Lab 1: Keyframe & Clip Controller

Published Edit ⋮

GPR-450 Advanced Animation Programming
Instructor: Daniel S. Buckstein
Lab 1: Keyframe & Clip Controller

Summary:
In this lab we explore the fundamentals of keyframe animation and time control by programming an abstract keyframe control interface. The lab is a required component of all future assignments, especially project 1 which builds directly off of what you do here.

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:
All animation stems from the change of some value over time. The main goal of this assignment is to create a basic time controller for keyframe animation so that we can convert real-time into frame time. You will think abstractly (i.e. without an application in mind) to implement an interface that will drive all future systems in the course.

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), complete the following steps:

1. ***Repository setup***: Complete the following steps to start:
   - *Create your repository online*: Once per team, using the online Git platform of your choice (GitHub, Pineapple, etc.), create a new repository, either from a template or afresh. If you use Pineapple, you must add your instructor as a collaborator. If you are working in pairs, ensure that both team members have write access to the repository.
   - *Clone your repository*: Using the local Git client of your choice (command line, Git Bash, Tortoise Git, SmartGit, etc.), clone the empty repository you just made.
   - *Clone and/or pull from the instructor's* ***starter repository (animal3D) (https://github.com/dbucksteincc/animal3D-SDK-202009FA-Animation.git)*** . If you plan on using your own fresh repository for assignments, simply clone the starter framework for class work. If you are using it for assignments as well as classes, after you have cloned your own repo as origin, add a new remote to your repository and give it an alias other than 'origin' (e.g. 'source'). Fetch and pull all branches from this repository. This starter framework will be used by the instructor to deliver useful content throughout the course. Note that you cannot push to this repository.
     - Before launching animal3D, clone and pull the latest revision of the **developer SDKs repository (https://github.com/dbucksteinccorg/DevSDKs-GPRO-2020.git)** . Run the batch file called *dev_install.bat*.
     - After pulling the latest revision in your local animal3D repository, run the CONFIG batch file as an administrator to launch a configuration project. It should open in

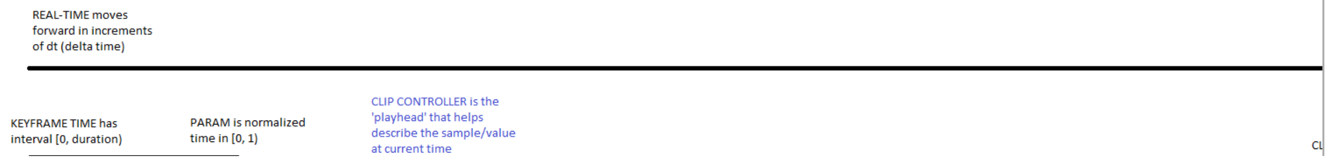
Visual Studio 2019 (which you should have installed). If it does not, manually open the utility (*utility/win/animal3D-VSSetPath.sln*) through Visual Studio 2019 as an administrator. Simply build the project and then close Visual Studio. Revert changes to the project through version control and delete any files that appeared from the build.
    - Now you can launch the framework ***exclusively with the LAUNCH batch file***. Never open animal3D manually (i.e. never navigate to and open the solution).
2. ***Data structures***: Implement the following decoupled data structures (the importance of decoupling and abstract thinking will be explained throughout) and associated methods:
    - ***Keyframe***: The core of "just data" in the context of animation programming. Describes a discrete sample of generic data with some value lasting some interval of time. An appropriate metaphor for this is a "moment in time".
        - *Index*: index in pool of keyframes (see below).
        - *Duration*: interval of time for which this keyframe is active; cannot be zero.
        - *Duration inverse*: reciprocal of *duration*.
        - *Data*: value of the sample described by a keyframe. For the purposes of this exercise, a simple integer will suffice. Use a random or predetermined value different from *index* so you can distinguish between them for any given keyframe.
        - *Constructor/factory/initialize*: set/change the duration (and inverse) and/or value of an individual keyframe.
    - ***Keyframe pool***: An unordered, unsorted collection of keyframes (*note: "unordered, unsorted" refers to the data itself; standalone keyframe data is meaningless until a clip arranges it and gives it meaning*).
        - *Keyframe*: array of all keyframes in the pool.
        - *Count*: number of keyframes in the pool.
        - *Constructor/factory/initialize*: allocate array of keyframes, initializing all to default values.
        - *Destructor/release/cleanup*: deallocate array.
    - ***Clip***: While a keyframe pool is an abstract collection of "just data", the clip is what actually gives it meaning as sequenced data. Decoupling these from keyframes is important because different clips may reference the same keyframe; individual clips do not own keyframes. A metaphor is the "timeline".
        - *Name*: string used to identify the clip.
        - *Index*: index of clip in pool of clips (see below).
        - *Duration*: duration of clip; can be calculated as the sum of all of the referenced keyframes or set first and distributed uniformly across keyframes; cannot be zero.
        - *Duration inverse*: reciprocal of *duration*.

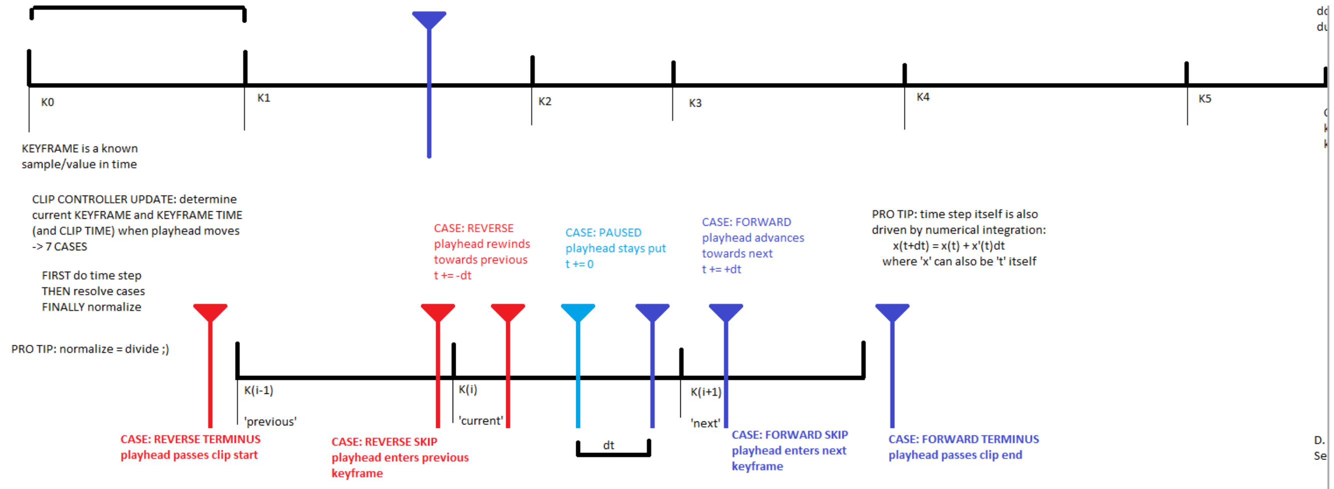
        - *Keyframe count*: number of keyframes referenced by clip (including first and last).
        - *First keyframe*: index of first keyframe in pool referenced by clip (see below).
        - *Last keyframe*: index of final keyframe in pool referenced by clip (see below).
        - *Keyframe pool*: pointer (C/C++) or reference (C#) to the pool of keyframes containing those included in the set; within the array, the clip will be the sequence

    of keyframes from *first* to *last*. Note: this is where the decoupling and abstraction comes into play: a clip does not own keyframes, it only references a set of some that exist elsewhere.
    - *Constructor/factory/initialize*: set name, referenced keyframe pool, first and last.
    - *Calculate duration*: calculate the total duration (and inverse) as the sum of all referenced keyframes.
    - *Distribute duration*: set duration (and inverse) directly and distribute it across all referenced keyframes to give them all a uniform duration.
  - ***Clip pool***: An unordered, unsorted collection of clips (*note: same as keyframes; clips themselves are just an arrangement of keyframes, what gives clips meaning is when they are played back in a certain way*).
    - *Clip*: array of all clips in the pool.
    - *Count*: number of clips in the pool.
    - *Constructor/factory/initialize*: allocate array of clips, initializing all to default values.
    - *Destructor/release/cleanup*: deallocate array.
    - *Get clip*: search for a clip by name, returning its index.
3. ***Clip controller interface***: Now that you have both raw and slightly organized data ("just data" up until now), it is time to create the thing that will make it actually functional: the *clip controller*. All of the decoupled data structures above have prepared you for this moment. The *clip controller* is the "playhead" that iterates over a clip to play back its keyframes in sequence. The chain of reference is: clip controller -> clip (by index in pool) -> keyframe (by index in pool). This is why decoupling is very important; so is separating "just data" from what it's used for!
  - ***Clip controller***: plays back clips, or sequences of keyframes over time. The metaphor for this is "playhead".
    - *Name*: identifies controller by name; this is not the same as the clip which it controls. An example of why these names should be different: controller "Dan" controls clip "Animation Programming" (it doesn't make sense if you say "Animation Programming" controls "Animation Programming").
    - *Clip*: index of clip to control in referenced clip pool (see below). This ultimately tells you which clip is being played back by the controller.
    - *Clip time*: current time relative to start of clip; should always be between 0 and current clip's duration.
      - With one specific exception (see 'terminus actions'), this value will be contained in the half-open interval [0, *current clip duration*).
    - *Clip parameter*: normalized keyframe time; should always be between 0 and 1.
      - With one specific exception (see 'terminus actions'), this value will be contained in the half-open interval [0, 1).
    - *Keyframe*: index of current keyframe in referenced keyframe pool (clip references keyframe pool). This ultimately tells you your progress in the clip.
    - *Keyframe time*: current time relative to current keyframe; should always be

between 0 and current keyframe's duration.
        - With one specific exception (see 'terminus actions'), this value will be contained in the half-open interval [0, *current keyframe duration*).
    - *Keyframe parameter*: normalized keyframe time; should always be between 0 and 1.
        - With one specific exception (see 'terminus actions'), this value will be contained in the half-open interval [0, 1).
    - *Playback direction*: the active behavior of playback; try +1 for forward, 0 for pause, and -1 for reverse.
    - *Clip pool*: pointer (C/C++) or reference (C#) to the pool of clips that the controller will ultimately control. The clip pool contains the clip (whose index you have). Optionally you may create a pointer/reference to the clip being controlled within the pool, and the keyframe currently being processed (whose index you also have).
    - *Constructor/factory/initialize*: set starting clip, keyframe and state.
- ***Clip controller update***: This is the core functionality of any animation. Implement the update method for the clip controller, taking in only a *positive time step* (e.g. 1/30 if running at 30 fps; usually provided by engine as 'dt' or 'delta time' or something of the like) and performing the following algorithm:
    - *Pre-resolution: apply time step*: one-time, ***single line of code applied twice*** to increment *keyframe time* and *clip time* by the time step.
        - *Hint: numerical integration for time itself.*
    - *Resolve time*: ***while unresolved***, continue to use playback behavior to determine the new *keyframe time* and *clip time*.
        - The update behavior should be 'looping' such that when the clip reaches either end, it starts playing again from the opposite end in the same direction.
        - *Hint: There are **7 cases** to check for resolution, at least one of which definitively terminates the algorithm.*
    - *Post-resolution: normalize time/parameters*: one-time, ***single line of code applied twice*** to calculate normalized *keyframe time* and *clip time* (which we will need later). These values must be between 0 and 1.
        - *Hint: Clamping is not the answer. I shake my head and fist at you if you do this.*

- Here is a diagram of the process:


REAL-TIME moves forward in increments of dt (delta time)
KEYFRAME TIME has interval [0, duration)
PARAM is normalized time in [0, 1)
CLIP CONTROLLER is the 'playhead' that helps describe the sample/value at current time
CL


K0
K1
K2
K3
K4
K5
KEYFRAME is a known sample/value in time
CLIP CONTROLLER UPDATE: determine current KEYFRAME and KEYFRAME TIME (and CLIP TIME) when playhead moves -> 7 CASES
FIRST do time step
THEN resolve cases
FINALLY normalize
PRO TIP: normalize = divide ;)
CASE: REVERSE playhead rewinds towards previous t += -dt
CASE: PAUSED playhead stays put t += 0
CASE: FORWARD playhead advances towards next t += +dt
PRO TIP: time step itself is also driven by numerical integration: x(t+dt) = x(t) + x'(t)dt where 'x' can also be 't' itself
K(i-1)
'previous'
K(i)
'current'
K(i+1)
'next'
dt
CASE: REVERSE TERMINUS playhead passes clip start
CASE: REVERSE SKIP playhead enters previous keyframe
CASE: FORWARD SKIP playhead enters next keyframe
CASE: FORWARD TERMINUS playhead passes clip end

4. ***Basic testing interface***: Now you must demonstrate your clip controller in action. Remember: this is an abstract activity, so you only need to show that it works. Implement the following for your testing application:
   - ***Interface***: Set up the following in your testing application:
     - Instantiate and set up a pool of keyframes (20+) and clips (5+).
     - Instantiate and set up a few clip controllers (3+) to update in parallel so you can see them do their work. They may reference the same clip because you did such a great job decoupling.
   - ***Input***: Add controls for the following tasks:
     - Select clip controller to edit.
     - Play/pause controller playback.
     - Set to first/last frame in current clip.
     - Change clip to control.
     - Flip playback direction.
     - Slow-motion (multiply the time step by a factor of less than one).
   - ***Update***: Perform the following tasks in your update loop:
     - Apply input to controllers.
     - Update all controllers.
   - ***Display***: Display the following information:
     - Display all controllers and their info (all members) simultaneously ***as text***. This includes showing the *data* and *index* of the current keyframe on display (which is why it is important to make them different when setting up and initializing keyframes). Optionally, allow the user to also view the individual clip and/or keyframe information. This is the abstraction coming into play; there is no application, character, game, film, etc. to manipulate; only the controller doing its work on raw keyframe time and presenting the results.

     - Display user controls for interaction (e.g. " play forward >   play reverse <   stop | cycle clip , . ").
     - Display feedback to user (e.g. point at the clip controller you are currently editing).

Bonus:

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

- ***Terminus actions (+1)***: Add "terminus" or "end actions" to the clip controller interface and modify the update function to account for more than just looping. Implement the following behaviors:
  - Stop at end if playing forward or at the beginning if playing reverse
    - *Note: This is the only event in which your time measurements will equal their respective durations, and your normalized times (parameters) will equal 1.*
  - Loop or repeat at end if playing forward or beginning if playing reverse
  - Ping-pong or switch playback direction: continue playing reverse if hitting the end playing forward, or forward if hitting the beginning playing reverse

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++ and/or C#.
  - 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.
  - If you are using Unreal, all code must be implemented directly in C++ (i.e. without Blueprint).
  - 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

Criteria	Ratings			Pts
IMPLEMENTATION: Architecture & Design Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine.	**1 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 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 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 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 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 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 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 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 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 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