OPTITEX

Tension XY calculations explanation

This code estimates how much the material is "distorted" away from its default shape. So stretched or sheared in either warp or weft direction.
 The engineer version is that what the function is actually doing is calculating the deformation gradient. The deformation gradient is a 3x2 matrix. It takes a vector in undeformed "material coordinates" (2D), and maps it to the current configuration (3D) of the triangle. So that you can use it to measure how undeformed material vectors are deformed in the current triangle.

Here's a triangle:

```
   V2

   |\

   |  \

V0  --- V1
```

The code calls the current positions of these three vertices p0, p1, and p2. We also need the undeformed positions of these triangles. For us, these correspond to the UV coordinates. Let’s call them (u0, v0), (u1, v1), (u2, v2).

We use the change in UV coordinates from the corner vertex to the other two.

get_pu1() = u1 - u0

get_pu2() = u2 - u0

get_pv1() = v1 - v0

get_pv2() = v2 - v0

Let's then get delta vectors of the current deformed triangle:

dx1 = p1 - p0

dx2 = p2 - p0

and for the original undeformed:

du1 = [get_pu1(), get_pv1()]

du2 = [get_pu2(), get_pv2()]

Given that the deformation gradient (I'll call it F), maps undeformed vectors to deformed vectors, it makes sense that it would map these edge vectors in UV coordinates to the edge vectors in 3D coordinates:

dx1 = F * du1

dx2 = F * du2

This is a system of equations that you can use to solve for F:

[dx1 dx2] = F * [du1 du2]

F = [dx1 dx2] * [du1 du2]^-1

This results in:

wu = (dx1 * get_pv2 - dx2 * get_pv1) / UV_det

wv = (dx1 * -get_pu2 + dx2 * get_pu1) / UV_det

UV_det is the original undistorted face area uv data.

This is exactly the columns of F above, just "manually" computing the inverse of the 2x2 matrix, using some precomputed values (including the precomputed area of the triangle, which corresponds to the determinant of the 2x2 matrix [du1 du2]).

Then it takes lu=||wu|| and lv=||wv||, the lengths of these two vectors. (also normalizes wu and wv)

wu and wv correspond to if you took the undeformed vectors (1,0) and (0,1), which are just the warp and weft vectors, and mapped them to 3D (wu = F * (1,0) and wv = F*(0,1)). So their lengths tells you how much the triangle has deformed in the warp and weft directions. A value of 1 means no deformation. Less than 1 means compressed, greater than 1 is stretched

```
lu = wu.NormalizeL2();
lv = wv.NormalizeL2();
```

Lastly it computes rlu and rlv from precomputed values. In the beginning, before the simulation these values are set as just the rest lengths of lu and lv from the initial triangle positions.

Now to get the tension XY we take the change in warp and weft deformations (lu-rlu) and (lv-rlv) and multipliy them by the stiffness to convert a geometric measure into a tension value. We use clipped stiffness value.

```
double m_KsXclip=min(5e4,m_KsX);
double m_KsYclip=min(5e4,m_KsY);

float   ampU=(lu-rlu)*(float)(m_KsXclip);
float   ampV=(lv-rlv)*(float)(m_KsYclip);
```

If these are less than 0 then it clamps it to 0 (so basically it ignores compression)

```
ampU=max(ampU,0.0f);
ampV=max(ampV,0.0f);
wu.SetLength(ampU);
wv.SetLength(ampV);
```

With these in hand, the code needs to convert them to a single number across each face. This can be tricky because the u direction might be really stretched with no deformation in the v direction, or they might be flipped, or there might be a little in each direction.

All the code does is take the combined length of wu and wv. If they are facing in opposite directions it uses wu-wv so that it doesn't accidentally show a small number; it's only interested in absolute values of the

OPTITEX

deformations.

```
float   dotProd=wu*wv;
 float  len=0;
if (dotProd<0)
len=(wu-wv).Length();
else
len=(wu+wv).Length();

pFaceWeights[f] = len;
```

It uses the dot product to see if they are facing opposite directions, but basically it just adds these two vectors together (if they are pointing opposite direction, it reverses wv. Then the length of this vector is the number used to color the vertices in the map.

Let's walk through an actual simple example as you requested. Let

u0 = [0, 0]

u1 = [1, 0]

u2 = [0, 1]

p0 = [0, 0, 0]

p1 = [1.1, 0, 0]

p2 = [0, 1, 0]

du1 = [1, 0]

du2 = [0, 1]

dx1 = [1.1, 0, 0]

dx2 = [0, 1, 0]

So very simple right triangle. The triangle hasn't moved, just one vertex has moved very slightly to the right.

So the inverse of [du1 du2] is very simple, since it is just the identity matrix.. This gives a deformation gradient of

F = [dx1 dx2] * [du1 du2]^-1 = [dx1 dx2] * I = [dx1 dx2]

wu and wv are just the columns of F, so wu = dx1 and wv = dx2. lu = 1.1 and lv = 1.0. rlu is 1.0 and rlv = 1.0.

Let kx and ky be the two stiffnesses, so then we have ampU = (1.1 - 1.0) * kx = 0.1 * kx and ampV = (1.0 - 1.0) * ky = 0.0.

Then the final tension value used is just 0.1 * kx.