- manuscript No.
(will be inserted by the editor)

Texture advection on discontinuous flows

Angel Rodríguez-Cerro · Ignacio García-Fernández · Rafael J. Martínez-Durá · Marta Pla-Castells

Abstract Texture advection techniques, which transport textures using a velocity field, are used to visualize the dynamics of a flow on a triangle mesh. Some flow phenomena lead to velocity fields with discontinuities that cause the deformation of the texture which is not properly controlled by these techniques. We propose a method to detect and visualize discontinuities on a flow, keeping consistent texture advection at both sides of the discontinuity. The method handles the possibility that the discontinuity travels across the domain of the flow with arbitrary velocity, estimating its speed with least squares approximation. The technique is tested with different sample scenarios and with two avalanche scenes, showing that it can run at interactive rates.

Keywords Flow visualization · Texture advection · Discontinuity · Computer animation

1 Introduction

Simulation and visualization of fluids are present in videogames, visual effects in feature films and computer animation. Computational fluid dynamics has become a common tool in the industry with growing level of realism and control [9,16]. In interactive applications, however, computation time is a key issue and high resolution fluid simulations are not always an option. When interactive frame rates are required, such as in virtual environments, the visualization of flow often relies on texture based approaches.

Texture advection techniques transport textures on a triangle mesh according to the velocity field of a flow. These techniques have been used to visualize a dynamic appearance on the surface of objects and specifically to visualize surface flow in substances such as water, lava, sand or fire. Nevertheless, as the texture coordinates are advected, the original texture appearance is lost due to its deformation. For this reason, different strategies have been proposed to control the level of texture deformation. In these techniques the vector field is supposed to be differentiable (or at least continuous).

However, some flow phenomena such as avalanches, some multiphase flows or even dense crowds of people, can show significant discontinuities in the velocity field. In these situations, two vertexes that are linked by an edge of the mesh but are separated by the discontinuity will have very different velocities. This shall cause an undesirable texture deformation, which is not properly handled by texture advection techniques that assume differentiability in the flow [17,18,27,28]. Figure 1, left, shows an example of the situation described; a texture is advected using the method described in [18] along a vector field that contains a horizontal discontinuity, undergoing noticeable deformation.

In this paper we present a technique that enables the visualization of flows with discontinuities in the spatial domain avoiding the distortion of the texture that appears in such regions. As inputs, our method takes a velocity vector field defined on an arbitrary triangle mesh and a texture, to produce an animation of the flow.

A. Rodríguez-Cerro · R.J. Martínez-Durá
LSyM. Universitat de Valencia, Valencia, Spain.
E-mail: angel.rodriguez@uv.es

I. García-Fernández
Departament d'Informatica. Universitat de Valencia, Valencia, Spain.

M. Pla-Castells
LISITT. Universitat de Valencia, Valencia, Spain.

Pre-print version. The final publication is available at Springer via `http://dx.doi.org/10.1007/s00371-015-1118-7`

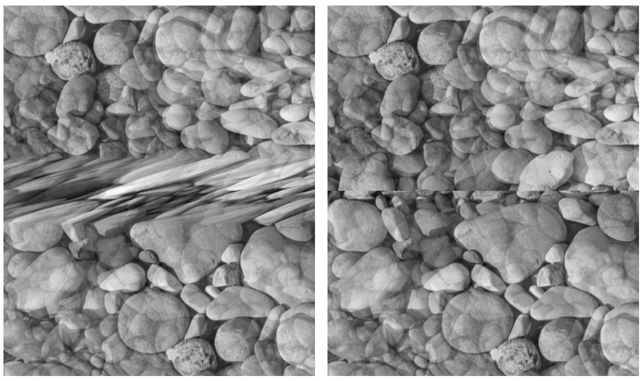
Fig. 1 Texture advection along a vector field with a discontinuity. Large velocity differences in near vertexes lead to texture deformation (left). With our technique the discontinuity is detected and remeshed to prevent texture deformation (right).

The main contribution of our paper is a technique for detecting and visualizing discontinuities in velocity fields. We are able of visualizing discontinuities that travel across the flow with arbitrary velocities (not necessarily advected with the flow). Our technique starts with the detection of a discontinuity by means of the evaluation of the vector field at the mesh vertexes. Following, we build an estimation of the region affected and we track its displacement. Then, in this region we modify the mesh in order to visualize the frontier that separates the regions with different velocities. Finally, we advect textures consistently at each side of the discontinuity by extrapolating texture coordinates. Figure 1, right, shows an example of the application of our technique to avoid the deformation observed in the left image.

The rest of this paper is structured as follows. In Section 2 we present an overview of other works related to flow visualization and deformation control. In Section 3 we present our approach to build a representation of a discontinuity in the vector field and how we determine its evolution. The remeshing process and the texture advection modification are described in Section 4. Section 5 presents the results of our work, including an analysis of several tests. Finally, Section 6 gives concluding remarks and future work.

2 Related work

Texture advection methods produce a visualization of a velocity field by means of the displacement of textures on a triangle mesh. These methods are used to animate flow by advecting either the texture coordinates on a fixed mesh or the triangle primitives. Laramee et al.[13] present an overview of flow visualization that reviews texture advection together with other methods.

van Wijk [26] proposes an Image Based Flow Visualization that advects dye in the texture space. Laramee et al. [14] address flow visualization of unsteady flow on surfaces. They use image space to overcome the problems that appear when working in physical space or in the surface parametrization space. Weiskopf and Ertl [25] combine image space and object space advection to visualize flow on arbitrary curved surfaces embedded in a three dimensional space.

Jobard et al. [10] add a particle system to the Eulerian advection to build a hybrid texture advection system. While the particle system is updated in a Lagrangian scheme, some of the particles' properties are updated in an Eulerian step using a texture. Rasmussen et al. [21] also use particle advection computed during the fluid simulation to determine the advection of rendering properties, including textures.

Yu et al. [27], use a Lagrangian approach to advect a particle system and incorporate a sprite texture on every particle. After blending the sprites, they achieve a river flow visualization with different physically based effects. The method is capable of showing discontinuities in the river sides which are defined at the beginning of the simulation as part of the input data. They later extend their method to improve conservation of texture properties by using a deformable grid that is carried by each particle [28].

The work by Kwatra et al. [12] advects a system of oriented elements for coherent texture synthesis. They visualize free surface flow generating the mesh and advecting fluid properties to synthesize the texture, based on sample texture images. They are able to handle avalanche like situations, such as a lava flow, but their method is aimed to offline rendering and is too expensive for interactive simulations. In a similar fashion, Bargteil et al. [1] generate a mesh that is tracked along time, allowing feature preservation.

In general, texture synthesis methods can be adequate for texture advection, as the synthesis can take into account time evolution [2,15]. In contrast, our approach focuses on visualization based on a predefined texture, but once discontinuity has been detected in the flow, it could be mapped into the texture generation process to obtain equivalent results without remeshing.

One of the main concerns when using techniques that advect an image along a vector field is the control of the texture deformation, and several authors have faced this problem. Max and Becker [17] derive a differentiable model for advection of texture coordinates, with a resetting strategy to avoid large texture deformations. They also propose transporting the triangle vertexes and a method to add or remove triangles in the domain boundaries when the flow causes the mesh to degenerate. Neyret [18] addresses deformation control by using a blend of several layers of texture. Using a different lifetime for every layer, he chooses the most appropriate layer according to a desired deformation level, defined by the user.

The different techniques that are applied to control texture deformation rely on the premise that flow is governed by a differentiable law and, thus, that the velocity field is continuous. For this reason they fail to keep texture deformation at low rates when discontinuities appear in the flow. In the next sections we address this situation and propose an strategy to visualize flows that are not continuous in certain regions of the domain.

Our technique is based on Eulerian texture advection strategies, based on a triangle mesh, as in the works by [17,18]. In the evaluation of our method we use Neyret's algorithm [18] for deformation control outside the discontinuities.

3 Detection and evolution of a discontinuity

We consider a vector field representing the velocity field of a flow, $\mathbf{v}(\mathbf{x})$ in a two dimensional domain. We use a visualization of the flow based on a texturized triangle mesh that covers its domain. We also assume that the texture is advected on that mesh following the velocity field using one of the texture advection methods described in the previous section.

We consider that the vector field contains a discontinuity and we assume that the points of discontinuity are the image of a differentiable curve on the flow domain. We shall work under the premise that we do not have an analytical description either of the shape of the discontinuity or of its evolution during the simulation. Moreover, the velocity of the discontinuity does not need to have any relationship with the vector field.

For every vertex, i, we shall consider its location, $\mathbf{x}_i$, and its texture coordinates, $\mathbf{c}_i = (u_i, v_i)$. Let i, j be the indexes of two vertexes in the triangle mesh connected by an edge, $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ will be the vector that connects them. In addition, for every vertex we have the value of the velocity field at its location, $\mathbf{v}_i = \mathbf{v}(\mathbf{x}_i)$.

In this section we describe how to detect the existence of the discontinuity by inspection of the vector field at the mesh vertexes and how to track its evolution. First we detect discontinuities associated to edges by comparing the velocity of adjacent vertexes. As an starting guess, we assume that the discontinuity does not travel across the domain. If, later, we detect that the discontinuity has left the initial edge and has moved to a neighbour one, we estimate the travel speed of the discontinuity and compute its evolution.

3.1 Description of the discontinuity

In order to build a representation of the discontinuity, we shall need to detect the points where it intersects the edges of the triangle mesh. We consider that a discontinuity of the vector field crosses edge ij if the directional derivative of $\mathbf{v}(\mathbf{x})$ along the edge $\mathbf{x}_{ij}$, estimated by the first order difference, is higher than a user defined threshold M

$$\nabla_{\mathbf{x}_{ij}} \mathbf{v}_i = \frac{\mathbf{v}_j - \mathbf{v}_i}{\|\mathbf{x}_{ij}\|} \geq M. \tag{1}$$

In that case, we add a discontinuity point $\mathcal{D}_{ij}$ to the list of intersections between the discontinuity and the triangle mesh.

For every discontinuity point we shall store information about its location and about its velocity. The intersection location will be described with a parameter $f_{ij} \in [0,1]$ that indicates the normalized distance from vertex $\mathbf{x}_i$ to the intersection along the edge, so that the intersection is located at point $\mathbf{d}_{ij}$ as

$$\mathbf{d}_{ij} = \mathbf{x}_i + f_{ij}\mathbf{x}_{ij}.$$

We assume that the discontinuity can travel along the flow domain and that a discontinuity point $\mathcal{D}_{ij}$ has a velocity $\mathbf{w}_{ij}$. This velocity does not need to have any relationship with the velocity field defined on the domain. Indeed, the fact that there is a discontinuity makes that the velocity field is not defined at that point. We want to approximate the velocity of the discontinuity point, $\mathbf{d}_{ij}$, along the edge $\mathbf{x}_{ij}$. To do this, we linearize the curve at $\mathbf{d}_{ij}$, approximating it by its tangent. We are interested on the velocity of the discontinuity only on the direction orthogonal to the curve, as the velocity in the tangential direction will not change the location of the intersection. Abusing of notation we will call the orthogonal velocity of the curve $\mathbf{w}_{ij}$.

Under the previous assumptions, we define the velocity of the intersection along the direction of the edge, w_{ij}, as follows. Let $s_{ij} = \|\mathbf{w}_{ij}\|$ be the modulus of the velocity, and let $\mathbf{s}_{ij} = \frac{\mathbf{w}_{ij}}{s_{ij}}$ and $\mathbf{u}_{ij} = \frac{\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|}$ be unitary vectors in the direction of the velocity and of the edge respectively. Then, the projection of vector $w_{ij}\mathbf{u}_{ij}$ onto the direction of $\mathbf{s}_{ij}$ has to be s_{ij}. That is

$$w_{ij}\mathbf{u}_{ij} \cdot \mathbf{s}_{ij} = s_{ij} \tag{2}$$

Figure 2 shows the variables that are used to describe a discontinuity point. In addition to this information, the time at which the discontinuity was created,

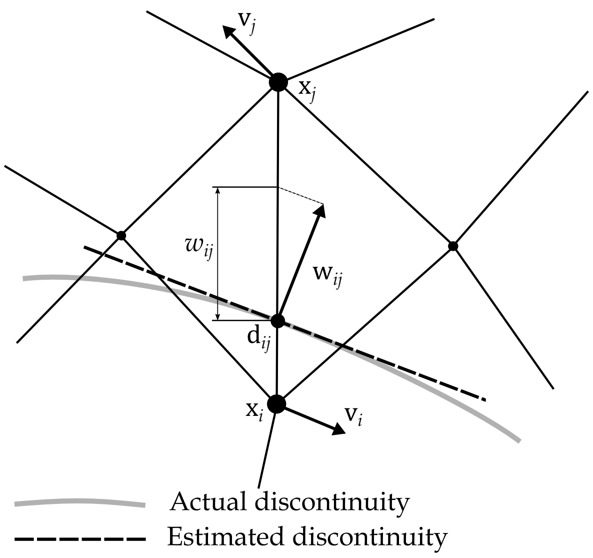

Fig. 2 Description of a discontinuity point and the variables involved.

t_{ij}, is also registered, and will be used later to update the estimation of w_{ij}.

All the variables related to $\mathcal{D}_{ij}$ have been defined considering that the edge is oriented, going from vertex i to vertex j. If we consider the inverse situation (going from vertex j to vertex i), then the corresponding quantities are $f_{ij} = 1 - f_{ji}$ and $w_{ij} = -w_{ji}$.

3.2 Evolution of discontinuity points

Next we describe our approach to track the displacement of a discontinuity point $\mathcal{D}_{ij}$ due to its velocity $\mathbf{w}_{ij}$. We need to determine the evolution of the variables f_{ij} and w_{ij}. When we detect a discontinuity point $\mathcal{D}_{ij}$, we store all its data while it remains active, that is, while condition (1) holds for the pair (i, j). During this time, we consider that w_{ij} is constant. Thus, every frame we update the location of the discontinuity point as

$$f_{ij}^{t+\Delta t} = f_{ij}^{t} + w_{ij}\Delta t$$

where superscript indicates time and Δt is the time between to consecutive frames.

If the discontinuity is actually moving with velocity $w_{ij} \neq 0$ along the direction $\mathbf{u}_{ij}$, then it will eventually reach one of the vertexes, say vertex j. When this happens, both vertexes i and j will be at the same side of the discontinuity and $\mathcal{D}_{ij}$ will no longer exist. This situation will be detected because equation (1) will not be met.

Moreover, the discontinuity will potentially intersect other edges $(j, \beta), \beta = k_1, \ldots, k_r$ that incide on vertex j. If this is the case, and some new discontinuities appear, then we need to estimate the velocity of the discontinuity when it crossed vertex j in order to compute the different $w_{j\beta}$. Figure 3 shows a possible transition of two discontinuity points into a new one across a vertex.

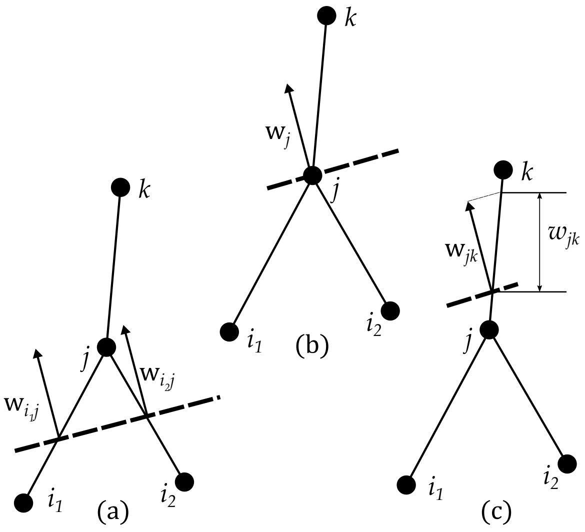

Fig. 3 Evolution of a discontinuity across a vertex. If a discontinuity has constant velocity, the associated points will eventually reach the end of their edges (a). Then, the old discontinuity points disappear (b), and new ones appear in the edges across the vertex (c).

Thus, let us consider the situation in which we had a set of n discontinuity points $\mathcal{D}_{\alpha j}, \alpha = i_1, \ldots, i_n$ that converge in vertex j at an instant of time t. The first step to estimate the velocity of the discontinuity when it crossed vertex j is to correct the values of $w_{\alpha j}$ by computing the actual average velocities as

$$\bar{w}_{\alpha j} = \frac{\|\mathbf{x}_{\alpha j}\|}{t - t_{\alpha j}} \tag{3}$$

where $t_{\alpha j}$ is the time at which $\mathcal{D}_{\alpha j}$ was created.

After the longitudinal velocities have been updated, the velocity of the discontinuity, $\mathbf{w}_j = \mathbf{s}_j s_j$, is estimated by least squares approximation. In order to state the least squares problem, we use equation (2), that defines the relationship between $\mathbf{w}_j$ and the different $\bar{w}_{\alpha j}$,

$$\bar{w}_{\alpha j}\mathbf{u}_{\alpha j} \cdot \mathbf{s}_j = s_j, \ \alpha = i_1, \ldots, i_n, \tag{4}$$

where $\mathbf{s}_j$ and s_j are the unknowns. If we divide (4) by s_j, and write $\mathbf{r}_j = \mathbf{s}_j / s_j$ then

$$\bar{w}_{\alpha j}\mathbf{u}_{\alpha j} \cdot \mathbf{r}_j = 1, \ \alpha = i_1, \ldots, i_n, \tag{5}$$

We can write the system of equations (5) in matrix form as $\mathbf{W} \cdot \mathbf{r}_j = \mathbf{1}$ where

$$\mathbf{W} = \begin{bmatrix} w_{i_1 j}\mathbf{u}_{i_1 j} \\ \vdots \\ w_{i_n j}\mathbf{u}_{i_n j} \end{bmatrix}; \qquad \mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{6}$$

If $n > 2$, that is, more than two discontinuity points converge in vertex j, then the system is overdetermined and we need to solve a least squares problem, as it has been stated earlier. If $n = 2$, then (5) is a 2×2 system of equations. Finally, if $n = 1$, then we do not have enough information to decide the direction and magnitude of $\mathbf{w}_j$. In that case we pick $\mathbf{w}_j = \bar{w}_{i_1 j}\mathbf{u}_{i_1 j}$, which accomplishes (5) trivially and is consistent with the information we have from the vanishing discontinuity point. Figure 4 represents the vectors involved in the least squares problem.

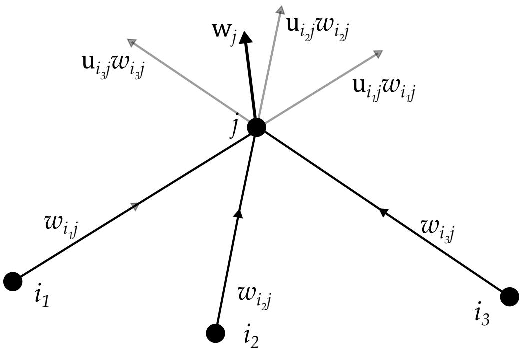
$\mathbf{w}_j$
$\mathbf{u}_{i_2j}w_{i_2j}$
$\mathbf{u}_{i_3j}w_{i_3j}$
$\mathbf{u}_{i_1j}w_{i_1j}$
j
w_{i_1j}
w_{i_3j}
w_{i_2j}
i_1
i_3
i_2

Fig. 4 When discontinuities cross a vertex, the new velocity is computed using the information of their previous estimated velocities by least squares approximation.

Once we have determined the value of $\mathbf{w}_j$ from (5) we initialize the velocity of all the edges (j, β) that have a discontinuity. Using (2) and the definition of $\mathbf{r}_j$ we set

$$w_{j\beta} = \frac{1}{\mathbf{r}_j \cdot \mathbf{x}_{j\beta}}, \ \beta = k_1, \ldots, k_r. \tag{7}$$

The procedure proposed to update w_{jk} depends on the existence of a prior discontinuity (i, j) that extinguishes when (j, k) appears. However, when a discontinuity point $\mathcal{D}_{jk}$ is detected for the first time by checking equation (1), we have no previous information about the exact intersection point or the velocity of the discontinuity. As an starting guess, we consider that the point is in the middle of the edge between vertexes j and k, and that it has zero velocity, by taking $f_{jk} = 1/2$ and $w_{jk} = 0$.

3.3 Issues on velocity estimation

The estimation of $\mathbf{w}_{ij}$ has been considered to be constant in the time lapse while the discontinuity remains in edge (i, j). Next we discuss how this assumption affects the estimation of the discontinuity.

If, in average, the actual velocity of the discontinuity point $\mathcal{D}_{ij}$ is higher than the estimated velocity $\mathbf{w}_{ij}$, then the estimated location $\mathbf{d}_{ij}$ will not reach $\mathbf{x}_j$. Before this happens, the actual discontinuity will cross the vertex and the discontinuity of edge (i, j) will disappear. This situation causes an undesired jump of the location $\mathbf{d}_{ij}$. This problem, however, is not difficult to overcome. The old discontinuity can be kept active until $\mathbf{d}_{ij} = \mathbf{x}_j$. The velocity would be increased by a factor to reduce the error.

If, on the contrary, the actual velocity of the discontinuity point $\mathcal{D}_{ij}$ is smaller than $\mathbf{w}_{ij}$, then the location $\mathbf{d}_{ij}$ will reach $\mathbf{x}_j$ before the actual discontinuity. When this happens the discontinuity point is retained at vertex $\mathbf{x}_j$ until the discontinuity actually reaches it. Then, the velocity is corrected applying equation (3).

A particular case of the previous situation is whenever the discontinuity stops before reaching vertex. In this case, the location of the estimated discontinuity would not be exact, but the visualization would not suffer from inconsistencies and the qualitative behaviour would be correct.

The last possibility that has not been considered so far is whenever the actual discontinuity changes its direction and exits the edge through the same vertex it entered. Visually, this is similar to the case when the actual velocity is higher than the estimated; it leads to a jump of $\mathbf{d}_{ij}$, which goes back to $\mathbf{x}_i$ when it is half way to the opposite vertex. Again, the issue can be overcome with the same strategy proposed above. But this case needs an additional consideration. If we apply equation (3) directly then we have $\bar{w}_{ij} = 0$, which might underestimate again the right value leading to a new jump in the next vertex. Our proposal is to consider that the point has moved under a constant negative acceleration. In this case, the exit velocity would be

$$\bar{w}_{ij} = -w_{ij}. \tag{8}$$

4 Remeshing of the triangles affected by the discontinuity

Using the list of discontinuity points built in the previous section, we now proceed to modify the mesh in order to avoid the texture deformation in the triangles affected by the discontinuity. During the process of rendering the triangle mesh, we check every triangle.

Then, if it has a discontinuity crossing any of its sides, the triangle is split into several subtriangles.

Depending on the number of sides of the triangle that are affected by a discontinuity we propose three different remeshing schemes, which are shown in Figure 5. The situation where a triangle involves three discontinuities is unlikely if we have a single discontinuity without intersections. However, this situation can happen when two discontinuities cross inside a triangle.

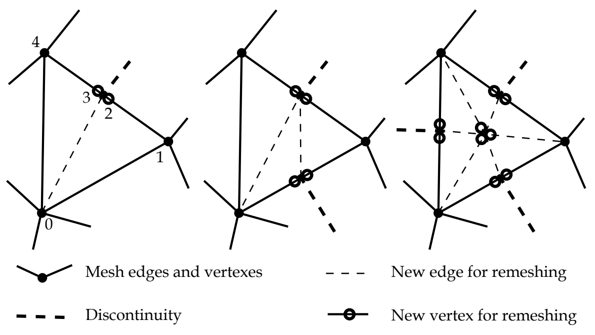

Fig. 5 When a triangle has edges with discontinuities it is remeshed. Depending on the number of discontinuities detected, we use one of the schemes of the figure. In the vertex that belong to the frontier we need to add double and triple vertexes to assign different texture coordinates to the same vertex in different triangles.

In all the discontinuity locations, $\mathbf{d}_{ij}$, two vertexes are introduced. By doing this, the triangle that includes the subedge from $\mathbf{x}_i$ to $\mathbf{d}_{ij}$ should have a different texture coordinate than the triangle that includes the subedge from $\mathbf{d}_{ij}$ to $\mathbf{x}_j$. In Figure 5 the points that contain a double vertex are indicated with a double dot. As an example, in the scheme for a single discontinuity we subdivide the original triangle in the three subtriangles 012, 023 and 034. Notice that the second triangle is degenerate and helps creating the visual discontinuity in the texture.

We need to assign a texture coordinate value to the new vertexes located at the discontinuity points, depending on their side of the discontinuity. However, texture coordinate is not defined in such points, as it is only stored on the vertexes of the original mesh. For this reason we need to estimate a value of the texture coordinate.

We propose to use linear extrapolation using the Jacobian of the texture coordinate at the vertexes of the triangle [4], estimated with the adjacent vertexes not affected by a discontinuity. Let $\mathbf{c}_i = (u_i, v_i)$ be the texture coordinates at vertex i. We develop the estimation for texture coordinate u and the development for coordinate v is completely analogous. If we consider the texture coordinate as a differentiable function of location then, by its Taylor expansion at $\mathbf{x}_i$, we have that the texture coordinate for a point $\mathbf{d}_i$ can be approximated as

$$u(\mathbf{d}_i) \simeq u(\mathbf{x}_i) + \nabla_{\mathbf{x}} u(\mathbf{x}_i) \cdot (\mathbf{d}_i - \mathbf{x}_i). \tag{9}$$

In order to estimate $\nabla_{\mathbf{x}} u(\mathbf{x}_i)$ we apply (9) to any vertex adjacent to $\mathbf{x}_i$ that is not separate by a discontinuity. That is, given $\mathbf{x}_{k_1}, \ldots, \mathbf{x}_{k_r}$ vertexes neighbour to $\mathbf{x}_i$ such that the edge (i, k_l) is not intersected by a discontinuity (i.e., it does not accomplish (1)), we have

$$u(\mathbf{x}_{k_l}) = u(\mathbf{x}_i) + \nabla_{\mathbf{x}} u(\mathbf{x}_i) \cdot \mathbf{x}_{k_l i}, \ l = 1, \ldots, r \tag{10}$$

which defines a system of linear equations on the unknown $\nabla_{\mathbf{x}} u(\mathbf{x}_i)$. Again, we use least squares approximation if the system is overdetermined.

It can happen that we have no neighbour vertexes available to estimate the gradient. In this case, the vertex is completely isolated by discontinuities. If we have an estimation of the gradient from a prior step we keep it. On the contrary we use a precomputed gradient that uses the undeformed texture coordinates.

If we are using a texture advection technique that involves blending of several texture layers, such as the method by Neyret [18], then we need to extrapolate the value for each layer. However the gradient only needs to be computed once, as textures of different layers are typically shifted a fixed amount.

5 Results

We have implemented a series of test scenarios with different discontinuity configurations and different velocity fields for the flow. The first two scenarios include two opposed velocity fields. In the first one a horizontal discontinuity separates both flows and in the second one the opposed flows are separated by a diagonal discontinuity that moves with varying velocity. A third scenario is composed by four vector fields pointing in the four main directions; up, down, left and right. Two discontinuities cross in the center of the domain, creating four quadrants that host the four velocity fields. The discontinuities spin around the cross, forming a pinwheel.

As all previous tests only consider straight discontinuities, we have also included a test with a curved frontier. The fourth test scenario is a circle moving around the center of the flow domain. Inside the circle the velocity field corresponds to the velocity of the circle itself and outside the circle the velocity field points downwards with constant velocity. Figure 6 shows the four velocity fields.

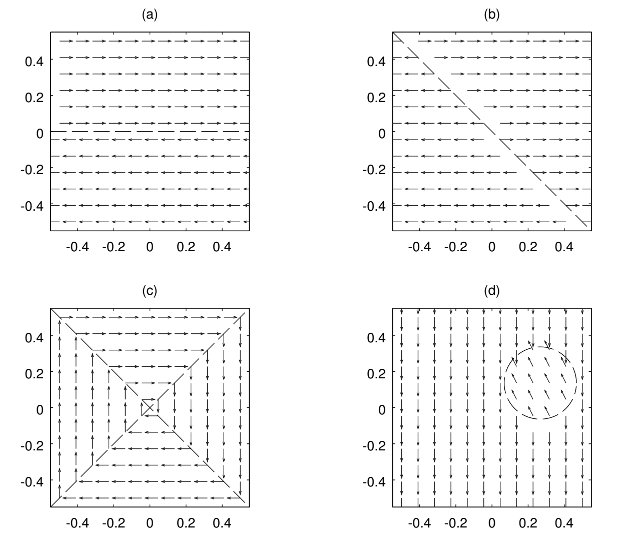

Fig. 6 The four test scenarios that have been used to evaluate the method proposed in this paper. The discontinuities are represented by dashed lines.

We have used a checkerboard texture to show our results. In all the tests, texture advection has been implemented using the blending approach proposed by Neyret [18]. For this reason, in the images the texture appears blended with itself with different transparency values. This blending is used to control texture deformation in many texture advection works.

The first test has been already shown in Figure 1 with a texture of stones. This test is somehow ideal in the sense that all the assumptions we take when the discontinuity is detected are met; the discontinuity is a straight line and it does not move. For this reason the discontinuity is detected exactly. The result can be seen in Figure 7.

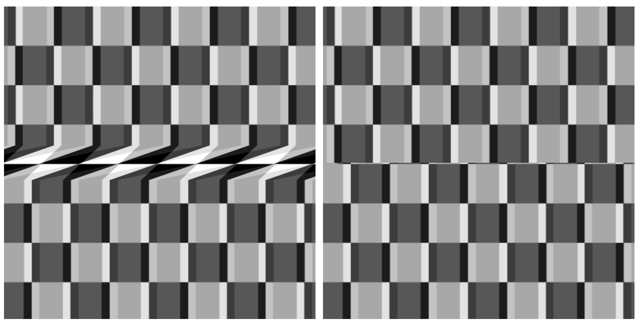

Fig. 7 Texture advection on a domain with two opposed vector fields separated by a horizontal discontinuity, without discontinuity detection (left) and with our discontinuity detection method (right).

In the second test we show that the remeshing algorithm reproduces oblique lines properly. In this test

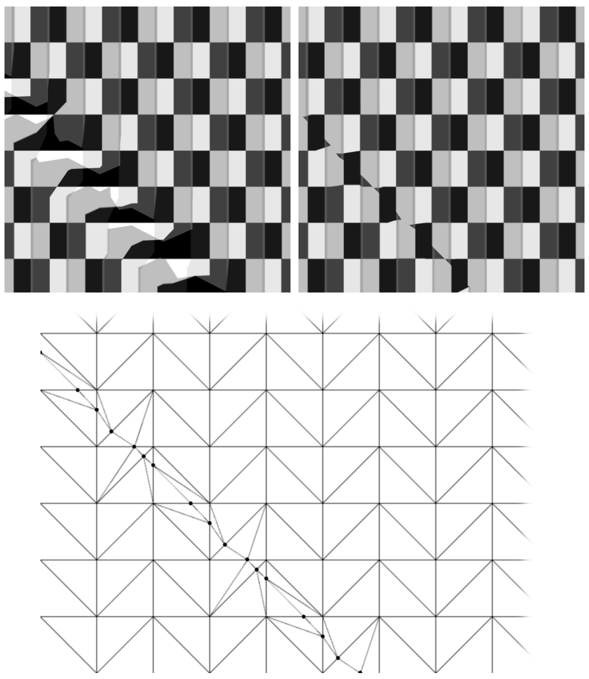

Fig. 8 Texture advection on a domain with two opposed vector fields separated by a diagonal discontinuity, without discontinuity detection (left) and with our discontinuity detection method (right). The remeshing scheme is shown in the bottom image, together with the estimated discontinuity points.

the discontinuity is moving with a velocity that follows a sinus function. For this reason it is continuously moving up and down, and changing its velocity. In this test the discontinuity jumps described in Section 3.3 can be observed, since we have not implemented in the tests any correction to prevent them. Such jumps are most noticeable when the actual discontinuity stops and changes the sign of its movement. The result, compared to standard texture advection, can be seen in Figure 8. In this Figure, a detail of the modified triangle mesh with estimated discontinuity points has been also included.

In the pinwheel test we can observe the behaviour of the method in a location with two discontinuities crossing (Figure 9). A remarkable property of this scenario is the fact that the farther a discontinuity point is from the center, the higher is its velocity. Moreover, as the discontinuities rotate, they change the direction of their velocity along time. These two properties make the quality of the velocity estimation to depends on the angle between an edge and the discontinuity. This produces the appearance of small corners in the approximation of the frontier between the different flows. This artifact, however, is corrected whenever the discontinuity reaches a vertex and is barely noticeable for dense meshes. The method we have proposed assumes that the discontinuity can be approximated locally with a single straight line. For this reason, the remeshing around the center is not always consistent with the underlying vector field. However, it is a very local effect and the overall behavior is considered positive.

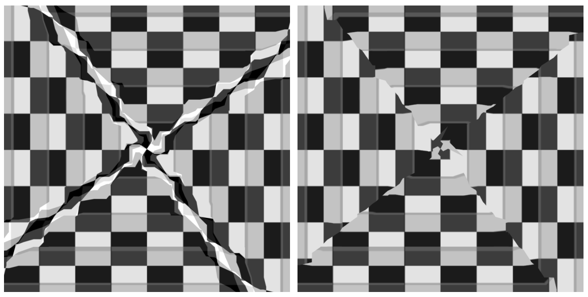
Fig. 9 Four vector fields divided by two crossing discontinuities. The discontinuities spin around the crossing point. Left image without discontinuity detection and righ image with discontinuity detection.

The travelling circumference test poses a more challenging scenario for the edge detection step. In this test the curvature of the discontinuity causes a larger estimation error in the local velocities. This makes more noticeable the appearance of small peaks in the frontier, already described in the pinwheel test. However, these effects reduce as the mesh resolution increases. Another cause for the appearance of this effect is a discontinuity almost parallel to an edge, which could be missed when crossing a relatively long edge. This situation can be overcome if angles close to 180° are avoided in the triangle mesh. Figure 10 shows the results of this test with a resolution of 160×160 cells, where peaks are barely noticeable. The texture in this demo has been generated with Perlin noise.

Besides the test scenarios, we have also implemented two demonstration examples to show the application of our method to real flows; we have reproduced two avalanche scenarios. Experimental studies show that avalanches in granular systems happen when a thin layer of rolling grains slide over a main body of static material. This process can be represented as a vector field defined on the surface of the granular system [8]. As avalanches usually affect bounded regions of a slope, the boundary of the avalanche will show a discontinuity in the vector field, which is zero outside that region [6, 5]. Avalanche animation an visualization has been addressed in the past [23, 19, 20, 24] with some authors reporting the use of texture advection.

We represent a landform with a regular triangle mesh, and simulate the evolution of an avalanche with the BCRE model [3]. This model uses a system of two

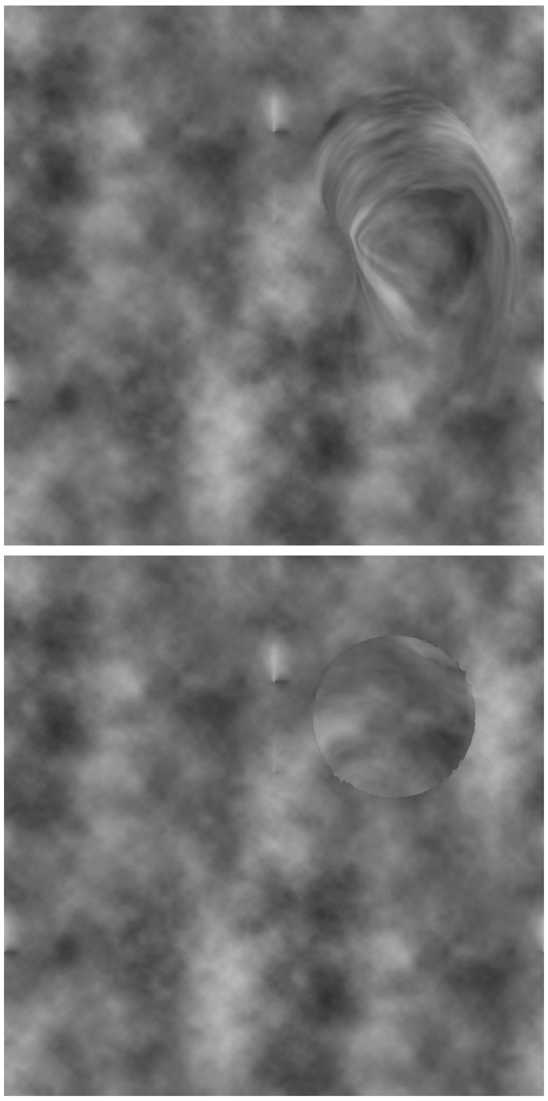
Fig. 10 A circle moving around the center of the domain, visualized with a perlin noise texture. The velocity field inside the ball coincides with the velocity of the ball and the velocity field in the background moves downwards. The top image shows advection without discontinuity detection and the bottom image with discontinuity detection.

partial differential equations to describe the evolution of the layer of rolling material. The discretization of the model on the mesh has been done using the Lax-Friedrichs finite difference scheme [22]. The first avalanche simulation is a landslide on an inclined plane. We cause the avalanche by adding material and let it evolve down the slope. The second scenario is a snow avalanche on a valley. Figures 11 and 12 show several images obtained during the simulation. In these scenarios we have included a bump-mapping texture that has been also advected with the flow when necessary. The discontinuity has been highlighted visually by shading some of the added vertexes. This kind of enhancements can be done without an additional cost, as all the vertex properties can be accessed during remeshing.

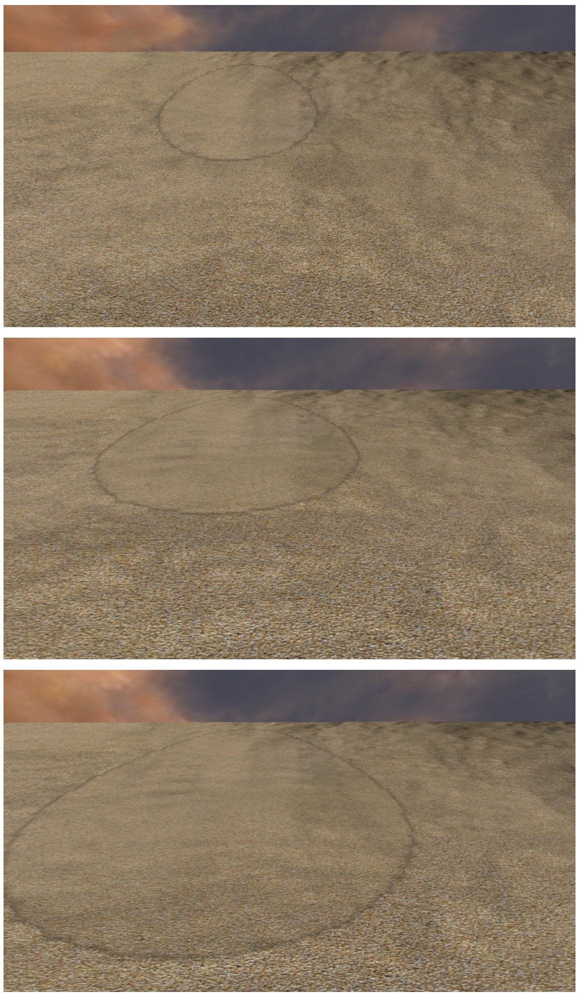
Fig. 11 Three images of the evolution of a landslide down a slope.

The technique can be applied to other phenomena, such as dense crowds [11]. In dense crowds, neighbouring regions can host groups of people walking in opposite directions. The method proposed here can be used to visualize these situations, with an appropriate texture blending in the frontier. The advection of properties other than texture coordinates, such as a normal map or a bump mapping texture, can also be used to synthesize the flow of droplets on surfaces [7].

5.1 Parallel computation and performance

The technique we propose is aimed to the visualization of flow in interactive environments, where performance is relevant to achieve the necessary frame rates for a good user experience. The methodology proposed is split into several steps that can be easily implemented for its execution in parallel and, more precisely, on the GPU. In our implementation, the remeshing algorithm has been implemented using a shader. This task is performed independently for every triangle in the GPU. The rest of the algorithm is computed on the CPU, including the solution of the two least squares problems proposed in Sections 3 and 4, but they could be computed in parallel for every vertex.

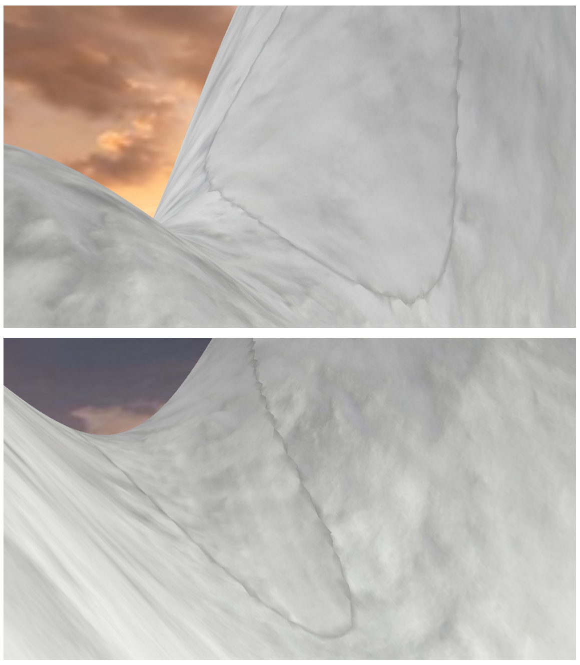
Fig. 12 Two images of the evolution of a snow avalanche down a valley.

During the tests we have registered the frame rates achieved to show that the method proposed can run at interactive rates. The different tests have been run on an Intel® Core™i7-3770K CPU @ 3.50GHz 3.90GHz, with 16GB RAM and an NVidia GForce GTX 660. The sample scenarios used to test the methodology have been implemented and tested with grid sizes between 10×10 and 30×30 divisions on the grid. In all these tests the frame rate was over 900fps.

The two avalanche scenarios have been computed at higher resolutions. The landslide on the slope has been simulated with a grid of 80×80 squares and it ran at around 25fps. In this scenario, the solution of the partial differential equation to get the velocity field took about a 10% of the computation time every frame. The snow avalanche has been simulated with a grid of 160×160 squares and it ran at 7fps. The solution of the partial differential equation took about the 14% of the computation time every frame. In this scenario, the computation of the new velocities and textures, including the solution of the least squares problems, took between a 25% and a 50% of the computation time in the frames when the avalanche is most active. Thus, a parallel execution of the discontinuity tracking process on the GPU is very likely to yield better results.

6 Conclusion and future work

We have presented a method for texture advection visualization that addresses the problem of texture deformation in flow discontinuities. We have proposed an algorithm to detect a discontinuity and track its evolution using only the information available at the vertexes of the triangle mesh. When a discontinuity is detected, the mesh is modified to provide a sharp representation of the frontier with consistent texture advection. The method has been derived considering that there is no information about the flow outside the grid nodes. However, the remeshing scheme can be improved if the velocity field is available in more locations, allowing further remeshing of the areas affected by a discontinuity.

Some issues have been observed in the form of small irregularities when the velocity of the frontier is not constant. Although dense meshes overcome the problem, some effort needs to be done to make the representation of the discontinuity more robust to these artifacts. There is still work in progress to extend parallelism to parts that now are run in the CPU. Moreover, a detailed study of the performance is yet to be done, in order to analyse the cost of every part of the algorithm and its growth when the number of vertexes increases.

Acknowledgements This research has been partially supported by grant GV/2012/007, from Generalitat Valenciana, Valencia, Spain.

References

1. Bargteil, A.W., Goktekin, T.G., O'Brien, J.F., Strain, J.A.: A semi-lagrangian contouring method for fluid simulation. ACM Trans. Graph. **25**(1), 19–38 (2006)
2. Bargteil, A.W., Sin, F., Michaels, J.E., Goktekin, T.G., O'Brien, J.F.: A texture synthesis method for liquid animations. In: SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 345–351 (2006)
3. Bouchaud, J.P., Cates, M.E., Ravi-Prakash, J., Edwards, S.F.: A model for the dynamics of sandpile surfaces. Journal de Physique I France **4**, 1383–1410 (1994)
4. Correa, C.D., Hero, R., Ma, K.L.: A comparison of gradient estimation methods for volume rendering on unstructured meshes. IEEE Trans. Vis. Comput. Graph. **17**(3), 305–319 (2011)
5. Daerr, A.: Dynamical equilibrium of avalanches on a rough plane. Physics of Fluids **13**, 2115–2124 (2001)
6. Daerr, A., Douady, S.: Two types of avalanche behaviour in granular media. Nature **399**, 241–243 (1999)
7. El Hajjar, J.F., Jolivet, V., Ghazanfarpour, D., Pueyo, X.: A model for real-time on-surface flows. The Visual Computer **25**(2), 87–100 (2009)
8. Hadeler, K.P., Kuttler, C.: Dynamical models for granular matter. Granular Matter **2**(1), 9–18 (1999)
9. Ihmsen, M., Akinci, N., Akinci, G., Teschner, M.: Unified spray, foam and air bubbles for particle-based fluids. The Visual Computer **28**(6-8), 669–677 (2012)
10. Jobard, B., Erlebacher, G., Hussaini, M.Y.: Lagrangian-eulerian advection for unsteady flow visualization. In: Proceedings of the Conference on Visualization '01, VIS '01, pp. 53–60. IEEE Computer Society (2001)
11. Kim, S., Guy, S., Hillesland, K., Zafar, B., Gutub, A.A., Manocha, D.: Velocity-based modeling of physical interactions in dense crowds. The Visual Computer pp. 1–15 (2014)
12. Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N., Carlson, M., Lin, M.C.: Texturing fluids. IEEE Transactions on Visualization and Computer Graphics **13**(5), 939–952 (2007)
13. Laramee, R.S., Hauser, H., Doleisch, H., Vrolijk, B., Post, F.H., Weiskopf, D.: The state of the art in flow visualization: Dense and texture-based techniques. Computer Graphics Forum **23**(2), 203–221 (2004)
14. Laramee, R.S., Jobard, B., Hauser, H.: Image space based visualization of unsteady flow on surfaces. In: Proceedings of the 14th IEEE Visualization 2003 (VIS'03), p. 18. IEEE Computer Society (2003)
15. Lefebvre, S., Hoppe, H.: Appearance-space texture synthesis. In: ACM SIGGRAPH 2006 Papers, pp. 541–548 (2006)
16. Lever, J., Komura, T.: Real-time controllable fire using textured forces. The Visual Computer **28**(6-8), 691–700 (2012)
17. Max, N., Becker, B.: Flow visualization using moving textures (1996)
18. Neyret, F.: Advected textures. In: In ACM SIGGRAPH/Eurographics symposium on Computer animation (2003), pp. 147–153 (2003)
19. Onoue, K., Nishita, T.: An interactive deformation system for granular material. Computer Graphics Forum **24**(1), 51–60 (2005)
20. Pla-Castells, M., García-Fernandez, I., Martinez-Dura, R.J.: Physically-based interactive sand simulation. In: Eurographics 2008 - Short Papers, pp. 21–24 (2008)
21. Rasmussen, N., Enright, D., Nguyen, D., Marino, S., Sumner, N., Geiger, W., Hoon, S., Fedkiw, R.: Directable photorealistic liquids. In: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 193–202 (2004)
22. Strikwerda, J.: Finite Difference Schemes and Partial Differential Equations, 2 edn. SIAM: Society for Industrial and Applied Mathematics (2004)
23. Sumner, R.W., O'Brien, J.F., Hodgins, J.K.: Animating sand, mud and snow. Computer Graphics Forum **18**(1), 17–26 (1999)
24. Tsuda, Y., Yue, Y., Dobashi, Y., Nishita, T.: Visual simulation of mixed-motion avalanches with interactions between snow layers. The Visual Computer **26**(6-8), 883–891 (2010)
25. Weiskopf, D., Ertl, T.: A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In: Proceedings of Graphics Interface 2004, GI '04, pp. 263–270 (2004)

26. van Wijk, J.J.: Image based flow visualization. ACM Trans. Graph. **21**, 745–754 (2002)
27. Yu, Q., Neyret, F., Bruneton, E., Holzschuch, N.: Scalable real-time animation of rivers. Computer Graphics Forum (Proceedings of Eurographics 2009) **28**(2) (2009)
28. Yu, Q., Neyret, F., Bruneton, E., Holzschuch, N.: Lagrangian texture advection: Preserving both spectrum and velocity field. IEEE Transactions on Visualization and Computer Graphics **17**(11), 1612–1623 (2011)