Transfinite Surface Library

ShapEx

July 21, 2015

1 Conventional Geometry

Conventional geometric entities are provided by a shared library, and are accessed through the header file `geometry.hh`. The main classes defined there are as follows:

- `Vector2D`, `Vector3D`: 2- and 3-dimensional vectors, with the usual operators.
- `Point2D`, `Point3D`: Aliases for `Vector2D` and `Vector3D`.
- `DoubleVector`, `Vector2DVector`, `Point2DVector`, etc.: Aliases for STL vectors containing the respective objects.
- `BSCurve`: A (non-rational) B-spline curve.
- `BSSurface`: A (non-rational) B-spline surface, with or without trimming. This is a dummy class only for data transfer.
- `TriMesh`: A triangle mesh, capable of writing Wavefront `.obj` files via `writeOBJ`(*filename*).
- `IGES`: An IGES file writer, exporting the function `writeSurface`(*surface*).

2 Transfinite Surfaces

The general interface for handling transfinite interpolation surfaces is in the header file `surface.hh`. It defines the abstract class `Transfinite::Surface`, which provides several functions (detailed below). For actual use, one has to use a child class—presently there are four options: `SurfaceSideBased`, `SurfaceCornerBased`, `SurfaceGeneralizedCoons`, `SurfaceCompositeRibbon`, corresponding to the patch types SB, CB, GC and CR in [1], respectively. One- and two-sided surfaces are not supported.

2.1 Surface Details

Transfinite interpolation patches have several constituents, such as ribbon surfaces, domain polygons, parameterization mappings, or blending functions. The above surface types already define the blending functions, but there are several choices in all of the other parameters.

Ribbons

In this library, we have only one ribbon type, implemented in `RibbonCompatible`, which can be formulated as follows (using the notations of [1]):

$$R_i(s,d) = P_i(s) + \left[-P'_{i-1}(1)^\perp \cdot (1-s) + P'_{i+1}(0)^\perp \cdot s\right] \cdot d,$$

where $v^\perp$ is the vector v projected into the tangent plane at the current point, i.e.,

$$v^\perp = v - n(vn),$$

n being the unit normal vector obtained by a rotation minimizing frame [3] between the normals of the end vertices.

Ribbons can also have *ribbon handlers* (unit vectors defining the central direction) and *multipliers* (a quantity that pulls or pushes the center of the ribbon). This variation is implemented in `RibbonCompatibleWithHandler`, which uses the following equation:

$$R_i(s,d) = P_i(s) + \left[-P'_{i-1}(1)^\perp \cdot \alpha_0(s) + T_i(0.5)^\perp \cdot \alpha_1(s) + P'_{i+1}(0)^\perp \cdot \alpha_2(s)\right] \cdot d,$$

where

$$\alpha_0(s) = 2(s-1)(s-0.5), \quad \alpha_1(s) = -4(s-1)s, \quad \alpha_2(s) = 2(s-0.5)s,$$

and $T_i(0.5)$ can be computed as

$$T_i(0.5) = H_i^\perp \cdot \frac{\left\|-P'_{i-1}(1)^\perp + P'_{i+1}(0)^\perp\right\|}{2} \cdot m_i,$$

with H_i and m_i being the ribbon handler and multiplier of the ribbon, respectively. Note, that this is constant w.r.t. s and d, so it is precomputed. The default direction is computed such that

$$T_i(0.5) = -\frac{1}{2}P'_{i-1}(1)^\perp + \frac{1}{2}P'_{i+1}(0)^\perp,$$

when $m_i = 1$.

Domain Polygons

There are three domain polygon computations implemented in the library. The simplest is `DomainRegular`, which uses a regular n-sided polygon as the domain. There is also `DomainCircular`, where an n-sided cyclic polygon is constructed in such a way that the central angles and the arc lengths of the boundary curves have a constant ratio, and `DomainAngular`, that takes both arc lengths and angles into consideration. For a detailed description of these methods, see [2]. SP patches use the angular domain, and all other surface types employ the regular domain.

Parameterization

Each of the patches has its own parameterization. For details on these mappings, see [1].

- SB patches use parallel parameterization (`ParameterizationParallel`): s is linear, as in the bilinear parameterization, and d is proportional to the perpendicular distance to the base side, with the farthest vertex set to $d = 1$.
- CB patches use bilinear parameterization (`ParameterizationBilinear`), or more correctly, linear parameterization, since this representation uses only side parameters.
- GC patches use bilinear-based interconnected parameterization (`ParameterizationInterconnected`).
- CR patches use a parameterization based on Wachspress coordinates λ_i ($i \in [1 \dots n]$):

  $$\begin{aligned} s_i &= \lambda_i / (\lambda_{i-1} + \lambda_i), \\ d_i &= 1 - \lambda_{i-1} - \lambda_i, \end{aligned}$$

  implemented in `ParameterizationBarycentric`. (This computation of the s parameter is not published yet.)

2.2 Surface Creation

After creating a surface object of the desired type, the following steps finalize the surface:

1. Set the boundary curves using `setCurves`(*curves*). Alternatively curves can be set one by one, using `setCurve`(*i*, *curve*) to set the *i*-th curve.
2. Call `setupLoop`() to normalize the curves and fill in adjacency information. This function has to be called every time a new curve is assigned to the surface.
3. Call `setRibbonMultiplier`(*i*, *multiplier*) and `setRibbonHandler`(*i*, *handler*) to set the central behavior of the *i*-th ribbon. (Only applicable when using ribbons with handlers.)

4. Call `update`() to generate ribbons and clear the cache. This function has to be called every time the edges (or their ribbon handlers / multipliers) have been changed. If only the i-th curve has changed, it is sufficient to call `update`(i). (But if the end tangent of the curve changes, `update` should be called for the adjacent side, as well.)

5. Use of the γ function [1] can be turned off by `setGamma`(`false`). It is on by default.

2.3 Evaluation

A point p in the 2D domain can be evaluated by calling `eval`(p). For convenience, there is an overloaded function that evaluates the surface using a given resolution, thereby creating a triangle mesh: `eval`(*resolution*).

2.4 Fitting

There are two options for fitting quadrilateral B-spline surfaces.

Central Split

This method splits the n-sided transfintie surface into n quadrilaterals, and fits a B-spline surface on each of them. (Except in the 4-sided case, where only one B-spline surface is used.) The function for this is `fitCentralSplit`(*ftol*, *ktol*, *density*). The parameters are:

- *ftol* is the main fitting tolerance to be achieved.
- *ktol* is the knot snapping tolerance, which is the smallest allowed difference between two knot values. As very close knots can cause artefacts, we recommend setting this value to at least 0.01 (curves are normalized to the interval [0,1]).
- *density* is the sampling density for the subdividing curves, the default is set to 30.

Trimming

This method fits a larger B-spline surface on the n-sided patch, and trims the sides with the original boundary curves. The function for this is `fitTrimmed`(*ftol*, *resolution*, *maxu*, *maxv*, *wcurv*, *wosc*). The parameters are:

- *ftol* is the main fitting tolerance to be achieved.
- *resolution* is the surface's sampling resolution, as in `eval`. Defaults to 15.
- *maxu* and *maxv* are the maximum number of control points in the u and v directions, respectively. These are needed because of the idiosyncrasies of the fitting algorithm. The default is 12 in both directions.

- *wcurv* and *wosc* are the curvature and oscilliation minimization weights, respectively. The default is 1e-6 for both.

3 Python Interface

In Python, we have supplied one command, `transfiniteSurface`(*curve_ list*, *surface_ type*, *split_ or_ trim*, *tolerance*), that creates a B-spline surface from a list of boundary curves. The following sections explain the arguments one by one.

3.1 Curve List

This is a Python list of curves: [*curves*]. Each curve is represented as a tuple: (*degree*, [*knots*], [*points*]). Knots are float values, points are lists of three floats. An example curve looks like this: (3,[0,0,0,0,1,1,1,1],[[0,0,0],[1,1,0],[2,1,1],[3,0,1]]).

There are a few assumptions about the curve list:

- There are at least 3 curves in the list. Two-sided surfaces are not supported.
- The curves are given in sequence, i.e., they represent the boundary edges of the surface in a clockwise or anti-clockwise manner. The parameterization of the curves can be arbitrary.
- Adjacent curves should enclose an angle between (around) 10 and 170 degrees. Angles outside this range can cause the surface to twist out of its boundary loop.

3.2 Surface Type

This determines the representation to use. It can be one of the following strings: 'SB', 'CB', 'GC', or 'CR', as explained in Section 2.

3.3 Split or Trim

This option chooses the fitting method (see Section 2.4). It can be one of the following strings: 'split' or 'trim'.

3.4 Tolerance

This is the fitting tolerance (*ftol* in Section 2.4).

3.5 Return Value

The type of the return value depends on the fitting method.

Split

The central split method always returns with a list of surfaces. For four-sided surfaces, this will be a one-element list, but for n-sided patches it will have n elements. Each surface is represented as a tuple: (*deg_u*, *deg_v*, [*knots_u*], [*knots_v*], [*points*]). The control points are stored in row-major order.

Trim

With this option, the return value is a single trimmed surface, represented as a tuple: (*surface*, [*param_curves*]). Here the second item of the tuple is a list of trimming curves in the parameter domain—their definition is the same as in Section 3.1, but using two-dimensional points.

References

[1] P. Salvi, T. Várady, A. Rockwood, *Ribbon-based Transfinite Surfaces.* Computer Aided Geometric Design, Vol. 31(9), pp. 613–630, 2014.

[2] T. Várady, A. Rockwood, P. Salvi, *Transfinite Surface Interpolation over Irregular n-sided Domains.* Computer Aided Design, Vol. 43(11), pp. 1330–1340, 2011.

[3] W. Wang, B. Jüttler, D. Zheng, Y. Liu, *Computation of Rotation Minimizing Frames.* Transactions on Graphics, Vol. 27(1), p. 2, 2008.

Appendix – Example Session

This is a transcript of a demo session. The output is formatted and commented.

```
$ python
python Python 2.7.6 (default, Jun 22 2015, 18:00:18)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pytransurf
>>> curve1 = (3, [0,0,0,0,0.267,0.694,1,1,1,1],
              [[53.918,-100,      8.475],
               [56.646, -92.446,  9.017],
               [59.028, -71.846,  7.977],
               [44.362,   7.722,  8.555],
               [12.857,  11.697,  8.492],
               [ 0.019,  11.841,  8.485]])
>>> curve2 = (3, [0,0,0,0,0.538,1,1,1,1],
              [[ 0.019,  11.841,  8.485],
               [ 0.846,  -8.867, -2.257],
               [ 0.145, -52.128,-28.903],
              [ 0.602, -89.791,-24.169],
              [ 0.414,-107.179,-23.856]])
>>> curve3 = (3, [0,0,0,0,0.594,1,1,1,1],
             [[ 0.414,-107.179,-23.856],
              [12.999,-107.324,-24.444],
              [36.495,-101.455,  0.997],
              [47.675,-100.196,  6.224],
              [53.918,-100,      8.475]])
>>> pytransurf.transfiniteSurface([curve1,curve2,curve3],'SB','trim',0.01)
((3, 3, # surface - deg_u, deg_v
  # knots_u:
  [-0.856, -0.856, -0.856, -0.856, -0.626, -0.512,
   -0.397, -0.282, -0.167, -0.053,  0.062,  0.177,
    0.292,  0.406,  0.521,  0.980,  0.980,  0.980, 0.980],
  # knots_v:
  [-0.986, -0.986, -0.986, -0.986, -0.820, -0.654,
   -0.570, -0.487, -0.404, -0.321, -0.155,  0.012,
    0.344,  0.344,  0.344,  0.344],
  # control points:
  [[  84.221,  -81.681,   8.833], [  79.594,  -84.601,   9.797],
   [  70.826,  -90.341,  10.946], [  61.096,  -96.390,  10.579],
   [  54.125, -100.002,   8.618], [  49.261, -102.273,   6.817],
   [  44.919, -104.295,   4.602], [  39.783, -106.875,   1.050],
   [  34.187, -110.240,  -4.026], [  27.163, -116.394, -12.416],
   [  20.900, -121.984, -14.580], [  15.589, -126.348, -13.819],
   [  82.869,  -76.954,  11.017], [  78.053,  -79.545,  11.684],
   [  69.266,  -84.781,  12.036], [  59.956,  -90.511,  10.026],
   [  53.024,  -94.337,   7.893], [  48.051,  -96.851,   6.209],
   [  43.526,  -99.161,   4.096], [  38.212, -102.149,   0.506],
   [  32.493, -105.918,  -4.775], [  24.253, -112.326, -12.991],
   [  18.193, -117.888, -17.039], [  13.193, -122.077, -18.750],
   [  80.793,  -68.950,  12.166], [  75.906,  -71.442,  12.459],
   [  67.127,  -76.464,  11.905], [  57.936,  -82.179,   9.104],
   [  51.156,  -86.213,   6.373], [  46.289,  -89.011,   4.264],
   [  41.711,  -91.662,   1.983], [  36.114,  -95.112,  -1.607],
   [  29.830,  -99.406,  -6.881], [  20.522, -106.291, -15.354],
   [  13.775, -111.927, -20.317], [   8.701, -115.912, -23.022],
   [  77.300,  -57.536,  11.890], [  72.274,  -60.117,  12.124],
   [  63.370,  -65.216,  11.125], [  54.231,  -71.168,   7.678],
   [  47.736,  -75.637,   4.142], [  43.147,  -78.862,   1.270],
   [  38.754,  -81.991,  -1.706], [  33.116,  -86.069,  -5.836],
   [  26.328,  -91.035, -11.274], [  15.471,  -98.824, -19.387],
   [   7.377, -104.824, -23.864], [   1.059, -108.926, -24.207],
   [  74.123,  -48.858,  11.391], [  69.050,  -51.569,  11.629],
   [  60.050,  -56.860,  10.569], [  51.055,  -63.104,   6.526],
   [  44.756,  -67.903,   2.288], [  40.242,  -71.425,  -1.012],
   [  35.890,  -74.880,  -4.384], [  30.257,  -79.407,  -8.990],
   [  23.237,  -84.917, -14.641], [  11.668,  -93.530, -22.783],
   [   2.616,  -99.991, -25.803], [  -5.026, -104.151, -23.221],
   [  70.195,  -40.319,  11.215], [  65.215,  -43.165,  11.299],
   [  56.310,  -48.723,  10.022], [  47.604,  -55.293,   5.092],
   [  41.404,  -60.416,   0.245], [  36.921,  -64.218,  -3.455],
   [  32.580,  -67.976,  -7.190], [  26.863,  -72.925, -12.132],
   [  19.565,  -78.988, -17.866], [   7.657,  -88.545, -25.700],
   [  -2.771,  -95.579, -25.506], [ -11.304,  -99.890, -20.507],
   [  65.659,  -32.053,  11.273], [  60.870,  -35.050,  11.084],
   [  52.263,  -40.880,   9.250], [  43.841,  -47.787,   3.433],
   [  37.669,  -53.235,  -1.918], [  33.184,  -57.304,  -5.976],
   [  28.801,  -61.348,  -9.998], [  22.935,  -66.706, -15.128],
   [  15.517,  -73.336, -21.023], [   3.063,  -83.901, -27.229],
   [  -9.275,  -91.624, -23.159], [ -18.457,  -96.306, -16.687],
   [  60.651,  -24.196,  11.463], [  56.140,  -27.328,  10.793],
   [  47.931,  -33.444,   8.197], [  39.756,  -40.688,   1.611],
   [  33.565,  -46.447,  -4.148], [  29.018,  -50.762,  -8.490],
   [  24.533,  -55.072, -12.709], [  18.560,  -60.833, -17.935],
   [  11.138,  -68.047, -23.669], [  -2.728,  -79.677, -26.698],
   [ -16.959,  -88.189, -19.296], [ -27.410,  -93.442, -10.821],
   [  55.339,  -16.799,  11.520], [  51.124,  -20.071,  10.299],
   [  43.319,  -26.476,   6.898], [  35.322,  -34.076,  -0.274],
   [  29.114,  -40.159,  -6.285], [  24.504,  -44.724, -10.773],
   [  19.928,  -49.298, -15.074], [  13.859,  -55.416, -20.339],
   [   5.917,  -63.244, -24.772], [  -9.881,  -75.901, -24.373],
   [ -26.135,  -85.327, -13.554], [ -38.239,  -91.311,  -2.773],
   [  49.927,   -9.890,  11.100], [  45.864,  -13.352,   9.574],
   [  38.389,  -20.054,   5.466], [  30.514,  -28.033,  -2.097],
   [  24.340,  -34.462,  -8.191], [  19.724,  -39.290, -12.656],
   [  15.041,  -44.120, -16.810], [   8.511,  -50.651, -21.292],
   [  -0.531,  -58.987, -24.116], [ -18.445,  -72.592, -20.230],
   [ -36.938,  -83.061,  -5.997], [ -50.982,  -89.928,   7.579],
   [  44.222,   -3.634,  10.564], [  40.320,   -7.289,   8.717],
   [  33.134,  -14.279,   4.004], [  25.315,  -22.624,  -3.728],
   [  19.081,  -29.380,  -9.648], [  14.334,  -34.468, -13.787],
   [   9.333,  -39.623, -17.216], [   2.118,  -46.528, -20.559],
   [  -8.180,  -55.231, -21.787], [ -28.492,  -69.822, -14.069],
   [ -49.321,  -81.339,   3.408], [ -65.509,  -89.201,  20.140],
   [  38.116,    1.824,  10.084], [  34.448,   -1.976,   7.827],
   [  27.457,   -9.267,   2.757], [  19.504,  -17.971,  -4.732],
   [  12.993,  -25.026, -10.094], [   7.887,  -30.360, -13.543],
   [   2.407,  -35.695, -16.238], [  -5.578,  -42.841, -18.426],
   [ -17.141,  -52.086, -17.316], [ -40.013,  -67.524,  -6.072],
   [ -63.213,  -80.106,  14.656], [ -81.690,  -89.037,  34.815],
   [  25.115,   11.079,   9.140], [  21.706,    6.864,   6.687],
      [  14.876,   -1.000,   1.466], [   6.234,  -10.204,  -4.992],
      [  -1.301,  -17.614,  -8.770], [  -7.425,  -23.159, -10.752],
      [ -14.150,  -28.843, -11.370], [ -23.957,  -36.703, -10.021],
      [ -38.011,  -46.810,  -4.638], [ -66.057,  -63.932,  14.030],
      [ -93.982,  -78.556,  41.159], [-116.997,  -89.649,  68.060],
      [  12.013,   14.621,   9.891], [   8.686,   10.196,   7.648],
      [   1.459,    2.396,   2.913], [  -8.988,   -6.389,  -1.617],
      [ -18.344,  -13.729,  -2.759], [ -25.897,  -19.493,  -2.134],
      [ -34.029,  -25.496,  -0.019], [ -45.793,  -33.773,   4.812],
      [ -62.354,  -44.517,  14.587], [ -94.182,  -63.074,  40.038],
      [-125.765,  -79.044,  72.266], [-151.802,  -91.722, 103.183],
      [   0.187,   14.508,   9.950], [  -3.101,   10.453,   7.719],
      [ -10.398,    3.199,   3.765], [ -22.063,   -5.136,   2.182],
      [ -33.255,  -12.755,   5.415], [ -42.543,  -18.987,  10.056],
      [ -52.586,  -25.537,  16.319], [ -66.785,  -34.427,  26.019],
      [ -85.675,  -45.327,  39.141], [-120.641,  -63.682,  67.047],
      [-154.789,  -80.013, 101.132], [-183.194,  -93.497, 134.500]]
    # parameter curves follow
    [(3, # param_curve 1 - degree
      # knots:
      [0, 0, 0, 0, 0.233, 0.467, 0.7, 0.933, 0.933, 0.933, 0.933],
      # control points:
      [[-0.855, -0.566], [-0.783, -0.612], [-0.623, -0.690],
       [-0.121, -0.832], [ 0.435, -0.966], [ 0.711, -0.961],
       [ 0.844, -0.954]]),
     (3, # param_curve 2 - degree
      # knots:
      [0, 0, 0, 0, 0.467, 0.933, 0.933, 0.933, 0.933],
      # control points:
      [[ 0.953, -0.950], [ 0.731, -0.713], [ 0.335, -0.279],
       [-0.123,  0.061], [-0.377,  0.245]]),
     (3, # param_curve 3 - degree
      # knots:
      [0, 0, 0, 0, 0.467, 0.933, 0.933, 0.933, 0.933],
      # control points:
      [[-0.483,  0.329], [-0.566,  0.230], [-0.744, -0.142],
       [-0.808, -0.429], [-0.839, -0.520]])])
   >>>
```