Parallel Multi-View Rendering on Multi-Core Processor Systems

Jan Ohlenburg, Wolfgang Broll
Collaborative Virtual and Augmented Environments Department
Fraunhofer FIT, Germany

Rendering Parallel Views

- dual-core and multi-core processing systems have now become a standard
- multi-threading rendering takes advantage of this trend
- bad multi-threading approach will result in performance loss

Our approach

- support for a high number of parallel render threads
- limited only by the capabilities of the computer
- each render thread has own
  - camera
  - render state
  - refresh rate
- resources are shared among the render threads
  - textures
  - display lists
  - vertex buffer objects

Performance issues

- avoid context switches
- synchronize update and render threads
  - no rendering may be active while updating

Straight forward approach

- update scene
- start render processes
- wait for next frame
- same refresh rate for all render threads

Our solution

- Render thread notifies update thread and performs the render process after the update (see Listing 1)
- The update thread (see Listing 2) waits for an update notification...
- ... locks each mutex of all render threads, ie waits until the last has finished rendering and prevents them from starting to render
- ... performs the update and unlocks the mutexes and notifies all render threads
- In case render threads notify the update thread during the update process, they can share the same update.

Rendering Stereo Pairs

- Quad-buffered stereo is not parallelizable
  - left and right view have to be rendered in one thread
- Stereo split view can be used with this approach
  - But: left and right view may display different frames
  - vertical sync may happen between buffer swaps.

```
while (! isTerminated ())
    mutex . lock ();
    manager -> notify ();
    managerCondition . wait ( lock );
    render ();
    condition . timed_wait (lock , nextFrameTime );
    mutex.unlock();
```

Listing 1: Render Thread Loop

```
while (! isTerminated ())
    mutex . lock ();
    condition . wait ( lock );
    for each render process
        renderer -> mutex . lock ();
    update ();
    for each render process
        renderer -> mutex . unlock ();
        renderer -> notify ();
    mutex.unlock();
```

Listing 2: UpdateThread Loop

Example 1:

- three views
  - two with maximum refresh rate
  - one with 10 fps
- update takes 10 msecs

⟶ optimal update rate of 90 ups (see Example 1)

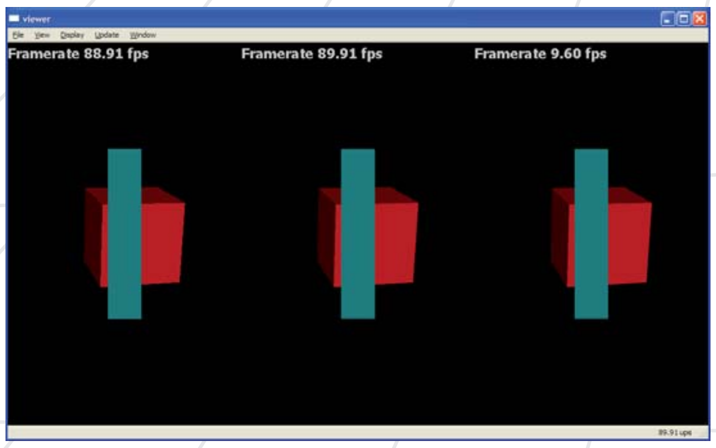
Example 2:

- same as above
- update takes 1 msec

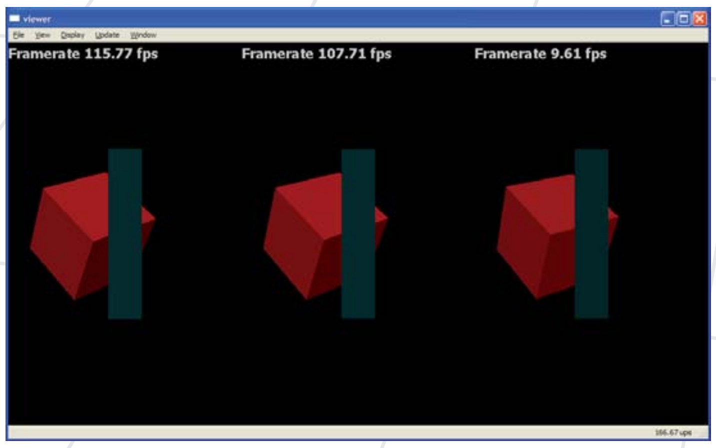
⟶ update rate of 160 ups, while refresh rate of views is 120 fps (see Example 2)

Example 3:

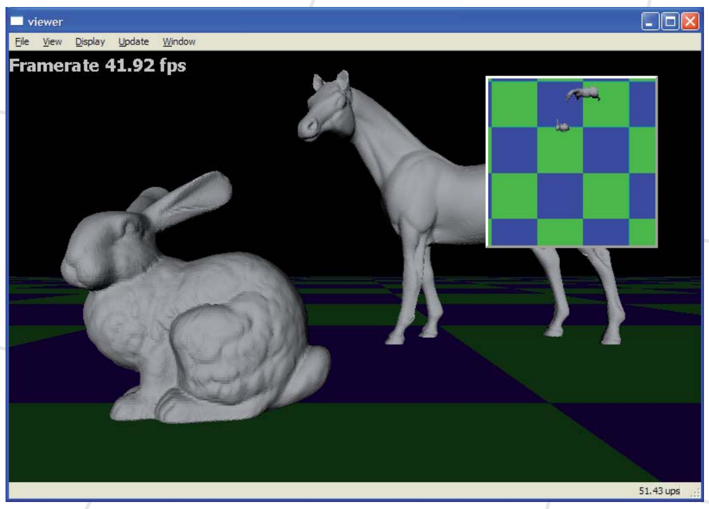
- using two views
- one as a top view of the other


Framerate 88.91 fps
Framerate 89.91 fps
Framerate 9.60 fps

Example 1


Framerate 115.77 fps
Framerate 107.71 fps
Framerate 9.61 fps

Example 2


Framerate 41.92 fps

Example 3

Parts of the work on Parallel Rendering were performed within the IPerG project.
IPerG is partially funded by the European Commission in FP6
(FP6-2002-IST-3-004457)

Fraunhofer Institut Angewandte Informationstechnik

Contact: jan.ohlenburg@fit.fraunhofer.de