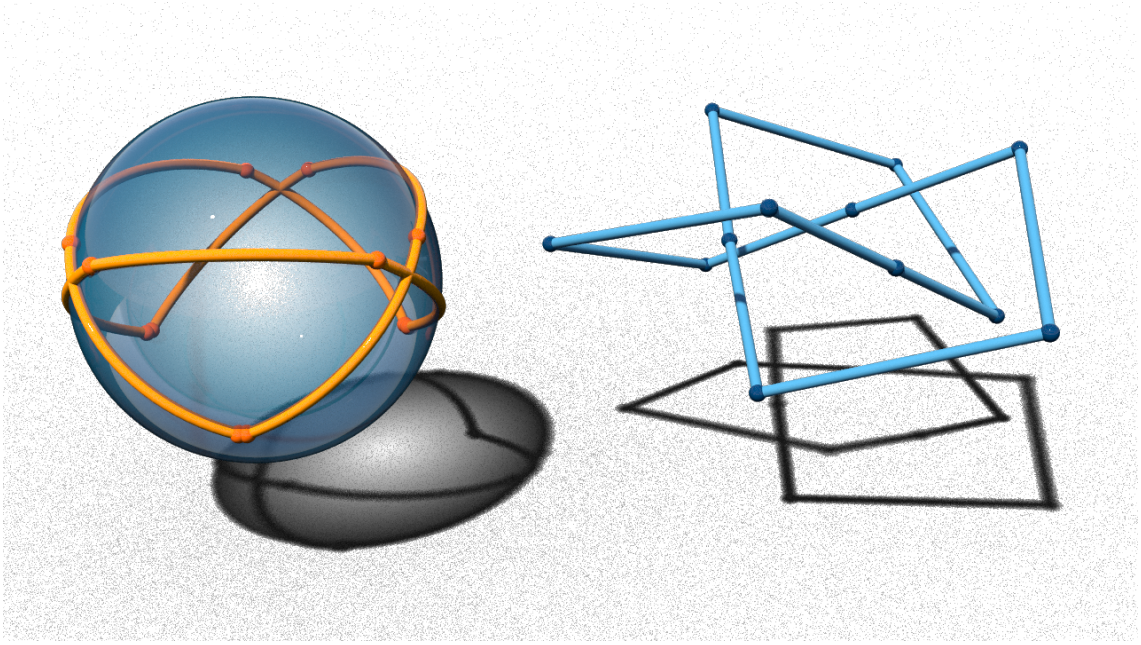
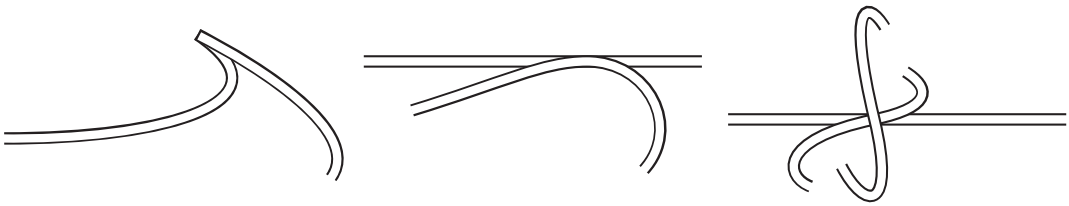
Technische Universität Berlin
Institut für Mathematik
Sullivan / Knöppel
http://www3.math.tu-berlin.de/geometrie/Lehre/SS17/MathVis SS 17

# Mathematical Visualization

## Assignment 4 - Tantrix[1]



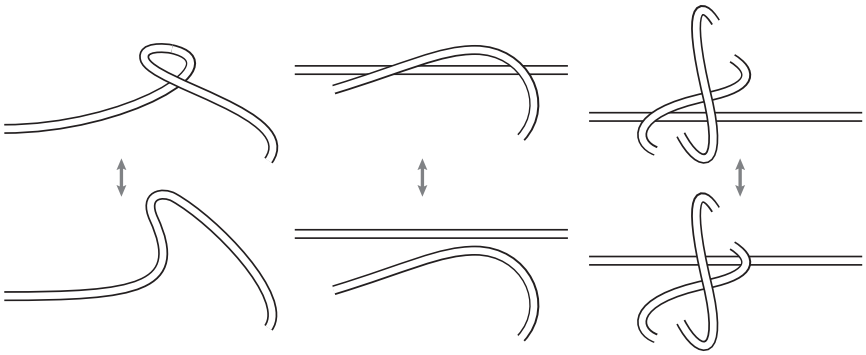
**Reidemeister moves.** The orthognonal projection of a knot $\gamma\colon \mathbb{S}^1 \to \mathbb{R}^3$ to a plane is called a diagram. A diagram is *regular* if the projection has finitely many transversal crossings, i.e. we want to avoid the following 3 situations:



1) Cusps - projection along a tangent direction of $\gamma$.
2) Non-transversality - projection along $\gamma(t_2) - \gamma(t_1)$ with $\gamma(t_2) - \gamma(t_1), \gamma'(t_1), \gamma'(t_2)$ coplanar.
3) Triple crossings - projection along a trisecant of $\gamma$.

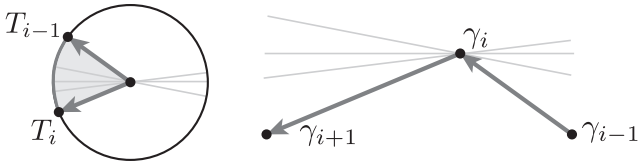
Each of the 3 situations above correspond to a so called Reidemeister move - a slight perturbation of the projection direction performs a Reidemeister move in the diagram:

[1] due 28.5.2017



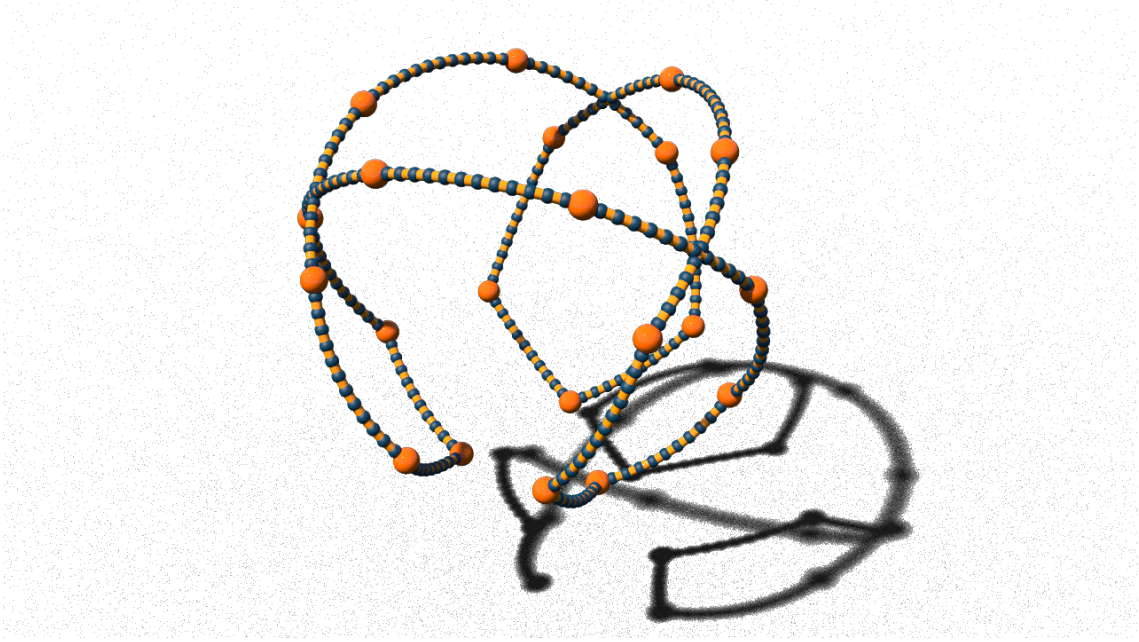
**The tantrix.** In the smooth case the *tantrix* $T$ of a knot $\gamma$ is define as $T = \gamma'/|\gamma'|$. Actually $T$ parametrizes the directions for which we see cusps in the corresponding knot diagram. Similarly, the discrete tantrix consists of the normalized edge vectors, i.e. $T_i = e_i/|e_i| \in \mathbb{S}^2$. If we connect the points of $T$ by spherical arcs instead of straight line we obtain a spherical polygon.

A *tangent direction* of a discrete curve $\gamma$ is the direction of a line through a point of $\gamma$ which lies in the osculating plane and does not separate the incident edges.

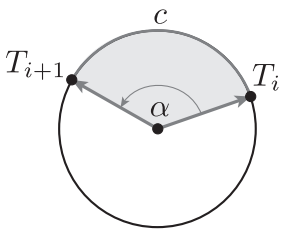



Actually each point on the (shorter) great circle arc from $T_{i-1}$ to $T_i$ is a tangent direction. So it makes perfect sense to consider $T$ as a spherical polygon rather than a polygon in Euclidean 3-space. The set of all tangent directions can then be parametrized by $T$ and $-T$.

To draw spherical polygons in Houdini we approximate each perfect spherical arc by a polygonal curve close to it. Therefore we subdivide the spherical polygon.



Consider the edge $c_i$ (shortest great circle arc) between $T_i$ to $T_{i+1}$ and let $d \in \mathbb{N}$. The length of $c$ equals the angle $\alpha_i$ between the vectors $T_i$ and $T_{i+1}$: $\alpha_i = \arccos\langle T_i, T_{i+1}\rangle$. We want to split $c$ into $d$ arcs of equal length.



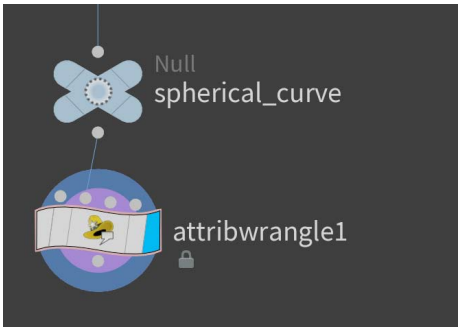

If $\alpha = 0$ there is nothing to do. Let us assume that $\alpha \neq 0$ and define $q \in \mathbb{S}^3$ as follows:

$$q = \cos(\tfrac{\alpha}{2d}) + \sin(\tfrac{\alpha}{2d})B_i, \quad where \quad B_i = \tfrac{T_i \times T_{i+1}}{|T_i \times T_{i+1}|}.$$

Then the points $\widetilde{T}_i^k := q^k T_i \bar{q}^k$, $k = 0, \ldots, d$, lie on the edge $c$, $\widetilde{T}_i^0 = T_i$, $\widetilde{T}_i^d = T_{i+1}$ and the spherical distance between $\widetilde{T}_i^k$ and $\widetilde{T}_i^{k+1}$ is $\frac{\alpha}{d}$. We have constructed an equidistant subdivision of the edge $c$.

Now, let $\alpha_0 > 0$. Then with $d = \lfloor \frac{\alpha}{\alpha_0} \rfloor$ we obtain a subdivision with edge length $\leq \alpha_0$. If we do this for each arc of $T$ we obtain a subdivision with edge length at most $\alpha_0$.

In Houdini we can use an *attribute wrangle node* to create such a subdivision. This will be the first time that we *create geometry from scratch* - we don't copy the geometry and modify it, but instead we wire the node which contains the spherical discrete curve to the second input of the attribute wrangle node (with 'Run over' set to 'Detail (only once)') and create a polygon with vertices glued to the points of the subdivision.



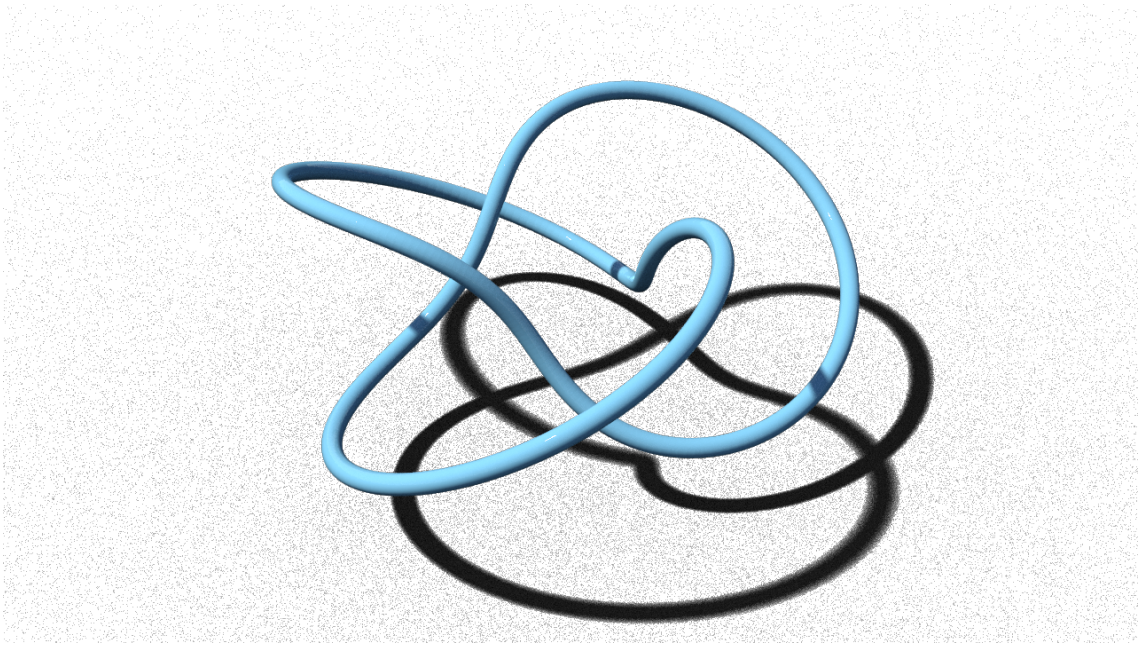

```
float maxedgelength = chf('maxedgelength');
int n = npoints(1);

// create a polygonal primitive, returns primitive number
int polygon = addprim(0,'poly');

//run over the points of the spherical curve
for(int i=0; i<n; i++){
    vector currT = attrib(1,'point','P',i);
    vector nextT = attrib(1,'point','P',(i+1)%n);
    float edgelength = acos(dot(currT,nextT));
    vector B = normalize(cross(currT,nextT));
    int d = floor(edgelength/maxedgelength);
    if(d!=0) edgelength/=d;
    vector4 q = quaternion(edgelength, B);
```

```
    // add subdivision points
    vector T_tilde = currT;
    do{
        int point = addpoint(0,T_tilde);
        addvertex(0,polygon,point);
        T_tilde = qrotate(q,T_tilde);
        d--;
    }while(d>0);
}
```

**Torus knots.** A nice class of knots are the so called *torus knots*, i.e. closed curves that lie on a torus of revolution. The formula is fairly easy: Let $R > r > 0$ and $p, q \in \mathbb{Z}$ be relatively prime. One form of a $(p, q)$-torus knot is given by $\gamma\colon \mathbb{S}^1 \to \mathbb{R}^3$ with

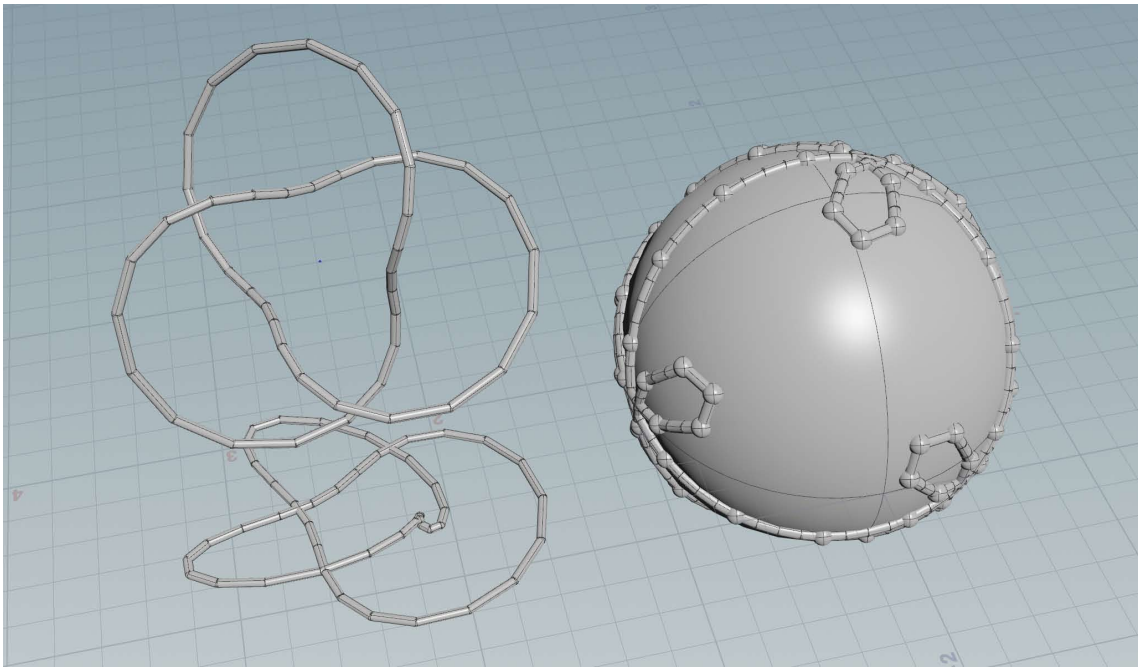
$$\gamma(\alpha) = \begin{pmatrix} \big(R + r\cos(q\alpha)\big)\cos(p\alpha) \\ \big(R + r\cos(q\alpha)\big)\sin(p\alpha) \\ r\sin(q\alpha) \end{pmatrix}$$

The numbers $p$ and $q$ determine how often the knot winds around the one and the other direction before it closes up. Below a picture of a $(2, 3)$-torus knot - a so called *trefoil knot*.



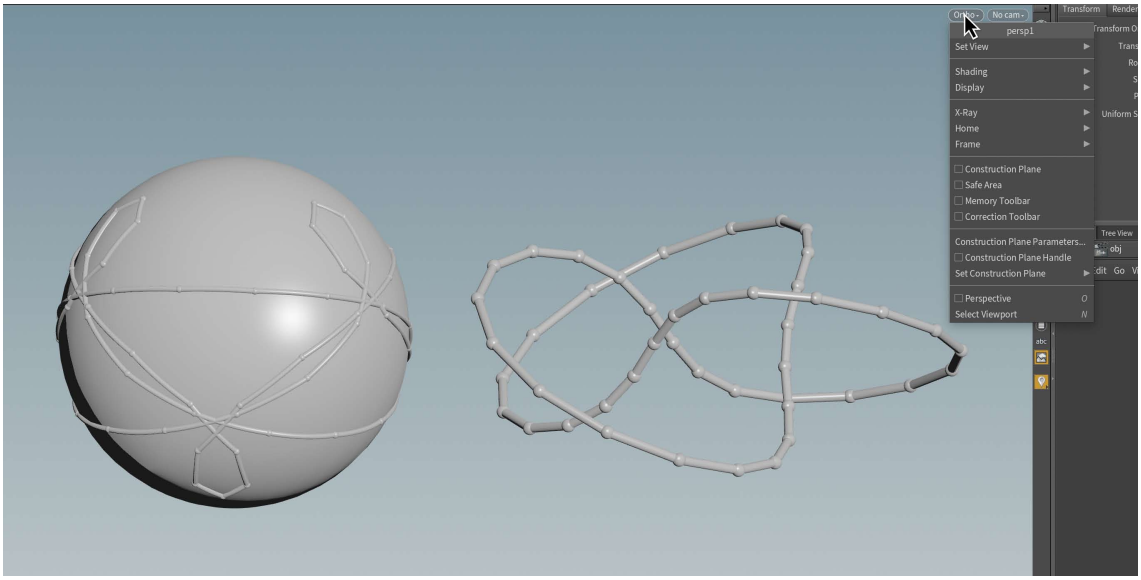
Discrete $(p, q)$-torus knots can be obtained then by by sampling. E.g. the very first picture above shows a roughly sampled trefoil knot and its tantrix. This is easy to implement in Houdini - the network just needs a *circle* and a *point wrangle node*.

**Visualizing the diagram.** To see the relation between $\pm T$ and Reidemeister-1-moves we can just project along an axis, turn the knot and watch the projection. In the picture below we project along the vertical axis.



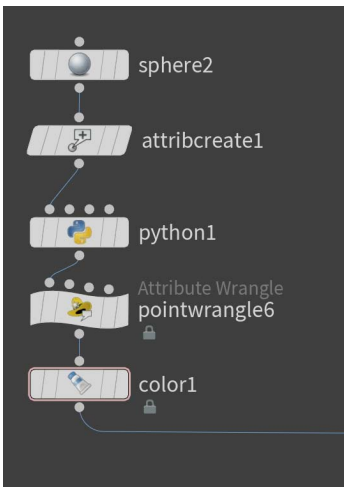
If we turn the knot in such a way that $\pm T$ sweeps over the north pole we see a Reidemeister-1-move in the projection. When $\pm T$ hits the north pole we see a cusp.

A probably more intuitive way to obtain the diagram is to change the camera projection from perspective to orthogonal. Therefore one can just uncheck the perspective checkbox which you find in the first drop down menu in the upper right corner of the scene view.




Then we can put the knot and the sphere side by side and rotate the whole scene. The projection direction corresponds the center of the projected sphere.

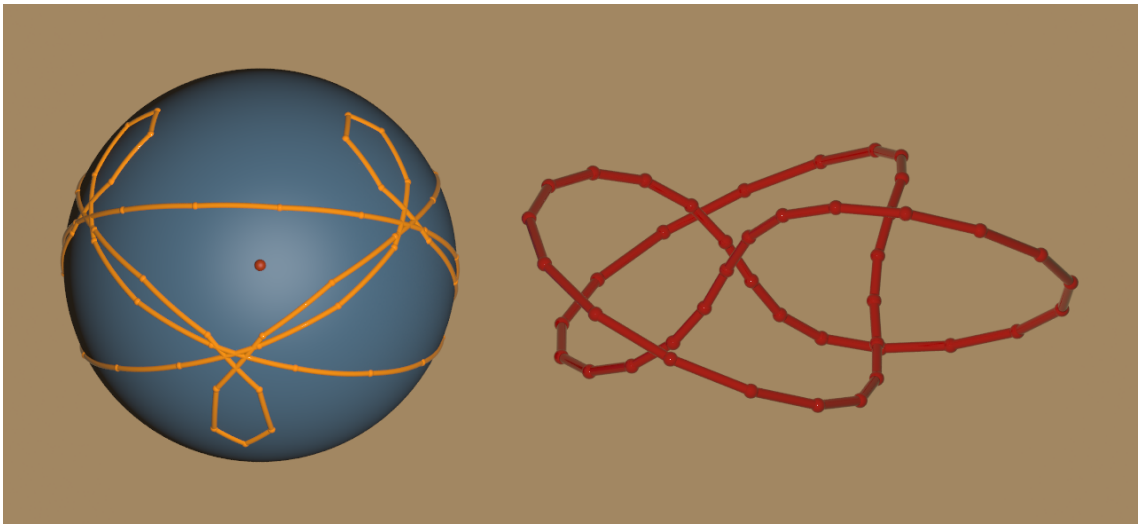
Though it is not so easy to see whether $\pm T$ contains this direction or not. Therefore we want to mark the projection direction by a small sphere. Therefore we create a small sphere primitve, define a global attribute `@proj`, read off the camera direction using a *python node* and move the sphere to the right position by a *point wrangle node*. The network then looks as shown in the picture below.




Here the corresponding python code:

```
# get the pane of the Scene View pane tab
viewer = hou.ui.findPaneTab('panetab1')

# view transform as hou.matrix4
t = viewer.curViewport().viewTransform()
v = hou.Vector4(0,0,1,0)
proj = v*t.inverted().transposed()
geo = hou.pwd().geometry()
proj = hou.Vector3(proj.x(),proj.y(),proj.z())
geo.setGlobalAttribValue('proj',proj)

# make sop time dependent
frame = hou.frame()
```

The reference to the SceneViewer pane tab can be found by pulling the Scene View tab into a python shell. The last line makes the node time dependent: If one hits the play button the node is cooked every frame. So the point position is updated while one rotates the scene.



**Exercise 4:** Wire up a network to visualize the relation of the tantrix and the knot diagram. Therefore generate a torus knot, paint its tantrix $T$ and $-T$ on the unit sphere, change the camera projection to be orthogonal and mark the camera direction on the sphere.