Constructing Streak Surfaces for 3D Unsteady Vector Fields: Supplementary Material

Tony McLoughlin[*] Swansea University
Robert S. Laramee[†] Swansea University
Eugene Zhang[‡] Oregon State University

1 Divergence Implementation

In order to simplify the implementation of splitting due to divergence, we test one quad edge at a time and update the topology of the mesh accordingly. For example, Algorithm 1 shows how we update the mesh topology if a quad's west edge length $|E_W| > d_{sep}$. First we test for an existing T-junction on the west edge. If there is one we use it, otherwise we interpolate a new vertex. Likewise for the east edge. Then we construct a new quad for the north half, as in of Figure 12. Another procedure is called to update the resulting topology (Quad::UpdateNeighborsWest()) shown in Algorithm 2. The methods that handle divergence associated with the other quad edges are similar to Quad::DivideWest().

Figure 12 shows a possible subtle configuration for the Quad::DivideWest() operation. We have to test whether the northwest vertex, V_{NW}, is a T-junction. If it is we change the T-junction's extra neighbor pointer to point to the newly inserted quad. The northern neighbor's southern pointer remains the same. The red lines indicate the correct pointer configuration after the split.

2 Convergence Implementation

In order to simplify the implementation of merging a pair of quads, we take a similar approach to that described in Section 1. We test one quad edge at a time, e.g. north, and test for candidate quads to merge with. Algorithm 3 shows the implementation used to merge with a northern neighbor. The topological cases associated with the merging are illustrated in Figure 7. First we test to see which case we fall into by the presence of a T-junction on the east edge of our western neighbor. A similar test is performed on the opposite side. The mesh topology is then updated according to the case the quad is in. The pseudo-code is given in Algorithm 3. Similar functions exist for the other directions e.g. Quad::MergeWest().

3 Shear Implementation

The implementation of shear is divided up into five basic functions:

1. Quad::shearEastWithT()
2. Quad::ShearEastNoT()
3. Quad::ShearEastUpdateMesh()
4. Quad::ShearEastDivideWestWithT()
5. Quad::ShearEastDivideWestNoT()

We identify two key cases when we perform the alteration to the mesh topology. Case 1 is the simpler case where there is a T-junction present that we can connect to, this is illustrated in Figure 8. The implementation is shown in Algorithm 4. In this case we simply reconnect the north edge to the T-junction. This may result in a new T-junction being added to the northern neighbor (top row).

The second case is slightly more complicated. Here we have no T-junction on the east side. This forms an intermediate triangle in the mesh. See Figure 9 and Algorithm 5. We decompose the triangle into three quads by inserting a new point in the middle of the intermediate triangle and on each of its edges. We then subdivide the triangle into three quads. This method is adapted from Alliez et al. [Alliez et al. 2003]. Alliez et al. show how triangular meshes can be converted to quad meshes.

Algorithms 5 - 8 show the implementation associated with Figures 9 and 10. The algorithm starts off by testing for the presence of a western T-junction on our eastern neighbor. It then calls Quad::shearEastUpdateMesh() shown in Algorithm 6. Algorithm 6 starts off by creating two new mesh vertices: the vertex on the north edge and a vertex in the center of the intermediate triangle.

It then calls procedures to create the new west and east quads shown in Figure 10. The sub-procedure for creating the new west quad is shown in Algorithms 7 and 8.

References

Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., and Desbrun, M. 2003. Anisotropic polygonal remeshing. *SIGGRAPH 03 - ACM Transactions on Graphics 22*, 3, 485–493.

[*]e-mail: cstony@swan.ac.uk
[†]e-mail:R.S.Laramee@swan.ac.uk
[‡]e-mail:zhange@eecs.oregonstate.edu

Algorithm 1 Quad::DivideWest(*void*)

This procedure is called whenever $|E_W| > d_{sep}$.

```
Quad newQuad;
Vertex newEastVertex, newWestVertex;

IF(this->HasT_Junction(WEST)) THEN
   newWestVertex = this->GetT_Junction(WEST).GetVertex();
ELSE
   newWestVertex = Mid(this->GetVertex(NW), this->GetVertex(SW));
   this->AddT_JunctionToWestNeighbor(newWestVertex, newQuad);
ENDIF

IF(this->HasT_Junction(EAST) THEN
   newEastVertex = this->GetT_Junction(EAST).GetVertex();
ELSE
   newEastVertex = Mid(this->GetVertex(NE), this->GetVertex(SE));
   this->AddT_JunctionToEastNeighbor(newEastVertex, newQuad);
ENDIF

//Construct a new quad object on north half.
newQuad.SetNWVertex(this->GetNWVertex());
newQuad.SetNEVertex(this->GetNEVertex());
newQuad.SetSWVertex(newWestVertex);
newQuad.SetSEVertex(newEastVertex);

//Update this quad's vertices
this->SetNWVertex(newWestVertex);
this->SetNEVertex(newEastVertex);

//Update neighbor pointers for new quad
this->UpdateNeighborsWest(newQuad);

RETURN;
```

Algorithm 2 Quad::UpdateNeighboursWest(*Quad newQuad*)

This procedure updates this quad's and neighboring quad's topology after a divide. See also Algorithm 1.

```
/* Update new quad's neighbor pointers */
newQuad.SetNorthNeighbor(this->GetNeighbor(NORTH));
newQuad.SetSouthNeighbor(this);

IF(this->HasT_Junction(WEST)) THEN
   newQuad.SetWestNeighbor(this->GetT_Junction(WEST).GetNeighbor());
   this->GetT_Junction(WEST).GetNeighbor().SetEastNeighbor(newQuad);
   this->deleteT_Junction(WEST);
ELSE
   newQuad.SetWestNeighbor(this->GetNeighbor(WEST));
ENDIF

IF(this->HasT_Junction(EAST)) THEN
   newQuad.SetEastNeighbor(this->GetT_Junction(EAST).GetNeighbor());
   this->GetT_Junction(EAST).GetNeighbor().SetWestNeighbor(newQuad);
   this->deleteT_Junction(EAST);
ELSE
   newQuad.SetEastNeighbor(this->GetNeighbor(EAST));
ENDIF

/**
* Update this quad's northern neighbor's southern
* pointer to new quad. This handles the subtle case
   *highlighted in Figure 12 of the manuscript.
*/
IF(this->GetNeighbor(NORTH).GetNeighbor(SOUTH) == this) THEN
   this->GetNeighbor(NORTH).SetSouthNeighbor(newQuad);
ELSE IF((this->GetNeighbor(NORTH).HasT_Junction(SOUTH))
  this->GetNeighbor(NORTH).GetT_Junction(SOUTH).SetNeighbour(newQuad);
ENDIF

/**
* Update this quad's northern neighbor's pointer to
* new quad.
*/
this->SetNorthNeighbor(newQuad);

/**
* If this quad had a northern T-junction,
* move it to the new quad.
*/
IF(this->HasT_Junction(NORTH)) THEN
   newQuad.SetT_JunctionNorth(this->GetT_Junction(NORTH));
   this->SetT_JunctionNorth(NULL);
ENDIF

RETURN;
```

Algorithm 3 Quad::MergeNorth(*void*)

This procedure is called whenever the edge length of the west AND east sides $< d_{converge}$. One of the following must also be true:
1. The NW vertex is a T-Junction OR
2. The NE vertex is a T-Junction OR
3. None of the corner vertices are T-Junctions.
In these three cases, this quad merges with it's northern neighbor.

```
/**
* If our WESTern neighbor has an EAST T-junction
* THEN delete it. (Cases 2,3,4,5)
* ELSE introduce WESTern T-junction to the new merged
* quad (this). (Cases 1,6)
*/
IF(this->GetNeighbor(WEST).HasT_Junction(EAST)) THEN
   this->GetNeighbor(WEST).GetT_Junction(EAST).DeleteVertex();
   this->GetNeighbor(WEST).DeleteT_Junction(EAST);
ELSE
   this->NewT_Junction(WEST);
   this->GetT_Junction(WEST).SetVertex(this.GetVertex(NW));
   this->GetT_Junction(WEST).SetWesternNeighbor(this.GetNeighbor(NORTH).GetNeighbor(WEST));
ENDIF

/* Perform mirror opposite on the EAST side. */
IF(this->GetNeighbor(EAST).HasT_Junction(WEST)) THEN
   this->GetNeighbor(EAST).GetT_Junction(WEST).DeleteVertex();
   this->GetNeighbor(EAST).DeleteT_Junction(WEST);
ELSE
   this->NewT_Junction(EAST);
   this->GetT_Junction(EAST).SetVertex(this.GetVertex(NE));
   this->GetT_Junction(EAST).SetEasternNeighbor(this.GetNeighbor(NORTH).GetNeighbor(EAST));
ENDIF

/* Update to new northern vertices. */
this->SetNWVertex.(this->GetNeighbor(NORTH).GetVertex(NW));
this->SetNEVertex.(this->GetNeighbor(NORTH).GetVertex(NE));
formerNorthNeighbor = this.GetNeighbor(NORTH);

/* Update new north neighbor's south pointer. */
this->GetNeighbor(NORTH).GetNeighbor(NORTH).SetSouthNeighbor(this);

/* Update new neighbor pointer. */
this.SetNorthNeighbor(this.GetNeighbor(NORTH).GetNeighbor(NORTH));

/* Delete old north neighbor quad object. */
formerNorthNeighbor.Delete();

RETURN;
```

Algorithm 4 Quad::ShearWithEastT$(void)$
This method is called whenever a quad is sheared and
there's a T-Junction on the east edge, i.e.
IF
1. $\frac{d_{short}}{d_{long}} < \varepsilon_{shear}$ AND
2. $\theta_{NorthEast} < \theta_{shear}$ AND
3. this-¿HasT_Junction$(EAST) ==$ TRUE
Shear adjustment is not applied to quads at the
boundaries of the mesh.

```
Quad NEquad = this->GetNeighbor(NORTH).GetNeighbor(EAST)
IF (NEquad == NULL)
   RETURN

/* IF our NORTHEASTern neighbor has a WESTern
 *    T-junction
 * THEN snap the two T-Junctions together
 * ELSE create a new EAST T-junction for
 *    our NORTHern neighbor.
 * Update our NE vertex AND delete our EASTERN T-Junction
 */
IF (NEquad.HasT_Junction(WEST)) THEN
  NEQuad.SetWestNeighbour(this->GetNeighbor(NORTH));
  NEquad.GetT_Junction(WEST).DeleteVertex();
  NEquad.DeleteT_Junction(WEST);
ELSE
  this->GetNeighbor(NORTH).NewT_Junction(EAST);
  this->GetNeighbor(NORTH).GetT_Junction(EAST).SetVertex(this->GetNEvertex())
  this->GetNeighbor(NORTH).SetEasternNeighbor(this->GetT_Junction(EAST)->GetNeighbor());
END IF
this->SetNEvertex(this->GetT_Junction(EAST)->GetVertex());

/* Set new south east vertex of northern neighbor */
this->GetNeighbor(NORTH).SetSEVertex(this->GetT_Junction(EAST)->GetVertex());

this->DeleteT_Junction(EAST);
```

Algorithm 5
Procedure: Quad::ShearEastNoT(void)
This method is called whenever a quad is sheared and there's no T-Junction on the EAST edge, i.e. IF
1. $\frac{d_{short}}{d_{long}} < \varepsilon_{shear}$ AND
2. $\theta_{NorthEast} < \theta_{shear}$ AND
3. this-¿HasT_Junction$(EAST) ==$ FALSE
Shear update is not applied to quads at the boundaries of the mesh.

```
Quad Nquad = this->GetNeighbor(NORTH)
IF (Nquad == NULL)
  RETURN

/*
* IF our EASTern neighbor has a WESTern T-junction
* THEN delete the T-Junction
* ELSE create a new EAST T-junction for
*      our NORTHern neighbor.
*/
IF (this->GetNeighbor(EAST).HasT_junction(WEST)) THEN
  this->GetNeighbor(EAST).GetT_junction(WEST).DeleteVertex()
  this->GetNeighbor(EAST).DeleteT_junction(WEST)
ELSE
  this->GetNeighbor(NORTH).NewTjunction(EAST)
  this->GetNeighbor(NORTH).GetTjunction(EAST).SetVertex(this->GetNEvertex())
  this->GetNeighbor(NORTH).SetEastNeighbor(this->GetNeighbor(EAST))
END IF

/* Update south-east vertex of out northern neighbor */
this->GetNeighbor(NORTH).SetSEVertex(this->GetSEVertex());

/*
* Create 2 new quads and 3 new T-junctions
* Update our vertices and topology.
*/
this->shearEastUpdateMesh()
RETURN
```

Algorithm 6 Procedure: Quad::shearEastUpdateMesh(void)
This method updates the mesh topology resulting from the shear. See Algorithm 5

```
Vertex vertexNorth, vertexCenter
Quad quadWest, quadEast
/* This quad’s north-east vertex is updated in Algorithm 5 */

vertexNorth  = new Vertex(interpolate(this->GetNEvertex(), this->GetNWvertex())
vertexCenter = new Vertex(interpolate(this->GetNWvertex(), this->GetNWvertex(),
                                      this->GetSEvertex(), this->GetSWvertex())

/* First, create the WEST quad. */
IF (this->HasT_Junction(WEST)
THEN
  quadWest this->shearEastDivideWestWithT(vertexNorth, vertexCenter);
ELSE
  quadWest this->shearEastDivideWestNoT(vertexNorth, vertexCenter);

/* Second, create the EAST quad. */
IF (this->HasT_Junction(EAST)
THEN
  quadEast this->shearEastDivideEastWithT(vertexNorth, vertexCenter);
ELSE
  quadEast this->shearEastDivideEastNoT(vertexNorth, vertexCenter);
/* Update the quadWest and quadEast neighbor topology. */
quadWest.SetEastNeighbor(quadEast);
quadEast.SetEastNeighbor(quadWest);

/* Update this quad’s vertices. */
this->SetNWvertex(quadWest.GetSWvertex());
this->SetNEvertex(vertexCenter);
this->SetSEvertex(quadEast.GetSEvertex());
/* SW vertex stays the same. */

/* Update this quad’s topology. */
this->SetNorthNeighbor(quadWest);
this->SetEastNeighbor(quadEast);
/* West and South neighbors remain the same */

/* Add new quads to central quad list. */
this->GetQuadList()->Add(quadWest, quadEast);
```

Algorithm 7
Procedure: Quad::shearEastDivideWestWithT(
Vertex vertexNorth, Vertex vertexCenter)
This procedure creates a new west quad when a sheared quad is identified. In this case, this quad has a WESTern T-junction.

```
Quad quadWest

/* Update the WEST quad’s vertices. */
quadWest.SetNWvertex(this->GetNWvertex());
quadWest.SetNEvertex(vertexNorth);
quadWest.SetSEvertex(vertexCenter);
quadWest.SetSWvertex(this->GetTjunction(WEST).GetVertex());

/* Update the WEST quad’s topology
(except for new EAST quad). */
quadWest.SetNorthNeighbor(this->GetNeighbor(NORTH));
quadWest.SetSouthNeighbor(this);
quadWest.SetWestNeighbor(this->GetTjunction(WEST).GetNeighbor());

/* Update our neighbor’s topology
(except for new EAST quad). */
this->GetNeighbor(NORTH).SetSouthNeighbor(quadWest);
this->GetTjunction(WEST).GetNeighbor().SetEastNeighbor(quadWest);

/* Delete the WESTern T-junction.*/
this->DeleteTjunction(WEST);

RETURN quadWest;
```

Algorithm 8 Procedure: Quad::shearEastDivideWestWithNoT(Vertex vertexNorth, Vertex vertexCenter)
This procedure creates a new west quad when a sheared quad is identified. In this case, this quad has no WESTern T-junction.

```
Quad quadWest

/* Update the WEST quad's vertices. */
quadWest.SetNWvertex(this->GetNWvertex());
quadWest.SetNEvertex(vertexNorth);
quadWest.SetSEvertex(vertexCenter);
quadWest.SetSWvertex(Interpolate(this->GetNWvertex(), this->GetSWvertex());

/* Update the WEST quad's topology
   (except for new EAST quad).*/
quadWest.SetNorthNeighbor(this->GetNeighbor(NORTH));
quadWest.SetSouthNeighbor(this);
quadWest.SetWestNeighbor(this->GetNeighbor(WEST));

/* Create a new T-junction for our WEST neighbor. */
this->GetNeighbor(WEST).NewTjunction(EAST);
this->GetNeighbor(WEST).GetTjunction(EAST).SetVertex(quadWest.GetSWvertex());
this->GetNeighbor(WEST).GetTjunction(EAST).SetEastNeighbor(quadWest);

/* Update our neighbor's topology
(except for new EAST quad). */
this->GetNeighbor(NORTH).SetSouthNeighbor(quadWest);

RETURN quadWest;
```

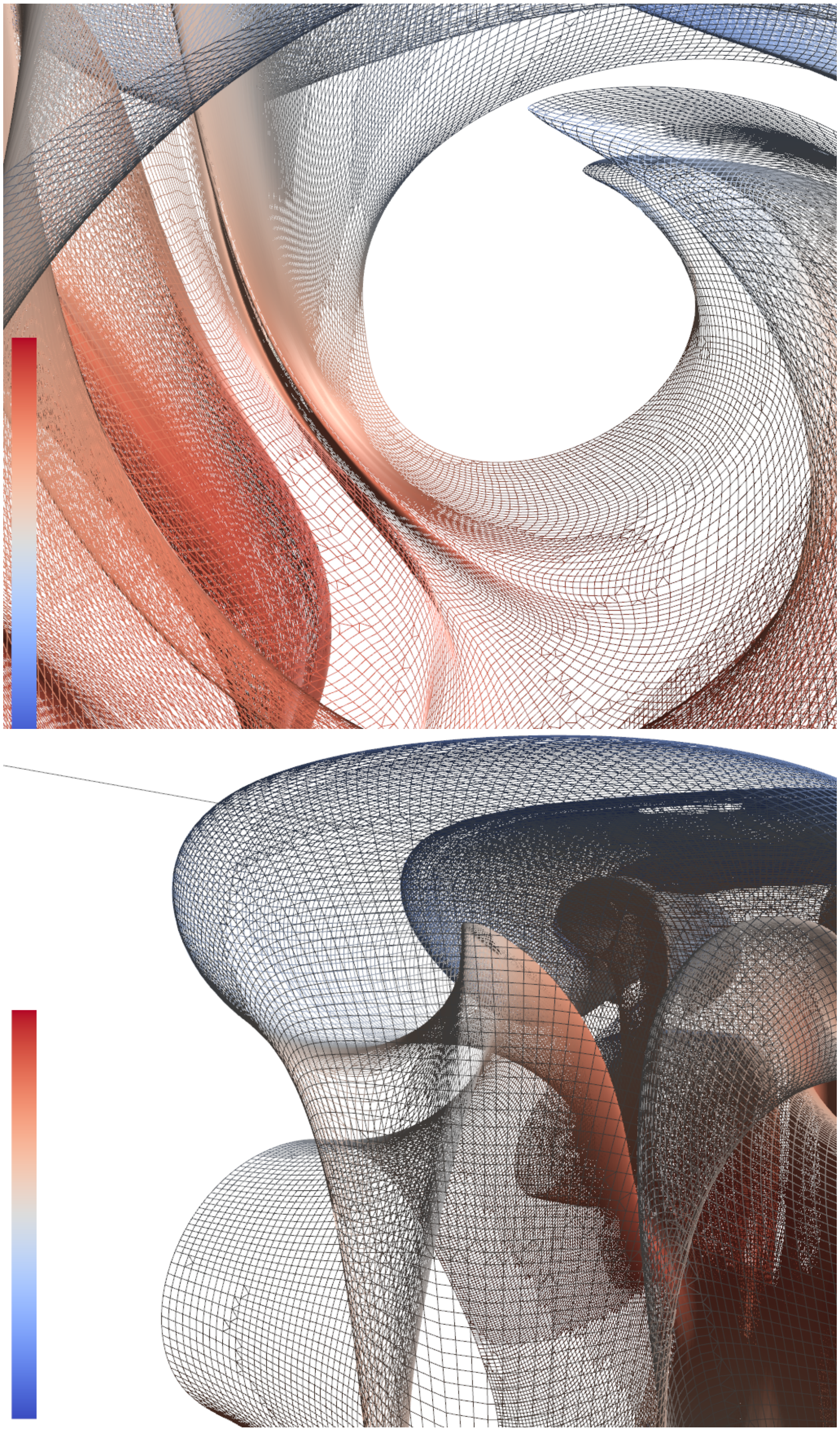
Figure 1: Even under strong deformation, the mesh remains well-structured and produces a smooth surface. This image also demonstrates the use of triangle fans to render quads that contain t-junctions.