Conservative Garbage Collectors for Realtime Audio Processing

Kjetil Matheussen

Norwegian Center for Technology in Music and the Arts (NOTAM)

August 20, 2009

Last 4 pages are missing from the proceedings!
Download complete paper from: http://www.icmc2009.org/papers
or pick up a photocopy from the registration desk.

What is a garbage collector?

- Automatically frees unreferenced memory.
- Without a garbage collector:

```
instrument = make_instrument();
instrument.play(20 seconds);
free(instrument);
```

- With a garbage collector:

```
instrument = make_instrument();
instrument.play(20 seconds);
```

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     1. Buses are automatically created and routed
     2. Generators are automatically created
     3. Variables are automatically stored in closure

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Example (CLM / SND-RT)

- Complete polyphonic midi synthesizer:
  1. ADSR envelope
  2. Manually implemented reverb
  3. Single sample processing
  4. 21 lines of code
  5. How:
     - 5.1 Buses are automatically created and routed
     - 5.2 Generators are automatically created
     - 5.3 Variables are automatically stored in closures

Why make a new garbage collector?

- Problems with other garbage collectors:
  1. Non-predictable performance. (May cause glitches)
  2. Provide a guaranteed worst-case, but how often does the worst-case occur?
  3. Complicated to use.
     - Could even reduce RT performance!
  4. Complicated to understand.

Why make a new garbage collector?

- Problems with other garbage collectors:
  1. Non-predictable performance. (May cause glitches)
  2. Provide a guaranteed worst-case, but how often does the worst-case occur?
  3. Complicated to use.
     - 3.1 Could even reduce DSP performance!
  4. Complicated to understand.

Why make a new garbage collector?

- Problems with other garbage collectors:
  1. Non-predictable performance. (May cause glitches)
  2. Provide a guaranteed worst-case, but how often does the worst-case occur?
  3. Complicated to use.
     3.1 Could even reduce DSP performance!
  4. Complicated to understand.

Why make a new garbage collector?

- Problems with other garbage collectors:
  1. Non-predictable performance. (May cause glitches)
  2. Provide a guaranteed worst-case, but how often does the worst-case occur?
  3. Complicated to use.
     - 3.1 Could even reduce DSP performance!
  4. Complicated to understand.

Why make a new garbage collector?

- Problems with other garbage collectors:
  1. Non-predictable performance. (May cause glitches)
  2. Provide a guaranteed worst-case, but how often does the worst-case occur?
  3. Complicated to use.
     3.1 Could even reduce DSP performance!
  4. Complicated to understand.

Why make a new garbage collector?

- Problems with other garbage collectors:
  1. Non-predictable performance. (May cause glitches)
  2. Provide a guaranteed worst-case, but how often does the worst-case occur?
  3. Complicated to use.
     - 3.1 Could even reduce DSP performance!
  4. Complicated to understand.

Why make a new garbage collector?

- Problems with other garbage collectors:
  1. Non-predictable performance. (May cause glitches)
  2. Provide a guaranteed worst-case, but how often does the worst-case occur?
  3. Complicated to use.
     - 3.1 Could even reduce DSP performance!
  4. Complicated to understand.

A solution:

- Exploit high memory bandwidth!
  1. Memory bandwidth has always increased exponentially.
     Contrary to CPU frequency, a future exponential increase in memory bandwidth is not unlikely.
  2. Memory is getting cheaper and cheaper

A solution:

- Exploit high memory bandwidth!

1. Memory bandwidth has always increased exponentially.
   - Contrary to CPU frequency, a future exponential increase in memory bandwidth is not unlikely.
2. Memory is getting cheaper and cheaper

A solution:

- Exploit high memory bandwidth!

1. Memory bandwidth has always increased exponentially.
   - Contrary to CPU frequency, a future exponential increase in memory bandwidth is not unlikely.
2. Memory is getting cheaper and cheaper

A solution:

- Exploit high memory bandwidth!

1. Memory bandwidth has always increased exponentially.
   - Contrary to CPU frequency, a future exponential increase in memory bandwidth is not unlikely.

2. Memory is getting cheaper and cheaper

A solution:

- Exploit high memory bandwidth!

1. Memory bandwidth has always increased exponentially.
   - Contrary to CPU frequency, a future exponential increase in memory bandwidth is not unlikely.

2. Memory is getting cheaper and cheaper

This works, because:

1. DSP algorithms needs little memory:
   - Can have a high memory overhead.
2. Except:
   - DSP algorithms may use much memory on audio data.
     However, audio data does not contain pointers!

This works, because:

1. DSP algorithms needs little memory:
   - Can have a high memory overhead.
2. Except:
   - DSP algorithms may use much memory on audio data. However, audio data does not contain pointers!

This works, because:

1. DSP algorithms needs little memory:
   - Can have a high memory overhead.
2. Except:
   - DSP algorithms may use much memory on audio data. However, audio data does not contain pointers!

This works, because:

1. DSP algorithms needs little memory:
   - Can have a high memory overhead.
2. Except:
   - DSP algorithms may use much memory on audio data.
     However, audio data does not contain pointers!

How does collector #1 work? (Initialization)

1. A fixed size heap is allocated.
   - Copying this heap must take less time than
     $$2 * (m - s)$$
     m = the duration of one audio buffer.
     s = the duration of processing an audio buffer.
   - Only used to store pointers, not audio data.

How does collector #1 work? (Initialization)

1. A fixed size heap is allocated.
   - Copying this heap must take less time than
     $$2 * (m - s)$$
     m = the duration of one audio buffer.
     s = the duration of processing an audio buffer.
   - Only used to store pointers, not audio data.

How does collector #1 work? (Initialization)

1. A fixed size heap is allocated.
   - Copying this heap must take less time than
     $$2 * (m - s)$$
     m = the duration of one audio buffer.
     s = the duration of processing an audio buffer.
   - Only used to store pointers, not audio data.

How does collector #1 work? (Initialization)

1. A fixed size heap is allocated.
   - Copying this heap must take less time than
     $$2 * (m - s)$$
     m = the duration of one audio buffer.
     s = the duration of processing an audio buffer.
   - Only used to store pointers, not audio data.

How does collector #1 work? (Basic technique)

- The garbage collection:
  1. Signal the garbage collector thread.
  2. Copy the heap.
  3. Run mark-and-sweep.
  4. Send back info about free memory.

How does collector #1 work? (Basic technique)

- The garbage collection:
  1. Signal the garbage collector thread.
  2. Copy the heap.
  3. Run mark-and-sweep.
  4. Send back info about free memory.

How does collector #1 work? (Basic technique)

- The garbage collection:
  1. Signal the garbage collector thread.
  2. Copy the heap.
  3. Run mark-and-sweep.
  4. Send back info about free memory.

How does collector #1 work? (Basic technique)

- The garbage collection:
  1. Signal the garbage collector thread.
  2. Copy the heap.
  3. Run mark-and-sweep.
  4. Send back info about free memory.

How does collector #1 work? (Basic technique)

- The garbage collection:
  1. Signal the garbage collector thread.
  2. Copy the heap.
  3. Run mark-and-sweep.
  4. Send back info about free memory.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Why is this a realtime collector?

- Performance is predictable:
  1. Worst-case is simulated at least once a second.
  2. Worst-case is simulated using a busy-looping timer.
  3. Garbage collectors are synchronized.
  4. Heap is not copied if the audio code spent a long time.
- Small chance of running out of memory by surprise:
  1. 3/4 of the heap is dedicated as safety buffer.
  2. New garbage collections are initiated very often.
  3. The garbage collection runs with realtime priority.

Rollendurchmesserzeitsammler

- An implementation of garbage collector #1.
- LGPL license
- Audio/Music code runs unmodified:
  - Easy to use.
  - Full DSP performance.
- Can replace BDW-GC in existing languages. For instance:
  - Bigloo
  - Mono
  - Stalin
  - D
- Used in the Snd-Rt music programming environment by:
  1. The "RT" programming language
  2. The Stalin scheme compiler

`http://users.notam02.no/~kjetism/rollendurchmesserzeitsammler/`

Rollendurchmesserzeitsammler

- An implementation of garbage collector #1.
- LGPL license
- Audio/Music code runs unmodified:
  - Easy to use.
  - Full DSP performance.
- Can replace BDW-GC in existing languages. For instance:
  - Bigloo
  - Mono
  - Stalin
  - D
- Used in the Snd-Rt music programming environment by:
  1. The "RT" programming language
  2. The Stalin scheme compiler

`http://users.notam02.no/~kjetism/rollendurchmesserzeitsammler/`

Rollendurchmesserzeitsammler

- An implementation of garbage collector #1.
- LGPL license
- Audio/Music code runs unmodified:
  - Easy to use.
  - Full DSP performance.
- Can replace BDW-GC in existing languages. For instance:
  - Bigloo
  - Mono
  - Stalin
  - D
- Used in the Snd-Rt music programming environment by:
  1. The "RT" programming language
  2. The Stalin scheme compiler

`http://users.notam02.no/~kjetism/rollendurchmesserzeitsammler/`

Rollendurchmesserzeitsammler

- An implementation of garbage collector #1.
- LGPL license
- Audio/Music code runs unmodified:
  - Easy to use.
  - Full DSP performance.
- Can replace BDW-GC in existing languages. For instance:
  - Bigloo
  - Mono
  - Stalin
  - D
- Used in the Snd-Rt music programming environment by:
  1. The "RT" programming language
  2. The Stalin scheme compiler

`http://users.notam02.no/~kjetism/rollendurchmesserzeitsammler/`

Rollendurchmesserzeitsammler

- An implementation of garbage collector #1.
- LGPL license
- Audio/Music code runs unmodified:
  - Easy to use.
  - Full DSP performance.
- Can replace BDW-GC in existing languages. For instance:
  - Bigloo
  - Mono
  - Stalin
  - D
- Used in the Snd-Rt music programming environment by:
  1. The "RT" programming language
  2. The Stalin scheme compiler

`http://users.notam02.no/~kjetism/rollendurchmesserzeitsammler/`

Benchmarks for Rollendurchmesserzeitsammler

- Computer: 1833Mhz Intel T5600, 667MHz DDR2
- Heap size: 1MB
- Max mem used: 2.8kB
- Playing: “Malagueña” by Ernesto Lecuona
- Blocking time during one audio buffer:

Mark-and-sweep			Rollendurch.		
Min	Avg.	Max	Min	Avg.	Max
0.00ms	0.03ms	0.16ms	0.02ms	0.03ms	0.10ms

Benchmarks for Rollendurchmesserzeitsammler

- Time simulating worst-case:

Mark-and-sweep			Rollendurch.		
Min	Avg.	Max	Min	Avg.	Max
n/a	n/a	n/a	0.42ms	0.43ms	0.47ms

- CPU usage to play the song.

Mark-and-sweep	Rollendurch.
3.15% (0.07%)	3.22% (0.16%)

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
    - Should not interfere with inner audio loops[1]

[1] Not required by any of my implementations[2]

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
  - Should not interfere with inner audio loops[1]

[1] Not required by any of the implementations tested

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
  - Should not interfere with inner audio loops[1]

[1] Not required by any of the implementations tested

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
  - Should not interfere with inner pointers[1]

[1] Not required by any of the implementations listed here

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap

    Should not interfere with inner audio loops.[1]

[1] Not required by any of 204 instruments included in S

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
    - Should not interfere with inner audio loops.[1]

[1] Not required by any of 294 instruments included in Sad

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
    - Should not interfere with inner audio loops.[1]

[1] Not required by any of 294 instruments included in Sa

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
    - Should not interfere with inner audio loops.[1]

[1] Not required by any of 294 instruments included in Snd

The 2nd garbage collector

- Does not copy the heap
  - Does not limit the amount of memory
- Uses at least twice as much memory
  - (writes/reads pointers to/from two memory blocks)
- The write barrier:
  - Simple and efficient
  - Only necessary when writing pointers to the heap
    - Should not interfere with inner audio loops.[1]

[1] Not required by any of 294 instruments included in Snd

Summary

- Current computers have high memory bandwidth, and lots of memory
- Copying the heap is a simple operation
- Simulating worst-case is a way to achieve predictable performance
- Many programming language implementations use a conservative collector

Questions...