

Edition multi-résolution d'une animation

L'objectif de cette partie est de transformer une animation par une édition multi-résolutions. Un peu comme les slider d'un equalizer permettent de changer le son des aigus aux graves : amplifier les grandes amplitudes pour exagérer un mouvement sans toucher aux petites amplitudes pour ne pas ajouter des tremblements (ou l'inverse). Tout est très bien expliqué dans l'article suivant section 2, en particulier 2.1 : [Motion Signal Processing](#)

1. Trace 3D de chaque articulation

L'objectif de cette partie est d'afficher une animation en traçant la courbe suivie par chaque articulation. Cela servira une fois la multi résolution calculée pour observer l'effet sur les courbes de trajectoires. Nous allons créer et ajouter un nouveau type d'"inspector" que l'on pourra nommer "Play Animation". Cette partie de l'interface permettra de choisir l'animation à afficher et offrira des cases à cocher pour chaque articulations indiquant si la trajectoire doit être affichée ou non.

- Créez un folder "Editor" qui contiendra le script PlayAnimationEditor.cs
- Rangez également tous les autres fichiers comme ceci



- Le squelette vide de PlayAnimationEditor.cs est plus bas
- Ouvrir la fenêtre avec le menu Window/PlayAnimationEditor
- Quand vous le script et repasser sous Unity, il y a une petite roue qui tourne en bas à droite indiquant que votre script compile, puis la ligne du bas indique la 1ere erreur

```
using System.Collections;  
  
using System.Collections.Generic;  
  
using UnityEngine;  
  
using System.Linq;  
  
using UnityEditor;
```

```
public class PlayAnimationEditor : EditorWindow
{
    // Current GameObject Selected on the Scene
    [SerializeField]
    protected GameObject m_skeleton;

    // List of all body joints of the current Skeleton
    // Transform -> Unity Type (Position, Rotation, Scale)
    // Matrix Transform
    [SerializeField]
    public List<Transform> m_BodyJoints;

    // Current AnimationClip Selected by the user
    // This AnimationClip will be played
    [SerializeField]
    protected AnimationClip m_animationClip;

    // Boolean to check if the animation is playing or not
    private bool m_b_isRunning = false;

    // Stocks the current Time
    protected float m_f_time = 0.0f;

    // Vector2 used to get the save the position of the scroll in the Window
    Vector2 m_scrollPostion = Vector2.zero;
```

```
// First frame in seconds (here it will be 0.0f)

private float m_f_startTime = 0.0f;

// Length of the AnimationClip in seconds

private float m_f_endTime;

// Frame Duration in seconds

private float m_f_frameDuration;

// accelerate or slow the animation

protected float m_f_scaleTime = 1f;


// Dictionnary of string & List<Vector3> which contains all the position
of one body Joint

private Dictionary<string, List<Vector3>> m_trajectories;


// Dictionnary of string & bool which contains all the bool used by the
toggle button

protected Dictionary<string, bool> m_toggleTraj;


// The new item in the menu

[MenuItem("Window/PlayAnimation", false, 2000)]

public static void DoWindow()

{

    PlayAnimationEditor m_window = GetWindow<PlayAnimationEditor>();

}
```

```
// Called when the user change the selected object in the scene windows
public void OnSelectionChange()
{
    // Pick the current selected gameObject in the Scene.
    m_skeleton = Selection.activeGameObject;

    // The user can pick in the void
    if (m_skeleton != null)
    {
        // Check if the current gameObject has an animator component or
        // an animation
        if (m_skeleton.GetComponent<Animation>() != null ||
            m_skeleton.GetComponent<Animator>() != null)
        {
            if (m_BodyJoints == null)
            {
                m_BodyJoints = new List<Transform>();

                // Get all the bodyJoints -> This is specific to the skeleton
                // that i used
                // One body joint is defined when
                m_BodyJoints = m_skeleton.GetComponentsInChildren<Transform>().Where(x => x.childCount != 0).ToList();

                //Repaint the currentWindow -> Call the OnGui function
                Repaint();
            }
        }
    }
}
```

```
// This function enables the editor to handle an event in the scene view.

// We need to redraw the curve & points when the user is interacting with
the SceneView

// In this function, you will have to code the drawing of the trajectories

public void OnSceneGUI(SceneView sceneView)

{

    // TODO

    // Parcourir toutes les articulations

    //     Si la togglebox est cochée

    //         Parcourir tous les points de la trajectoire et les afficher

    // Vous pourrez appeler la fonction de dessin d'un point et/ou
Handles.DrawLine(l_oldPoint, l_currentPoint);

}


// Init the trajectories for each body joints for the current Animation
Clip

private void initTrajectories()

{

    // Enable the Animation Mode if disabled

    if                                     (!AnimationMode.InAnimationMode())
AnimationMode.StartAnimationMode();


    // TODO

    // Créer ou vider dictionnaire m_trajectories qui va contenir la list
des points de la trajectoire

    // Créer ou vider dictionnaire m_toggleTraj qui va contenir la list
des bool indiquant si la trajectoire est visible ou non

    // Remplir le m_trajectories[m_BodyJoints[i].name] avec les positions
des points
```

```
//          Voir          AnimationMode.BeginSampling();          ....
AnimationMode.EndSampling(); qui se trouve dans playAnimation

}

// OnGUI is called for rendering and handling GUI events.

// Use OnGUI to draw all the controls of your window.

public void OnGUI()

{

    // We need to select a GameObject in the Scene

    if (m_skeleton == null)

    {

        EditorGUILayout.HelpBox("Please      select      a      GameObject.",
MessageTypes.Info);

        return;

    }

    // Check if the current GameObject is active

    if (!m_skeleton.active)

    {

        EditorGUILayout.HelpBox("Please select a GameObject that is
active.", MessageTypes.Info);

        return;

    }

    // Check if the current GameObject has an Animator or Animation
Component

    if      (m_skeleton.GetComponent<Animator>()      ==      null      &&
m_skeleton.GetComponent<Animation>() == null)

    {
```

```
        EditorGUILayout.HelpBox("Please select a GameObject with an  
Animator Component or Animation.", MessageType.Info);  
  
        return;  
  
    }  
  
    // Update the scroll Position  
  
    m_scrollPostion = GUILayout.BeginScrollView(m_scrollPostion, false,  
false);  
  
    // Begin a vertical group that will contains all the gui element we  
will declare between the BeginVertical and the EndVertical  
  
    EditorGUILayout.BeginVertical();  
  
    // Create a "Listener" on the next GUI Element in order to track some  
change on it  
  
    EditorGUI.BeginChangeCheck();  
  
    // Create an Object Field of type AnimationClip  
  
    // This allows the user to select which AnimationClip he wants to  
play  
  
    m_animationClip = EditorGUILayout.ObjectField("Current Animation  
Clip", m_animationClip, typeof(AnimationClip), false) as AnimationClip;  
  
    // If the animation clip has changed, we need to compute the new  
Trajectories !  
  
    if (EditorGUI.EndChangeCheck())  
  
    {  
  
        initTrajectories();  
  
    }  
  
    // If the user has selected an AnimationClip  
  
    if (m_animationClip != null)  
  
    {
```

```
// Get the Length of the current Animation
m_f_endTime = m_animationClip.length;

// Get the frame Duration of the current Animation
m_f_frameDuration = 1.0f / m_animationClip.frameRate;

// An example for a Slider with change detect

// So we need to call The EditorGUI.BeginChangeCheck() in order
to create a "Listener"

EditorGUI.BeginChangeCheck();

// Then we create the Object that we want to track some change
on

m_f_time = EditorGUILayout.Slider("Time (seconds)", m_f_time,
m_f_startTime, m_f_endTime);

// If the user has modified the Slider Precision here, we can
detect it and call a fonction for example

if (EditorGUI.EndChangeCheck())

    samplePosture(m_f_time);

EditorGUI.BeginChangeCheck();

// Then we create the Object that we want to track some change
on

m_f_scaleTime = EditorGUILayout.Slider("Scale Time",
m_f_scaleTime, 0.0f, 2.0f);

if (!m_b_isRunning)
{
    // Create a Button in order to plays the AnimationClip

    if (GUILayout.Button("Start Animation"))
    {
        // Starts the Coroutine that will play the Animation

//Swing.Editor.EditorCoroutine.start(repeatAnimation(m_f_frameDuration));
```



```
        // Coroutine is running

        m_b_isRunning = true;

    }

}

else

{

    // Stop the Coroutine

    if (GUILayout.Button("Stop Animation"))

    {

        //
Swing.Editor.EditorCoroutine.stop(repeatAnimation(m_f_frameDuration));

        m_b_isRunning = false;

    }

}

// TODO IHM :

// faire un for sur les articulations, creer des boites a cocher
pour chaque articulation

// Voir la doc :
https://docs.unity3d.com/ScriptReference/EditorGUILayout.Toggle.html

// Utilisez la variables : m_toggleTraj[m_BodyJoints[i].name]

}

// End the vertical group

EditorGUILayout.EndVertical();
```

```
// Stop the Scroll

GUILayout.EndScrollView();

}

// Call at each frame

// In this function, we will play the Animation

private void Update()

{

    // TODO

    // Verifier que m_skeleton m_animationClip, m_b_isRunning sont init

    // le temps : m_f_time

    // appeler samplePosture qui est ue fonction un peu plus bas


    SceneView.RepaintAll();

}


// Sample our Skeleton at the time given in parameter for the
currentAnimationClip

private void samplePosture(float p_f_time)

{

    // Check if the Game isn't running & the Animation Mode is enabled

    if (!EditorApplication.isPlaying && AnimationMode.InAnimationMode())

    {
```

```
        // We need to BeginSampling before the SampleAnimationClip is
called

        AnimationMode.BeginSampling();

        // Samples the animationClip (m_animation) at the time (m_f_time)
for the skeleton (m_skeleton)

        // If the GameObject & the AnimationClip are different -> no
errors are trigger but nothing happen

        AnimationMode.SampleAnimationClip(m_skeleton,  m_animationClip,
p_f_time);

        // Ending the Sampling of the Animation

        AnimationMode.EndSampling();

        // Repaint The SceneView as the skeleton has changed

        SceneView.RepaintAll();

        // Repaint the GUI as we are changing the variable m_f_time on
which we have a slider

        Repaint();

    }

}

void OnFocus()

{

    // Remove delegate listener if it has previously

    // been assigned.

    SceneView.onSceneGUIDelegate -= this.OnSceneGUI;

    // Add (or re-add) the delegate.

    SceneView.onSceneGUIDelegate += this.OnSceneGUI;

}

void OnDestroy()

{
```

```
// When the window is destroyed, remove the delegate
// so that it will no longer do any drawing.

SceneView.onSceneGUIDelegate -= this.OnSceneGUI;

}

}
```

C.2. Les filtres de trajectoire

- Ajouter un bouton "Gaussienne" qui passe un filtre gaussien sur un clip. Dans la fonction OnGuit() ajoutez

```
if (GUILayout.Button("Gaussian filter"))
{
    GaussianAnim();
}
```

- puis la fonction

```
public void GaussianAnim()
{
    AnimationClip clip = new AnimationClip(); // comportera la copie de
m_animationClip mais filtrer

    // Copy the m_animationClip in the local variable clip
    clip.legacy = m_animationClip.legacy;

    foreach (EditorCurveBinding binding in
AnimationUtility.GetCurveBindings(m_animationClip))
    {
```

```
        AnimationCurve curve =
AnimationUtility.GetEditorCurve(m_animationClip, binding);

        //TODO : editer chaque courbe ici avec

        // Parcourir toute la courbe avec comme longueur : curve.length
        // float v = curve.keys[time].value;

        // curve.MoveKey(time, new Keyframe(time, v));

        AnimationUtility.SetEditorCurve(clip, binding, curve);
    }

    // Si vous avez besoin de (quaternion+translation) il faut les
regrouper les courbes

    // il y a 7 courbes par articulations, par exemple pour le noeud
"root" il y a

    // "rootT.x" "rootT.y" "rootT.z" pour la translation

    // "rootQ.x" "rootQ.y" "rootQ.z" "rootQ.w" pour le quaternion

    // Il faut donc regrouper ces 4 courbes en un tableau de quaternion

    // Save the local variable clip

    AssetDatabase.CreateAsset(clip, "Assets/Gaussian.anim");
}
```

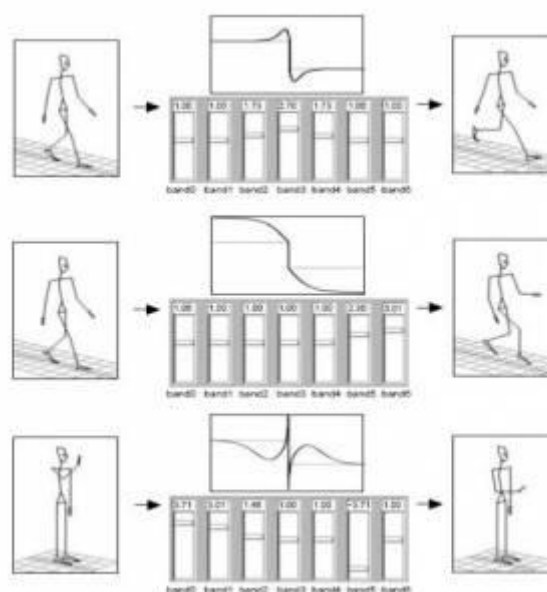
Un fois le filtre bien maîtrisé, passez à l'aspect multi-résolution.

- Pour chaque rotation de chaque articulation, construire la représentation multi-résolution. Une animation d'une articulation se reconstruit en partant de la valeur moyenne G_n + la somme de toutes les variations aux différentes résolutions.

• Foreach channel of each joint



- La ligne tout en haut est la courbe originale, les lignes d'en dessous sont des moyennes et des déplacements (=des écarts à la moyenne, valeurs rouges). Pour passer d'une ligne à la ligne d'en dessous, on moyenne deux par deux les valeurs et on obtient : la moyenne et le déplacement. Pour revenir à la courbe d'animation on remonte en additionnant à la moyenne le déplacement. Les sliders sont des coefficients multiplicateurs qu'on applique aux déplacements (valeurs rouges), ce qui change le signal quand on remonte.
- Par exemple avec une courbe ayant 2 valeurs. Courbe = 10, 14
- On obtient (en descendant) : Moyenne = 12; déplacement = -2, +2 (car $12-2=10$ et $12+2=14$)
- Si on multiplie les déplacements par 1.5 on obtient : Moyenne = 12; déplacement = $-2 \times 1.5 = -3$, $+2 \times 1.5 = 3$
- En reconstruisant la courbe (remontée) on obtient : Courbe = $12-3=9$, $12+3=15$
- On a gardé une courbe centrée en 12 mais ayant une amplitude plus grande.
- Maintenant avec une courbe de 256 valeurs. On peut descendre 8 fois, car la courbe peut être moyennée par 2, 8 fois. Cela donne 8 sliders.
- Demander à l'utilisateur via l'interface d'Unity les n (n=8 pour une animation à 256 valeurs) coefficients, multiplier chaque variation par ce coefficient puis reconstruire le signal.



Annexe : Faire du debug avec Unity et Visual Studio

- Unity vous affiche en rouge en bas des actions non valide
- Pour afficher du texte et une variable dans la console

```
Debug.Log("vSpeed = " + Input.GetAxis("Vertical"));
```



- Vous pouvez attacher Unity à Visual Studio et mettre des points d'arrêt. Pour cela, dans la barre des boutons vous cliquez sur "Attacher à Unity" puis le triangle de run.