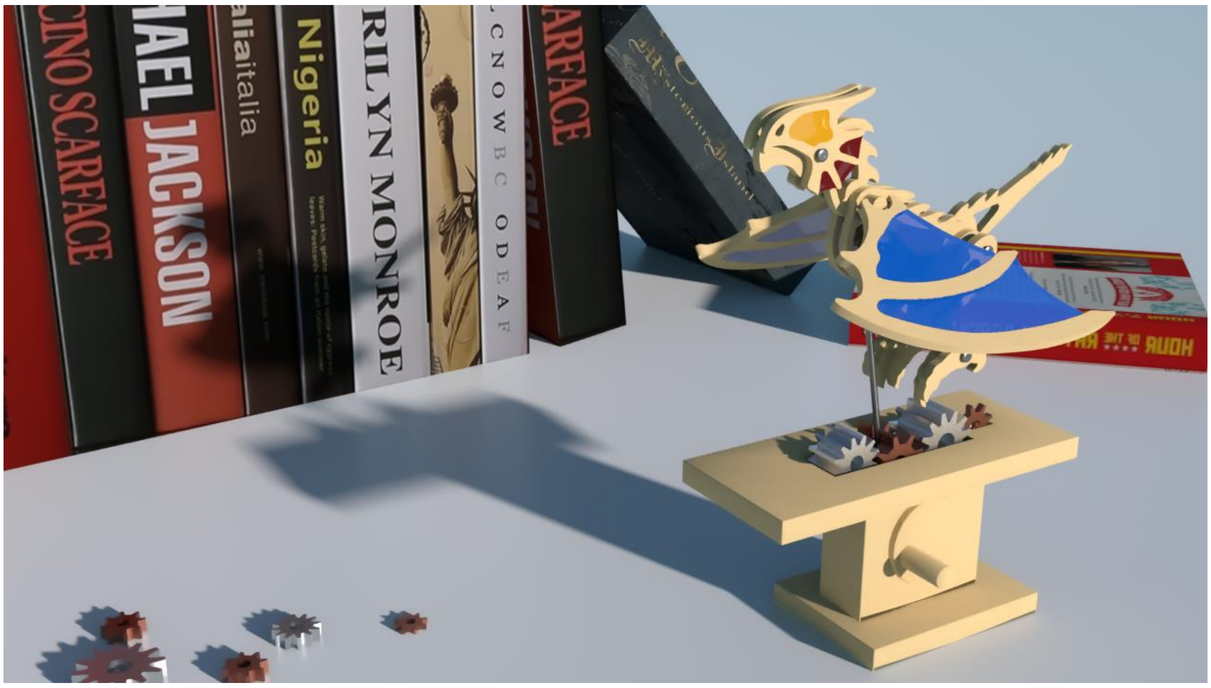
Procedural Automata

Breakdown

Author: Arrow Lyu
Tool: Houdini
Final File: Procedural_Automata_v08.hipnc

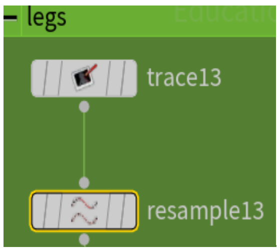


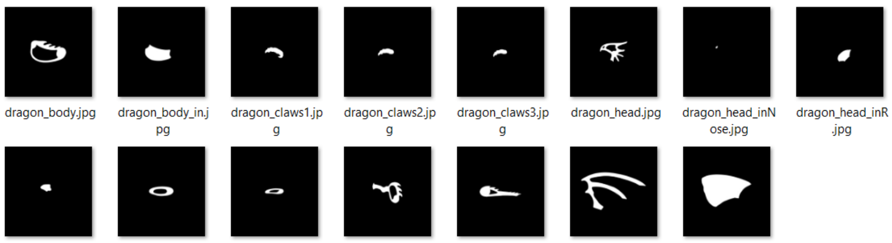
Project Summary

This project is a fully procedural automata rig built inside Houdini. Inspired by RadugaGrad's mechanical dragon project (https://www.youtube.com/watch?v=5sJ8YnSItcg time stamp: 8:37). I set out to recreate a functioning gear-based animation system from scratch using only procedural modeling and VEX expression-driven logic in Houdini.

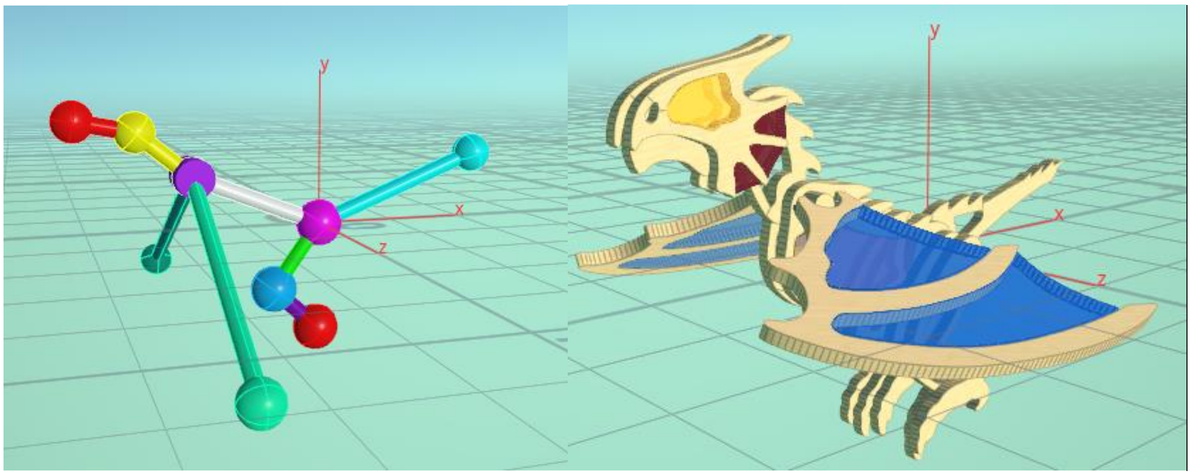
Technical Breakdown

- **Blueprint Tracing**
- Each mechanical part was designed by hand-drawing 2D silhouettes (white shape on black background) and importing them using the Trace node in Houdini.
- The shapes were resampled to reduce point density, allowing easier procedural control downstream.

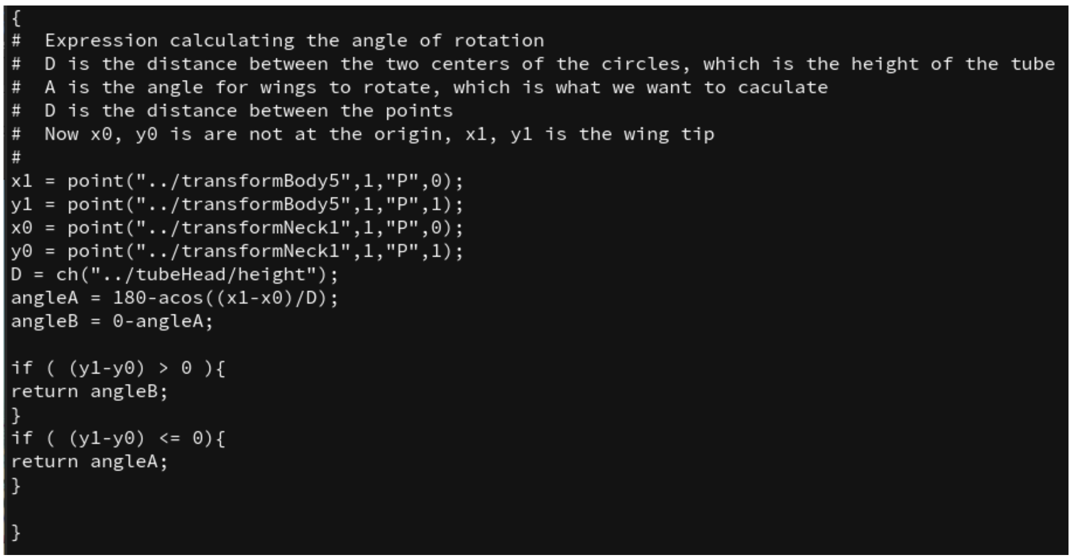




- **Structural Design**
- Built the dragon’s body using primitive geometry (tubes and spheres).
- All individual parts are referenced and positioned relative to joint points, which are dynamically computed and used as origin pivots for animation.
- I mainly use Sin($F) and Cos($F) to animate the parts. I use the function “point("../obj",n1,"P",n2)” to get the (x,y,z) location of a certain point.
- Then I reference each piece of the automata to the location of their joint. Before I move each piece to the joint’s location, I have to transform it to make sure the origin is where the joint will be at.



- **Attribute-Driven Animation**
- To animate procedurally, each part must be driven by the motion of the tube of the frame. The movement of each limb is driven by trigonometric expressions.
- The goal is to get the angle A, which represents how much the tube rotates compared to its joint. Here we can see the joint is the origin (x0, y0), the length D of the tube is already known, the tip point location of the rod is (x1, y1) which can be known by using function “point("../obj",n1,"P",n2)”. Then use the format angleA = arccos(cosA) = arccos((x1-x0)/D) to calculate angle A. The if loop here I just used to make sure the returned angle can bring its “-” when y1<y0, which will reflect the correct rotation of the tube.

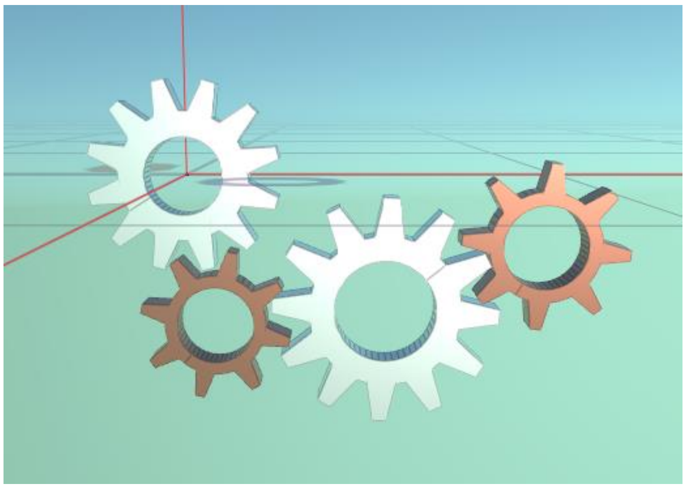


```
{
#  Expression calculating the angle of rotation
#  D is the distance between the two centers of the circles, which is the height of the tube
#  A is the angle for wings to rotate, which is what we want to caculate
#  D is the distance between the points
#  Now x0, y0 is are not at the origin, x1, y1 is the wing tip
#
x1 = point("../transformBody5",1,"P",0);
y1 = point("../transformBody5",1,"P",1);
x0 = point("../transformNeck1",1,"P",0);
y0 = point("../transformNeck1",1,"P",1);
D = ch("../tubeHead/height");
angleA = 180-acos((x1-x0)/D);
angleB = 0-angleA;

if ( (y1-y0) > 0 ){
return angleB;
}
if ( (y1-y0) <= 0){
return angleA;
}

}
```

- **Gear System Logic**
- To animate interlocked gears, rotation is passed along a chain using this formula:
- ch("../rotateGreenGear/rz") * (number of former gear teeth) / (number of current gear teeth)
- This enables precise synchronization between gears of different sizes and configurations.



Learning Outcome

This project demonstrates how code and procedural logic can drive mechanical systems in 3D, highlighting my understanding of mathematics, VEX programming, and animation logic inside Houdini.