IOT PROJECT -PHASE 4
Smart water fountains
7125-PPG INSTITUTE OF TECHNOLOGY COIMBATORE

Submitted by
712521104019-R.MANORANJITHAM
712521104020-G.MAYURI

Spectators greatly appreciate majestic musical fountains such as the Bellagio music fountain in Las Vegas and the Magic Fountain of Montjuic in Barcelona. A plethora of water jets and colored lights gives these fountains a gorgeous appearance. But what further distinguishes them is that their displays are synchronized with accompanying music.

Fountains that just make patterns with water jets have now developed into multimedia shows with music, light, and special effects. A musical fountain with synchronized water and music creates an atmosphere that can be exciting or romantic. The fountains in Las Vegas and Barcelona are huge, but smaller musical fountains are now appearing in many urban spaces.

Whatever the fountain size, the choreography requires painstaking, expert programming, and their creators can only achieve this after careful analysis of the music. Skilled musical-fountain programmers can spend days or even weeks creating a new performance, but this is expensive. So, they rarely change the routines, and they repeat a limited program every day. For this reason, programmers usually choreograph the routines to classical music or well-known pop songs rather than very new music.

There have been several attempts to write software to automate a musical fountain's response to a piece of music. However, these schemes are based largely on the music's simple audio features, such as volume. These programs are seldom used to control real music fountains because the shows that they produce are rather prosaic. Even when experts create musical fountain shows, the shows usually require several trial runs. Yet, they often omit these trial runs because of the cost and inconvenience involved in taking a fountain out of commission. Some programmers use computer simulations to avoid this problem. However, these simulations'

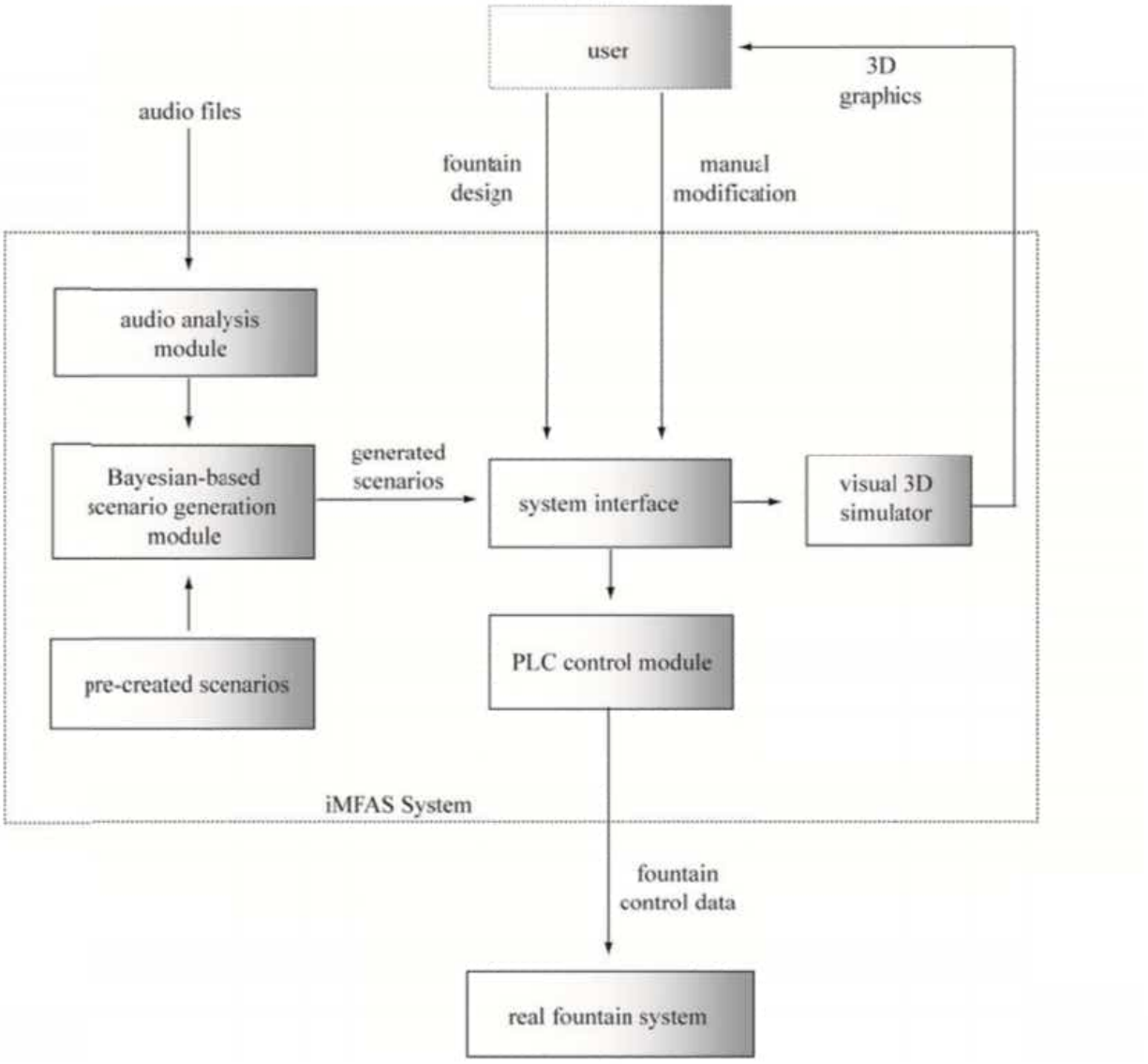

Figure 1. Overview of our intelligent musical-fountain-authoring system (Imfas). The system largely provides the fountain design, musical-fountain scenario generation, and control of a programmable logic controller (PLC). The PLC automates the electomechanical process and is widely used for controlling fountain hardware.

Designing a fountain

As Figure 2 shows, a user can design a new fountain with any configuration of nozzles and lights through a 2D interface and can then simulate a resulting 3D display.

Users can model the water shapes that most nozzles create by modifying a straight jet of water. Figure 3 shows the modifications our system supports. Figures 3a and 3b show a fixed nozzle, which creates a jet of height V_{height} and width V_{width} at an angle of θ_{init} to the ground plane. The jet's projection on the ground plane makes an angle of θ_{init_dir} with the predefined axis. Figure 3c shows a nozzle that can also move through θ_{left} and θ_{right} angles from its original direction; θ_{ang} represents the angle between this movement's plane and the axis, and V_{ang} represents the movement's speed. Figure 3d shows a rotating nozzle, with a velocity of V_{rot}. By adjusting all these values, users can create jets of desired shapes.

Designers can group and operate several nozzles of the same type to produce more-complicated shapes. To help design these composite nozzles more easily, our system provides some templates.

After setting the parameters to create a nozzle, or selecting a predefined nozzle template, users can start laying out a fountain using a geometric tool to move and align each nozzle by dragging the nozzles to various locations on the screen (see Figure 2a). They can then simulate the resulting fountain in real time (see Figure 2b). Users can also group several nozzles to generate a control unit so that they can operate these nozzles together.

In most musical fountains, lights are near the nozzles. A single-colored light usually requires on/off operation, but some lights can produce different colors by mixing (for example, a combination of red, green, and blue lights can make seven colors). If a light is turned on when the water is spouting through the nozzles, the water takes on that light's color. We let each nozzle have its own light, and users can set the color.

A large musical fountain such as at the Bellagio will have well over 1,000 nozzles, and a medium-sized fountain has roughly 500. Our system can model this many nozzles, provided that a high-level GPU core supports the graphics. We'll discuss this aspect in greater detail later. However, a simulation is often more effective if the user models a limited number of representative nozzles.

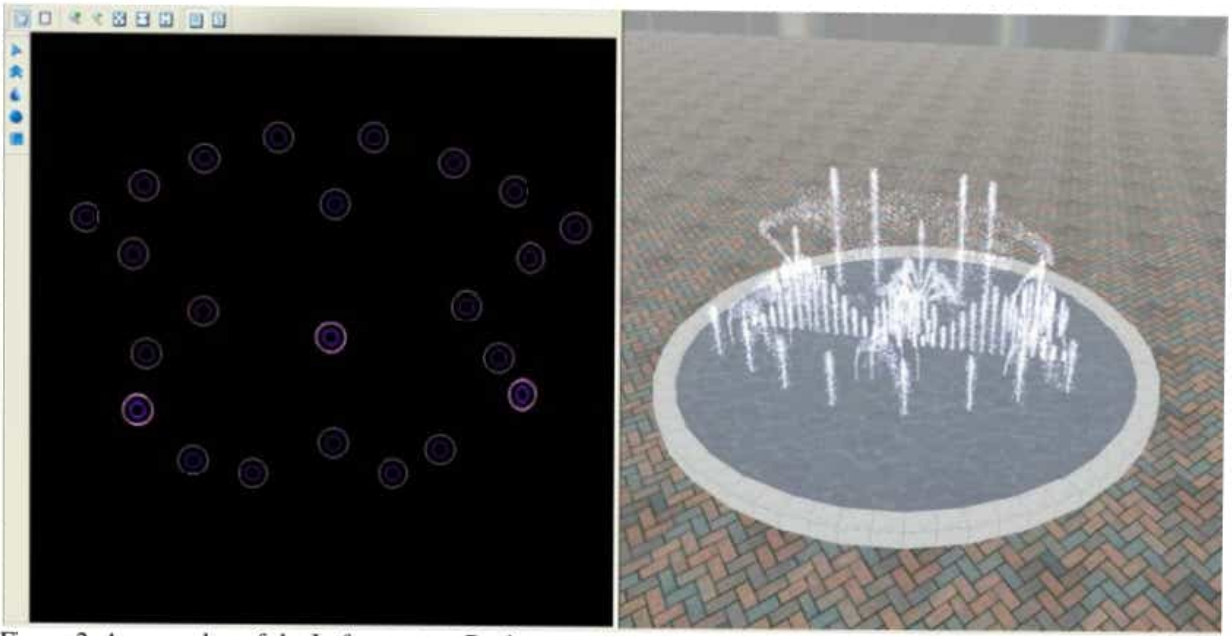

Figure 2. A screenshot of the Imfas system. Designers can use a 2D input interface (on the left side of the screen) to create a 3D simulation in the middle of the screen. A window for setting parameters is on the right side of the screen. Along the bottom of the screen is the timeline from Figure 9.

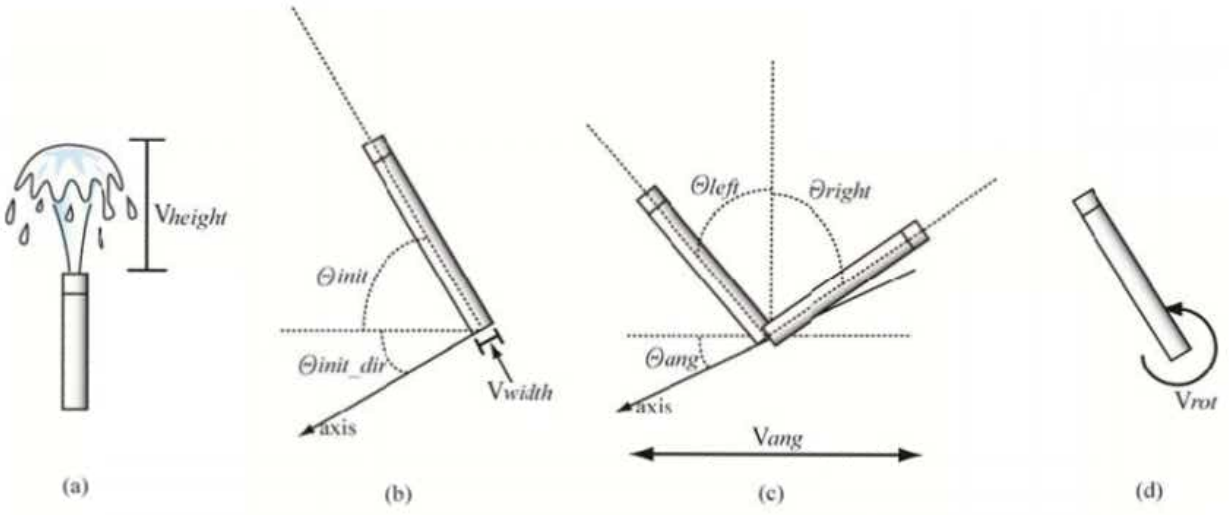

Figure 3. Nozzle design parameters. Users can model the various nozzle shapes by modifying a straight jet of water using several shape-related and movement-related parameters.

Automatic generation of scenarios

Our system's main feature is the automatic generation of musical-fountain scenarios. Here, we briefly explain how a fountain works and introduce the generation method based on audio-signal analysis and a probability-based network.

Fountain mechanism

Briefly, a fountain works using water pumps—usually under the fountain—that force water through different-shaped pipes to the nozzles, which correspond to the style of jet they're intended to produce. Each pump corresponds to a fountain's control unit because all the jets that it powers operate together (see Figure 4a). The water from a pump can also be directed by solenoid valves, which control the water's flow to particular nozzles (see Figure 4b). Large fountains with many nozzles usually use solenoid valves. We refer to a water pump or solenoid valve as a control unit.

Detecting onsets

A fountain scenario's quality depends heavily on music and jet synchronization. Our system achieves synchronization by extracting temporal music features that are appropriate for scenario generation. It detects onsets in the audio signal and changes the control units' operation at these times to correlate the musical-fountain scenario with the music.

There are several ways to detect onsets. We use spectral flux, which we can compute quickly and accurately using a Fourier transform—Figure 5 shows this diagrammatically. We calculate the amplitude $X(n,k)$ of the nth frame and kth frequency bin in an audio signal's Fourier transform as follows:

$$X(n,k) = \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} x(hn+m)w(m)\exp\left(-\frac{2i\pi mk}{N}\right),$$

where $x(n)$ is an audio signal in the time domain, $w(m)$ is a function of size N or a smoothing kernel, and h is the size of a hop, or time shift, between adjacent windows. A Fourier transform analyzes the input audio signal and calculates the amplitude of each signal's audio frequency in the window. We use the Hanning window, and the number of samples is 2,048. The hop size is 439 samples, so windows

overlap 78.5 percent with each other. Input audio files are in stereo and are sampled at 44.1 kHz, and we use 512 frequency bins in the Fourier transform.

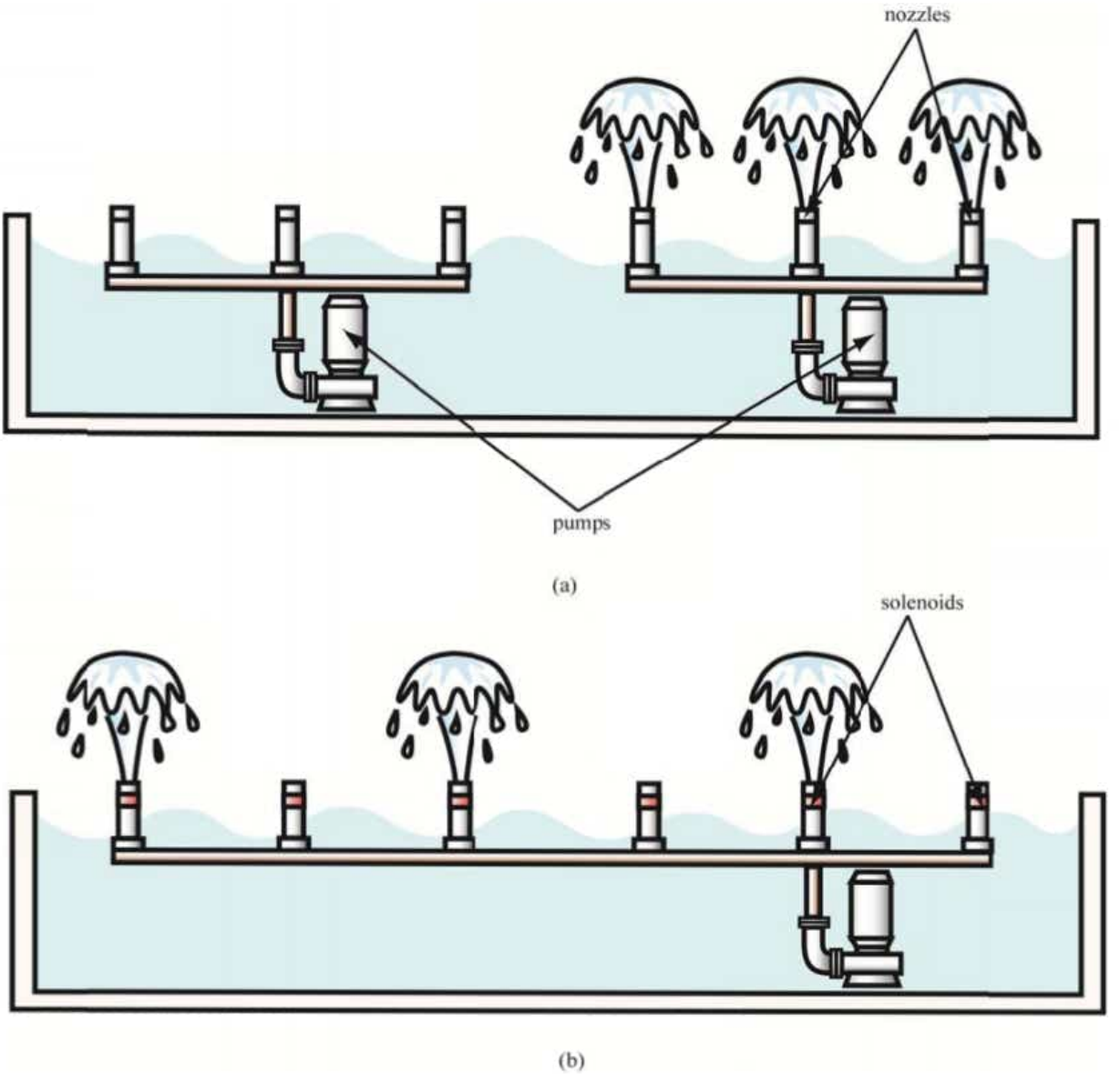

Figure 4. Fountain mechanics. Two units typically control the water's flow: (a) The basic control unit is a pump that forces water through pipes to the nozzles, and (b) the other control unit is a solenoid valve that controls the flow to particular nozzles.

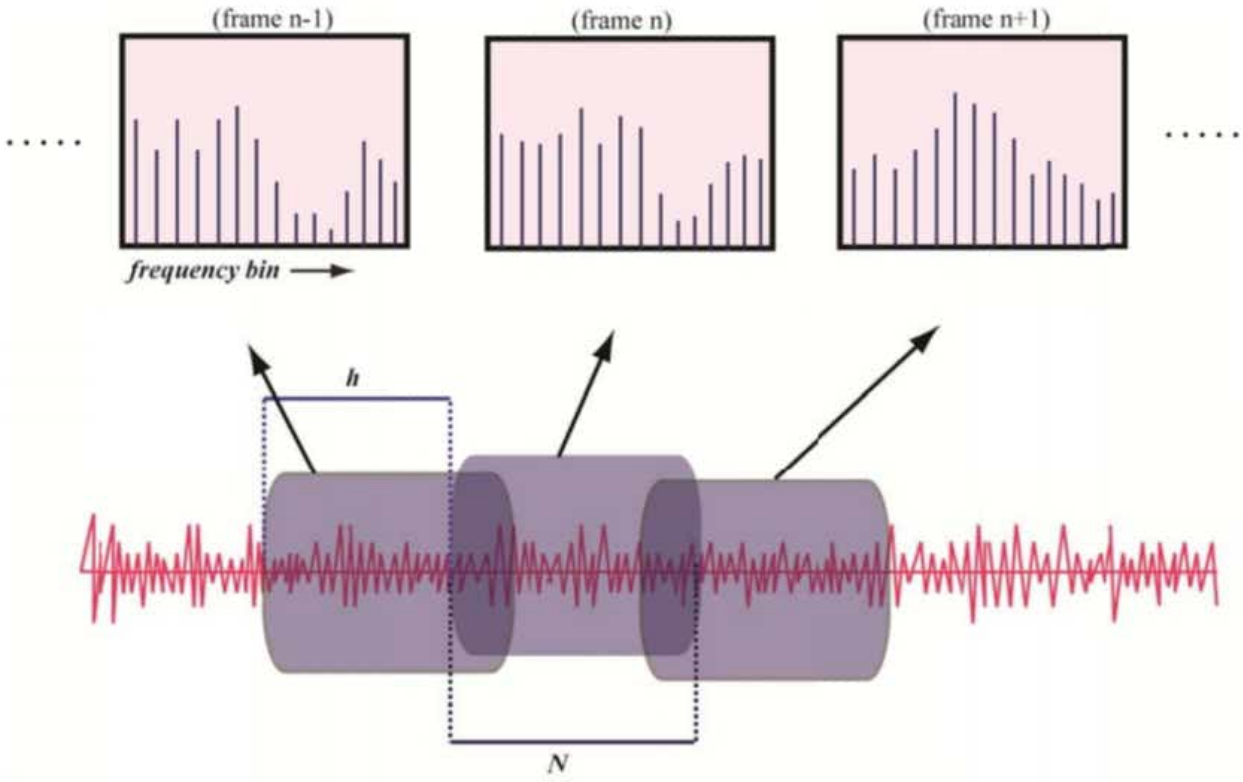

Figure 5. Diagrammatic representation of a Fourier transform. The transform analyzes frequency information of input signals. The transform takes several windows (size N) of the signal and analyzes it in the windows. The windows usually overlap, and the time shift between adjacent windows is called a hop size (h). The Fourier transform information in each window is called a frame.

We can now determine the spectral flux as follows:

$$SF(n) = \sum_{k=1}^{N} H(|X(n,k)| - |X(n-1,k)|,$$

where $H(x)$ is the half-wave rectifier function, such that $H(x) = (x + |x|)/2$.

The spectral flux expresses the differences between the power spectrum of the current and previous frames. An onset event will likely correspond to a significant change between the power spectrums of two successive frames (see Figures 6a and 6b). We use a peak-picking process to select the exact onsets, which must fulfill this condition:

$$SF(n) \geq SF(k) \text{ for all } k \text{ such that } n - w \leq k \leq n + w.$$

The parameter w controls the number of input-audio-signal onsets selected (see Figures 6c and 6d). Users can choose this parameter, which is directly related to the resulting fountain scenario's "busyness." For more on onset detection, see the related sidebar.

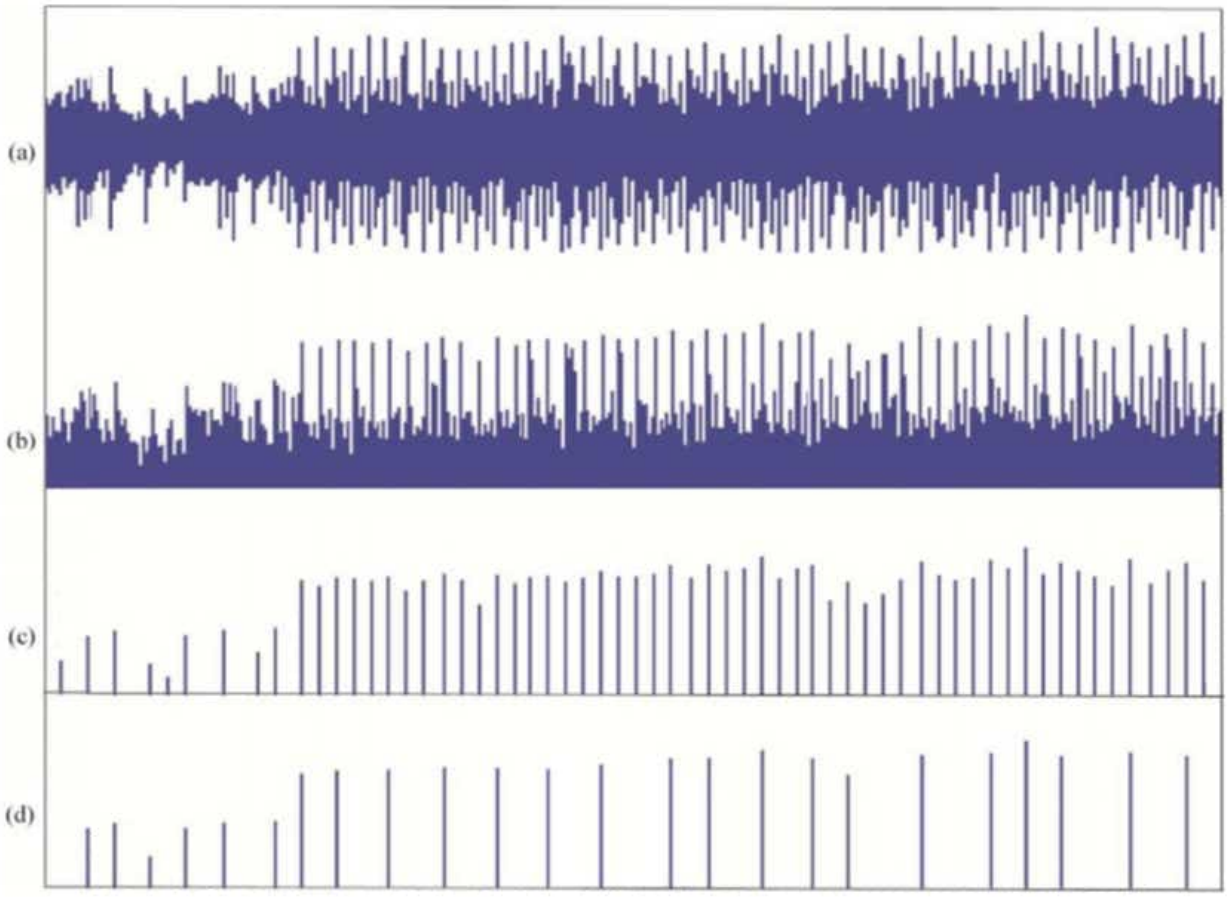

Figure 6. Onset detection. (a) The original audio signal. (b) The original audio signal's spectral flux. (c) Peak picking with w = 100. (d) Peak picking with w = 150. The parameter w, used in the peak-picking process, controls the number of selected onsets.

Visual 3D simulation

To immediately verify automatically generated scenarios, you can use a particle model to simulate the jets of water moving under gravity and then render the particles. We've implemented our own particle dynamics engine and use Nvidia's CUDA (Compute Unified Device Architecture) GPU to perform rendering by shading. In our test, we used a Nvidia Geforce 9800 graphics card.

Each particle represents a small volume of water. Initially, our system gives it an initial position and velocity, which are updated at each time step:

$$p_{new} = p_{old} + v_{old} \cdot t$$
$$v_{new} = v_{old} + a \cdot t$$

where p is the particle's position, and v is its velocity. A particle's acceleration a corresponds to gravity. The time parameter t controls the particle simulation's playback speed.

We assume that particles don't collide. This simplifying assumption greatly improves the simulation's speed because the GPU can calculate each particle's movement in parallel. Particles are deemed to disappear if they fall to the ground or if a given time has expired. These particles are then recycled.

The system directly uses the particles' positions after every time step for rendering in the GPU. The system generates a cube at each particle's position using the vertex shader. Then, spheres replace the cubes via the pixel shader. Each sphere's radius is inversely proportional to the time elapsed since the start of the corresponding particle's movement. This is because a compact volume of water coming through a nozzle gradually spreads out to become a cloud of small drops. The system also stores lighting data in the GPU and applies it to each sphere.

We use from 500 to 1,000 particles to simulate water spouting from one nozzle. We can simulate more than 1,000 nozzles in real time using our GPU implementation.

The positions of the particles after every time-step are directly used for rendering in the GPU. A cube is generated at the position of each particle using the vertex shader. Then the cubes are replaced by spheres using the pixel shader. The radius of each sphere is in inverse proportion to the time elapsed since the start

Controlling fountain hardware

A programmable logic controller (PLC) generally controls the water pumps, the solenoid valves, the lights, and the equipment that moves the nozzles. The PLC is interfaced with a computer, usually through a serial port with RS-232, which is a standard of serial binary data connecting, or a LAN.

The blocks on the timeline specify when the control units are turned on and off, and the system converts them into control data and sends them to each PLC channel. This assures that the scenarios Imfas generates can be directly exported to real fountain hardware without converting them into specific control system formats. We've implemented a software module to control several kinds of PLCs.

High-quality offline rendering

A graphical simulation of higher quality than what we could generate in real time could be useful for presenting a completed scenario. We can generate high-quality offline video as follows:

1. We execute fountain simulations in RealFlow (www.realflow.com), which transforms the simulation particles' trajectories into meshes.
2. We then access the stored mesh data from Maya (www.autodesk.com/maya). We can then set the location or mesh-data scaling efficiently using a Maya Embedded Language (MEL) script.
3. Finally, we perform rendering in Maya using the "ocean" material model.

Figure 10 shows a frame from a video we generated using this method.

```python
import time

def water_fountain():
    print("Water fountain is running...")
    time.sleep(2)  # Simulating water flowing
    print("Water is flowing!")
    time.sleep(1)
    print("Water fountain is turned off.")

# Call the function to run the water fountain
water_fountain()
```

This program defines a function called `water_fountain()` that simulates the water fountain. It prints out messages to indicate the different states of the fountain, and uses the `time.sleep()` function to introduce delays for a more realistic simulation.

Here's the output of the Python program for the water fountain:

Water fountain is running...

Water is flowing!

Water fountain is turned off.

Real-time music input

One of our aims is to create musical-fountain shows in real time from a variety of music sources. We've developed a prototype in which the system can analyze an audio stream in real time, and the onsets the system detects immediately modify the fountain show. Although implementing a real-time technique poses several difficulties, such as noise, we're exploring more-elaborate techniques such as real-time beat tracking. Our goal is to let users perform a musical-fountain show immediately based on the music from an onlooker's portable MP3 player or a real musical performance.

Conclusions

After watching a famous South Korean musical-fountain show for several days, we noticed that the scenarios were always the same. The fountain's operators confirmed that they'd changed the scenarios only twice in three years because of budget constraints. Regardless of how spectacular a fountain show is, if the music and scenarios remain the same, the show can lose its appeal. So, we've developed Imfas to improve this situation.

Because scenario generation depends on audio-signal analysis, including better or different analysis techniques should lead to improved, or more versatile, scenario generation. For example, the system could use voice-separation techniques—which try to isolate a singer's voice from an instrumental accompaniment—to detect a vocal passage's beginning and end. This would provide additional onsets that the system could use to synchronize fountains' jets. Probability-based approaches such as a Bayesian network tend not to produce scenarios with large-scale coherence (for example, a scenario in which all the control units are turned on in sequence). We're now exploring additional scenario-generation methods based on patterns to support these more structured scenarios. We believe that Imfas could reduce musical-fountain programming costs, which should allow more changes. We're beginning to market this system, and you can find out more about our project at http://visualcomputing.yonsei.ac.kr/project/mf/.