Representing Smoke in Computer Graphics

By DAVID STOPFORD

N.C.C.A., Bournemouth University

Abstract

In this paper an overview is presented of the methods used in computer graphics to represent the movement and surface properties of smoke. The main current methods of rendering smoke will be explored, those being particles, hypertextures, physical simulation, and sourcing live elements. There will also be an investigation into ways of combining different approaches to achieve different effects. The emphasis will be on evaluating the various traits of each technique and determining suitable situations in which it may be implemented.

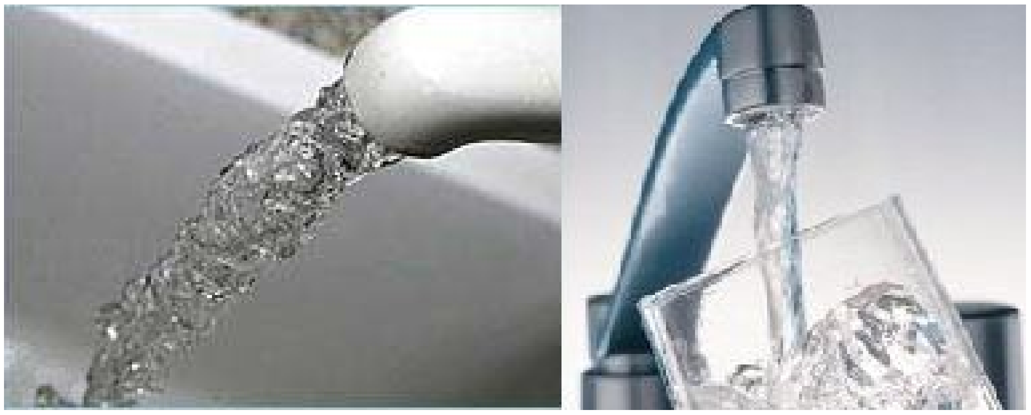
Introduction

It is common for modern computer graphics to require smoke elements within a scene. Due to the frequency with which visible gasses are encountered in reality, their absence would be notable in many CG situations. Furthermore, the field of special effects makes extensive use of smoke effects due to the relative safety of rendering explosions and pyrotechnics artificially on a computer.

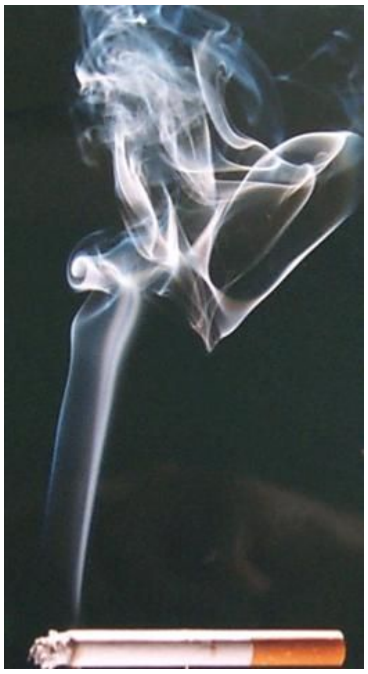
To fully understand how best to simulate smoke, it is necessary to understand the properties and behaviour of real life smoke. By knowing which base elements are necessary to mimic, generalisations can be chosen that result in a suitably convincing model.

The visible element of smoke is not a fluid, but rather a suspension of many tiny solid particles called an aerosol or particulate. [18]. In steam and mist, these particles are tiny droplets of water. The particles are suspended in a gas, from which they inherit their fluid (liquid or gas) like movement. However, one should note that the movement is not always consistent with that of a pure fluid [16]. This is because the physical state of the volume is not uniform – there are areas of gas, as well as solid. This influences the way in which smoke behaves, however it is not sufficiently significant to create visually discernible discrepancies. As such, it is acceptable to ignore this in a purely aesthetic use of computer graphics.

A fluid will always move in one of two ways, laminar or turbulent **(see figure 1).** In laminar flow, streams of fluid move parallel to each other, creating continuous areas of movement. An example of this would be water flowing over the back of a spoon. In turbulent flow, the movement is irregular and contains apparently random irregularities and vortices. This is the most common type of flow, and gives smoke its characteristic look. It is also the hardest to replicate on a computer, due to the complexity of the movement. An example of turbulent flow would be the smoke coming from a car's exhaust.



[Figure 1] Here the difference between different types of flow can be seen. On the left is turbulent flow [19] and on the right is laminar flow [23]. Liquid has been chosen as an example as it is easier to visualise the differences. Since both liquids and gases are fluids however, the concept is the same for smoke **[see figure 2].**



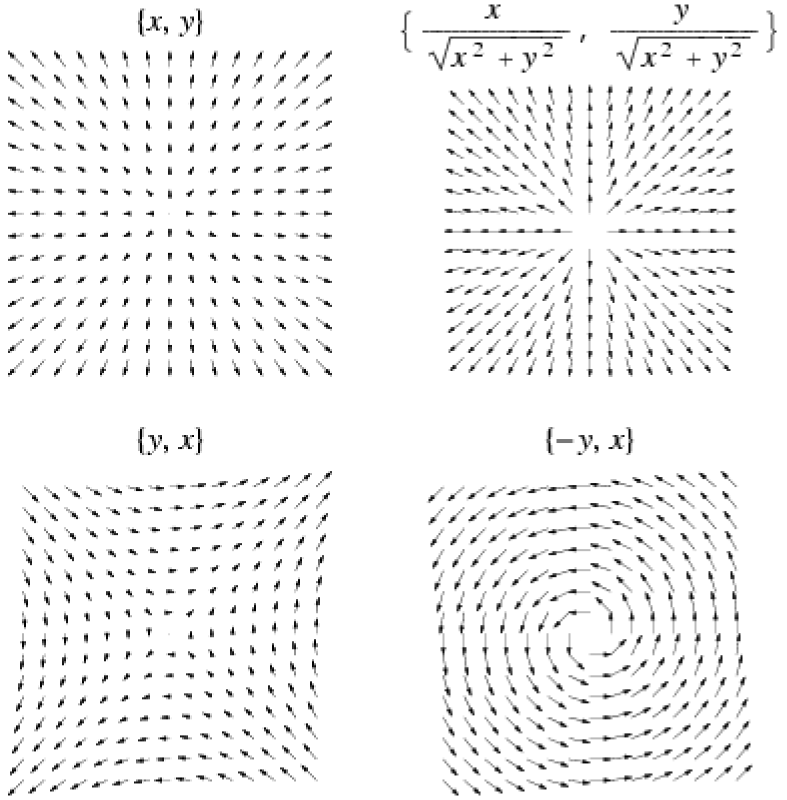
[Figure 2] [21] In this example there is a clear transition between laminar and turbulent flow. In most cases the variety of smoke seen in the top half of the image is the kind we wish to reproduce digitally. It is also the more complex of the two varieties of fluid flow.

Techniques used in Creating Computer Generated Smoke

Force fields

"In general physics, a force field is a vector field representing the gradient of a potential. The vectors that are the values of a force field are forces, and so measured in units of force such as newtons and pounds-force." [17]

A vector field causes movement due to the existence of a potential. Potential exists where a property has different values across an area. The greater the difference, the steeper the gradient of a potential and the faster objects will move. In real life, it is necessary for a force field (see figure 3) to be either magnetic, electric, or gravitational. In computer graphics, there is usually no need to make this distinction, since we are not dealing with objects that have real-life properties. It is sufficient for the physical properties of a simulated smoke particle to be limited to mass.


$\{x, y\}$
$\left\{ \frac{x}{\sqrt{x^2 + y^2}}, \frac{y}{\sqrt{x^2 + y^2}} \right\}$
$\{y, x\}$
$\{-y, x\}$

[Figure 3] This shows how a field covers an area, having both magnitude and direction [9]. An object in the field inherits velocity as a product of the vector field and the object's mass.

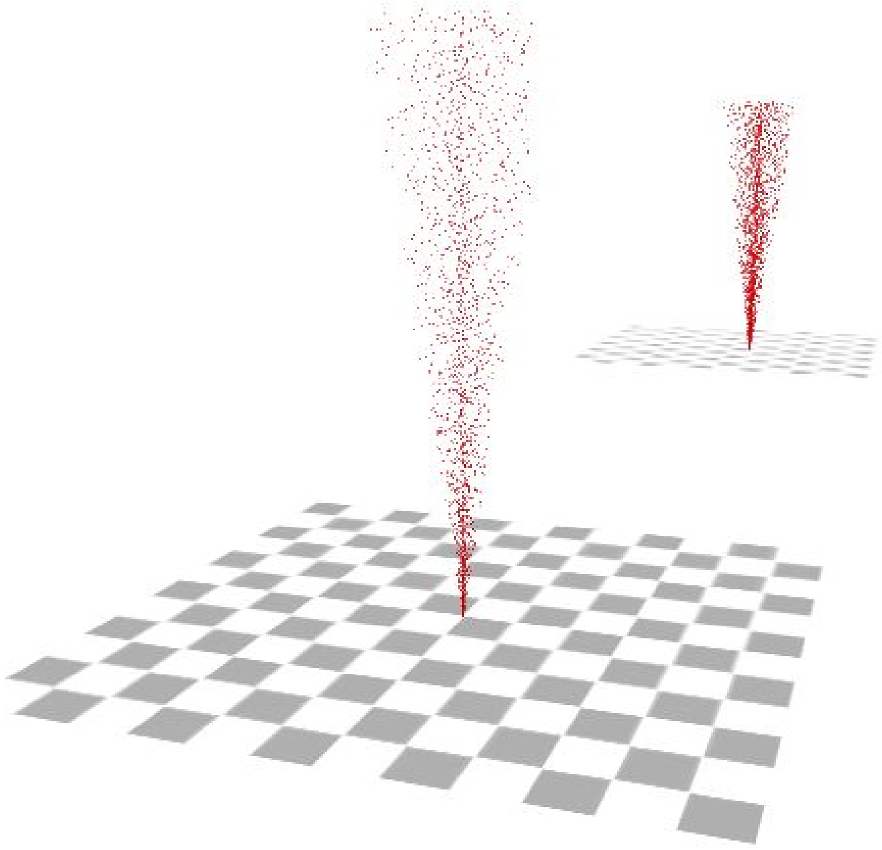
Force fields can be used in computer graphics to control the movement of a large number of dynamic objects. Often more than one is used simultaneously, and they can have their properties animated to achieve the desired effect. They provide high level control, which makes controlling large numbers of objects possible. Force fields are used in nearly all methods of generating smoke where the movement needs to be simulated. An example of their use would be to have one force field generating turbulence, with another adding velocity in a single direction. This ability to combine force fields provides a high degree of flexibility.

Particles

The simplest technique for representing smoke is by using particles whose movements are affected by force fields. The particles form a cloud of primitive objects that define a volume [12]. They can crudely estimate the movement of smoke, although it needs to be manually refined by eye until the desired look is achieved. Unlike a real cloud of smoke, there are no interactions between particles. The body of the smoke is represented by the particles. Since a particle represents only a point in space, it can be considered to have an infinitely small volume.

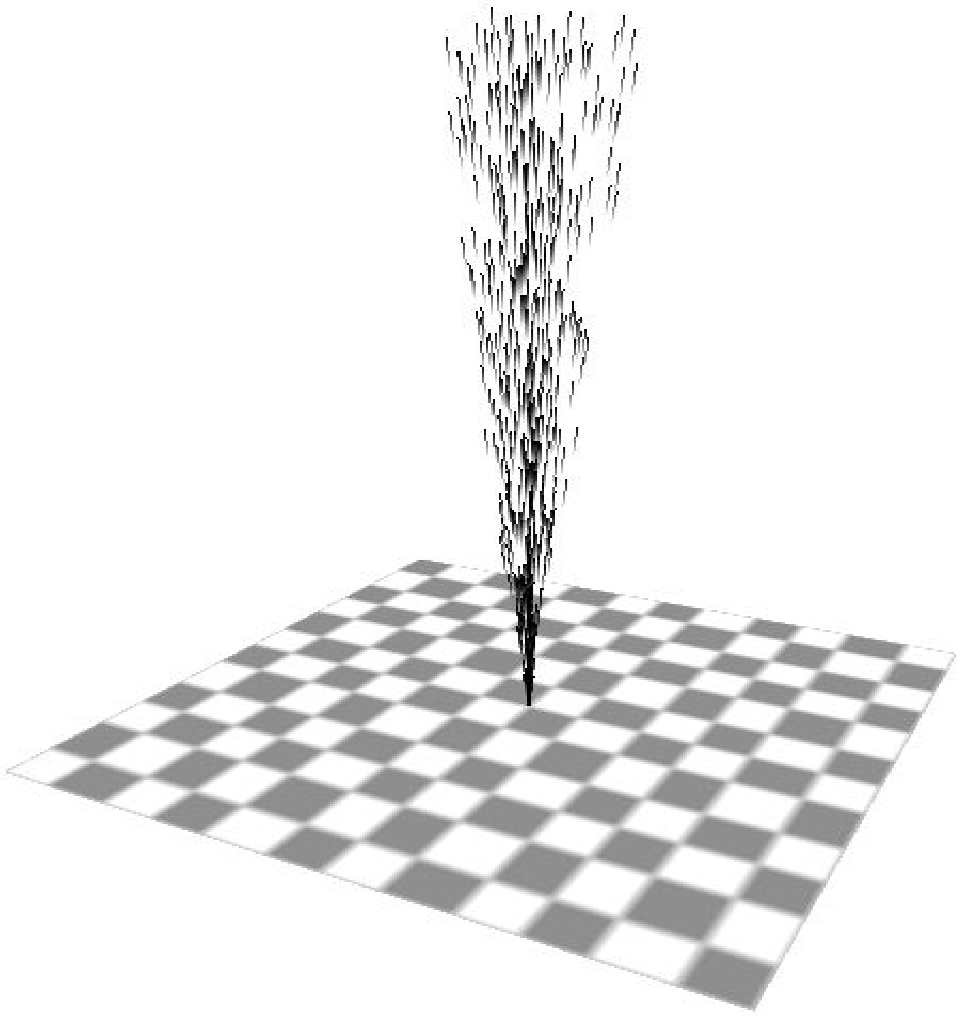
Particle systems consist of a large number of points, all contained within a single system. Most commonly particles are spawned from an emitter, and die after a set lifespan. This provides a continuous stream of moving particles. It is also possible to have a set number of particles in the system, which are given a starting position, and which can live forever. Forces are then applied to the system as a whole, simplifying the process of controlling such a large amount of data. Each particle is given a position, velocity, and acceleration. This information is sufficient to calculate the particle's new position after the effects of different forces have been applied. Other user-defined attributes can also be added, to allow for more complex effects to be built up. For example, each particle can be assigned a mass, despite this being a meaningless concept for an object without volume. The mass could then be used as an inverse scaling factor to the particle's accelerations.

Since the particles have no volume, rendering them presents a problem. In order to appear on screen they must appear to have a profile from the view of the camera. The simplest approach to overcome this is to see if a pixel contains a particle within its area. If this is the case, then the whole pixel is rendered in the colour of the particle. This is an inaccurate method of rendering particles because it does not take into account the effects of perspective – that is, they should appear to become smaller as they move farther away from the camera. Visually, the effect of this is that the particles appear to grow larger as they become more distant (**see figure 4**). This is an optical illusion caused by our brain's expectation for them to shrink according to distance and perspective.



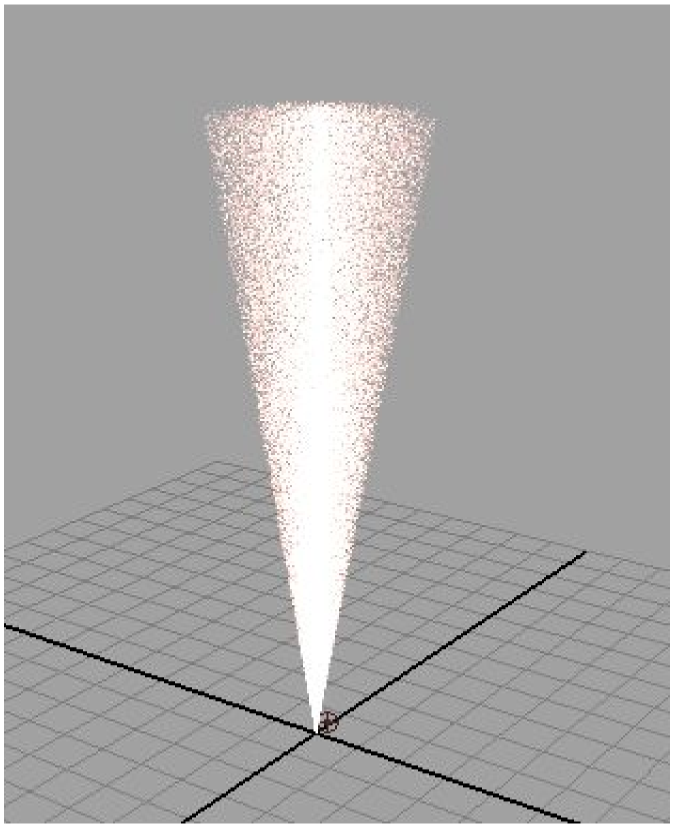
[Figure 4] Above, the two particle systems are identical, with the exception that one is further from the camera than the other. The more distant particles appear to be larger, because one pixel of screen space covers a larger area of world space at that point. In fact, all particles above are only one pixel in size.

With particles it is easy to remove the effects of temporal aliasing, without reverting to the use of computationally expensive motion blur techniques. This is because the motion vector of the particle is a known value, and applies only to a single point. Complex rotation and scaling that is possible on a two or three dimensional surface are not possible with a one dimensional particle. By drawing a faded trail behind a particle directly into the scene, the effects of motion blur can be efficiently reproduced (see figure 5). The size of the trail is proportional to the speed of the particle, to represent the distance travelled since the previous frame. This system of motion blurring particles is not entirely accurate. A particle does not necessarily follow a straight line, and may follow a curved path during the time between two frames. In this case, the curved motion blur would not be rendered, and only a straight line representing the particles final velocity would be seen. This method is often a usable alternative to performing the motion blur in post (after the completion of a render).



[Figure 5] Here the effects of motion blur are simulated by drawing a fading trail behind each particle. The length of the trail is the distance travelled by the particle between frames.

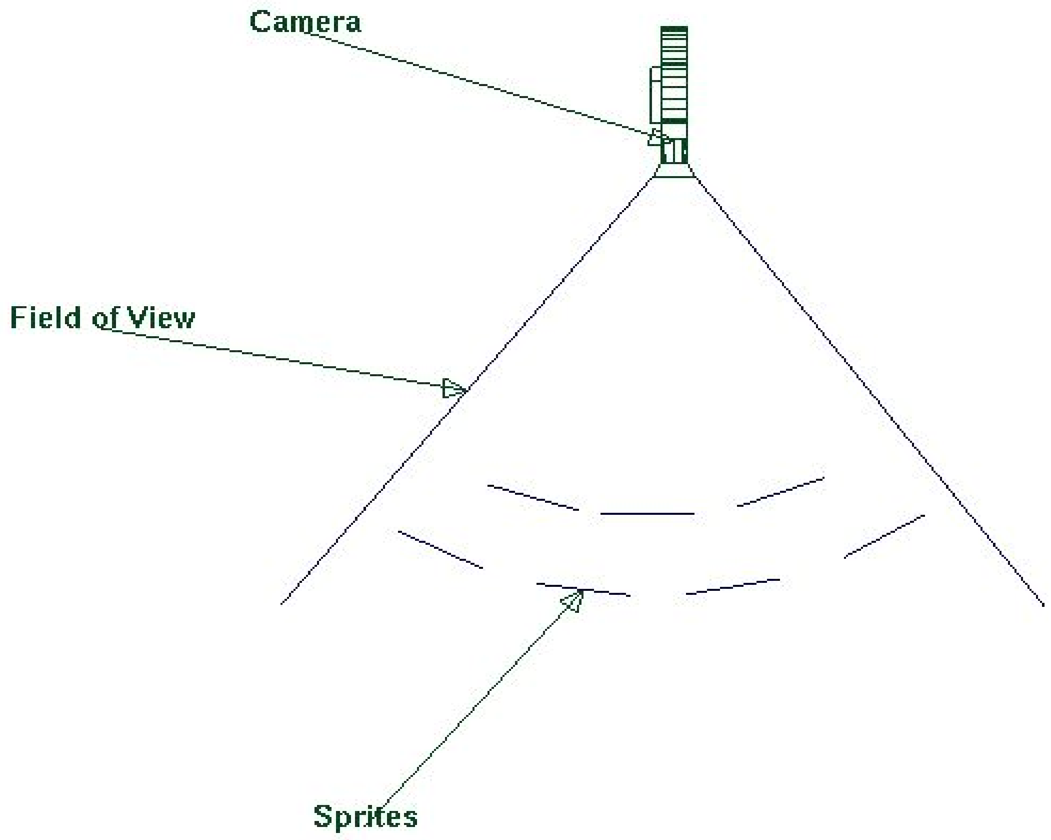
A slightly more advanced method of rendering particles is to use an additive colour method. With this technique, each particle adds its colour value to that of the pixel within which it is contained. In some situations this can provide a slightly more believable image, however there are also significant downsides. It is very easy for a pixel to become completely white **(see figure 6)**. Due to the limited dynamic range of the rendered image, any detail beyond this point is lost. It is also unconvincing for particles of a non-white colour to appear white when layered. Using this additive method also inherits the problem where the rendered particles do not scale correctly with perspective. As the particles become more distant, there is the effect of decreasing overall brightness of the rendered particles due to their being further apart on screen – the same amount of colour is spread over a wider area.


[Figure 6] In the above example, the high number of particles soon builds up to an unrealistic blown out look. Moving the camera further away would darken the particle colour.

In real life smoke appears to be a fluid. In reality it is a solid, composed of millions of tiny specks of dust. The mass of each speck is so small that its movement is almost entirely governed by that of the air in which it is suspended. This explains why its movement resembles that of a fluid. Given this information, it might be expected that particle systems are a good method for representing smoke. However, there are problems that make them less than ideal. It would be beyond the ability of today's PCs to easily store and calculate the movements of such a large number of particles. While it is possible, it precludes it from being a sensible technique in most cases. It is also worth noting that with this technique, it is still necessary to simulate or estimate (using force fields) the fluid movement of the air in which the particles are suspended, which is a very time consuming process when many particles are simulated.

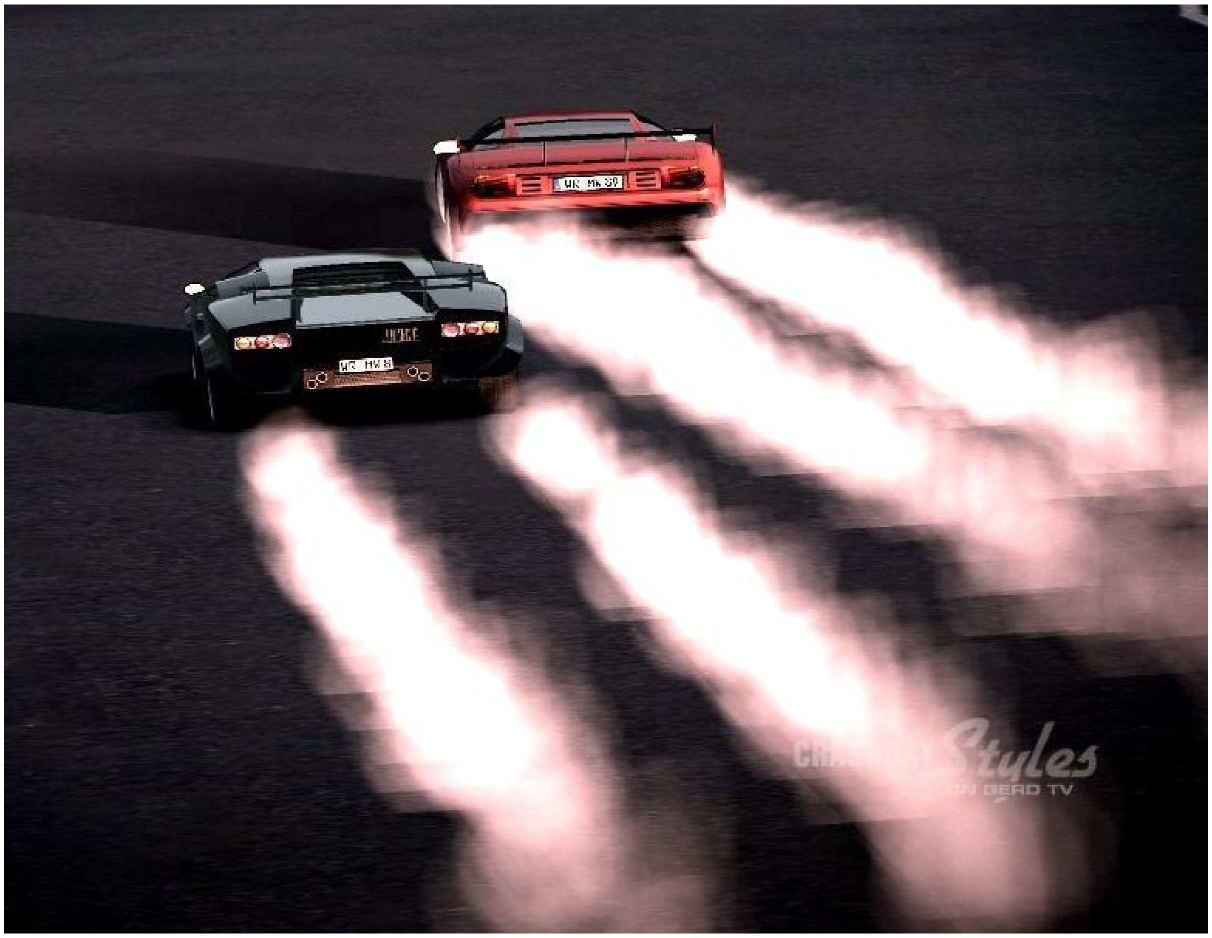
Sprites

To overcome these drawbacks, it is very common to use sprites as a representation of the volume of smoke. With this technique, a quad (rectangular polygon face) is parented to each particle in a particle system. The quad is constrained such that its normal always faces the camera **(see figure 7)**. Typically each quad is then texture and alpha mapped to represent a collection of smoke particles. The system of sprites is then animated as usual through the use of force fields. The effect is an approximation of how the smoke would behave if each of its speck's movement were simulated. The limitation is that the individual specks of smoke represented by one sprite cannot move relative to each other. An advantage is that limited self-shadowing within the area represented by the sprite can be baked into the texture, since the smoke particles' positions relative to each other do not change.



[Figure 7] A top-down diagram showing how sprites are aligned. Each sprite's normal points towards the camera, resulting in a curved layout. In this example the curvature is fairly noticeable due to the sprite's close proximity to the camera. In most situations the sprites will be some distance from the camera, resulting in a near-parallel layout.

Apart from this limitation to realistic movement, there are visual artefacts due to the way it is rendered. This is caused by the discrepancy between real smoke which occupies a volume, and sprites which are placed on discrete planes. The intersection between a plane and opaque environment geometry can become visible, breaking the illusion of a volumetric body **(see figure 8)**.

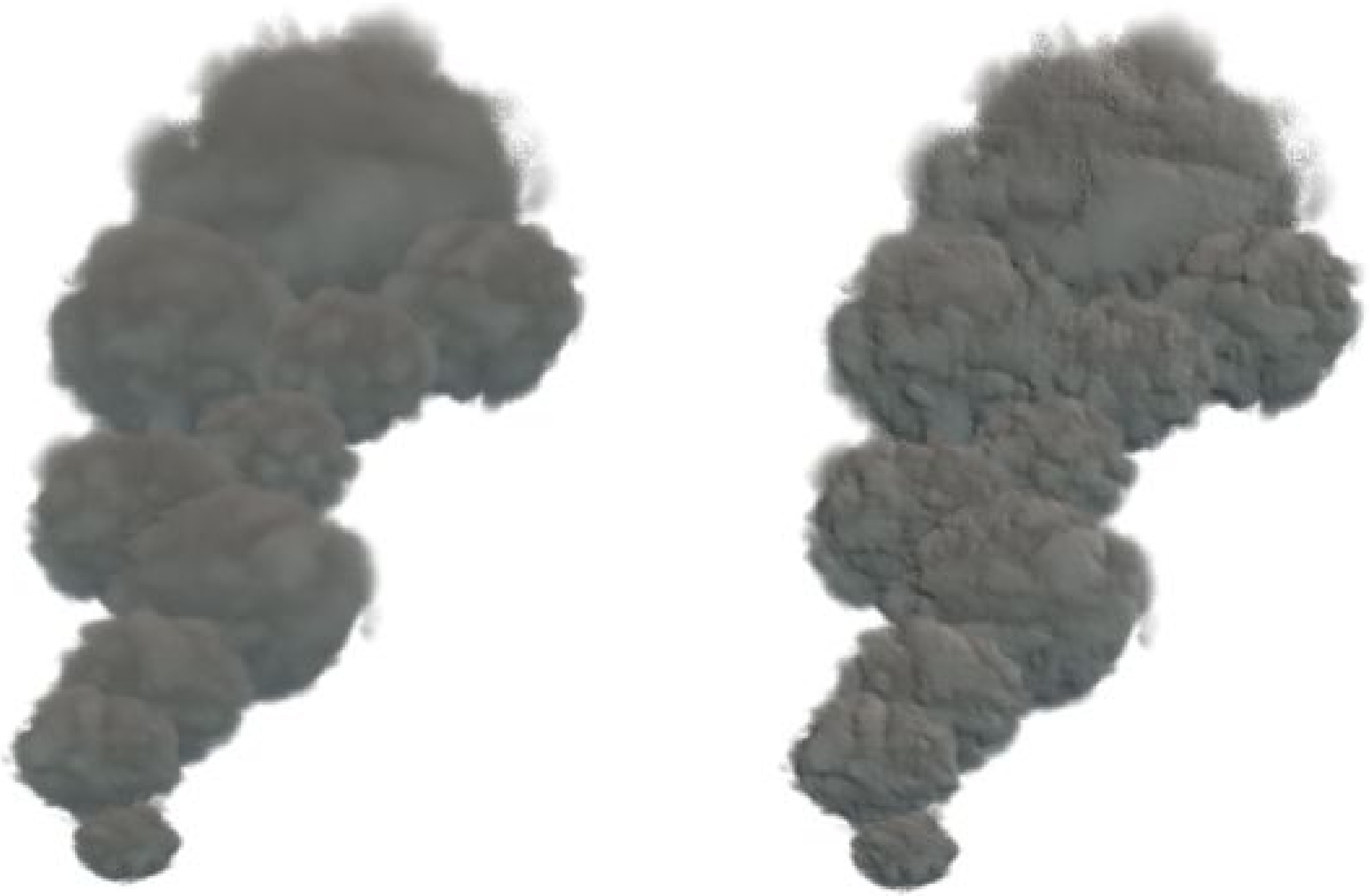


[Figure 8] The above example is taken from the game Crashday [6]. The point at which each sprite intersects with the ground creates an unwanted hard edge, which betrays the non-volumetric nature of the smoke.

Sprites are still widely used to represent smoke since it is possible to use them in a way such that their limitations are not visible. The best case is when none of the sprites need to intersect another object, for example the steam coming from a train, or the smoke from a bonfire. Sprites are very fast to render, since they use small amounts of geometry and require only simple lighting.

Despite being fast, the use of only simple lighting on sprites is not ideal, due to the fact that it poorly approximates the interaction of light with real smoke. The surface of the smoke is treated as a simple plane, with no variation across the surface. This contrasts with real smoke, which features very complex surface detail. To simulate this, one technique is the use of bump maps **(see figure 9)**. This is a technique that perturbs the normals of a surface based on a greyscale heightmap.

The geometry remains unchanged, and only the lighting is adjusted. This technique is more convincing than merely adding more detail into the colour map, as it is light angle dependant. This means that as a light moves relative to the object, the small bumps on the surface appear to be lit correctly. This technique can be applied to smoke, in order to increase the realism of the surface's interaction with light.



[Figure 9] Here the effects of adding a bump map are demonstrated. On the left hand side is a standard implementation of sprites, with each sprite holding only a colour channel and a transparency channel. The result is not realistic, due to the fact that the sprite's illumination betrays their two dimensional nature. On the right hand side, a bump map has been added to each sprite, perturbing their normals. The effect is more realistic, adding shadow to indentations as defined by the height map. If a light source were to move across the front of the smoke cloud, the shadows would move in accordance. The bump maps also serve to camouflage the boundaries of the sprites, due to the increased level of detail.

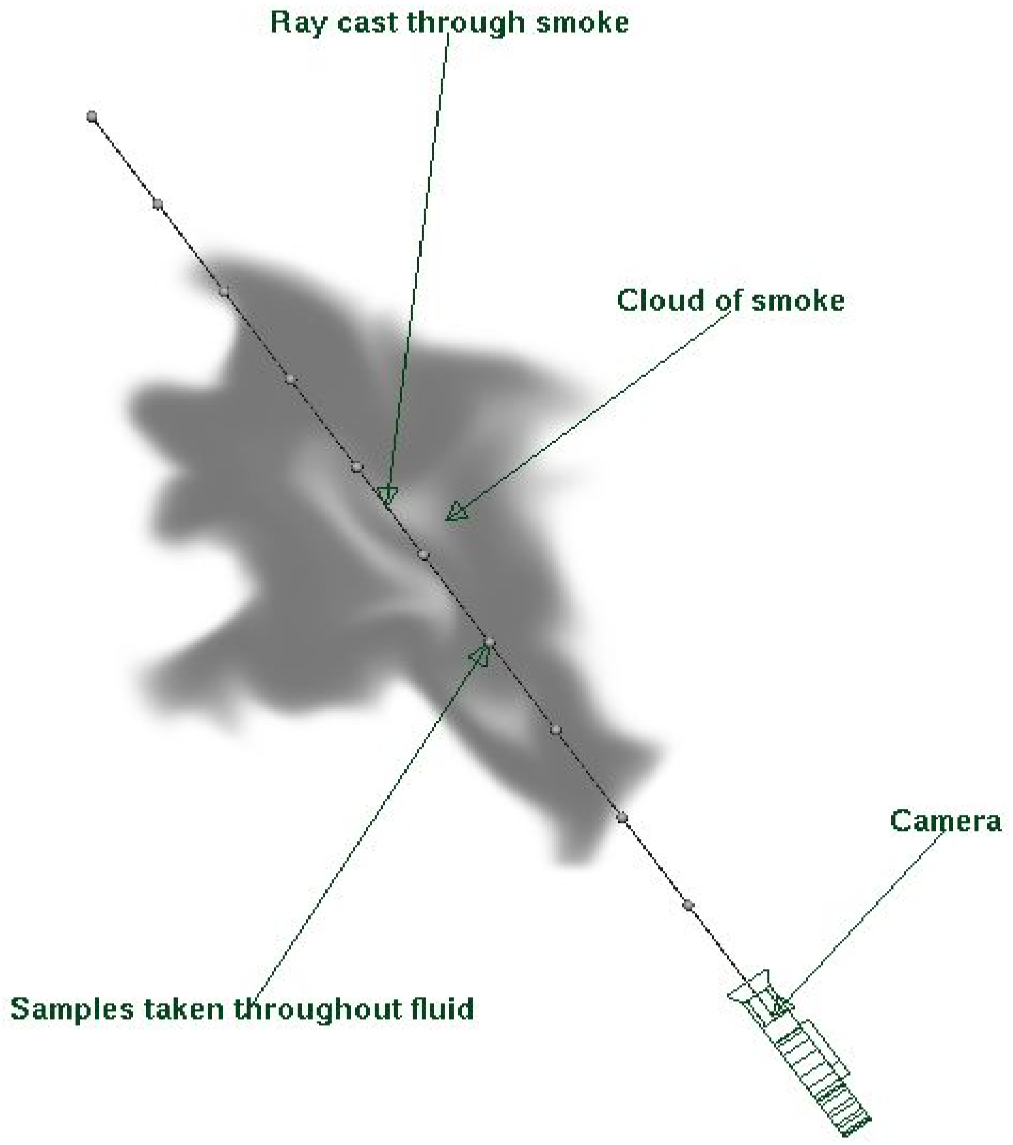
In real life, the properties of a smoke cloud change with age. Due to the constant random motion of molecules (Brownian motion), there will be a tendency for smoke to move from areas of high density to areas of low density, resulting in diffusion [15]. This can be represented with the sprites by increasing their size and transparency as they age. To do this, the sprites inherit the age attribute of the particles, and use it to control their size and transparency attributes. With volumetric smoke (see below), as the density decreases more light can penetrate the cloud. This means that as the cloud becomes more transparent at the top, the colour becomes lighter. At the bottom where the smoke is still dense, there is more self shadowing and the smoke appears to be darker. This effect can be simulated with sprites by lightening the colour as the particles age. Another factor to take into account is to avoid obvious repetitions in the texture, which will look very unconvincing to the viewer. An easy method with which to have unique texturing on each sprite is to use two dimensional procedurally generated textures.

To increase the realism of the sprite's movement, it is necessary to recreate the way in which real smoke tends to roll in on itself as it rises, caused by convection currents of hot air. It is not possible to do this realistically with sprites because one of their axes is constrained so that it points towards the camera. This is the only axis around which the sprite can rotate. A partially convincing rotation can still be implemented around this axis, which breaks up the shape of the smoke somewhat. This technique is only convincing when used sparingly, as it makes the boundaries of the sprites more apparent.

Volumetric smoke

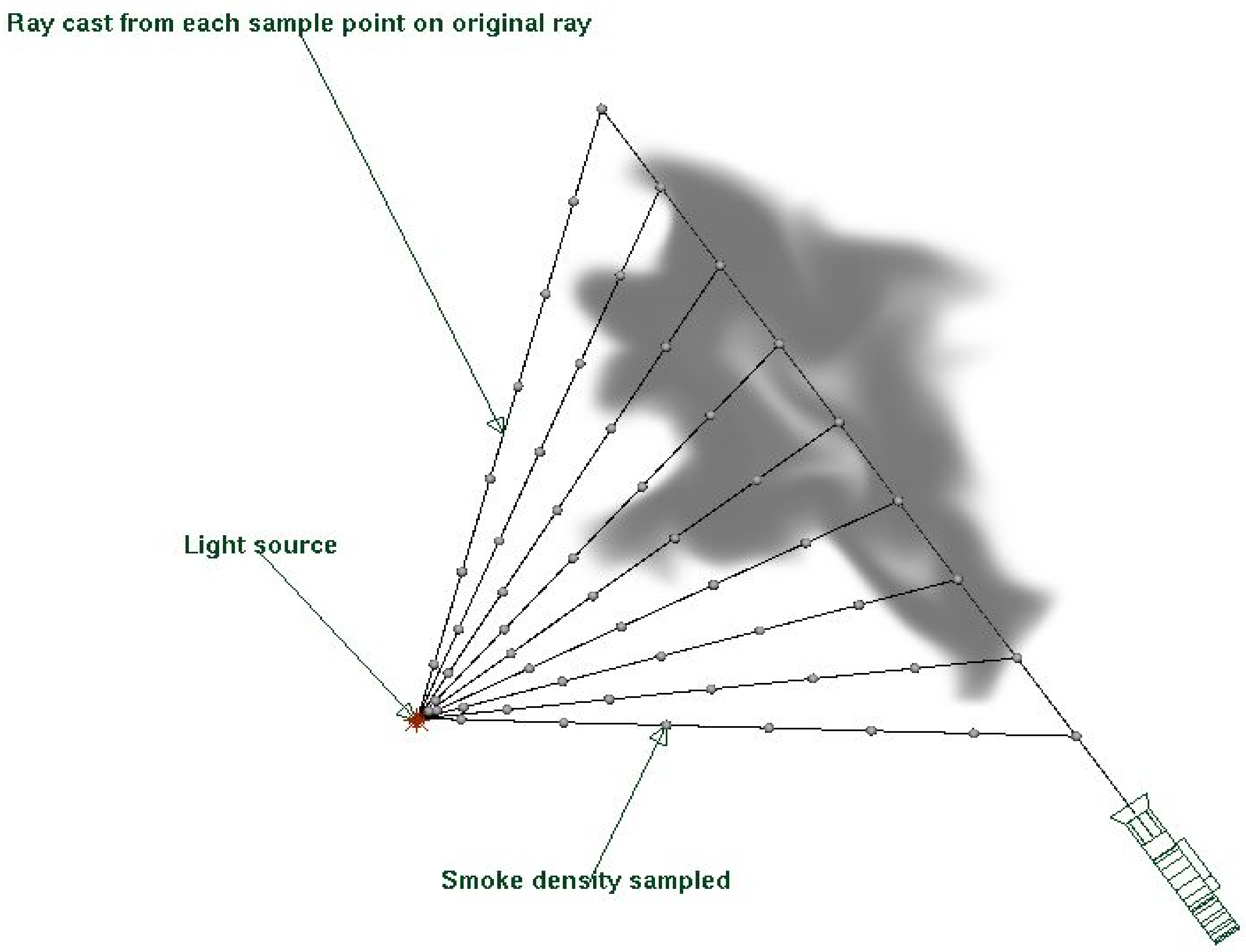
To render smoke with a realistic depth to it, a volumetric approach can be taken. Instead of representing only a two dimensional projection of the smoke, as in with sprites, the smoke occupies a volume and has properties that vary continuously throughout. This allows the cloud to have dense and sparse areas, for example. By having the cloud of smoke described in three dimensions, it can be viewed from any angle and still hold up visually. This also means that it can interact with the environment in a realistic manner, since it is formed in the same way as a real cloud of smoke.

Volumetric phenomena such as smoke are rendered through a process known as raymarching [10, 8]. This is the volumetric equivalent of raytracing. With raytracing, a ray is cast from the camera for each pixel of a scene. The point at which it intersects with a surface is detected, allowing the depiction of scene geometry. This technique is only suitable for solid surfaces however, since only one sample is taken at the point of intersection. With raymarching, the technique is extended to work through a volume of participating media. As the ray is traced through the volume, samples are taken at regular intervals and the density at that point examined **(see figure 10)**. The final colour value for the pixel will be the sum of the densities sampled through the volume. To increase the detail of the raymarch, samples can be taken at smaller intervals.



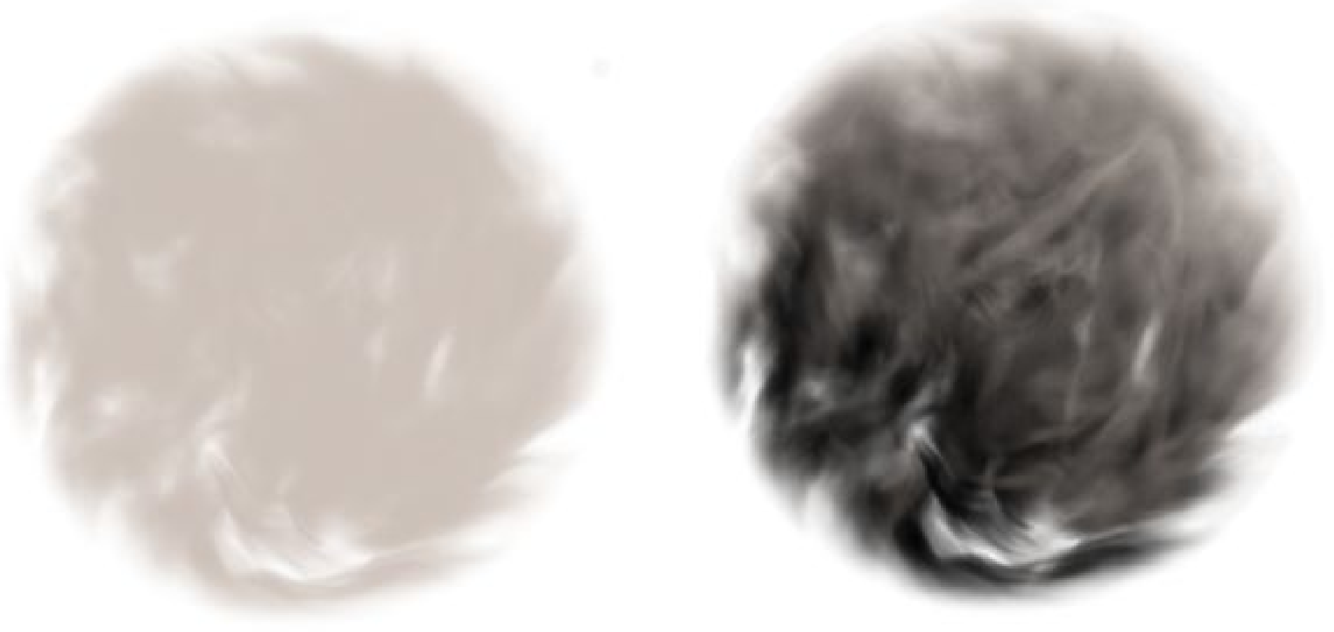
[Figure 10] This diagram shows the technique used to perform raymarching. The ray is cast from the camera and travels through the fluid, with samples being taken at regular intervals. In this case, the samples are very far apart for the sake of clarity, however in a real implementation the samples would occur with a much higher frequency.

This technique alone will produce a render that accurately represents the density of the smoke. However, the interaction between the smoke and light are not taken into account, creating an unconvincing flat look **(see figure 12)**. Shadowing can be implemented by raymarching from each sample point to the light source **(see figure 11)**. This allows the calculation of the total density of smoke that is occluding that point inside the cloud.



[Figure 11] In this example the technique by which self-shadowing is implemented is shown. Rays are marched from each sample point to the light source. As can be seen, the number of samples required is greatly increased. It is also proportional to the number of light sources – adding another light would double the number of samples necessary.

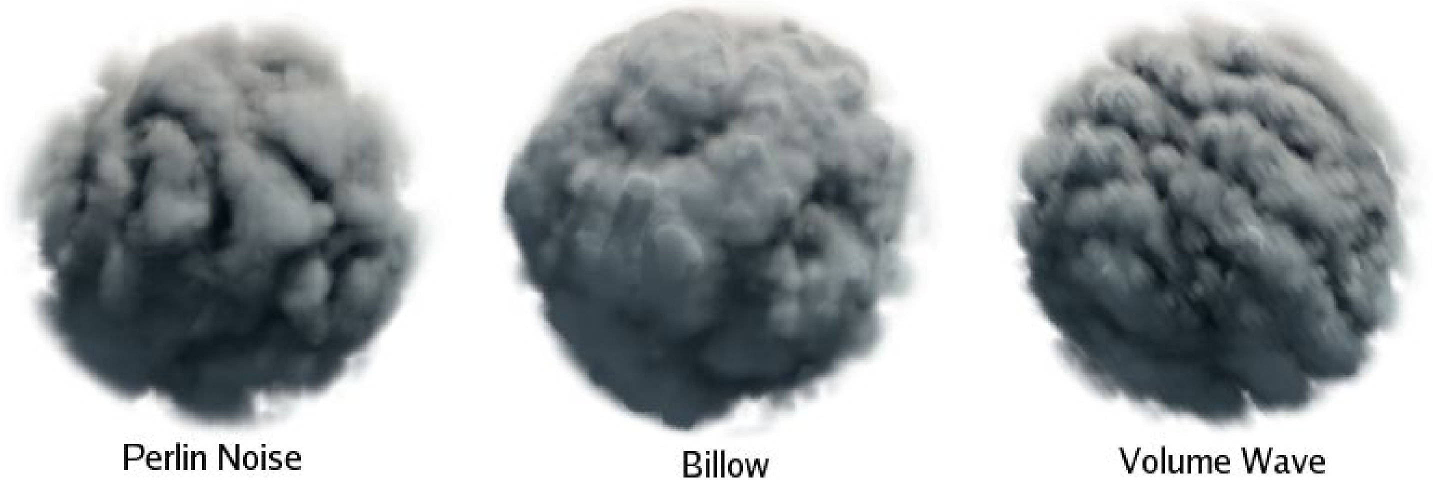
In these diagrams, a somewhat simplified depiction of the process is presented. In order to increase efficiency, methods to narrow down the area in which samples are taken are used. By only taking samples within the bounding box of the smoke, it is possible to check only the volume in which it is possible for density to be present. Another technique that may be implemented on top of this is to stop the raymarch after a certain density has accumulated [10]. This may be used for example when the volume has become completely opaque and no more light from behind the smoke can reach the camera.



[Figure 12] In this diagram, the smoke on the left contains no self-shadowing. The raymarcher has accurately calculated the density of the fluid, which has affected the transparency of the smoke. However, because the smoke is of uniform colour the result is still unconvincing, and appears to have no depth. On the right hand side, shadowing has been calculated for one light source. The density of the cloud at each pixel is still represented by the transparency of the image, however the light level now also affects the colour of the pixels. This result is now much more convincing, and usable as a technique for representing smoke.

Hypertextures

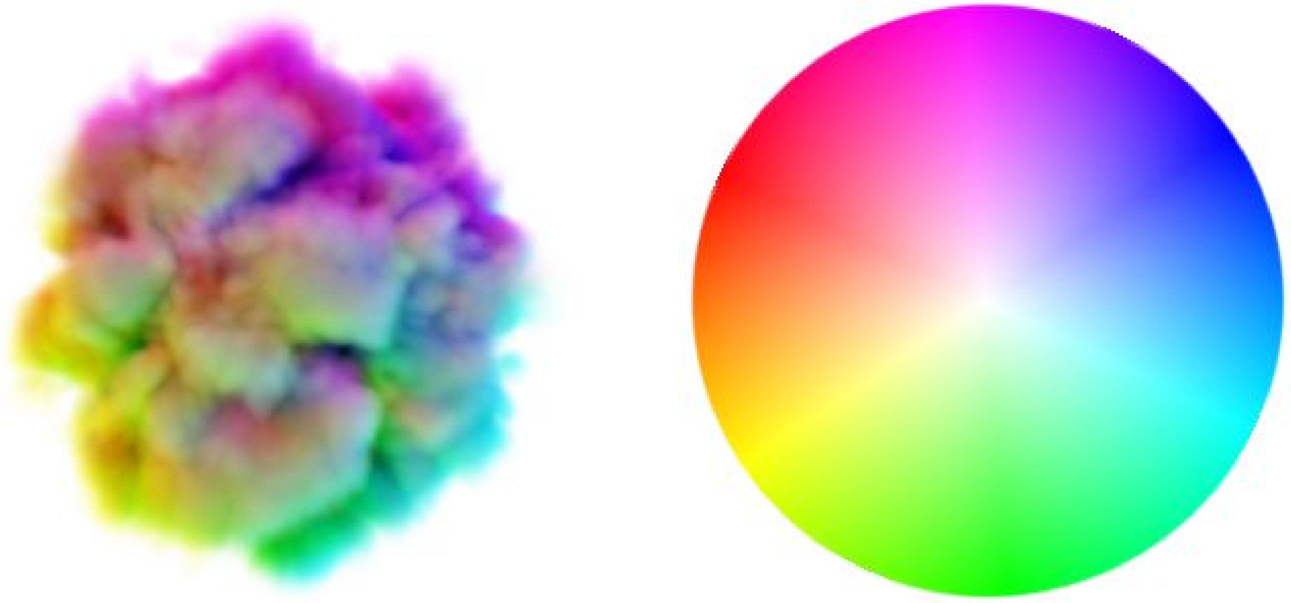
In order to be able to raymarch through a participating medium, it is first necessary to define the density information for the cloud. A common and flexible method of doing this is to use hypertextures, which *"model phenomena intermediate between shape and texture by using space-filling applicative functions to modulate density"* [11]. This means they are an extension of procedurally generated three dimensional textures for solid surfaces, filling a volume instead. There are two benefits to the use of hypertextures, speed and detail. Procedural noise hypertextures can be calculated in a relatively short period of time, and detail can be added recursively where a fractal approach is taken. The downside of using hypertextures is that it is hard to specifically control the positioning of attribute variations. Furthermore, procedural noise provides only a rough estimation of the variations in smoke, and is not physically accurate.



[Figure 13] These are renders of the same volumetric smoke cloud using three different types of noise hypertextures to map the opacity. They use different algorithmic approaches to making noise. Perlin noise is the most commonly used. It is fast to calculate, but is somewhat inflexible. Billow is very slow to calculate, but is capable of creating the most convincing plume effect. Volume wave takes a non-fractal approach, instead summing intersecting waves in three dimensional space. These are just a few of the types of procedural texture available, shown here to demonstrate their differences.

Post Lighting

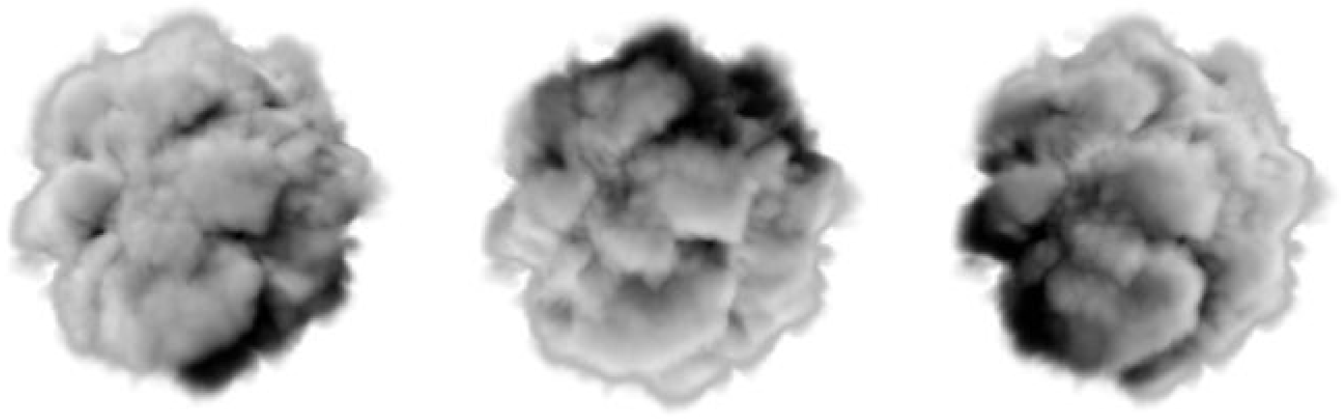
As shown above, adding lighting calculations to a raymarch significantly adds to the render time. Where many lights are required, it can become extremely slow to render correctly. It is possible to work around this by using a technique called post lighting. When this method is used, the render does not contain the lighting for the actual scene, but instead the normals of the smoke are calculated. The three axes of x, y, and z are represented by the red, green, and blue channels of the image **(see figure 14)**. By knowing the normals of the image, it is possible to adjust the lighting upon them arbitrarily after the render has finished.



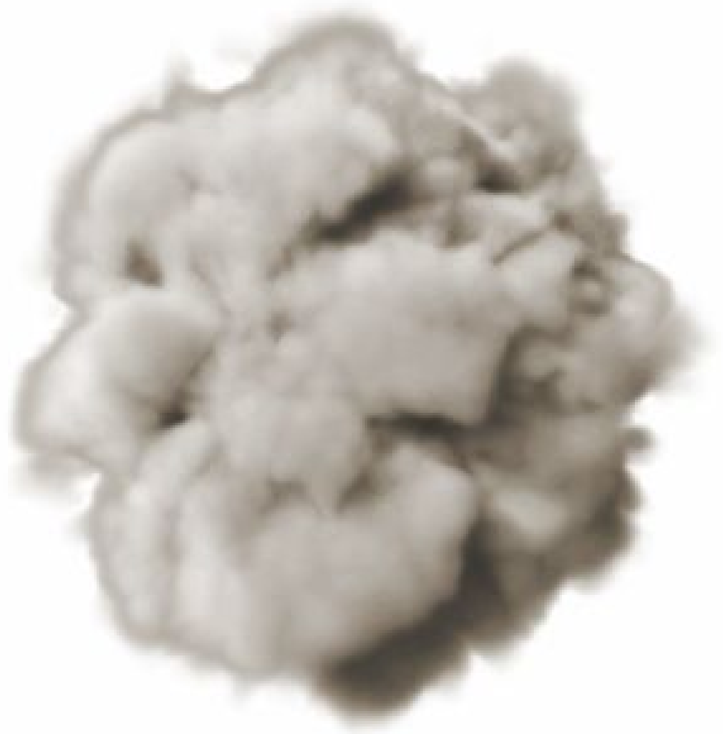
[Figure 14] The image on the left above has been rendered with the normals of the smoke instead of lighting information (colour is necessary for correct viewing). The colour of each pixel is the sum of its red, green, and blue values. These directly correlate to the normal's rotation along the x, y, and z axes at that point. On the right is a corresponding colour wheel, which we use to choose the direction of the lighting.

Since a volumetric object does not have a solid surface, the normals can not be calculated from a single point. Instead, a different technique involving the use of coloured lights must be used. Three directional lights are placed in the scene at right angles to each other, such that they coincide with the world space axes of the scene. The smoke is coloured white, so that it inherits the colour of the lighting correctly, without being multiplied by a diffuse value. During the lighting calculations, samples are taken throughout the volume of the fluid. At each point, the received illumination of the three lights varies due to the different amounts of smoke surrounding the point. The final colour of the pixel is the layered sum of these lighting samples. While it is impossible to correctly store volumetric normal information in a two dimensional representation, this technique yields results that are satisfactory for the purposes of two dimensional relighting.

To relight a normal mapped image, we take advantage of the fact that different colour channels represent light coming from different angles. In the image above, if we wished to light the smoke from the top right only, we would extract the blue channel only as luminance information **(see figure 15)**. This can then be recoloured arbitrarily to simulate different light colours. To add more lights, we simply repeat the process with other colour channels and add to the final result **(see figure 16)**.



[Figure 15] Shown above are the three colour channels used in **figure 14**. Note that when separate, each channel resembles the smoke lit from solely the x, y, or z axis. .



[Figure 16] In this image the channels in **figure 15** are combined, allowing arbitrary relighting. In this case, the red and green channels have been combined to simulate a light coming from the left hand side.

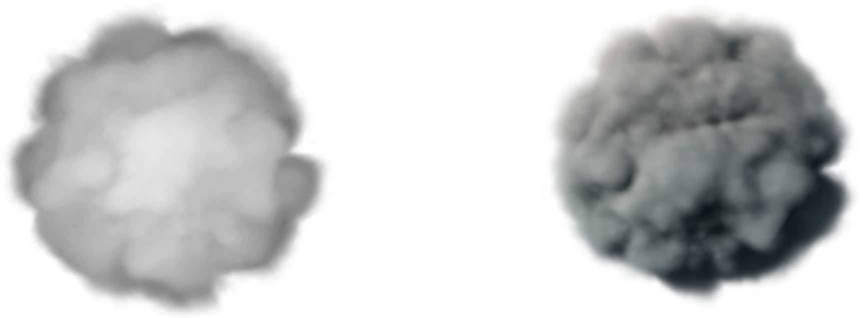
Since this technique requires three lights to generate a render describing normals, it is fairly slow to render. However, if the smoke is then relit using four or more virtual lights, then render time has been saved. Post lighting provides a further benefit in that the turnaround time for different lighting setups is very small. Lights can be moved in real time around the smoke because only simple calculations in two dimensions are necessary. Hence even if only three or less lights are needed in a scene, it can still be time efficient to perform post lighting as there is significantly less turnaround time.

Since we only have a two dimensional representation of the object's normals, with post lighting we can only place lights around the object in the hemisphere closest to the camera. For most cases this is acceptable, however it precludes the possibility of lighting the smoke from behind to create a rim light effect.

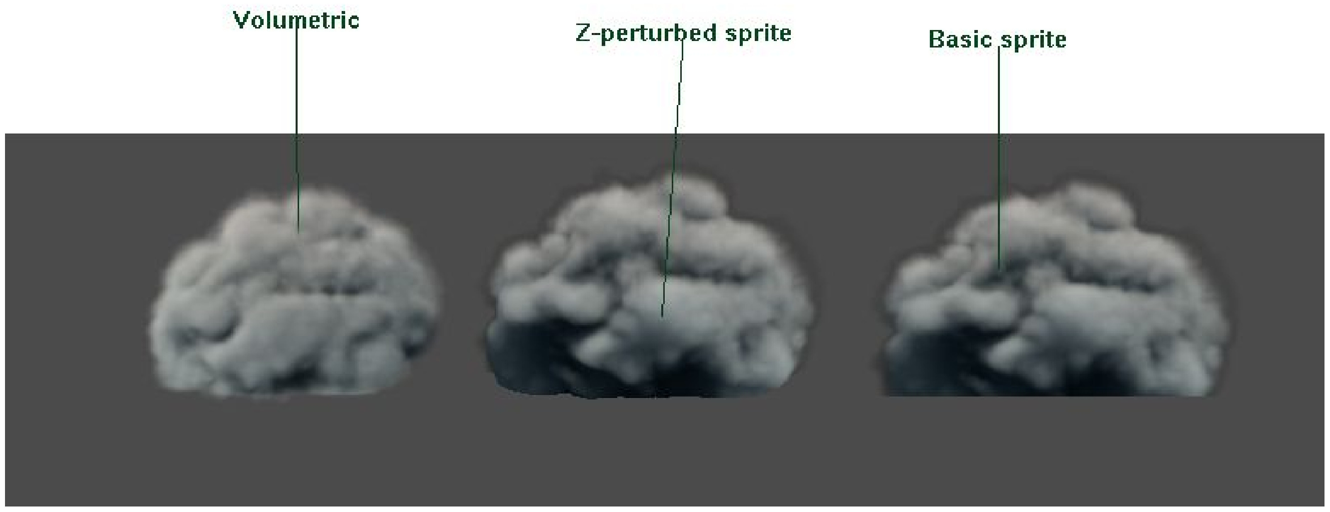
Depth Sprites

It is possible to artificially distort the intersection between a sprite and the environment geometry through the use of depth sprites [3],[6] **(see figure 18)**. With this technique, each sprite has a z-depth stored in a channel **(see figure 17)**. This represents the distance of a point on the sprite relative to the position of the camera. The z-depth map and the diffuse texture can be created simultaneously, by pre-rendering a volumetric source image.

When rendered, the sprite is still a flat plane, however its normal z-depth values in the scene are offset by those in the z-depth map. This has the effect of distorting the sprite's apparent distance from the camera across its area, and creating a visually more convincing intersection. Using a z-depth requires a fifth channel – Red, Green, Blue, Alpha, and Z Depth.



[Figure 17] These are renders of a volumetric cloud of smoke. On the left is the z-depth – the brighter the pixel, the closer its proximity to the camera. On the right is the diffuse render, showing the combination of lighting and colour values.

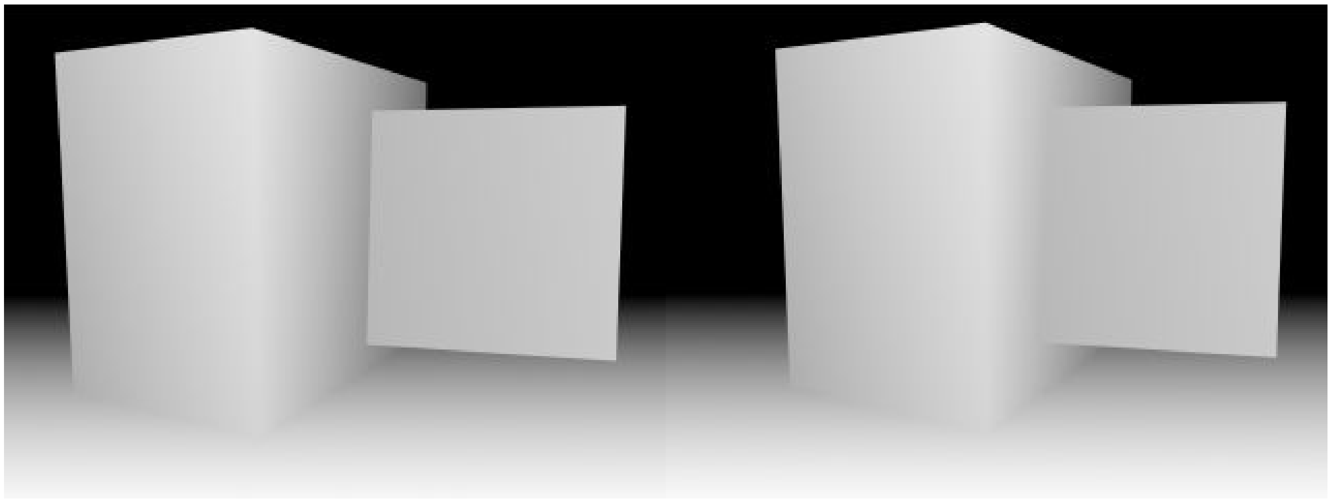


[Figure 18] This image displays the different methods of smoke generation to show the benefits of depth sprites. On the left is the original volumetric cloud. This has an accurate intersection with the environment. In the middle is a z-perturbed depth sprite. The intersection is not straight, as the sprite contains information about the relief of the smoke. This is a rough approximation of the interaction between a volumetric cloud and the environment. On the right is a normal sprite. Since this is just a basic plane, the intersection with the environment is an unconvincing straight line.

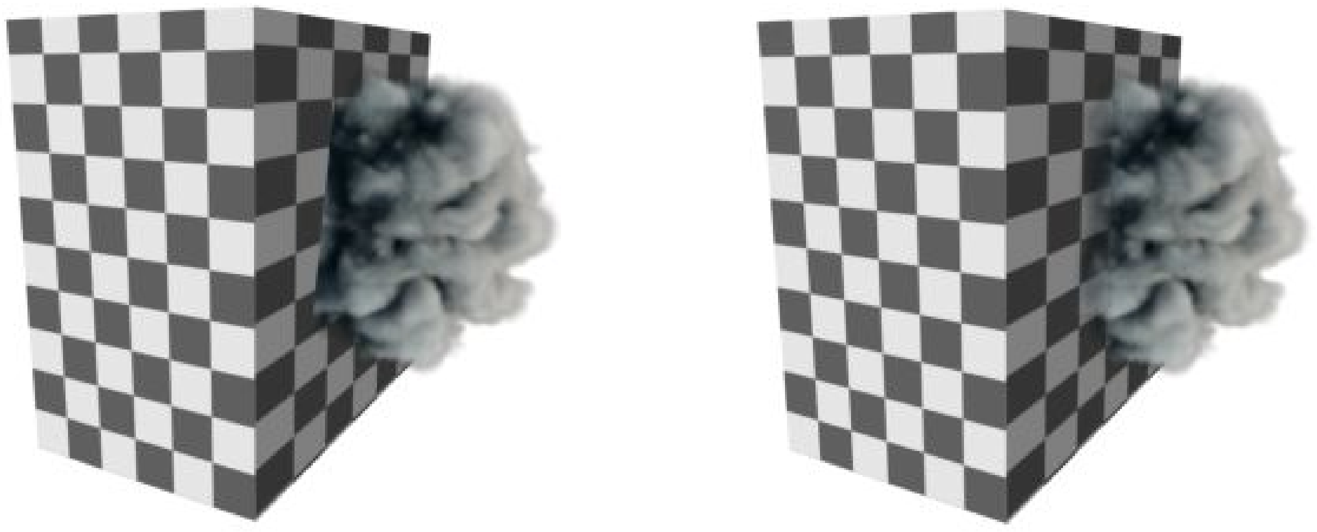
Despite their increased realism, depth sprites are not completely convincing. While they do perturb the intersection between the cloud and its environment, there is no gradient in the transparency which results in an unrealistic hard edge. This is because the z-depth pass can only store the closest point in the cloud at each pixel, discarding any volumetric information. With real smoke, as well as computer generated volumetric smoke, the opacity is built up by the thickness of the volume. This effect can not be achieved through depth sprites alone.

Z Feathering

While depth sprites are a step towards more realistic sprite-environment intersections, the hard edge produced is far from convincing. The soft edge provided by volumetric smoke can be simulated through the use of a technique known as Z feathering. This technique can be applied in realtime, since only the z-depth of the sprites and environment needs to be checked. The technique involves checking each pixel of the environment that is adjacent to a sprite's edge (**see figure 19**). The z-depths of the environment pixel and the adjacent sprite pixel are compared. If they are equal, then it is known that the sprite intersects the environment at that point. The sprite can then be made transparent at that point, and faded to opaque over a distance. This removes the appearance of hard edges where the sprite intersects the environment.



[Figure 19] Here is a z-depth render of a scene with a sprite in an environment. The sprite's normal is constrained such that it always faces the camera. On the left, the sprite is not intersecting the environment, and no z-feathering is required. Note that there is a sharp change in intensity values at the edge of the sprite. On the right, the sprite is intersecting with the environment, and the cube and sprite share the same intensity value at the point of intersection. This indicates that z-feathering is required, and the sprites opacity faded off before the intersection.



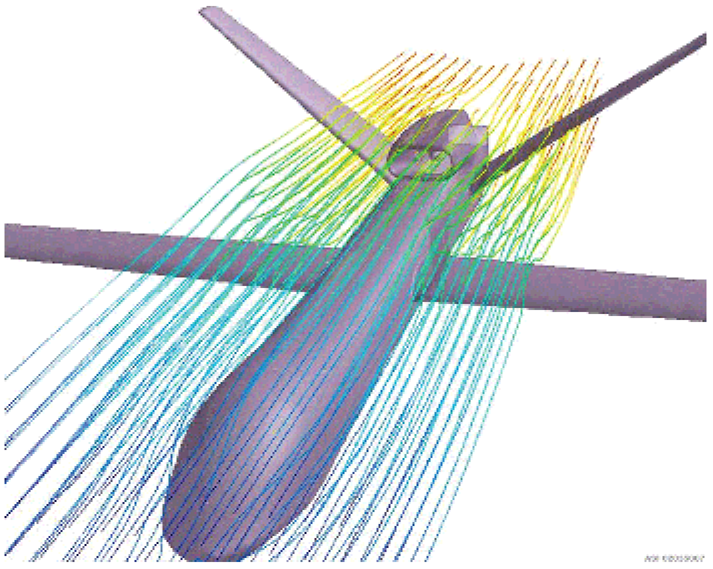
[Figure 20] Note here the intersections between the smoke sprite and the environment. On the left there is no z-feathering, and a hard line marks the intersection between the two. On the right, z-feathering has been implemented and a much less noticeable soft boundary marks the intersection.

It is also possible to implement z-feathering using a technique that makes no use of the z-depth in the scene. Instead, a more intuitive world-space approach is taken. For each pixel on a sprite, rays are cast out in a disc shape along the plane in space formed by the sprite. If a ray intersects with environmental geometry, then it is known that this point on the sprite is close to an intersection. The transparency of the sprite's pixel at that point is made proportional to the distance from the geometry. This method is slower however, as it requires the use of raytracing for the render.

In general z-feathering creates a superior visual result in comparison to depth sprites. This is because depth sprites only solve an error that is not highly visible, whereas the intersection artefacts that are obscured by z-feathering are highly visible.

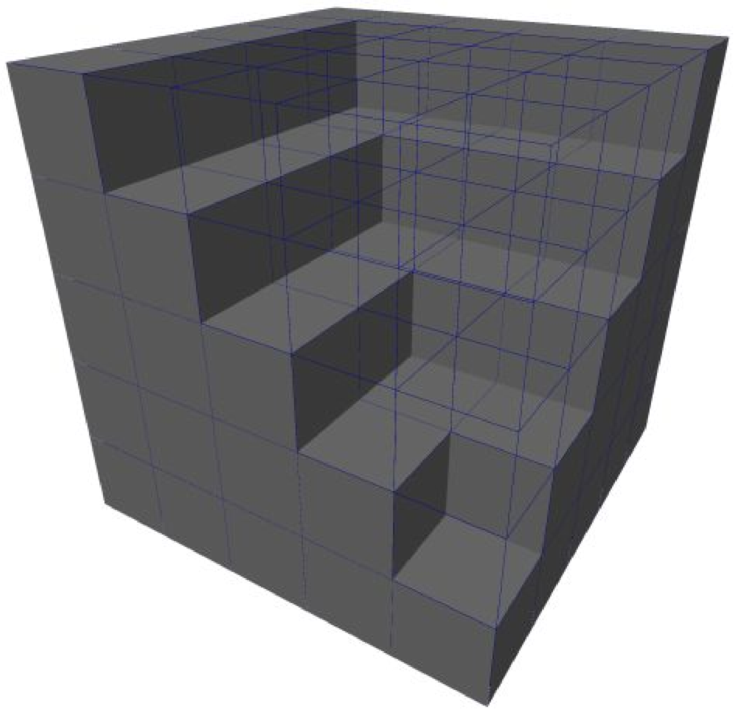
Fluid Dynamics

Fluid dynamics is not specific to computer graphics, but rather refers to the study of the movement of gases and liquids in general. The use of fluid simulation in computer graphics is unique in that it does not seek to make an accurate simulation of the fluid's behaviour, as in for example vehicular aerodynamics testing [2] **(see figure 21)**. Instead, it attempts to merely recreate a visually acceptable approximation that the viewer will be unable to distinguish from an accurate simulation. This is possible because the human brain can only loosely predict the many complex factors that will affect the movement of a fluid. This inaccuracy of perception allows for approximations to be used in the simulation, vastly reducing the time in which results are calculated.



[Figure 21] This is an example of fluid dynamics used where visual appearance is not important, but the accuracy of the simulation is. In this case the simulation is of the aerodynamics of airflow along the surface of the aircraft. [7]

To simulate visually acceptable smoke, the most common technique is the use of Navier-Stokes equations [13]. These equations are not specific to computer graphics, in fact they are over one hundred years old, however they can be used to make an accurate simulation of fluid. One implementation of the simulation takes place within a cuboid container, composed of an array of voxels. Each voxel is a three dimensional pixel, and takes the form of a cube that stores values **(see figure 22)**. Just as a pixel stores the colour at a specific point in the area of an image, a voxel stores values for the volume of space that it covers. Depending on the accuracy and type of implementation it may include many fluid attributes such as density, velocity, colour, temperature, opacity, and so on. The detail of the fluid simulation can be increased by using smaller voxels, so more can fit into the same volume.



[Figure 22] This is a render of a voxel grid. Some voxels are completely opaque, and some are transparent. A wireframe mesh marks the boundary of the voxels, showing the placement of the empty voxels nearest the camera.

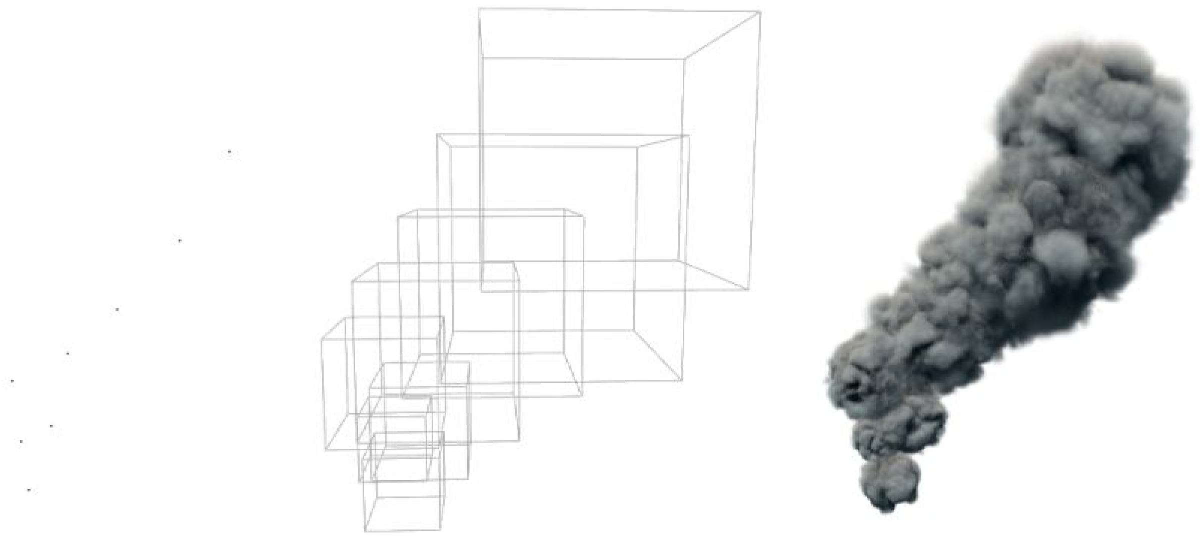
To accurately describe a fluid, the voxels will require different levels of opacity, where increased opacity in a voxel represents a higher density of smoke particles. No actual discreet particles are simulated or rendered however, only their average properties throughout the volume of a voxel. The method by which the smoke's density is represented is relatively trivial compared to the calculation of the movement. To perform the simulation, time is split up into discreet periods called timesteps. Typically there are multiple timesteps per frame, effectively supersampling (increasing simulation accuracy) the simulation relative to the frame rate. At the beginning of every timestep, each voxel's velocity value is increased by the spatially corresponding value in the force field. Velocity is then transferred between adjacent voxels through a process known as advection. This is the transport of a conserved attribute through a vector field [1] – in this case velocity through the fluid.

In real life, every object is compressible to some extent, due to the gaps between molecules. The larger the gaps, and hence the less dense the material, the more compression is possible. Therefore, under pressure, solids exhibit very little compression, fluids exhibit a limited degree of compression, and gases compress significantly. In real life, gases are rarely compressed when their movement is not restricted by a container. An unrestricted gas is only compressed to high degree when a very strong force is acting upon it, for example after an explosion [5]. When simulating fluid dynamics for a purely visual requirement, such phenomena need not be physically simulated and the ability to compress a fluid is ignored. While this introduces physical inaccuracies, it also decreases the number of calculations necessary, while maintaining a visually acceptable appearance. At each timestep of the simulation, the total velocity entering a voxel must be made to equal the total mass leaving. This forcing of velocity conserves mass.

Combining Hypertextures with Particles

One of the drawbacks of using normal particles is that they provide a poor representation of the way smoke reflects light. Each particle has no volume, and as such can only represent a single point in space. This makes it impossible to create an accurate shading model, as light requires a surface with which to interact. It is possible however to quickly achieve somewhat realistic movement of the particles, due to the fact that calculating the dynamics for individual points is relatively fast. When using hypertextures to render smoke volumetrically, the appearance of smoke is greatly improved, due to a shading model that more accurately represents the properties of real smoke. However, there is little control over the movement of the smoke because we cannot apply different parameters to different areas of the texture. Our only possibilities are to change the attributes of the entire texture (for example the noise frequency), or to move the entire volumetric container. Real smoke has a tendency to break up into different sections, and the turbulence is random throughout. To move the entire cloud at once creates an unrealistic movement that would quickly betray the digital nature of the smoke.

It is possible to combine these two techniques, in order to utilise the advantages of both. This will provide smoke that is volumetric, but does not require the use of slow fluid dynamics algorithms. A standard particle system is created, with a low number of particles representing the general movement of the smoke. A sphere of hypertextured smoke is then parented to each particle **(see figure 23)**. This allows a particle system to control the movement of volumetric smoke. The hypertextured spheres can inherit properties from the particles and use them to drive their own attributes. For example, the spheres can increase in radius as the particles age, to simulate the dissipation of smoke. To remove the popping effect as particles die, the hypertexture's transparency can also be faded out proportional to the particle's age.

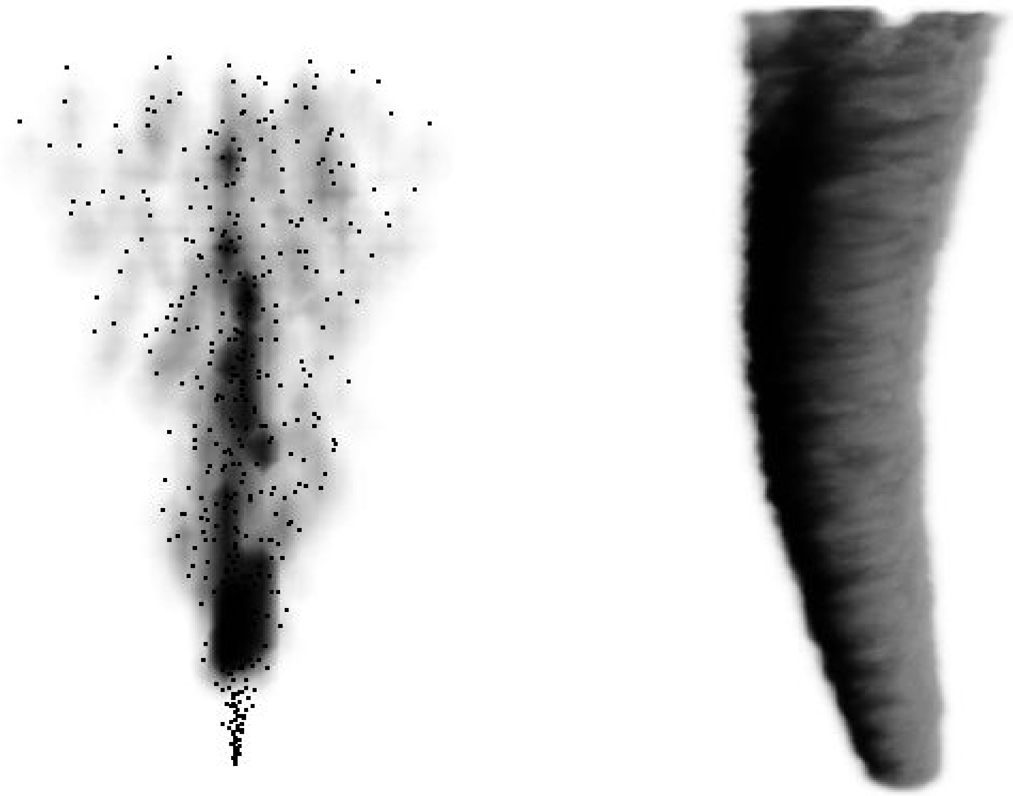


[Figure 23] Here we can see the steps involved in combining particles and hypertextures. On the left is the basic particle system, which needs to contain only a few points. Each point will determine the movement of the entirety of smoke parented to it. The middle image shows the bounding boxes of the hypertextures, which are parented to the particles. On the right is the final image where the hypertextures have been raymarched with self shadowing. The result is very convincing, due to the volumetric nature of the smoke. This means that the intersecting volumes of different hypertextures will interface in a visually correct manner.

The render times using this technique are comparable to those when using fluid dynamics; the finished frame above took one minute to render on a dual 1.8ghz machine. The speed benefit lies in the fact that the time taken to simulate the movement is negligible, due to the very low number of particles. To compute the movement of a similar plume of smoke with fluid dynamics could take many hours. The resulting movement however would be more realistic. This technique is most effective when a long, thin, continuous stream of smoke is needed, for example a column of smoke or a meteor trail. This is because intricate details like vortices cannot be simulated, as it would require the movement of sections of smoke relative to itself. Also, it would be difficult to model the effect of the environment on the smokes movement, since there are so few points with which to adjust movement. A good analogy would be to consider these as three dimensional sprites, because a single point is determining the movement of a larger volume of smoke.

Combining Fluid Dynamics with Particles

Fluid dynamics is the best way in which to achieve realistic looking results, both for the movement and the shading of the smoke. It can be difficult however to control the movement of the smoke. This is not an error in the technique, but rather a side effect of having a physically accurate model – smoke is difficult to control in real life too. To overcome this problem, it is possible to use particles to control the initial positioning of smoke, and subsequently use fluid dynamics to move it **(see figure 24)**. In this way, each particle is treated as a smoke emitter. An example of where this could be used is if one wanted many small streaks of smoke. The particles would leave a trail behind them, which would dissipate in a physically correct manner.

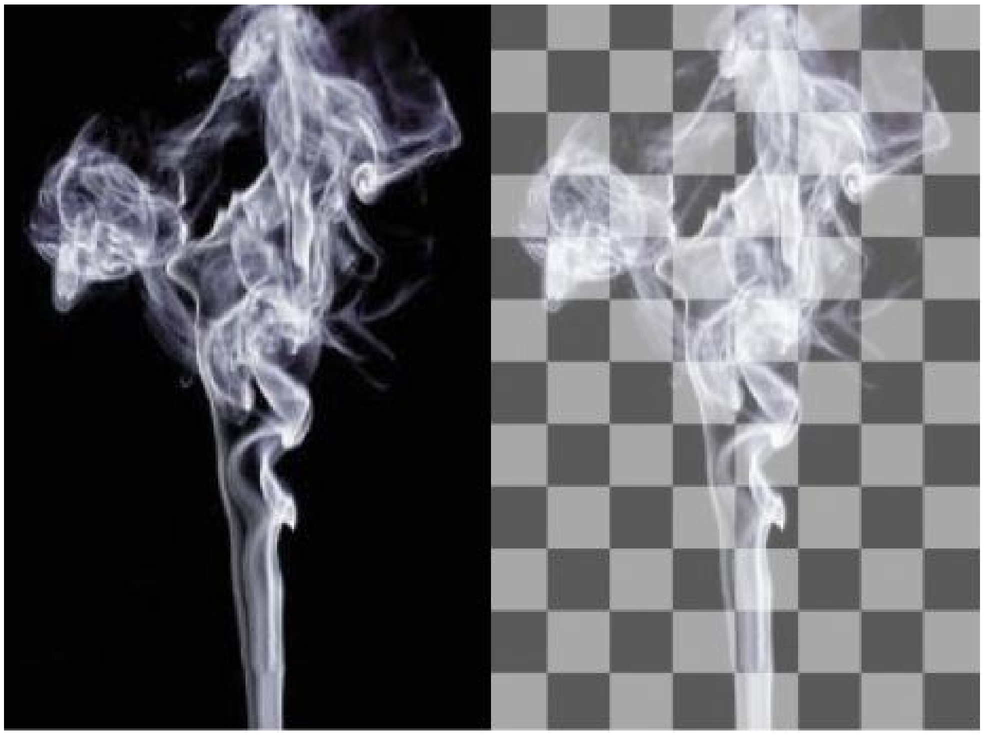


[Figure 24] On the left is a simple implementation of particles seeding the density of a fluid, shown to demonstrate the technique. Once the density has been positioned, it is then under the influence of the fluid dynamic solver, which causes dissipation. On the right hand side is a tornado made using this technique, which would have been very difficult to implement using standard fluid emitters.

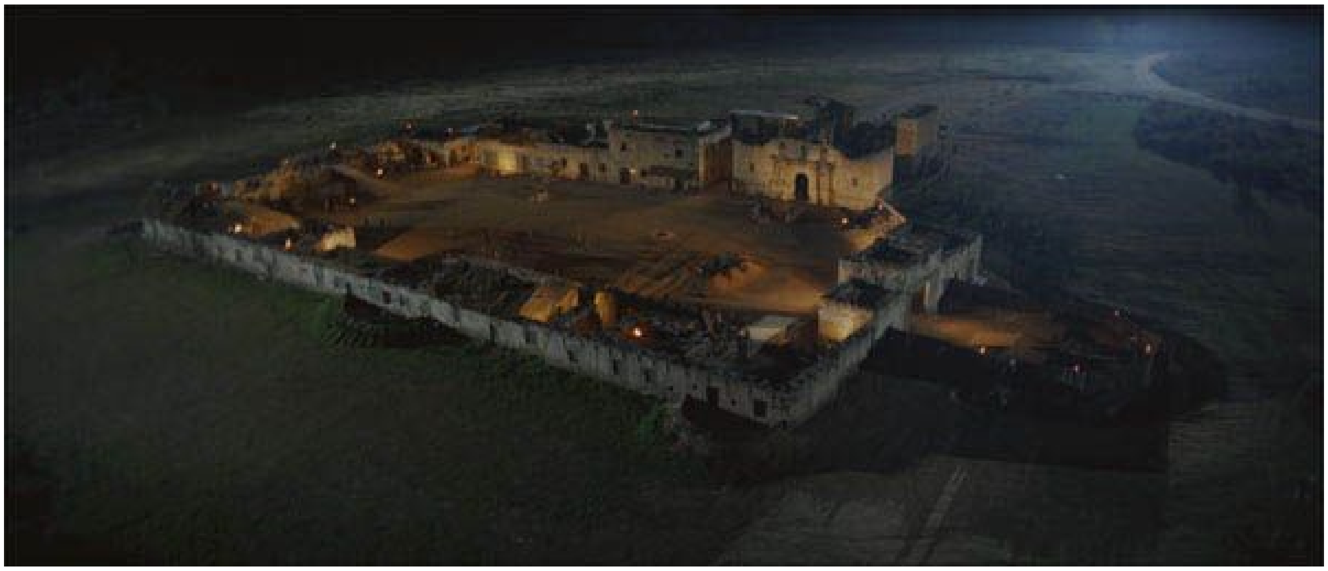
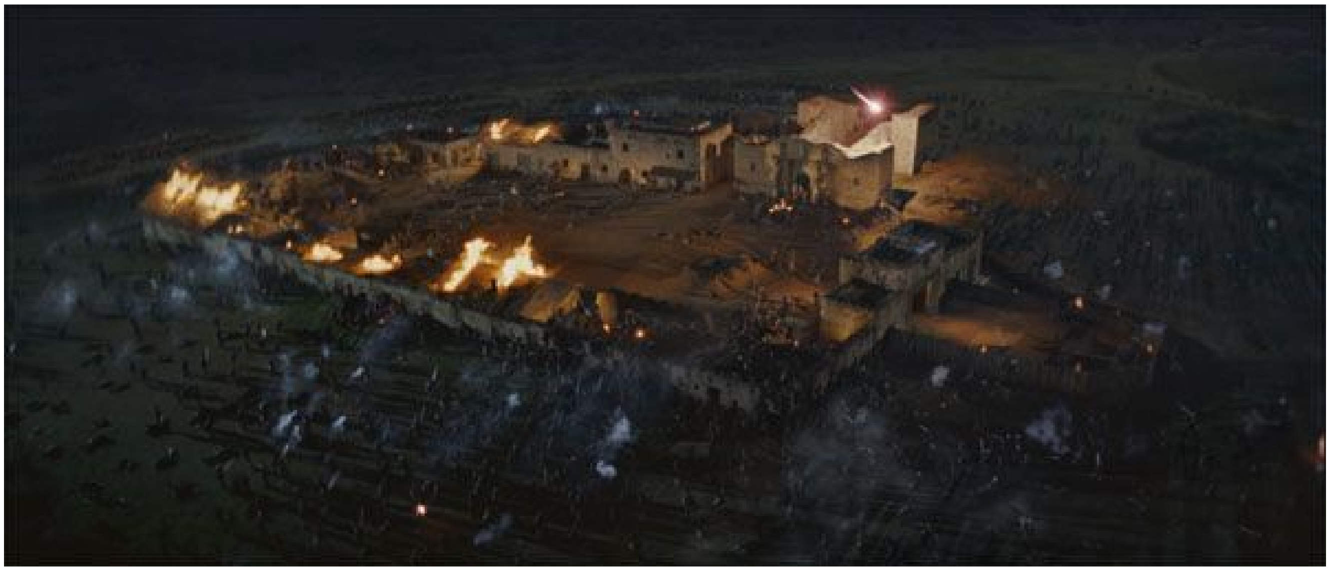
Practical Effects

Prior to the age of digital special effects, the only way to achieve the use of smoke in a shot was to use real life smoke. Since it would not always be practical to do this at full scale, for example a city on fire, techniques were developed to film the smoke separately and then combine it with another shot. The smoke would be filmed once on a black background, and then a procedure known as double exposure would be used [4]. This is where the film is rewound, and then filmed again with the desired background. The effect is that the areas of the frame where smoke was originally filmed will be brightened, such that it appears the two shots were in the same scene.

Even today where other techniques are available, sometimes the most appropriate approach is to draw reference from real life smoke. This gives the advantage of requiring no simulation of movement, and very fast render times. The required smoke can be filmed in front of a uniformly coloured screen, allowing the smoke to be digitally isolated from the background. The smoke elements can then be placed on planes in the environment, or composited in after rendering. By filming the smoke on a black background, it is very simple to create a transparency channel based on the luminance of the image **(see figure 25)**. This method of layering is called screening, and is the digital equivalent of double exposure.



[Figure 25] Here a transparency channel has been created for a photograph of some smoke. On the left is the original image [24], which contains only red green and blue channels, with no transparency. On the right is the image after being placed in front of a chequered background. The result is a convincing visual interaction between the foreground and background.



[Figure 26] An example of real smoke composited into a scene, from the film "The Alamo" [22]. The smoke in the foreground was shot separately from the fort, and layered in afterwards.

Using fire to create smoke is not always the most practical approach, due to the safety issues involved. The smoke created by fire is hazardous when breathed in, and as such can be problematic when filming. Instead, it is common to use different particle suspensions which are easier to manage. A simple method is to add water to solid carbon dioxide (dry ice), which then vaporises creating a fog of carbon dioxide. Due to its low temperature, it will sink, which limits its usefulness when representing normal smoke. Other techniques involve the use of dust like substances like flour, or heated vapours that will move in a manner similar to smoke.

Using footage of smoke has many benefits, which make it a commonly used technique. However, there are situations in which it is not a practical approach to take. The alpha-mapped planes onto which the smoke photography is placed are similar to sprites, in that they do not have a volume. Instead it shows only a two dimensional projection of the smoke in the scene. As such, the technique does not work well where there is intersection with geometry, as the intersection betrays the two dimensional nature of the smoke. An implementation of z-feathering can be used to hide the intersection, however this creates the impression that smoke is disappearing unnaturally, due to the fact it is not retaining volume. This would not look realistic where the smoke was subject to high levels of scrutiny by the viewer. Furthermore, it can be difficult to set up smoke in real life if it needs to match a complex environment. For example, a swinging pipe emitting steam would have to be synchronised with the digital version, and would be impracticably difficult to set up correctly.

Conclusion

There are many techniques available to someone looking to use smoke as part of a digital special effect. Using computer graphics, there are solutions which take a variety of approaches meaning a suitable method can be found for representing smoke in most situations. In conclusion, a summary is presented comparing the benefits and drawbacks of each technique.

– The use of particles on their own is a fast method with which to create smoke where there is little self shadowing. Since they do not model the reflectance of light, they can only represent simple clouds of smoke with near-uniform colour. Particles are very fast to both simulate (calculate their movement), and render (calculate their shading).

– Sprites can be used where the smoke needs to have sufficient density to obscure the background, without having to resort to the use of millions of particles. They are poor at representing volume, and do not hold up well where they need to interact with the environment. Their lighting is inaccurate due to the fact that their normals are determined by simple planes. These artefacts can be limited through the use of z-feathering and bump maps, however these solutions are not perfect. Sprites are fast to render and simulate.

– Hypertextures provide a convincing shading model due to the fact they correctly model the way light is reflected by a partially transparent volume. Correctly animating hypertextures can be difficult due to the fact that their detail is defined procedurally. This can be overcome by controlling segments of hypertexture with particles, in which case the simulation runs at the speed of the particle system. This solution only approximately models the movement of real smoke. Lighting of hypertextures is slow to calculate, however this can be overcome through the use of post lighting. Where more than three lights are required for illumination, this is an efficient and visually convincing method of lighting the smoke.

- Fluid dynamics can create smoke that is indistinguishable from the real thing. If a realistic result is required, this technique will produce the most convincing results. The downside is that like real smoke it is hard to control exactly where the smoke goes. This can be overcome by using force fields to control a particle system, and then emit fluid from the particles. This is then controlled by the fluid dynamics solver, which will move the smoke in a physically accurate manner. The simulation and rendering are very slow with fluid dynamics.

- Practical effects are capable of producing photo-realistic results, due to the fact that they are from a photographic source. Their use requires careful planning, as any interactions with the scene may not be convincing. They are very fast to render, due to the fact that their final form is merely a textured plane. The setup time is high, as filming equipment and a suitable backdrop need to be organised. Where a suitable source library of footage already exists, it is a fast method with which to implement smoke in a scene.

References:

[1] ANSWERS, (date unknown): *Advection* [Online] Available from:
http://www.answers.com/advection&r=67
[Accessed 6 March 2006]

[2] ALIABADI, TEZDUYAR, 1994. Parallel Fluid Dynamics Computations in Aerospace Applications
ln: T. TEZDUYAR, M. KAWAHARA, T. HUGHES, ed: *International Journal for Numerical Methods in Fluids*
John Wiley & Sons, Ltd., 783 – 805

[3] ATI, (date unknown): *Depth Sprites* [Online]
ATI Technologies Inc.. Available from:
http://www.ati.com/developer/samples/DepthExplosion.html
[Accessed 6 March 2006]

[4] BIZONY, P., 2001. *Digital Domain: The Leading Edge of Digital Effects*
London, Aurum Press, 18.

[5] BRINSMEAD, D. Principle scientist at Alias. Personal correspondence, 2 March 2006. Available from:
http://forums.cgsociety.org/showthread.php?t=323426
[Accessed 6 March 2006]

[6] DOLLNER, NIENHAUS, 2004. Sketchy Drawings
ed: *Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa, 2004 Stellenbosch, South Africa*
New York, NY**,** USA. ACM Press, 73 – 81.

[7] FLUENT, (date unknown): *Flow Modeling Examples* [Online] Available from:
http://www.fluent.com/solutions/examples/x209.htm
[Accessed 6 March 2006]

[8] HERBORG, P., 2003. *An Implicit Surface Raytracer* [Online] Available from:
http://www.daimi.au.dk/~herborg/sc/
[Accessed 6 March 2006]

[9] MATHWORLD, 1999: *Vector Field* [Online]
CRC Press LLC. Available from:
http://mathworld.wolfram.com/VectorField.html
[Accessed 6 March 2006]

[10] MCELL, 1999. *3D Raymarching/Raytracing Algorithm* [Online] Available from:
http://www.mcell.cnl.salk.edu/Documentation/Workshops/1999/raytrace.html
[Accessed 6 March 2006]

[11] PERLIN, HOFFERT, 1989. Hypertexture
ed: *Proceedings of the 16th annual conference on Computer graphics and interactive techniques, 1989.*
New York, NY, USA. ACM Press, 253 – 262.

[12] REEVES, W., 1983. A Technique for Modeling a Class of Fuzzy Objects
ed: *ACM Transactions on Graphics (TOG) Volume 21, Issue 10*
New York, NY, USA. ACM Press, 91 – 108.

[13] STAM, J., 2001. A simple Fluid Solver Based on the FFT
ed: *Journal of Graphics Tools, Volume 6 , Issue 2*
43 – 52

[14] VCE, 2006. *Bluescreen and Greenscreen Photography Tips* [Online] Available from:
http://www.vce.com/bluescreen.html
[Accessed 6 March 2006]

[15] WIKIPEDIA, 2006: *Diffusion* [Online] Available from:
http://en.wikipedia.org/wiki/Diffusion
[Accessed 6 March 2006].

[16] WIKIPEDIA, 2006: *Dust* [Online] Available from:
http://en.wikipedia.org/wiki/Dust
[Accessed 6 March 2006].

[17] WIKIPEDIA, 2006: *Force Field* [Online] Available from:
http://en.wikipedia.org/wiki/Force_field_(physics)
[Accessed 6 March 2006].

[18] WIKIPEDIA, 2006: *Particulate* [Online] Available from:
http://en.wikipedia.org/wiki/Particulate
[Accessed 6 March 2006].

Image references

[19] BBC, (date unknown). Turbulent flow of water [Online] Available from:
http://www.bbc.co.uk/science/humanbody/mind/articles/disorders/ocd.shtml
[Accessed 6 March 2006]

[20] Game Rankings, (date unknown): Crashday [Online] Available from:
http://www.gamerankings.com/htmlpagesscreens/925730.asp
[Accessed 6 March 2006]

[21] Grey Bruce Health Unit, 2005: Some Facts about Second-Hand Smoke [Online] Available from:
http://www.publichealthgreybruce.on.ca/Tobacco/SecondHandSmoke/SecondHandSmokeFacts.htm
[Accessed 6 March 2006]

[22] Matte World, (date unknown): The Alamo [Online] Available from:
http://www.matteworld.com/film/2004/alamo.html
[Accessed 6 March 2006]

[23] U.S. Department of the Interior, 2005: Laminar flow of water [Online] Available from:
http://www.usbr.gov/lc/socal/wtrcons.html
[Accessed 6 March 2006]

[24] The Eastern Shore of Virginia Network, 2004: Smoke on black background [Online] Available from:
http://www.esva.net/~lights/turb.htm
[Accessed 6 March 2006]

Acknowledgements

I would like to thank Eike Anderson for guiding me with this project, and Jimmi Gravesen for his work on a raytraced approach to z-feathering.