Towards More Realistic Sound in VRML

Sean Ellis*

Superscape VR plc†

Abstract

While many pleasing effects are possible using the current VRML sound model [9], it falls short of producing convincing aural environments. Sound sources are not easily affected by their environment, and workarounds for this often increase file sizes dramatically.

This paper presents possible methods for defining additional sound cues in such a way as to allow rendering of ambience, time-of-flight delays, and Doppler shifts within the limited processor power afforded by typical VRML browser systems.

The proposed system will be easily scaleable to any number of sound sources and environmental parameters at any time. In addition, identification of aurally less important elements can be made simple and automatic, allowing dynamic load balancing with other tasks

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: 3-Dimensional Graphics and Realism - Virtual Reality

Additional Keywords: virtual worlds, virtual environments, sound, auralization, infinite impulse response filters.

1. AURALIZATION PROPERTIES

The current sound model takes into account only properties of each sound source itself, and the listener's position relative to it. This is equivalent, in visual terms, to rendering only the light sources in a scene.

A full auralization of the scene [6] is too expensive for modest hardware to perform in real time, so we must concentrate only on the most important aspects of the aural environment.

1.1 Sound Sources

Sound source properties should be consistent with existing VRML worlds, and the existing VRML property set is largely adequate for specifying the shape of the sound field emanating from a source and the pitch at which it is played.

* sellis@superscape.com

† Cromwell House, Bartley Wood, Hook, Hants RG27 9XA, England.

1.2 Listener

The parameters for the listener are dependant on the individual user rather than the environment, and are thus properly addressed as a browser set-up and implementation problem rather than an issue for the world designer. However, for the implementer's sake it is worth reviewing the properties of the listener.

A complete listener model should take into account a head-related transfer function (HRTF) which applies different filter parameters to a sound depending on its direction relative to the head of the listener. An accurate HRTF model is surprisingly complex and is computationally expensive without specialized hardware.

A much simplified HRTF just takes into account the position and direction of the listener's ears, substituting the subtle filtering effects of the head and outer ear with a relatively cheap calculation of different volumes for each ear. This is already mandated by the VRML specification. More accurate direction perception can be accomplished at little extra cost by also modelling the time difference between the two ears.

1.3 Environment

The current VRML sound model specifies no properties for the environment of a sound, so this is where most improvement is required.

Noticeable effects of the environment on sounds include attenuation with distance, time delay, Doppler shift, ambience in interior spaces, occlusion of sound by intervening objects, reflection, refraction and interference.

The earlier items in the list are relatively easy to model, becoming more and more expensive until interference, which requires an approach analogous to ray tracing.

Attenuation with distance is already modelled, and is analogous to perspective in visual environments.

Time delay and Doppler shift are important consequences of the low speed of sound. However, the diversity of applications for VRML imply that the sound speed should be settable by the world designer. For entertainment, especially, users have been conditioned to expect sound and vision to coincide exactly, even where large distances are involved. The sound speed must therefore be specified as a parameter of the world.

Ambience and occlusion are more expensive effects. Both can already be modelled to some extent by overlapping different sound fields playing pre-filtered samples, but the applications for this often involve looped ambient sounds. In order to reduce the apparent repetition of these sounds, they are often large, and this is exacerbated by having to download multiple copies.

By specifying transformations on these sounds, we can download a single instance of the sample data and use it again and again in different contexts.

2. PROPAGATION EFFECTS

Both time delay and Doppler shift can be explained and modelled effectively by regarding the propagation of sound.

The delay between a stationary listener and a stationary source is almost trivial to calculate; it is simply the distance between them divided by the speed of sound.

For a moving listener, the listening position will be different at the beginning and end of any particular period. Thus, over this period the portion of sound heard can be more or less than the actual sound emitted in the same period.

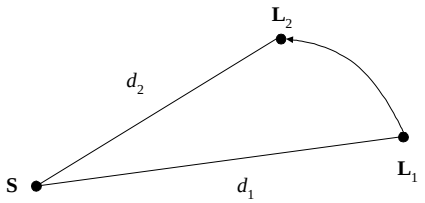

Figure 1. Stationary Source With Moving Listener

Figure 1 shows the situation of a moving listener **L** and a static source **S**. The time delays from the source at the two listener positions are proportional to the two distances d_1 and d_2.

So, if the source starts playing at time t_0, and at time t_1, the listener is at distance d_1, the sound sample heard will be s_1 seconds into the sound, where s_1 is:

$$s_1 = t_1 - t_0 - d_1 D \quad (1)$$

And D is the propagation delay per metre (the inverse of the speed of sound). A similar expression holds for the sample at time t_2. It is possible that one or both of s_1 or s_2 may be less than 0; if this is the case, then the sound has not yet reached the listener position.

The pitch shift factor associated with the motion will be:

$$Shift = \frac{t_2 - t_1 - (d_2 - d_1)D}{t_2 - t_1} \quad (2)$$

For a moving source, the modelling is somewhat more complicated, as the position of the source is required as it was when the emitted sound reaches the ear. In general, this is difficult to calculate, especially if the source is moving supersonically, but can be simplified if we assume that the velocity of the source is subsonic and constant over the time of emission of the sound. The position of the source at a given time before the present then becomes a linear equation in position and velocity.

3. AMBIENCE

The proposed sound environment model would specify ambience as a set of sound transformations over the same type of ellipsoidal volume that is used by the sound source at present.

Users are already familiar with these volumes, and it makes it very easy to specify transformation volumes that match existing sound volumes exactly.

In addition, we will need to specify two sound transformation functions (in some way), T_{near} and T_{far}. Obviously, T_{near} is to apply when the listener is near the source, and T_{far} is used when the listener is far away. A plan view of a typical sound volume is shown in Figure 2.

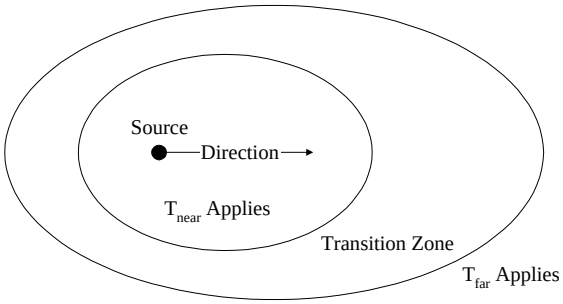

Figure 2. Areas Of Application For Sound Transforms

The relative weights of transforms T_{near} and T_{far} are defined so that they always sum to 1 within the transition zone. In practice, a simple linear combination will usually suffice.

An example of this type of transformation is to mimic the effect of selective attenuation with distance. The sound of a distant engine is muffled, as the higher frequencies do not propagate as efficiently as lower frequencies. In this case, T_{near} would be an identity transform and T_{far} a low-pass filter.

In addition to per sound source transforms, an ambient sound transformation should also be introduced as a bindable node. This specifies a transformation applied to all sounds on the current listener, and is most often used in enclosed volumes such as buildings. The binding and unbinding of this node would be performed by scripts or proximity sensors.

4. OCCLUSION

Occlusion can be modelled along line-of-hearing from the sound source to the listener. A spherical volume can be specified at an arbitrary position together with a sound transformation.

The transformation specified by the occlusion volume is not applied abruptly; the soft edges required can be approximated by weighting the contribution of an effective absorbing volume by the apparent displacement of the volume from the sound source, and also by its apparent size.

Since the function is approximate and not physically based, it need not be entirely accurate as long as it captures the required qualities expected from sound occlusion. This can be a quite simple function based on the vectors to the sound source and to the occlusion volume, which are easily obtained from the scene.

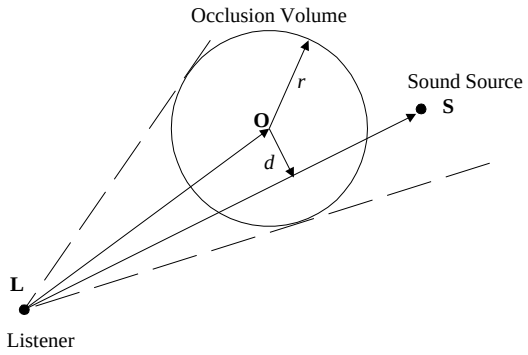

Figure 3. Model of Sound Source Occlusion

Figure 3 shows an effective absorbing volume at **O**, with radius *r*, which occludes the sound source **S** as seen from the viewpoint **L**. A very simple model of absorption would weight the contribution from 1 if **LS** goes through **O** ($d = 0$), through to 0 if **LS** is tangent to the circle ($d = r$).

For $d \geq r$, the displacement weighting W_d is 0, otherwise it is

$$W_d = \left(\frac{r - d}{r} \right) \tag{3}$$

In addition, this would be weighted by another factor based on the apparent size, *a*. This is determined by the radius and distance, and an exaggeration parameter, *k*:

$$a = k \frac{r}{|\mathbf{LO}|} \tag{4}$$

The apparent size weighting W_a needs to go from 0 when a=0, to 1 when a=∞ (covering the entire screen).

$$W_a = 1 - \left(\frac{1}{a + 1} \right) \tag{5}$$

The total weighting factor *W* is simply the product of W_d and W_a, and this reduces nicely to:

$$W = \frac{k(r - d)}{kr + |\mathbf{LO}|} \tag{6}$$

Note that even though this model is a very crude approximation to the actual effects involved, it will almost certainly perform well enough for the majority of users. In fact, even cruder approximations for |**LO**| and *d* based on Manhattan distances will almost certainly produce believable effects.

5. SOUND TRANSFORMATIONS

The format for sound transformations must allow for various different effects. The most common of these are low pass filters, high pass filters, reverberations and echoes.

Perhaps surprisingly, all of these can be represented and implemented using a single model - the Infinite Impulse Response Filter (Figure 4.) [4]

A sampled infinite impulse response filter takes a stream of input values (x_n) and produces a stream of output values (y_n) based on the current value and some previous values of both the input and output streams.

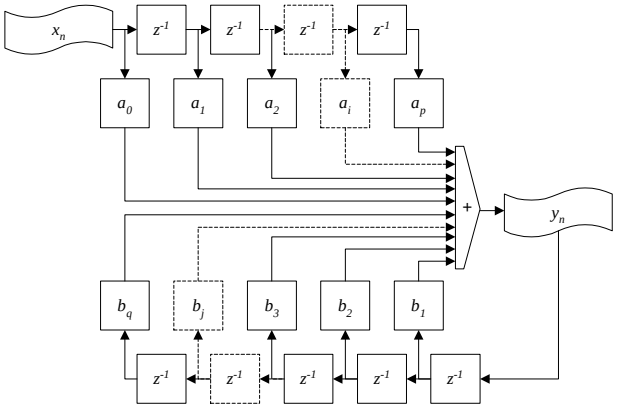

Figure 4. Infinite Impulse Response Filter

The symbol z^{-1} is used to represent a time delay of one sample (we will see why later on). Samples delayed from before the start of the signal are assumed to have a 0 value.

The values a_0 to a_p and b_0 to b_q are the parameters of the filter - their values determine the filter's response to the input signal. Note that the number of input parameters *p* need not be the same as the number of output parameters *q*, and also that there is no parameter b_0.

By treating the input and output as mathematical functions, and by using the trick of regarding z^{-1} not as a time delay but as a complex variable, some techniques become available to us that allow design of these filters based on the required filter characteristics.

When designing a sound transformation, we need to obtain its impulse response. This is what the output would be for a very powerful, short pulse of input. For echo effects, we can model the way that echoes are distributed in time and directly specify the impulse response. From this we can now use a *z*-transform to convert it into an equation in *z* [2].

For a filter, we want to set its frequency response (i.e. how it cuts and boosts various frequencies). Although it is possible to determine the impulse response and then use a *z*-transform, various effective methods exist for directly deriving the *z*-transform from a given frequency response.

In general, the *z*-transform *F(z)* of a given function *f(n)* (with sample interval *T*) is:

$$F(z) = \sum_{n=0}^{\infty} f(nT) z^{-n} \tag{7}$$

This can always be expressed as a fraction of two polynomials in the complex variable z^{-1}. This can in turn be rearranged into a canonical form, with the parameters a_i and b_j from Figure 4 directly represented by coefficients of z^{-1}.

This canonical form is as follows:

$$F(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_i z^{-i} + ... + a_p z^{-p}}{1 - (b_1 z^{-1} + b_2 z^{-2} + ... + b_j z^{-j} + ... + b_q z^{-q})}$$

It can be arranged that the number of non-zero values of a_i and b_j is the factor that determines the time taken to evaluate this expression.

For filtering effects, the number of these significant powers of z^{-1} will be relatively small, and the number of non-zero values of a_i and b_j will be small also. For echo chamber effects, the largest required power of z^{-1} will be quite large (typically several thousand), but the number of non-zero values for a_i and b_j will still be quite small. This lends itself quite neatly to a sparse array representation for the coefficients a_i and b_i. The chain of z^{-1} operators can be modelled using circular buffers, the speed of which is largely independent of their size.

6. REPRESENTATION

The representation of the required sound parameters would require the addition of several nodes to the VRML specification.

6.1 Sound Physics Node

This is a bindable node, which acts in a similar manner to fog.

```
SoundPhysics
{
  exposedField SFFloat delay 0.003
  eventIn      SFBool  set_bind
  eventOut     SFBool  isBound
}
```

The field `delay` is the propagation delay in seconds/meter. Setting this to 0 gives an infinite sound speed. The default value of 0.003 is equivalent to the actual speed of sound at sea level, 330 metres per second.

6.2 Sound Transform

This node is usually specified as part of a group with the sound source to which it applies.

```
SoundTransform
{
  exposedField SFVec3F direction 0 0 1
  exposedField SFVec3F location  0 0 0
  exposedField SFFloat maxBack   1.0
  exposedField SFFloat maxFront  1.0
  exposedField SFFloat minBack   1.0
  exposedField SFFloat minFront  1.0
  field MFFloat transformNear [ 0 1 ]
  field MFFloat transformFar  [ 0 1 ]
}
```

The fields `direction`, `location`, `maxBack`, `minBack`, `maxFront` and `minFront` all have the same meanings as the standard sound source definition.

The fields `transformNear` and `transformFar` contain the values of the *a* and *b* parameters for each of the infinite impulse response filter definitions.

These parameters are stored in a sparse array representation, with the first of each pair of values specifying a power of z^{-1}, and the second specifying the actual weight applied. Positive powers are used for the *a* parameters, and negative values for the *b* parameters. Only non-zero weights are specified.

The default value is an identity filter.

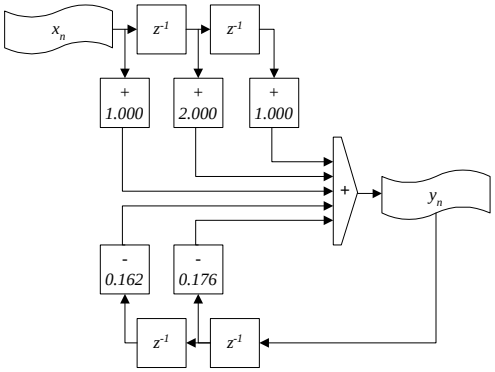

Figure 5. Two Pole Low Pass Butterworth Filter

As an example, here is the filter definition corresponding to the filter in Figure 5:

```
transformNear [  0  1     1  2    2  1
                -1 -0.176    -2 -0.162 ]
```

With an 11kHz sample rate, this is a 6db/octave low-pass Butterworth filter with a corner frequency of 3kHz.

6.3 Ambient Sound Transform

This is a bindable node, which acts in a similar manner to fog.

```
AmbientSoundTransform
{
  field        MFFloat transform [ 0 1 ]
  eventIn      SFBool  set_bind
  eventOut     SFBool  isBound
}
```

The `transform` field is the transform to apply, specified in the same format as in the sound transform node.

6.4 Occlusion Volume

The occlusion volume is specified as a transformation and a radius. The volume is assumed to be centred on the origin of the group-local co-ordinate system

```
SoundOcclusion
{
  exposedField SFFloat radius 1
  field        MFFloat transform [ 0 1 ]
}
```

The exposed field `radius` is the radius of the volume, in metres.

Again, `transform` contains the specification of the filter parameters, in the same format as the sound transform node.

7. IMPLEMENTATION

Implementation of the system involves processing all playing sounds through their respective active sound transforms. All of the methods above lend themselves easily to numerical methods, and can take advantage of both common processor architectures such as the Intel Pentium, and are also well-suited to the kinds of Digital Signal Processing chips found on advanced sound hardware.

A preliminary investigation suggests that on an Intel Pentium processor, a general-purpose filter can process one element in approximately 25 clock cycles. This can be optimised for specific cases, however.

Assuming no optimisation, and relatively simple filters of the type shown in Figure 5, we can estimate a base performance for this technology. On an Intel Pentium processor, each sample uses five filter elements at 25 clock cycles each, plus an estimated set-up overhead of another 25 clock cycles, for a total clock cycle count of 150. At a relatively modest clock speed of 100MHz, this is 1.5μs per sample, or a total sample throughput rate of 660kHz.

With optimisation, total sample throughput rates of 1200kHz and more should be available on this kind of platform, scaling almost linearly with processor speed.

8. LOAD BALANCING

A maximum throughput, as calculated above, of 660kHz, gives us 10 simultaneously active sound filters with an output sample rate of 12kHz at under 20% processor utilization.

Obviously, if we are using higher output sample rates or more active filters, this processor utilization figure will increase. We need to reserve a large percentage of the processing power for calculation of the VRML scene, so some kind of load balancing must be used to support a trade-off between sounds and other, more important tasks.

Several possibilities are open to the implementer when the sound load becomes too heavy. The most brutal one is just to stop playing low-priority sounds, as the current implementation does. We can also check the weighting functions of the active filters and remove those with lower weights altogether.

On systems with variable output sample rates, it would also be possible to reduce the output sample rate, with corresponding loss of quality. However, the parameters set for the filters assume a particular output sample rate, and recalculating them on the fly may be expensive.

More elegant options exist, however.

Within one filter, it is possible to combine filter parameters, so that those with lower weights can be discarded without too much loss in quality. This is very similar to the idea of levels-of-distance in the scene, except that the generation of the new parameters can be entirely automatic. Switching to these lower-resolution versions of the sound filter will cut the processor load by reducing the number of filter parameters considered.

In addition, in the same way as we can combine multiple spatial transformations into a single matrix, it is also possible to combine the parameters for several successive filters into a single more complex filter. This is a relatively expensive operation, but would only need to be done when the filter weights themselves change. The combined filter can then be subject to the parameter reduction operations already discussed.

In this manner, it should be possible to reduce the ongoing processor load to any desired percentage of the total system load.

9. COMPARISON WITH JAVA 3D

Since the Java 3D library [5] will probably be a popular adjunct to VRML, it is important to review the similarities and differences between the two, with a view to data exchange.

Java 3D specifies more information about sound falloff with distance, supplying multiple attenuation factors and the distances at which they apply. If required, this data could be represented on the `Sound` and `SoundTransform` nodes by replacing the `minFront` and `maxFront` fields by a series of distance/ attenuation pairs stored as a `MFVec2f`. The same applies to the `minBack` and `maxBack` fields.

Doppler shift is also accounted for by Java 3D, although the effective velocity of the sound source relative to the listener is not automatically generated, but must be supplied by the process that is moving the sound source or listener. The speed of sound is not modelled directly, but with a shift exaggeration parameter on each individual sound source.

Java 3D's ambience model allows for low pass filtering and reverberation. As has been discussed, these can both easily be modelled using infinite impulse response filters. Distance filters are specified in a similar manner to attenuation, supplying a corner frequency and distance for each element in an array.

The Java 3D model is sufficiently similar to the proposed VRML sound model to make effective data interchange possible.

10. CONCLUSION

It seems feasible to specify a much more satisfying sound model for VRML with the addition of a few well-defined nodes. All of these build on knowledge already familiar to the VRML user.

Authoring packages will be required to generate the filter parameters, but this process is well-defined and algorithms are readily available [1][3][7].

ACKNOWLEDGEMENTS

I would like to acknowledge the contribution of Dr. M. J. Usher of the University of Reading for his tutelage in information and signal theory [8].

REFERENCES

[1] Fisher, *Interactive Digital Filter Design*, University of York Web Publishing, *http://dcpu1.cs.york.ac.uk:6666/~fisher/mkfilter/* .

[2] Meldrum, J., *The Z-transform*, University of Strathclyde Web Publishing, *http://www.spd.eee.strath.ac.uk/~interact/ztransform/page1.html* .

[3] Parks, T. W. and Burrus, C. S., *Digital Filter Design*, John Wiley and Sons, Inc., 1987.

[4] Robinson, A., "Infinite Impulse Response Filters", *Speech Analysis*, Cambridge University Web Publishing, *http://svr-www.eng.cam.ac.uk/~ajr/SpeechAnalysis/node15.html* .

[5] Sun Microsystems Inc., *The Java 3D API Specification*, Sun Microsystems Web Publishing, *http://java.sun.com/products/java-media/3D/forDevelopers/3Dguide/* .

[6] Takala, T. and Hahn, J. "Sound Rendering", *ACM Computer Graphics (Proceedings of SIGGRAPH 1992)*, 26(2), pages 211-220.

[7] Thede, L., *Analog and Digital Filter Design Using C*, Prentice Hall , 1996.

[8] Usher, M. J., *Information Theory for Information Technologists*, Macmillan, 1984.

[9] VRML Consortium *et al.*, *The Virtual Reality Modeling Language*, ISO/IEC DIS 14772-1, 4 April 1997.