Checkpoint-Resume and Incremental Processing

Recent extensions to job scripting, dispatching, and the Dashboard add interesting new capabilities related to incremental computation. Tractor also supports a general "retry from last checkpoint" scheme. Both features integrate with the new RenderMan 19 RIS checkpoint and incremental rendering features.

Retry from last partial result

As an example, the Pixar RenderMan renderer, prman, has supported a built-in retry scheme for several years, when using compliant output formats.

```
A basic render command:
   prman frame123.rib
```

If that process is killed for some reason after an hour of rendering, but it produced a partial image file along the way, then the rendering can often be restarted from that point.

```
Restart from the last image checkpoint:
   prman -recover 1 frame123.rib
```

The initial render is equivalent to using -recover 0 since the zero specifies that the rendering should start from the beginning, and ignore (overwrite) any pre-existing output files.

A simple Tractor job script that invokes that basic render command can be written in terms of the Tractor "%r" run-time substition parameter that Tractor will fill in when the command is launched.

```
Job -title {simple recover example} -subtasks {
     Task {Frame 123} -cmds {
         RemoteCmd {prman -recover %r frame123.rib} -service PixarRender
     }
}
```

Tractor substitutes the digit zero for %r in the argument list on the first launch of the command or when a user requests a full restart of the task; and it substitutes the digit one when a user requests a resume from the last checkpoint.

So the Tractor support of this feature assumes that the restart behavior of many applications, or wrapper scripts around them, can be controlled with some command-line parameter that accepts a simple '0' or '1'.|

Incremental Checkpoint-Resume Iterations

Introducing the new Tractor job scripting controls:

```
-resumeblock
-resumewhile
-resumepin
```

The goal is to mark a block of tasks (an entire dependency subtree) as group that will be re-run when some elements have reached a checkpoint but are still incomplete.

The -resumeblock option goes on the **Task** at the root of the group. The -resumewhile clause goes on a **RemoteCmd** that obeys the checkpoint-resume semantics. The -resumepin 1 option is an additional qualifier on commands with -resumewhile, indicating that restarts must occur on the same blade as the initial pass of that command.

NOTE: Within the subtree indicated by the -resumeblock Task, not all tasks will be restarted on each pass. Only those commands whose -resumewhile clause evaluates to "true" (non-zero) will be retried on the next pass. Some of those commands may completely finish several passes before others in the block. Once a command's resumewhile clause evaluates to "false" (zero), it will not be restarted by this looping mechanism. Also note that *any* task that depends on a resumed task, up through the root of the root of the resume block, will also be retried. A retry due to a -resumewhile behaves just like a manual task retry in that regard: dependent (successor) task are retried as well.

The substitution patterns %r, %R, and %q are filled in by tractor-blade before launching each command: %r is 0 when a task is beginning a fresh start and an integer greater than zero when the user or system is requesting a recover from checkpoint; %R is the "loop count" for commands that are being restarted due to the -resumewhile construct; and %q has the value 1.0 when a task is being executed due to final quality runs from the subtasks it depends upon, it will be less than 1.0 if some subtask only reached a checkpoint during the current resumewhile loop. Many launched commands will not support these distinctions, it is up to the job author to use these optional parameters appropriately for each command.

```
Job -title "checkpoint-resume iteration example" -service PixarRender -subtasks {
    Task {the wait} -subtasks {
        Task {the loop} -resumeblock 1 -subtasks {
            Task {render f1} -cmds {
                RemoteCmd {
                    prman -Progress -recover %r -checkpoint 0,15s /net/rib/rings.rib
                } -resumewhile {testcheckpoint exr /tmp/t.exr} -resumepin 1
            }
            Task {script test1} -cmds {
                RemoteCmd {echo otherApp --pass %R} -resumewhile {exitcode 123}
            }
        } -cmds {
            RemoteCmd "echo looping"
        }
    } -cmds {
        RemoteCmd "echo loop done"
    }
}
```