A Simple Parametric Model of Plain-Knit Yarns

Keenan Crane

April 12, 2023

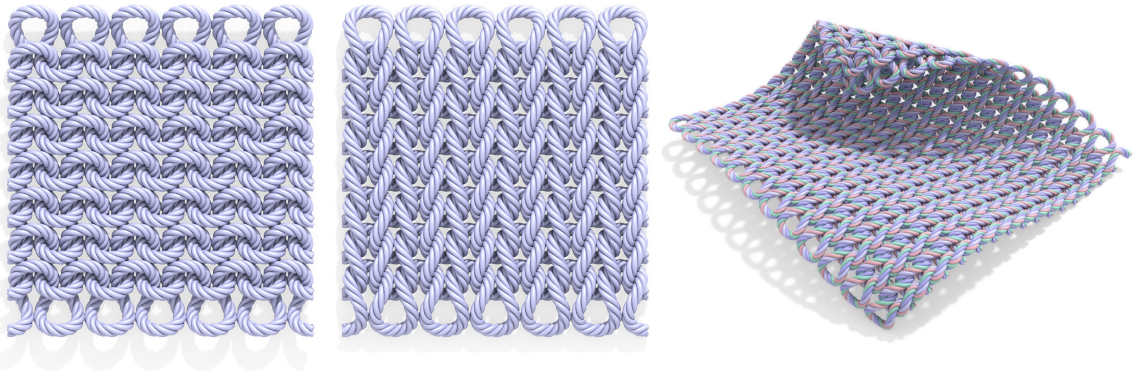


Figure 1: *Plain knit yarns viewed from the back* (left)*, front* (center)*, and mapped onto a developable surface* (right).

Abstract: *This short note gives closed-form expressions for smooth curves resembling a plain-knit stitch, suitable for appearance modeling and previsualization. A unique feature of this model is that it also describes the twisted yarn fibers running along this stitch, making it suitable for modeling marled yarn. Adjustable parameters control the knit spacing, loop roundness, yarn thickness, and rate of fiber twisting. Tileable displacement maps and code for generating curves are also provided.*

1 Overview

A *plain stitch*, also known as a *stockinette stitch*, *jersey stitch*, *flat stitch*, or *felt stitch*, is a basic knitting pattern formed by rows of identical yarn loops. Though a beautiful array of decorative knit stitches appear in the design of textiles, the plain stitch is the baseline pattern used for common garments, upholstery, *etc.* Hence, visual simulation of everyday objects can benefit from a cheap, simple, and controllable model of the kind presented here. Parameterization of twisted fibers makes our model especially suitable for *marled* yarns, where several fibers or *rovings* of different color are twisted together to form a single yarn.

The geometry of the plain stitch pattern arises from an interplay between elastic forces (bending and twisting of a rod) and collision forces (frictional contact between neighboring yarns), and has in the past been modeled via numerical optimization [Yuksel et al., 2012], physical simulation [Kaldor et al., 2010], and via a helicoid model [Wadekar et al., 2020]. Parametric models have long been used in the textile industry to understand and predict physical properties of knitted garments (dimensions, weight, material usage, *etc.*). For instance, Chamberlain [1926] gives a 2D description that is useful for reasoning about material properties, but not sufficient for capturing 3D appearance. To our knowledge, the first parameterization of a three-dimensional plain knit stitch was given by Peirce [1947, Part IV], as a 2D piecewise function with straight and circular parts, composed with a mapping to a 3D cylinder. The lack of curvature continuity, and the piecewise nature of this function, makes it difficult to develop a parameterization of the secondary twisted fibers. Moreover, Peirce's curve does not provide many parameters to adjust the shape or appearance of the knit pattern (only the yarn diameter and wale/course spacing). On the other hand, geometrically simple components make the curve and its parameters easy to analyze, whereas quantities like arc length and maximal thickness must be determined numerically for our model (Section 4). Leaf and Glaskin [1955] and Munden [1959] refine Peirce's model and connect it to empirical data; Munden in particular observes that yarn shape is largely geometric and variational in nature, *i.e.*, arising from energy minimization that does

not depend strongly on yarn properties or stitch length. Twisted fibers can be seen in many papers from the computer graphics literature (*e.g.*, Kaldor et al. [2010]), but no explicit equations are given. In general, we were unable to find any parameterized version of the twisted fiber curves in the literature, though given the cultural importance of knitting it is very likely that such a parameterization has been written down somewhere.

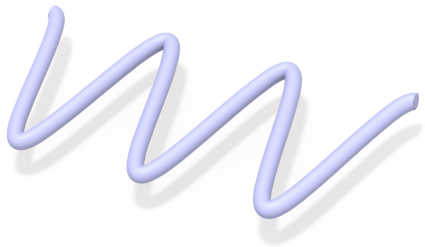
In this note, the geometry of each row (corresponding to the course direction) is given as a parameterization of the centerline, *i.e.*, the curve passing through the center of the yarn; the geometry of the yarn itself can be approximated by a tubular neighborhood of some radius $R > 0$ around this centerline. Multiple copies of this curve, offset along the wale direction, can be used to model a piece of knitted material. Secondary curves describe the centerlines of individual fibers; here, a simple correction to the Frenet frame yields a good approximation to the twist-free (Bishop) frame—providing a near-uniform rate of twist around the yarn centerline (Section 3). Though our parameterization has no direct physical origin (*e.g.*, it does not minimize a potential energy), it nonetheless captures the appearance of real knit yarn reasonably well at intermediate scales (see [Zhao et al., 2016] for models of yarn microstructure, and [Xu et al., 2001; Jakob et al., 2010] for a discussion of photorealistic rendering). Moreover, a closed-form expression is efficient to evaluate, has well-defined derivatives of all orders, has easily-adjustable parameters, and lends itself well to constructing secondary curves describing twisted yarn fibers (which are difficult or expensive to capture via optimization and simulation). This pattern can then be mapped onto a surface to obtain an appearance similar to a knit garment—Figure 1, *right* shows an example achieved by composing the yarn curve with a parameterized developable surface, using the z-coordinate of the curve to determine an offset in the normal direction. If more physical results are desired, these initial curves can be further optimized or simulated to better match the behavior of physical yarn [Kaldor et al., 2010; Yuksel et al., 2012]. Appendex A provides displacement maps that can be used for finer patterns (or on more complex geometry).

2 Yarn

To develop a parameterized yarn, we start with a simple sinusoid

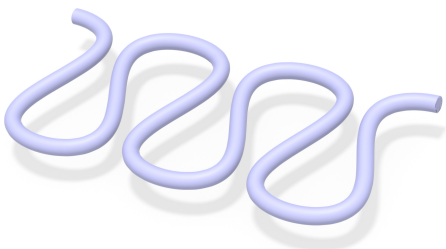
$$t \mapsto (t, h\cos(t), 0),$$

parameterized over values $t \in \mathbb{R}$, where $h > 0$ controls the height of the loops:

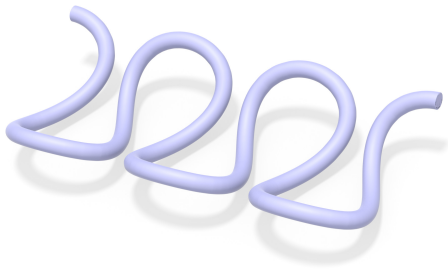


To get rounder loops, we can modify the x-coordinate so that it "overshoots" at the end of each loop, and "backtracks" slightly before making the next loop. For instance, we can add some multiple a of $\sin(2t)$ to get a curve

$$t \mapsto (t + a\sin(2t), h\cos(t), 0).$$



Finally, in order for each row to loop around the next, the curve must bend out of plane. We can model this bending by a sinusoid $\cos(2t)$ in the z-direction of twice the frequency as in the wale (y) direction, with some magnitude $d > 0$:



The final yarn centerline is then parameterized by a curve $\gamma : \mathbb{R} \to \mathbb{R}^3$, given by

$$\boxed{\gamma(t) := (t + a\sin(2t), h\cos(t), d\cos(2t)).} \tag{1}$$

The parameter a controls the roundness of the loops, the parameter $h > 0$ controls the height of the yarn in the wale direction, and the parameter $d > 0$ controls the depth of oscillations in and out of the plane. Figure 2 shows an example.

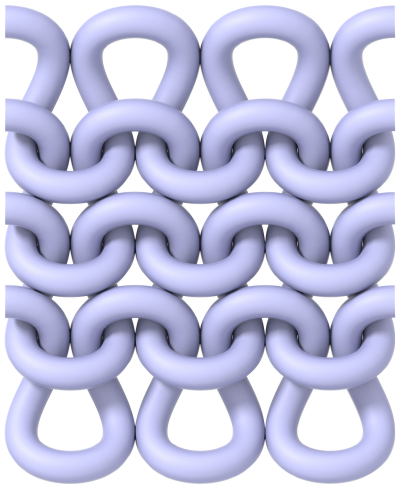


Figure 2: *Several copies of the yarn curve $\gamma(t)$ for $a = 3/2$, $h = 4$ and $d = 1$. For a tube radius $R = 4/5$, offsetting rows by $h + 1/2$ in the wale direction (*i.e.*, y-direction) generally yields a snug fit between curves.*

3 Yarn Fibers

3.1 Frenet Frame

To describe a curve winding around the yarn, we first construct the Frenet frame for $\gamma(t)$, given by three orthonormal vectors

$$e_1(t) := \frac{\frac{d}{dt}\gamma(t)}{\left|\frac{d}{dt}\gamma(t)\right|}, \qquad e_2(t) = \frac{\frac{d}{dt}e_1(t)}{\left|\frac{d}{dt}e_1(t)\right|}, \qquad e_3(t) := e_1(t) \times e_2(t).$$

these directions describe the tangent, normal, and binormal of $\gamma(t)$, respectively. (note that to define e_1 we must normal $d\gamma/dt$, since γ is not an arc-length parameterized curve.) The unnormalized directions $\tilde{e}_1, \tilde{e}_2$ are given by

$$\begin{aligned}
\tilde{e}_1(t) &= (1 + 2a\cos(2t), -h\sin(t), -2d\sin(2t)), \\
\tilde{e}_2(t) &= -\tfrac{v(t)}{2u(t)^{3/2}}((1 + 2a\cos(2t)), -h\sin(t), -2d\sin(2t)) - \tfrac{1}{u(t)^{1/2}}(4a\sin(2t), h\cos(t), 4d\cos(2t)),
\end{aligned}$$

where

$$\begin{aligned} u(t) &:= (1+2a\cos(2t))^2 + h^2\sin(t)^2 + 4d^2\sin(2t)^2 \\ v(t) &:= 2h^2\cos(t)\sin(t) + 16d^2\cos(2t)\sin(2t) - 8a(1+2a\cos(2t))\sin(2t). \end{aligned}$$

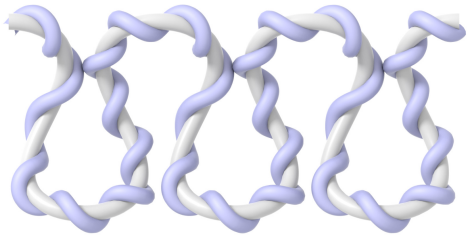
applying the normalizations $e_1 = \tilde{e}_1/|\tilde{e}_1|$ and $e_2(t) = \tilde{e}_2/|\tilde{e}_2|$ and taking the cross product $e_3 = e_1 \times e_2$ then gives an orthonormal frame.

3.2 Fiber Curves

Using the Frenet frame, a curve twisting around $\gamma(t)$ can be expressed as

$$t \mapsto \gamma(t) + r(\cos(\theta(t))e_2(t) + \sin(\theta(t))e_3(t)),$$

where r is the yarn thickness, and $\theta(t)$ is some function that gives the angular location of the fiber relative to the local frame of $\gamma(t)$ at time t. Suppose we let $\theta(t) = \omega t$, where $\omega > 0$ controls the rate of twisting. Then we almost get the behavior we want, except that there are a different number of twists on either side of the loop:



Even though the angular velocity ω is constant, the apparent non-uniformity arises from the fact that the Frenet frame does not turn at a constant speed along the curve (Figure 3, *left*). One way to rectify this situation would be to use the *Bishop frame* (*i.e.*, frame of least twist) rather than the Frenet frame to define the yarn fibers. However, the Bishop frame requires global integration along the curve (and these integrals generally are not available in closed form), whereas the Frenet frame can be evaluated locally at each point. Hence, we instead approximately correct for the twisting of the Frenet frame by simply adjusting the angle of rotation by a term $-2\cos(t)$, yielding an overall angle function $\theta(t) = \omega t - 2\cos(t)$ (Figure 3, *right*).

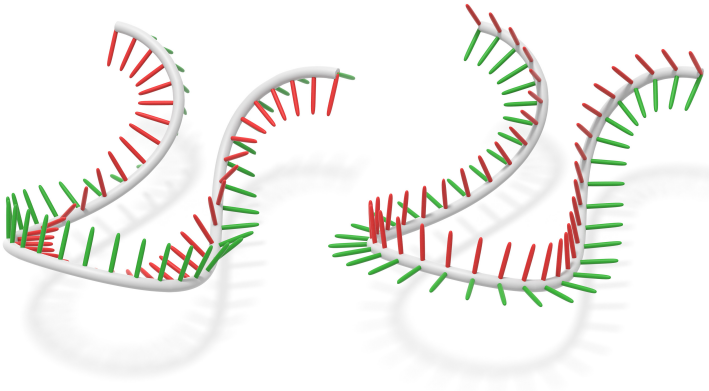
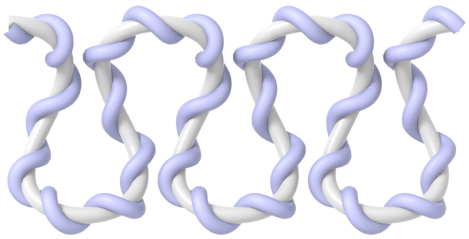


Figure 3: *If we use the Frenet frame* (left) *to twist fibers around the yarn, the twists will be uneven, since the frame itself exhibits significant nonuniform twist. Adjusting the Frenet frame* (right) *gives us a more parallel frame, which in turn enables us to define uniformly-twisted fibers.*

The resulting fiber curve now twists uniformly along the entire loop:



Finally, to fill in the rest of the yarn, we can draw several copies of the fiber curve where the phase has been shifted by uniform increments ϕ. Our final fiber curve is then given by

$$\boxed{\eta_\phi(t) := \gamma(t) + r(\cos(\theta_\phi(t))e_2(t) + \sin(\theta_\phi(t))e_3(t)),} \tag{2}$$

where

$$\theta_\phi(t) := \omega t - 2\cos(t) + \phi.$$

Figure 4 shows an example.

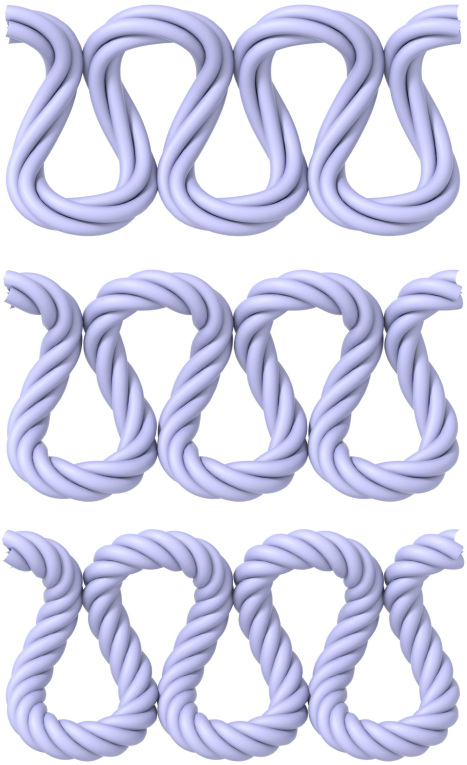


Figure 4: *Fiber curves for twist frequencies* $\omega = 2$ (top), $\omega = 4$ (middle), *and* $\omega = 6$ (bottom). *Here* $r = 1/2$ *and we plot four fiber curves with phases* $\phi = k\pi/2$, $k = 0, 1, 2, 3$. *The tube radius used for visualization is* $R = 7/20$.

4 Arc Length and Thickness

Curve length is useful for anticipating how much yarn will be needed to knit a garment. Note, however, that since our model is not based on physical equations—and because material properties can vary significantly among yarn types and manufacturing processes—the lengths described here will provide only a rough proxy for the actual quantity of yarn used to knit a real garment.

The arc length ds_γ of the yarn curve $\gamma(t)$ is given by

$$ds_\gamma = \sqrt{(1 + 2a\cos(2t))^2 + h^2\sin(t)^2 + 4d^2\sin(2t)^2}\,dt. \tag{3}$$

As with most curves, $\gamma(t)$ does not admit a closed-form arc-length reparameterization—nor its its arc length even directly integrable. In contrast to Peirce [1947] and Leaf and Glaskin [1955], who give approximations of curve length for their parameterized models, we advocate the use of numerical integration to obtain an accurate estimate of length. For instance, one could integrate Equation 3 over the interval $t \in [0, 2\pi]$ using any standard numerical quadrature scheme (e.g., midpoint quadrature) to get the length of a single loop. Alternatively, the length of a polyline approximation of $\gamma(t)$ gives a perfectly good (secant) approximation, especially for a large number of sample points.

The tube thickness R used to give thickness to centerline curves will depend on the curve parameters a, h, d, r, ω, as well as any physical or aesthetic considerations. One possible choice is to use the smallest R such that sweeping a disk of radius R along the curve (and contained in the plane orthogonal to the curve tangent) will not yield any self-intersections [Gonzalez and Maddocks, 1999, Section 3]. Like curve length, there is no simple expression for this radius as a function of the curve parameters—if an exact value is needed, one can apply the *octrope* algorithm of Ashton and Cantarella [2005] to the polygonal approximation provided by the code in Appendix B.

Acknowledgements. Thanks to Mark Gillespie and Jim McCann for insightful discussions and useful pointers to early literature on yarn geometry, and to Rob Pieké for pointing out errors in the original draft.

References

Ashton, T. and Cantarella, J. (2005). A fast octree-based algorithm for computing ropelength. In *Physical And Numerical Models In Knot Theory: Including Applications to the Life Sciences*, pages 323–341. World Scientific.

Chamberlain, J. (1926). Hosiery yarns and fabrics.

Gonzalez, O. and Maddocks, J. H. (1999). Global curvature, thickness, and the ideal shapes of knots. *Proceedings of the National Academy of Sciences*, 96(9):4769–4773.

Jakob, W., Arbree, A., Moon, J. T., Bala, K., and Marschner, S. (2010). A radiative transfer framework for rendering materials with anisotropic structure. In *ACM SIGGRAPH 2010 papers*, pages 1–13.

Kaldor, J. M., James, D. L., and Marschner, S. (2010). Efficient yarn-based cloth with adaptive contact linearization. In *ACM SIGGRAPH 2010 papers*, pages 1–10.

Leaf, G. and Glaskin, A. (1955). The geometry of a plain knitted loop. *Journal of the Textile Institute Transactions*, 46(9):T587–T605.

Munden, D. (1959). The geometry and dimensional properties of plain-knit fabrics. *Journal of the Textile Institute Transactions*, 50(7):T448–T471.

Peirce, F. T. (1947). Geometrical principles applicable to the design of functional fabrics. *Textile Research Journal*, 17(3):123–147.

Wadekar, P., Goel, P., Amanatides, C., Dion, G., Kamien, R. D., and Breen, D. E. (2020). Geometric modeling of knitted fabrics using helicoid scaffolds. *Journal of Engineered Fibers and Fabrics*, 15:1558925020913871.

Xu, Y.-Q., Chen, Y., Lin, S., Zhong, H., Wu, E., Guo, B., and Shum, H.-Y. (2001). Photorealistic rendering of knitwear using the lumislice. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 391–398.

Yuksel, C., Kaldor, J. M., James, D. L., and Marschner, S. (2012). Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Transactions on Graphics (TOG)*, 31(4):1–12.

Zhao, S., Luan, F., and Bala, K. (2016). Fitting procedural yarn models for realistic cloth rendering. *ACM Transactions on Graphics (TOG)*, 35(4):1–11.

A Displacement Maps

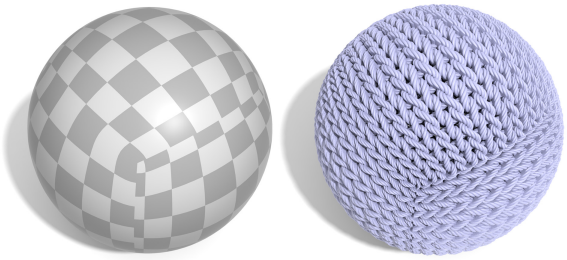
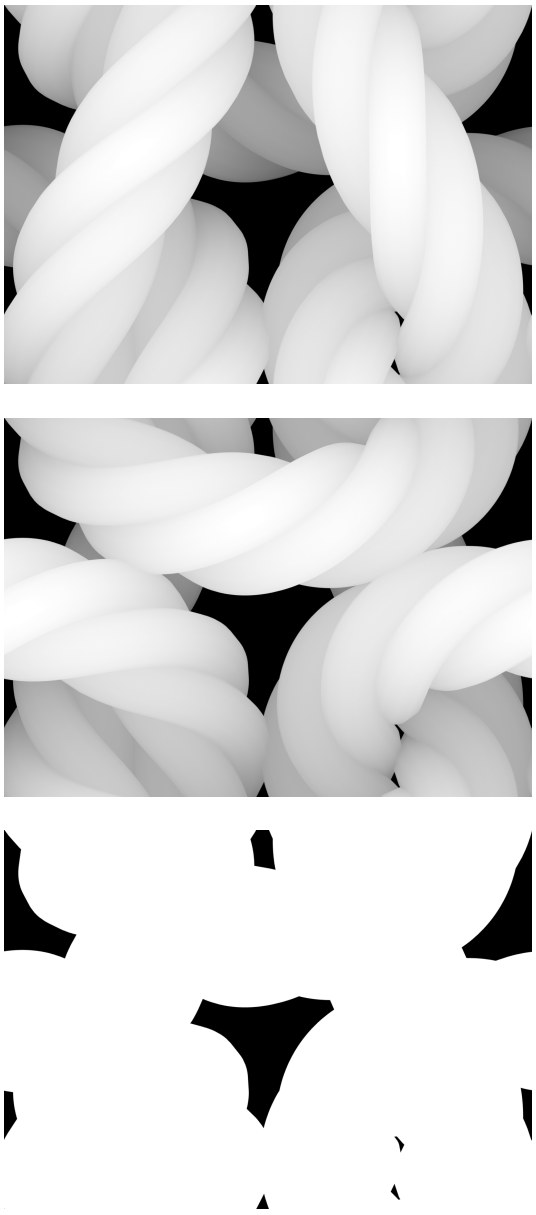


Figure 5: *Parameterized surface mesh* (left) *rendered with displacement maps based on the yarn fiber geometry* (right).

For rendering knit materials, it can be more efficient (or simply convenient) to use a displacement map rather than explicit geometry. The images below provide tileable patterns for front and back displacements, as well as the associated alpha matte. An example is shown in Figure 5.



B Code

```
#include <stdio.h>
#include <math.h>

struct Vector3
{
   double x;
   double y;
   double z;
};

double sqr( double x )
{
   return x*x;
}

double norm2( struct Vector3 v )
{
   return sqr(v.x) + sqr(v.y) + sqr(v.z);
}

double norm( struct Vector3 v )
{
   return sqrt( norm2( v ));
}

struct Vector3 scale( double c, struct Vector3 v )
{
   struct Vector3 cv;
   cv.x = c * v.x;
   cv.y = c * v.y;
   cv.z = c * v.z;
   return cv;
}

struct Vector3 cross( struct Vector3 u, struct Vector3 v )
{
   struct Vector3 w;
   w.x = u.y*v.z - u.z*v.y;
   w.y = u.z*v.x - u.x*v.z;
   w.z = u.x*v.y - u.y*v.x;
   return w;
}

struct Vector3 yarnCurve( double t, double a, double h, double d )
{
   struct Vector3 gamma_t;

   gamma_t.x = t + a*sin(2.*t);
   gamma_t.y = h * cos(t);
   gamma_t.z = d * cos(2.*t);

   return gamma_t;
}
```

```
void frenetFrame( double t, double a, double h, double d,
                  struct Vector3* e1, struct Vector3* e2, struct Vector3* e3 )
{
   double u_t, v_t, x_t, y_t;

   e1->x = 1. + 2.*a*cos(2.*t);
   e1->y = -h*sin(t);
   e1->z = -2.*d*sin(2.*t);

   u_t = norm2(*e1);
   v_t = 2.*h*h*cos(t)*sin(t) + 16.*d*d*cos(2.*t)*sin(2.*t) -
         8.*a*(1. + 2.*a*cos(2.*t))*sin(2.*t);
   x_t = 1./sqrt(u_t);
   y_t = v_t/(2.*pow(u_t,3./2.));

   e2->x = y_t*(-1. - 2.*a*cos(2.*t)) - x_t*4.*a*sin(2.*t);
   e2->y = y_t*h*sin(t) - x_t*h*cos(t);
   e2->z = y_t*2.*d*sin(2.*t) - x_t*4.*d*cos(2.*t);

   *e1 = scale(  x_t, *e1 );
   *e2 = scale( 1./norm(*e2), *e2 );
   *e3 = cross( *e1, *e2 );
}

struct Vector3 fiberCurve( double t, double a, double h, double d,
                           double r, double omega, double phi )
{
   struct Vector3 gamma_t;
   struct Vector3 eta_t;
   struct Vector3 e1, e2, e3;
   double theta_t;

   gamma_t = yarnCurve( t, a, h, d );
   frenetFrame( t, a, h, d, &e1, &e2, &e3 );
   theta_t = t*omega - 2.*cos(t) + phi;

   eta_t.x = gamma_t.x + r*( cos(theta_t)*e2.x + sin(theta_t)*e3.x );
   eta_t.y = gamma_t.y + r*( cos(theta_t)*e2.y + sin(theta_t)*e3.y );
   eta_t.z = gamma_t.z + r*( cos(theta_t)*e2.z + sin(theta_t)*e3.z );

   return eta_t;
}

void writeYarnCurves(
      const char* filename, /* output filename */
      int nRows, /* number of rows in wale direction */
      double rowOffset, /* row spacing in wale direction */
      int nLoops, /* number of loops in course direction */
      int samplesPerLoop, /* sample points for each loop */
      double a, /* loop roundness */
      double h, /* loop height */
      double d ) /* loop depth */
{
   double t;
   int row, loop, sample;
   int nPoints;
```

```
   double dt, y0, t0;
   struct Vector3 gamma_t;
   FILE* out;

   out = fopen( filename, "w" );

   nPoints = 0;
   dt = (2.*M_PI)/(double)samplesPerLoop;

   /* write vertices */
   for( row = 0; row < nRows; row++ )
   {
      y0 = rowOffset * (double)row;

      for( loop = 0; loop < nLoops; loop++ )
      {
         t0 = 2.*M_PI * loop;
         for( sample = 0; sample < samplesPerLoop; sample++ )
         {
            t = t0 + dt*(double)sample;

            gamma_t = yarnCurve( t, a, h, d );

            fprintf( out, "v %e %e %e\n",
                  gamma_t.x,
                  gamma_t.y + y0,
                  gamma_t.z );
         }
      }
   }

   /* write polylines */
   for( row = 0; row < nRows; row++ )
   {
      fprintf( out, "l" );
      for( loop = 0; loop < nLoops; loop++ )
      {
         for( sample = 0; sample < samplesPerLoop; sample++ )
         {
            fprintf( out, " %d", nPoints+1 );
            nPoints++;
         }
      }
      fprintf( out, "\n" );
   }
   fclose( out );
}

void writeFiberCurves(
      const char* filename, /* output filename */
      int nRows, /* number of rows in wale direction */
      double rowOffset, /* row spacing in wale direction */
      int nLoops, /* number of loops in course direction */
      int samplesPerLoop, /* sample points for each loop */
      double a, /* loop roundness */
      double h, /* loop height */
```

```
      double d, /* loop depth */
      double r, /* yarn radius */
      double omega, /* amount of fiber twist */
      int nFibers ) /* number of fibers around yarn */
{
   int row, fiber, loop, sample;
   int nPoints;
   double y0, t0;
   double dt, dphi;
   struct Vector3 eta_t;
   FILE* out;

   out = fopen( filename, "w" );

   nPoints = 0;
   dt = (2.*M_PI)/(double)samplesPerLoop;
   dphi = (2.*M_PI)/(double)nFibers;

   /* write vertices */
   for( row = 0; row < nRows; row++ )
   {
      y0 = rowOffset * (double)row;

      for( fiber = 0; fiber < nFibers; fiber++ )
      {
         double phi = dphi * (double)fiber;

         for( loop = 0; loop < nLoops; loop++ )
         {
            t0 = 2.*M_PI * loop;
            for( sample = 0; sample < samplesPerLoop; sample++ )
            {
               double t = t0 + dt*(double)sample;
               eta_t = fiberCurve( t, a, h, d, r, omega, phi );
               fprintf( out, "v %e %e %e\n", eta_t.x, y0 + eta_t.y, eta_t.z );
            }
         }
      }
   }

   /* write polylines */
   for( row = 0; row < nRows; row++ )
   {
      y0 = rowOffset * (double)row;

      for( fiber = 0; fiber < nFibers; fiber++ )
      {
         fprintf( out, "l" );
         for( loop = 0; loop < nLoops; loop++ )
         {
            for( sample = 0; sample < samplesPerLoop; sample++ )
            {
               fprintf( out, " %d", nPoints+1 );
               nPoints++;
            }
         }
```

```
         fprintf( out, "\n" );
      }
   }
}

int main( int argc, char** argv )
{
   const int nRows = 16;           /* number of rows generated */
   const int nLoops = 12;          /* number of loops in each row */
   const int samplesPerLoop = 64;  /* points sampled per period */
   const int nFibers = 4;          /* number of twisted fibers */
   const double a = 3/2.;          /* loop roundness */
   const double d = 1.;            /* loop depth */
   const double h = 4.;            /* loop height */
   const double w = h + 1./2.;     /* row spacing */
   const double r = 1./2.;         /* yarn radius */
   const double omega = 5.;        /* fiber twist */

   writeYarnCurves( "yarn.obj", nRows, w, nLoops, samplesPerLoop, a, h, d );
   writeFiberCurves( "fibers.obj", nRows, w, nLoops, samplesPerLoop, a, h, d,
                     r, omega, nFibers );

   return 0;
}
```