The 7th International Symposium on Virtual Reality, Archaeology and Cultural Heritage
VAST (2006)
M. Ioannides, D. Arnold, F. Niccolucci, K. Mania (Editors)

Real-time Animation of Various Flame Shapes

F. Bridault[1] F. Rousselle[1] C. Renaud[1] M. Leblond[1]

[1]Laboratoire d'Informatique du Littoral

Abstract

Working on the computer reconstruction of the Gallo-Roman forum of Bavay, we try to improve the feeling of immersion in the virtual environment. One way to achieve this is to provide realistic and dynamic light sources. In this context, we need to model candles, oil lamps, torches or bonfires. We propose in this paper a model that can handle complex flames in real-time and manage interactivity. The fire is considered as a set of linear flames whose shapes are defined by the geometry of the combustible and the fuel distribution. Each individual flame is represented by a textured NURBS surface. Then, combining several real-time effects such as glow and true transparency, we are able to make the NURBS surfaces merge in a convincing way, and to give the impression of a real fire.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computing Methodologies]: Computer Graphics Three-Dimensional Graphics and Realism

1. Introduction

Fire is a challenging problem in computer graphics. Describing flames is of primary interest for movie-making, animated films and video games. This can also be important for virtual environments, especially when these are reconstructions of ancient sites. Indeed, fire was the only source of light until the end of the 19th century. The lack of fire in such virtual environments would decrease realism, and would reduce the possibility of representing scenes in the night or allowing real-time walkthroughs in dark rooms.

An example of such application is the *CyberForum* project which aims to reconstruct and to visualize interactively the Gallo-Roman forum of *Bagacum* (Bavay) in the north of France. This building was built between Flavian (70 A.C.) and Severian (beginning of 3rd century) periods and its impressive dimensions (240 by 110 meters) make this forum the largest known in Gaul to this day. Due to the damage caused over time it is often difficult for any visitor to understand the architectural design of the site. The virtual reconstruction of the different buildings of the forum was thus performed two years ago and is extensively used for visitors through a 3D stereo interactive movie (figure 1). It is also employed by archaeologists as a research tool in order to understand some parts of the site or to compare some of their hypotheses. They are therefore interested in visualizing some parts of the forum lit by flames in order to make better investigations, as it was done in Knossos [IC03]. For the visitors, the addition of real-time flames can improve the feeling of immersion in the virtual site.

Figure 1: *Interactive 3D stereoscopic movie and projection room (courtesy of musée/site archéologique départemental de Bavay; A.Solé 2006)*

In this paper, we introduce an extension of a real-time flame model [BLLRR06] that can handle complex flames thanks to the use of multiple NURBS surfaces. We also present a new intuitive and interactive way to build flames

with *virtual wicks* and to model the fuel distribution. Last, we outline the combination of existing multiple real-time rendering techniques to get a realistic appearance.

After words the paper is organized as follows. We first talk about previous work in section 2. Then, section 3 summarizes the mechanism of the flame model we proposed in [BLLRR06], which is the basis of our method. We describe extensively our approach to the modeling of complex flames in section 4, while section 5 presents the techniques we used for the rendering. Section 6 shows our current results and we finally give and discuss some perspectives in section 7.

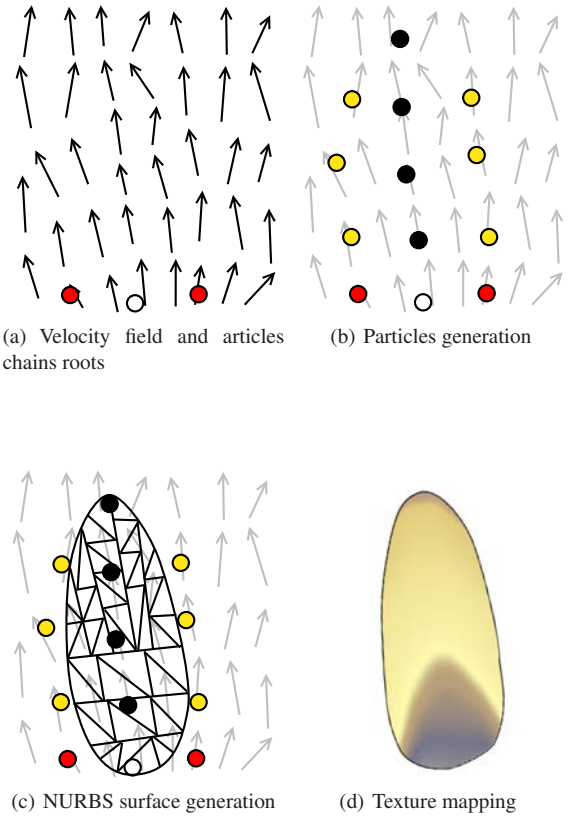
2. Previous Work

The literature contains many techniques to display flames on computers, but to our knowledge, none was really suitable for real-time virtual environments. The first kind of technique uses physics to describe the combustion process. This was done first with simple laminar flames, first static [Ina90], then dynamic [Rac96]. Diffusion equation [SF95] could also be employed, although it implies a high number of parameters, to describe larger turbulent fire. Nguyen *et al.* [NFJ02] use two incompressible Euler equations in addition to a method known as the level-set equation to build an implicit surface representing the blue core. Rendering is done according to the model of the black-body radiation which leads to a very realistic simulation. Lamorlette *et al.* [LF02] also published a realistic model designed for production environments. This model is also interesting since it is able to describe a wide range of flame types, from a simple candle to a dragon's breath. The obvious drawback for all these methods is the computation time, making them unsuitable for our real-time constraint.

Other methods therefore try to avoid physics and make approximations in order to obtain a better computation time. Particle systems [Ree83] are widely used. They make it possible to describe fuzzy objects with a low computing cost. Beaudouin *et al.* [BPP01] use particle chains evolving in a velocity field and describe a single flame using a potential equation. Pszczolkowska [Psz04] also applies a similar method, adding Perlin's noise [Per85] to model turbulence effects, and a Gauss function instead of a potential equation. Nevertheless, neither of them reach real-time frame rate.

Wei *et al.* [WLMK02] introduced a GPU implementation of the Lattice Boltzmann Model to model air flows. They release particles in the velocity field and then render the fire with textured splats. This model is fast and is able to achieve true real-time simulation and animation, however the fire looks more like a burned gas cluster than real flames.

Rather than simulating the flames, some approaches capture real flames data. In the context of photo-realistic illumination by flames for virtual heritage, Chalmers *et al.* [DC01, Cha02] measure a candle flame illumination using a spectroradiometer. A video-captured flame is then incorporated into a virtual scene and its illumination is computed by approximating the flame shape with several emitting spheres. Later, Hasinoff and Kutulakos [HK03] developed an approach to reconstruct a 3D flame from two 2D orthogonal views. Recently Ihrke and Magnor [IM04] presented a tomographic method for reconstructing a volumetric model of flames from multiple camera images. Although these models are of high quality, they are of course restricted to non interactive animations because they rely on a static set of real images.



(a) Velocity field and articles chains roots (b) Particles generation

(c) NURBS surface generation (d) Texture mapping

Figure 2: *Mechanism of our previous model for a candle (side view)*

3. A simple model

Considering no existing model fits the needs for virtual environments, a real-time model for flames was introduced in our previous work [BLLRR06]. Our goals were to simulate accurately the dynamic of simple flames and to render them realistically in real-time. We used a Navier-Stokes equations solver based on the well-known implementation of Stam [Sta99, Sta03] to create a velocity field. External forces were added at the bottom of the grid to simulate the buoyancy. Particle chains were generated in the velocity field

from generation points called *roots* (figure 2(a)). Two kinds of chains were created : the *lead skeletons* and the *peripheral skeletons* (figure 2(b)). The former define the vertical boundaries and the latter the horizontal boundaries of the flame. The particles from these chains were used as control points to build a NURBS surface which approximates the shape of the flame (figure 2(c)). Last, a transparent 2D texture was mapped onto the surface to render the colors of the flame (figure 2(d)).

This method was quite fast and yet had several limitations. Above all, it was restricted to small flames and could not produce complex flames such as torches or camp fires. Then it lacked a simple way to handle the buoyancy of the flames. Forces were defined at the bottom of the flame but there was no understandable model to describe them, although it is crucial to have one, because in this way a simple interface can be defined for the potential users of such a technique. Last, rendering of the flames could be improved.

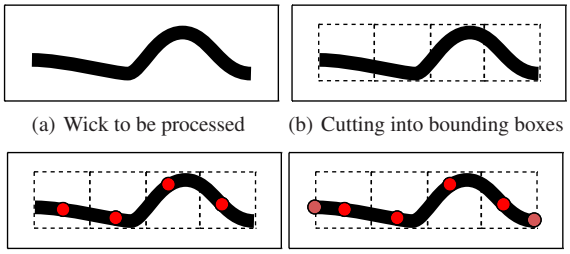
4. Towards complex flame modeling

We introduce several concepts here. First we consider a *virtual wick* for a linear flame. Then we associate a *Fuel Distribution Function* (FDF) with the wick in order to compute the buoyancy of the flame. That being done, we generate a NURBS surface for the wick in the velocity field. Last, to build a more complex fire such as torches or bonfires, we consider a set of flames.

4.1. The virtual wick

As observed during our experiments, the shape of the wick has an impact on the final shape of the flame, that's why it is important to represent it. Actually, the wicks we define should be considered as an interactive tool for the user to place fire fronts and to draw somehow a flame. Indeed, it will be shown later that many wicks are employed to define larger flames although this does not really correspond to reality. Therefore the wicks themselves can be displayed or not according to the type of fire being rendered. The wicks we use are generally long cylindrical shapes as seen in figure 3(a). They can be modeled in any 3D modeler and then imported in our software to be used as a source of fire.

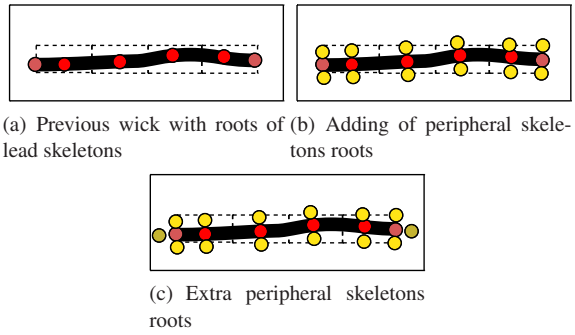
Whereas the roots of the particles chains were placed by hand in our previous work, we here have to set a method to place them automatically on the wicks. The wick in bounding boxes is divided according to the number n_l of lead skeletons we want (figure 3(b)). Generally, two or three lead skeletons are used per flame. Then we compute the barycenter of the vertices in each bounding boxes. This is where the roots of the lead skeletons are placed (figure 3(c)). In addition, two extra lead skeletons roots are added on each extremity of the wick, so that there is a total of $N_l = n_l + 2$ lead skeletons (figures 3(d) and 4(a))). After that, two roots of peripheral skeletons are placed on each side of a lead



(a) Wick to be processed (b) Cutting into bounding boxes

(c) Adding of lead skeletons roots (d) Extra lead skeletons roots

Figure 3: *Placement of the roots of the lead skeletons on the wick (side view)*



(a) Previous wick with roots of lead skeletons (b) Adding of peripheral skeletons roots

(c) Extra peripheral skeletons roots

Figure 4: *Placement of the roots of the peripheral skeletons on the wick (top view)*

skeleton root, on the wick sides (figure 4(b)). Two extra peripheral skeletons roots are also added at the extremities of the wick (figure 4(c)). This way, there is therefore a total of $N_p = N_l \times 2 + 2$ peripheral skeletons.

4.2. Fuel Distribution Function

The shape of the wick is not sufficient to describe the shape of the flame; fuel is in fact one of the most important factors. In the real world, wood, oil or wax don't produce the same kind of fire. But we can also notice that the fuel is not spread uniformly on the burning surface. In oil lamps, the capillary action in the wick is the main factor. However our real-time constraint does not permit us to perform complex computations. Therefore, we need a simple process to model the fuel distribution.

Buoyancy is simulated by adding external forces in the velocity field of the solver. A simple function $F(u_i)$ is used to describe their distribution. Each root of the lead skeletons is associated with a value u_i in the range $[-1;1]$ or $[0;1]$ according to the type of function. Considering the wick in one dimension, the value of the root *on the extreme left* u_0, is set to -1 or 0 and *on the extreme right* u_{n-1} is set to 1. The other values $u_1 ... u_{n-2}$ are linearly interpolated.

$$A = \begin{pmatrix}
L_{0,0} & P_{0,0} & P_{1,0} & \cdots & P_{M-2,0} & P_{M-1,0} & L_{M-1,0} \\
L_{0,0} & P_{0,1} & P_{1,1} & \cdots & P_{M-2,1} & P_{M-1,1} & L_{M-1,0} \\
L_{0,1} & P_{0,2} & P_{1,2} & \cdots & P_{M-2,2} & P_{M-1,2} & L_{M-1,1} \\
L_{0,2} & P_{0,3} & P_{1,3} & \cdots & P_{M-2,3} & P_{M-1,3} & L_{M-1,2} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
L_{0,N_l-2} & P_{0,N_p/2-3} & P_{1,N_p/2-3} & \cdots & P_{M-2,N_p/2-3} & P_{M-1,N_p/2-3} & L_{M-1,N_l-2} \\
L_{0,N_l-1} & P_{0,N_p/2-2} & P_{1,N_p/2-2} & \cdots & P_{M-2,N_p/2-2} & P_{M-1,N_p/2-2} & L_{M-1,N_l-1} \\
L_{0,N_l-1} & P_{0,N_p/2-1} & P_{1,N_p/2-1} & \cdots & P_{M-2,N_p/2-1} & P_{M-1,N_p/2-1} & L_{M-1,N_l-1} \\
L_{0,N_l-1} & P_{0,N_p/2} & P_{1,N_p/2} & \cdots & P_{M-2,N_p/2} & P_{M-1,N_p/2} & L_{M-1,N_l-1} \\
L_{0,N_l-2} & P_{0,N_p/2+1} & P_{1,N_p/2+1} & \cdots & P_{M-2,N_p/2+1} & P_{M-1,N_p/2+1} & L_{M-1,N_l-2} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
L_{0,2} & P_{0,N_p-3} & P_{1,N_p-3} & \cdots & P_{M-2,N_p-3} & P_{M-1,N_p-3} & L_{M-1,2} \\
L_{0,1} & P_{0,N_p-2} & P_{1,N_p-2} & \cdots & P_{M-2,N_p-2} & P_{M-1,N_p-2} & L_{M-1,1} \\
L_{0,0} & P_{0,N_p-1} & P_{1,N_p-1} & \cdots & P_{M-2,N_p-1} & P_{M-1,N_p-1} & L_{M-1,0}
\end{pmatrix} \quad (1)$$

The fuel distribution function $F(u_i)$ itself can be any of the following : linear, bilinear, exponential, Gaussian, random. Of course it can also be any user-defined function. When we add the forces corresponding to the buoyancy of the flame, we put a vertical force in the voxel of the solver grid where each root is placed. The force f_i is finally computed as :

$$f_i = F(u_i).C + F_{ext}(u_i) \quad (2)$$

where C is a constant factor that makes it possible to scale the height of the flame. So the flame profile will have a similar shape to this of the function F. F_{ext} is an additional forces function that adds a swinging effect. It can be a periodical function, noise or random function that allows some animation to be added and emphasizes the fact that this is a dynamic light.

4.3. Linear flames

The flame is built in a similar manner to the simple flame model, using the standard GLU NURBS interface. The control points matrix of the NURBS surface is composed of all the particles of the peripheral skeletons and the lowest and the highest particles of the lead skeletons. There is a line in the matrix for each peripheral skeleton. Its size is thus $N_p \times (M+2)$ where M is the number of particles in one skeleton. Each line of this matrix starts with the lowest particle of the nearest lead skeleton, then takes all particles of one peripheral skeleton and ends with the highest particle of the same lead skeleton. Denoting m the index of the particle in the skeleton, n the index of the skeleton, $L_{m,n}$ the particles of the lead skeletons and $P_{m,n}$ the particles of the peripheral skeletons, we can write the matrix A as shown in equation 1.

It can be noticed that peripheral skeletons $0, 1, N_p - 1$ reference the same lead skeleton because it is the nearest one for all of them. Peripheral skeletons $N_p/2-2, N_p/2-1, N_p/2$ also do so for the same reason. Figure 5 shows different flame profiles generated by different FDFs.

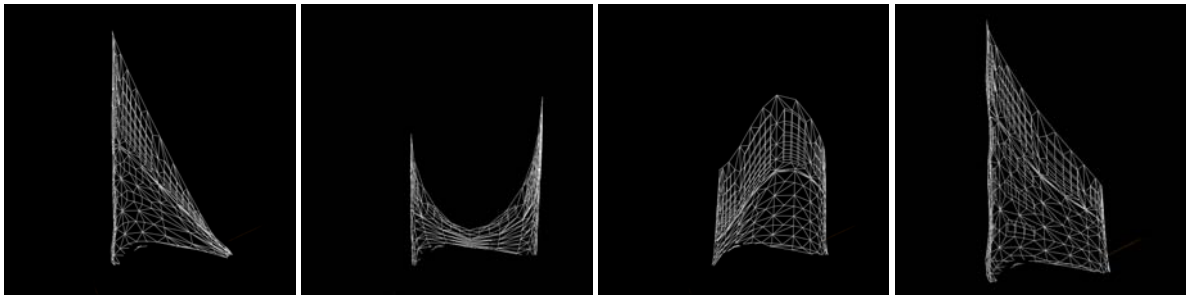
4.4. Complex flames

To build complex flames with NURBS surfaces, we assumed that any fire is a set of several independent flames. Indeed it seemed very difficult to represent a whole fire with a single animated NURBS surface. It's easier to achieve that with many smaller flames. Therefore any fire is defined by several linear flames.

An important point to understand is that we are not going to do anything on the geometry to merge the flames. That would be hard to do and above all very time-consuming. Thus any linear flame is built and rendered independently, and we will deal with their merging in a post-rendering phase described in the next section. Thus the wicks can be placed arbitrarily, and it will look even better if they are overlapping. Figure 6(a) shows a typical layout for the bottom of a torch and figure 6(b) for a campfire.

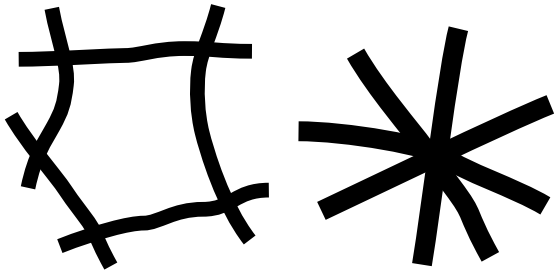
5. Rendering

We cannot rely only on 2D texture mapping to render multiple NURBS surfaces. As we said previously, the flames overlap, so the first thing to do is to deal with their transparency. Moreover we want a smooth appearance as far as possible, and most of all overlapping flames should visually merge together. Also, an important effect is the glow of the flame, because it is the only way to distinguish bright sources of light [NKON90].



(a) Linear (b) Bilinear (c) Gauss (d) Exponential

Figure 5: *Various linear flame profiles according to different FDFs with three lead skeletons*



(a) Torch model (b) Campfire model

Figure 6: *Top view of typical placements of the wicks*

5.1. True transparency

In the standard rendering pipeline of GPUs, we have to use alpha blending and sort the objects according to their depth from the back to the front if we want to see all the flames with transparency. Nevertheless this becomes quite tricky with our flames because they overlap and it is therefore often difficult to choose the first flame to be rendered.

That's why we implemented the method called *Depth Peeling*, which makes it possible to render objects with true transparency regardless of the order in n-passes. This was first realized using Virtual Pixel Maps [Mam89] and Dual Depth Buffers [Die96]. Later, graphics hardware functionalities allowed the technique [Eve02] to be accelerated. To display n transparency layers, the method uses n passes. The first pass retrieves the first depth layer with the usual depth test. The following passes extract the next layers using the depth from the previous pass to eliminate the previous depth layers.

The simplest way to explain its GPU implementation is to consider two depth buffers and two depth tests. The first depth test is the usual GL_LESS or GL_LEQUAL testing which keeps the current fragment if it is the nearest to the viewer. The second depth test rejects the active fragment if it is nearer than the depth of the pixel rendered at the previous pass. The first pass consists in rendering the scene with only the first test enabled, while the first depth buffer is saved in the second depth buffer. The next passes render the scene with both depth tests enabled. Each depth layer is rendered using additive blending.

With current graphics hardware, this can be done quite easily. We have used ARB_fragment_shadow extension to perform the second depth test quickly as well as the Frame Buffer Object (FBO) extension to avoid read-backs of the depth buffer, by attaching FBOs to depth textures. As we have now to render the flames many times for each frame, the cost of their rendering must be reduced. A straightforward way is to generate an OpenGL *display list* while rendering the NURBS surfaces in the first pass. Then this list can be used to render them in later passes.

However, our first results were slow because OpenGL kept doing the tessellation of the NURBS when using the display list. Indeed to really avoid tessellation, the NURBS surface must not be generated in GLU_NURBS_RENDERER mode. This has to be done in GLU_NURBS_TESSELLATOR mode, because that way we can define our own callback functions which will call the standard OpenGL functions to generate the vertices, normals and texture coordinates. That being done, the display list contains glVertex, glNormal, glTexCoord calls instead of the gluNurbsSurface calls. Thanks to that the frame rate drastically improved, for instance by a factor of four with four layers.

Last, the depth peeling algorithm has been slightly modified. Indeed, it assumes that all the scene is rendered in full transparency. But this is not exactly what we want, since we only want to apply it to the flames. Furthermore, rendering the whole scene would obviously slow down the process. Yet if the standard depth peeling algorithm is only applied to the flames, we will miss the occlusions of the other scene objects. When multiple flames are rendered, particular cases can even occur where an object in the scene is placed between two flames. That means that each pass of the depth peeling must perform a third depth test. We therefore pre-

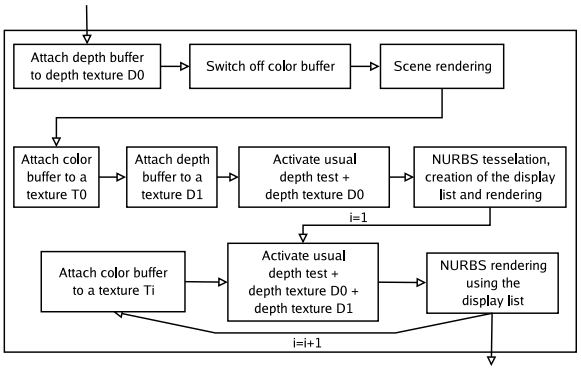


Figure 7: *Depth peeling process for flames*

render the scene without the flames in a FBO with color buffer writing disabled and depth buffer attached to a second depth texture. Thus a fragment is only displayed if it passes the two depth tests from the depth peeling and the depth test with this new depth texture. Figure 7 summarizes our implementation.

5.2. Glow

To render the glow of the flame, we implemented the technique described in [JO04]. It consists in a post-processing of the 2D rendering of the scene. Glowing objects are rendered separately. The frame buffer is rendered in a 2D texture at a lower resolution than the viewport. This texture is then filtered to produce a blur effect using a separable function, typically Gaussian. Thanks to this property, the blur can be performed in two passes. It is first performed in the x direction with n pixels and then in y with n pixels instead of $n \times n$ pixels in one pass. This makes it possible to do this on the GPU because this implies few texture lookups. Last the scene is rendered normally and the blurred texture is applied on top of it using additive alpha blending.

Our application of the algorithm is somehow similar. The following kernel is used to perform the blur in one direction :

$$K = \frac{e^{-x^2}}{\sigma^2} \tag{3}$$

where σ is equal to 1.5. These settings have proved to give good results. It is indeed important that the bandwidth is not too large, because it will produce too blurry a flame, and we would lose details in its texture.

However a blur at the same resolution as the viewport produced an interesting anti-aliasing effect, making the edges of the flame fairly smooth. We thus perform two different blurs. A first one is performed at the same resolution and a second at a lower resolution, generally four times smaller. The normal rendering of the flame is not used at all, we only render

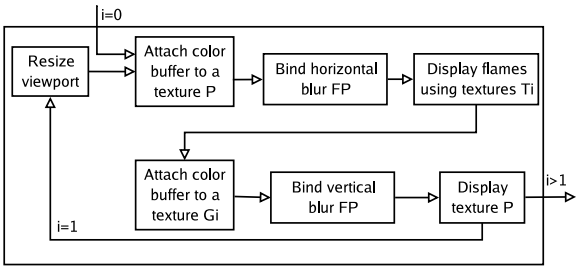


Figure 8: *Glow process for flames*

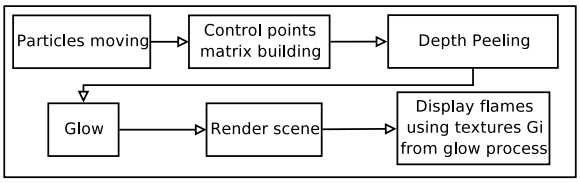


Figure 9: *Rendering process for flames*

the two blurred ones. Note that to make the glow process efficient, it is crucial to render the flames only once, especially in this context because we use depth peeling which requires n passes for each rendering. That's why the rendering is stored in a texture which is the input of both blur processes. Figure 8 summarizes the glow process while figure 9 shows the whole rendering process.

6. Results

The same parameters as in [BLLRR06] are used for the solver. A resolution of $15 \times 15 \times 15$ for the grid is indeed enough and allows us to keep real-time frame rates. We can employ as many skeletons as we want for each flame. The larger they are, the more skeletons we need, but of course the more time consuming it is. Between ten and sixteen skeletons per flame is a good compromise. The skeletons themselves have a height of less than nine particles. The benchmarks were performed on a Pentium-M 2 Ghz with a NVIDIA GeForce Go 6600 graphics card and the screen resolution is 1024×768 pixels.

Figure 11 compares the rendering with and without the post-rendering processes. This highlights that the normal texture of the flame is not very bright. Indeed, as additive alpha blending is used in the depth peeling and the glow process, we must avoid saturation. Nonetheless, by managing this accumulation of intensities carefully, we can create the merging effect we were looking for. Although the assumption of a set of linear flames is not physically correct, we can see that it is finally visually convincing.

On the contrary to our previous paper, we did not consider the spatial distribution of the light. Indeed, it is harder to capture a photometric solid for the flames described

Figure 10: *A Roman draper shop illuminated by an oil lamp and a candle (67752 polygons, 2 independent flames, 25 fps)*

here than for candles. Thus we used a standard per-pixel lighting, using the spectral properties of a real flame. Figures 13 shows a campfire and figure 12 a torch in the baker shop of the forum. Both models produce 45 fps. Without the post-rendering processes, they reach 57 fps. The glow costs around 3 fps and the depth peeling with four layers 9 fps. Small flames can also benefit from this new rendering process (figure 10). The appearance of flames is however much better when animated, videos are available on our web site `http://www-lil.univ-littoral.fr/~bridault`.

7. Conclusion

We have presented a model that can handle complex flame rendering and animation in real time. The use of virtual wicks allows the user to put and merge flames onto any object easily. Far from here we have used our own software, but the NURBS surfaces used for modeling are available in standard graphics APIs. Thus our model can be integrated into any real-time application. The careful use of depth peeling and glow techniques helps to improve the realism of the flame rendering considerably, especially for complex flames like camp fires or torches that may appear in ancient illumination sources.

Future work will focus on different topics arising from flame simulation. Firstly we will study the problem of managing several flame-based sources. Indeed the fluid dynamic solver is still a bottleneck due to its high computational requirement. As soon as several independent sources are used, several independent solvers will have to be run, accordingly reducing the frame rate. We didn't deal with shadows rendering here yet, but as we stated in our previous paper, our model is compatible with common techniques. Smoke simulation involves finding a way to render its main curls and to consider participating media in order to simulate its propagation through the scene. Finally fire propagation could be simulated by adding a temporal dimension to our virtual wicks allowing them to be animated.

References

[BLLRR06] BRIDAULT-LOUCHEZ F., LEBLOND M., ROUSSELLE F., RENAUD C.: Enhanced illumination of reconstructed dynamic environments using a real-time flame model. In *Afrigraph '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (New York, NY, USA, 2006), ACM Press, pp. 31–40.

[BPP01] BEAUDOIN P., PAQUET S., POULIN P.: Realistic and controllable fire simulation. In *Proceedings of Graphics Interface 2001* (2001), Watson B., Buchanan J. W., (Eds.), pp. 159–166.

[Cha02] CHALMERS A.: Very realistic graphics for visualising archaeological site reconstructions. In *Proceedings of the 18th spring conference on Computer graphics* (Avril 2002), ACM Press, pp. 43–48.

[DC01] DEVLIN K., CHALMERS A.: Realistic visualisation of the pompeii frescoes. In *Proceedings of the 1st international conference on Computer graphics, virtual reality and visualisation* (2001), ACM Press, pp. 43–48.

[Die96] DIEFENBACH P. J.: *Pipeline rendering: interaction and realism through hardware-based multi-pass rendering*. PhD thesis, Philadelphia, PA, USA, 1996.

[Eve02] EVERITT C.: Interactive order-independent transparency. tech. rep., 2002.

[HK03] HASINOFF S. W., KUTULAKOS K. N.: Photo-consistent 3d fire by flame-sheet decomposition. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision* (Washington, DC, USA, 2003), IEEE Computer Society, p. 1184.

[IC03] IOANNIS R., CHALMERS A.: High fidelity lighting of knossos. In *VAST2003: 4th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage* (2003), ACM SIGGRAPH Publications.

[IM04] IHRKE I., MAGNOR M.: Image-based tomographic reconstruction of flames. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2004), ACM Press, pp. 365–373.

[Ina90] INAKAGE M.: A simple model of flames. In *Proceedings of the eighth international conference of the Computer Graphics Society on CG International '90: computer graphics around the world* (Institute of Systems Science, Singapore, july 1990), pp. 71–81.

[JO04] JAMES G., O'RORKE J.: Real-time glow. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics* (2004), 343–361.

[LF02] LAMORLETTE A., FOSTER N.: Structural modeling of flames for a production environment. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 729–735.

[Mam89] MAMMEN A.: Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl. 9*, 4 (1989), 43–55.

[NFJ02] NGUYEN D. Q., FEDKIW R., JENSEN H. W.: Physically based modeling and animation of fire. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 721–728.

[NKON90] NAKAMAE E., KANEDA K., OKAMOTO T., NISHITA T.: A lighting model aiming at drive simulators. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), ACM Press, pp. 395–404.

[Per85] PERLIN K.: An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3 (1985), 287–296.

[Psz04] PSZCZOLKOWSKA D.: Visual model of fire. In *Eurographics 2004 Short Presentations* (Warsaw University of Technology Poland, 2004), 2004 E. A., (Ed.).

[Rac96] RACZKOWSKI J.: Visual simulation and animation of a laminar candle flame. In *International Conference On Image Processing and Computer Graphics* (1996).

[Ree83] REEVES W. T.: Particle systems : a technique for modeling a class of fuzzy objects. *ACM Trans. Graph. 2*, 2 (1983), 91–108.

[SF95] STAM J., FIUME E.: Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM Press, pp. 129–136.

[Sta99] STAM J.: Stable fluids. In *Siggraph 1999, Computer Graphics Proceedings* (Los Angeles, 1999), Rockwood A., (Ed.), Addison Wesley Longman, pp. 121–128.

[Sta03] STAM J.: Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference* (2003).

[WLMK02] WEI X., LI W., MUELLER K., KAUFMAN A.: Simulating fire with texture splats, 2002.