Lab 4: Locomotion & Control

Published | Edit | ⋮

GPR-450 Advanced Animation Programming
Instructor: Daniel S. Buckstein
Lab 4: Locomotion & Control

Summary:
In the course so far, we have focused on three fundamental components of animation programming: time control, keyframes and clips (lab 1 and project 1); spatial pose and hierarchical animation algorithms (lab 2 and project 2); and putting those two topics together with blend operations, animation blending and layering (lab 3 and project 3). In all of that we have missed one critical part of real-time animation: the player's input, and actually controlling our character! In this lab we explore fundamental locomotion, and set up some temporal and spatial control algorithms, to begin implementing humanoid character controller in project 4.

Submission:
Start your work immediately by ensuring your coursework repository is set up, and public. Create a new main branch for this assignment. ***Please work in pairs (see team sign-up) and submit the following once as a team***:

1. Names of contributors
   e.g. **Dan Buckstein**
2. A link to your public repository online
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework.git**
   (note: this not a real link)
3. The name of the branch that will hold the completed assignment
   e.g. **lab0-main**

4. A link to your video (see below) that can be viewed in a web browser.
   e.g. <insert link to video on YouTube, Google Drive, etc.>

Finally, please submit a ***5-minute max*** demo video of your project. Use the screen and audio capture software of your choice, e.g. Google Meet, to capture a demo of your project

as if it were in-class. This should include at least the following, in enough detail to give a thorough idea of what you have created (hint: this is something you could potentially send to an employer so definitely show off your professionalism and don't minimize it):

- Show the final result of the project and any features implemented, with a voice over explaining what the user is doing.
- Show and explain any relevant contributions implemented in code and explain their purpose in the context of the assignment and course.
- Show and explain any systems source code implemented, i.e. in framework or application, and explain the purpose of the systems; this includes changes to existing source.
- **DO NOT AIM FOR PERFECTION, JUST GET THE POINT ACROSS**. Please mind the assignment rubric to make sure you have demonstrated enough to cover each category.
- **Please submit a link to a video visible in a web browser, e.g. YouTube or Google Drive.**

Objectives:
This assignment focuses on the player's interaction with an in-game character. You will apply everything you have previously learned to create the building blocks of a character controller.

Instructions & Requirements:
DO NOT begin programming until you have read through the complete instructions, bonus opportunities and standards, start to finish. Take notes and identify questions during this time. The only exception to this is whatever we do in class.
Using the framework and object- or data-oriented language of your choice (e.g. Unity and C#, Unreal and C++, animal3D and C, Maya and Python), complete the following steps:

1. ***Repository setup***: See lab 1 for the complete setup instructions.
   - Check out the '*anim/control*' branch from the course repository, it has some stuff you can start with.
2. ***Input node***: Create a hierarchical node and spatial pose for positioning the character in the world.
   - Since the character has a hierarchy of its own, we need a spatial pose to represent the entire skeleton as an object. This is an individual node that can behave like any other, in that it may be affected by blend operations, such as lerp.
   - The skeletal hierarchy as a whole is a "child" of this node; if this node moves, the whole skeleton follows. Spatially, it should be located directly under the skeleton's root node. Use existing systems in your selected engine/framework to achieve this

     (e.g. Unity's built-in hierarchy; create a "scene graph" for scene objects in animal3D, etc.).
3. ***Locomotion algorithms***: The following algorithms serve as the bases for *locomotion*, or controlled movement. Here we discuss the algorithms themselves; see part 5 (testing) for how they are applied and demonstrated.

- **Euler integration**: Implement Euler's method of integration for the target variable, given its derivative:
  - *Math*: $x_{t+dt} = x_t + x_t^{'}dt = x_t + (dx_t/dt)dt$
  - *Explanation*: Euler's method represents a regular discrete integral (as opposed to continuous, which is currently impossible in computing; it is an approximation of the real deal). It begins with the current state of some variable (value 'x' at time 't') and adds its current derivative (rate of change of 'x' at time 't') scaled by the current differential (time step, delta-time or 'dt'). The classic example of this in physics is integrating velocity into position, but it can also be used to integrate acceleration into velocity. If you know the the rate of change of the target variable, you can use this method.
  - *UML prototype*: The function takes the current value 'x (at t)', its derivative 'dx/dt' and differential 'dt', returning the integrated value at the end of the step 'x (at t+dt)'.

    ```
    +fIntegrateEuler(x : ftype, dx_dt : ftype, dt : float) : ftype
    ```

    - *Note*: Here, "ftype" is just some generic floating point type, be it a scalar (e.g. float) or vector (vec2, vec3), etc. The differential 'dt' is a scalar.
- **Kinematic integration**: Implement kinematic integration for the target variable, given its first and second derivatives:
  - *Math*: $x_{t+dt} = x_t + x_t^{'}dt + x_t^{''}dt^2/2 = x_t + (dx_t/dt)dt + (d^2x_t/dt^2)dt^2/2$
  - *Explanation*: Kinematic integration represents a discrete "second integral", or "the integral of the integral". The first two terms are the same as Euler's method (add scaled rate of change to current value), but we also integrate the second derivative (rate of change of rate of change of 'x' at time 't') scaled by half of the squared differential. In physics, this can be used to integrate both acceleration and velocity into position; sadly, however, we don't use it to integrate velocity because we don't typically use its second derivative.
  - *UML prototype*: The function takes the current value 'x (at t)', its derivative 'dx/dt', its second derivative 'd2x/dt2' and differential 'dt', returning the integrated value at the end of the step 'x (at t+dt)'.

    ```
    +fIntegrateKinematic(x : ftype, dx_dt : ftype, d2x_dt2 : ftype, dt : float) : ftype
    ```
- **Interpolation-based integration**: Implement "fake" integration using linear interpolation:
  - *Math*: $x_{t+dt} = \mathrm{LERP}_{x_t,x_c}(u) = x_t + (x_c - x_t)u$
  - *Explanation*: This is not a physics-backed integration algorithm, but it is a related and invaluable substitute in and locomotion, approximately equivalent to Euler's method. Here we start with the usual "current value" but we "fake" velocity by interpolating it towards some ***target value***, 'x_c'. This is a pre-determined "goal" value for what would otherwise be the final result of Euler's method, therefore it is

a stand-in for 'x (at t+dt)'. Instead of the usual differential, we use some interpolation parameter 'u' to control the strength of the velocity. When the parameter is zero, the velocity cancels out and the result is fixed at the original value 'x (at t)'; when the parameter is one, the velocity takes full strength and the result is fixed at the target value 'x_c'. Therefore, the parameter behaves as a measurement of how far between the current and target value we want to be!

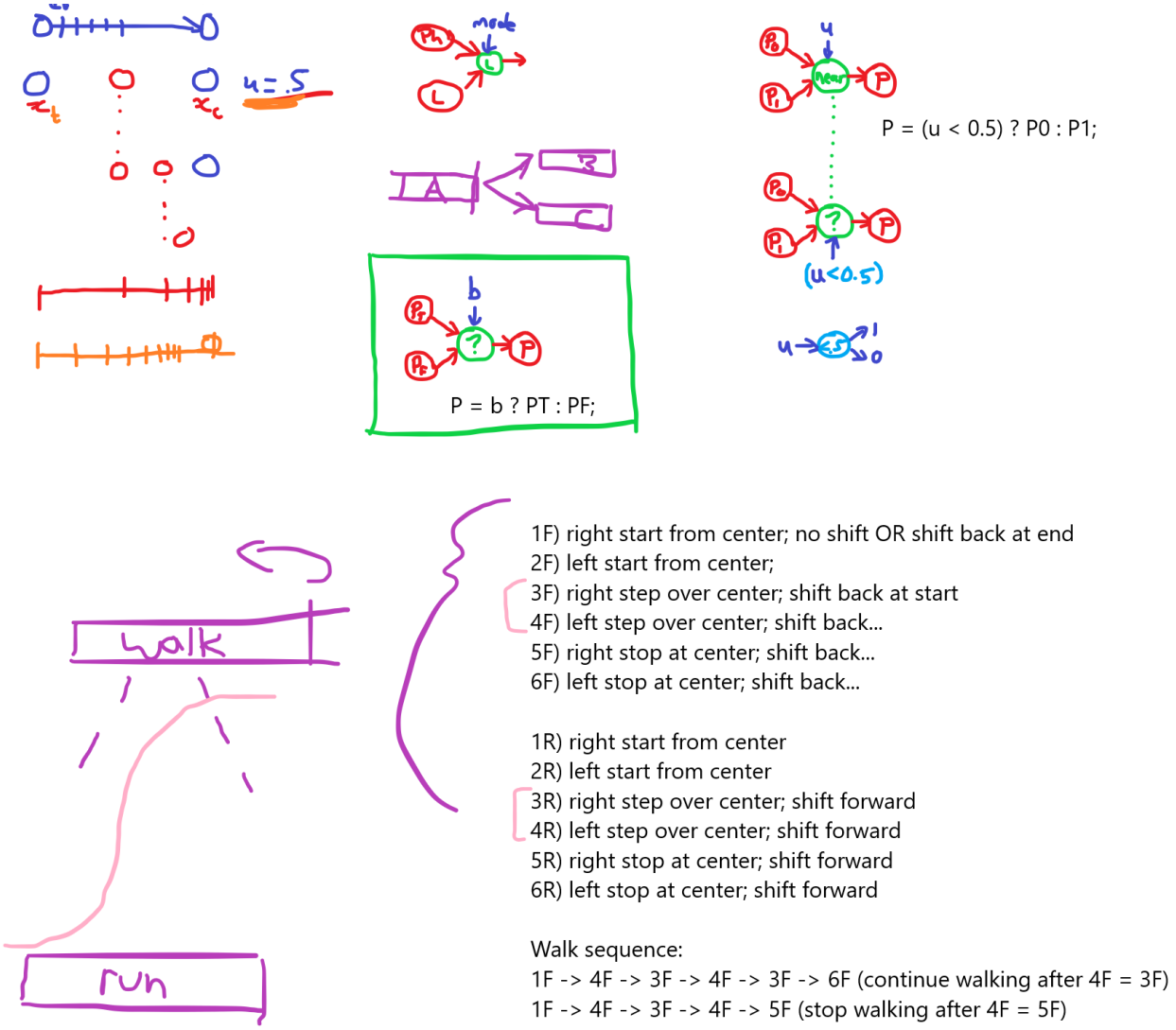
- *UML prototype*: The function takes the current value 'x (at t)', the target value 'x_c' and the interpolation parameter 'u', returning some blend between the current and target value.

```
+fIntegrateInterpolated(x : ftype, xc : ftype, u : float) : ftype
```

4. ***Branching transitions***: Implement a new type of clip transition that allows for branching or conditionals. See part 5 for how this is applied.
   - The branching transition performs some test to determine what to do after a clip ends.
   - This could be implemented as a new data structure, a function pointer, a manager of some sort, etc. This part of the assignment is intended to be creative, but we will discuss some strategies in class.
5. ***Basic testing interface***: Implement the following for your testing application:
   - ***Interface***: Set up the following in your testing application:
     - Instantiate a clip controller or a simple timer to maintain and update the current keyframe time. Have a minimum of two clips: idle, and anything else.
     - Instantiate and set up one hierarchy, skinned mesh optional.
   - ***Input***: Add the following inputs using either the keyboard or a gamepad:
     - **Locomotion controls**: Implement two bi-directional, double-axis input methods for controlling the character's *locomotion* and *orientation*. E.g. for locomotion, use WASD or the left joystick; for orientation, use IJKL or the right joystick. Implement and demonstrate the following modes for updating the character's motion (describe which one "feels" the best to control):
       - **Direct value**: Inputs control *position and rotation directly*.
         - The 2D vector from your locomotion input (WASD or joystick) directly maps to the character's *position in world-space* (AD or horizontal tilt for WS or vertical tilt for Y axis).
         - The horizontal axis of your orientation input (JL or horizontal tilt) directly maps to the character's *rotation about the world's "up" axis* (e.g. default is Z in animal3D). Note that since the input is in unit range [-1, +1], you may scale it to rotation range [-pi, +pi].
       - **Control velocity**: Inputs control *velocity and angular velocity directly*.
         - The 2D vector from your locomotion input directly maps to the character's *velocity in world space*. Integrate this into position using *Euler's method*.
         - The horizontal axis of your orientation input directly maps to the character's *angular velocity about the world's "up" axis*. Integrate this into rotation

using *Euler's method*.
        - **Control acceleration**: Inputs control *acceleration and angular acceleration directly*.
            - The 2D vector from your locomotion input directly maps to the character's *acceleration in world space*. Integrate both the current velocity and this acceleration into position using *kinematic integration*, *then* integrate acceleration into velocity using *Euler's method*.
            - The horizontal axis of your orientation input directly maps to the character's *angular acceleration about the world's "up" axis*. Integrate both the current angular velocity and this angular acceleration into rotation using *kinematic integration*, *then* integrate angular acceleration into angular velocity using *Euler's method*.
        - **Fake velocity**: Inputs control the *target position and rotation*.
            - The 2D vector from your locomotion input directly maps to the character's *target position in world space*. Integrate using *interpolation*.
            - The horizontal axis of your orientation input directly maps to the character's *target rotation about the world's "up" axis*. Integrate using *interpolation*.
        - **Fake acceleration**: Inputs control the *target velocity and angular velocity*.
            - The 2D vector from your locomotion input directly maps to the character's *target velocity in world space*. Integrate the current velocity into position using *Euler's method*, *then* integrate velocity using *interpolation*.
            - The horizontal axis of your orientation input directly maps to the character's *target angular velocity about the world's "up" axis*. Integrate the current angular velocity into rotation using *Euler's method*, *then* integrate angular velocity using *interpolation*.
    - **Branching transition**: Another input key or gamepad button forms the basis of your branching transition; if all of the above inputs are zero or near zero, your transition should target the 'idle' clip; otherwise, target the other selected clip.
- ***Update***: Perform the following tasks in your update loop:
    - Apply all inputs to the controlled character node and perform any spatial and temporal updates.
    - When the clip controller reaches the end of the current clip, execute the branching transition.
- ***Display***: Display the following information:
    - Display the input mode used for locomotion and orientation (they may be different).
    - Display the clip controller's info. Indicate the branch options.

6. ***Gallery***:
    - Here is the conditional node diagram and example from class:



Bonus:
You are encouraged to complete one or more of the following bonus opportunities (rewards listed):

- ***Procedural pitch (+1)***: As far as rotation goes, our controller currently rotates about the vertical axis only ("up"). Implement a new hierarchical "delta pose" that is used to control the character's pitch; the rotation is divided evenly among the pitch axis of all of the spine joints so the character

  appears to bend or curl. Aside from this change, it still appears to be in whatever other pose is currently being played back. Test and demonstrate all of the required integration algorithms on this pose using the vertical axis of your orientation control (IK keys or vertical joystick axis), selecting the best one for the final display.
- ***Second-order interpolation-based integration (+1)***: While the interpolation-based approach to

integration behaves like Euler's method for locomotion, there exist ways to "fake" the kinematic method incorporating a second derivative for position/rotation. Implement some interpolation algorithm to "fake" kinematic integration (e.g. quadratic Bezier interpolation).

Coding Standards:
You are required to mind the following standards (penalties listed):

- ***Reminder: You may be referencing others' ideas and borrowing their code. Credit sources and provide a links wherever code is borrowed, and credit your instructor for the starter framework, even if it is adapted, modified or reworked (failure to cite sources and claiming source materials as one's own results in an instant final grade of F in the course)***. Recall that borrowed material, even when cited, is not your own and will not be counted for grades; therefore you must ensure that your assignment includes some of your own contributions and substantial modification from what is provided. ***This principle applies to all evaluations***.
- ***Reminder: You must use version control consistently (zero on organization)***. Commit after a small change set (e.g. completing a section in the book) and push to your repository once in a while. Use branches to separate features (e.g. a chapter in the book), merging back to the parent branch (dev) when you stabilize something.
- ***Visual programming interfaces (e.g. Blueprint) are forbidden (zero on assignment)***. The programming languages allowed are: C/C++, C# (Unity) and/or Python (Maya).
  - If you are using Unity, all front-end code must be implemented in C# (i.e. without the use of additional editors). You may implement and use your own C/C++ back-end plugin. The editor may be used strictly for UI (not the required algorithms).
  - If you are using Unreal, all code must be implemented directly in C/C++ (i.e. without Blueprint). Blueprints may be used strictly for UI (not the required algorithms).
  - If you are using Maya, all code must be implemented in Python. You may implement and use your own C/C++ back-end plugin. Editor tools may be used for UI.
  - You have been provided with a C-based framework called *animal3D* from your instructor.
  - You may find another C/C++ based framework to use. Ask before using.
- ***The 'auto' keyword and other language equivalents are forbidden (-1 per instance)***. Determine and use the proper variable type of all objects. Be explicit and understand what your data represents. Example:
  - auto someNumber = 1.0f;
    - This is a float, so the correct line should be: float someFloat = 1.0f;
  - auto someListThing = vector<int>();
    - You already know from the constructor that it is a vector of integers; name it as such: vector<int> someVecOfInts = vector<int>();
  - Pro tip: If you don't like the vector syntax, use your own typedefs. Here's one to make the previous example more convenient:
    - typedef vector<int> vecInt;

    - vecInt someVecOfInts = vecInt();
- ***The 'for-each' loop syntax is forbidden (-1 per instance)***. Replace 'for each' loops with traditional 'for' loops: the loops provided have this syntax:
    - In C++: for (<object> : <set>)...
    - In C#: foreach (<object> in <set>)...
- ***Compiler warnings are forbidden (-1 per instance)***. Your starter project has warnings treated as errors so you must fix them in order to complete a build. ***Do not disable this***. Fix any silly errors or warnings for a nice, clean build. They are generally pretty clear but if you are confused please ask for help. Also be sure to test your work and product before submitting to ensure no warnings/errors made it through. This also applies to C# projects.
- ***Every section/block of code must be commented (-1 per ambiguous section/block)***. Clearly state the intent, the 'why' behind each section/block. This is to demonstrate that you can relate what you are doing to the subject matter.
- ***Add author information to the top of each code file (-1 for each omission)***. If you have a license, include the boiler plate template (fill it in with your own info) and add a separate block with: 1) the name and purpose of the file; and 2) a list of contributors and what they did.

Points 5

Submitting a text entry box or a website url

Due	For	Available from	Until
-	Everyone	-	-

GraphicsAnimation-Master-Range

Criteria	Ratings			Pts
IMPLEMENTATION: Architecture & Design Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine.	**1 to >0.5 pts** **Full points** Strong evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely both efficient and functional.	**0.5 to >0.0 pts** **Half points** Mild evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely either efficient or functional.	**0 pts** **Zero points** Weak evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely neither efficient nor functional.	1 pts
IMPLEMENTATION: Content & Material Practical knowledge of content relevant to the discipline and course (e.g. shaders and effects for graphics, animation algorithms and techniques, etc.).	**1 to >0.5 pts** **Full points** Strong evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely both efficient and functional.	**0.5 to >0.0 pts** **Half points** Mild evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely either efficient or functional.	**0 pts** **Zero points** Weak evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely neither efficient nor functional.	1 pts
DEMONSTRATION: Presentation & Walkthrough Live presentation and walkthrough of code, implementation, contributions, etc.	**1 to >0.5 pts** **Full points** Strong evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely both accurate and confident.	**0.5 to >0.0 pts** **Half points** Mild evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely either accurate or confident.	**0 pts** **Zero points** Weak evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely neither accurate nor confident.	1 pts
DEMONSTRATION: Product & Output Live showing and explanation of final working implementation, product and/or outputs.	**1 to >0.5 pts** **Full points** Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable.	**0.5 to >0.0 pts** **Half points** Mild evidence of correct and stable final product that runs as expected; end result is largely either correct or stable.	**0 pts** **Zero points** Weak evidence of correct and stable final product that runs as expected; end result is largely neither correct nor stable.	1 pts

Criteria	Ratings			Pts
ORGANIZATION: Documentation & Management Overall developer communication practices, such as thorough documentation and use of version control.	**1 to >0.5 pts** **Full points** Strong evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely both documented and organized.	**0.5 to >0.0 pts** **Half points** Mild evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely either documented or organized.	**0 pts** **Zero points** Weak evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely neither documented nor organized.	1 pts
BONUSES Bonus points may be awarded for extra credit contributions.	**0 pts** **Points awarded** If score is positive, points were awarded for extra credit contributions (see comments).		**0 pts** **Zero points**	0 pts
PENALTIES Penalty points may be deducted for coding standard violations.	**0 pts** **Points deducted** If score is negative, points were deducted for coding standard violations (see comments).		**0 pts** **Zero points**	0 pts
				Total Points: 5