Project 1: Interactive Keyframe Control

Published Edit ⋮

GPR-450 Advanced Animation Programming
Instructor: Daniel S. Buckstein
Project 1: Interactive Keyframe Control

Summary:

In this project you will use the principles learned from lab 1 (controlling time as you know it) to further your expertise on keyframe and pose-to-pose animation. The purpose is to develop some algorithm, editor extension, tool, interactive demo, etc. to serve the purpose of keyframe control. Be sure to define your contribution and its purpose concisely. Whereas lab 1 was designed to introduce time as an abstract animation ***tool***, this assignment transfers that knowledge into some practical ***application***.

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.>

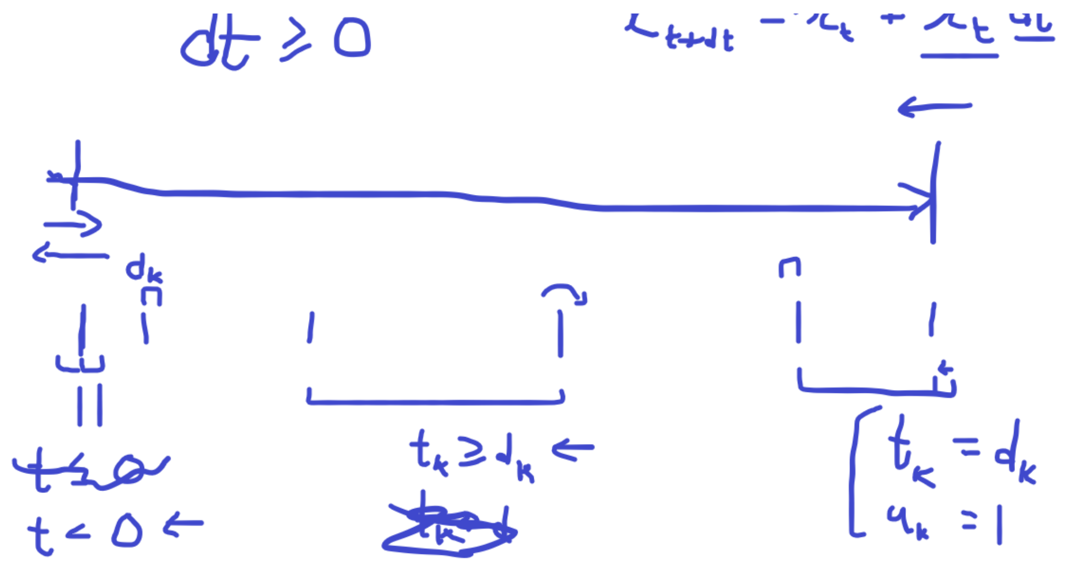
Finally, please submit a ***10-minute max*** demo video of your project. Use the screen and audio capture software of your choice, e.g. Google Meet, to capture a demo of your project as if it were in-class. This should include at least the following, in enough detail to give a thorough idea of what you 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:
Build some portfolio-worthy tool to show off the concepts discussed for lab 1 and beyond. We explore basic spatial poses as our primary application.

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. ***Lab 1 extension***: Complete all requirements of lab 1, then proceed to add the following support:
   - ***Clip transitions***: Add a data structure to describe a transition action. Upon reaching or passing either terminus of a clip, the controller should read the transition and follow its instructions.
     - The transition data structure should specify information about the target clip (clip pool and index) and the (re)initialization parameters for the clip controller when it triggers a transition.
     - In the *clip* interface, add a member for a forward transition (triggered when playing forward and the controller passes the end of a clip) and a reverse transition (triggered when playing in reverse and the controller passes the beginning of a clip).
     - Improve the controller's update algorithm such that when a clip controller triggers one of these transitions during the update routine, the controller switches to the new clip by following the transition's parameters, setting its own values to the resulting keyframe and clip. ***This becomes part of the keyframe time resolution; do not simply set the new time to zero unless the transition instructs it to do so.***
   - Here is a transition diagram:



2. ***Application***: All project concepts must demonstrate interactive control of *waypoints*, or positional key samples (just position), that can just be held in one or more arrays. Design and implement one of the following concepts to apply your keyframe control:
   - *Clip pool file format*: In this case, waypoints are 3D points in space. Implement a parser for a clip file format such as the one provided in the framework (*resource/animdata/sprite_anim.txt*). Use your controller to set some object's position to the waypoint whose index is described by *the current keyframe's data member* (recall this is just an integer). The object should appear to flash between positions any time the clip controller ticks. A clip is one sequence of waypoints that can play back and transition automatically with a clip control. The user can also manually select clips and keyframes.
   - *Sprite animation*: In this case, the waypoints are 2D cell coordinates in a sprite sheet. Simplify your life by using a sheet whose cells are the same size (in texture space, 1/sz). The sprite itself does not move in space, but *the current keyframe's data member* selects a cell to display in a clip; if a clip is one row in the sprite sheet, then a keyframe is a column. The sprite can play and transition on its own, but user can intervene any time to select a new clip.
   - *Graph editor*: Implement a visual and interactive editor for waypoint data, such as Maya's graph editor, or a timeline. Waypoints are broken down into three "channels": 'x, 'y' and 'z', all as a function of time, which can all be used to move positional data around as a function of time (e.g. x(t), y(t) and z(t) where the user can change the value at a given time). You will have one clip and clip controller per channel, and each will affect the position of a target object independently (e.g. 'x' can tick while 'y' remains constant).

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

- ***Complete another project (+3)***: Feeling confident about modularity and scalability in your design? Use the same systems in a second of the above project proposals.

- ***Linear interpolation (+2)***: Implement your own linear interpolation (LERP) algorithms for waypoints and use it to blend between waypoints in the set.
  - The general formula for LERP is:
    - $\mathrm{LERP}_{x_0,x_1}(u) = x_0 + (x_1 - x_0)\,u$
  - Values of 'x' represent your waypoints, where '*x0*' is current and '*x1*' is next. Your clip controller and its update algorithm will require additional considerations. The interpolation parameter '*u*' is already in your clip controller (*hint: it's one of the things you added for lab 1*).
    - *Clip pool file format*: Interpolate between 3D waypoints. This must remain consistent when transitioning between clips.
    - *Sprite animation*: Sprite animation is normally done using the "nearest" interpolation function (keyframe-by-keyframe), however interpolating between cells using LERP will produce a kind of film strip effect.
    - *Graph editor*: Draw lines between each keyframe data in the timeline; each channel should interpolate individually over time.

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

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	10
Submitting	a text entry box or a website url

Due	For	Available from	Until
-	Everyone	-	-

GraphicsAnimation-Master-x2

Criteria	Ratings			Pts
IMPLEMENTATION: Architecture & Design Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine.	**2 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.	**1 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.	2 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.).	**2 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.	**1 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.	2 pts
DEMONSTRATION: Presentation & Walkthrough Live presentation and walkthrough of code, implementation, contributions, etc.	**2 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.	**1 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.	2 pts
DEMONSTRATION: Product & Output Live showing and explanation of final working implementation, product and/or outputs.	**2 pts** **Full points** Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable.	**1 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.	2 pts

Criteria	Ratings			Pts
ORGANIZATION: Documentation & Management Overall developer communication practices, such as thorough documentation and use of version control.	**2 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.	**1 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.	2 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: 10