

Projet de Modélisation et Ingénierie du vivant

Quentin AUGRAIN, Florent MALLARD

INSA de Rennes
5INFO

25 janvier 2016

Table des matières

1	Travaux pratiques 1 : Session tutorielle	3
1.1	Exploration du projet	3
1.2	Simulation physique	3
1.3	Conclusion	8
2	Travaux pratiques 2	9
2.1	Interaction	9
2.2	Moniteur haptique	11
2.3	Manipulation du foie	13
3	Travaux Pratiques 3	15
3.1	Manipulation du foie	15
3.2	Améliorer le contrôle haptique	17
3.3	Introduction de la reconnaissance de gestes	18
4	Travaux pratiques 4 : Personnalisation	24

1 Travaux pratiques 1 : Session tutorielle

1.1 Exploration du projet

Question 1 : Comme indiqué dans les commentaires de la fonction, il y a deux fonctions dans la boucle :

- Bouger le monde ;
- Afficher le monde.

Cela signifie que la fonction va tout d'abord calculer le prochain mouvement, pour ensuite afficher la nouvelle position. Pour le moment, la phase de calcul n'est pas implémentée et rien ne bouge dans la simulation.

1.2 Simulation physique

Question 2 : Nous ajoutons la gravité au « force_accumulator », et obtenons le résultat montré à la FIGURE 1, c'est-à-dire que rien ne se passe. En effet même si nous calculons maintenant les forces, nous ne les appliquons pas au mesh, ce qui résulte en un mesh toujours fixe. Il nous faut donc appliquer ces forces à chaque particule du mesh afin qu'un effet se fasse ressentir.

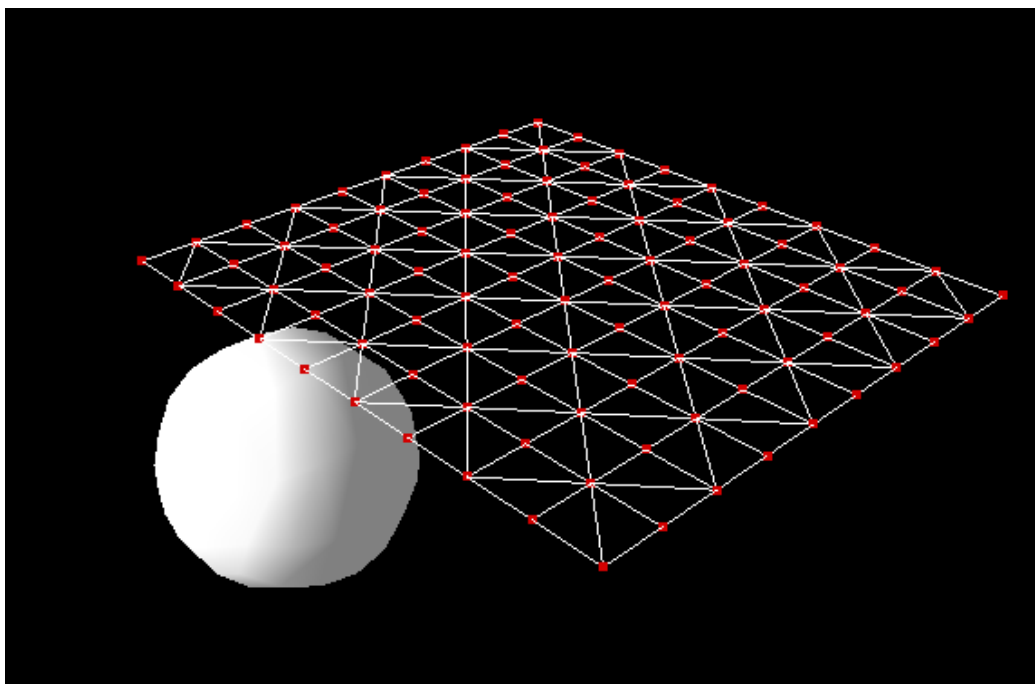


FIGURE 1 – Aucun changement n'est visible même si les forces changent.

Question 3 : Maintenant que nous appliquons les forces, le mesh tombe et finit par disparaître de l'écran. Il semble qu'il n'y ait pas de limite physique à sa chute. On peut donc envisager un moyen afin que même si aucune autre interaction que la gravité ne s'applique au mesh, il reste dans le champ de l'écran.

Question 4 : Une autre solution a été proposée, il s'agit de fixer le premier rang des particules. Ainsi, le mesh reste visible et a un effet dit « de rideau ». Pour ce faire, il nous faut nullifier les forces de ces particules. Nous avons compté dix particules sur ce rang, nous avons donc ajouté une boucle comme montré dans le code ci-dessous. Cette boucle met simplement les forces appliquées à ces dix particules à zéro, empêchant ainsi tout mouvement.

```
1 void Simulator::Update()
2 {
3     // Define the simulation loop (the methods are not in order)
4     //ApplyVelocityDamping( ... )
5
6     ComputeForces();
7
8     //fix first row of particules to simulate hanging
9     for (int i = 0; i < 10; i++)
10    {
11        m_Mesh->particles[i].force_accumulator = Maths::Vector3(0, 0, 0);
12    }
13
14    Integrate();
15
16    //UpdateManipulator();
17 }
```

Question 5 : Le comportement est pour le moins étrange, car les particules oscillent de plus en plus jusqu'à disparaître. Pour le moment la simulation n'est pas stable, car les forces appliquées aux particules ne se compensent pas. Ceci aboutit, plus ou moins rapidement en fonction des paramètres de simulation choisis, à un écran contenant uniquement le premier rang de particules, demeuré fixe comme précisé à la question précédente. Nous donnons ci-après notre implémentation.

```
1 void Simulator::ComputeForces()
2 {
3     for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
4     {
5         Particle *particle = &(m_Mesh->particles[p]);
6     }
```

```

7 // gravity
8 particle->force_accumulator = Maths::Vector3(0, -GRAVITY_CONSTANT, 0);
9
10 // neighbors forces
11 for (unsigned int n = 0; n < particle->neighbors.size(); n++)
12 {
13     Particle *neighbor = particle->neighbors[n];
14
15     float dij = particle->pos.distance(neighbor->pos);
16     float dij_init = particle->init_pos.distance(neighbor->init_pos);
17
18     Maths::Vector3 diff_pos = neighbor->pos - particle->pos;
19     diff_pos.normalise();
20
21     particle->force_accumulator += diff_pos * K * (dij - dij_init);
22 }
23 }
24 }

```

Question 6 : Le paramètre « dt » définit la vitesse de la simulation à travers le nombre d'itérations à chaque fois. Une valeur plus petite correspond donc à une simulation plus rapide, car on calcule la position et les forces après un instant plus grand, le mouvement a donc été plus important. K définit la rigidité. Celle-ci va aider à obtenir l'effet de rideau désiré. Sachant cela, nous n'avons pas réussi à stabiliser la simulation. En effet même si le rideau est considéré comme plus rigide, les forces des particules ne se compensent pas et donnent le même résultat que précédemment.

Question 7 : Nous ajoutons la boucle demandée qui effectue n appels à « Update » avec un timestep désormais divisé par n . Le but de cette manipulation est d'avoir un mouvement plus petit, car calculé sur un plus petit pas de temps. Grâce à cela les interactions calculées sont plus fines et donc plus réalistes. Bien entendu, cela implique aussi plus de calculs, et donc un coût plus important pour le processeur. Le CPU est bien plus actif après que nous ayons implémenté cette modification.

```

1 void Simulator::Update()
2 {
3     // Define the simulation loop (the methods are not in order)
4     for(int j = 0; j < nb_interactions; j++){
5         //ApplyVelocityDamping( ... )
6
7         ComputeForces();
8
9         //fix first row of particules to simulate hanging

```

```

10 for (int i = 0; i < 10; i++)
11 {
12     m_Mesh->particles[i].force_accumulator = Maths::Vector3(0, 0, 0);
13 }
14
15 Integrate();
16
17 //UpdateManipulator();
18 }
19 }

```

Question 8 : En amortissant la chute des particules, nous arrivons à stabiliser la simulation. L'amortissement simule l'interaction des particules avec l'air ambiant. C'est lui qui va réussir à compenser les interactions des particules entre elles et empêcher une oscillation de plus en plus importante comme observé précédemment.

```

1 void Simulator::ApplyVelocityDamping()
2 {
3     for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
4     {
5         Particle *particle = &(m_Mesh->particles[p]);
6
7         for (unsigned int n = 0; n < particle->neighbors.size(); n++)
8         {
9             Particle *neighbor = particle->neighbors[n];
10
11             Maths::Vector3 vrel = neighbor->vel - particle->vel;
12
13             Maths::Vector3 diff_pos = particle->pos - neighbor->pos;
14             diff_pos.normalise();
15
16             particle->force_accumulator += diff_pos * (diff_pos.dotProduct(vrel) * D);
17         }
18
19         //add some friction
20         particle->force_accumulator += -D * particle->vel;
21     }
22 }

```

Question 9 : Avec $dt = 1/200$, $K = 300$, $D = 40$ et $n = 20$, nous arrivons à une simulation stable. Mais ces paramètres sont très coûteux en ressources car avec $dt = 1 / 200$ et $n = 20$, cela signifie que nous effectuons quatre mille fois la boucle « Update » par seconde. Plus tard, et sur des ordinateurs moins

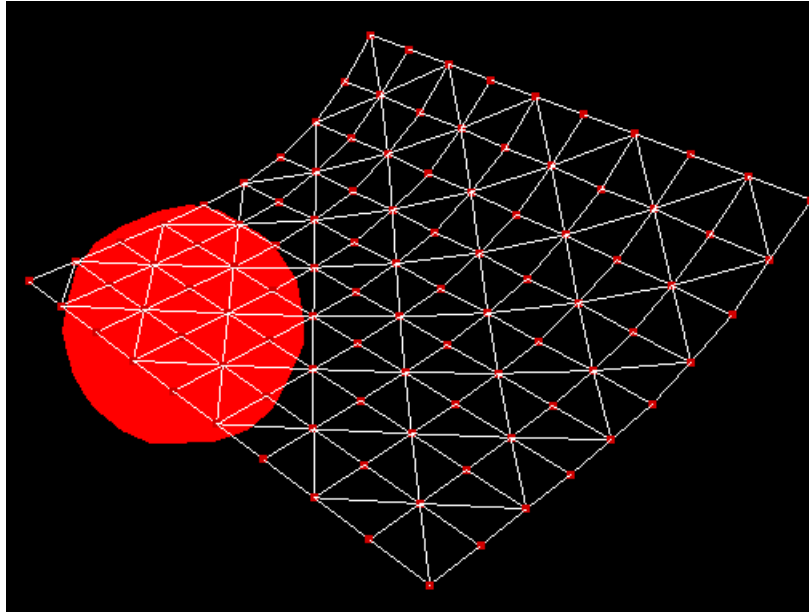


FIGURE 2 – Le mesh pendant la chute.

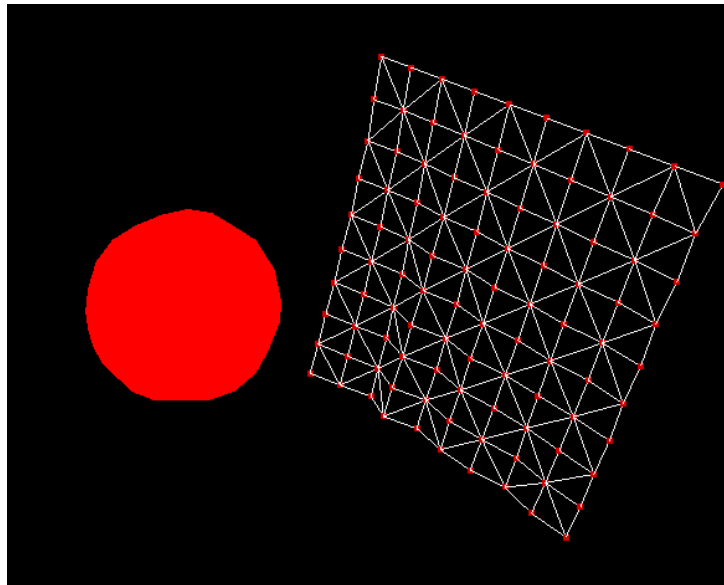


FIGURE 3 – Le mesh stabilisé.

puissants, nous préférons utiliser un dt de $1/50$. Cela n'affecte pas trop la simulation mais permet de diminuer significativement le coût en ressources.

1.3 Conclusion

Question 10 : Le modèle de Mass-Spring-Damper est assez simple à utiliser dès lors que les formules à appliquer sont à disposition. En revanche, pour obtenir une simulation stable, le moindre écart dans les paramètres peut créer une simulation tout à fait exotique. En effet nous avons vu que diminuer la rigidité du mesh pouvait conduire à une simulation qui n'était pas réaliste, car le mesh se comportait plus comme une plaque pivotante que comme un rideau. Appliquer un amortissement trop fort rend la simulation trop longue pour être réaliste.

2 Travaux pratiques 2

2.1 Interaction

Question 1 : Baisser la valeur d'amortissement résulte en une accélération de la chute et en une baisse de la stabilité. En effet, l'amortissement sert à compenser les interactions entre les particules, qui font osciller le mesh, et à simuler une interaction avec l'air. Diminuer ces « frottements » accélère donc la chute du rideau, et les particules ont des interactions plus fortes qui déstabilisent la simulation. Après quelques recherches, la vitesse relative est simplement une soustraction des vecteurs vitesses du voisin par rapport à la particule.

Question 2 : Afin de simuler le sol, nous implémentons une limite de hauteur en dessous de laquelle les particules ne peuvent descendre, ayant pour effet de simuler un plan. Cette méthode nous a posé un petit problème lorsque les particules descendaient un peu plus bas que le seuil fixé, car nous supprimions alors toutes les forces sur les particules, et le manipulateur n'interagissait plus avec.

```
1 // remove all forces on y axis starting y = 0 and below to simulate floor
2 for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
3 {
4     if (m_Mesh->particles[p].pos.y <= 0)
5     {
6         m_Mesh->particles[p].force_accumulator.y = 0;
7         m_Mesh->particles[p].vel.y = 0;
8     }
9 }
```

Question 3 : Afin de ne pas passer trop de temps sur une question qui ne nous semblait pas essentielle, nous avons simplement changé la couleur de notre manipulateur, que nous montrons ci-dessous :

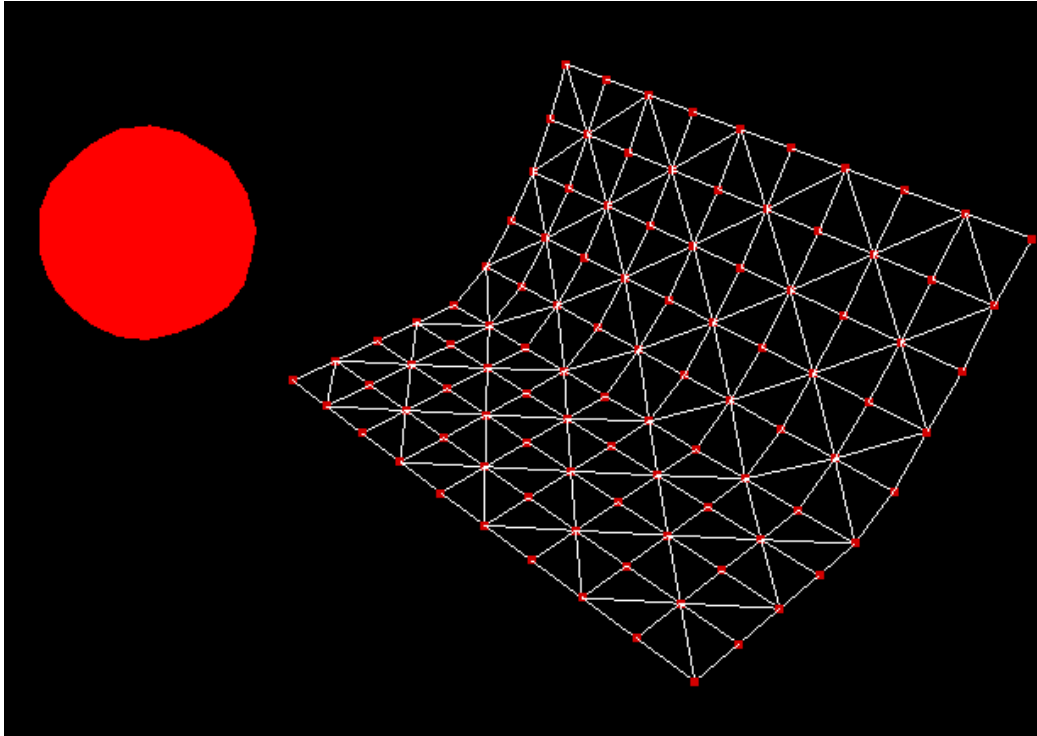


FIGURE 4 – Rideau tombé sur le sol.

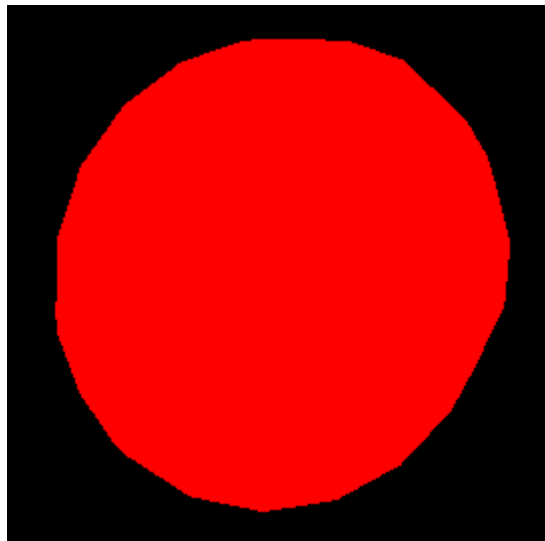


FIGURE 5 – Le manipulateur est maintenant de couleur rouge.

Question 4 : Le coefficient C , s'il est négatif, crée une attraction entre le manipulateur et le mesh. En revanche, s'il est positif, les deux objets se

repoussent. Celui-ci peut permettre de simuler les propriétés adhérentes ou répulsives (à cause du contact) de différents outils qui pourraient être utilisés en chirurgie. Avec un coefficient assez fort, les particules ont un mouvement de rebond, car elles sont repoussées loin de la boule à son contact, puis reviennent à leur position initiale et ainsi de suite. Il faut donc être modéré dans les valeurs que nous donnons au coefficient.

```

1 void Simulator::UpdateManipulator()
2 {
3     if(m_Manipulator)
4     {
5         Maths::Vector3 manipulator_pos = m_Manipulator->getPosition();
6         Maths::Real manipulator_radius = m_Manipulator->getRadius();
7
8         for (unsigned int p = 0 ; p < m_Mesh->particles.size() ; p++)
9         {
10             //repulsion of the sphere
11             Maths::Vector3 m_force = Maths::Vector3(0.0f, 0.0f, 0.0f);
12
13             if (m_Mesh->particles[p].pos.distance(manipulator_pos) < manipulator_radius
14                 && !m_Mesh->particles[p].fixed)
15             {
16                 //m_Mesh->particles[p].fixed = true;
17
18                 m_force = m_Mesh->particles[p].pos - manipulator_pos;
19                 m_force.normalise();
20                 m_Mesh->particles[p].pos = manipulator_pos + m_force * manipulator_radius
21                     * penalty;
22
23                 //impose velocity ZERO perpendicular to the sphere surface
24                 m_Mesh->particles[p].vel -=
25                     m_force*m_Mesh->particles[p].vel.dotProduct(m_force);
26             }
27         }
28     }
29 }

```

2.2 Moniteur haptique

Question 5 : Nous devons effectuer des modifications afin de rendre possibles les manipulations avec le client haptique. Ainsi, nous avons choisi d'associer la position du manipulateur à celle du client haptique. De cette manière, les mouvements du manipulateur sont limités, car le bras articulé l'est lui-même, et on a un contrôle très instinctif.

```
1 manipulator.setPosition(haptic_client.getPosition());
```

Question 6 : Pour améliorer le réalisme, un retour de force est rendu possible par le client haptique. Cette fonction se révèle très utile afin de simuler la résistance que l'on peut rencontrer lors d'une manipulation, surtout dans un domaine comme la chirurgie.

```
1 //Update haptic simulation here!!
2 for (unsigned int p = 0; p < mesh.particles.size(); p++)
3 {
4     if (mesh.particles[p].pos.distance(manipulator.getPosition()) <
5         manipulator.getRadius())
6     {
7         retour += mesh.particles[p].pos - manipulator.getPosition();
8     }
9 }
10 retour = -retour;
11 haptic_client.setForce(retour/4);
```

Question 7 : Le client haptique dispose de quatre boutons, pouvant être associés aux fonctions de notre choix. Pour pouvoir utiliser ces boutons, une fonction « HapticButtonClicked » a été mise à notre disposition afin de la compléter. Son rôle se résume pour le moment à détecter lorsqu'on appuie sur un des boutons du client haptique. Cette détection se fait grâce au code suivant.

```
1 //haptic buttons handling
2 if (haptic_client.isButtonPressed(0)){
3     //simulator.fixParticles();
4     std::cout << "Simulation paused";
5     paused = true;
6 }
7 if (haptic_client.isButtonPressed(1)){
8     std::cout << "Simulation unpaused " ;
9     paused = false;
10 }
11 if (haptic_client.isButtonPressed(2)){
12     std::cout << "Manipulation stopped" ;
13     move_mode = MOUSE;
14 }
15 if (haptic_client.isButtonPressed(3)){
16     std::cout << "Manipulation started";
17     move_mode = HAPTIC;
18 }
```

2.3 Manipulation du foie

Question 9 : Après changement du mesh, l'aspect de la simulation est donné dans la FIGURE 6. Après quelques changements dans les paramètres de simulation, nous décidons que la simulation est stable avec les mêmes valeurs que précédemment. Pour rappel, celles-ci étaient : $dt = 1/50$, $K = 300$, $D = 40$ et $n = 20$.

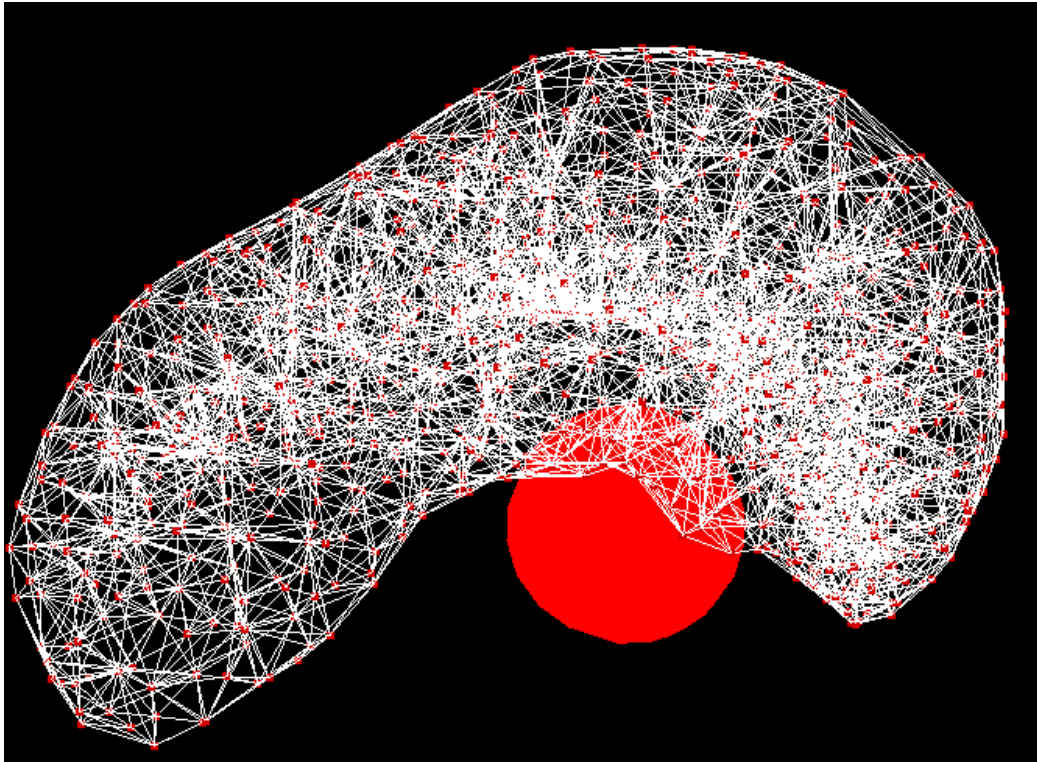


FIGURE 6 – Mesh représentant un foie.

Question 10 : Dans un premier temps, nous avons fixé tous les points en contact avec le manipulateur, à l'appui sur la touche « f ». Dans un second temps, nous avons cherché les positions des sphères qui avaient un rendu que nous avons jugé correct au lancement de la simulation. Dans un souci de visibilité, les sphères « de fixation » n'ont pas été dessinées.

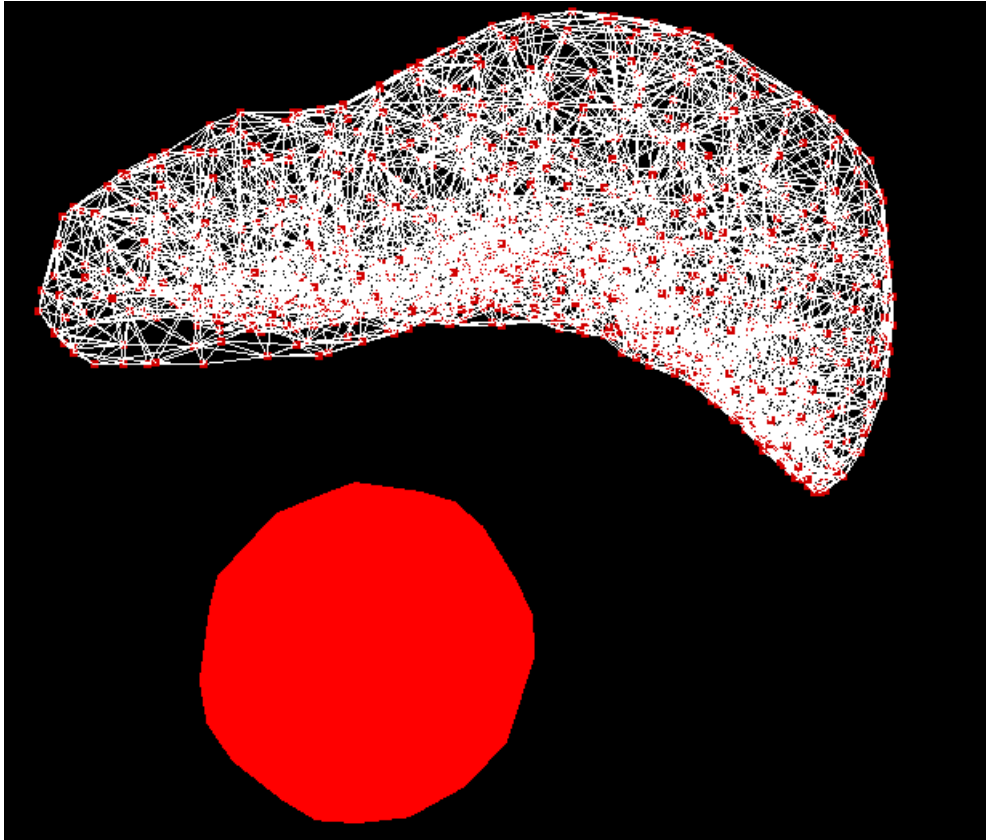


FIGURE 7 – Le mesh du foie fixé.

```

1 void Simulator::fixParticlesinSphere(RigidSphere* sphere)
2 {
3     Maths::Vector3 sphere_pos = sphere->getPosition();
4     Maths::Real sphere_radius = sphere->getRadius();
5
6     for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
7     {
8         if (m_Mesh->particles[p].pos.distance(sphere_pos) < sphere_radius)
9         {
10             m_Mesh->particles[p].fixed = true;
11         }
12     }
13 }

```

Question 11 : Afin d'atteindre les objectifs fixés, nous avons fixé une dizaine de points, un time step de $1/50$ e de seconde, et 20 boucles avant d'afficher le résultat. En somme nous avons gardé les paramètres trouvés auparavant.

3 Travaux Pratiques 3

3.1 Manipulation du foie

Question 1 : Dans un premier temps, nous avons associé au bouton 0 du client haptique la fixation des particules. Nous n'avons pas noté de changement particulier au niveau des performances. Nous avons ensuite délaissé cette fonction au profit de la pause dans la manipulation.

Question 2 : Afin de mieux repérer les particules fixées, nous avons décidé de les colorer en bleu. Le résultat de cette modification est montré dans la FIGURE 8. Nous avons choisi le bleu car, selon nous, c'est une couleur qui se voyait suffisamment facilement et qui n'était pas associée à un état négatif.

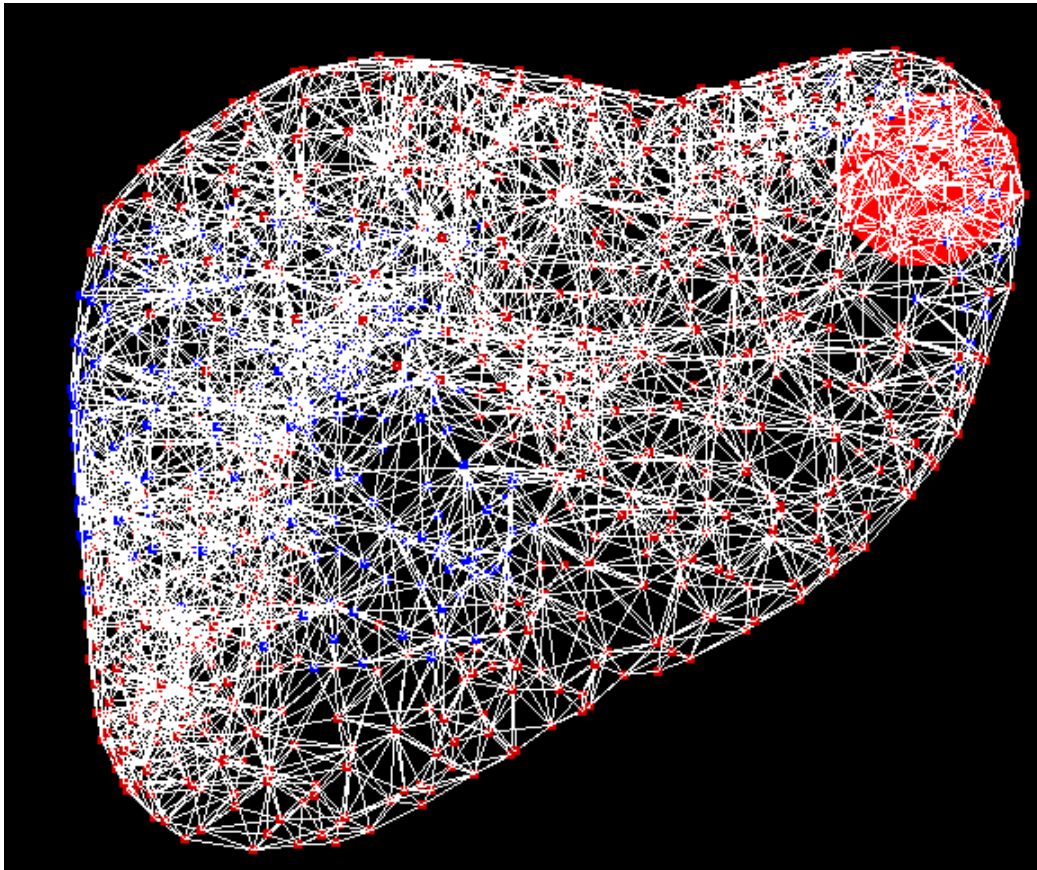


FIGURE 8 – Coloration des points fixés.

Question 3 : Nous avons choisi de fixer des points à l'intérieur du foie, afin que la simulation soit plus réaliste. En effet, de cette manière nous préservons la « texture » du foie en autorisant la partie supérieure à bouger, comme si le foie était posé. Nous avons également souhaité conserver les lobes du foie afin de pouvoir les soulever. Ceci rend la manipulation facile tout en gardant un réalisme dans la simulation, ce qui est le but final de ce projet.

Question 4 : Nous avons choisi de sauvegarder les particules fixées, plus particulièrement leur numéro dans le tableau des particules du mesh, dans un fichier de type texte, nommé « save ». Si ce fichier existe déjà, il est chargé au lancement du programme. Celui-ci est écrasé à chaque appui sur la touche « s », ne laissant la possibilité que d'avoir une seule sauvegarde, à moins de la renommer manuellement. La fonction de sauvegarde telle que nous l'avons implémentée est donnée ci-dessous.

```
1 void Simulator::saveFixedParticles()
2 {
3     std::ofstream f;
4     f.open("save.txt");
5
6     if (f.is_open()){
7         for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
8         {
9             if (m_Mesh->particles[p].fixed)
10            {
11                f << p << "\n";
12            }
13        }
14    }
15    f.close();
16 }
```

Au chargement, nous cherchons ce fichier de sauvegarde. S'il n'existe pas à l'endroit où il est sauvegardé initialement, alors aucune particule n'est fixée et la simulation commence. En revanche, dans le cas contraire nous parcourons le fichier et fixons les particules dont le numéro apparaît. Le code suivant remplit ce rôle.


```

1 void Simulator::restoreFixedParticles()
2 {
3     std::ifstream f("save.txt");
4
5     if (f)
6     {
7         std::string line;
8
9         while (std::getline(f, line))
10        {
11            m_Mesh->particles[atoi(line.c_str())].fixed = true;
12        }
13    }
14    f.close();
15 }

```

Cette méthode a nécessité l'ajout d'une autre fonction, car le mesh restauré n'était plus accessible avec le manipulateur, car il était trop haut. Cette fonction « drop_mesh » est donnée ci-dessous.

```

1 void Simulator::dropMesh(float offset)
2 {
3     for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
4     {
5         m_Mesh->particles[p].pos.y -= offset;
6     }
7 }

```

3.2 Améliorer le contrôle haptique

Question 5 : Afin de pouvoir alterner entre l'utilisation de la souris et le moniteur haptique, nous avons choisi d'ajouter une variable à notre code. Celle-ci peut prendre deux valeurs qui correspondent aux deux modes possibles. Ainsi, lors de l'appui sur la touche « m », on change le mode courant. Cet ajout s'accompagne bien sûr de vérifications de sa valeur afin d'exécuter la partie de code correspondant au bon mode.

```

#define HAPTIC 1
#define MOUSE 0

```

FIGURE 9 – Les deux modes possibles sont des entiers.

```

1 if (!paused){
2     //Update the simulation
3     simulator.Update();
4
5     if (move_mode == HAPTIC){
6
7         manipulator.Move((last_haptic_x - haptic_client.getPosition()[0]) /
8             scale_factor,
9             (last_haptic_y - haptic_client.getPosition()[1]) / scale_factor,
10            (last_haptic_z - haptic_client.getPosition()[2]) / scale_factor);
11    }
12 }
13
14 last_haptic_x = haptic_client.getPosition()[0];
15 last_haptic_y = haptic_client.getPosition()[1];
16 last_haptic_z = haptic_client.getPosition()[2];

```

Question 6 : Dans le but de contrôler avec le plus de précision possible le manipulateur, l'implémentation d'un facteur qui jouerait sur la sensibilité des manipulations était demandée. Ce facteur, nommé « `scale_factor` », peut être changé dynamiquement. Nous avons choisi les touches « i » et « d » afin d'effectuer les modifications. Le code suivant montre la manière dont nous avons implémenté cette amélioration. Nous pouvons y voir que nous avons choisi un pas d'une valeur de 0.2.

```

1 if (key == 'i'){
2     scale_factor += 0.2;
3 }
4 if (key == 'd'){
5     scale_factor -= 0.2;
6 }

```

Question 8 : Nous avons choisi d'appuyer sur un des boutons du bras articulé afin de désactiver le mouvement du manipulateur. De cette manière nous pouvons remettre le bras à la position désirée avant de reprendre la manipulation.

3.3 Introduction de la reconnaissance de gestes

Question 9 : Après ajout du Leap motion, nous pouvons visualiser la représentation de notre main à l'écran. Pour le moment, certains gestes sont

reconnus, mais ne sont associés à aucune action. Pour le moment, les mouvements reconnus sont :

- Le pincement entre le pouce et l'index ;
- Le balayage de la main.

Par la suite nous déciderons d'ajouter le pincement entre le pouce et le majeur.

```
1 void leapCheckPinchGesture()  
2 {  
3     if (!listener.isHandVisible()) return;  
4  
5     Vector3 thumb = listener.getFingerTipPosition(0);  
6     Vector3 index = listener.getFingerTipPosition(1);  
7  
8     float dist = thumb.distance(index);  
9  
10    if (dist < 0.3f)  
11    {  
12        std::cout << "Pinch between thumb and index detected!!! " << dist <<  
13            std::endl;  
14        Vector3 diff = thumbPos - thumb;  
15    }  
16 }
```

```
1 void leapCheckSwipeGesture()  
2 {  
3     float speed;  
4     Vector3 direction;  
5     int finger;  
6  
7     if (listener.isSwipe(speed, direction, finger))  
8     {  
9         std::cout << "[LeapMotion]" << " Swipe gesture(" << finger << ") : " << speed  
10            << ", " << direction << std::endl;  
11    }  
12 }
```

Question 10 : Nous avons choisi d'associer au pincement entre le pouce et l'index la translation du mesh actuel. Au pincement entre le pouce et le majeur, nous effectuons une rotation du mesh, et lors du balayage de la main, nous activons ou désactivons la manipulation avec le bras articulé. Nous avons hésité à ajouter le mode « scalpel » dans la rotation engendrée par le balayage, mais nous avons trouvé que la précision des mouvements et la qualité des manipulations pourrait s'en ressentir. En effet, si lors d'un balayage le scalpel est activé, il y a un risque que l'utilisateur coupe des liens sans le vouloir.

```

1 void Simulator::translateMesh(Maths::Vector3 t)
2 {
3     for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
4     {
5         m_Mesh->particles[p].pos -= t;
6     }
7 }
8
9
10 void Simulator::rotateMesh(Maths::Vector3 prevPos, Maths::Vector3 currPos)
11 {
12     // get rotation quaternion
13     Maths::Quaternion q;
14     q = prevPos.getRotationTo(currPos);
15
16     for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
17     {
18         m_Mesh->particles[p].pos = q * m_Mesh->particles[p].pos;
19     }
20 }

```

```

1 void leapCheckPinchGesture()
2 {
3     if (!listener.isHandVisible()) return;
4
5     Vector3 thumb = listener.getFingerTipPosition(0);
6     Vector3 index = listener.getFingerTipPosition(1);
7     Vector3 middle = listener.getFingerTipPosition(2);
8
9     float dist = thumb.distance(index);
10    float dist2 = thumb.distance(middle);
11
12    if (dist < 0.3f)
13    {
14        std::cout << "Pinch between thumb and index detected!!! " << dist <<
15            std::endl;
16        Vector3 diff = thumbPos - thumb;
17        thumbPos = thumb;
18        simulator.translateMesh(diff);
19    }
20    else if (dist2 < 0.3f)
21    {
22        std::cout << "Pinch between thumb and middle detected!!! " << dist <<
23            std::endl;
24        simulator.rotateMesh(thumbPos, thumb);
25        thumbPos = thumb;
26    }
27    else

```

```

26 {
27     thumbPos = thumb;
28 }
29 }

```

Pour le mode « scalpel », nous avons modélisé une sphère au bout de l'index. Celle-ci ne sert que pour des raisons de facilité d'implémentation et n'apparaît en aucun cas. Lorsque la sphère se trouve entre deux points, la liaison est rompue, comme montré dans la FIGURE 10.

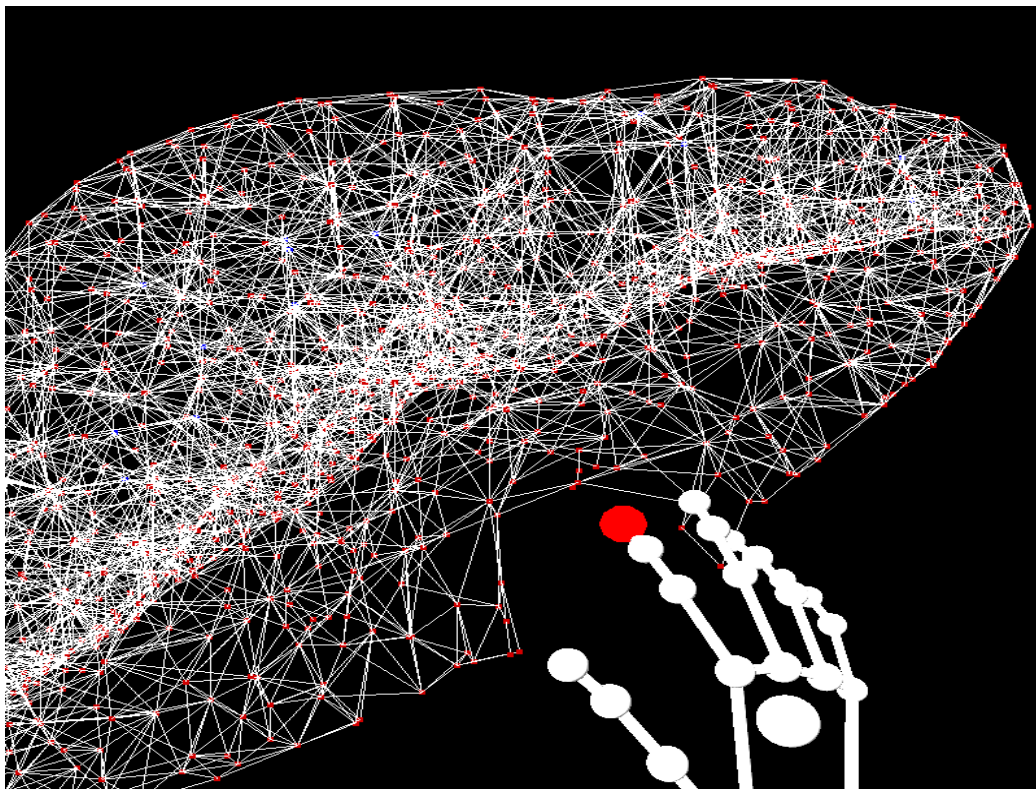


FIGURE 10 – Le mesh est découpé par le doigt avec une boule rouge au bout.

Afin de réaliser cette interaction avec le foie, nous avons recouru à plusieurs méthodes que nous donnons ci-dessous.

```

1 void Simulator::checkCut()
2 {
3     Maths::Vector3 scalpel_pos = m_Scalpel->getPosition();
4     Maths::Real scalpel_radius = m_Scalpel->getRadius();
5
6     for (unsigned int p = 0; p < m_Mesh->particles.size(); p++)
7     {

```

```

8 Particle *particle = &(m_Mesh->particles[p]);
9
10 for (unsigned int n = 0; n < particle->neighbors.size(); n++)
11 {
12     Particle *neighbor = particle->neighbors[n];
13
14     // distance between scalpel and vector between particle and neighbor
15     Maths::Vector3 link = Maths::Vector3(neighbor->pos.x - particle->pos.x,
16     neighbor->pos.y - particle->pos.y,
17     neighbor->pos.z - particle->pos.z);
18     Maths::Vector3 particle_scalpel = Maths::Vector3(particle->pos.x -
19     scalpel_pos.x,
20     particle->pos.y - scalpel_pos.y, particle->pos.z - scalpel_pos.z);
21     Maths::Vector3 neighbor_scalpel = Maths::Vector3(neighbor->pos.x -
22     scalpel_pos.x,
23     neighbor->pos.y - scalpel_pos.y, neighbor->pos.z - scalpel_pos.z);
24
25     Maths::Real distance = link.crossProduct(particle_scalpel).length() /
26     link.length();
27
28     if (distance < scalpel_radius && particle_scalpel.dotProduct(scalpel_pos) <
29     particle->pos.distance(neighbor->pos) &&
30     neighbor_scalpel.dotProduct(scalpel_pos) <
31     particle->pos.distance(neighbor->pos))
32     {
33         // remove particle from neighbor.neighbors and neighbor from
34         // particle.neighbors
35         CutLinks(particle, neighbor);
36     }
37 }
38 }
39 }
40
41 void Simulator::CutLinks(Particle* particle, Particle* neighbor)
42 {
43     for (std::vector<Particle*>::iterator iter = particle->neighbors.begin(); iter
44     != particle->neighbors.end(); ++iter)
45     {
46         if (*iter == neighbor)
47         {
48             particle->neighbors.erase(iter);
49             break;
50         }
51     }
52
53     for (std::vector<Particle*>::iterator iter = neighbor->neighbors.begin(); iter
54     != neighbor->neighbors.end(); ++iter)
55     {
56         if (*iter == particle)

```

```
48     {  
49         neighbor->neighbors.erase(iter);  
50         break;  
51     }  
52 }  
53 }
```

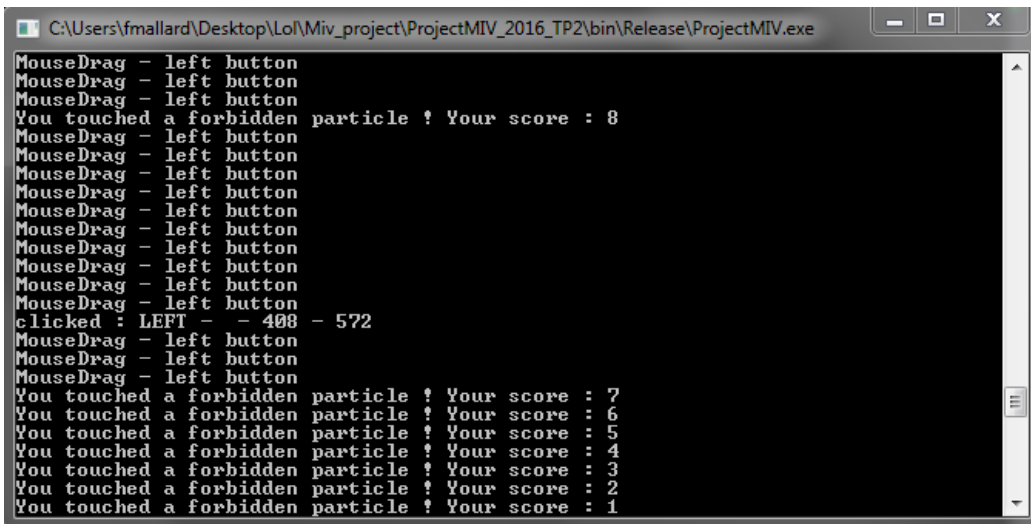
Dans un souci de visibilité, nous ne dessinons pas les particules qui ne sont plus reliées à aucun voisin.

4 Travaux pratiques 4 : Personnalisation

Afin de pouvoir évaluer l'utilisateur lors de ses manipulations, nous avons défini un critère de test simple : l'utilisateur n'a pas le droit de toucher de particules fixes. La simulation commence avec un score de 10. Celui-ci ne peut augmenter, en revanche il décroît unitairement lorsque la sphère touche une particule fixe. Dans le souci de rendre l'évaluation réaliste, et de ne pas mettre fin à la simulation au premier contact, nous laissons à l'utilisateur une « pause » de deux secondes durant lesquelles il ne peut plus perdre de points. Il a ainsi le temps de retirer la sphère de la zone et retrouver un espace autorisé.

```
1 void Simulator::dontTouchFixedparticles()
2 {
3     Maths::Vector3 manipulator_pos = m_Manipulator->getPosition();
4     Maths::Real manipulator_radius = m_Manipulator->getRadius();
5
6     t = time(0); // current time
7
8     if (t - time_up > 2){
9         for (unsigned int p = 0; p < m_Mesh->particles.size(); p++){
10             {
11                 if (m_Mesh->particles[p].pos.distance(manipulator_pos) < manipulator_radius
12                     && m_Mesh->particles[p].fixed)
13                 {
14                     score--;
15                     time_up = t;
16                     std::cout << "You touched a forbidden particle ! Your score : " << score
17                         << "\n";
18                 }
19             }
20         }
21     }
```

Nous nous sommes rendus compte de cette nécessité lors du premier test, durant lequel le score a tout de suite été mis à zéro en raison du nombre de boucles par seconde. Nous avons également accompagné le score de logs en console afin que l'utilisateur puisse en avoir la connaissance.

A screenshot of a Windows console window titled "C:\Users\fmallard\Desktop\Lo\Miv_project\ProjectMIV_2016_TP2\bin\Release\ProjectMIV.exe". The console displays a series of log messages. It starts with several "MouseDown - left button" entries. Then, it shows "You touched a forbidden particle ! Your score : 8". This is followed by more "MouseDown - left button" entries, then "clicked : LEFT - - 408 - 572". After another "MouseDown - left button", there is a sequence of seven "You touched a forbidden particle ! Your score : [7-1]" messages, where the score decreases from 7 to 1. The console window has a standard Windows interface with a title bar, minimize, maximize, and close buttons, and a vertical scrollbar on the right side.

```
C:\Users\fmallard\Desktop\Lo\Miv_project\ProjectMIV_2016_TP2\bin\Release\ProjectMIV.exe
MouseDown - left button
MouseDown - left button
MouseDown - left button
You touched a forbidden particle ! Your score : 8
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
MouseDown - left button
Clicked : LEFT - - 408 - 572
MouseDown - left button
MouseDown - left button
MouseDown - left button
You touched a forbidden particle ! Your score : 7
You touched a forbidden particle ! Your score : 6
You touched a forbidden particle ! Your score : 5
You touched a forbidden particle ! Your score : 4
You touched a forbidden particle ! Your score : 3
You touched a forbidden particle ! Your score : 2
You touched a forbidden particle ! Your score : 1
```

FIGURE 11 – Logs du score dans la console.