Strandbeest | Breakdown

Rendering Statistics

renderer - mantra

average render time - 47 minutes

image resolution - 1280 x 720

number of lights - 2 (sun & skylight with HDR)

geometry complexity - 367 primitives

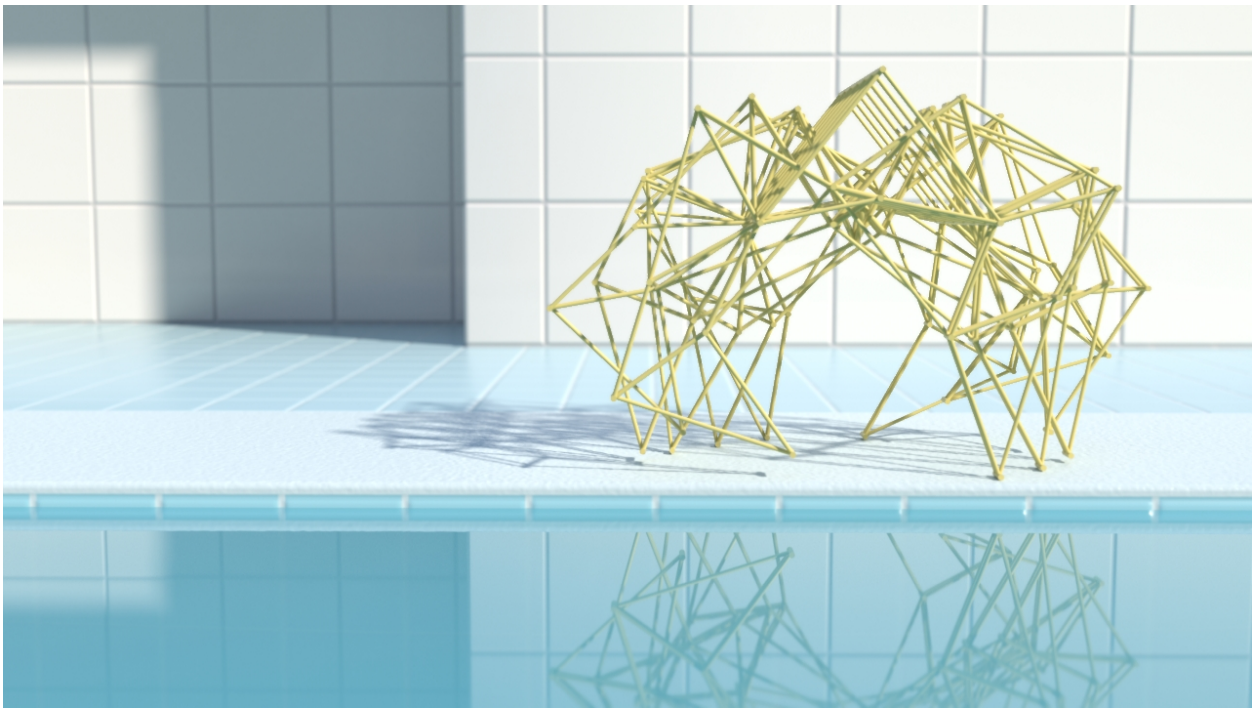
Sampling

noise value - 0.01

pixel samples - 5 x 5

min rays - 16

max rays - 32



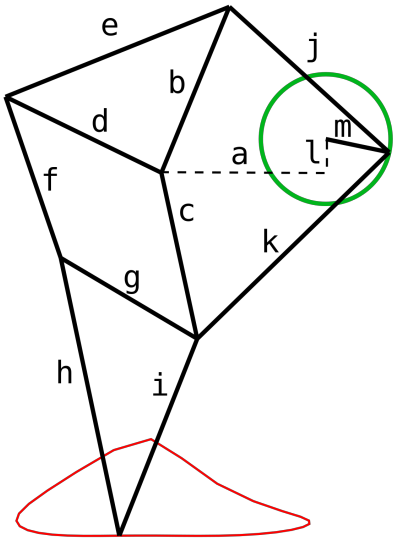
Project Description

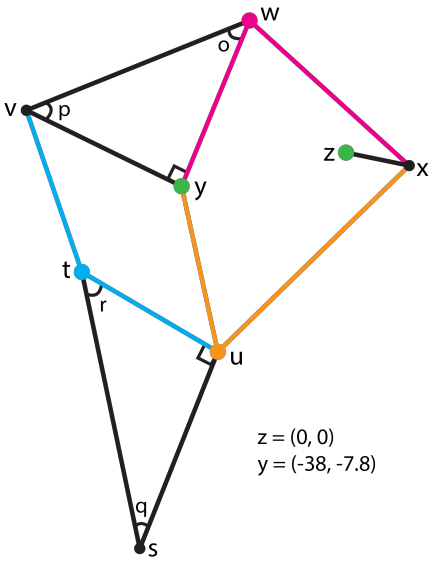
For my procedural animation I replicated Theo Jansen's *Strandbeest*, a series of complex kinetic sculptures based on triangles that mimic animals walking. Because his large sculptures are usually released on beaches, I had my smaller *Strandbeest* walk along the edge of an indoor pool.

The movement of the legs is driven by the rotation of a rod in the center, and the specific length and placement of the rods of the legs results in the animal-like walking.

Strandbeest Math

a=38.0
b=41.5
c=39.3
d=40.1
e=55.8
f=39.4
g=36.7
h=65.7
i=49.0
j=50.0
k=61.9
l= 7.8
m=15.0





The rods must each be a specific length (Jansen's "Holy Numbers"), seen in the diagram on the left. In addition, points Y and Z and angles O, P, Q, R, and the right angles are fixed.

The problem: calculating the movement of the rods based on the rotation of rod M.

The solution: three 2-point constraints, seen in the colored rods in the diagram on the right. The pink and orange rods must be calculated first in order to get the position of points U and V. The third 2-point constraint of the blue rods can then be calculated. Because the angles in triangles VWY and TUS are fixed, rods D and E follow the rotation of rod B, and rods I and H follow the rotation of rod G.

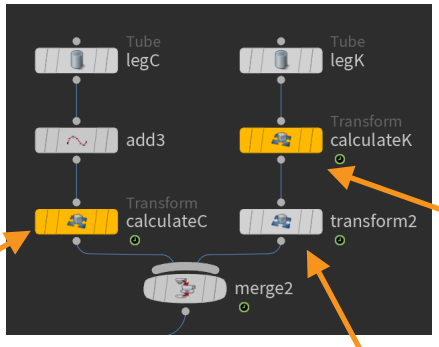
2-Point Constraint Example in Hscript (orange rods)

```
{
R = ch("../legK/height");
r = ch("../legC/height");
x1 = ch("./tx");
y1 = ch("./ty");
x0 = point("../shaftRotate",1,"P",0);
y0 = point("../shaftRotate",1,"P",1);
D = sqrt(pow(x1-x0,2) + pow(y1-y0,2));
d = (R*R - r*r + D*D)/(2.0*D);

angleG = acos((D-d)/r);
angleT = acos( ( -y1+y0 )/D);
angleS = 180-angleT;

angleRot = angleS - angleG;
return 270 + angleRot;
}
```

rotate z




```
{
R = ch("../legK/height");
r = ch("../legC/height");
x1 = ch("../calculateC/tx");
y1 = ch("../calculateC/ty");
x0 = point("../shaftRotate",1,"P",0);
y0 = point("../shaftRotate",1,"P",1);
D = sqrt(pow(x1-x0,2) + pow(y1-y0,2));
d = (R*R - r*r + D*D)/(2.0*D);

angleE = acos(d/R);
angleT = acos( ( -y1 + y0 )/D);

angleRot = angleT - angleE;
return  270 - angleRot;
}
```

rotate z

Translate	point("../shaftRotate",1,"P",0)	point("../shaftRotate",1,"P",1)	point("../shaftRotate",1,"P",2)
Rotate	0	0	-90

The rods start in an "L" shape at the origin, with leg K vertical and leg C horizontal. The transform node moves rod K to point X and rotates it to meet with rod C at point U.

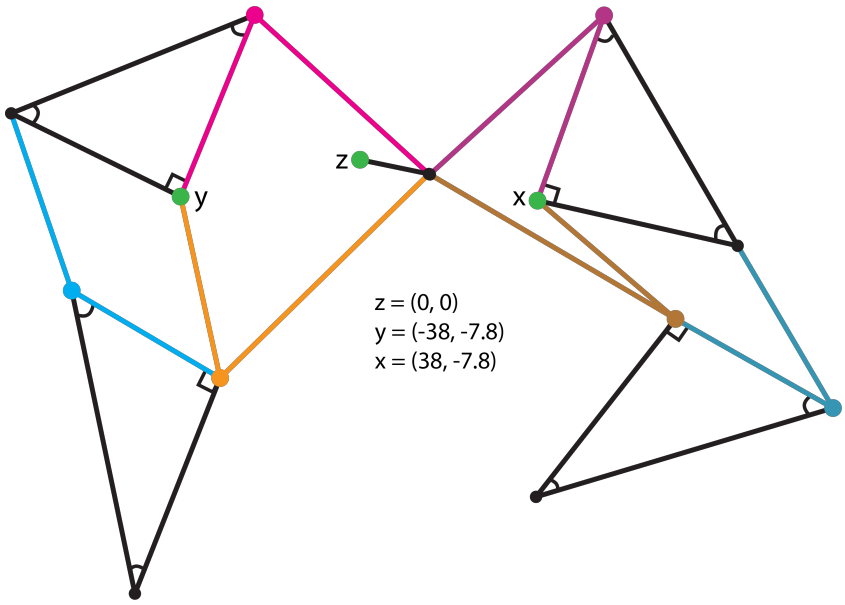
The 2-point constraint calculations use dot product and the pythagorean theorem. For more information, visit Professor Fowler's website: http://deborahfowler.com/MathForVSFX/DotProduct.html

Creating the Right Leg

The left leg **can't be mirrored** to create the right leg, so the calculations used for the left leg must be adjusted with the new positions of the right leg.

For example, the reference axis for the blue rods changes from (+) x to (-) x, and the position of point y changes.

The left and right leg can then be copied to create the multi-legged sculpture, using the copy number to vary the rotation of leg M.



Additional Problem: Floating Feet

Problem: the feet would touch the ground and then float until being lifted again, rather than sticking to the ground.

Solution: clamp the translate y channel of the feet geo.

```
{
data = point("../transform4",6,"P",1);
value = clamp(data, -9.02, 100);
return value;
}
```

Custom Parameters

Custom parameters were created on the top level to easily control the speed, leg variation, and number of legs, as well as shift the position.