Project 4: Procedural Animation & Character Controllers

Published Edit ⋮

GPR-450 Advanced Animation Programming
Instructor: Daniel S. Buckstein
Project 4: Procedural Animation & Character Controllers

Summary:

This project integrates the outcomes of all of the course so far: time control tools in project 1, spatial and hierarchical animation tools in project 2 and blend trees and layering tools in project 3. While these tools specialize in automating the animation pipeline, we must still account for player interaction (we're focused on making game technology, after all) and real-time decisions that can change the behavior of a character on the fly. Building on the principles explored in lab 4, we combine everything into a functional and interactive humanoid character controller. ***Note: This assignment is a good opportunity to prototype or get ready for your final project; keep it simple for now and work on polishing and integrating more later!***

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:
Put it all together by creating a humanoid character that feels good to control.

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:

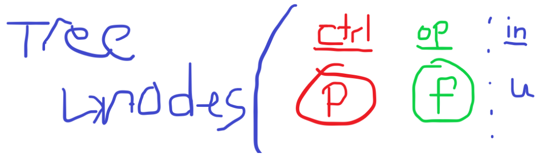
1. ***Character controller and manager***: Implement a data structure or interface responsible for managing the input, simulation and display of a generic humanoid character.
    - **Input**: Process control methods and buttons, such as keyboard, mouse and gamepad. Demonstrate all three input devices and your ability to swap between keyboard+mouse and gamepad.
    - **Simulation**: Process clip controllers, blending, apply inputs and set up for display.
    - **Display**: Display the skeleton in the current pose (skinned character model optional). Also enable a debugging mode with which you can toggle the input poses, and display the status of clip controllers.
2. ***Test character***: Implement a character for demonstration of your controller system. The starter character provided with lab 4 (branch '*anim/control*') will suffice. You will need multiple clip controllers and blend trees to make this work.
    - Implement the following behaviors:
        - **Transition between idle, walk, run and jump**: Use blending and control to trigger and blend between these four basic animation clips. For simplicity's sake and keeping this more in the scope of a prototype, these can be very simple clips, i.e. just a couple of keyframes. The point is to play with control and integrate the systems, not having the best assets.

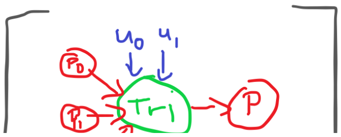
- **Procedural looking**: Set a target locator in the world (e.g. an imaginary light in the sky) that the character can use at will to adjust the orientation of the "neck" node. This may be achieved using the fundamental and simplest inverse kinematics (IK) pipeline: the neck transformation is set as a "look-at" matrix, and the IK pipeline is executed to bring that joint back to a "delta pose" that can be used for blending with the above clips.
- **Procedural grabbing**: Set a target locator in the world (e.g. an object or ledge) that the character can grab. This is achieved using a basic chain IK solver such as the one in the slides (by yours truly) or FABRIK (research). The end effector moves towards the locator when the character approaches, and the affected limb (e.g. arm) switches from the complete FK pipeline to an IK-FK blend. The effect is that the character appears to grab the locator.

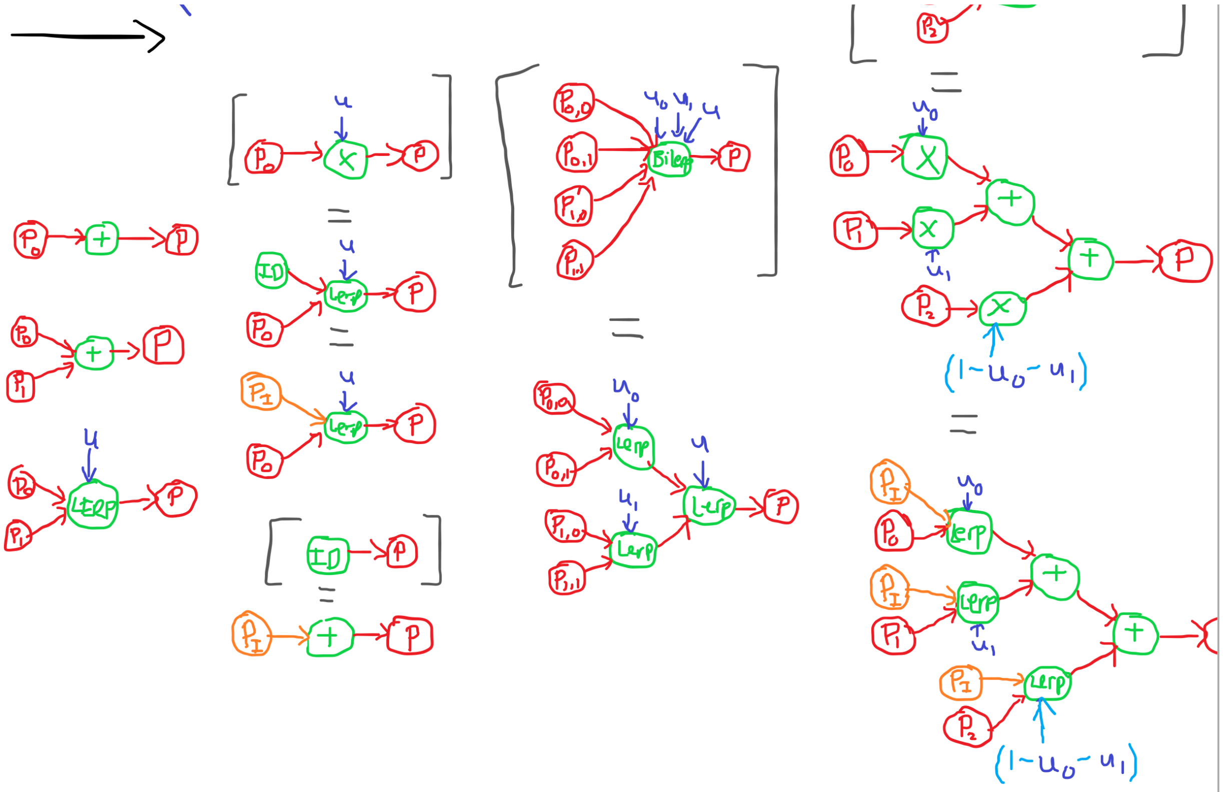
- Here are some resources, same as last time and then some:
  - Rotations & transforms
  - Skeletal IK
  - Blend operations
  - Blend trees
  - [Animation Bootcamp: An Indie Approach to Procedural Animation (https://www.youtube.com/watch?v=LNidsMesxSE&ab_channel=GDC)](https://www.youtube.com/watch?v=LNidsMesxSE&ab_channel=GDC)

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

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

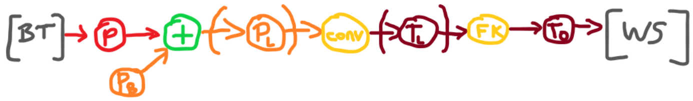




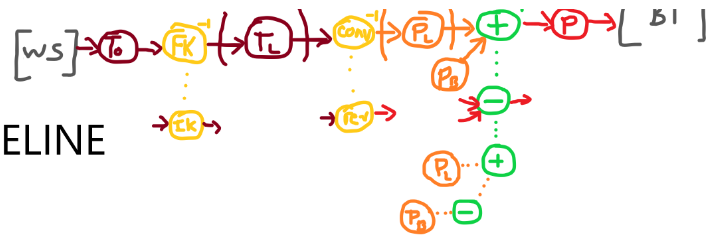


blend tree

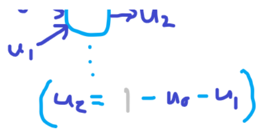
FK PIPELINE

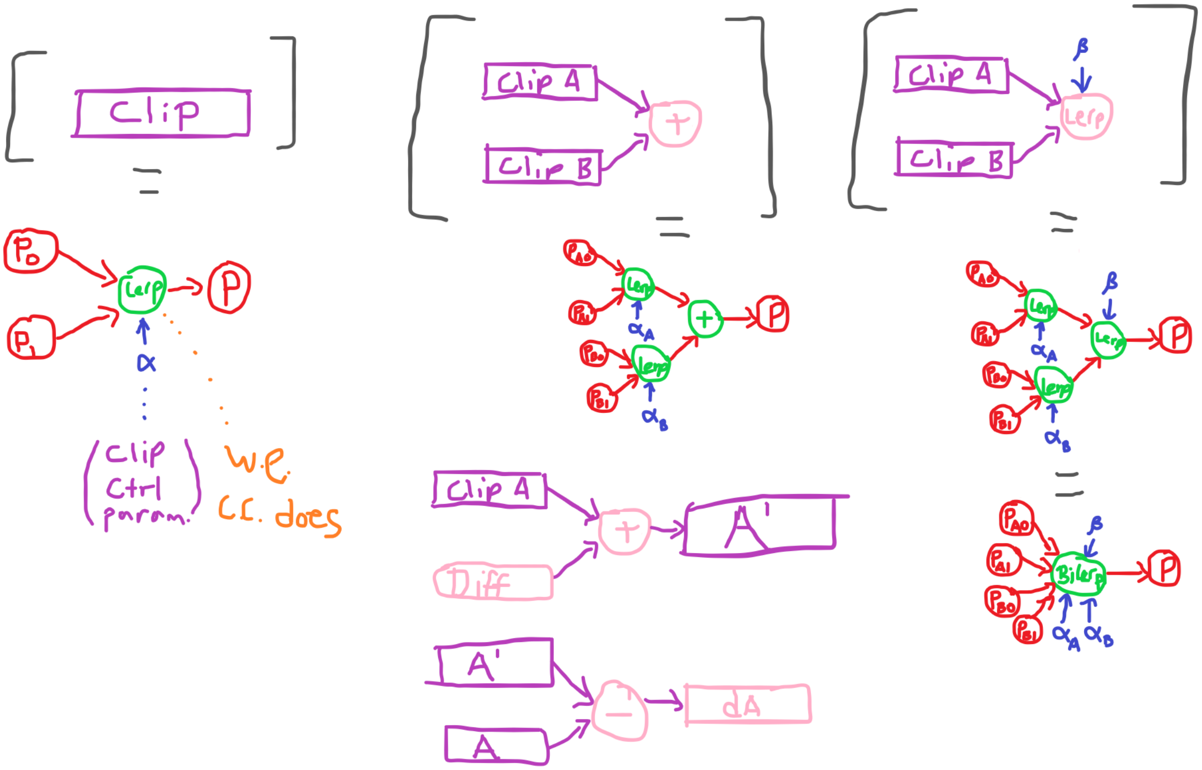


world solver



IK PIPELINE





$$\underleftarrow{R_{xyz} = R_x R_y R_z}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \end{bmatrix} \quad R_y = \begin{bmatrix} c_y & 0 & s_y \\ 0 & 1 & 0 \\ -s_y & 0 & c_y \end{bmatrix} \quad R_z = \begin{bmatrix} c_z & -s_z & 0 \\ s_z & c_z & 0 \end{bmatrix}$$

Euler Angle
Extraction
Rotation M

$\begin{bmatrix} 0 & S_x & C_x \end{bmatrix}$

- D.Buc

$C_x = \cos x \qquad S_x = \sin x$

$C_y = \cos y \qquad S_y = \sin y$

$C_z \; \cos z \qquad S_z \; \sin z$

The key is to first understand the fo
rotation matrix... then you can deco
and figure out the data used to ma

This example shows how a rotation
constructed if the Euler order is XY
the general format and trigonomet
extract the X angle.

$$R_{xyz} = R_{xy}R_z = R_xR_{yz}$$

$$= \begin{bmatrix} \boxed{C_y} & 0 & S_y \\ \boxed{S_xS_y} & C_x & -S_xC_y \\ -C_xS_y & S_x & C_xC_y \end{bmatrix} \begin{bmatrix} C_z & -S_z & 0 \\ S_z & C_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{xyz} = \begin{bmatrix} \underline{C_yC_z} & -\underline{C_yS_z} & \boxed{S_y} \\ S_xS_yC_z + C_xS_z & -S_xS_yS_z + C_xC_z & \left[\begin{matrix} \underline{-S_xC_y} \\ \\ \underline{C_xC_y} \end{matrix}\right] \\ -C_xS_yC_z + S_xS_z & C_xS_xS_z + S_xC_z & \end{bmatrix}$$

$\xcancel{y = \frac{\sin^{-1}(S_y)}{} = \sin^{-1}(R_{20}}$

$\xcancel{S_y = \cos y}$

$\operatorname{atan2}(s, c) \qquad \tan = \frac{\sin}{\cos}$

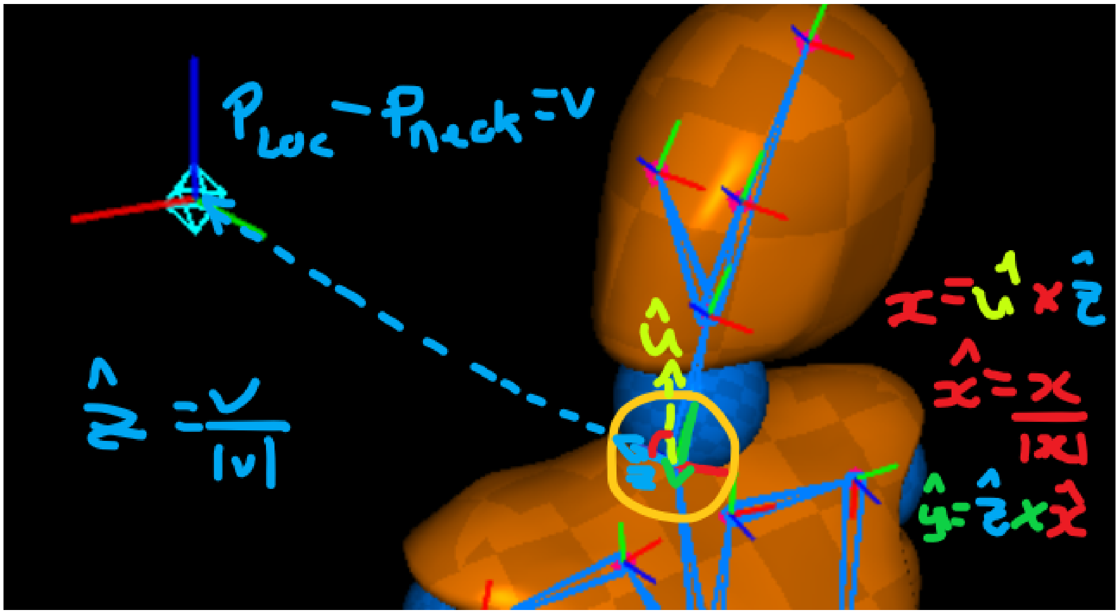
$\operatorname{atan2}(S_xC_y, C_xC_y) \longrightarrow = \underline{\operatorname{atan}\left(\frac{S_x\cancel{C_y}}{C_x\cancel{C_y}}\right)} = \left(\right.$

$= \operatorname{atan2}(-R_{2,1}, R_{2,2})$

"REVERT" blend operation (inverse of "CONVERT"): extract spatial pose from transformation matrix
1) extract translation, 4th column of matrix
2) extract scales, magnitudes of first three columns
3) normalize first three columns (divide by extracted scales), resulting in rotation matrix with above format
4) extract Euler angles using trigonometry on pertinent elements of matrix

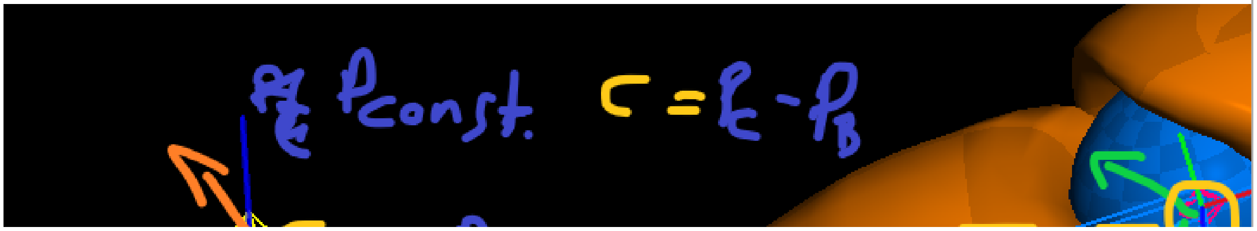
$\boxed{T} \rightarrow \boxed{\text{Rev}} \rightarrow$

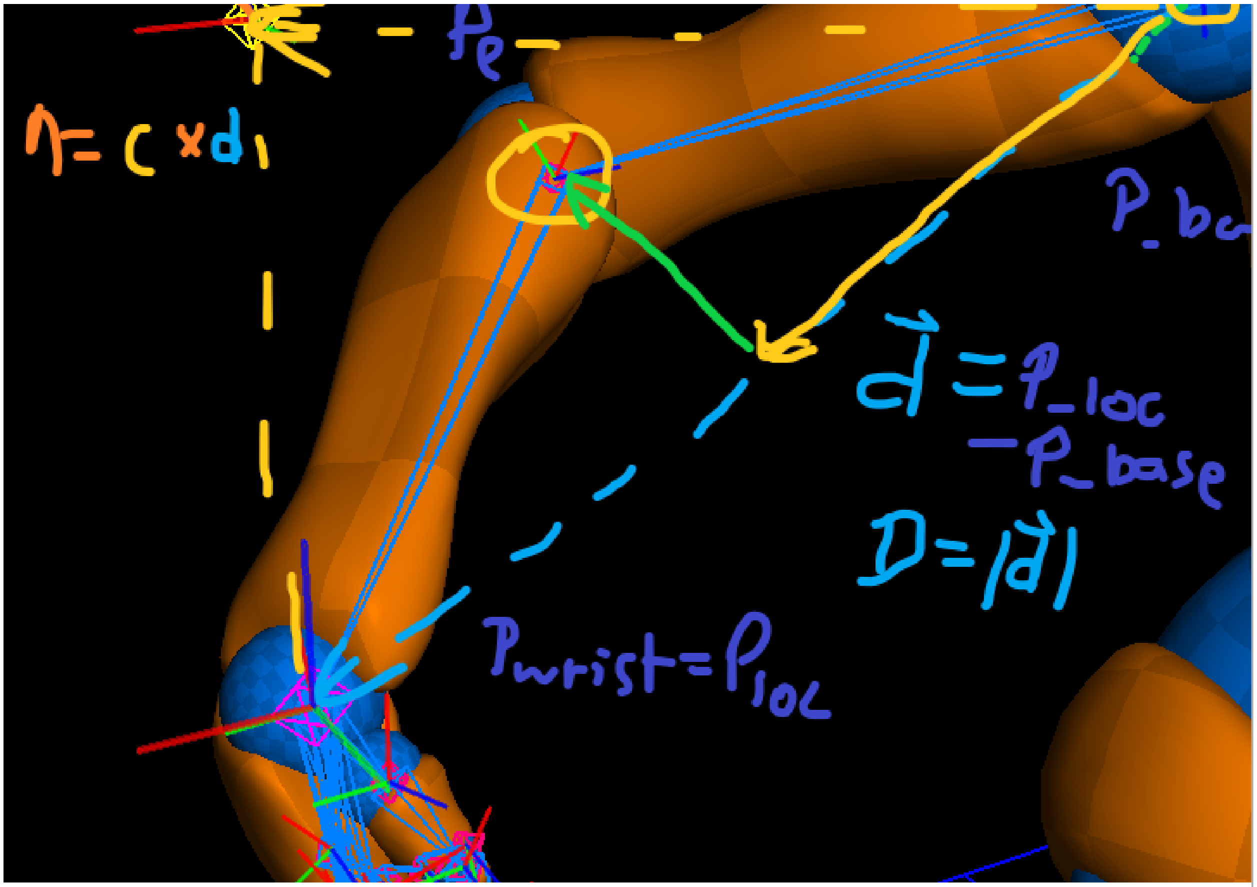
$T_{...} := [R$

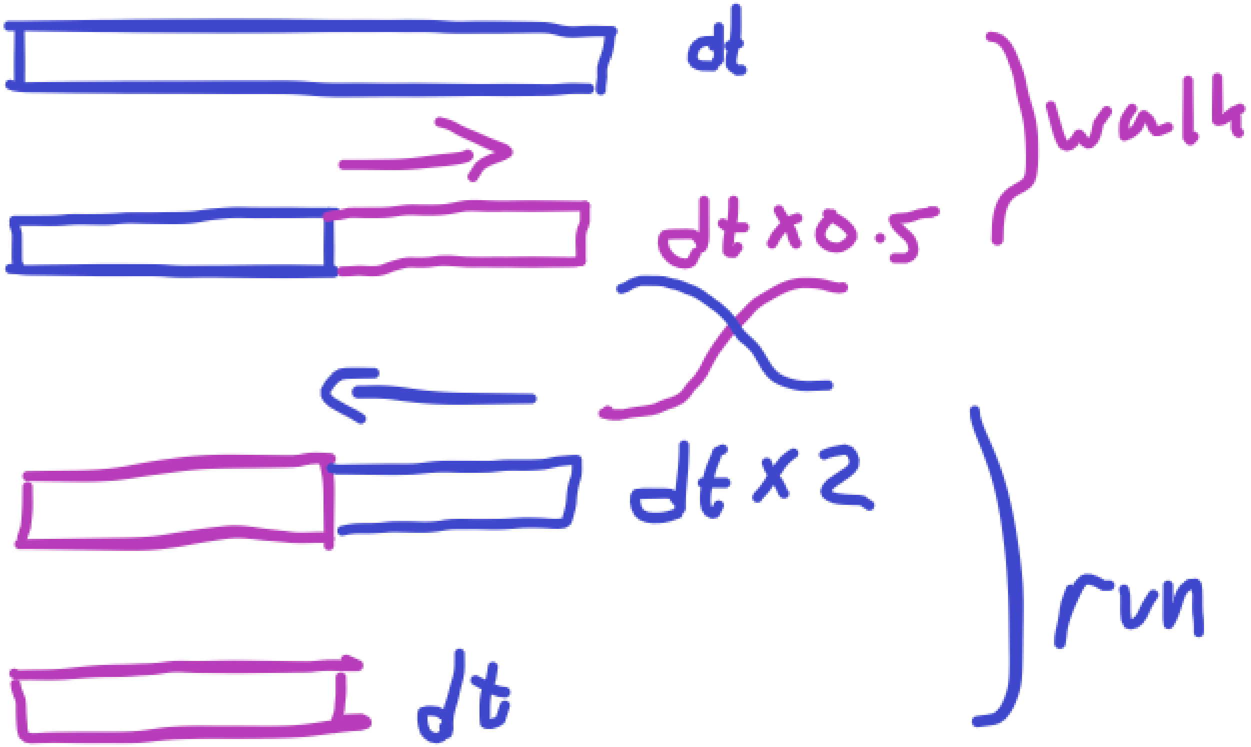

$P_{Loc} - P_{neck} = v$
$\hat{z} = \frac{v}{|v|}$
$\hat{u}$
$x = \hat{u} \times \hat{z}$
$\hat{x} = \frac{x}{|x|}$
$\hat{y} = \hat{z} \times \hat{x}$

$R = [\hat{x}\ \hat{y}$

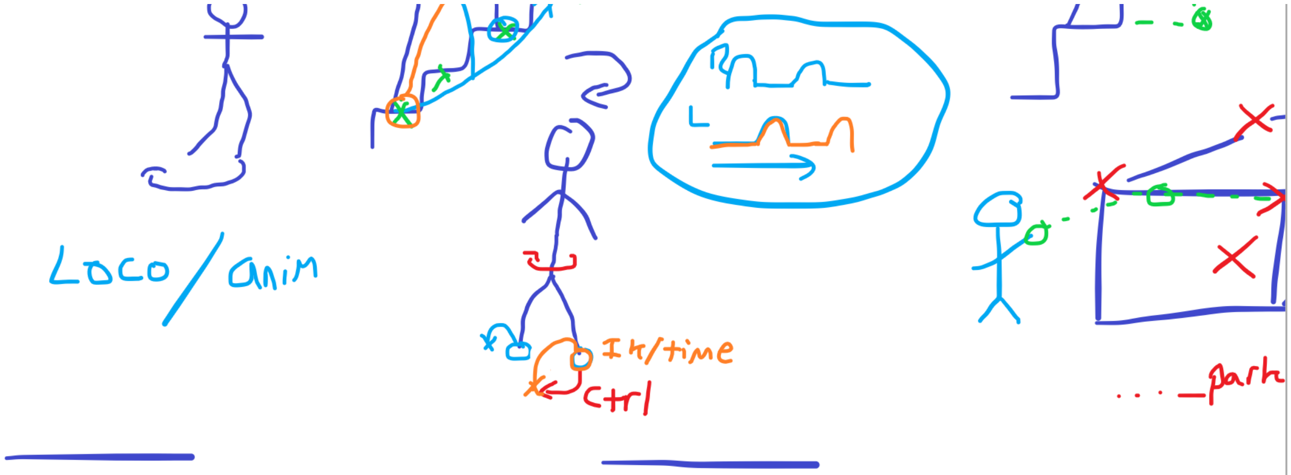
MN




P_e
$r = c \times d_1$
P_ba
$\vec{d} = P\_loc - P\_base$
$D = |\vec{d}|$
$P_{wrist} = P_{loc}$

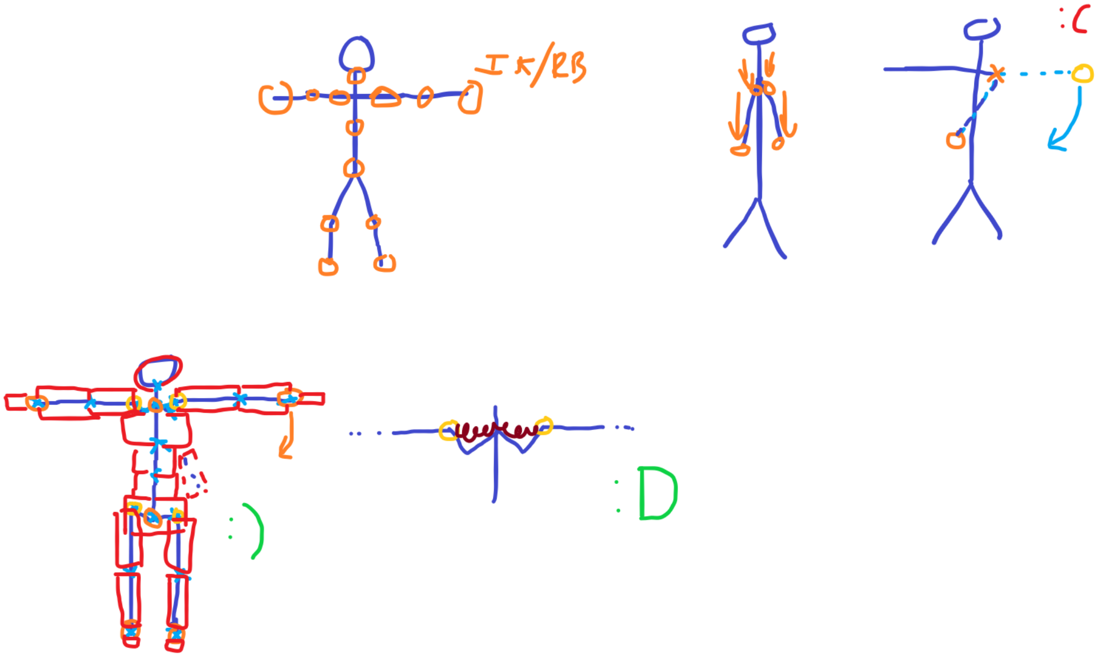

dt
dt×0.5
walk
dt×2
run
dt

1) Foot skate



2) Phys.-Based Anim.

E.g. Ragdoll



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

- ***Unlevel ground (+2)***: Implement some varying grounds to extend and improve the robustness of your control system. Examples:
    - A platform, onto and off of which the character can jump.
    - An inclined plane or stairs, exploring the issue of foot-skate.

    - A conveyor belt.

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