EPITA

undergraduate (first year) project

15.03.2018

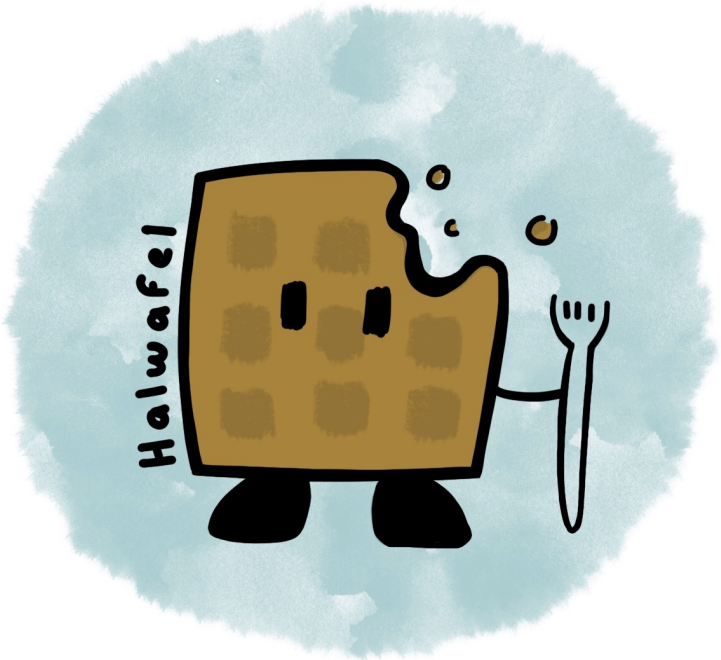
1st Presentation report

THE FRAME WORKSHOP

3D Animation software

Halwafel

Brian Bang, Etienne Benrey, Léane Duchet, Julie Hazan



Contents

1 Achievements **2**
- 1.1 Graphic User Interface (GUI) . . . 2
- 1.2 Sound and Mesh import . . . 5
- 1.3 Skeleton creation . . . 6
  - 1.3.1 Manual creation . . . 6
  - 1.3.2 Automatic creation . . . 6
- 1.4 Website . . . 12

2 To be done by the next presentation **13**
- 2.1 Graphic User Interface (GUI) . . . 13
- 2.2 Creation of motion . . . 13
- 2.3 Website . . . 14

Introduction

The Frame Workshop is a 3D animation software which will allow the user to make short animations. For now, the mesh and sound imports are implemented as well as the tools to create a skeleton and part of the Graphic User Interface. In this report will be presented : what has been done for now and what will be done by the next presentation.

1 Achievements

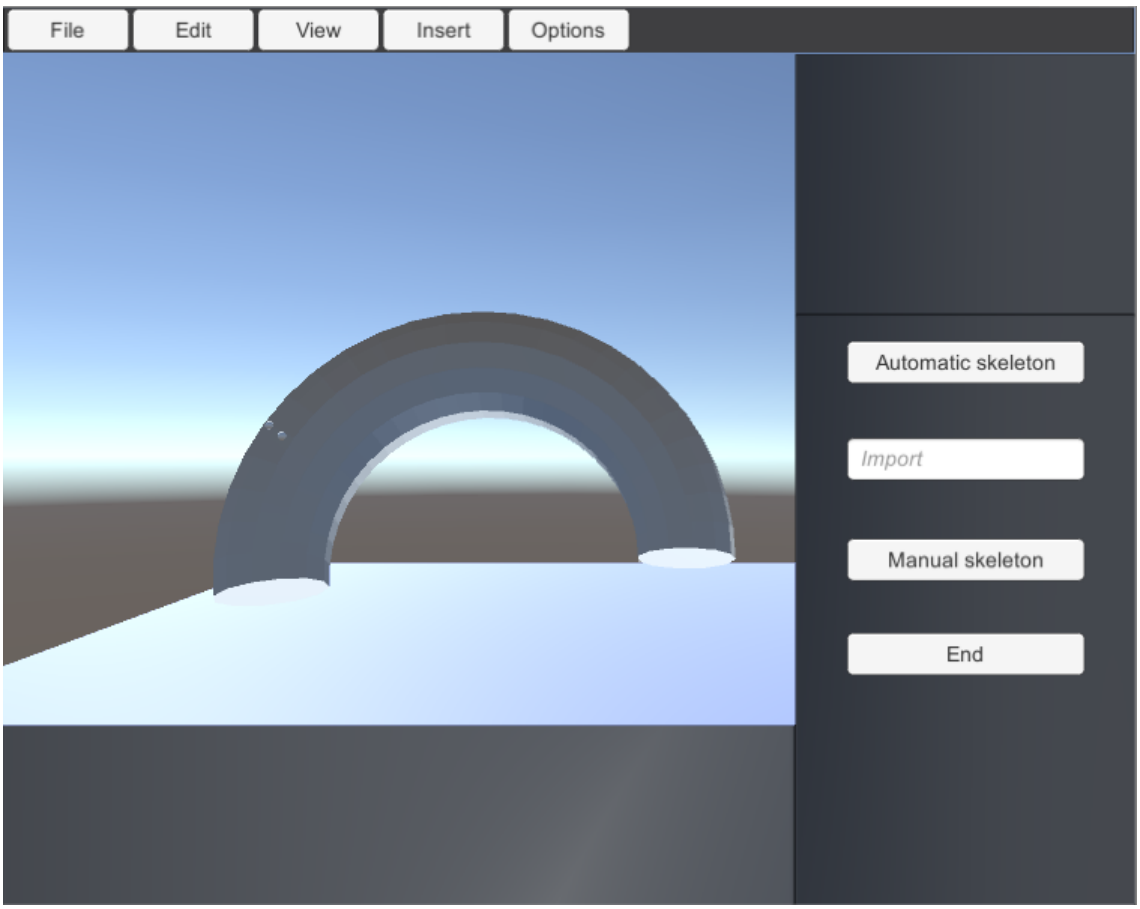
Part of the project	**What had to be done by the first presentation**	**Done parts of the project**
Mesh + Sound import	100%	100%
Skeleton creation	100%	70%
Graphic User Interface (GUI)	30%	30%
Creation of motion	0%	0%
Background imaging + sound	0%	0%
Particle animation	0%	0%
Website	40%	50%

Done parts of the project	**Brian Bang**	**Etienne Benrey**	**Leane Duchet**	**Julie Hazan**
Mesh + Sound import	X			X
GUI				X
Skeleton creation		Point Class + Finding collision with the mesh + Finding the extremities	Tree Class + Creation of the intermediate points and links	

1.1 Graphic User Interface (GUI)

This part of the GUI work consisted in managing the visual aspect of the program and the interactions with its different functionalities. First, we handled the general aspect of the interface, organizing the window and deciding where to place the buttons and other UI elements. Then, another important part of the work was to facilitate the user's direct interaction with the scene and its objects. Meaning how we can modify the view in the 3D scene, move an object and even build up manually the skeleton.

Creation of interface using UI elements



Interface using UI element

The interface's general aspect was made using UI panels, whose position depends on the camera. For any movement of the camera, the interface would move accordingly. Panels also possess a useful property which is the adaptation to the size of the screen. The sizes of the panels will be stretched if the program is opened on a larger screen or on the contrary, they will shrink if the main window is reduced.

Our interface consists of four panels: a tool bar, two aligned panels on the right for the inventory (up right) and one for the interaction with the scenes (down right) and one last panel for the time lines. The rest of the screen is occupied by the 3D view. The interface will also be accessible as a set of independent windows that can be dragged around and resized by the user. this version hasn't been implemented yet but we managed to write a script that generates a draggable window. / The next step will be to report the information of UI panels on those windows and make them resizeable, this should be done by the newt presentation.

The interface also needs some elements to allow the user to interact with the program, so several UI objects were implemented such as buttons, a text box, sliders and many more that will be added in the next months. Those elements do not have the same distortion properties as the panel: they are fixed. This means that their size won't be affected by a change of size of the screen. Tool bar buttons are an exception as they are not fixed vertically, their height adapts to the one of the tool bar panel. The anchor tool of unity was very useful at this moment: it determines the level of distortion of an element, relatively to an axis and the parent of the object.

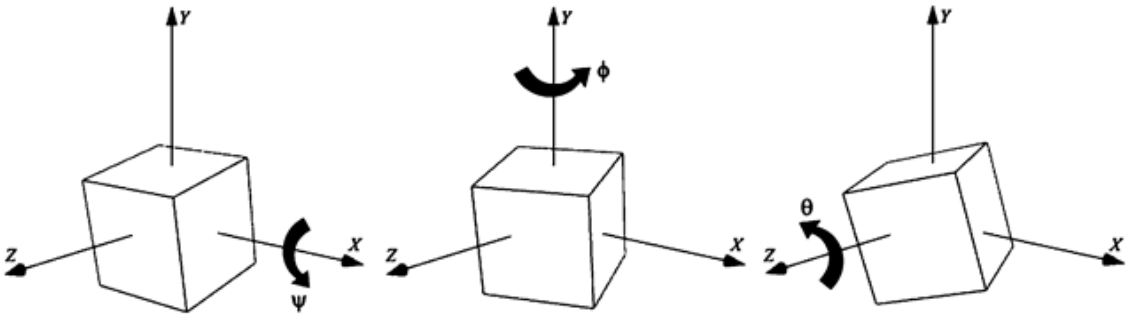
For now, the tool bar is composed of six buttons. Then, on the interactions panel, we implemented three buttons that correspond to three interactions with the scene. The first

one is the automatic creation of the skeleton. The user will have to select an object of the scene and spheres, representing the points of the skeleton will be displayed. The second one is an input field that should be used for the import of sounds and 3D models. The path of the file to import should be inserted in the text box, then press enter. The file will be copied into the "Models" or "Sounds" folder in the Assets of the project. If the model is a FBX file, it will be inserted directly in the scene.

Camera movement

To allow the user to set up his scene and move around in the 3D view, we had to manage camera displacement and rotation. A first attempt was to synchronize the mouse movements with the one of the camera. This method was very intuitive but could not allow many displacement possibilities : the Y axis wasn't taken into account. The second idea was to take the same displacement commands as Unity. This means the camera movement had to be decomposed into three sections : rotation (right click and dragging), side movements (scroll wheel click and dragging) and zoom (scroll wheel).

The rotation of an object in Unity can be identified by the Euler angles. This method compares the local axes of a rigid body with fixed axes. Any orientation can be described by three elementary rotations: the three Euler angles. this will allow use to affect the three dimensional orientation of the camera by applying a 3D vector.



Rotation of cube with Euler angle representation

Here, we can see the three components determining the orientation of the cube. the three Euler angles are Ψ, Φ and Θ, respectively along the x, y and z axes.

In our case, the rotation affects only the x and y axes of the camera, we don't want the camera rolling on the side. The program starts when the scroll wheel is pressed down. At this moment, the position of the mouse is stored as the initial position. If the mouse is dragging, we compute the difference between the initial position and the current position of the mouse, this will give us the two dimensional vector of rotation (the last coordinate, z, is set to zero) . The vector is then added to the Euler angles of the camera, multiplied by the time variable and the chosen speed.

The side movements was the simplest section to implement. The program must detect anytime the scroll wheel is being pressed and subtract the position of the mouse (multiplied by the speed and the time variable) to the camera position. Only the x and y axes need to be updated as side movements only deal with parallel displacements to the screen.

Finally, the zoom section consists of a linear displacement along the z axis (in Unity, the z axis represents the depth of the scene). However, we had to take into account the orientation of the camera: the local z axis of the camera may not be aligned with the absolute z axis. A solution to this problem was to link the camera to an empty game object, a parent, that would always keep the same orientation and will follow the camera into its movements. to obtain the z camera axis, we simply have to get the localEulerAngles of the camera and only take the z component. Therefore, every position changes should be applied to the parent of the camera (in this way, the local camera position is always the zero vector) and any rotation should be applied to the local camera Euler angles, not affecting its parent.

Drag and Drop of objects

Another important element that allows the user to interact with the objects in the scene is to be able to drag and drop them around in the 3D view. We need first to have a function that returns the selected object. This is done by using the raycast Unity tool. It draws a perpendicular line to the screen, with its origin located on the pointer. The generated ray then intersects with the collider of the object and returns the associated game object.

When the mouse is dragging, its position is being added to the one of the target. In addition to this, we wanted the dragging of the object to be more precise and also to be able to move the object along the z axis (which could not be done with the mouse movements) so the idea was to decompose the movements of the target along the three absolute axes. In this way, if the x key is pressed down, the object must only move along the x axis (respectively with the y and z keys).

1.2 Sound and Mesh import

This part was complicated, at the beginning our goal was to open the file dialog and let the user choose the file he wanted, the program was supposed to check whether the file was an audio file or a 3D model. When we started, the first issue that occurred was that we didn't know where the chosen file was going, so we created two files Models and Sounds and changed the target path depending on the extension of the chosen file. Afterwards, while we were running the program it took between one and two minutes to do the whole copying process. Also during Runtime mode we don't have access to the UnityEditor class which is the class we need to open the file panel to let the user pick the file he needs so we had to opt for another method.

The other method may not be the most efficient considering that the user has to seek for his files, indeed what we did is asking for the path to the file the user wants to import and then we copy it in a resource folder to the corresponding format (i.e Models or Sounds). Since the latter was quicker than the first method we used, we decided to keep this one despite its difficulty to use.

1.3 Skeleton creation

1.3.1 Manual creation

Point class

This class is used to make the nodes of the skeleton. It takes a name and a Vector3 of Unity as parameters. The first parameter is made to be able to have a name for each point in Unity and the second to place it considering the axis of Unity. A created point also has a Vector3 to represent its rotation, which will be useful when creating motion.

The Point class has different methods so that the rotation and the position can be modified by adding to the coordinates. Each point will have to be added in a tree, and thus has its position in the tree which will be modified at he creation of the tree it is part of.

Tree class

This class is used to make the links between the points. It is an implementation of a general tree, with the root and a list containing its subtrees. We can either add a full subtree or a point to that tree (if a point is added, a tree is first created and then added to the list of subtrees).

Manual skeleton creation

The user has the possibility, as an alternative of the automatic skeleton creation, to generate and place himself the points of the skeleton of a specified object. the process begins by clicking on the "Manual skeleton" button, the user must also select an object on the scene by clicking on it. This part uses the raycast functionality of Unity which allows the program to identify an object on the scene located under the pointer, when the left mouse button is activated. A red sphere is then added to the scene which represents the root of the skeleton. It is set as a child of the object which means every movement of the parent object will affect the position of the root, therefore the position of the skeleton.

The rest of the skeleton consists of a general tree, with each node a sphere with its position. There are three ways to complete the tree: by pressing the E key, the user will be able to add a child to the last created sphere, when '+' is pressed, it will add a child to the parent of the last created object. The user can also simply select a sphere from the skeleton and chose to add a child to it, to drag it around or add a child to its parent.

Due to the relation child parent between spheres, there exists a hierarchy of movement. Thus, a change in the position of a sphere will affect all its children but the position of the parent will remain unchanged. This will be very useful when implementing the animation of the skeleton.

1.3.2 Automatic creation

As we have some problems with the function finding the collisions with the mesh, the other two functions which use it are not completely functional and only partially works.

Finding collisions with the mesh

One of the most important features we have to implement in order to automatically create a skeleton is finding the collisions between various line segments and the object's

mesh. This allows us to know when points in 3D space are inside or outside the mesh, if a line segment is fully included in the mesh, etc. Since there are no pre-implemented methods that fill a similar role, we had to implement one ourselves. The method functions in the following way: for each triangle of the object's mesh, the program determines if the line segment intersects the triangle. If so, that intersection point is added to a list in the form of an object of class Point. This is done in four consecutive steps:

1. First, the function computes the intersection between the line and the plane defined by the three points of the triangle. This intersection can either be empty, in which case we can ignore this triangle, a single point, or an infinity of points. The latter is extremely rare, as even if the line and the triangle are parallel and aligned, the lack of precision of a floating-point number will slightly skew this result. If such a case does happen, it can be ignored, as one of the triangles connected to this one will also be intersected by the line. This means that all cases in which the intersection is not a single point can be ignored.

   Finding the intersection between the plane and the line requires the plane Cartesian equation of the plane, so a normal vector to the plane, and a parametric equation of the line. This leads to solving the following system:

$$S = \begin{cases} a*x + b*y + c*z + d = 0 \\ x = \alpha * t + x_M \\ y = \beta * t + y_M \\ z = \gamma * t + z_M \end{cases}$$

   with $M \begin{pmatrix} x_M \\ y_M \\ z_M \end{pmatrix}$ a point of the line and $\vec{n} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ a normal vector to the plane defined by the points of the triangle.

   With a little computation, we get the resulting solution for $I \begin{pmatrix} x_I \\ y_I \\ z_I \end{pmatrix}$, the intersection between the line and the plane:

$$S = \begin{cases} t = \frac{-(a*x_A + b*y_A + c*y_A + d)}{a(x_B - x_A) + b(a(y_B - y_A) + c(a(z_B - z_A)} \\ x_I = (x_B - x_A) * t + x_M \\ y_I = (y_B - y_A) * t + y_M \\ z_I = (z_B - z_A) * t + z_M \end{cases}$$

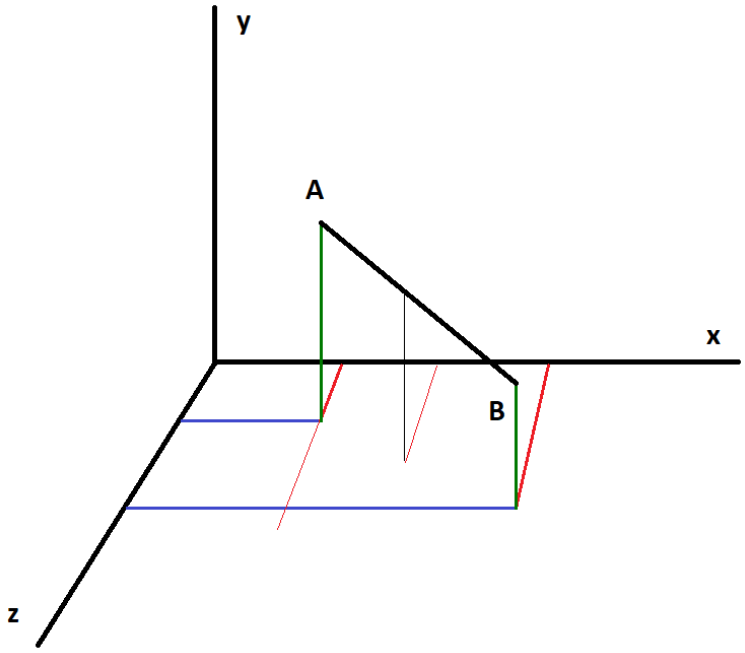
   with $A \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix}$ and $B \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix}$ the two points defining the line.

   If the denominator of t is null, then we are in one of the cases in which the intersection is not a single point, so we can simply make an exception for when this is the case. This allows for a clear and unique formula for each one of the coordinates of I.

2. Second, the program checks if this intersection point is included in the triangle. The method used is to check, for each of the three sides of the triangle, if the intersection point is on the same side of the line (defined by the studied side) as the vertex not included in the line. This is done by multiplying the coordinates of the vector product of

the vectors connecting both points (the intersection point and the triangle's vertex that is not included in the line) with one of the points of the line, and the vector connecting the two points defining the line. If the two points are on the same side of the line, then the vector products will have the some orientation, and the scalar multiplication of their coefficients will be positive. If they are on opposite sides, implying that the intersection point is not included in the triangle, then the scalar multiplication of their coefficients will be negative. Testing this for all three sides determines if the point is in the triangle or not.

3. We now have a list with all the intersections between the line and the mesh. However, what is requested is only the ones between the two given points, not on the whole line. As such, the next step is to remove all points that are not in this range. This is done by looking at simply one coordinate at a time. Assuming that the difference between the x coordinates of the two points is not null, any point between the two points will have their x coordinates between the x coordinates of the two points. In order to avoid problems with float precision, the program does this for the coordinate (x, y or z) for which the difference is the greatest, ensuring that we do not miss any elements.



Identification of point in line segment

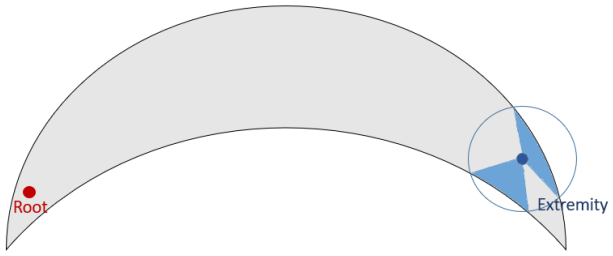
4. The final step is sorting the points, so that they are in order. This essentially means sorting them by increasing distance to one of the points. To do so, the program generates a list containing tuples (implemented as arrays of size 2) with the distance and the index of point in the original list of points. A simple bubble sort is applied to this list of distances, and then a new list is created and returned, according to the new order of indexes. At this point, the functions has finished running and returns the resulting list of 3D points.

Despite having the functionalities of this function working, we are still having difficulties getting the desired results. We know through various tests, that this is due to the fact that within the function, the points defining the triangles of the mesh are not accurate, in the

sense that they do not correspond to the actual points of the mesh. This means that we receive as output intersections with planes that are not that of the game object that we are studying.

Finding the extremities

Now that the tools are in place to start finding points, the program can attempt to find the extremities of the mesh. To do so, the program traverses through the whole inside of the mesh, looking at points at regular intervals. Each one of these points is analyzed to know whether it is an extremity or not. Within a radius determined by the size of the object and the user-defined density, the program checks how much of the point is surrounded by mesh, and how much of it is not. This is done by approximation, looking at 26 different angles, all at an angle $\frac{\pi}{2}$ of one another. If the point is considered to be an extremity, it is added to the list of points that will be the list of extremities. Since the positions that are looked at are spaced out by a distance that is proportional to the radius of the circle, overlapping points that would correspond to the same extremity are avoided.



Finding extremities

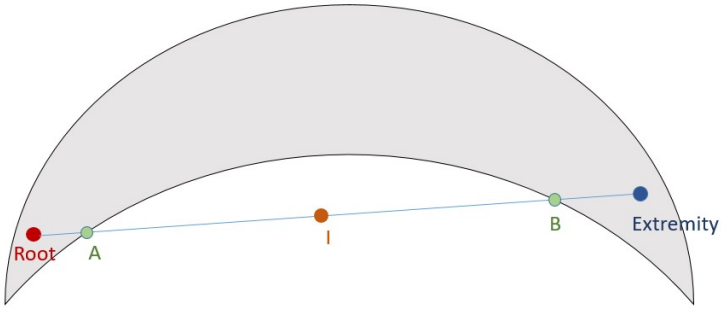
Creation of points and links between the root and the extremities

The next thing that has to be done is the links between the root chosen by the user and each one of the extremities of the mesh. For each one of the end, the goal is to either directly link the root with the extremity or to find intermediate points to be able to link them.

If it is possible to link them directly, that is to say that the function determining the collisions did not find any collision with the mesh, then the extremity is added in a tree that will become a subtree of the root.

Otherwise, we start to search for possible intermediate points that allow us to create links. We will do this in four steps.

1. If the point cannot be linked to the extremity that means that there should be at least two points of collision with the mesh in between those two, so we take those two points that will be called A and B, and we first search for the middle I of the segment [AB].



Finding the center of AB

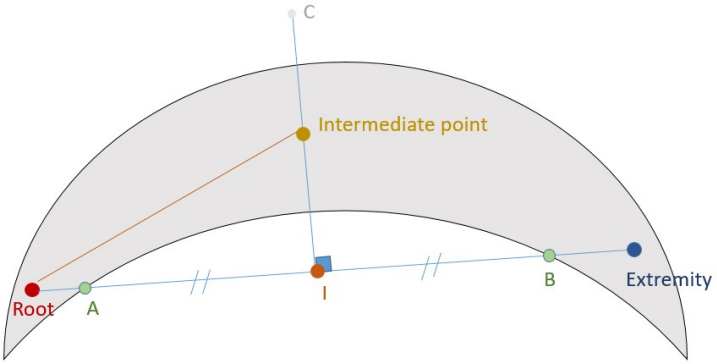
2. After finding I, the points of the circle of center I and a given radius R (big enough so that there could possibly be collisions with the mesh between those points and I) have to be found. This circle is the intersection of the sphere of center I and radius R with the plane containing I and orthogonal to AB. The points of this circle are given by :

$$C : \begin{cases} x = \frac{-(2*y_I+R*sin(\Phi)*sin(\theta))*(y_B-y_A)-2*z_I+R*cos(\Phi))*(z_B-z_A)}{x_B-x_A} \\ y = y_I + R * sin(\Phi) * sin(\theta) \\ z = z_I + R * cos(\Phi) \end{cases}$$

with $0 \leq \Phi \leq 2\pi$ and $0 \leq \theta \leq \pi$.

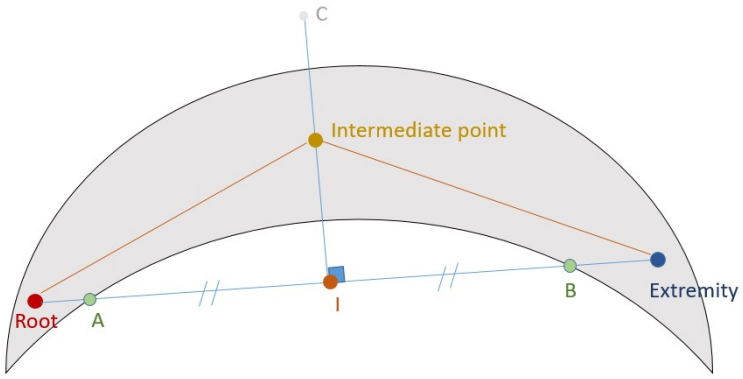
If we have $x_B - x_A = 0$ then the intersection is computed differently and will give formulas with either $y_B - y_A$ or $z_B - z_A$ as the divisors. As A and B can't be the same point, we will always have at least one those divisors different to 0, giving us all the possibilities for the difference between A and B.
We then check for different angles of θ and Φ whether we have collisions with the mesh between I and the computed point. If we do not we just continue to search computing another point's coordinates until we either find one that crosses the mesh at least twice or we have done a complete circle, that last case means we have no way of creating an intermediate point using this point I so the first step is redone on the right and on the left of I (using I as A or B, like in a binary search).

3. If a point for which the mesh is crossed at least twice was found, then a point that is in the mesh is taken (between the first and second collision with the mesh). Then, if this point can be linked to the root, a point of the Point class of the skeleton is created and is linked to the root using a Tree (it becomes a subtree of the root). If not, other points are tried until one that works is found or until too many tries have be done. If no possibility has been found then we go back to the first step.



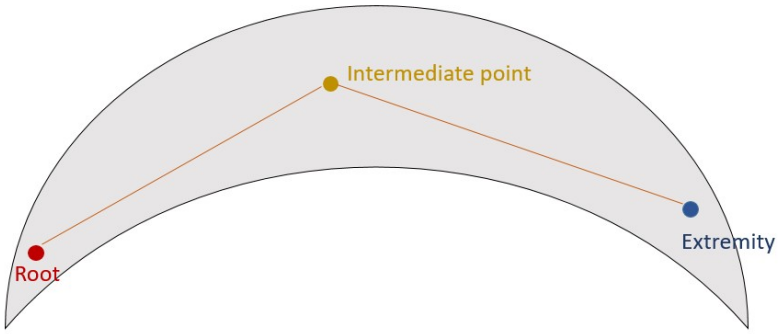
Finding a point in the mesh and linking it to the root

4. If the root has successfully been linked to an intermediate point, the extremity is linked to that point if it is possible, else the first step is redone, using the intermediate point just created as the root. The other extremities are tested to see if it is possible to link them to this intermediate point. If it is, then it is done else, the whole five steps are done on the next extremity, if it is not already linked to the root.



Linking the intermediate point to the extremity

The skeleton that is obtained is composed of the root which is linked to all the extremities through the different intermediate points.



Final skeleton

1.4 Website

The website has been started for this first presentation. We filled it with the most common information, description of the project, team members, book of specifications. The website will be updated for the next presentations following what we will do next.

To do the latter we have downloaded a template from the website TtemplateD, and created different sections (Homepage, Book of specifications, About us, FAQ's, Contact us). Everything has been done using HTML (HyperText Markup Language) which is used for the creation of websites, and CSS (Cascading Style Sheets) which is used to handle the presentation or layout of a website that is to say its design. Almost every section will be updated. We have translated the few pages we have in French and made links between corresponding pages (e.g if you are on the English version of the "About us" page and want it in French then you will be redirected to the French "About us" page and not the homepage.)

Finally we needed to find a way to host the website for free, so we looked into some free DNS providers, though the process wasn't easy to understand. Then we found out that GitHub provides a webpage hosting service called GitHub pages to make it easy to host one's webpage, and this is the method we went for. Almost everything has been set so we're ahead of schedule on this part.

2 To be done by the next presentation

2.1 Graphic User Interface (GUI)

For the GUI, the first thing that will have to be implemented is scripts for each already created button and change the location of some of the buttons as well as add some buttons for the new tools that will be made for the motion.

- The import button will be placed under the insert button.
- The option button have to be linked to a script which will allow the user to move the windows freely around the screen.
- The tool buttons will contain all the tools used for creating the animation (so for the second presentation, only the button for the motion will be added in there).
- The view button will be used to change the angle of 3D view window. There will be a top view, bottom view, front view, right and left view.
- In the file folder, there will be buttons that will allow the user to save the project (two, one to save in place one to choose where he wants to save), to create a new project and load an already created project and finally a button to be able to render the animation.
- The user will be able to load an object onto the scene and remove it from the scene and from the inventory.

There also are 4 different types of timeline that needs to be implemented.

- The Visual timeline manages the visual effects, which include movements, objects appearing and disappearing, particles. (As the particles will not be implemented for the next presentation, only movements and objects appearing and disappearing will be shown on this timeline.)
- The Sound timeline. It will be shown on the GUI but will not be functional as the backround sounds will not be implemented.
- The Camera timeline which will contain all the camera movement in the scene.
- Finally, there will be the Scene timeline which will regroup all the others.

2.2 Creation of motion

The goal for the creation of motion for the second presentation would be to be able to make the different points move all together. We have some ideas concerning this part, one of them would be to move the fewest points possible by rotating a given point around another point and doing so until we get to the wanted position. Another idea would have been to allow the user to draw a line from a chosen point to another point in space and allow him to set the step along this line. Though such a method would imply that we may increase the distance between points and that would totally deform the skeleton.

Let us go more in depth on the first possibility. This method would require two things: the initial position of the point and the position we wish the point to be in at the end of

the movement. The movement is hence created point by point, each movement of a point directly affecting that of its children, and possibly that of its parents. According to how we envision this method to work (based on the research that we have done on the subject), the first case would be if the wanted resulting point is on the circle of center the point's parent, and radius the distance between the parent and the point. This means that we have to apply a simple rotation to the point in order to get it to the wanted position. If the point is not on the circle we mentioned previously, then we have to apply the same process to the parent, rotating it around its own parent, in order to have the child in range of the wanted final position. It is important to note that when a point is moved, all its children are moved in the exact same way. That is to say, if a point rotates by an angle δ around its parent, then the point's children will also rotate by an angle δ around the point's parent (and not the point). This will act recursively, as affecting a child will affect it's children's children, continuing in this manner. This means that moving a point will mean moving all the nodes of all the branches of the tree that start at this node in the same exact way.

As mentioned before, moving a point might imply moving its parent. If moving its parent is not enough, then we will move the parent's parent, and so forth. We want to move the parent to the maximum of its capacity before moving its parent, to ensure that moving one point will not somehow move a point at the other end of the object, which would make for quite unusual movements.

This method is that one that we are looking to use, but further research might encourage us to use different methods, such as the one mentioned earlier.

2.3 Website

The Book of Specifications section should be replaced by a download section where the documents such as the book of specifications, the reports and the program, when ready, will be downloadable. We're also planning on translating the book of specifications to have a better grasp of the project to anyone who would have no idea of what the project is about. A log book should be set up to have a better overview of what have been done and the other pages will be filled as the project goes.

Conclusion

That was the first step of the project; the website, the import and the GUI are on schedule, though we still need to catch up on the automatic skeleton generation. So this will need to be done by the next presentation along with the creation of motion and other updates on the website and the GUI.

We decided to do something different from a game and we start realizing that it is more difficult than what we expected. We still are quite positive that we will manage until the end, even with the hardships we will have to overcome.