Assignment 4: Rigging & Skinning

CMSC 23700: University of Chicago
Due: March 1st 11:59PM

Introduction

To prepare a character for animation, you need to rig and skin the character so you can manipulate its pose and movement easily. Rigging provides the character with a skeletal structure that defines its set of available movements, and skinning sets up vertex-bone correspondences so the character's geometry(its mesh) moves naturally as the skeleton moves: each vertex will have a list of weights that specifies how a particular bone influences its position. After rigging & skinning, you only need to specify rotations at each joint to animate the character. In this assignment, you will implement rigid skinning and linear blend skinning for meshes. You will be using Polyscope to render the results interactively.

Required packages

- python >= 3.9
- numpy >= 1.21
- polyscope >= 2.4.0
- transforms3d (install with pip install transforms3d)

Visualization

In this assignment we make use of an interactive GUI implemented in Polyscope to visualize the mesh and vertex weights. Use the slider to change the angles of the corresponding joints and enable 'weights(vertex scalar)' for the skin mesh to check the weights. It's also helpful to set the transparency for the skin mesh to see the bones inside and enable 'Edges' on the skin mesh. It's not necessary to add new features to the visualization and the instructions below specifically prohibit doing so.

Submission Instructions

Your submission should consist of `joint.py`, `skin.py`, and `task3.py`, `task3.png`, `documentation.pdf`. **Do not** alter/implement additional files, functions, or class attributes (all the locations where you are required to modify have been marked with "**TODO**"). **Do not** import additional external libraries outside of the ones we've specified above.

If you are planning to use your late bank but are checking your assignment output using the autograder - put the following text in your **`documentation.pdf`** during submission testing: *Please do not grade this submission yet, I am using my late bank.*

Documentation

You should thoroughly and thoughtfully document your work in this project, **both in code** and in a **`documentation.pdf`** file you submit alongside the code and image.

- In **`documentation.pdf`**, document your high-level progress through the assignment: your initial planned approaches, any issues you ran into during implementation, and how you solved them.
- In **`documentation.pdf`**, draw and include an annotated diagram or sketch you use to derive your solution. Graphics algorithms are especially suited for the use of diagrams to aid understanding and debugging. **Please create and include at least one such diagram in `documentation.pdf` as you go through the assignment. Drawing by hand is fine. Diagrams from conversations with a TA/instructor are fine, but must be your own work.** Here are some examples of things you may want to visualize/sketch out for this assignment:
  - How you compute distance from a point to a line segment for linear blend skinning
  - How you decide on your added joints and their axes for the arm skinning task

  There is no hard standard for how detailed such a diagram must be, but they should be sufficiently illuminating in order to have helped you implement the subproblem in question.
- In **`documentation.pdf`**, cite all resources, online articles, Q&A threads, etc. you use in your code. In line with the Generative AI policy on the course website, you must also cite any use of Generative AI involved:
  - You must include the questions you used and GenAI answers (e.g. a link to the saved conversation if applicable; a text paste of the conversation otherwise)
  - You may **not** use GenAI to describe your code or write any part of `documentation.pdf`
  - **You must only use GenAI for the permissible uses described in the Generative AI policy on the course website.**
- In **code comments**, thoroughly document nontrivial lines and leaps of logic. We encourage documenting the shapes of numpy arrays, even if they may be obvious in the moment. Comment thoughtfully so that you may recall what you did upon re-reading your code in six months; consider the thought process of a hypothetical reader who is in the course but has not done this assignment.

Framework

The `joint.py` file contains the `Joint` class that represents a joint and its attached bone. The `skeleton.py` file contains the `Skeleton` class that stores all of the joints in a list. The skeletal hierarchy is organized in a way that every joint contains a reference to its parent and the root joint of the skeleton has parent `None`. Each joint in the hierarchy can be represented by a local transform with respect to its parent. The `position` attribute in the `Joint` class stores the position of the joint with respect to its parent. The `joint_axis` attribute represents the axis along which a joint can rotate. The `joint_angle` attribute stores the current rotation angle of the joint. The `length` attribute stores the length of the bone in the skeleton and it points to the `+X` direction by default.

The `skin.py` file contains the `SkinMesh` class that represents the mesh to which the skeleton is bound. The untransformed mesh vertices are stored in `original_positions`, and the transformed vertices are stored in `transformed_positions`. You can get original vertex locations and set new vertex locations with `get_vertex()` and `set_transformed_vertex()`. For smooth skinning, `weights` is used to store the weights for each vertex with respect to all the joints in a bound skeleton. You can use `set_vertex_weight()` and `get_vertex_weight()` to set and retrieve the weights. For Task 1 and Task 2, the skin mesh will be a cylinder that is animated by two bones.

Run `render.py` to visualize the specific task you are on. In particular, the weight visualization will help you understand the debug your weighting code correctly. *Note: If you run render.py before beginning the assignment, the slider elements will not respond to angle changes because you haven't implemented functions that deal with rotations!*

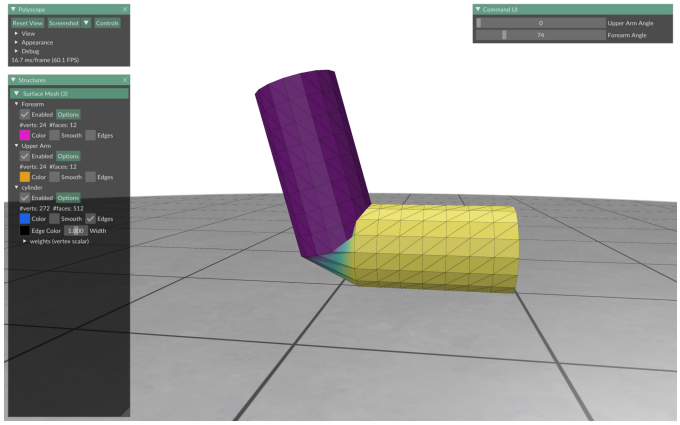

Figure 1: Weight visualization

Task 1: Rigid Skinning

In Task 1, you will implement rigid skinning through functions in `joint.py` and `skin.py`. Task 1 has the following subtasks, which should be completed in the order given:

Computing Transforms

In this subtask, you will compute the local, world, and binding transforms of each joint. Implement the following functions in `joint.py`, hints are provided for all functions: `get_local_matrix()`, `get _world_matrix()`, `compute_binding_matrix()`.

1. `get_local_matrix()`: returns the transformation matrix in the joint's local coordinate. Should be a composition of translation and rotation.

2. `get_world_matrix()`: returns the transformation matrix in the world coordinate. Should be a composition of the joint's parent's transform and the joint's local transform. *Note: make sure your* `get_local_matrix()` *is correct before implementing this function.*

3. `compute_binding_matrix()`: sets `self.binding_matrix` to the binding matrix which transforms a vertex in world space to local space. This is the inverse of the world matrix. *Note: make sure your* `get_world_matrix()` *is correct before implementing this function*

Computing Rigid Skinning

In this subtask, you will implement rigid skinning in the function `rigid_skinning()` in `skin.py`. For rigid skinning, each vertex is attached to one bone only. You can find which bone a vertex is attached to using `get_rigidly_attached_joint()`. Pseudo code is provided. Run Task 1 in `render.py` to visualize your result.

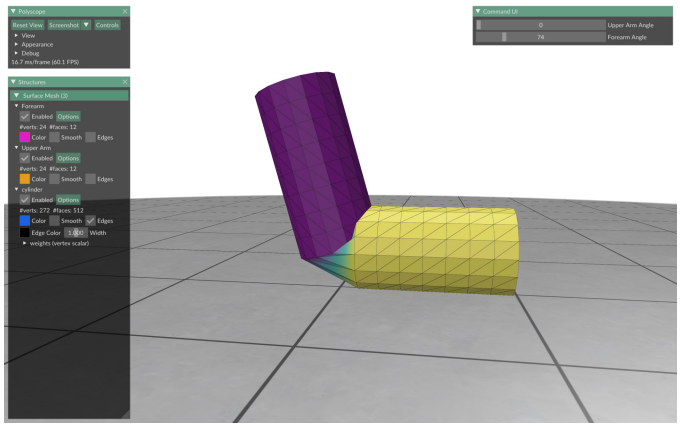

Figure 2: This shows correct rigid skinning

Task 2: Linear Blend Skinning

In Task 2, you will implement linear blend skinning, which creates smoother meshes in animation tasks.

Point-Line Segment Nearest Distance

In this subtask, you will implement `compute_distance_to_line_segment()` in `skin.py` to compute the nearest distance from a point to a line segment. *Note: this is slightly harder than computing distance to a line since you are calculating the distance to a finite length segment.*

Computing Vertex Weights

Once you have finished nearest distance, you will implement `compute_linear_blend_weights()` in `skin.py`. The vertex weight for a particular bone tell you how much that bone "influences" the vertex, and the weights are computed as a function of the distance(we consider the fourth power of the inverse distance: $\frac{1}{\text{distance}^4}$). Pseudo code is provided. *Note: you will have to normalize the weights of the vertices to make sure they sum up to* 1 *for each vertex.*

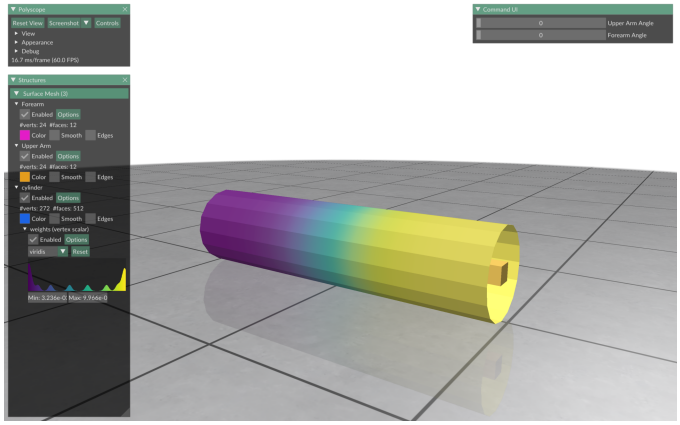

Figure 3: Correct weights should look like this

Linear Blend Skinning

In this subtask, you will implement `linear_blend_skinning()` in `skin.py`. In contrast to rigid skinning, now the transforms are weighted combinations of all transformed positions. See the pseudo code for details. Run Task 2 in `render.py` to see your result.

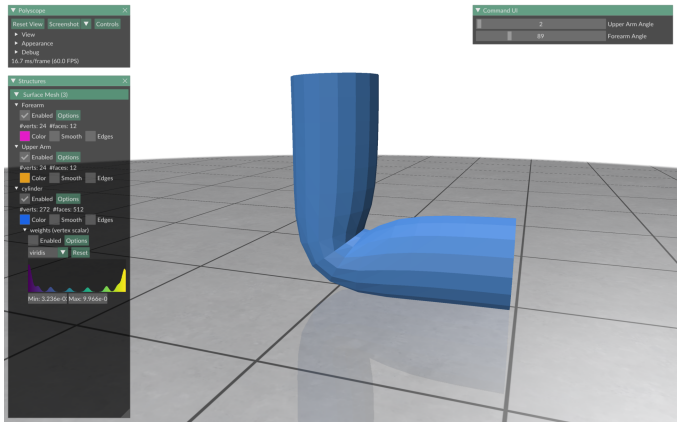

Figure 4: If your linear blend skinning is implemented correctly, it should look like this

Task 3: Arm Skinning

In Task 3, you will use linear blend skinning and rig a provided arm mesh. You will need to create new joints and place them along the arm. We have already added the locations for the first two bones. In particular, you should use additional joints at the wrist and within the fingers to increase realism of the skinning. To get full points, you don't need to be exhaustive and add multiple joints per finger, just make sure that the fingers can move separately from the arm/wrist is fine.

At minimum, you must implement the following:

- One new joint for each finger
- One new joint at the wrist
- Each new joint should allow the respective appendage to move **independently** of the other parts of the arm.

Go to the `Task3` class in `task3.py` and modify the code within `__init__`. You don't need to modify `render()` and `set_joint_angle()`, slider elements and bone meshes will be automatically added once you bind your custom skeleton. Run Task 3 in `render.py` to see your result. Once you have this working, **pose the arm mesh to your liking, take a screenshot, and submit this image as `task3.png`.**

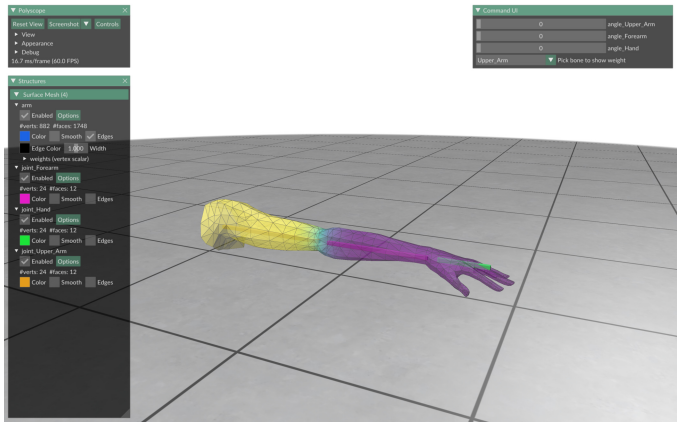

Figure 5: Skinning a custom mesh, notice the extra bone at the wrist.

Acknowledgements

The ideas in this assignment heavily borrow from the hard work of Wojciech Jarosz, Benedikt Bitterli, and Dario Seyb at Dartmouth. This assignment was also created by UChicago student Jack (Zihan) Zhang.