# Procedural Modeling & Animation: Moose Automaton

**Creator & Author**: Lyuying Guo

**Software**: Houdini 20.5.278

**Languages**: VEX, HScript

**Class**: SCAD VSFX721 Project3

**Professor**: Deborah Fowler

**Academic Quarter**: Spring 2025

**Document creation time**: May 26, 2025

## Statistics

**Rendering**

| | |
|---|---|
| **Renderer**: | Karma CPU |
| **Average Render Time**: | 1.8 min/frame (SCAD Render Farm) |
| **Resolution**: | 1920 x 1080 |
| **Number of Lights**: | 3 (1 Dome Light, 1 Distant Light, 1 Area Light) |

**Sampling** (Karma Render Settings default)

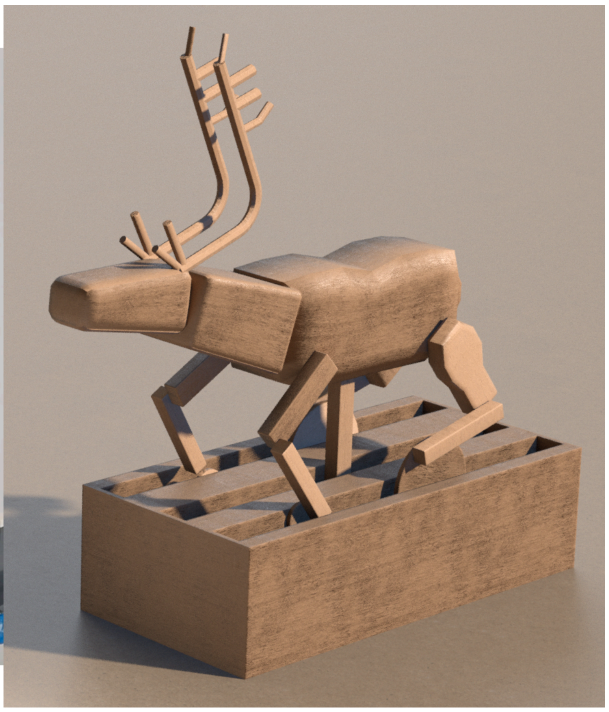
| | |
|---|---|
| **Primary Samples**: | 9 |
| **Diffuse Limit**: | 1 |
| **Reflection Limit**: | 4 |
| **Refraction Limit**: | 4 |
| **Volume Limit**: | 0 |
| **SSS Limit**: | 0 |
| **Color Limit**: | 10 |

**Geometry**

| | |
|---|---|
| **Primitives**: | 1,501 |
| **Points**: | 2,367 |
| **Mesh**: | 15 |

## General Description

The prompt of this project is to procedurally create a mechanical moving device in Houdini. I chose to imitate the mechanics of a LEGO moose automaton to explore how procedural modeling and animation can be utilized to create the impression of organic life.



Left: Reference

Right: Final Render

## Technical Breakdown

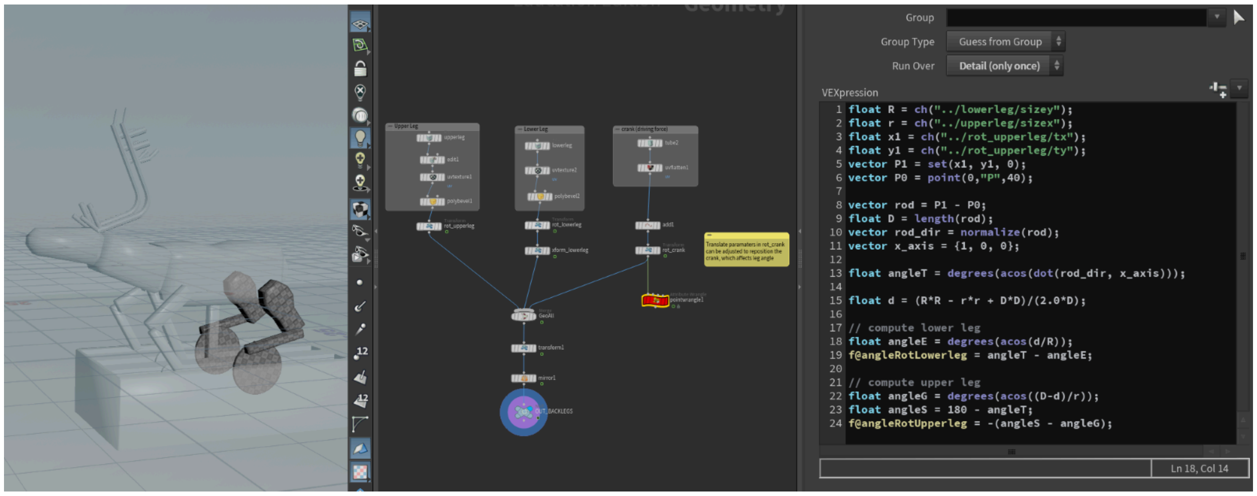
## Procedural Animation

**Mechanism 1: Running Legs**

**Problem**: 2-Point Constraint Animation

**Solution**: Dot Product, Pythagorean Theorem

The core problem to solve in this project is to “fake” a running leg mechanism with math. No RBD simulation is used. My core solution is to construct a triangle from the upper leg, lower leg, and an imaginary rod that connects those two. Utilizing the dot product function, dot(), in Houdini VEX, I calculate the angle between two existing edges with known values. Afterwards, I use those angle values to calculate how much the upper leg and lower leg need to rotate from a referential x-axis in order for the triangle relationship to exist, regardless of the frame we are on ($F).



Group Type: Guess from Group
Run Over: Detail (only once)

VEXpression

```
float R = ch("../lowerleg/sizey");
float r = ch("../upperleg/sizex");
float x1 = ch("../rot_upperleg/tx");
float y1 = ch("../rot_upperleg/ty");
vector P1 = set(x1, y1, 0);
vector P0 = point(0,"P",40);

vector rod = P1 - P0;
float D = length(rod);
vector rod_dir = normalize(rod);
vector x_axis = {1, 0, 0};

float angleT = degrees(acos(dot(rod_dir, x_axis)));

float d = (R*R - r*r + D*D)/(2.0*D);

// compute lower leg
float angleE = degrees(acos(d/R));
f@angleRotLowerleg = angleT - angleE;

// compute upper leg
float angleG = degrees(acos((D-d)/r));
float angleS = 180 - angleT;
f@angleRotUpperleg = -(angleS - angleG);
```

This method is adapted from the sample file created by Dr. Deborah Fowler, my professor for this class. The sample file builds the Dot Product calculation with Pythagorean Theorem from scratch. I re-wrote the code in a way that approaches the problem from vectors and dot product and utilizes the dot() function in Houdini in order to avoid typos in Pythagorean Theorem calculations.

I built one leg with this method, then transformed and mirrored copies to form the other three legs. The two front legs have slightly different speeds to imitate the rhythm of a real moose running, which is achieved by adjusting the Rotate Z parameter in rot_crank (Transform node) differently. Rotate Z parameter has a HScript expression containing $F to make the animation change per frame.

## Mechanism 2: Rotating Neck & Head

**Problem**: Rotation around a designated point

**Solution 1**: Pivot Transform (higher-level, artist friendly)

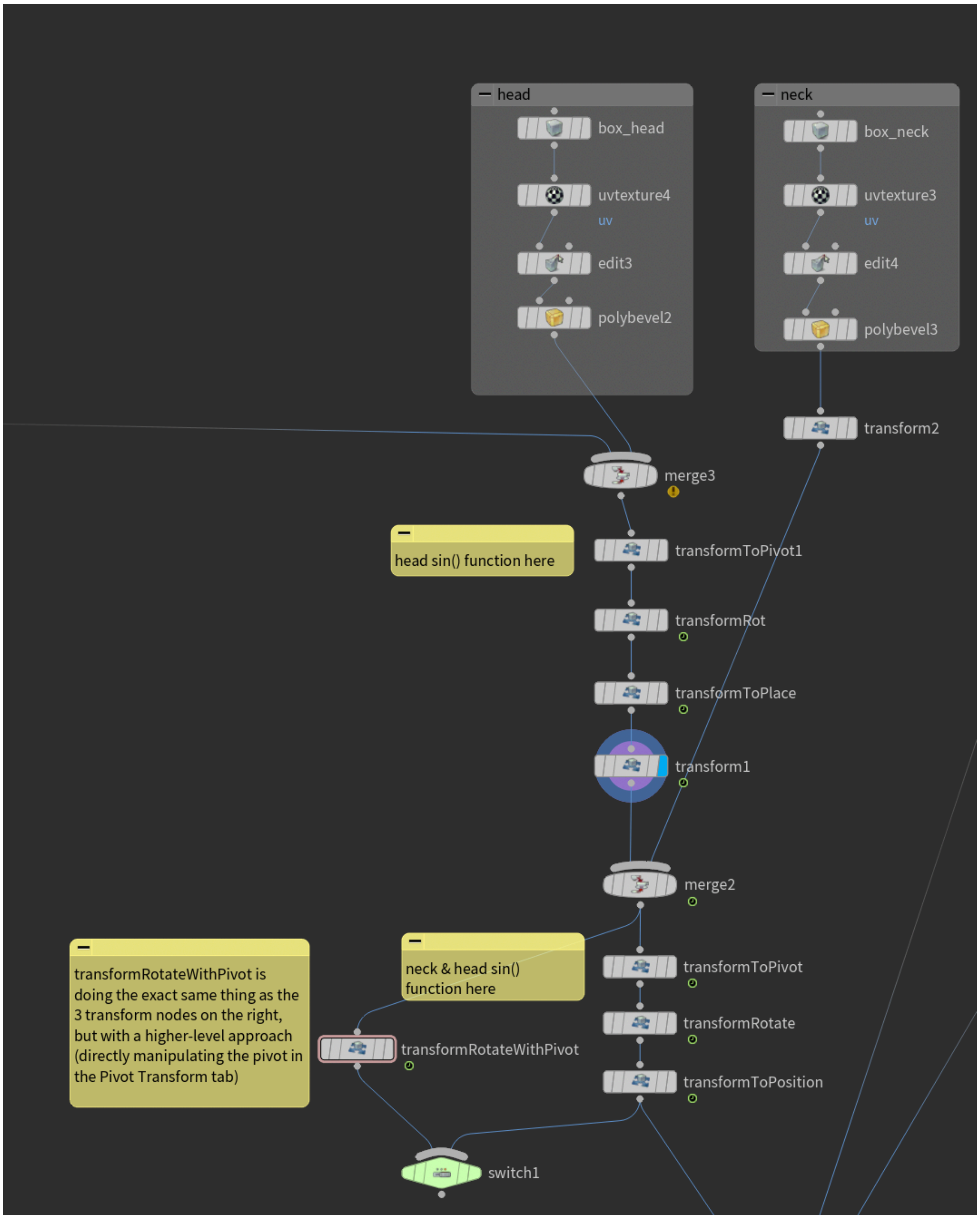
**Solution 2**: Matrix transformation (lower-level, graphics engineer friendly)

The core idea is to move the pivot of the neck from the center of the mesh to one side (base of neck). To do that, the most straightforward way is to manually move the pivot, which is achieved by adding a Transform SOP after the neck mesh, and manually type in (or drag) numbers into the Pivot Translate parameters under the Pivot Transformation drop-down menu. (P.S. Houdini does not offer a way to directly manipulate pivots in the viewport like the Maya “d” hotkey.)

Another graphics-engineer-friendly approach utilizes the logic of OpenGL matrix transformation. We create a neck at world origin, translate the neck to where it can rotate naturally around the origin (neck’s pivot. * Any Houdini mesh’s real pivot is always at world origin unless you explicitly change it in a Transform SOP > Pivot Transform), then move that whole system (mesh + pivot) to where you want it to be in the scene.

To make the neck rotate around the tip of the neck, we add a copy of the same system on top and merge it into the neck.

Both methods showcased below – the Switch node is to switch between the two methods.



head sin() function here

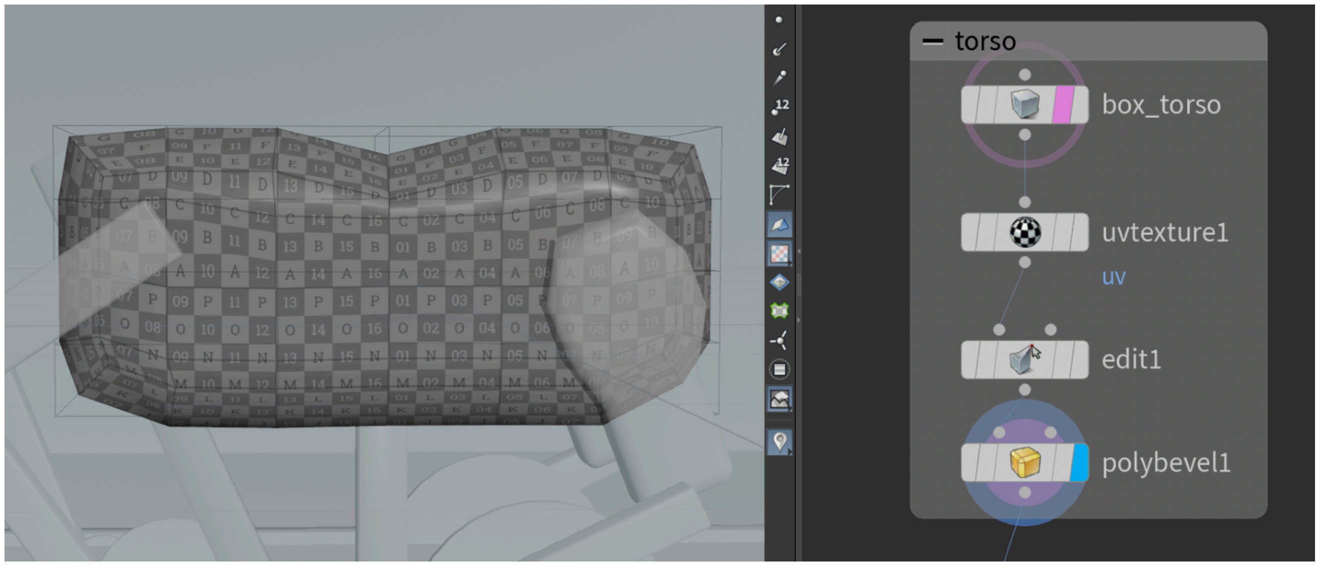
neck & head sin() function here

transformRotateWithPivot is doing the exact same thing as the 3 transform nodes on the right, but with a higher-level approach (directly manipulating the pivot in the Pivot Transform tab)

# Procedural Modeling

## Model Type 1: Organic Boxes

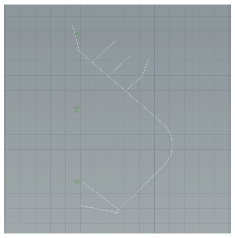
Most of the pieces that make up the body of the moose are made from a box with divisions, then using an Edit SOP to manually manipulate the points in the viewport, then applying a Poly Bevel SOP afterwards to smoothen the sharp edges. A key point here is to unwrap UVs *before* the SOP where I animate the models so that the models won't swim in textures.

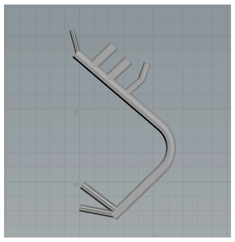


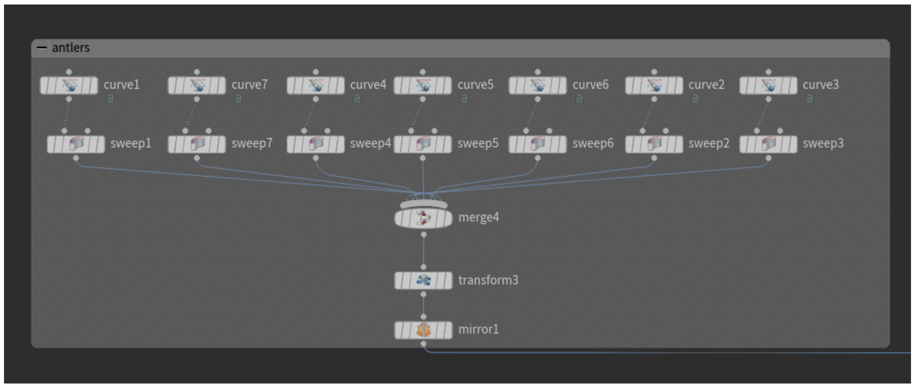
## Model Type 2: Curves

**Model Type 2: Curves** (*Method credit to my classmate Itim Kongsakulvatanasook)

All the parts making up the antlers are created with a Curve SOP with Primitive Type set to Bezier Curve, then applying a Sweep node after the curve with Surface Shape set to Round Tube to generate the mesh. The "Create Rounded Corners" option in the Curve SOP is extremely helpful for creating the major angle in the antlers.







# References

Reference LEGO moose automaton:
https://youtu.be/L6EW7HEzHT0?si=Sm3Em0w2p__wJ5lD

Solving 2-Point Constraint problem with Dot Product explanation:
https://deborahrfowler.com/MathForVSFX/DotProduct.html

Sample Houdini file that solves 2-Point Constraint problem with Dot Product in VEX:
https://deborahrfowler.com/MathForVSFX/DotAndPythagorean/H18/dotPythagoreanInActionColorPointWrangle.hipnc

Sample Houdini file that explains transformation order in Houdini:
https://deborahrfowler.com/HoudiniResources/FAQ/Transformations/TeapotTransforms.hipnc