Project 3: Animation Blending & Layering

Published | Edit | ⋮

GPR-450 Advanced Animation Programming
Instructor: Daniel S. Buckstein
Project 3: Animation Blending & Layering

Summary:
This project integrates the outcomes of all of the projects and labs so far: project 1 covered fundamental keyframe and clip control, project 2 covered fundamental skeletal animation principles, and lab 3 introduced a variety of pose blending operations. Here we explore the data structures, algorithms and tools of animation blending and layering techniques by extending the skeletal and blending principles to entire animation clips.

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:
Furthering the theme of decoupled systems and data, the main goal of this assignment is to use spatial and hierarchical concepts to represent entire spatial pose and temporal clip data. The data can blend and flow as directed by organizational data structures such as blend trees.

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. ***Additional blend operations***: Implement the following spatial pose and hierarchical pose blend operations:
   - **Smoothstep/easing interpolate/blend/mix**: Implements an easing interpolation algorithm for poses. Note that pose components may have different rules for interpolation.
     - *Formats:* $\mathrm{smoothstep}_{P_0,P_1}(u)$*;* $\mathrm{easing}_{P_0,P_1}(u)$.
     - *Return*: smoothstep blend between control poses.
     - *Controls* (2): initial and terminal spatial poses.
     - *Inputs* (1): blend parameter; an input of 0 results in P0; an input of 1 results in P1; any other input between 0 and 1 results in an "easing" blend or mixture of the two control poses.
   - **Descale/bi-directional scale**: Similar to scalar division, this operation calculates the "descaled" pose, which is some blend between the identity pose and the inverted control pose. Also consider using this to repurpose the 'scale' operation to be bidirectional (allows a bi-directional parameter).
     - *Formats:* $\mathrm{descale}_P(u)$*;* $\mathrm{div}_P(u)$.
     - *Return*: blend between identity and inverse control pose.
     - *Controls* (1): spatial pose.

- *Inputs* (1): blend parameter; an input of 0 results in the identity pose; an input of 1 results in the inverted control pose; any other input between 0 and 1 results in some pose that is not identity but not quite the inverted control pose.

- **Convert**: Utility operation that performs the "convert" step for a spatial/hierarchical pose (convert raw components into transforms). This may simply update the control node itself, and return it as a pointer/reference for chaining, instead of returning an updated copy of the control pose.
  - *Formats:* $\mathrm{CONVERT}_P(\ )$.
  - *Return*: updated control pose.
  - *Controls* (1): control pose to be updated.
- **Revert/restore**: Utility operation that performs the opposite of convert (restore raw components from transforms); behaves in a similar fashion.
  - *Formats:* $\mathrm{REVERT}_P(\ )$; $\mathrm{RESTORE}_P(\ )$.
  - *Return*: updated control pose.
  - *Controls* (1): control pose to be updated.
- **Forward kinematics**: Utility operation that performs the fundamental forward kinematics operation, converting the provided local-space transform into the target object-space transform; behaves in a similar fashion as convert.
  - *Formats:* $\mathrm{FK}_{H,P_{obj},P_{loc}}(\ )$.
  - *Return*: object-space transform.
  - *Controls* (3): hierarchy; object-space transform; local-space transform.
- **Inverse kinematics**: Utility operation that performs the fundamental inverse kinematics operation, converting the provided object-space transform into the target local-space transform; behaves in a similar fashion as convert.
  - *Formats:* $\mathrm{IK}_{H,P_{obj},P_{loc}}(\ )$.
  - *Return*: local-space transform.
  - *Controls* (3): hierarchy; object-space transform; local-space transform.

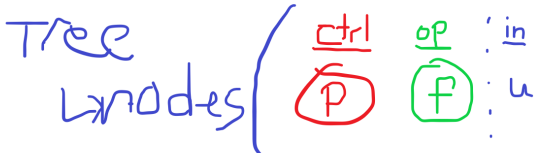
2. ***Animation blending and clip control***: The primary contribution for this project is to implement some tool or interface that artists/designers may use to configure animation clips and sequences. The interface must be flexible enough to enable ***animation blending*** and ***layering***. The traditional data structure used to organize animations in this way is a ***blend tree***, whose nodes represent the poses and blend operations described above and in lab 3.
   - Implement a few operations that take in ***clips*** as the control parameters and return a pose blended from the current time in each clip.
     - LERP: Perform key-pose interpolation on each clip, then LERP the two current poses.
     - ADD: Perform key-pose interpolation on each clip, then ADD the two current poses.
     - SCALE: Perform key-pose interpolation on a single clip, then SCALE the current pose.
   - You may use the existing hierarchy structure to organize the blend tree; it is just a

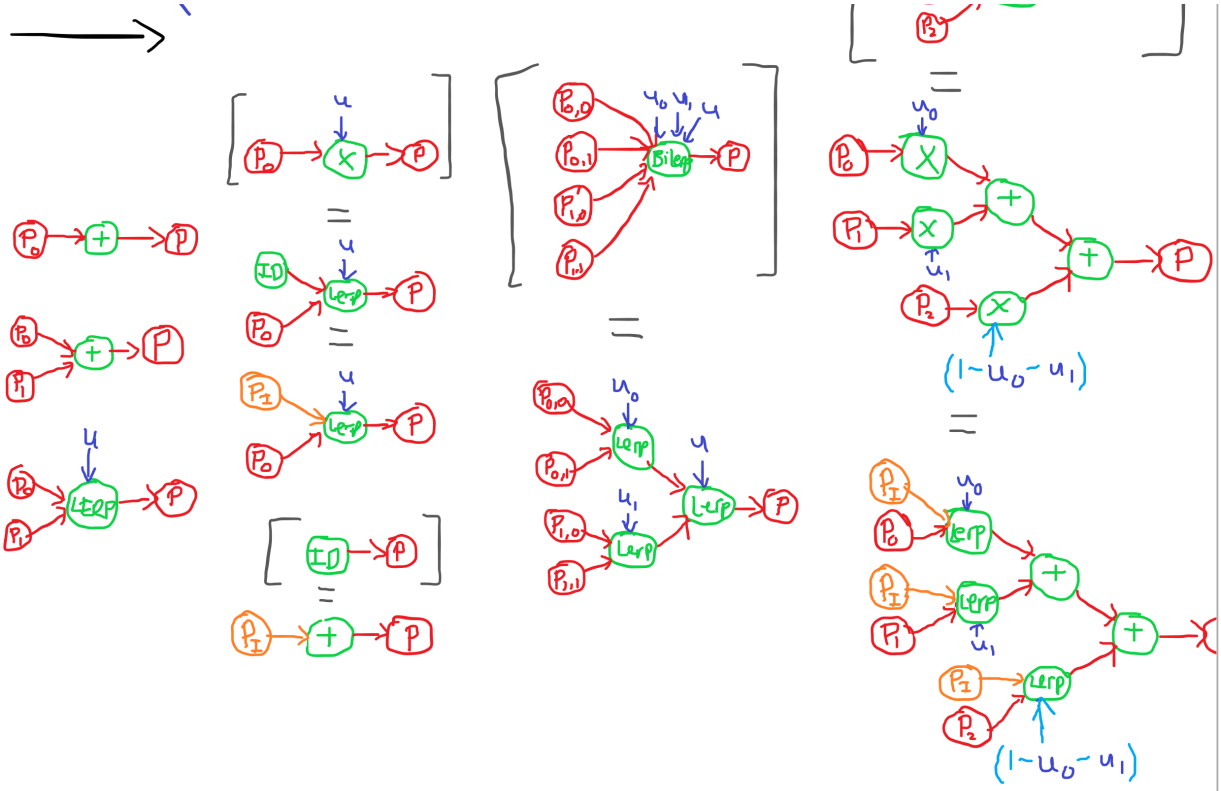
decoupled description of node relationships, after all!

- Create multiple clips and clip controllers to represent different animations and sequences. The clip controllers may be manipulated and played back independently.
  - Example: a "walk animation" may actually be the "sum" of 3 layered clips: legs walking, arms swinging and random body wobbling (re. live demo).
- Implement your animation pipeline in the form of a blend tree as follows:
  - Each clip controller determines which key poses to select and the interpolation parameter between them. These key poses are the "leaves" of your blend tree.
  - Perform an appropriate interpolation operation to calculate the current delta pose. The operation used represents another node in the tree, and the result can be fed into subsequent nodes as input.
  - Each delta pose calculated is a node, and so on. The final result is concatenated with the base pose (which may be saved as a persistent leaf node), converted and passed through forward kinematics to get the final pose. Per part one of this project, all of these operations may be represented as nodes; therefore the final output pose is your "root".
- Design and implement some interface to make this setup easier for artists and designers.
  - Example: an inspector/editor interface for editing blend operations in a tree.
  - Visualize the tree in action by displaying clip controller data, inputs, nodes, operations and each step in the tree until the final result is produced (display each input/output pose and operation from, leaf to root).
- Here are some resources:
  - Blend operations
  - Blend trees
  - **Animation Bootcamp: An Indie Approach to Procedural Animation (https://www.youtube.com/watch?v=LNidsMesxSE&ab_channel=GDC)**

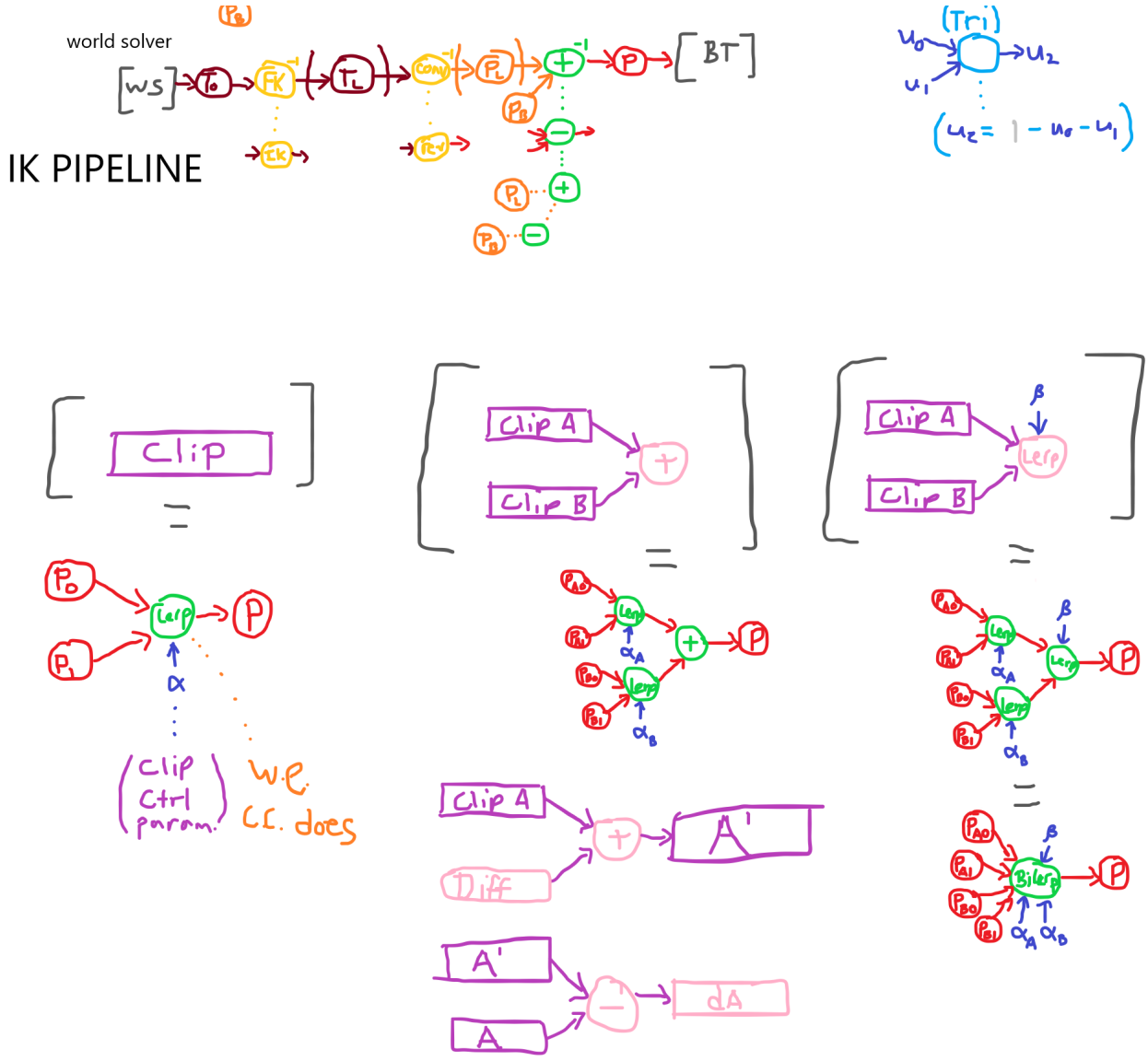
(https://www.youtube.com/watch?v=LNidsMesxSE&ab_channel=GDC)

- Some blend operations represented visually as their own blend trees:





blend tree FK PIPELINE



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

- ***Per-channel blending (+1)***: Write a utility to break down a blend tree into per-channel blend trees. I.e. the blend tree represents instructions for every active 1D channel (some float with respect to time). This extends to hierarchical blending.
- ***Cubic Hermite (+2)***: Implement and demonstrate a cubic Hermite operation for clips, where a clip may be used to describe a tangent handle for other clips.
  - The cubic Hermite function uses the "rate of change" of its primary control points to determine the shape of the "curve". In this case, control points are poses and rates

    of change are some displacement from those poses to others. An alternative version uses tangent handles, where the rate of change is the difference:
    - *Formats:* $\text{Hermite}_{P_0,dP_0,P_1,dP_1}(u)$; $\text{Hermite}_{P_0,Q_0,P_1,Q_1}(u)$.
    - *Return*: cubic Hermite spline interpolation between control poses.
    - *Controls* (4): initial and terminal spatial poses, and either their respective

"tangents" (dP) or "tangent handles" (Q).
    - *Inputs* (1): blend parameter; an input of 0 results in P0; an input of 1 results in P1; any other input between 0 and 1 results in a spline interpolation between the two control poses.
- ***Blend graph (+2)***: Upgrade the blend tree into a graph, which allows for reusable operations. You are basically building Blueprint, but each node is a pose or blend operation!

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 10

Submitting a text entry box or a website url

Due	For	Available from	Until
-	Everyone	-	-

GraphicsAnimation-Master-Range-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 to >1.0 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 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.	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 to >1.0 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 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.	2 pts
DEMONSTRATION: Presentation & Walkthrough Live presentation and walkthrough of code, implementation, contributions, etc.	**2 to >1.0 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 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.	2 pts
DEMONSTRATION: Product & Output Live showing and explanation of final working implementation, product and/or outputs.	**2 to >1.0 pts** **Full points** Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable.	**1 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.	2 pts

Criteria	Ratings			Pts
ORGANIZATION: Documentation & Management Overall developer communication practices, such as thorough documentation and use of version control.	**2 to >1.0 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 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.	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