Hatching for Metaball Surfaces

Ferenc Tükör
Supervised by: László Szécsi[*]

Institute of Computer Graphics
Budapest University of Technology and Economics
Budapest / Hungary

Abstract

This paper presents a highly parallel algorithm for the stylized, real-time display of fluids and smoke. We use metaballs to define a fluid surface from a particle-based fluid representation, but instead of the costly complete reconstruction of this surface, we only trace the motion of random seed points on it. Hatching strokes are extruded along the lines of curvature. We propose methods for hidden stroke removal and density control that maintain animation consistency.

Keywords: Hatching, NPR, Metaball

1 Introduction

Hatching is an artistic technique that is often emulated in stylistic animation. Implicit surfaces are becoming increasingly important in real-time applications for visualizing fluids simulated by particle-based methods. Thus, we aim to extend real-time hatching to deforming implicit surfaces, and specifically to metaballs. In addition to enabling fluid rendering in hatching-style NPR, we also consider this approach a more feasible alternative to expensive polygonization [12, 18], ray casting [15, 3], or screen-space filtering [20], when visualizing scientific isosurface or fluid simulation data.

2 Previous work

In pencil drawings artists convey the shape and illumination of objects with the density and orientation of thin hatch lines [23, 7]. To mimic this, we should define a *density* and a *direction field* in the image plane that is as close as possible to what an artist would use. Density is influenced by illumination, while the direction field is determined either by the *principal curvature directions* [6], the tone gradient [10], or in case of animation, the direction of motion.

Hatch strokes should appear hand-drawn, with roughly similar image-space width, dictated by brush size, but they should also stick to surfaces to provide proper object space shape and motion cues. Hatches can be generated into textures and mapped onto animated objects, with level-of-detail mechanisms to approximate image space behavior [17]. In absence of surface parametrization, this approach is not applicable to implicit surfaces.

Hatch strokes can be generated directly in image space [9, 11], but if they are fixed in their position and cling to the view plane instead of the animated objects, movement will appear as if seen through textured glass. This is known as the *shower door effect*. In order to avoid this disturbing phenomenon, lines can be moved along with an optical flow or image space velocity field, but placing new strokes on emerging, previously non-visible surfaces still poses problems. Especially if strokes are long, following curvature or feature curves of object surfaces, they should appear consistent even when only tiny fractions have become visible. This cannot be assured when only using image space information. For implicit surfaces, it is often prohibitively expensive to render a full image, or even to find isosurfaces in some pixels.

Several works [13, 19] proposed the application of *particles* or *seeds* attached to objects, extruding them to hatch strokes in image space. The key challenge in these methods is the generation of the world-space seed distribution corresponding to the desired image-space hatching density. This approach is well-suited to implicit surfaces.

Much effort was directed at rendering implicit surfaces, esp. metaballs photorealistically in real time, based on ray casting [8]. This is computationally demanding as it requires a high number of field function evaluations to find the visible isosurface in every pixel. The stylization of the result is straightforward only with image-space techniques, as no surface parametrization or visibility-independent object-space shape information is extracted.

Several aspects of stylized rendering of implicit surfaces have been studied. Brazil et al. [22] use *seed points* to generate *render points* on isosurfaces. They require the user to edit seed point distribution manually, excluding application for deforming surfaces. Elber [5] proposed the approach of first obtaining a Euclidean-space on-surface uniform point distribution, then extruding strokes along symbolically computed principal curvature directions [26, 16]. For the generation of uniformly distributed points on implicit surfaces, they refer the reader to Witkin [24], who

[*]szecsi.laszlo@gmail.com

proposes an adaptive resampling of deformed implicit surfaces, for purposes of sculpting, by the means of *repulsion forces*, *fissioning* and *killing* operating on a set of *floater particles*. Kooten et al. [21] employ a similar concept more specifically for isosurface rendering of metaball models. Both solutions require a full *self-spatial join* on surface particles to compute repulsion forces, and allow particles to float on surfaces. We consider this detrimental for our purpose of hatching stylization, as hatch lines not moving with the surface could provide inappropriate motion cues. A rejection-based density control approach from [19] does not require repulsion forces to achieve desired distribution.

3 Implicit surfaces and metaballs

An implicit surface is defined as an isosurface at value L of field function $f(\mathbf{x})$ with the implicit equation $f(\mathbf{x}) = L$. Its gradient $\mathbf{g}(\mathbf{x})$ is $\nabla f(\mathbf{x})$.

Metaballs [14, 2] constitute a special case where the fluid is represented by a number of balls or *atoms* as

$$f(\mathbf{x}) = \sum_{j=0}^{M-1} f_j(\mathbf{x}) = \sum_{j=0}^{M-1} \rho_j\left(\left\|\mathbf{x} - \mathbf{a}_j\right\|\right), \qquad (1)$$

with M as the number of atoms, ρ_j the generator of *radial basis function* $f_j(\mathbf{x})$ for an *atom* centered at $\mathbf{a}_j$. If $\rho_j(r)$ has finite support, i.e. $\exists R_j \in \mathbb{R} : \forall r > R_j : \rho_j(r) = 0$, then we call R_j the *effective radius* of atom j.

The gradient $\mathbf{g}(\mathbf{x})$ and Hessian $\mathbf{H}(\mathbf{x})$ can be computed as sums of atom gradients $\mathbf{g}_j(\mathbf{x})$ and atom Hessians $\mathbf{H}_j(\mathbf{x})$.

Gaussian and mean curvatures K and H, the principal curvatures κ_1 and κ_2, principal curvature directions $\mathbf{t}_1, \mathbf{t}_2$ can be computed [1] using the Hessian $\mathbf{H}(\mathbf{x})$. Where the determinant $D = H^2 - K$ is close to zero, the principal curvatures are not well defined, and we regard the surface point as umbilical.

The approach we employ extrudes textured triangle strips along a metaball surface, in the principal curvature directions. The method we use to move seeds along a metaball surface is similar to [21]. First we cover formula derivations for popular radial basis functions to get the above-mentioned, necessary quantities then we describe the details of curvature computation.

3.1 Gradients and Hessians

In order to be able to evaluate the curvature formulae, we need to compute the field function, its gradient, and Hessian. Those are all obtained as the sum of respective functions for metaball atoms. Here we give the formulae for the infinite support *Blinn* [2] and finite support *Wywill* [25] functions. We give all base functions, gradients and Hessians as functions of $\mathbf{d} = \mathbf{x} - \mathbf{a}$, where $\mathbf{a}$ is the atom position. This is to avoid having to subtract $\mathbf{a}$ at every instance of $\mathbf{x}$.

Before the derivations let us introduce the vectors of pure and mixed second-order partial derivatives as

$$\mathbf{p} = \begin{bmatrix} \frac{\partial^2 \mathbf{f}}{\partial x^2} & \frac{\partial^2 \mathbf{f}}{\partial y^2} & \frac{\partial^2 \mathbf{f}}{\partial z^2} \end{bmatrix}^{\mathrm{T}}$$

and

$$\mathbf{m} = \begin{bmatrix} \frac{\partial^2 \mathbf{f}}{\partial x \partial y} & \frac{\partial^2 \mathbf{f}}{\partial y \partial z} & \frac{\partial^2 \mathbf{f}}{\partial z \partial x} \end{bmatrix}^{\mathrm{T}}.$$

With these Hessian is

$$\mathbf{H} = \begin{bmatrix} p_x & m_x & m_z \\ m_x & p_y & m_y \\ m_z & m_y & p_z \end{bmatrix}.$$

The Blinn base function is:

$$f^{\mathrm{Blinn}}(\mathbf{d}) = \frac{1}{\|\mathbf{d}\|^2}.$$

The gradient is:

$$\mathbf{g}^{\mathrm{Blinn}}(\mathbf{d}) = -\mathbf{d}\frac{2}{\|\mathbf{d}\|^4}.$$

Let us introduce the notation for a *swizzle* of a vector $\mathbf{y}$

$$\mathbf{y}_{yxz} = \begin{bmatrix} y_y \\ y_x \\ y_z \end{bmatrix},$$

and similarly for any order of elements. With this the vector of pure second derivatives $\mathbf{p}(\mathbf{d})$, using $\mathbf{e} = \mathbf{d} \circ \mathbf{d}$, where $\circ$ is the Hadamard product operator, is:

$$\mathbf{p}^{\mathrm{Blinn}}(\mathbf{d}) = \frac{6\mathbf{e} - 2\left(\mathbf{e}_{yzx} + \mathbf{e}_{zxy}\right)}{\|\mathbf{d}\|^6}.$$

The vector of mixed second derivatives $\mathbf{m}(\mathbf{d})$ is

$$\mathbf{m}^{\mathrm{Blinn}}(\mathbf{d}) = \frac{8\mathbf{d} \circ \mathbf{d}_{yzx}}{\|\mathbf{d}\|^6}.$$

The Wywill base function has finite support. Let R be the effective atom radius, and introduce the shorthand $\delta = \|\mathbf{d}\|/R$. With these, the Wywill base function is:

$$f^{\mathrm{Wywill}}(\mathbf{d}) = \begin{cases} 0 & \text{if } \delta > 1, \\ 1 + \frac{-4\delta^6 + 17\delta^4 - 22\delta^2}{9} & \text{if } \delta \leq 1. \end{cases} \qquad (2)$$

With

$$G = \frac{4\left(6\delta^4 - 17\delta^2 + 11\right)}{9R^2},$$

the gradient is:

$$\mathbf{g}^{\mathrm{Wywill}}(\mathbf{d}) = \begin{cases} \mathbf{0} & \text{if } \delta > 1, \\ -\mathbf{d}G & \text{if } \delta \leq 1. \end{cases}$$

The vector of pure second derivatives $\mathbf{p}(\mathbf{d})$, using $\mathbf{e} = \mathbf{d} \circ \mathbf{d}$ is:

$$\mathbf{p}^{\mathrm{Wywill}}(\mathbf{d}) = \begin{cases} \mathbf{0} & \text{if } \delta > 1, \\ \frac{4\mathbf{e}\left(17 - 12\delta^2\right)}{R^4} - \begin{bmatrix} G \\ G \\ G \end{bmatrix} & \text{if } \delta \leq 1. \end{cases}$$

The vector of mixed second derivatives $\mathbf{m}(\mathbf{d})$ is:

$$\mathbf{m}^{\text{Wywill}}(\mathbf{d}) = \begin{cases} \mathbf{0} & \text{if } \delta > 1, \\ \mathbf{d}_{xyz} \circ \mathbf{d}_{yzx} \frac{4\left(12\delta^2 - 17\right)}{9R^4} & \text{if } \delta \leq 1. \end{cases}$$

3.2 Curvature computation

Here we continue using the notations for the vectors of pure and mixed second order partial derivatives and the Hessian from Section 3.1.

The following method of curvature computation is based on [1]. All quantities are functions of $\mathbf{x}$, which we will omit in the notation for ease of reading.

The Gaussian curvature K is

$$K = -\frac{1}{\|\mathbf{g}\|^4} \begin{vmatrix} \mathbf{H} & \mathbf{g} \\ \mathbf{g}^{\mathrm{T}} & 0 \end{vmatrix}. \tag{3}$$

With normal $\mathbf{n} = -\frac{\mathbf{g}}{\|\mathbf{g}\|}$, and Laplacian $\Delta\mathbf{f} = \frac{\partial^2 \mathbf{f}}{\partial x^2} + \frac{\partial^2 \mathbf{f}}{\partial y^2} + \frac{\partial^2 \mathbf{f}}{\partial z^2}$ the mean curvature H is

$$H = \frac{1}{\|\mathbf{g}\|} \left[\mathbf{n}^{\mathrm{T}} \mathbf{H} \mathbf{n} - \Delta\mathbf{f}\right].$$

The principal curvatures are:

$$\begin{aligned} \kappa_1 &= H + \sqrt{(H)^2 - K}, \\ \kappa_2 &= H - \sqrt{(H)^2 - K}. \end{aligned} \tag{4}$$

We need to construct the matrix

$$\left(\mathbf{n} \cdot \mathbf{n}^{\mathrm{T}} - \mathbf{I}\right) \mathbf{H} - \mathbf{I} \kappa_1 \|\mathbf{g}\|,$$

where $\mathbf{I}$ is the identity matrix, then take the maximum length one out of the three possible pairwise cross products of its rows. Normalized, it gives principal direction $\mathbf{t}_1$. Then, $\mathbf{t}_2 = \mathbf{t}_1 \times \mathbf{n}$.

Using the *swizzle* notation from Section 3.1 the determinant of equation 3 can be computed without explicitly constructing the matrix as

$$\begin{aligned} \begin{vmatrix} \mathbf{H} & \mathbf{g} \\ \mathbf{g}^{\mathrm{T}} & 0 \end{vmatrix} = \\ & 2(\mathbf{p} \circ \mathbf{m}_{yzx}) \cdot (\mathbf{g}_{yzx} \circ \mathbf{g}_{zxy}) \\ & - (\mathbf{p}_{zxy} \circ \mathbf{p}_{yzx}) \cdot (\mathbf{g} \circ \mathbf{g}) \\ & + (\mathbf{m} \circ \mathbf{m}) \cdot (\mathbf{g}_{zxy} \circ \mathbf{g}_{zxy}) \\ & - 2(\mathbf{m}_{xzy} \circ \mathbf{m}_{yxz}) \cdot (\mathbf{g}_{xzy} \circ \mathbf{g}_{zyx}). \end{aligned} \tag{5}$$

4 Seeds and their motion explained

Seeds are particles moving along the deforming isosurface. The velocity vector used to move a seed is found by using the formulae described in [21]. There are three effects that contribute to this motion: fluid motion, field shift, and correction.

4.1 Complete seed velocity

4.2 Fluid motion

The fluid medium itself is moving. Its motion is defined for atoms with atom velocities $\mathbf{q}_j$. How we construct the flow velocity at a point from these relies on the requirement that points on the isosurface should remain on the isosurface. How much the linear motion of an atom influences the isosurface depends on the length of the base function gradient at the isosurface point. Thus, linear atom velocities should be weighted with this gradient length to get the flow velocity:

$$\mathbf{v}^{\text{fl}}(\mathbf{s}) = \frac{\sum_{j=0}^{M-1} \left\|\mathbf{g}_j(\mathbf{s}_k)\right\| \mathbf{q}_j}{\sum_{j=0}^{M-1} \left\|\mathbf{g}_j(\mathbf{s}_k)\right\|}.$$

The seeds need to travel along the isosurface, so the fluid velocity must be projected on it. The component perpendicular to the surface is found as

$$\mathbf{v}_k^{\text{perp}} = \frac{\mathbf{g}(\mathbf{s}_k) \left(\mathbf{v}_k^{\text{fl}} \cdot \mathbf{g}(\mathbf{s}_k)\right)}{\|\mathbf{g}(\mathbf{s}_k)\|^2},$$

and thus the projected fluid velocity is

$$\mathbf{v}_k^{\text{pfl}} = \mathbf{v}_k^{\text{fl}} - \mathbf{v}_k^{\text{perp}} = \mathbf{v}_k^{\text{fl}} - \frac{\mathbf{g}(\mathbf{s}_k)(\mathbf{v}_k^{\text{fl}} \cdot \mathbf{g}(\mathbf{s}_k))}{\|\mathbf{g}(\mathbf{s}_k)\|^2}.$$

4.3 Surface pull

Seeds need to move towards the isosurface either because they are distant due to initial or accumulated error, or because the isosurface itself has moved. For both effects, we will be able to find the desired rate of change in field value at the seed $\delta = \frac{\partial f(\mathbf{s}_k)}{\partial t}$, and need to compute the seed velocity $\mathbf{v}_k^{\text{pull}} = \partial \mathbf{s}_k / \partial t$ from this. We move the seed along the gradient, so $\mathbf{v}_k^{\text{pull}} = \xi \mathbf{g}(\mathbf{s}_k)$ with some ξ. It must be true that

$$\delta = (\xi \mathbf{g}(\mathbf{s}_k)) \cdot \mathbf{g}(\mathbf{s}_k).$$

Solving this for ξ gives

$$\xi = \frac{\delta}{\|\mathbf{g}(\mathbf{s}_k)\|^2},$$

and then

$$\mathbf{v}_k^{\text{pull}} = \frac{\mathbf{g}(\mathbf{s}_k)\delta}{\|\mathbf{g}(\mathbf{s}_k)\|^2}.$$

Correction

As neither the temporal nor the spatial linearizations applied are accurate, the seeds positions would accumulate error and drift away from the isosurface. Also, when initialized, the seeds are at random positions and need to be drawn to the isosurface rapidly. Therefore, a correction term with boldness factor Φ is applied. The boldness factor Φ is the inverse of the time in which the seed is supposed to reach the isosurface. Thus, δ^{corr} is $(L - f(\mathbf{s_k}))\Phi$.

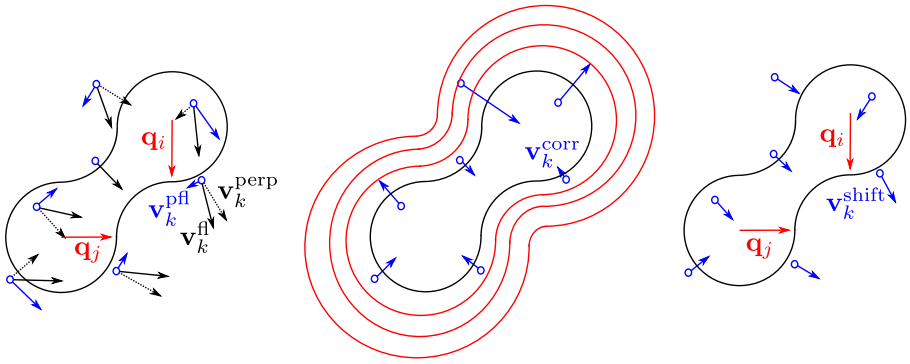

Figure 1: The components of seed velocity: the surface-projected fluid velocity (*left*), the correction term toward the surface along the gradient (*center*), and the term following the isosurface shift due to chaging field values (*right*).

However, large Φ values can lead to instabilities near strongly non-linear regions of the field function.

$$\mathbf{v}_k^{\text{corr}} = \frac{\mathbf{g}(\mathbf{s_k})(L - f(\mathbf{s_k}))\,\Phi}{\|\mathbf{g}(\mathbf{s_k})\|^2}.$$

Field shift

As atoms move, the field value at a $\mathbf{s}_k$ is going to increase or decrease. This change will make the isosurface of L move along the gradient. The rate of change at seed k due to atom j moving is:

$$\delta_j^{\text{shift}} = -\mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j,$$

and the total effect of all atoms is:

$$\delta^{\text{shift}} = -\sum_{j=0}^{M-1} \mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j.$$

This makes the shift velocity:

$$\mathbf{v}_k^{\text{shift}} = -\frac{\mathbf{g}\sum_{j=0}^{M-1} \mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j}{\|\mathbf{g}\|^2}.$$

All terms, save for the unprojected fluid velocity, contain the gradient divided by its length squared. Their sum can therefore be written as:

$$\mathbf{v}_k = \mathbf{v}_k^{\text{fl}} - \tag{6}$$

$$\frac{\mathbf{g}(\mathbf{s_k})}{\|\mathbf{g}(\mathbf{s_k})\|^2}\left(\mathbf{v}_k^{\text{fl}} \cdot \mathbf{g}(\mathbf{s}_k) + (f(\mathbf{s_k}) - L)\,\Phi + \sum_{j=0}^{M-1} \mathbf{g}_j(\mathbf{s}_k) \cdot \mathbf{q}_j\right)$$

A visual representation of this equation can be seen in Figure 1.

5 The algorithm

Our algorithm moves seeds along a metaball surface similar to [21], applies a screen-space approximate version of the density control approach from [19], and extrudes textured triangle strips along principal curvature directions.

We propose a solution for the seed visibility problem based on the idea employed by *variance shadow maps* [4]. The algorithm performs the following steps in every frame of an animation:

1. Initialization of spawned seeds.
2. Seed animation.
3. Seed filtering by visibility testing and rejection.
4. Curve extrusion from seeds.
5. Triangle strip extrusion from curves.
6. Stroke weighting and rendering.

Along the process, various weighting factors are computed for the seeds—w^{prox} for proximity to isosurface, w^{age} for age , w^{vis} for visibility, w^{rej} for density control by rejection. The product of these w^{Π} is used in the final rendering step for opacity weighting, with the optimization that seeds with zero weight need not be extruded into hatch strokes. The weight without density control, $w^{\text{pre}} = w^{\text{prox}} w^{\text{age}} w^{\text{vis}}$ is used for estimating *pre-rejection density*.

When seeds are initialized, they are placed randomly on atom-centered spheres within the effective radius. They are not guaranteed to be on the compound isosurface, and the isosurface-projected distribution might not be uniform. Those requirements are to be achieved by consequent seed animation and rejection steps, over the course of several frames. Seed points are re-initialized after a fixed lifetime to avoid excessive clustering. Seed point ages are evenly distributed, so that only a small percentage of seeds are re-initialized in every frame. Weight w^{prox} is computed as a smooth step function on the difference of the field value at the seed point and the desired isosurface. This is to eliminate seeds not yet converged to the surface. Weight w^{age} fades to zero at the beginning and the end of the seed lifetime to avoid suddenly appearing and disappearing hatch lines.

Seed point animation is based on the technique proposed by [21], without using repulsion forces to achieve

uniform density, thus eliminating the need for a self-spatial join on seeds. Seed animation according to Section 4.3 requires the field value and the gradient. We compute these, and also the world space *stroke direction* along the isosurface. The computation of the stroke direction involves first finding the pure and mixed second derivatives forming the Hessian, the principal curvatures and curvature directions, the determinant D indicating whether the seed is at an umbilical point, the cosine of the view angle $\cos\Theta$ indicating whether the seed is near a silhouette, and the local illumination V at the seed, normalized to a desired tone.

Generally, the stroke direction is the principal curvature direction of the isosurface, but near umbilical points, we employ a custom direction, obtained as the cross product of the surface normal and a per-atom direction vector. The choice of this per-atom vector might be random, or subject to artistic consideration. In order to produce simple outlines, a different direction scheme is applied to lines near the silhouette: the stroke direction there is perpendicular to both the view direction and the surface gradient (see Figure 2). The three direction schemes are combined based on D and $\cos\Theta$, so that there are no abrupt changes in the stroke direction. For any direction $\mathbf{t}$, the corresponding curvature κ can be found as $\kappa = \kappa_1 (\mathbf{t}\cdot\mathbf{t}_1)^2 + \kappa_2 (\mathbf{t}\cdot\mathbf{t}_2)^2$.

Seed points have to pass two filters to see if they should be extruded into hatch strokes. The first is the visibility test needed to decide if the seeds are seen from the camera. For this purpose, we render all seeds as isosurface-oriented billboards into a low-resolution buffer, outputting fragment depths and their squares. The purpose here is to approximate the depth of the isosuface itself by using the depth values of the billboards covering it. Using the idea of *variance shadow maps* [4], this low-resolution depth map is heavily filtered by two-pass separable Gaussian filtering. The resulting approximate variance depth map can be used for a smooth and lenient rejection of hidden seeds, producing visibility factor w^{vis}. Using this visibility factor to modify hatch stroke opacity causes strokes at and behind the boundaries of the surface to fade out smoothly, enabling partially visible strokes to appear. As we are emulating the hand-drawn style, the error—from approximating the isosurface with billboards, using a low-resolution map, aggressive filtering, and testing for visibility only at seeds—is not only acceptable, but welcome.

The second rejection step is to achieve an illumination-dictated screen space density of seed points (Figure 3). The *full cover density* Υ^{full} is an artistic parameter that specifies the seed density corresponding to surfaces devoid of illumination. This, modulated by seed tone V_k gives the desired on-screen density near a seed. Let us refer to the local density of all screen-projected seed points (weighted by w^{pre}) as Υ^{pre}. The ratio of $V_k\Upsilon^{\text{full}}/\Upsilon^{\text{pre}}$ gives the percentage of seed points to be kept. If all seed points have a random normalized priority value p_k, then those with priorities above the desired percentage should be rejected. The Υ^{pre} density is approximated by rendering all visible seeds, extruded into approximate hatch strokes, with additive blending, weighted by w^{pre} into a low-resolution buffer, and performing heavy filtering to eliminate rasterization artifacts. Note that what we get is not exactly the density of seeds, but an approximate density of hatching coverage. Thus, it helps to eliminate not only the clustering of seeds, but also the clustering of aligned strokes. Weight w^{rej} is computed as a smooth step function of $V_k\Upsilon^{\text{full}}/\Upsilon^{\text{pre}} - p_k$. Thus, rejection is performed smoothly, thus avoiding temporal visual artifacts, i.e. suddenly disappearing, appearing, or flickering hatch lines.

The seeds surviving visibility testing and rejection are extruded into curves. For short strokes, it is sufficient to use the local curvature at the seed, but longer lines require integration along the isosurface. In the latter case, visibility testing has to be performed for all samples. Curves are extruded into triangle strips to a uniform image space width. This width, and also the length of strokes, is an artistic parameter.

In the final rendering step the stroke is textured with an artist-drawn stroke image, with weights applied as opacity modifiers. We only discard the seeds if the weight would indeed be zero.

6 Implementation

The steps of our algorithm are implemented in five passes, depicted in Figure 4.

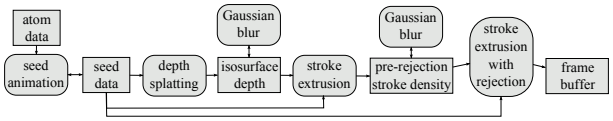

Figure 4: Shader passes of the implementation.

The first pass performs seed animation. All seed data is stored in textures used as data tables, where rows correspond to atoms, and the elements of the rows are individual seed points. Aging and re-initialization of seeds is performed by a rotating pipeline. In fact, in every texture row, seed attributes are shifted out to the right and reinitialized seeds shift in from the left, at a constant rate. The textures are also shifted vertically, to account for newborn and dying atoms, if so dictated by fluid simulation. For computation of quantities derived from the field function we used a regular grid space subdivision scheme to access relevant atoms.

The second pass produces the variance depth map of the isosurface to be used for a visibility filtering. Billboards are only extruded for seeds already converged to the surface to avoid unnecessary occlusion by seeds that are still trying to find their place. The depth values are blurred using a Gaussian filter, in accordance with the VSM technique, eliminating jagged edges in the depth map that could cause flickering hatch strokes in the final image.

The fourth pass is used to produce an image of Υ^{pre} values. These are needed for rejection of seeds later, to

Figure 2: Hatching of an LSD molecule discarding seeds near silhouettes (left) and rotating strokes to produce outlines (right).

achieve uniform screen space density. It extrudes hatch strokes from all visible seed points, and applies the same opacity weighting to them—for visibility, age and proximity to the isosurface—, as would be when rendering on-screen strokes. Rejection for density, however, is not applied, since the goal is to approximate the hatching density from all visible seeds. After visibility determination and curve extrusion, the hatch strokes are rendered, given color and opacity values that smoothly fall off towards the edges of the strokes. The output of this pass is rendered to a texture, using additive blending to generate high density values for high density areas on the screen.The Υ^{pre} density values also need to be blurred, to avoid rasterization artifacts caused by jagged edges of approximate hatch strokes.

In the final pass, the process of rejection and opacity weighting based on visibility and hatch stroke extrusion is the same as it was during rendering the Υ^{pre} density. In addition, this pass also weights seed points using the Υ^{pre} values, and illumination values calculated on the fly, before extruding the hatch strokes themselves. If the compound weight of the seed is positive, the strokes are extruded, textured, and opacity is modulated by all weighting factors.

7 Results and future work

We ran our tests on a PC with an ATI5850 graphics card. At a resolution of 1024×768, with 65K seeds, which we deemed sufficient for rendering quality, and regardless of the number of atoms, we measured frame rates around 20 FPS.

Extruding long hatch curves requires several curvature samples on the isosurface, and as curves travel into zones of different curvature characteristics they tend to cross each other. Density estimation at seeds is also less accurate in this case. Therefore, we wish to investigate the possibility of using several linked seeds points for every hatch curve. Another limitation of the method is that the seed density cannot exceed what is provided by rendering all seeds at unit weight. This is made worse if the distribution of seed points gets uneven because of seed motion. Thus, we plan to add seed fissioning and killing to improve performance and provide much wider level-of-detail support without increasing the seed count.

8 Acknowledgments

This work has been supported by OTKA PD-104710 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

[1] Alexander G Belyaev, Alexander A Pasko, and Tosiyasu L Kunii. Ridges and ravines on implicit surfaces. In *Computer Graphics International, 1998. Proceedings*, pages 530–535. IEEE, 1998.

[2] J.F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256, 1982.

[3] N.K.R. Bolla. High quality rendering of large point-based surfaces. Master's thesis, International Institute of Information Technology, Hyderabad-500 032, INDIA, 2010.

[4] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 symposium*

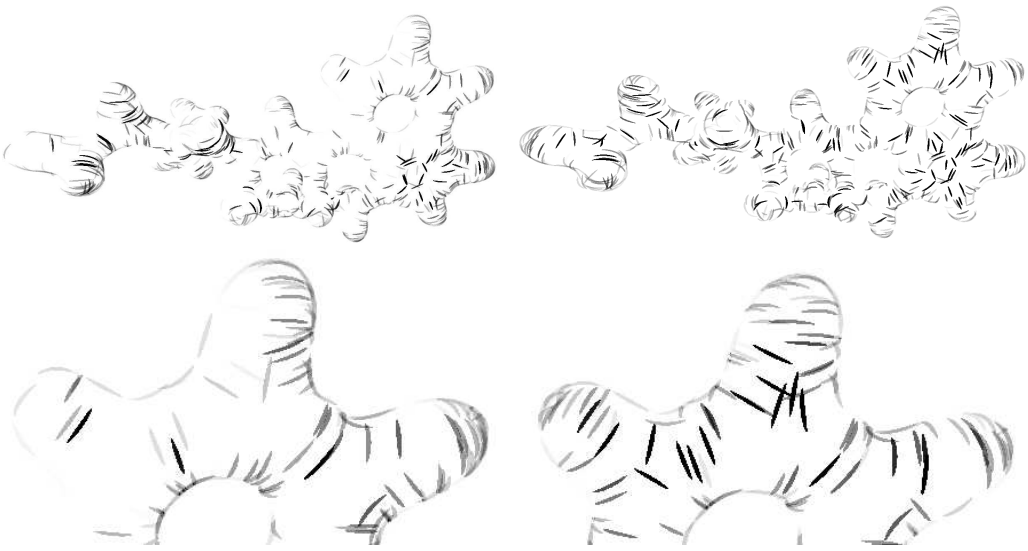

Figure 3: Uniform hatching of an LSD molecule with and without illumination.

on Interactive 3D graphics and games, pages 161–165. ACM, 2006.

[5] Gershon Elber. Interactive line art rendering of freeform surfaces. In *Computer Graphics Forum*, volume 18, pages 1–12. Wiley Online Library, 1999.

[6] Ahna Girshick, Victoria Interrante, Steven Haker, and Todd Lemoine. Line direction matters: an argument for the use of principal directions in 3d line drawings. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 43–52. ACM, 2000.

[7] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

[8] Y. Kanamori, Z. Szego, and T. Nishita. GPU-based fast ray casting for a large number of metaballs. In *Computer Graphics Forum*, volume 27, pages 351–360, 2008.

[9] Yongjin Kim, Jingyi Yu, Xuan Yu, and Seungyong Lee. Line-art illustration of dynamic and specular surfaces. In *ACM Transactions on Graphics (TOG)*, volume 27, page 156. ACM, 2008.

[10] Yunjin Lee, Lee Markosian, Seungyong Lee, and John F Hughes. Line drawings via abstracted shading. In *ACM Transactions on Graphics (TOG)*, volume 26, page 18. ACM, 2007.

[11] Zoltán Lengyel, Tamás Umenhoffer, and László Szécsi. Screen space features for real-time hatching synthesis. In *Proceedings of the 9th conference of the Hungarian Association for Image Processing and Pattern Recognition*, KEPAF '13, pages 82–94, 2013.

[12] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

[13] Barbara J Meier. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484. ACM, 1996.

[14] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modeling by distribution function and a method of image generation. *The Transactions of the Institute of Electronics and Communication Engineers of Japan*, 68(Part 4):718–725, 1985.

[15] T. Nishita and E. Nakamae. A method for displaying metaballs by using bézier clipping. In *Computer Graphics Forum*, volume 13, pages 271–280. Wiley Online Library, 1994.

[16] Afonso Paiva, Emilio Vital Brazil, Fabiano Petronetto, and Mario Costa Sousa. Fluid-based hatching for tone mapping in line illustrations. *The Visual Computer*, 25(5-7):519–527, 2009.

[17] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, page 581. ACM, 2001.

[18] László Szirmay-Kalos, György. Antal, and Ferenc Csonka. *Háromdimenziós grafika, animáció és játékfejlesztés*. ComputerBooks, Budapest, 2003.

[19] T. Umenhoffer, L. Szécsi, and L. Szirmay-Kalos. Hatching for motion picture production. In *Computer Graphics Forum*, volume 30, pages 533–542, 2011.

[20] W.J. van der Laan, S. Green, and M. Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, pages 91–98. ACM, 2009.

[21] K. van Kooten, G. van den Bergen, and A. Telea. Point-based visualization of metaballs on a GPU. *GPU Gems*, 3:123–148, 2007.

[22] Emilio Vital Brazil, Ives Macêdo, Mario Costa Sousa, Luiz Velho, and Luiz Henrique de Figueiredo. Shape and tone depiction for implicit surfaces. *Computers & Graphics*, 35(1):43–53, 2011.

[23] Georges Winkenbach and David H Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM, 1994.

[24] Andrew P Witkin and Paul S Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 269–277. ACM, 1994.

[25] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The visual computer*, 2(4):227–234, 1986.

[26] Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High quality hatching. In *Computer Graphics Forum*, volume 23, pages 421–430. Wiley Online Library, 2004.