Constant Rate Envelope Generators

The algorithms described above produce segments where the duration exactly matches the specified time. The rate of change for the segment varies based on the amplitude levels. It is also possible to implement the envelope segment so that the rate is constant. The time of the segment will then vary depending on the start and end levels. Figure 1 shows the difference between the two types.

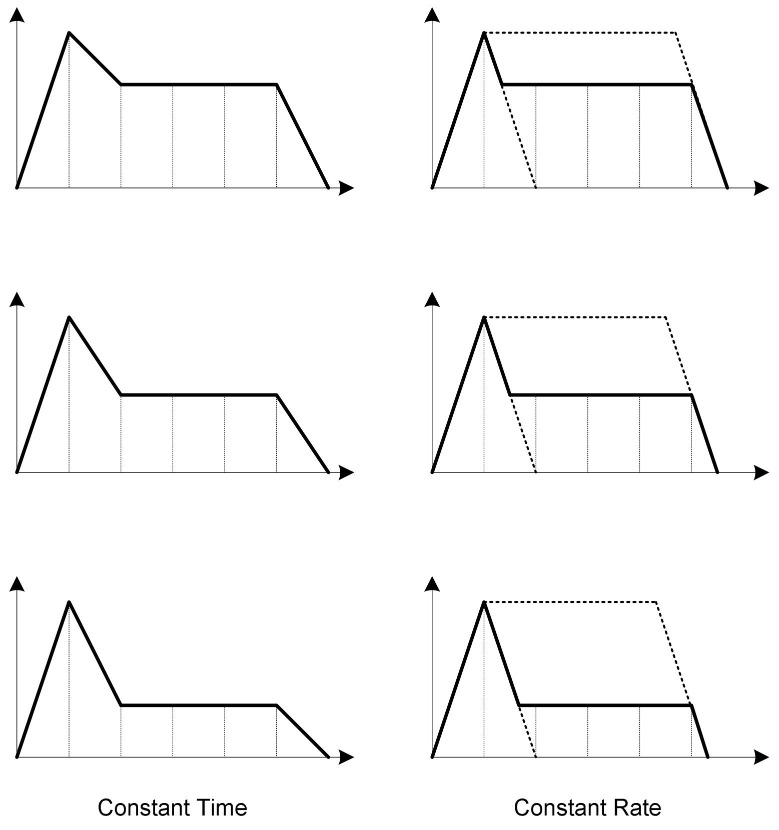

Figure 1 - Constant Time vs. Constant Rate

Note how the durations of the decay and release segments vary as the sustain level is lowered. For short durations, there is little perceptible difference between the two types, but as the duration increases the curves become significantly different.

Constant rate envelope generators are simple and efficient since increment calculations do not need to be performed on segment transitions. To implement a constant rate envelope segment, we calculate the per-sample amplitude increment based on the peak level. When the current level passes the end level for the segment, the segment is considered complete. In the following program, we assume that the sustain level is equal or less than the peak level and that the start and end levels are zero. This is typical of ADSR envelopes.

```
Init()
{
   volume = 0;
   envState = 0;
   atkIncr = peakAmp;
   if (atkTime > 0)
      atkIncr /= (atkTime * sampleRate);
   decIncr = peakAmp;
   if (decTime > 0)
      decIncr /= (decTime * sampleRate);
   relIncr = peakAmp;
   if (relTime > 0)
      relIncr /= (relTime * sampleRate);
}

Generate()
{
   switch (envState) {
   case 0:
      if ((volume += atkIncr) >= peakAmp) {
         volume = peakAmp
         envState = 1;
      }
      break;
   case 1:
      if ((volume -= decIncr) <= sustainAmp) {
         volume = sustainAmp;
         envState = 2;
      }
      break;
   case 2:
      if (noteOffSignaled)
         envState = 3;
      break;
   case 3:
      if ((volume -= decIncr) <= 0) {
         volume = 0;
         envState = 4;
      }
      break;
   }
   out = volume * sin(phase);
   if ((phase += phaseIncr) >= twoPI)
      phase -= twoPI;
   return out;
}
```

Curved envelope segments can be generated by applying a transform to the linear amplitude value. In the simple case, an exponential curve can be implemented by generating a normalized amplitude range of [0,1] and then square the amplitude value and multiply by the actual peak amplitude.

```
   out = peak * volume * volume * sin(phase);
```

Other curves can be implemented using an exponential or log function, or a lookup table. For a lookup table, we calculate the index into the table incrementally then retrieve the amplitude from the table. The table can be initialized with any appropriate curve, and each segment may contain its own curve. In the following example, we calculate a dB curve for attack, decay and release. Note that we explicitly set the first entry to zero since the exponential function can never produce a value of zero.

```
Init()
{
   envTblLen = 960; // 96 dB range
   atkTable[envTblLen];
   decTable[envTblLen];
   atkTbl[0] = 0;
   decTbl[0] = 0;
   for (index = 1; index < envTblLen; index++) {
      atkTbl[index] = pow(10, (959 - index)/-200);
      decTbl[index] = atkTbl[index];
   }
   envState = 0;
   volume = 0;
   index = 0;
   atkIncr = envTblLen;
   if (atkTime > 0)
      atkIncr /= (atkTime * sampleRate);
   decIncr = envTblLen;
   if (decTime > 0)
      decIncr /= (decTime * sampleRate);
   relIncr = envTblLen;
   if (relTime > 0)
      relIncr /= (relTime * sampleRate);
}

Generate()
{
   switch (envState) {
   case 0:
      if ((index += atkIncr) >= envTblLen) {
         index = envTblLen-1;
         envState = 1;
      }
      volume = atkTbl[index];
      break;
   case 1:
      if ((index -= decIncr) <= 0)
         index = 0;
      volume = decTbl[index];
      if (volume <= sustainAmp) {
         volume = sustainAmp;
         envState = 2;
      }
      break;
   case 2:
      if (noteOffSignaled)
         envState = 3;
      break;
   case 3:
      if ((index -= decIncr) <= 0) {
         index = 0;
         envState = 4;
      }
      volume = decTbl[index];
      break;
   }
   out = peakAmp * volume * sin(phase);
   if ((phase += phaseIncr) >= twoPI)
      phase -= twoPI;
   return out;
}
```

A constant rate envelope generator has the interesting side effect that the actual decay and release times vary with the peak and sustain levels. In other words, a louder sound has a longer duration. This effect is analogous to striking a physical object, such as a drum. A light tap on the drum produces a soft sound that decays quickly to an inaudible level. A hard strike produces a sound that is not only louder, but also has a longer decay. Thus, a constant rate generator is a good choice for physical modeling synthesis. However, this effect can create problems when several loud notes are played in succession using a melodic instrument patch. The long decay time will likely cause the notes to overlap and the resulting sound may exceed the maximum amplitude of the system. In any case, the result is not intuitive and likely unwanted. To avoid this effect, MIDI keyboard synthesizers usually include a rate scaling value based on the key velocity. The key velocity is applied to both the peak amplitude and rate values. By scaling the release rate proportionately, the resulting sound can maintain a constant release time for all volume levels. A software synthesizer instrument can include a similar feature, or we can simply adjust the decay and release rates with parameters set for each note.