2025-1 Computer Graphics Homework Report

Hierarchical Model of Crawling and Attacking Snake

Moon Ju Hyeon (2020-19669)

1. Features Overview & User Guide

1.1. Overview

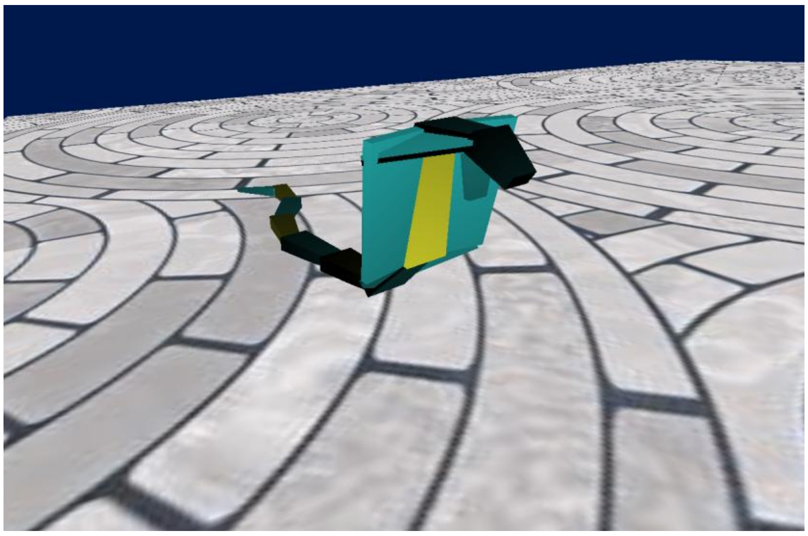
Figure 1. Program running example

This project is an implementation of the cobra snake model that consists of 17 components including bodies, tail, necks, side wings, head and jaw. All components are generated using projective transformation and non-uniform scaling from the basic geometry such as cubes or triangular pillars. Generated components are connected to each other by hierarchy and have 2-DOF R-R joints or 1-DOF R joints. There are 5 different movements that can easily be triggered by pressing a number key, and transition from one movement to another is interpolated smoothly. Finally, a simple rotating ground image is also available, which makes the snake movement more realistic and aesthetically appealing.

1.2. Installation

Instructions on how to install and run the program are written in the README.md file. Assuming you properly cloned or downloaded the repository, you can simply activate the Conda virtual environment and see the result by running below lines.

```
cd path/to/cloned/repo/
conda env create -f environment.yml
conda activate snake_env
python main.py
```

1.3. How to Play

Use keyboard inputs (num 1~5) to control the movement of the snake. You can try various combinations of multiple movements and orientations (e.g. move+lift head, stop+attack). Also, use mouse inputs to control the view. Grab and drag to change the view angle and scroll to zoom in or out.

Input	Control
key 1	Start moving forward. Direction is not adjustable.
key 2	Stop moving forward.
key 3	Lift head. This is a prerequisite of key4: attack.
key 4	Attack! This is available only after key3: lift head.
key 5	Lower head.
mouse drag	Change view angle (Trackball viewer).
mouse scroll	Zoom in/out

Table 1. Control guidelines

2. Implementation

2.1. Shapes and Orientation (geometry.py / snakemodel.py)

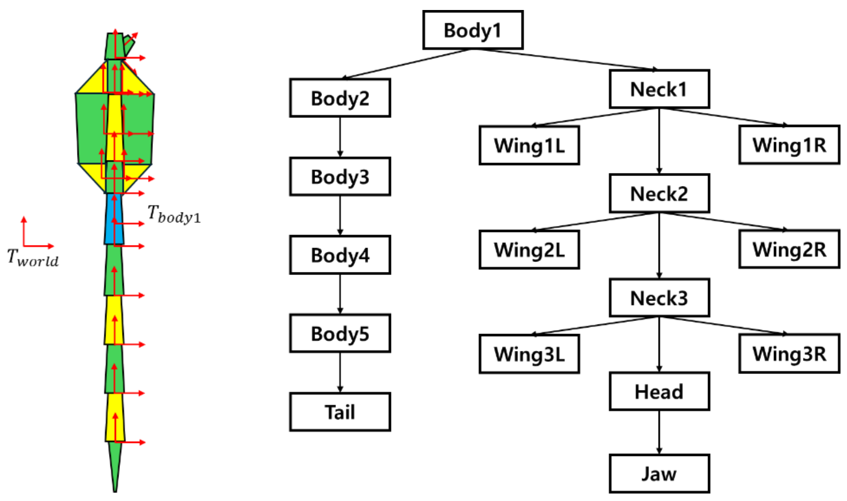

Figure 2. Overall model hierarchy

Non-uniform scaling is used for cobra wing geometry. 1st and 3rd wings are deformed from triangular pillars to a long triangular plate. Detailed images are shown in the figure below.

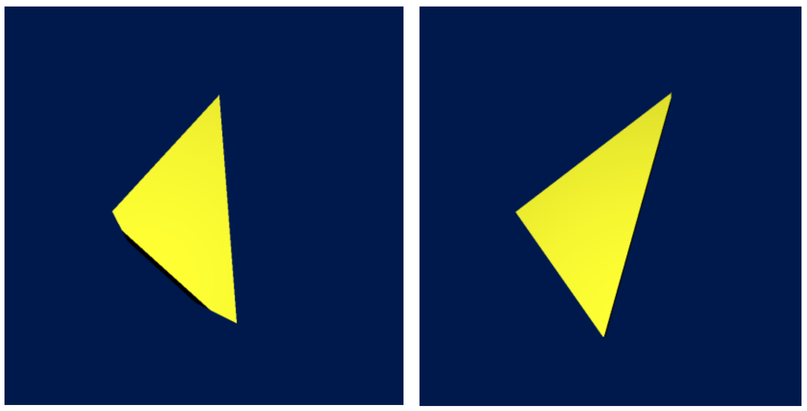

Figure 3. Non uniform scaling example

Projective transformation is utilized to create all other geometries, to create tapered rectangular plates.

Example image and transformation matrix used are shown in the figure below.

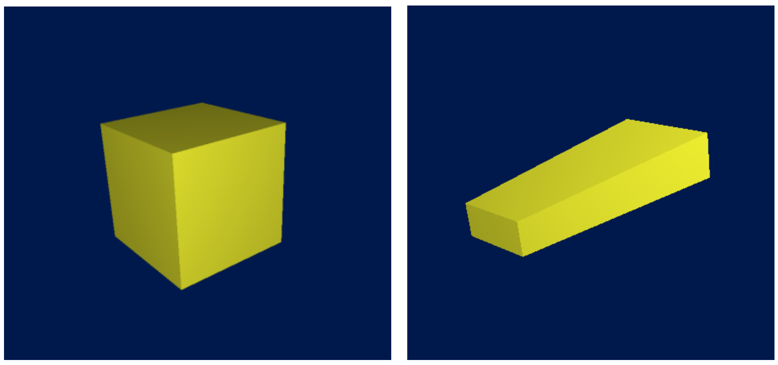

Figure 4. Projective transform example

To efficiently manipulate the orientation of each shapes, new functions move() and deform() are implemented in the module "geometry", within the primitive shape classes. Default pyglet geometries consist geometry.matrix object which returns current transformation matrix of the object. I separated this into geometry.movement and geometry.deformation and multiply it to the whole matrix whenever one of them is updated. The geometry.deformation saves object shape deformation and basis adjustments. The geometry.movement saves the orientation. Deformation is fixed after a snake instance is initialized, and all the hierarchical trees and movements are updated in movement matrix only.

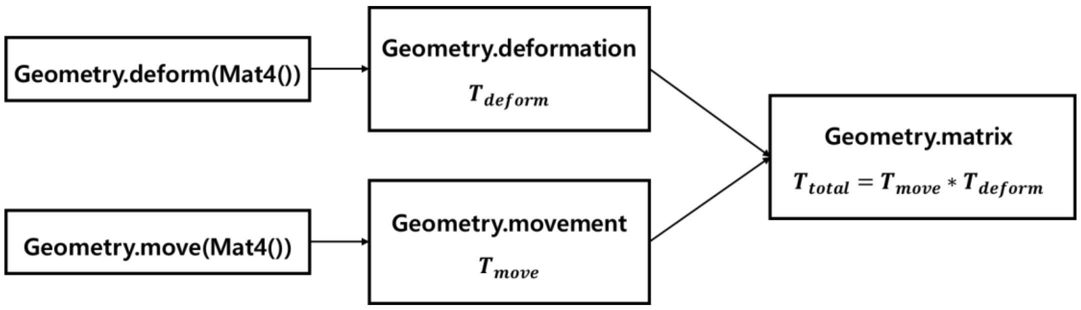

Figure 5. Orientation update method schematic

Orientation of a snake instance is updated by calling set_orientation() function. This function receives angles of each joint, convert it into a transform matrix, apply local joint angles to each component, and apply hierarchical movement passed down from the higher-level nodes. Body1, which is a root node, has 6 DOF from the world coordinate. Body2~5, neck1~3, head, and all wings are connected to its parents by 2-DOF R-R joint (bend and tilt). Jaw is connected to the head by 1-DOF R joint (bend only).

2.2. Motion (motion.py / utils.py)

Joint angles for a specific orientation are stored as a dictionary object. Functions for superposition and interpolation between different orientation dictionaries are implemented separately, allowing multiple motion sequences to overlap or transition smoothly.

For animation, a timeCounter class is implemented in utils.py. It tracks elapsed time in each program loop, enabling precise timing after command inputs and ensuring smooth execution of both repetitive and transient motions.

The forward motion of the snake is generated using a phased sine function to create a wave along the tail. To enhance realism, observations from real-world snake movement were incorporated, resulting in greater wave intensity in the tail compared to the body. Additionally, the ground texture rotates in a circular motion, reinforcing the illusion of forward movement. When stopping, the motion transitions smoothly by interpolating from the moving angles to a resting position.

$$\theta(i) = \left(\frac{1}{2} + \frac{i^2}{18}\right) A\sin\left(\frac{t}{w} + \varphi + fi\right)$$

$$f = frequency, A = wave\ amplitude, \varphi = initial\ phase, w = wave\ width$$

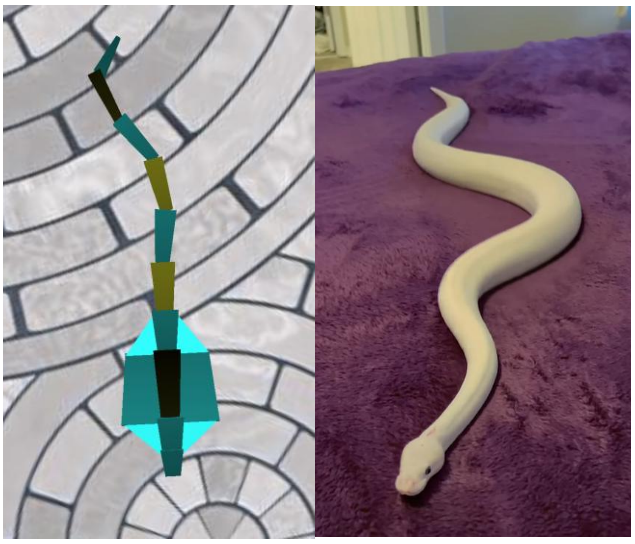
Figure 6. Motion comparison with the real snake

Lift head and lower head motion move the head up and down like a cobra observing an object or getting ready for an attack.

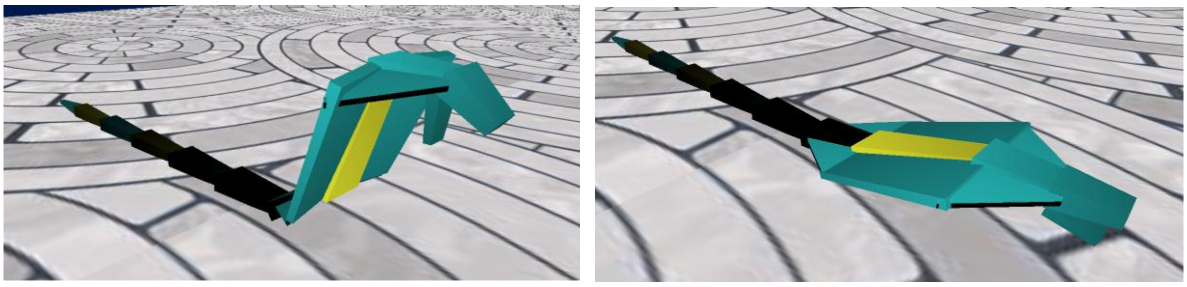
Figure 7. Lift and lower head motion

Attack motion lowers head and bites the object in front of the snake very quickly. This is only triggered when the head is lifted.

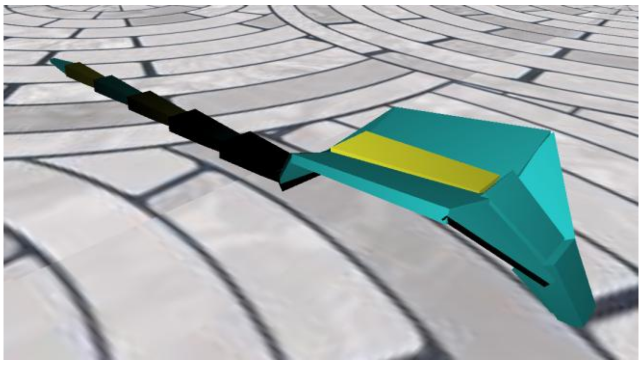
Figure 8. Attack motion

2.3. Control (main.py / utils.py / camera.py)

Control functions are mostly based on computer graphics course homework example codes. Only additional feature is a eventWatcher class, that saves the current status of a model or control inputs. Motions are assigned to the number key inputs 1~5. Also, camera zoom-in and out is also added for convenience.

3. References

[1] SNU MRL, Trackball Viewer, GitHub Repository. Available: https://github.com/snumrl/TrackballViewer

[2] SNU Intelligent Motion Lab, SNU Computer Graphics, GitHub Repository. Available: https://github.com/SNU-IntelligentMotionLab/SNU_ComputerGraphics