Coroutine Synchronization in AVS

Gudrun Klinker

Cambridge Research Lab
Digital Equipment Corporation

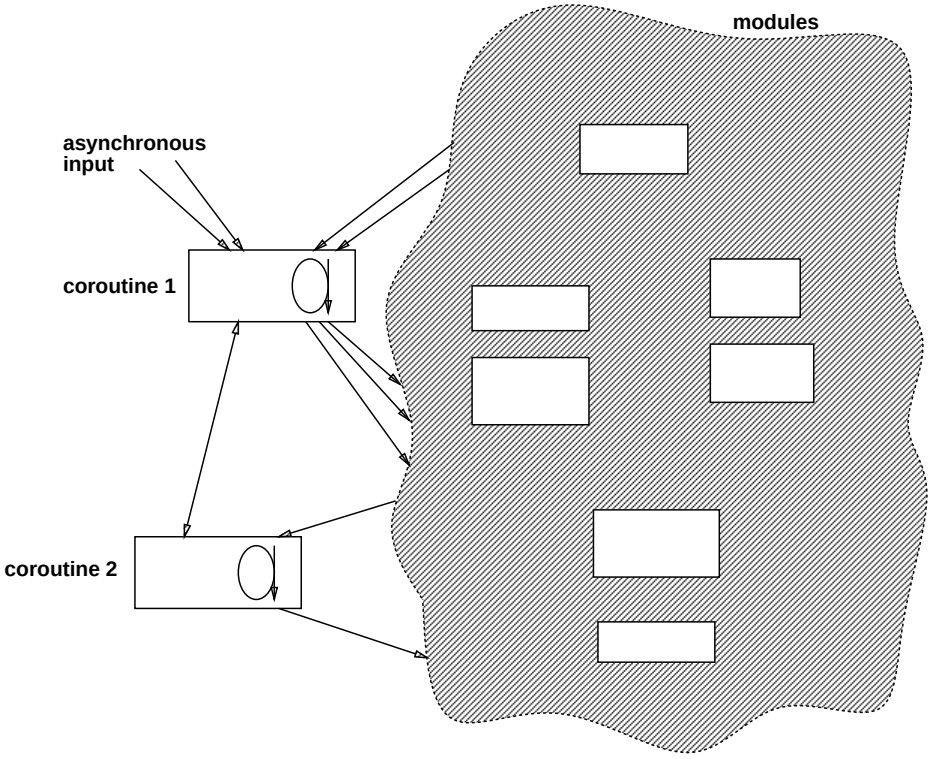
AVS '94 Developer's Track

Tue, May 3, 1994

Use of Coroutines

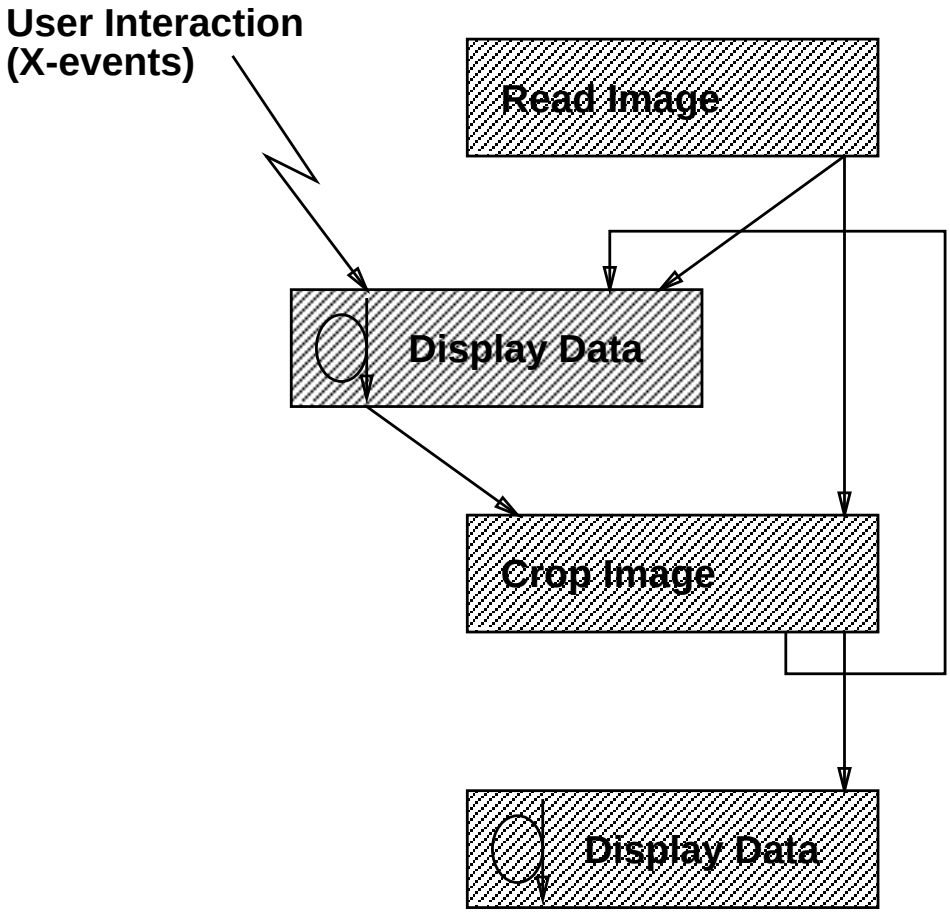
- User interaction
- Asynchronous, external input
- Simulation

Synchronization Problem

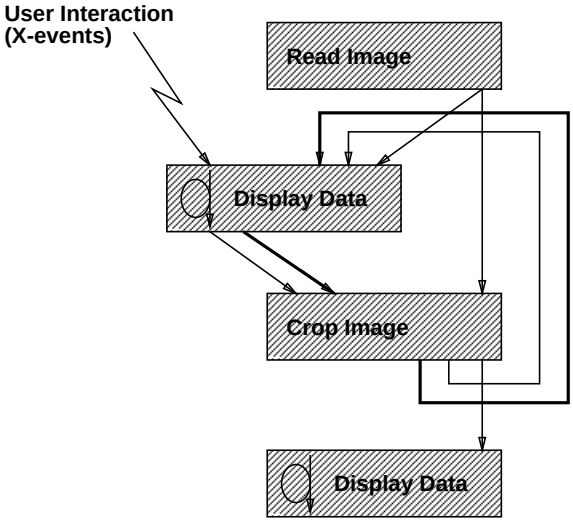


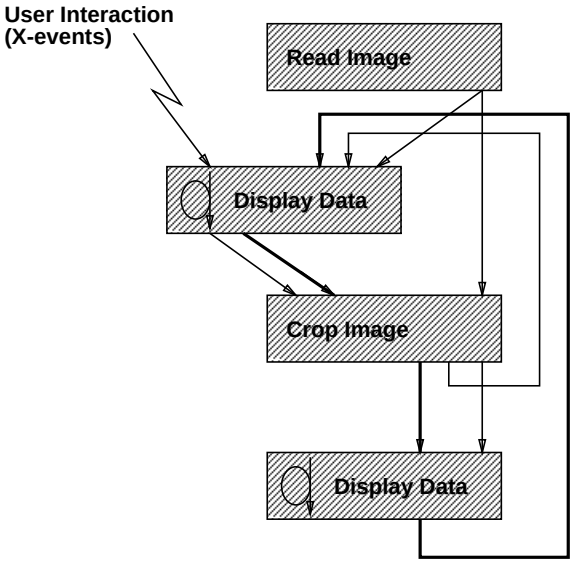
- How much processing can be performed while maintaining adequate response to asynchronously arriving or generated data?
- How can several coroutines communicate efficiently with one another?

Example: Unsynchronized Data Magnification



Synchronized Data Magnification





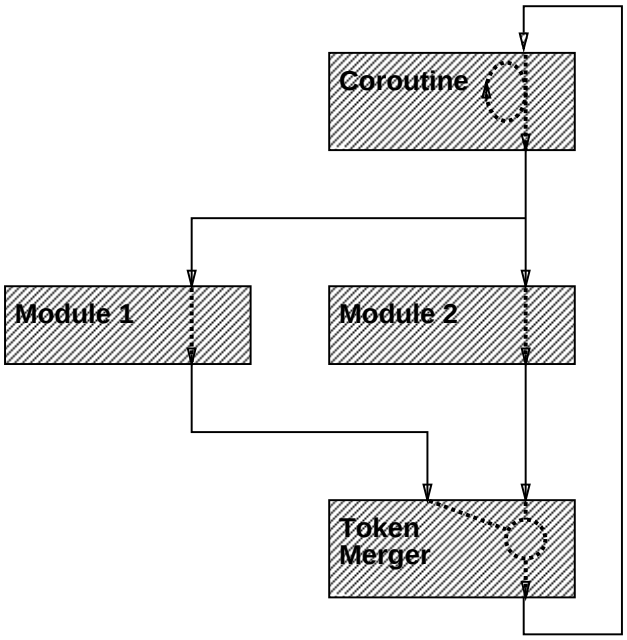
Synchronization by Token Flow

```
/* coroutine  with token passing */

main (...) {
   while (TRUE) {
      wait until something (user_io) happens
      get all new inputs and parameters

      if  (intoken up-to-date)  {
         process user_io etc.
         generate new outtoken
         send out changed data and new outtoken
      }
      else {
         save changed parameters
         send out appropriate data
      }
   }
}
```

One coroutine, several modules

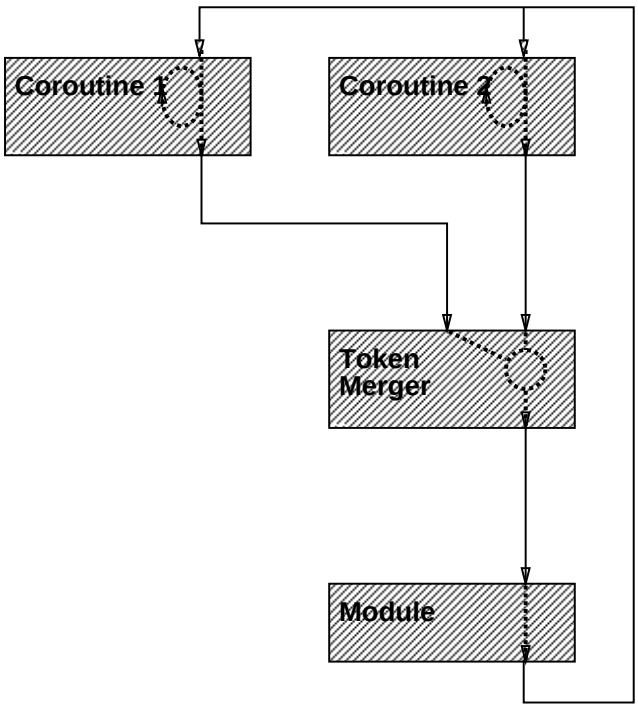


Strategies for token merging:

- Wait for first token.
- Wait for all tokens.
- Wait for a specific token.

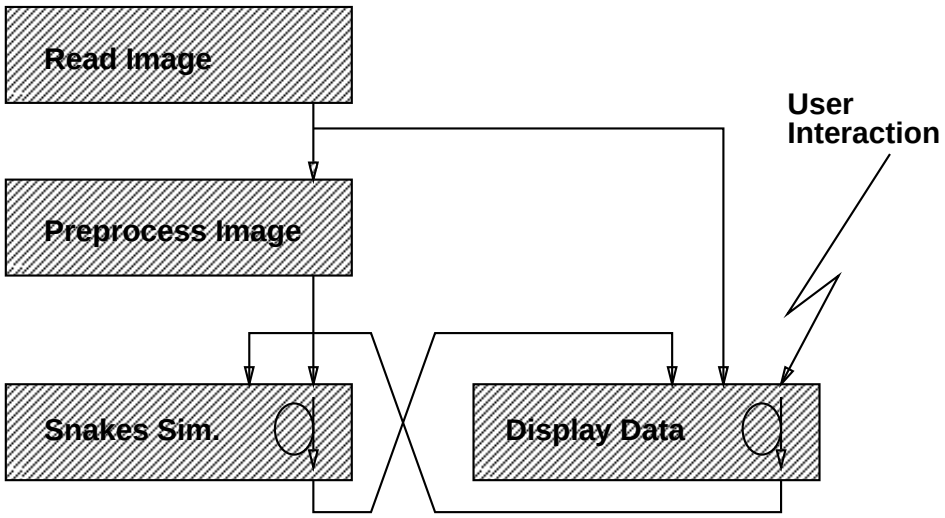
Several coroutines, one module

E.g., for cursor linking and tele-collaboration.

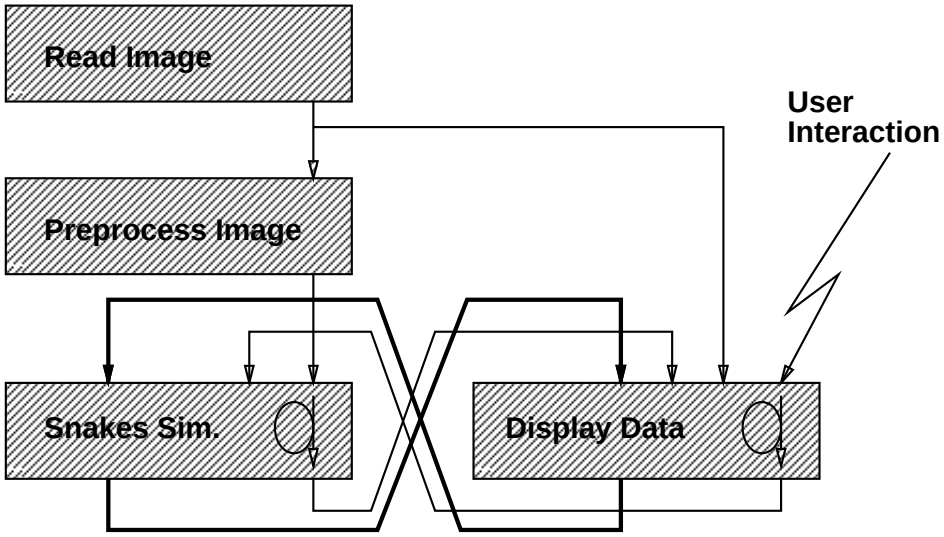


Several coroutines

E.g., for interactive, physically-based simulation: "Snakes".



Synchronized Snakes



Conclusions

- Token flow allows developers to define synchronization constraints a runtime.
- Token forwarding is easy.
- Token generation and maintainance is hard:
  - Input buffering
  - Infinite loops
  - Deadlocks from unexpected flow interrutions
  - Premature token updates due to shared memory.
- We need higher-level tools to manage control flow (token flow).
- We need several, semantically different, views of AVS data flow networks.