Vis Comput (2012) 28:691–700
DOI 10.1007/s00371-012-0684-1

ORIGINAL ARTICLE

Real-time controllable fire using textured forces

Jake Lever · Taku Komura

Published online: 5 May 2012

Abstract Fluid dynamics can produce realistic looking fire effects, which are heavily used in animation and films. However, the parameters of the various underlying physical equations are not intuitive enough to be controlled easily. As a result, animators face problems when editing the fine details of the fire, especially the turbulence and growth at the fire surface. In this paper, we propose a new approach to enable animators to interactively edit such fine details using textured forces. These techniques involve mapping a texture onto the simulation that controls the creation of new forces, growing the fire into specific shape and adding the natural turbulence of fuel ignition. These textures can be edited using an intuitive user interface that allows forces to be painted directly onto the fire. Our system can be integrated into existing GPU fluid solvers to run in real-time. As a result, it is applicable for interactive applications such as 3D computer games.

Keywords Fire animation · Fluid simulation

1 Introduction

Fire effects based on fluid dynamics have been used in motion pictures and video games to captivate the audience and make the experience more immersive. Particle systems in particular are straightforward to visualise and understand but lack the correct physical motion that appears in real fire. Fluid simulation holds the key to the correct simulation of fire. Although the computational cost of simulation is fairly high, recent approaches for parallelising fluid simulation using the graphics processing unit (GPU) have enabled real-time animation on consumer level personal computers.

Getting the correct appearance for the fire normally requires careful control of the various parameters of the underlying physical equations. These parameters are neither intuitive nor straightforward to understand. It is especially difficult for the artists to create fire effects with a particular shape and turbulence. The main source of turbulence that occurs inside the flame is due to the pressure caused when fuel ignites and occurs on the interface between the fuel and the burning flames.

This paper presents a new method of controlling the shape of fire that is suited to both simple fire effects and more dynamic simulations with fully controllable shapes. A 2D or 3D texture mapped onto the simulation controls the strength of additional forces which control the growth of the flame and add artificial turbulence. This allows for easy control of the fire through textures that can be procedurally generated or created by an artist. Moreover, these textures can be adapted frame-by-frame to animate fire into different controllable shapes while maintaining the visually pleasing turbulence caused by burning fuel.

This method can be executed entirely on the GPU in real-time, including simulation, rendering and interaction stages. Using NVIDIA's CUDA frees the CPU to manage other parts of the animation process making it ideal for both video games and fast animation tools for visual effects. Throughout the paper, we show various types of fire that can be pro-

Electronic supplementary material The online version of this article (doi:10.1007/s00371-012-0684-1) contains supplementary material, which is available to authorized users.

J. Lever · T. Komura (✉)
School of Informatics, The University of Edinburgh,
10 Crichton Street, Edinburgh, UK
e-mail: tkomura@ed.ac.uk

J. Lever
e-mail: jake.lever@gmail.com

duced and simulated interactively by simply changing the texture pattern applied to the simulation.

1.1 Previous work

Fires have been simulated by particle systems [24, 25, 30, 31], cellular automata [4], flame primitives [2], and fluid dynamics [22]. Methods based on particle systems and cellular automata have advantages given their simple methods for controlling the fire. However, the resulting animation fails to produce realistic results, especially the volumetric effect of actual flames. Methods based on flame primitives have difficulties in simulating the real turbulence inherent in fire.

Fire simulation is a subset of the larger fluid dynamics research area. Recent advances in fluid simulations [28, 29] have enabled realistic fire animations. The burning flame can be treated as an incompressible fluid, and is thereby governed by the incompressible Navier–Stokes equations.

Nguyen et al. [22] utilises a surface-tracking system that tracks the barrier between the burning flames and the fuel and calculates the velocities of these two systems independently. Hong et al. [13] make use of detonation shock dynamics to produce features such as cellular patterns that appear as visually distinct patterns in the fire. These methodologies simulate effects such as ignition turbulence by tracking the interface of the reaction zone where fuel ignition occurs. These techniques are categorised as the two-phase approach, in comparison to the classic one-phase approach [11] that does not separate the models of the unburnt fuel and the burning fire. The two-phase approach involves numerous physical parameters which are not intuitive and difficult to control. In addition to that, they require a significant amount of memory and computation, and thus it is not easy to evaluate the results in real-time.

Controllable fluids have been an active research area in the last few years. Foster and Metaxas [10] propose a method of high-level control of fluid simulation. Foster and Fedkiw [9] suggest a method by constraining the velocity field at some locations. Treuille et al. [33] use a keyframe method to control the fluid animation. McNamara et al. [20] accelerates this method by the adjoint method. Fattal and Lischinski [7] add a driving force that attracts the fluid to a specific target and a gathering term that prevents them from diffusing to control the shape of fluids. The method by Shi and Yu [27] adds velocity at the boundary of the target shape. Lamorlette and Foster [18] share a similar goal of making easily controllable but also visually appealing fire, but focuses on a stochastic model based on Kolmogorov noise.

Although these methods are suitable for controlling the global shape or flow of the fluid, a significant challenge involves controlling the local details such as the turbulence and flame growth using a simple and understandable interface which is the main focus this paper. Our approach aims to create fire effects with visually appealing textures effects similar to [13] without the large computational cost.

Many have looked at how artificial noise can be added to a simulation. Vorticity confinement in [8] shows a method of reintroducing lost turbulence into the simulation. Wavelet noise was used in [16] and procedural turbulence in [21] to calculate subgrid details of a simulation. Schechter and Bridson [26] also develops plausible turbulence that is added on top of the existing simulation. These papers focus on increasing the visible quality of fluid simulations while our use of procedural noise differs in that it is the main force emulating the fire movement. These techniques could be used in conjunction with our fire model although would decrease the performance of the real-time system.

Fast and stable fluid simulation techniques are implemented on the GPU [12] to animate fluids such as smoke, fire and liquid in real-time. We take a similar approach to [5] in order to solve the fluid equations in real-time. Crane et al. [5] examine how the fluid solver can be executed in real-time using various GPU optimisations to simulate a 3D domain and look at techniques for rendering the 3D simulation domain using GPU volume rendering. Horvath and Geiger [14] examine how the GPU can be used for fire effects based on a particle system and integrated into a production environment.

1.2 Our approach

In order to achieve real-time simulation and user interaction with the fire, we utilise a classic one-phase fluid approach [11], which requires less memory and computation and through our methods can be controlled very easily. Once the isosurface of the fire is found, we synthesise realistic fires by adding trigonometric forces at the interface. The animator has access to the fine details of the fire shape through the force texture and the intuitive painting interface provided. Figure 1 outlines an example of using textures to control the shape with added artificial turbulence.

1.3 Contribution

- A simple method to control the turbulence and shape of a fire based on forces created by a texture pattern
- An intuitive method to interact with the shape of the fire using a paintable isosurface, and hence control the internal forces
- These techniques can be executed in real-time on the GPU and carry significantly low computational and memory cost

2 Overview

In our approach, a uniform 3D grid is used to define the temperature field of the fuel and an equally sized vector field is used to define the velocity of the fluid. We first track the fuel

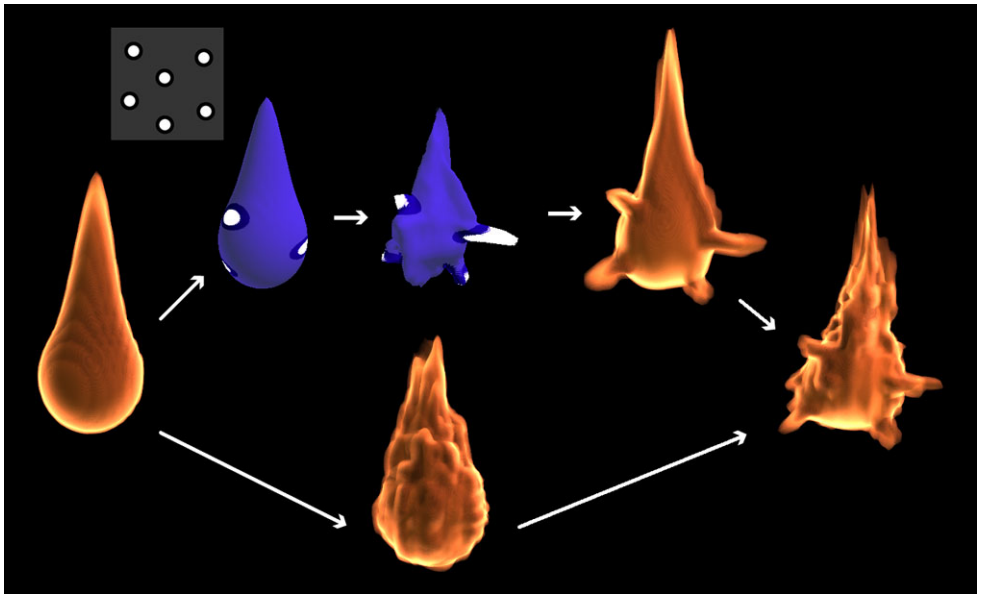

Fig. 1 This shows a basic process of applying a 2D dotted texture (at *top-left*) to the normal forces only (at *top*) with a simple sine-based noise force (at *bottom*) and combining them (at *right*)

isosurface in the temperature field outlined in Sect. 3. Next, forces based on the physical instability and additional user input are computed and added into the domain. These additional forces depend on a 2D or 3D force texture outlined in Sect. 4. A simple user interface discussed in Sect. 5 allows the user to adjust the textures and hence the shape of the fire. Once all the external forces are computed, they are used to update the velocity and temperature field using an incompressible Navier–Stokes solver (Sect. 6). We briefly explain the method of parallelisation in Sect. 7 which allows the animation to execute in real-time, and show the experimental results in Sect. 8. The advantages of this approach are investigated in Sect. 9.

3 Fire isosurface

When simulating fire, finding the interface between the fuel and the ignited fire is important to create realistic animation. The chemical reaction of the fuel ignition adds external forces to the fluid due to pressure changes. In addition to this, the discontinuity of the fuel at the two sides causes instability resulting in turbulence. Our approach artificially defines an isosurface inside the fire and adds appropriate forces at this surface.

This isosurface is also key to how the animator interacts with the fire as they can apply a texture to the isosurface or paint onto the isosurface that then directly affects the additional forces added to the fire.

3.1 Isosurface definition

A level-set function is an often used technique to track a fire isosurface [22], where the scalar value at each grid point is the distance to the isosurface. However, animating a level-set function correctly is costly and requires additional memory storage. Instead the temperature field is used to define the isosurface at a predefined scalar value such that $temp(x, y, z) = t_{\text{ignition}}$, affectively defining an isothermal contour in the temperature field. The resulting surface is outlined in Fig. 2. By using lower values of t_{ignition} the boundary is closer to the visual edge of the fire. t_{ignition} would depend on the type of fuel being simulated and on the temperature of the source added to the simulation.

Each grid cell is examined to see whether it is intersected by the isosurface, similar in concept to the marching cubes algorithm [19]. This is achieved by comparing the values of the neighbouring cells to see if they fall on the other side of the isosurface. By using the preexisting temperature field, no additional memory storage is required to track the ignition boundary. Additional normal and noise forces are then added to the simulation dependent on the isosurface. The results of these forces are illustrated in Fig. 1.

3.2 Normal force

The normal force controls the growth of the fire and simulates the physical effect of igniting fuel pushing the flame outward. It can also be used to pull the fire in to emphasize specific shapes. Given a cell at the position (i, j, k) on the isosurface, the normal of isosurface can be approximated using the discrete gradient of the temperature field at that point as shown below.

$$grad(i, j, k) = \begin{pmatrix} \frac{F_{i+1,j,k} - F_{i-1,j,k}}{2} \\ \frac{F_{i,j+1,k} - F_{i,j-1,k}}{2} \\ \frac{F_{i,j,k+1} - F_{i,j,k-1}}{2} \end{pmatrix}$$

$$f_{\text{normal}}(i, j, k) = -\frac{grad_{(i,j,k)}}{\|grad_{(i,j,k)}\|}$$

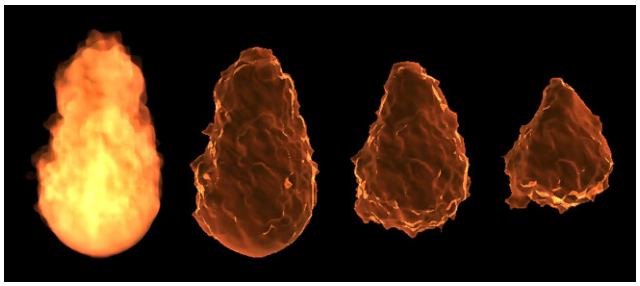

Fig. 2 A simulated fire and the interfaces defined at 0.5 (*middle-left*), 1.0 (*middle-right*), and 1.5 (*right*)

As the gradient points toward higher temperature, and toward the source of the fire, the normal force is defined as the negative of the normalised gradient for that point. This simulates the expansion of the fire due to ignition and is a powerful tool in shaping the fire.

3.3 Noise force

In order to add more additional turbulence and reduce the likelihood of the simulation reaching stable equilibrium, velocity created using procedurally generated noise is added to the cells on the isosurface. Various noise functions could be used to create a variety of effects. Sinusoidal noise varies smoothly creating nicely flickering flames, making it very controllable. The noise function shown below depends on the 3D position (i, j, k) of the isosurface cell and the current frame number t.

$$f_{\text{sine noise}}(i, j, k) = \begin{pmatrix} \sin(ij + t) \\ \cos(jk + t) \\ \sin(ik + t) \end{pmatrix}$$

The function can easily be adapted for different requirements of turbulence. By changing the frequency of the trigonometric function, faster and more tightly packed turbulence will be created.

A basic sine/cosine function can become visually periodic, particularly if no other forces act on the fire for a significant period of time. To avoid this issue, the trigonometric functions are shifted by a random number every 100 frames. Hence, the noise will be continuous for the majority of the time but will shift occasionally removing the visual repetitiveness. This shift is less visually apparent than the repetitive movement, especially if the rate of change is adjusted randomly.

Noise based on the tangent function shown below creates very aesthetically pleasing turbulence. While this type of noise does not work as well with textured normal forces to create a particular shape for the fire, it can be used very successful as textured noise to create impressive resulting images.

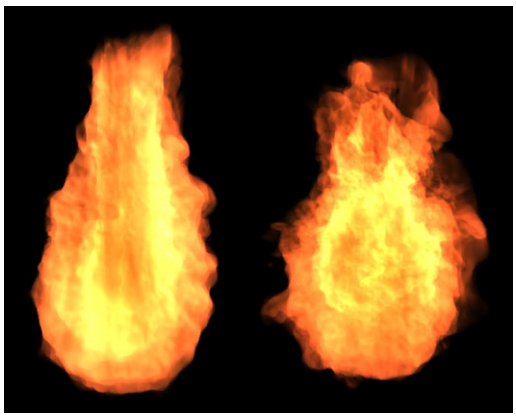

Fig. 3 This illustrates the different effects from sine-based (*left*) and tan-based (*right*) noise functions

$$f_{\text{tan noise}}(i, j, k) = \begin{pmatrix} \tan(ij + t) \\ \tan(jk + t) \\ \tan(ik + t) \end{pmatrix}$$

Figure 3 shows examples of the different noise types. It is clear that noise based on sinusoidal functions does not affect the overall shape of the fire and only provides local turbulence. However, trigonometric noise based on tangent functions affects the entire shape of the fire and, while more aesthetically pleasing, is more challenging to control.

4 Force textures

We propose a method to let animators control the fine details of the fire by using 2D and 3D textures. The designed textures are used as multipliers of the normal and noise forces, effectively controlling the size and turbulence of the fire. This approach has the following advantages: (1) It allows animators to paint onto a texture which when applied to the simulation creates additional turbulence or growth in the fire. (2) It allows for easy importing of existing 2D/3D texture data from other applications with minimal modification. (3) A large wealth of expertise can be applied to optimise the work flow in order to make the creation of specific fire effects very fast. For creating animation of fire changing its shape, we can simply switch between different textures. If we want them to morph smoothly, interpolation techniques such as [32] can be applied. In the rest of this section, we explain the details of using 2D and 3D textures to control the forces.

4.1 2D textures

We map 2D textures onto the isosurface of the fire and use it as the multiplier of the appropriate forces, predominately

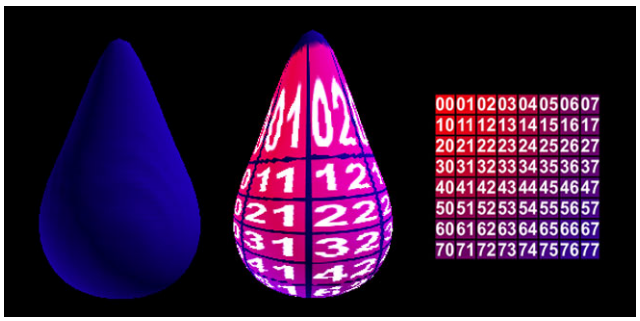

Fig. 4 This shows a grid texture mapped onto a stable fire isosurface

the normal force. The greatest issue with using a single 2D texture is how to generate the texture coordinates. Here, we use spherical mapping that uses the angles between the grid cell and the centre of the fire source. This effectively wraps the entire isosurface in the texture and means that the same area of texture will always relate to the same basic region of the isosurface. This technique is outlined in Fig. 4. A static texture applied to the different force multipliers will create a stable and effective fire effect as shown in Fig. 5. A normal texture value used to affect the individual forces will vary from 0 to 1. In order to stop the fire growing outward constantly, the range of the texture values is adjusted so that some negatives forces are applied, hence the value varies from -0.5 to 0.5. For procedurally generated textures, the approaches suggested in [1] and [17] could also be used to develop a texture over the fire isosurface. Spherical mapping will only suit a subset of source types. For other source types such as a plane, plane projection could be used to map a texture onto the side of the fire.

Obviously, this technique suffers from distortion as the fire grows and the interface changes shape therefore meaning an area of the texture will move slightly on the surface of the fire. A 3D texture outlined in Sect. 4.2 allows for finer adjustments while a 2D texture even with distortion can control the general shape of the fire.

4.2 3D textures

The use of 3D textures allows more specific control over the finer details of the fire. Instead of the texture only controlling forces around the fire isosurface, the texture can control velocities at any point inside the domain. This gives the animator much more control over finer details and avoids the problems of distortion from which 2D textures can suffer. As the 3D texture affects the entire simulation domain, all velocity cells lookup a point inside the texture and use that as the multiplier for the local normal force. Hence, areas with large texture values will increase the fire growth in those areas, meaning that the fire can be drawn out into specific shapes with realistic looking flame growth.

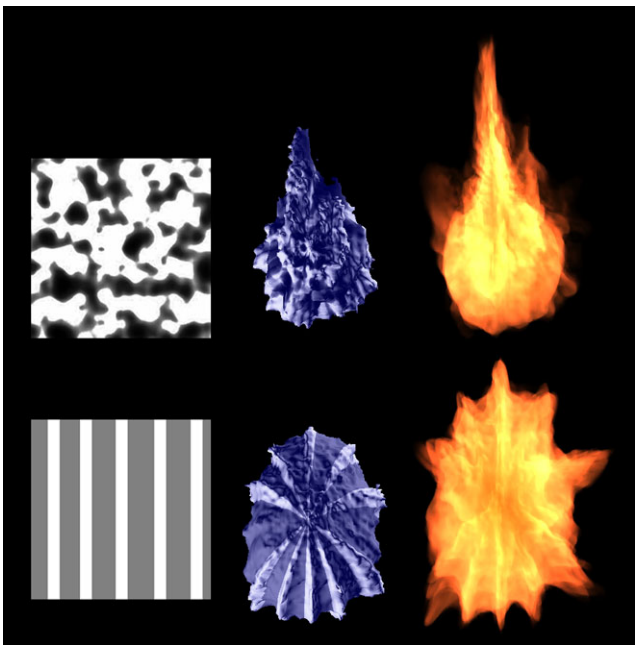

Fig. 5 This shows the musgrave [6] and striped textures applied to the fire isosurfaces statically to create two different fire effects

At each time-step, noise forces are also added in the same way as the 2D approach, by adding trigonometric velocities at the cells intersected by the isosurface. It is only applied at the fire isosurface to maintain the visual effect of the turbulence along the ignition boundary but the normal force is applied in all cells to allow the grow outwards more quickly and speed up in locations away from the isosurface.

By using the simple painting method outlined in Sect. 5, the user can paint onto areas of the surface from which the fire should grow. This allows the animator to control the growth of the fire around a dynamic scene while maintaining the aesthetically pleasing turbulence created by the noise functions.

5 User interaction with fire

We propose an intuitive sketch interface to draw the 2D and 3D textures onto the fire. As shown in Fig. 6, the animator can interact with the fire easily painting forces directly onto the isosurface of the fire. The location of the user click on screen is saved and the 3D position on the interface is recovered from the rendering information. In the case of the 2D wrapped texture, a 2D circle brush is then applied to that point in the 2D texture either increasing or decreasing the texture values. In case of 3D texture, a spherical brush is used to affect texture values around a particular 3D point. Figure 11 outlines the basic effect possible of adding shapes onto the interface of the fire.

The changes to the texture are directly visible to the user as they are rendered on top of the isosurface. The user can

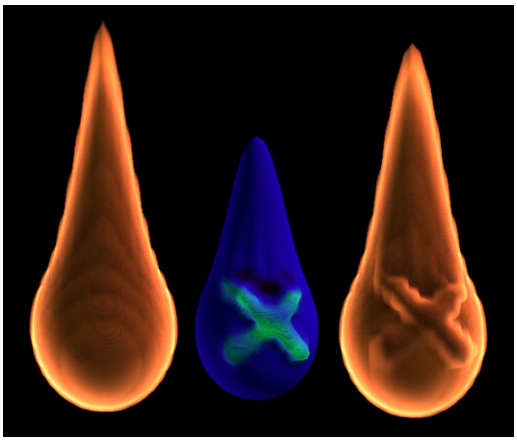

Fig. 6 This outlines the simplicity of painting forces onto the fire isosurface and the resulting change in the fire shape. The *green paint* shows fire growing outward and the *dark red paint* signifies pulling the fire back to emphasis the shape

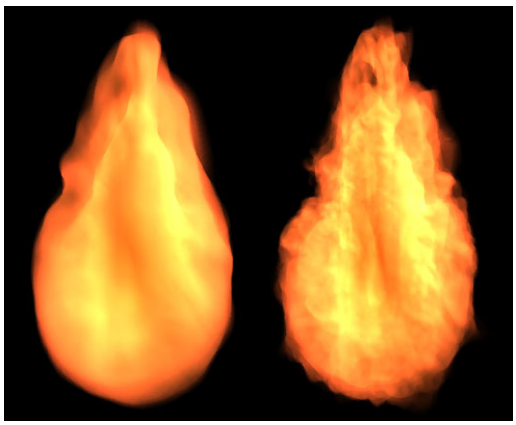

Fig. 7 Example of fire simulation with (*left*) and without (*right*) diffusion

then see the changes to the shape of the fire that are caused, make any necessary changes or undo mistakes using a simple backtracking system that stores previous version of the texture. It also reduces the number of simulation variables that the animator needs to understand and control as the majority of control is through the texturing system.

6 Incompressible Navier–Stokes solver

The fluid solver is based on Jos Stam's Stable Fluids [28] method using the basic semi-Lagrangian advection method and a Gauss–Seidel linear solver to find the solution to the divergence Poisson equation. Basic additions to the solver are outlined below.

Decrementing Temperature Advection is used to reduce the temperature of the fire as it moves away from the fire source toward a lower limit of zero. It uses the simple equation below where *burnRate* depends on the type of fuel being simulated and can be adjusted to increase the size of the overall fire.

$$temp_{t+1}(pos) = \max\big(temp_t(pos - \Delta t\, V_t) - burnRate, 0\big)$$

Buoyancy Forces apply an automatic upward motion dependent on the temperature of the fire which is similar to the approach of [8], assuming that the ambient temperature is zero. The equation below outlines this simple process that moves the fire upward.

$$f_{\text{buoyancy}}(i, j, k) = \begin{pmatrix} 0 \\ temp(i, j, k) \\ 0 \end{pmatrix}$$

Boundary Conditions control how the fire interacts with the boundary of the simulation domain. Open boundary conditions are used assuming that the source will remain inside the domain so that the temperature flows out of the domain.

Diffusion of the temperature field is not applied as it dampens out much of the interesting details of the fire as shown in Fig. 7.

6.1 Stability

The basic Stable Fluids solver gives stability through an advection scheme that only uses velocities that already exist in the system. Furthermore, the calculation of a divergent-free velocity system stops strange visual artefacts such as temperature "black holes." In order to preserve stability in the simulation, it is important that added forces do not push velocities too high such that the advection scheme begins jumping large distances and visual artefacts appear.

The normal and noise forces should not cause any unexpected behaviour as long as their magnitudes are not set too high. Normal forces just increase the natural growth of the fire and noise forces add basic turbulence at a thin surface inside the domain.

Therefore like any fluid simulation, the animator must not increase the external forces too drastically or visual artefacts will appear. When a more extreme effect is required by the animator, the animation step size should be altered so as to keep the forces low but increase the visible speed of the fire.

7 Implementation on GPU by CUDA

Using CUDA, the GPU can calculate the temperature and velocities of multiple cells at once allowing for real-time simulation. The issues of implementing our system onto the GPU are explained in the following subsections. We first explain about the memory allocation in Sect. 7.1. Several steps outlined above and in [28] were combined into a small set of kernels. Each kernel is executed once per grid cell in the

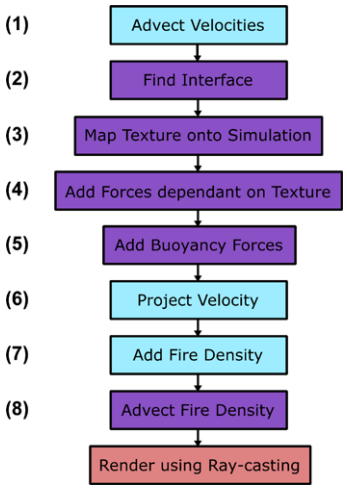

Fig. 8 The breakdown of the simulation with items numbered for Sect. 7. *Blue* tasks are standard tasks used in general fluid simulations. *Purple* tasks are specific for this fire simulation technique. The *red* rendering task is independent of the simulation methods

domain each time step and is given the basic variables of the simulation as arguments. These kernels are explain in Sects. 7.2 and 7.3. Finally, we discuss about issues about the memory transfer in Sect. 7.4.

7.1 Memory allocations

Our implementation makes use of CUDA textures as they give optimised access patterns for spatially-similar accesses and fast linear interpolation. However because of the CUDA memory layout, kernels cannot directly write to textures. Therefore kernels write to global memory which is then copied very quickly to a CUDA array bound to a texture.

3D textures are required for the temperature field, the velocity field and appropriate texture space is needed for the force texture. Additional memory space is required for the writable global memory for the textures, plus for storing divergence and intermediate results during the divergence calculation.

7.2 Simulation kernels

The *Vector Animate* kernel executes vector advection and calculates additional forces including from the textured isosurface (which are steps 1–5 in Fig. 8). For each cell, it calculates the vector at the next time step, then adds the buoyancy forces. It then calculates what additional normal and noise forces should be applied depending on if the cell lies on the isosurface and the corresponding texture value.

The *Temperature Animate* kernel executes temperature advection and adding sources (which are steps 7 and 8 in Fig. 8). It executes the simple decreasing advection and then checks to see if the cell is inside a source. This checks against a 3D texture that contains the various locations of the fire sources or can be hard-coded for speed.

The linear solver (covering step 6 in Fig. 8) requires three different kernels for *Divergence Calculation*, *Poisson Solving*, and *Setting Zero Divergence*. A GPU implementation of a similar linear solver is outlined in [5].

7.3 Interaction kernels

In order to allow for real-time interaction with the force textures, several additional kernels were used. The kernel used to the render the fire isosurface stores the 3D location of the isosurface for each on-screen pixel to a separate buffer. When the user clicks on the surface, the location of the click is retrieved and passed to the relevant *Brush* kernel for the 2D or 3D approach. These kernels are executed over the relevant section of the force texture and find texels within the radius of effect and increment/decrement their values.

7.4 Data transfer over the memory

The costliest part of a GPU-based application is the memory transfer to and from the GPU. Our implementation only uses memory transfers during the initiation and shutdown of the application to load and save textures. A set of kernels is used to set up the initial conditions for the simulation. Once the simulation has been executed, the same memory location is then used by the renderer also executed on the GPU. The textures are uploaded from the CPU to the GPU and then edited manually on the GPU as required by the animator. Additional keyframes are also stored on the GPU within the limits of the GPU memory. Otherwise, keyframes are copied asynchronously to and from the CPU so that the user does not notice any visible latency. High shared memory usage and careful global memory accesses reduce the execution time of the threads greatly. For general CUDA optimisation guidelines, the reader is pointed toward [23].

8 Results

Various examples of fire were created using 2D and 3D textures. Some examples were created from keyframe textures, and some were designed by an animator.

By utilising a 2D texture that varies over time, the shape and turbulence of the fire can be adapted from frame-to-frame. One approach offsets the texture coordinate by a particular value every time step which creates the effect of a continuous moving texture (assuming that the texture was wrapped) and causes the shape and turbulence applied across the fire to be constantly changing. This is best used for general patterns such as in Fig. 9 which applies a striped texture with an increasing V coordinate such that in appears to move upward.

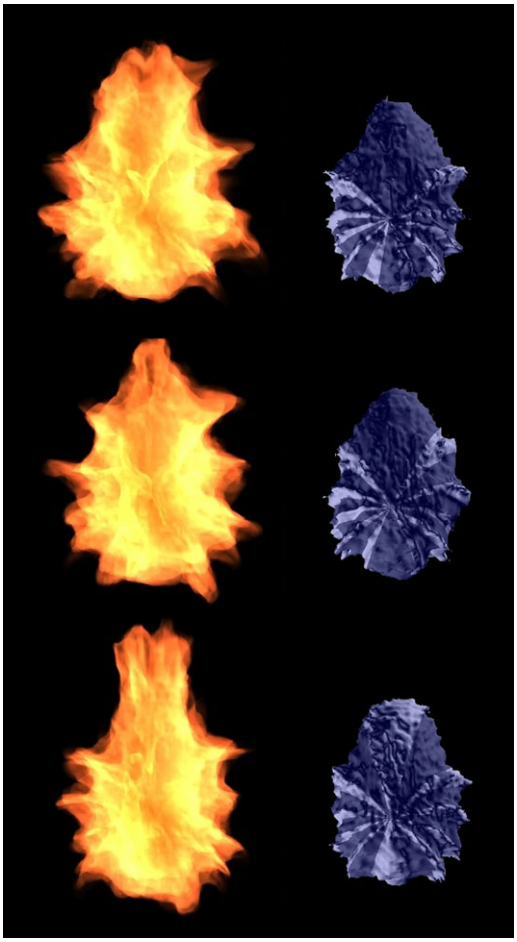

Fig. 9 Three frames from an animation of a striped texture which is moved upward through the flame by incrementing the V texture coordinate

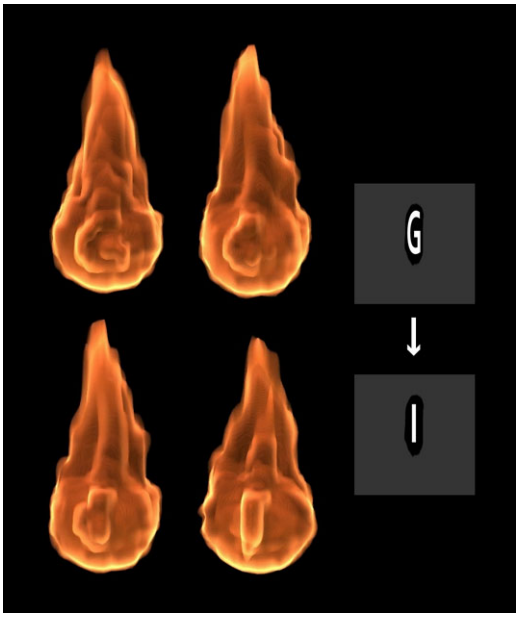

Fig. 10 A keyframe animation of letters blended from one texture to another

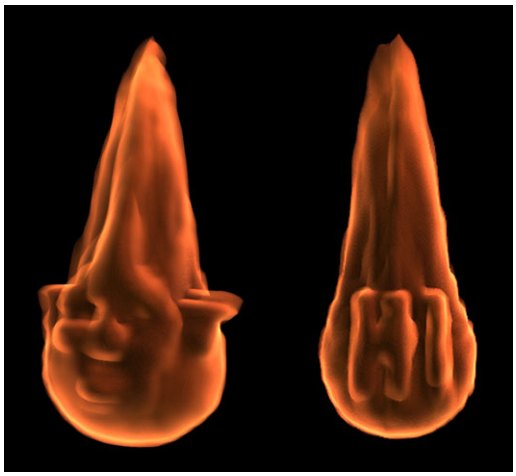

Fig. 11 Two examples of shapes growing out from the surface of the fire using forces painting onto the surface of the fire and saved into a 3D texture

An alternative approach uses a 3D texture to store a set of 2D animation frames as slices. Then to animate the feature, the texture lookups incrementally move deeper through the texture going from frame to frame. By using a 3D texture, hardware linear interpolation can be used to blend between frames smoothly. An example of animating letters modelled by 2D textures blended from one to another are shown in Fig. 10.

Finally, we asked an animator to design some fires using our system. The animator used the 3D texture interface to design a fire of a character face and one with letters on it (Fig. 11). The animator, who had little knowledge of the working system, commented on the intuitive aspect of the interface especially given that the images were produced in under fifteen minutes.

The figures shown throughout the paper are simulated in a $64 \times 64 \times 64$ domain. The textures used were generated using the GPL licensed graphics tool Blender [3]. The images are rendered using a volume rendering approach based on [15] and [5]. Two different rendering colourings were used to best display the shape of the fire for Figs. 1, 6, 10, and 11 and the turbulence effect in Figs. 3, 5, 7, and 9.

The system was implemented using CUDA and executed on an NVIDIA Quadro 5800 graphics card with 4 GB memory. Figure 12 shows the achievable frame rates for simulations using an animated texture. The additional cost of animating the texture is negligible. Even though 3D texturing requires more texture accesses, the actual simulation speed difference is minimal.

Domain size	Steps (sec)
32	638
64	142
96	44.7
128	19.2
160	10.0

Fig. 12 Resulting execution speeds in simulations steps per second for cube domains on an NVIDIA Quadro 5800

9 Discussion

Many different approaches are taken to animating the movement of fluids in order to control their shape. In order to create proper visible flow, it can be concluded that the velocity field must be adapted, which is particularly challenging given the issues in visualising a 3D velocity field. Textured forces offer a new model for interaction with the velocity field while maintaining visually impressive results.

Furthermore, instead of using a system that requires much trial and error, such as particle based approaches, to find the perfect result this allows the animator to interactively edit the fire during the animation and control the resulting image based on physical simulation. By working directly with the rendered fire, the animator can easily understand the effects of the controls and quickly adjust the simulation toward the desired effect.

A few of the effects shown in the figures could possibly also be achieved through careful addition of sources inside the simulation. Instead of adjusting the forces, other approaches would simply create new sources where the fire was to grow. However, that approach will lack the visible flow of the fire and animating between different shapes of fire remains a significant challenge. Our approach gives both an intuitive model of fire growth for the animator, and an extremely simple way of blending between two different patterns while maintaining realistic fire flow.

Furthermore, the ability to control the fire using textures is highly dependent on the resolution of the texture and also the size of the simulation domain. These techniques could be improved by linking with a subgrid noise approach such as [16] and [21] in order to improve the visual quality of the image. Linear interpolation is used given its fast hardware implementation but can cause visual degradation. An improved interpolation technique would improve the texture mapping quality and could also allow for more complicated transitions between frames.

2D and 3D textures can be used together to gain further control over the fire. 2D textures which offer simple fire shapes and a very easy animation approach can control the noise or predominantly the normal forces in the fire at the interface. Meanwhile, a 3D texture can grow the fire into more specific shapes by applying controlled normal forces.

10 Conclusion and future work

This paper introduces a real-time technique that allows easy control of the shape and turbulence of a fire using textured forces.

Our approach uses a 2D or 3D texture defined across the simulation to control the growth and turbulence of the fire. Trigonometric based turbulence and buoyancy forces are added to a stable fluid solver in order to create a realistic fire effect.

These techniques integrate easily into a normal fluid solver and can be linked with an appropriate real-time volume rendering system. It requires minimal additional memory and the extra memory accesses are easily optimised so that the full simulation and rendering can execute in real-time.

This texturing approach offers many advantages over other techniques for animating fire. Most artists already have a deep knowledge of texturing and imaging software that will make this simulation much easier to understand. Furthermore, with minor editing, existing texture assets can be imported to be used as fire animations. Moreover, animating the fire uses the same principals as animating drawn frames. An interactive interface allowing the user to directly influence the animation of the fire in real-time is of real importance for increasing animator work flow and gaining better results from physical simulations.

Creating arbitrary shapes of fire is a challenging problem compared to other fluids such as smoke and water, as done by other researchers [7, 20, 27, 33]. This technique allows for specific local control of the growth and turbulence of the fire and will provide an aesthetically pleasing global shape. However, this requires careful control by the artist so that additional turbulence created by the noise forces does not overwhelm the shape created using the normal forces. Automatically adjusting the normal forces to interact better with noise forces is one of our future research topics.

Acknowledgement The authors thank the anonymous reviewers for their constructive comments. This work was partly funded by EU IST-FP7-IP TOMSY (Ref:270436) and EPSRC (EP/H012338/1).

References

1. Bargteil, A.W., Sin, F., Michaels, J.E., Goktekin, T.G., O'Brien, J.F.: A texture synthesis method for liquid animations. In: Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Sept (2006)
2. Beaudoin, P., Paquet, S., Poulin, P.: Realistic and controllable fire simulation. In: Proc. of Graphics Interface 2001, pp. 159–166 (2001)
3. Foundation, B.: Blender 2.48 (2009). www.blender.org
4. Chiba, N., Ohkawa, S., Muraoka, K., Miura, M.: Twodimensional visual simulation of flames, smoke and the spread of fire. J. Vis. Comput. Animat. **5**(1), 37–54 (1994)

5. Crane, K., Llamas, I., Tariq, S.: Real-time simulation and rendering of 3D fluids. In: GPU Gems, vol. 3, pp. 633–675. NVIDIA (2007)
6. Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: Texturing and Modeling: A Procedural Approach. Morgan Kaufmann, San Francisco (2002)
7. Fattal, R., Lischinski, D.: Target-driven smoke animation. ACM Trans. Graph. **23**(3) (2004)
8. Fedkiw, R., Stam, J., Jensen, H.W.: Visual simulation of smoke. In: Proc. SIGGRAPH'01 (2001)
9. Foster, N., Fedkiw, R.: Practical animation of liquids. In: Proc. SIGGRAPH'01, pp. 23–30. ACM, New York (2001)
10. Foster, N., Metaxas, D.: Controlling fluid animation. In: Proc. of Computer Graphics International, pp. 178–188 (1997)
11. Foster, N., Metaxas, D.: Modeling the motion of a hot, turbulent gas. In: Proc. SIGGRAPH'97, pp. 181–188 (1997)
12. Harris, M.: Fast fluid dynamics simulation on the GPU. In: GPU Gems, pp. 637–665. NVIDIA (2004)
13. Hong, J.-M., Shinar, T., Fedkiw, R.: Wrinkled flames and cellular patterns. In: Proc. SIGGRAPH'07 (2007)
14. Horvath, C., Geiger, W.: Directable, high-resolution simulation of fire on the CPU. ACM Trans. Graph. **28**(3), 1–8 (2009)
15. Ikits, M., Kniss, J., Lefohn, A., Hansen, C.: Volume rendering techniques. In: GPU Gems, pp. 667–692. NVIDIA (2004)
16. Kim, T., Thürey, N., James, D., Gross, M.: Wavelet turbulence for fluid simulation. ACM Trans. Graph. **27**(3), 1–6 (2008)
17. Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N., Carlson, M., Lin, M.: Texturing fluids. IEEE Trans. Vis. Comput. Graph. **13**(5), 939–952 (2007)
18. Lamorlette, A., Foster, N.: Structural modeling of flames for a production environment. In: Proc. SIGGRAPH'02, pp. 729–735 (2002)
19. Lorensen, W.E., Cline, H.E.: Marching cubes: a high resolution 3D surface construction algorithm. SIGGRAPH Comput. Graph. **21**(4), 163–169 (1987)
20. McNamara, A., Treuille, A., Popovic', Z., Stam, J.: Fluid control using the adjoint method. ACM Trans. Graph. (SIGGRAPH 2004), **23**(3) 449–456 (2004)
21. Narain, R., Sewall, J., Carlson, M., Lin, M.C.: Fast animation of turbulence using energy transport and procedural synthesis. ACM Trans. Graph. **27**(5), 1–8 (2008)
22. Nguyen, D.Q., Fedkiw, R., Jensen, H.W.: Physically based modeling and animation of fire. In: Proc. SIGGRAPH'02 (2002)
23. NVIDIA: CUDA Good Programming Guide (2009)
24. Perry, C.H., Picard, R.W.: Synthesizing flames and their spreading. In: Fifth Eurographics Workshop on Animation and Simulation, pp. 105–117 (1994)
25. Reeves, W.T.: Particle systems a technique for modeling a class of fuzzy objects. ACM Trans. Graph. **2**, 91–108 (1983)
26. Schechter, H., Bridson, R.: Evolving sub-grid turbulence for smoke animation. In: Proc. of 2008 ACM/Eurographics Symposium on Computer Animation (2008)
27. Shi, L., Yu, Y.: Controllable smoke animation with guiding objects. ACM Trans. Graph. **24**(1), 140–164 (2005)
28. Stam, J.: Stable fluids. In: Proc. SIGGRAPH'99 (1999)
29. Stam, J., Fiume, E.: Turbulent wind fields for gaseous phenomena. In: Prof. SIGGRAPH'93, pp. 369–376 (1993)
30. Stam, J., Fiume, E.: Depicting fire and other gaseous phenomena using diffusion processes. In: Proc. SIGGRAPH'95, pp. 129–136 (1995)
31. Takahashi, J., Takahashi, H., Chiba, N.: Image synthesis of flickering scenes including simulated flames. In: IEICE Transactions on Information Systems E80-D(11), pp. 1102–1108 (1997)
32. Todo, H., Anjyo, K.-i., Baxter, W., Igarashi, T.: Locally controllable stylized shading. ACM Trans. Graph. **26**, 17 (2007)
33. Treuille, A., McNamara, A., Popovic', Z., Stam, J.: Keyframe control of smoke simulations. ACM Trans. Graph. **22**(3), 716–723 (2003)

Jake Lever completed a Bachelors of Software Engineering at the University of Edinburgh in 2009 with a successful year at the California Institute of Technology. His varied work has included Harvard's Connectome Project, marine research in the South Pacific, and emulation of visual problems in children at Glasgow's Yorkhill Children's Hospital.

Taku Komura is currently a Reader in the School of Informatics, The University of Edinburgh. Before joining The University of Edinburgh on 2006, he worked as an Assistant Professor at City University of Hong Kong and as a postdoctoral researcher at RIKEN, Japan. He received his Ph.D. (2000), M.Sc. (1997), and B.Sc. (1995) in Information Science from the University of Tokyo. His research interests include human motion analysis and synthesis, physically-based animation, and real-time computer graphics. His research area covers computer graphics, robotics, and biomechanics.