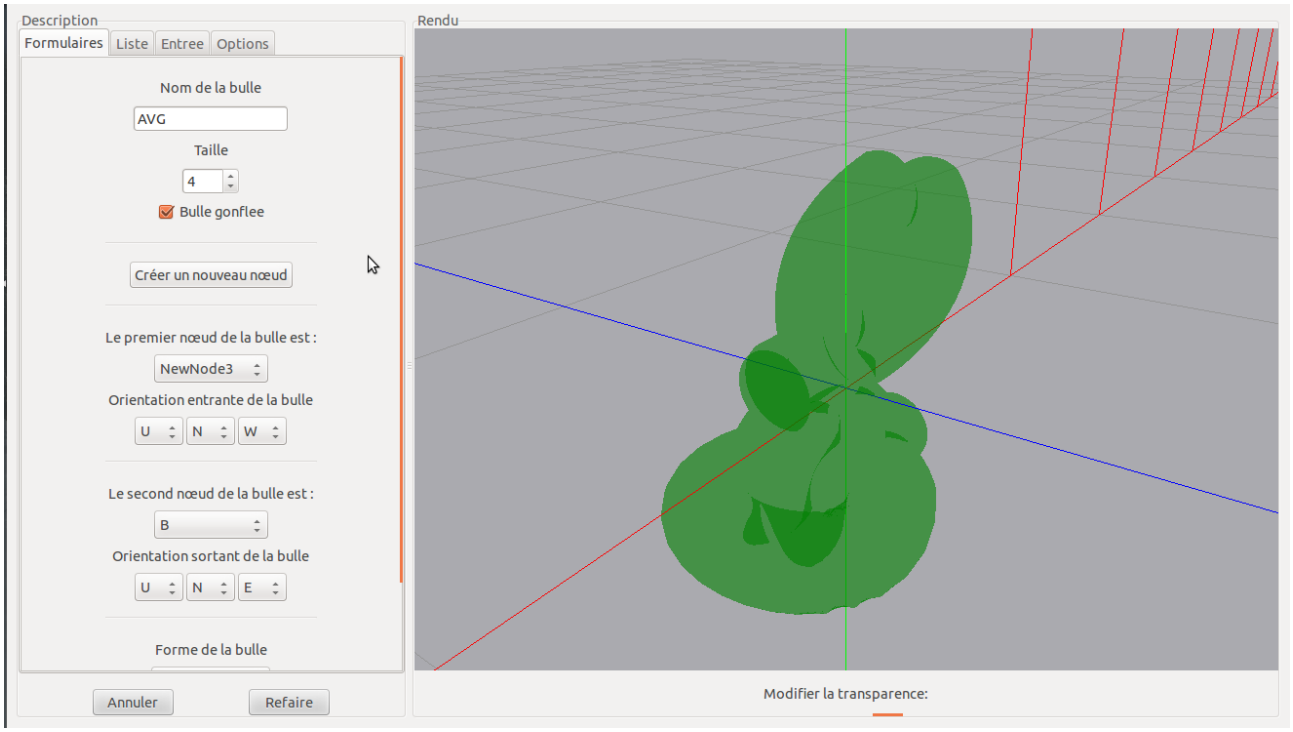


FIGURE 19 – Formulaire de création/modification d'une bulle

9.2.4 La création (ou modification) d'un nœud

Elle ne nécessite que son nom. Le nœud est ensuite enregistré dans les données concernant la sculpture et pourra être donné comme nœud d'entrée et/ou de sortie de ses nœuds.

9.2.5 La création (ou modification) d'une boucle

L'implémentation de cette partie du formulaire est en cours. les champs demandés seront :
- son nom (une chaîne de caractères alphanumériques sans espace)
- les bulles qui le composent le tour de la boucle
- les bulles qui passent dans la boucle, ainsi que la longueur de bulle qui y passe.

9.3 Fonctionnement

Pour pouvoir utiliser les données que l'utilisateur renseigne dans les différents formulaires, il a fallu créer une sctructure de données adaptée. Cette structure doit
- contenir les différentes informations de chaque formulaire
- être accessible à tout moment et dans tous les formulaire (afin de pouvoir récupérer par exemple la liste des nœuds d'une sculpture dans le formulaire ballon ou la liste des bulles d'un ballon dans le formulaire dudit ballon).

Pour répondre à ces exigences, plusieurs classes ont été créées.
- la classe `save` est la classe centrale de cette section. Une instance de cette classe est créée à chaque fois qu'un utilisateur valide un formulaire. On sauvegarde tout au long de la création la liste de `save` engendrée.
- les classes `fsculpt`, `fballoon`, `fbubble` et `fnode` et `floopnode` contiennent respectivement les données concernant une sculpture, un ballon, une bulle, un nœud ou une boucle qui ont été décrits ci-avant. Ces classes sont analogues à celles utilisées dans le corps du programme, comme expliqué section 3.
- la classe `data`, qui est un singleton, permet de stocker toutes ces données.

L'objectif de la classe `save` est de sauvegarder l'action que l'on a accomplie. Elle peut par exemple concerner la création d'une bulle avec tous ses paramètres, la création d'un ballon, la modification de la taille d'une bulle... Son principe est de créer un objet par élément créé (par exemple une bulle, un ballon) ; et de pointer vers l'objet que l'on traite. Ainsi, on peut facilement ajouter ou enlever une modification, ainsi que traduire cette liste en un mot du langage.

Lorsque l'utilisateur le souhaite (quand il veut afficher ou sauvegarder la sculpture), ou quand on veut vérifier que les informations entrées sont cohérentes (quand un formulaire est validé), on calcule le mot correspondant à la liste de `save`. On peut ensuite, si besoin, afficher les erreurs envoyées par le parser. C'est le mot ainsi calculé, enregistré dans un fichier texte, qui permet d'enregistrer la sculpture et de la récupérer ultérieurement.

Pour ouvrir un fichier, le parser lit un mot du langage (issu d'un fichier par exemple) et crée à partir de là les instances de classes nécessaires à l'affichage et/ou à la modification de la sculpture.

Lors de l'affichage d'une sculpture, si celle-ci est issue des formulaires ou de l'ouverture d'un fichier, l'utilisateur peut également consulter la liste de `save` qui la décrit ainsi que le mot associé. Les figures 20 et 21 montrent cette possibilité.

9.4 Optimisations

Il reste à finaliser le formulaire concernant les boucles. De plus, on peut souhaiter une interface plus intuitive pour manipuler la liste de `save`. En effet, il serait agréable de pouvoir supprimer ou modifier directement une instruction en la voyant dans la liste. Pour l'instant, on peut uniquement annuler la dernière action, et refaire la dernière action annulée.

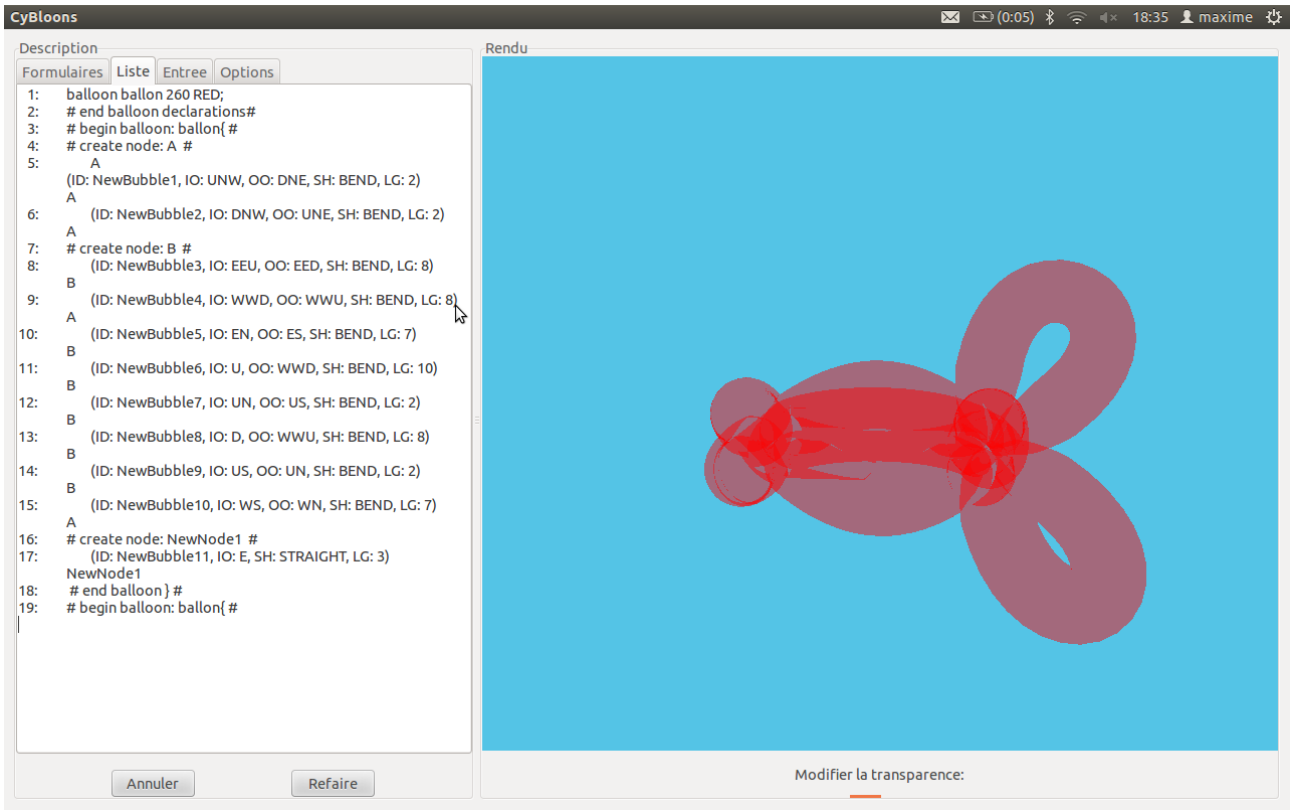


FIGURE 20 – Liste des `save` de la sculpture

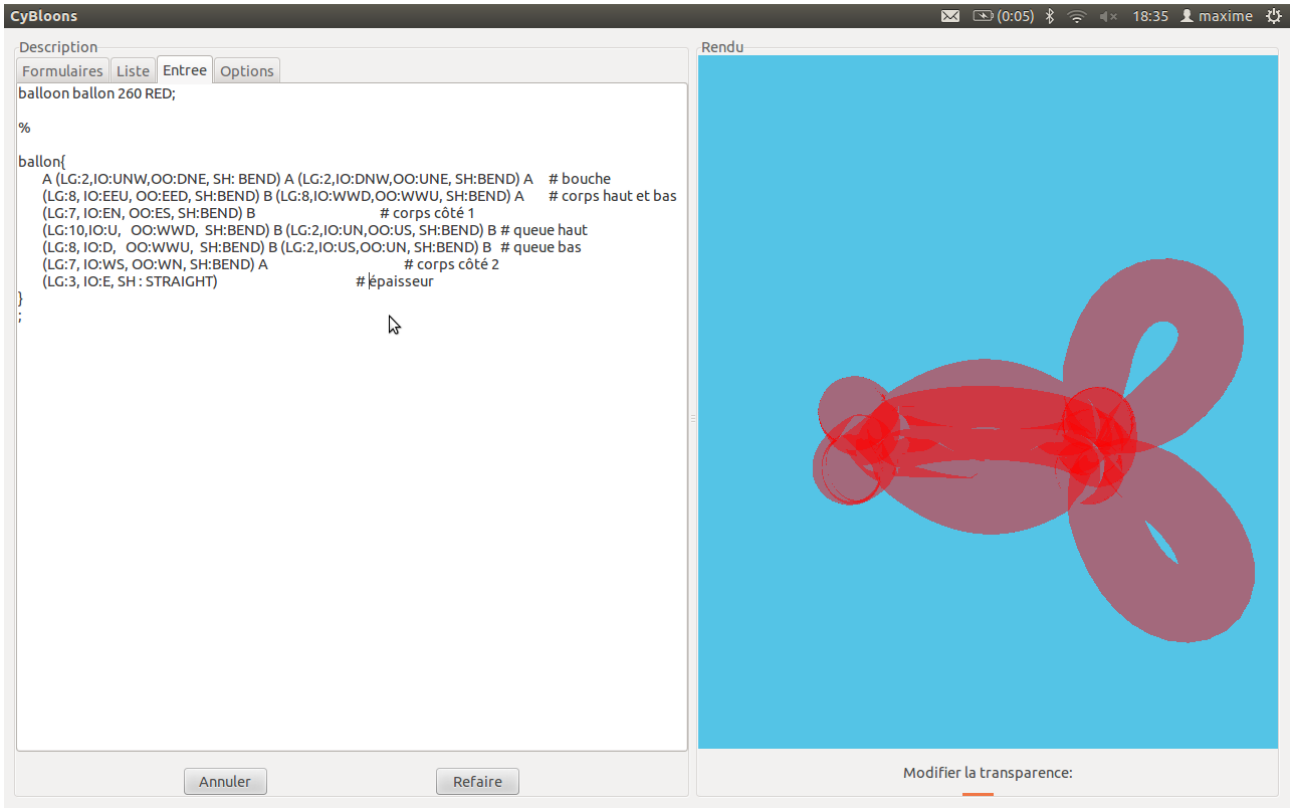


FIGURE 21 – Mot associé à la sculpture

10 Tutoriels

Camille Brasseur

Les tutoriels doivent permettre à l'utilisateur de voir à l'écran (et de reproduire en réel) les étapes successives de la création d'une sculpture. Cela reprend donc immédiatement les étapes décrites dans les formulaires et donc la liste créée lors de l'utilisation de ces formulaires.

Ils ont pu être facilement réalisés puisqu'ils utilisent des outils précédemment mis en place pour le logiciel : la liste des instructions (les instructions sont de type `save`), les différentes catégories d'instructions (les classes `fculpt`, `fballoon`, `fbubble` et `fnode`) et l'affichage d'une sculpture à partir d'une telle liste.

Une fois la liste récupérée, le tutoriel se déroule en deux temps. D'abord, il faut créer les ballons. Dès cette étape, toutes les déclarations de ballons de la sculpture sont faites. Mais il faut aficher pour l'utilisateur une étape pour chaque ballon en indiquant comment ce ballon est identifié et de quelle longueur il doit être gonflé. Ensuite, il faut indiquer à l'utilisateur chaque nouvelle bulle qu'il forme et où à quel nœud d'arrivée cette bulle se racroche. Au sein du tutoriel, l'utilisateur doit également pouvoir à n'importe quel moment voir les étapes précédentes pour les refaire.

10.1 Création d'un ballon

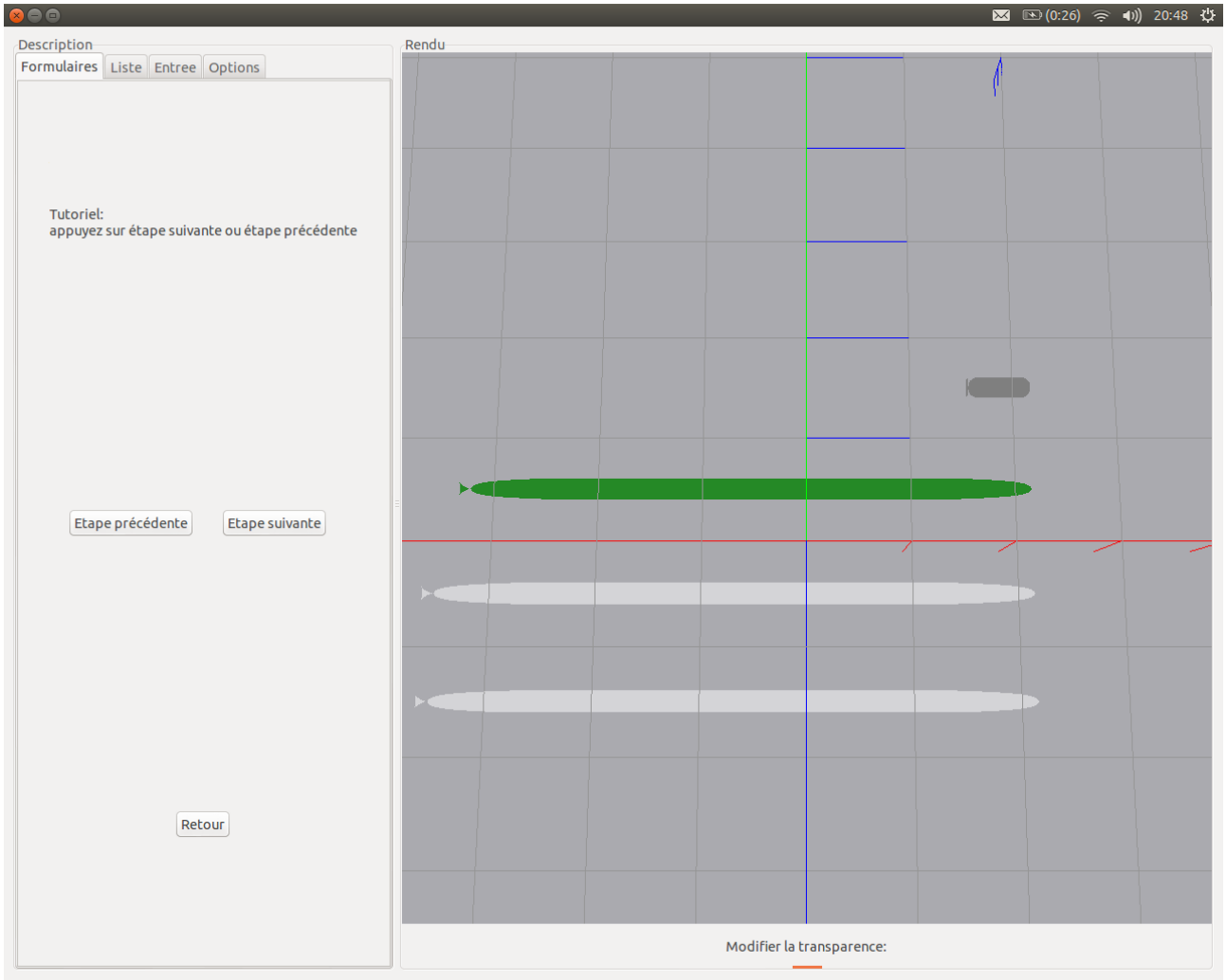


Figure 22 – Gonflage de plusieurs ballons

Le tutoriel travaille avec deux listes de `save`. La première est la liste des instructions déjà faites dans le tutoriel et correspond donc à ce qui s'affiche à l'écran. La seconde possède les instructions qui doivent encore être lues.

Lors de l'ouverture du formulaire de tutoriels, une initialisation des données a lieu. La liste d'instructions faites est vidée ; la liste d'instructions à faire initialisée avec la liste présente dans le singleton `data`. La taille totale gonflée est calculée pour chaque ballon. Lors de la création d'un ballon, pour l'utilisateur, il faut afficher uniquement une bulle de cette longueur. C'est ainsi que sont faits les premiers affichages du tutoriels. La liste de save n'est modifiée que le temps de l'affichage ; elle retrouve ensuite son état initial qui consiste en la définition des ballons. Un compteur est utilisé pour savoir le nombre de ballons gonflés et savoir quelles bulles finales il faut afficher.

Par la suite, la taille de la bulle « finale » est recalculée dès qu'une bulle est ajoutée (ou défaite dans le cas d'un retour en arrière). Cette bulle est toujours affichée à la suite des bulles que l'utilisateur forme, jusqu'à avoir une longueur nulle. Ainsi, l'utilisateur voit à l'écran son ballon tel qu'il est et non pas uniquement les bulles qu'il a formées.

10.2 Gestion d'une étape

Lors du passage à une étape suivante, le logiciel considère l'élément `save` suivant dans la liste des instructions à faire. Généralement, cette instruction va être dépilée et ajoutée à la fin des instructons faites. Plusieurs cas existent :

- L'instruction ne peut pas être la création d'un ballon ou la fin des créations. Si c'est le cas, la liste d'instruction à faire n'est pas une liste créée par le parser à partir d'un mot. Le tutoriel ne pourra donc pas être créé.
- Si l'instruction concerne un ballon, ce peut être l'ouverture d'un ballon (l'utilisateur doit prendre un autre ballon) soit la fermeture (il doit poser son ballon, un autre va être utilisé). Une simple instruction est donc affichée.
- Si l'instruction concerne une bulle, la création se fait en deux étapes. D'abord, l'utilisateur forme la bulle puis il attache son nœud au bon endroit. Dans le tutoriel, deux instructions se succèdent également. L'instruction liée à cette bulle n'est pas dépilée lors de la première insctruction ; elle est utilisée pour créer la bulle au nœud final détaché. Un booléen permet de savoir si une bulle vient d'être formée et doit être reliée au bon endroit.
- Si l'instruction concerne un nœud (ou une boucle), on passe directement à l'instruction suivante. En effet, la création de nœud est enregistrée dans la liste de `save` et sert pour l'affichage mais ne sert pas pour le tutoriel.

10.3 Retour en arrière

Pour retourner en arrière, il « suffit » de rempiler les deux dernières instructions lues dans la liste des instructions à faire (et donc de les supprimer de celles faites) et de passer de nouveau à l'étape suivante. Cependant, il faut gérer les instructions qui ne sont reliées à aucune étape (création de ballon ou de nœud) et les instructions qui prennent deux étapes (création de bulle).

Lorsque l'utilisateur veut revenir en arrière alors que le tutoriel n'est pas commencé, un simple message s'affiche l'invitant à passer à l'étape suivante. Lorsqu'il revient en arrière pendant le gonflage des ballons, il ne faut pas effacer les `save` présent dans la liste qui sont uniquement les définitions de ballons. C'est le compteur de ballons gonflés qui est décrémenté pour que l'instruction de gonflage précédente soit redonnée. Lorsque l'étape précédente est une bulle, on prend en compte (et on modifie) le booléen qui concerne la bulle nouvellement formée pour savoir si une deuxième instruction doit bien être rempilée ou non. Lorsque l'étape précédente concerne une instruction non utilisée dans le tutoriel (création de nœud), l'instruction est bien déplacée d'une liste à l'autre mais cela ne compte pas dans les deux instructions qu'il faut rempiler.

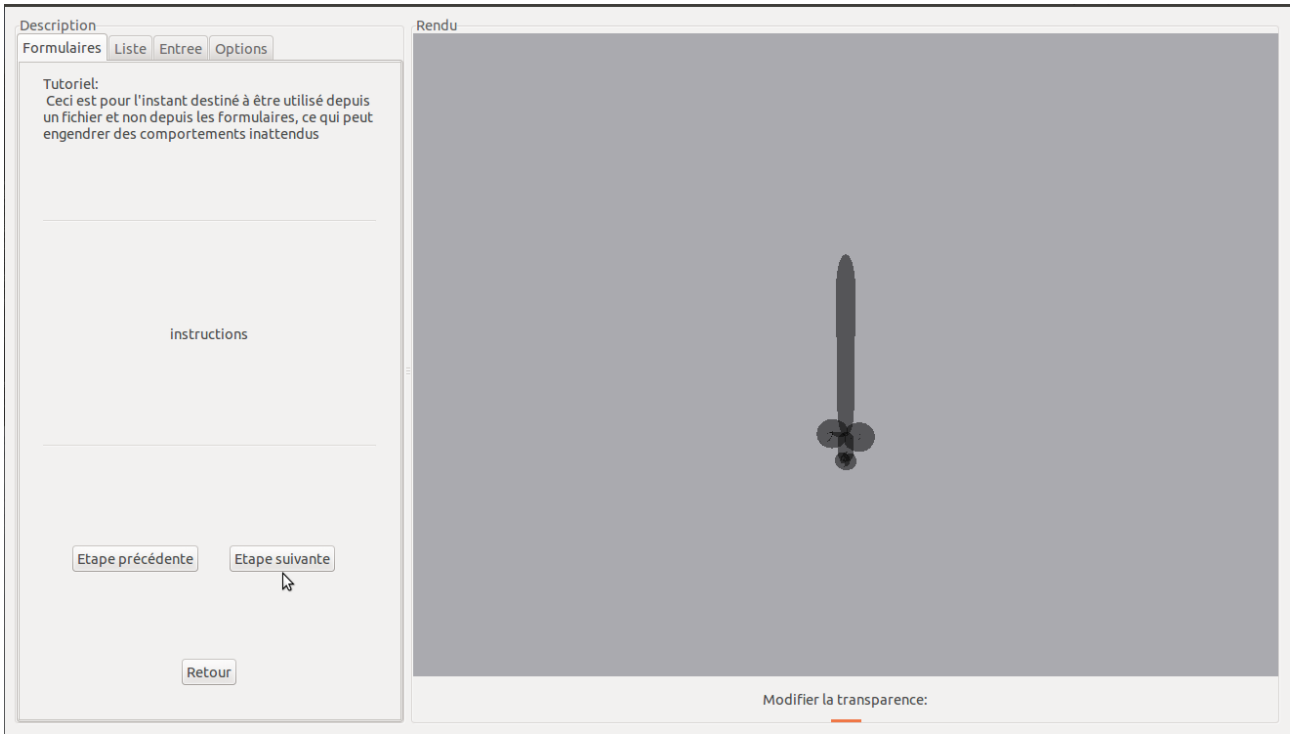


FIGURE 23 – Tutoriel pour une épée : étape 0

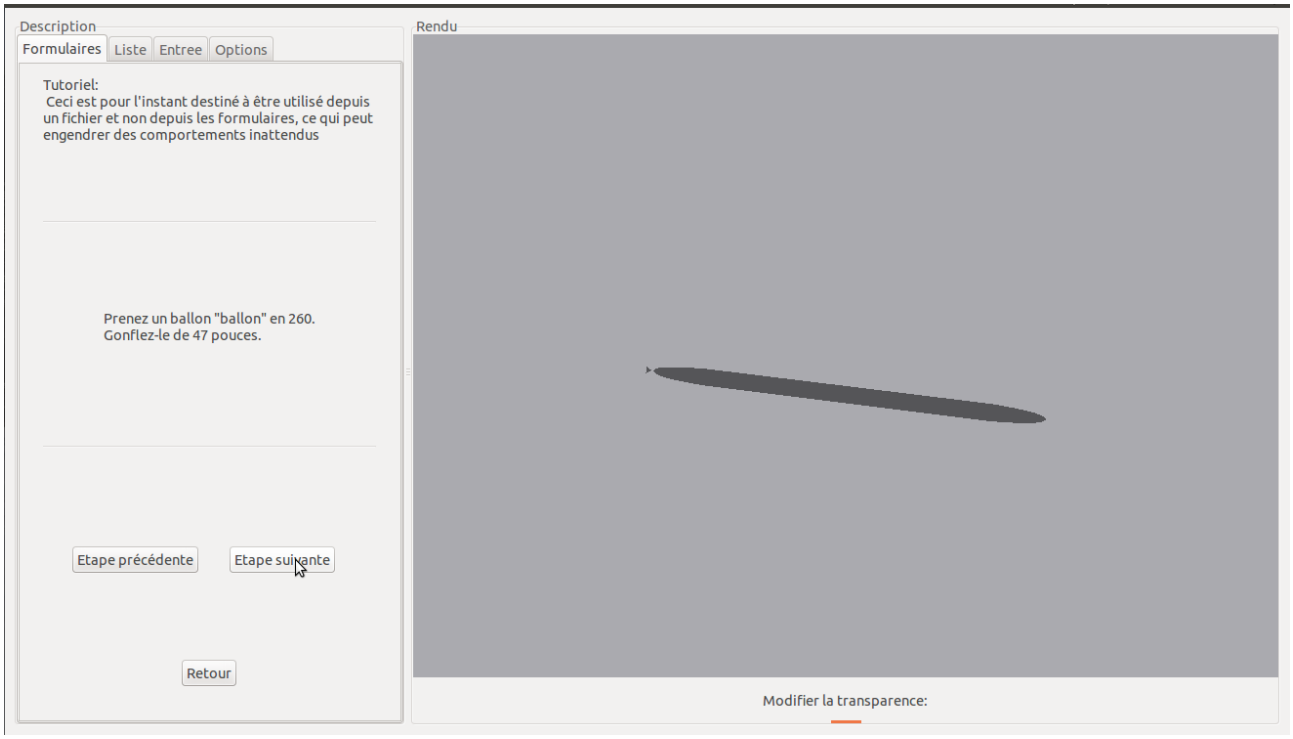


FIGURE 24 – Tutoriel pour une épée : étape 1

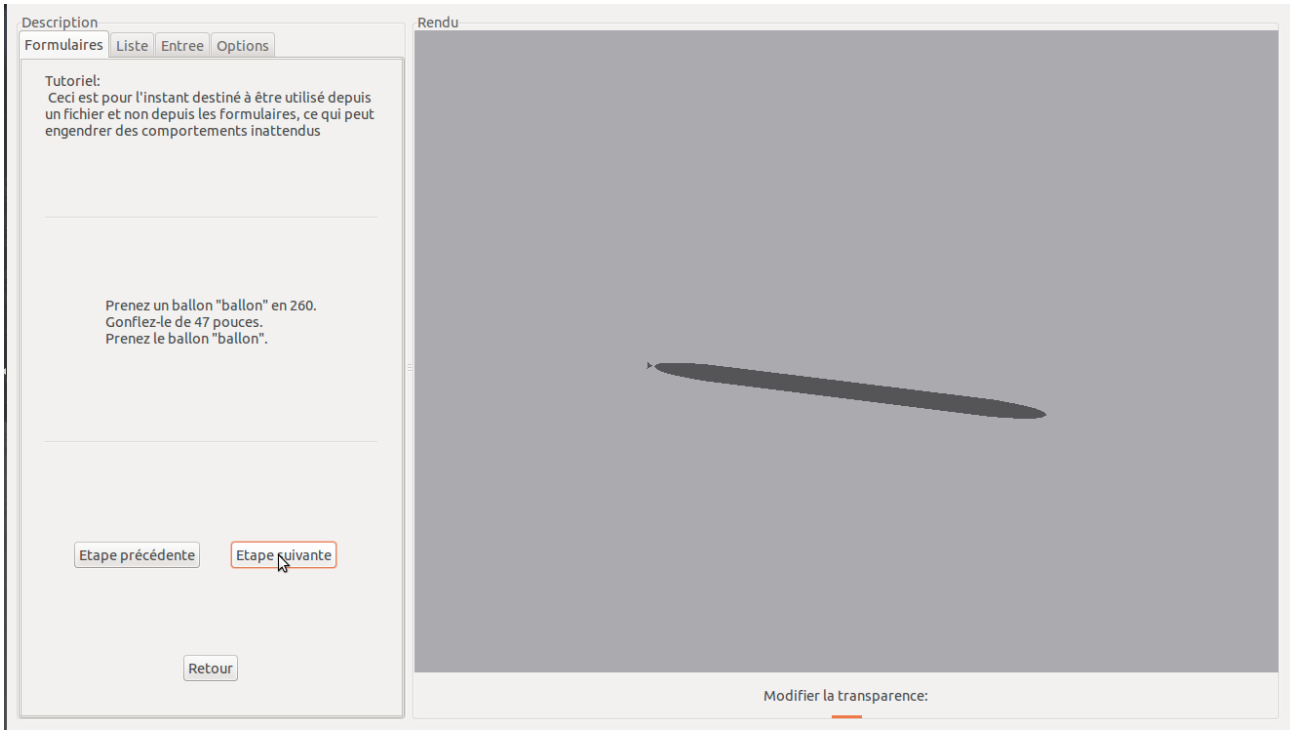


FIGURE 25 – Tutoriel pour une épée : étape 2

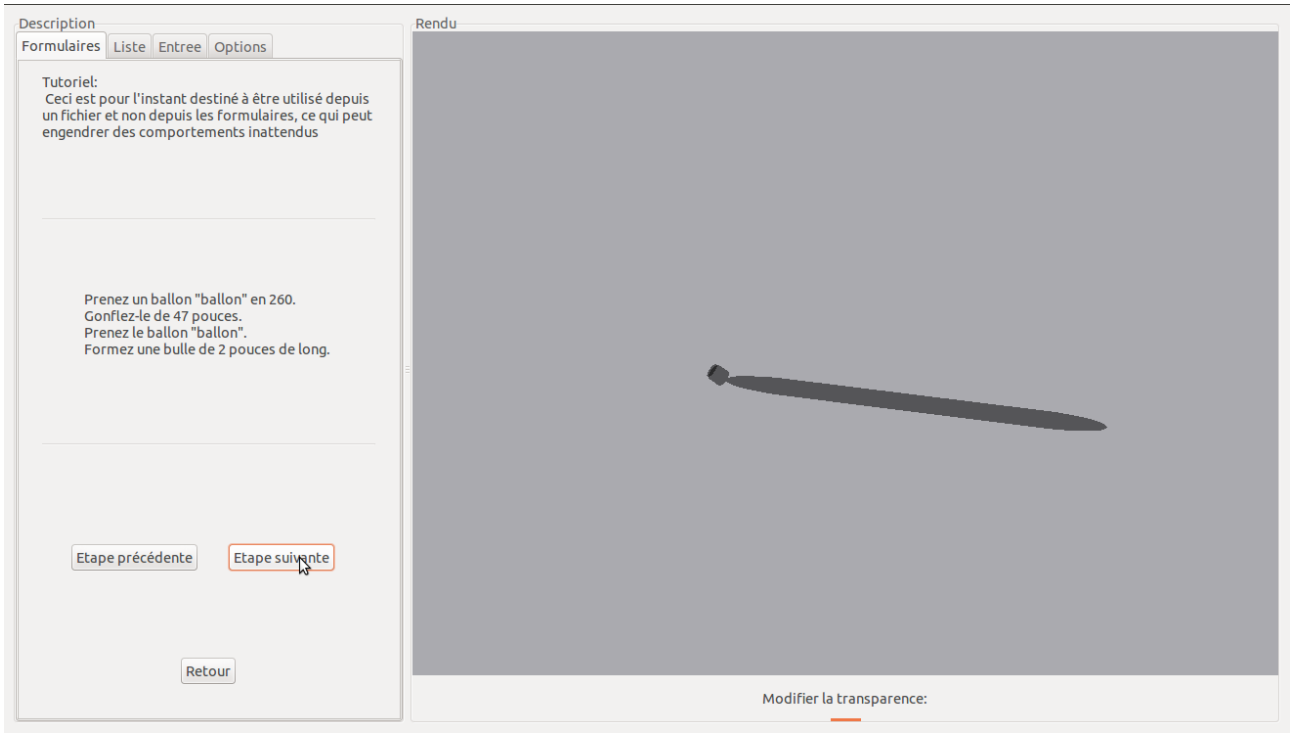


FIGURE 26 – Tutoriel pour une épée : étape 3

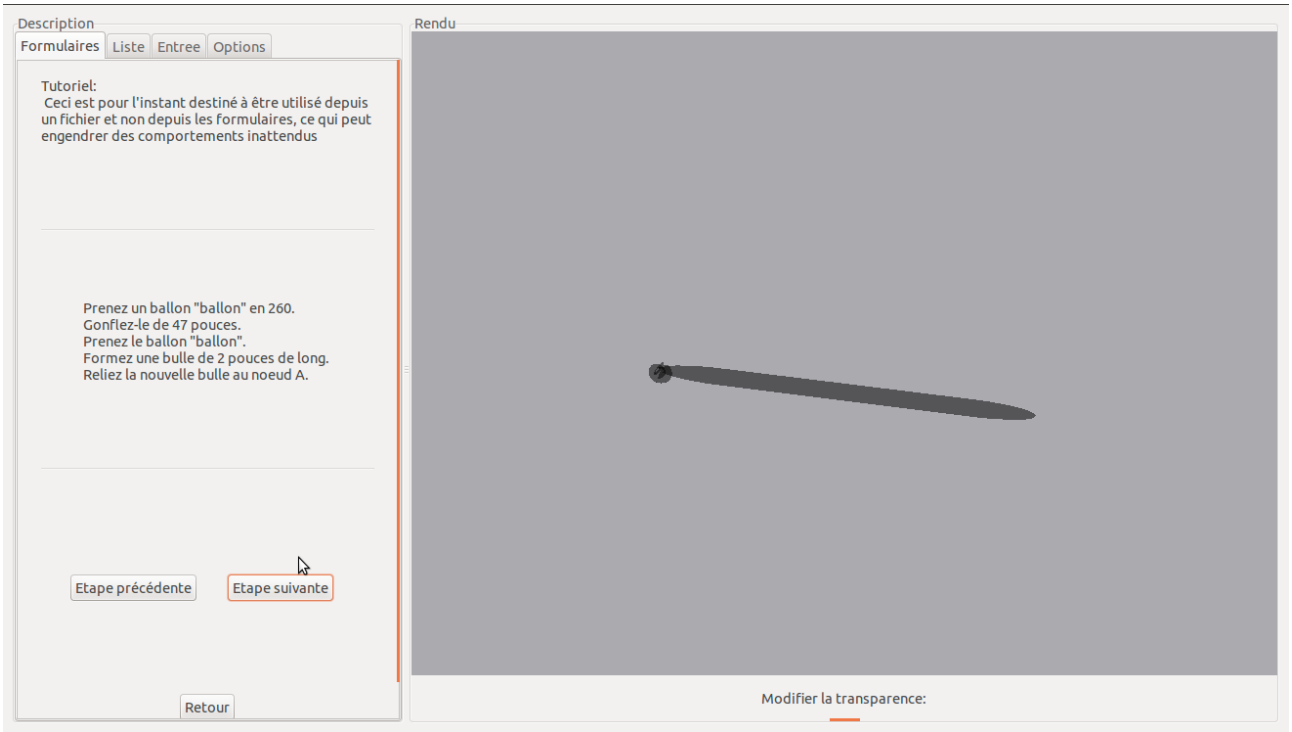


FIGURE 27 – Tutoriel pour une épée : étape 4

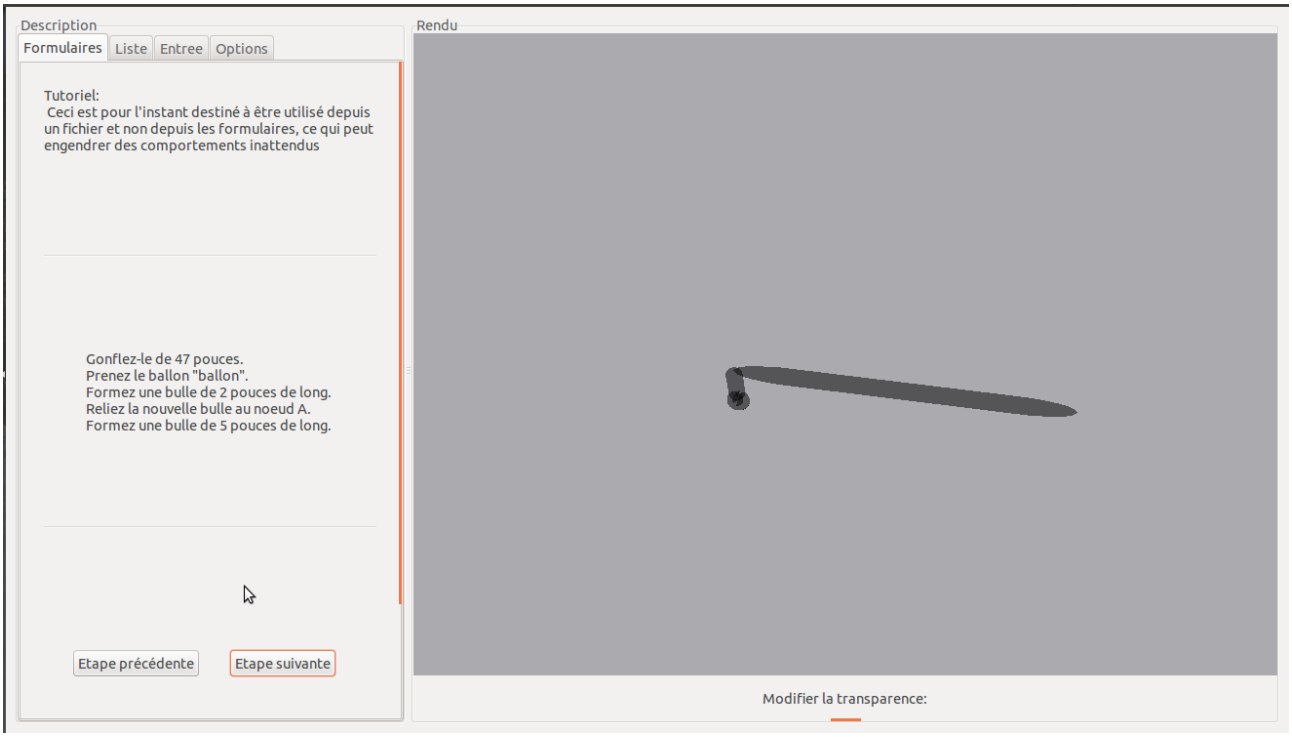


FIGURE 28 – Tutoriel pour une épée : étape 5

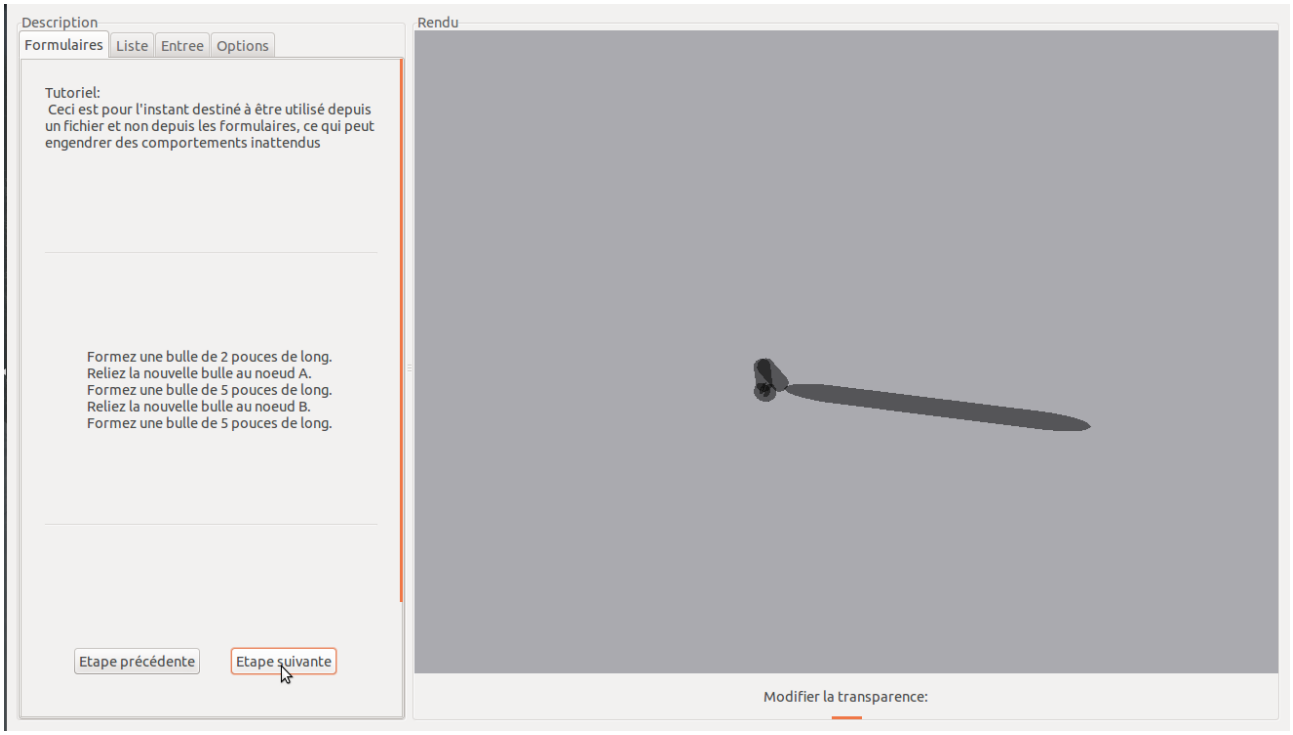


FIGURE 29 – Tutoriel pour une épée : étape 6

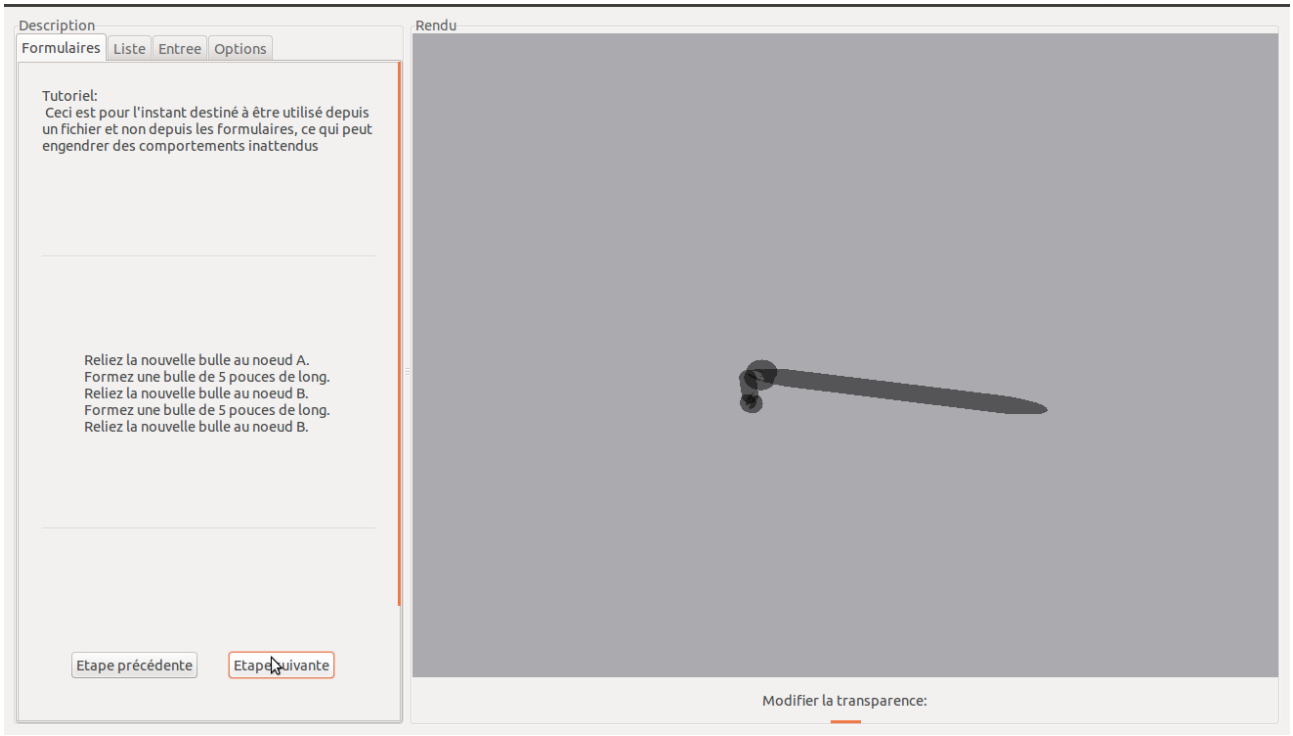


FIGURE 30 – Tutoriel pour une épée : étape 7

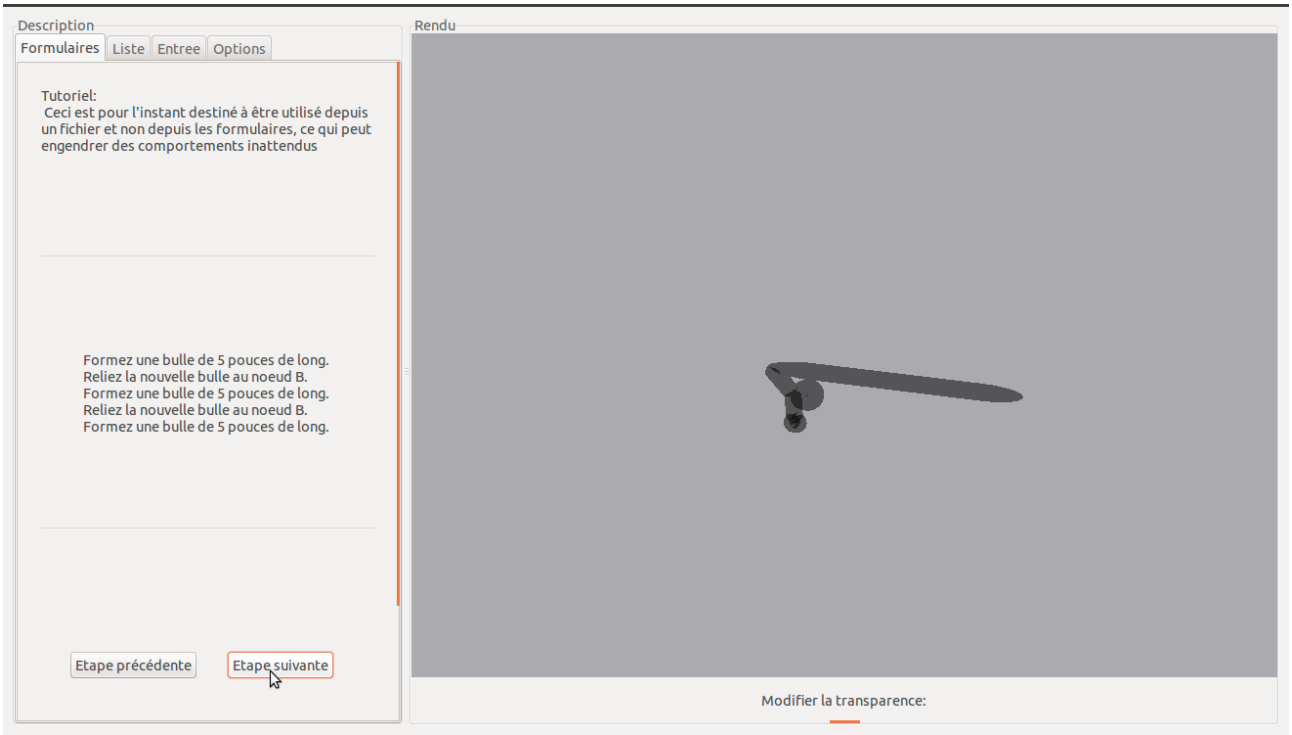


FIGURE 31 – Tutoriel pour une épée : étape 8

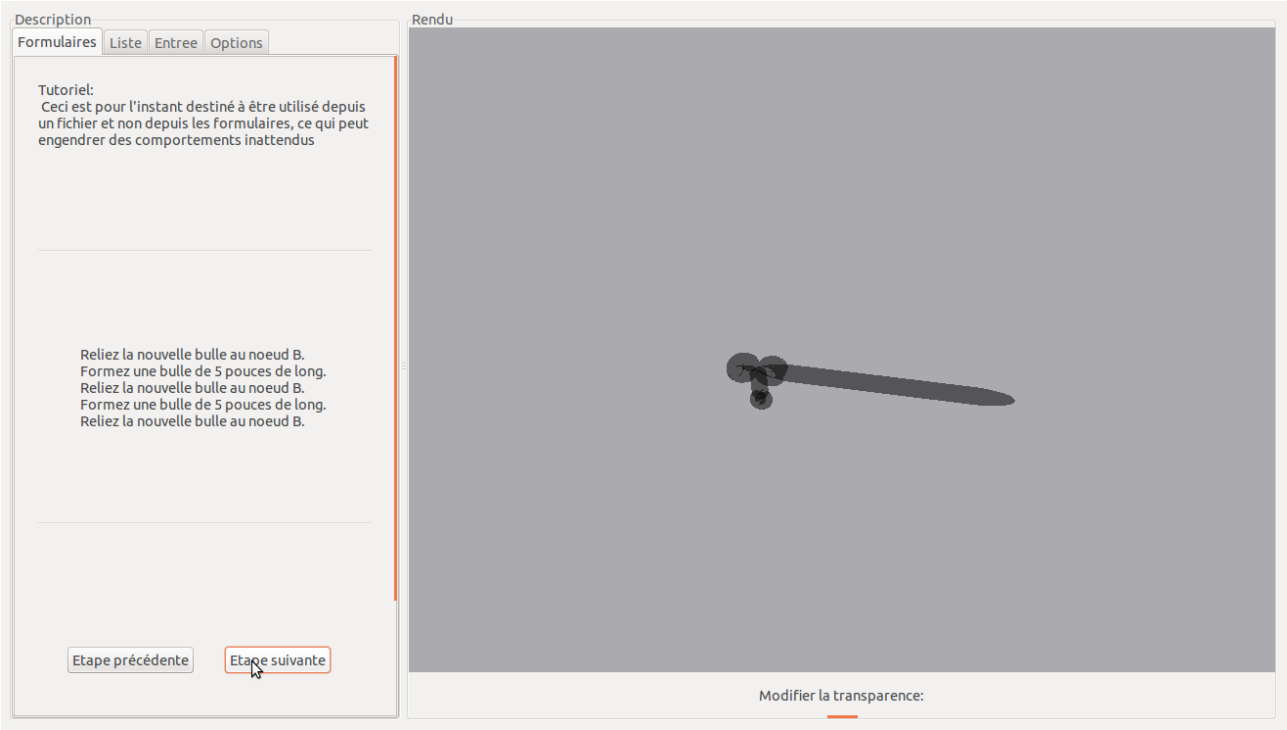


FIGURE 32 – Tutoriel pour une épée : étape 9

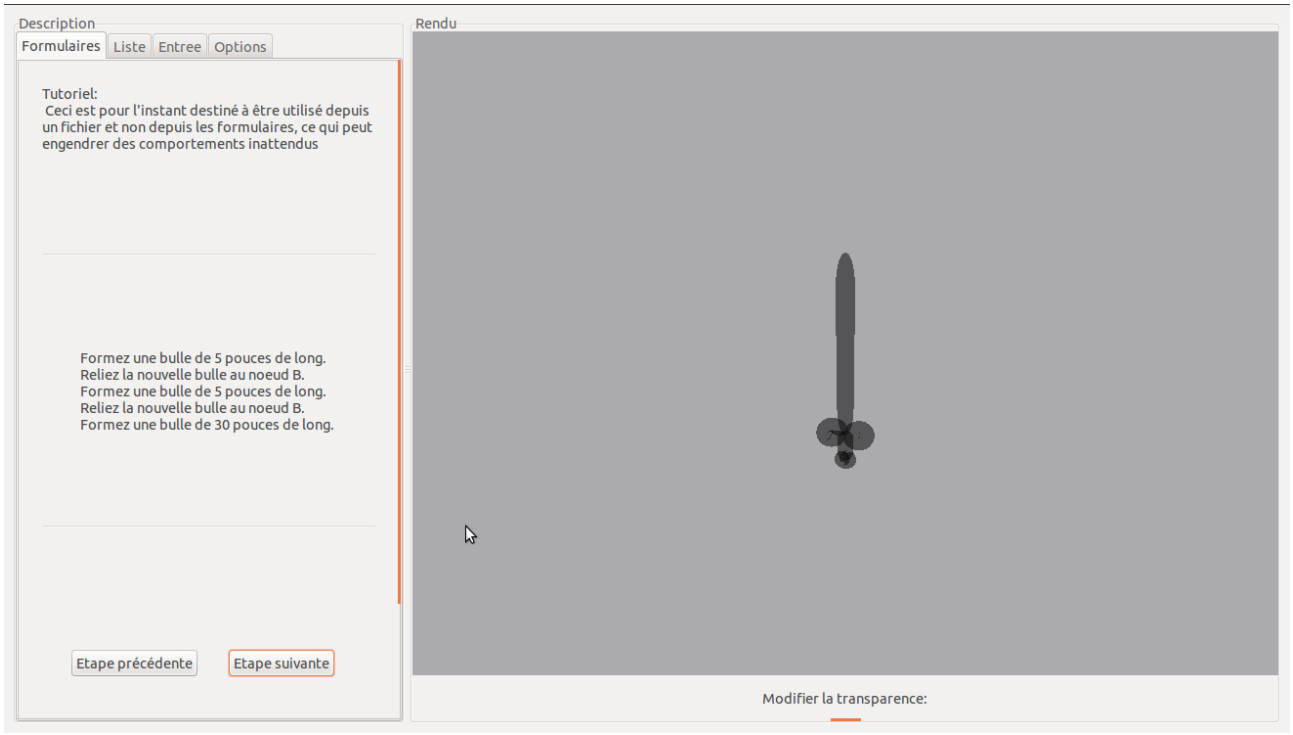


FIGURE 33 – Tutoriel pour une épée : étape 10

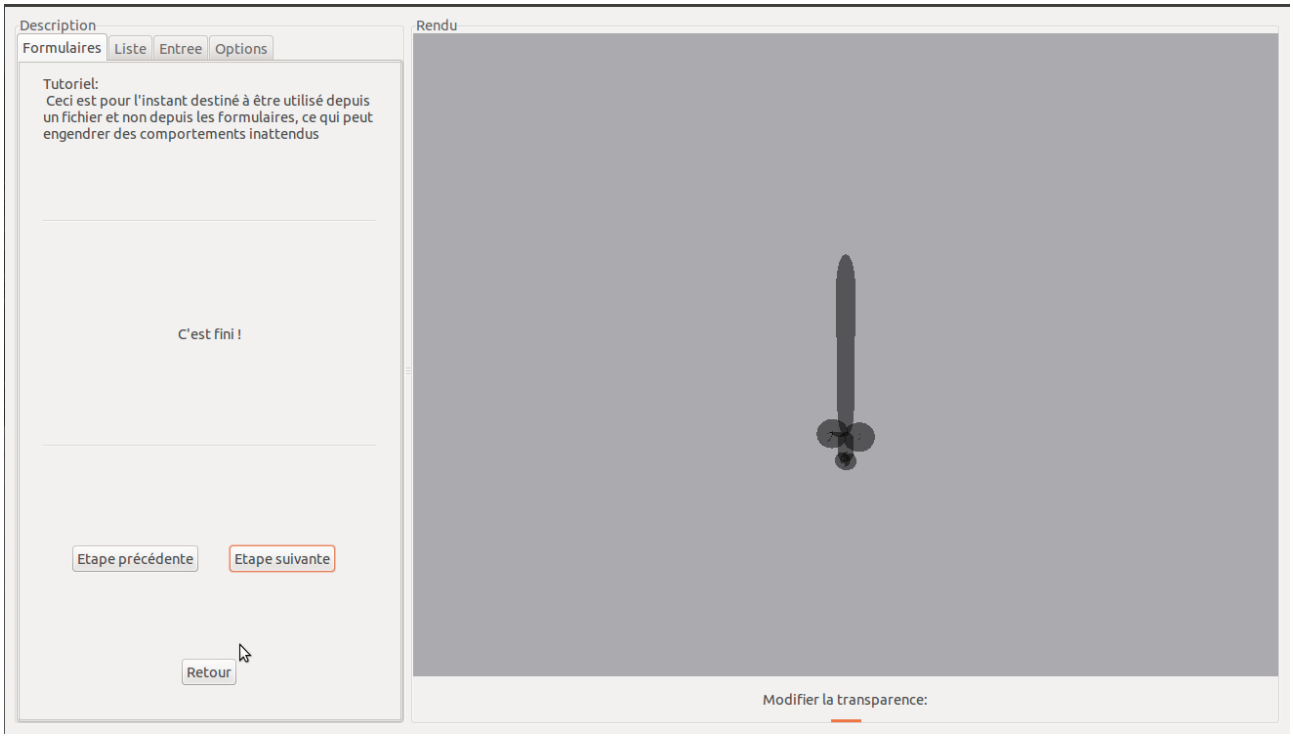


FIGURE 34 – Tutoriel pour une épée : étape finale

Quatrième partie

Suites

Le projet a rempli presque tous ses objectifs de départ. La création d'un mot à partir d'une création via l'interface cliquer-glisser n'est pas encore gérée. Mais cela pourra se faire si le projet continue.

Quelques soucis doivent pouvoir se régler dans les semaines à venir. La compilation sous windows pose pour le moment problème mais nous espérons pouvoir trouver une solution et ainsi permettre au grand public de tester le logiciel. Les boucles ne sont pas réellement bien gérées ; les bulles se placent correctement grâce aux orientations définies manuellement mais le logiciel ne calcule pas lui-même les positions correctes. Ce calcul peut se faire et être inclus au calcul de la position des nœuds mais cette étape peut prendre du temps.

Enfin le logiciel est actuellement en français avec interface cliquer-glisser en anglais. Nous voulons pouvoir proposer les deux langues. Ce n'est pas compliqué mais cela prend du temps. Aussi cet aspect du logiciel a-t-il été laissé de côté. Nous avons favorisé la finalisation des formulaires, qui elle aussi prenait du temps, à ces deux derniers points.

L'ambition de partir d'une image réelle pour reconstruire l'objet-ballon associé via notre modélisation a été laissée de côté mais c'est une évolution possible du logiciel. Maintenant que les outils de base ont été mis en place, il est plus facile d'imaginer comment parvenir à cela. On peut passer par l'analyse d'une image 3D codée dans un format courant (IGES, VRML ou X3D par exemple) et mettre en place un algorithme pour y reconnaître des bulles ou des formes pouvant être simplifiées en une bulle. On se ramènerait alors à la situation du cliquer-glisser : les bulles existent et il doit être possible de calculer un mot qui leur est associé et donc de reconstruire la sculpture.

C'est un problème complexe et nous n'avons pas la prétention de le résoudre dans un avenir proche mais cela peut s'envisager pour la suite.

Ce projet continue. S'il vous intéresse, n'hésitez pas à envoyer un mail à `cybloon@gmail.com`.