SoftParticles Sample

Collapse All

Path

Source	*SDK root* \Samples\C++\Direct3D10\SoftParticles
Executable	*SDK root* \Samples\C++\Direct3D10\Bin\ *platform* \SoftParticles.exe

Sample Overview

Particle systems are a common method for creating volumetric effects in games. They can be used to simulate such things as fire, clouds, smoke, dust, glowing projectiles, magic spells, and so on. A common technique is to use 2D camera-aligned quads centered at each particle to represent the volume that the particle represents. Because the quads follow the camera, they give the illusion of a substance filling a 3D volume. However, this illusion often breaks down when the 2D sprites used to visualize the particles intersect with the world geometry. The intersection of the 2D quad with the 3D world geometry creates a hard, straight line — on one side of the line is particle, and on the other is world geometry. Figure 1 shows an example of this.

Figure 1. Hard, Flat Particles

2D Particles with Depth

The first approach to dealing with this is to perturb the depth being output from the pixel shader by a depth stored in the particle texture. This makes the intersections with the world geometry follow the contour of the particle. Figure 2 shows an example of this.

Figure 2. Depth Particles

2D Soft Particles

In the basic flat particle system, the line where the particle quad intersects the geometry is a giveaway that the particles are not actually 3D. To avoid this, the sample can read back the depth buffer as a texture. In the shader, this depth value is sampled and tested against the depth value being rendered for the current pixel in the particle. The alpha value increases as the difference between the depth value in the buffer , and then depth being written out from the pixel shader decreases. Therefore, the particle becomes more transparent as it approaches intersection with the scene geometry.

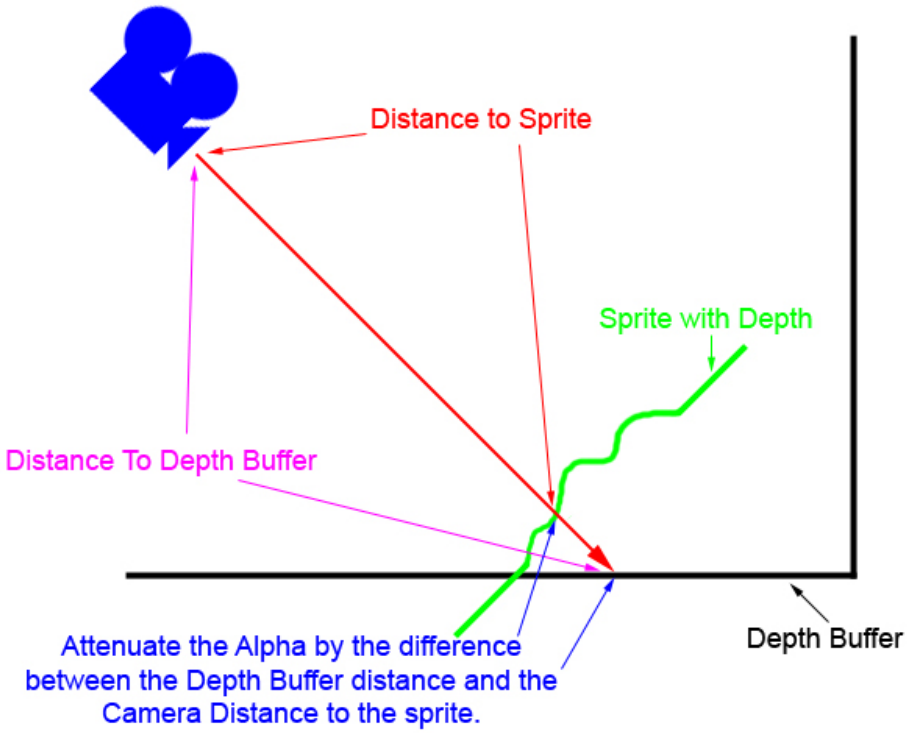
Figure 3. 2D Soft Particles

Depth Soft Particles

The same approach can be used with particles that output depth. The only difference is that the depth compared with the depth buffer is not the depth from the quad, but the depth from the quad augmented by the depth stored in the particle texture.

Figure 4. Depth Soft Particles, example

Figure 5. Depth Soft Particles, explanation



Animating the 2D Particles

To animate the 2D particles, the sample uses a series of animated particle texture *frames* stored in a volume texture. The first slice of the volume texture stores the first frame of the animation; the last slice stores the last. As the particle ages, the shader uses the age (in this case a float in the range of [0..1]) to look up a color in a 1D texture. This color modulates the color of the entire particle and makes the particle appear dark near its birth and lighter just before death.

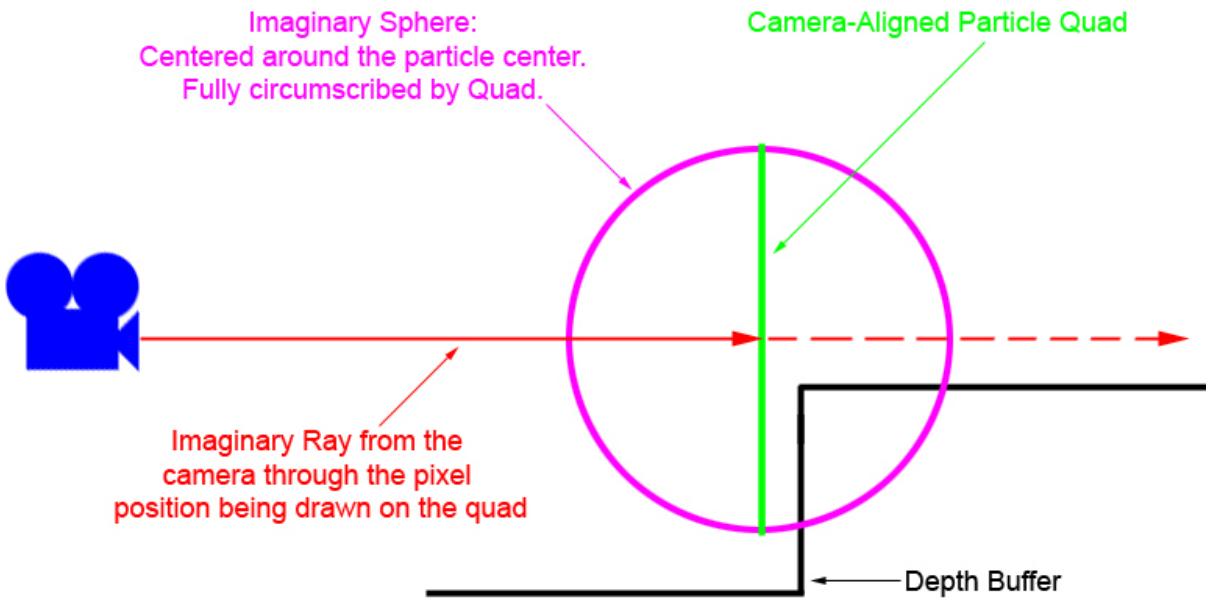
Volumetric Particles

The volumetric techniques in the sample give the particles the appearance of true 3D volume instead of the flat look of 2D particles. To achieve this, the sample traces rays through an imaginary 3D primitive circumscribed by the camera-aligned particle quad. In this case, the sample uses a sphere as the imaginary primitive. There are three main reasons for choosing a sphere. First, it is always possible to

circumscribe a sphere in a camera-aligned quad. Second, a sphere is a very symmetrical shape. It looks the same no matter how you orient it. Third, the intersection between a ray and a sphere is easy to compute.

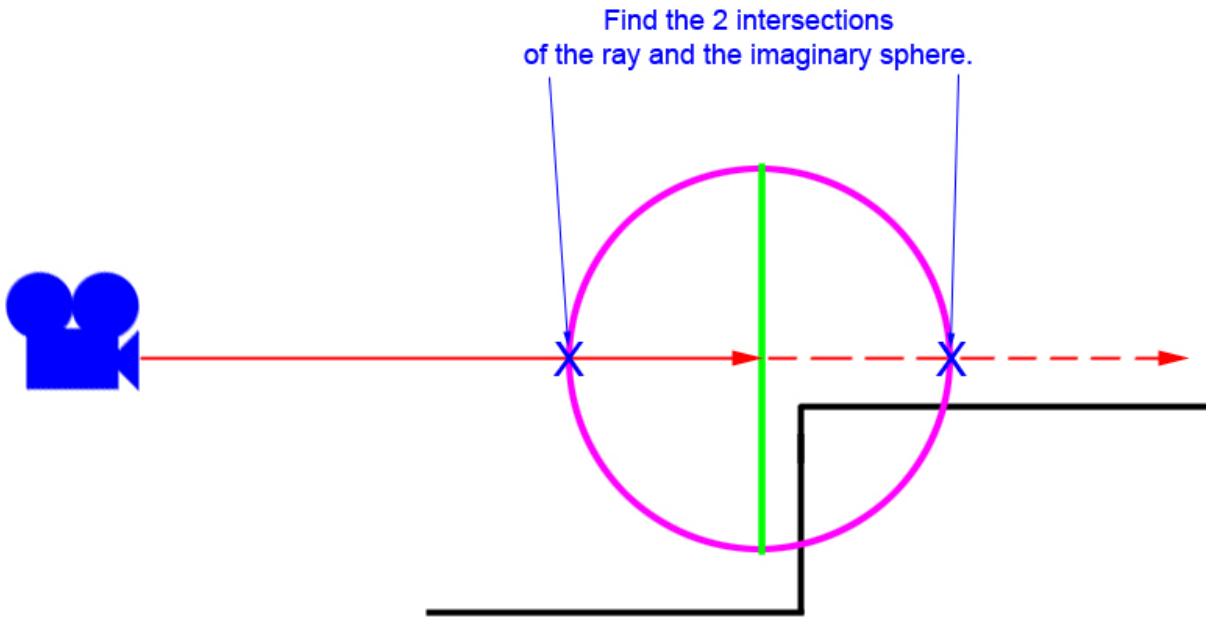
The main process is as follows. The particles are handled in the same manner as in the 2D technique until they reach the pixel shader. The pixel shader computes a ray from the eye to the pixel being drawn on the quad.

Figure 6. Volumetric Particles, Main Process



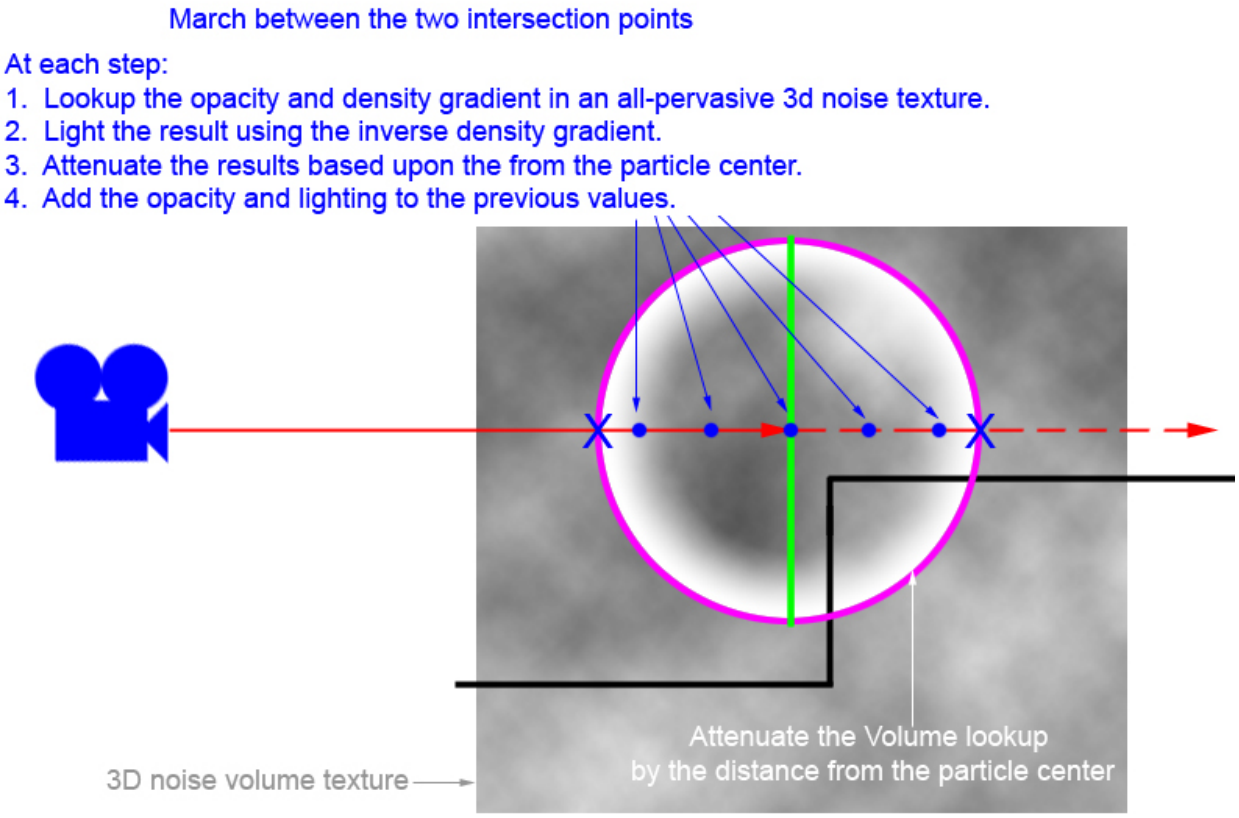
It then calculates the intersections of this ray and the imaginary sphere circumscribed by the particle quad. If no intersections occur, the ray is discarded.

Figure 7. Volumetric Particles, Calculate the Intersection



Using a fixed step-size, the shader determines the number of steps needed to march from one intersection point to the other. Along the way, several octaves of noise are sampled. These are stored in a 4-channel volume texture. The RGB channels contain the inverse density gradient. The alpha channel contains the density. Lighting is computed at the point using a combination of the normal from the center of the sphere at the point and the density gradient. Each step is summed with the previous step, and the final result is the opacity and color value for the ray traced through the volume. This result is modified slightly by some overall opacity values to get a specific look.

Figure 8. March Between the Two Intersection Points



If the ray intersects the depth buffer before it exits the sphere, the shader changes the rear exit point to be the depth buffer intersection. Optionally, to achieve the soft intersection with world geometry, the alpha can be attenuated based upon the distance between the sample and the depth buffer.

Figure 9. Ray Intersecting the Depth Buffer

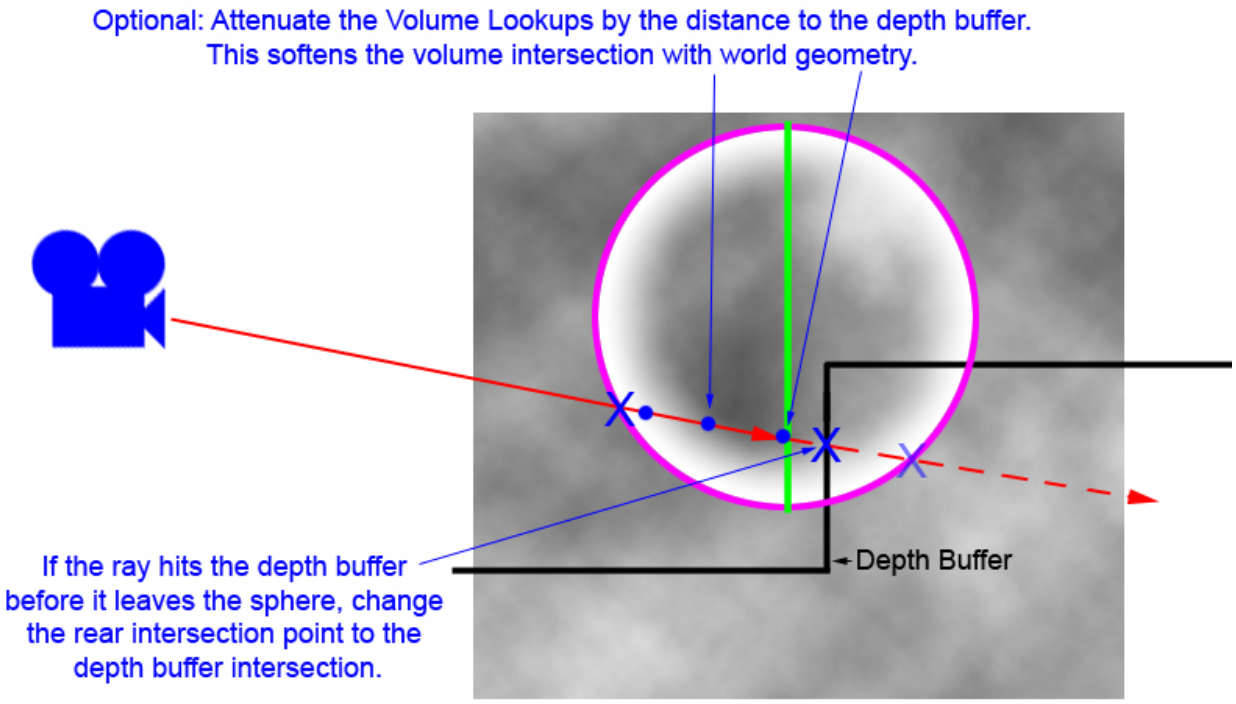


Figure 10. Hard Volume Particles

Figure 11. Soft Volume Particles

Animating the Volume Particles

Although the particles are animating (moving away from the source), added detail is gained by animating the texture coordinates used to calculate the lookup in the noise volume texture. By moving the texture coordinates in one direction, the smoke is made to look like it is moving in the opposite direction.