Rendering: A case study of workflow management + cloud computing

Michael J Pan
Nephosity

20 April, 2010

Motivation

- Rendering is compute intensive.
- nVidia has announced RealityServer, a cloud enabled rendering platform using MentalRay's renderer, iray.
- Can we do it affordably?
- Can we do the same with an open source software?

Common speedup strategies

Common strategies to speeding up rendering, each with its pros and cons

- Specialized hardware, i.e. GPUs
- Multi-threading
- Message Passing Interface (MPI)

GPU rendering

- hardware optimized (faster than software)
- high end GPUs are extremely expensive
- limited number of GPUs per machine
- highly specialized code

Multi-threaded rendering

- ability to make use of multiple cores
- each thread renders a tile of the final image
- limited to a single machine

MPI-based rendering

- use a cluster of machines, e.g. Beowolf
- each client renders a tile of the final image
- MPI can be unstable, and does not handle changes to cluster membership

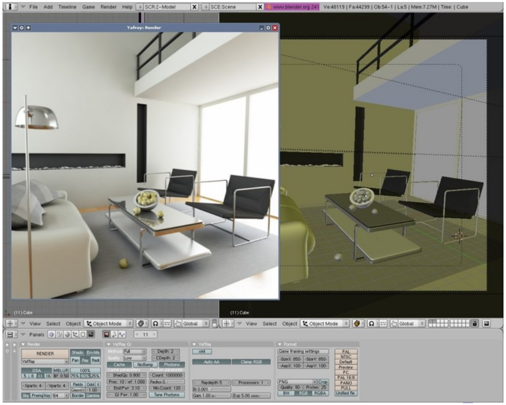
Design

- process based, tiled rendering
- process based composition of individual tiles into final image
- dynamically generated workflow, i.e. number of tasks generated are driven by various parameters, including tile size, number of tiles, resolution
- dynamically allocated cluster for executing pomset tasks

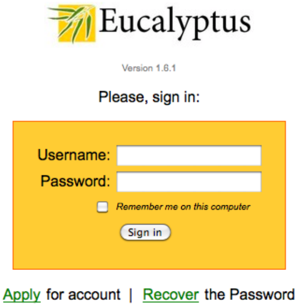
Software

- Yafaray, an open source renderer, used in Blender, an open source 3D content creation manager
- Eucalyptus, an open source cloud controller
- pomsets, an open source, cloud enabled workflow management system

Blender & Yafaray


File Add Timeline Game Render Help SCR:2-Model SCE:Scene www.blender.org 241 Ve:46119 | Fa:44299 | Ob:54-1 | La:5 | Mem:7.27M | Time: | Cube
Yafray: Render
(11) Cube
View Select Object Object Mode Global
Panels
RENDER
YafRay

Eucalyptus


Eucalyptus

Version 1.6.1

Please, sign in:

Username:
Password:
Remember me on this computer
Sign in

Apply for account | Recover the Password

Implementation steps

- Implementation of process based tile rendering and composition
- Integration of the workflow management system
- Deployment in the cloud

Process based tile rendering and composition

Implementation steps

- Yafaray uses threads to render each tile. Implement a command to do the same work as a thread
- A single thread composites the tiles into a final image. Implement a command to composite the tiles.
- The dimensions (size and position) of the tiles are determined internally. Open up the API to enable external command to specify tile dimensions
- Implement a script that will manage the tasks
  - decide the tiling info, i.e. number of tiles, size of each tile, etc.
  - generate and execute a task to render each tile
  - wait for all tiles to complete, then composite the tiles together

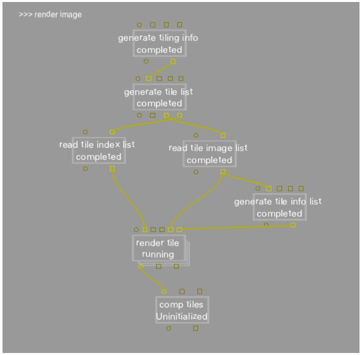
Integration of the workflow management system

Missing functionalities of the script:

- parallel execution
- process pooling: ability to parallelize tile rendering without overwhelming system if all tasks are executed simultaneously
- distributed computing: ability to execute on multiple compute nodes

All are provided by pomsets, which also handles the parameter sweep and dependency condition handling functionalities implicitly coded into the script. Modify script so that instead of executing a command, it creates a node in the pomset that when executed, will execute the command.

Resulting pomset



Deployment onto the cloud

- get an account on a cloud controller
- provide pomsets with the cloud controller credentials
- execute

Specs

Scenes

- Scene 1:
  - 2046 primitives
  - 375x375 pixels
- Scene 2:
  - 68384 primitives
  - 1280x720 pixels

Performance will vary according to hardware used

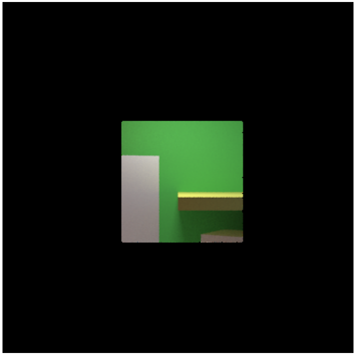
- numbers are based upon 1.66GHz core, 2GB ram
- approximately 3.5x compute time with only 512mb ram

Some definitions

- Wall clock time: Time from the start of the first task to the end of the final task
- Total compute time: Sum of the time spent on computation of all compute nodes

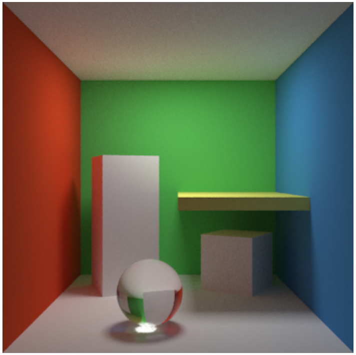
Scene 1

A 128x128 tile



Scene 1

Final image composited from all tiles

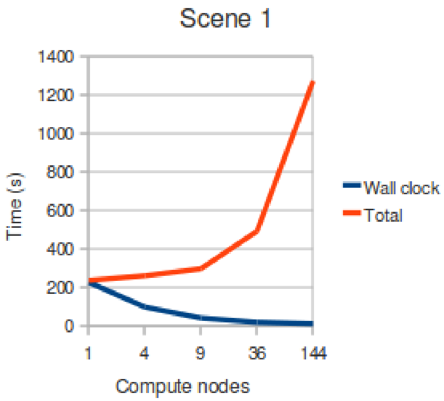


Render time statistics for Scene 1, in seconds

Tile size/count	Min/tile	Max	Avg	StdDev	Total
Original					231.787
512/1	226.380	226.380	226.380	0.000	235.723
256/4	33.244	97.701	62.322	24.029	259.760
128/9	25.678	40.655	31.744	4.304	296.287
64/36	10.883	18.540	13.369	1.861	491.950
32/144	7.879	10.879	8.761	0.543	1273.961

Wall clock is 5-10s more than the max tile time.

Scene 1 render times

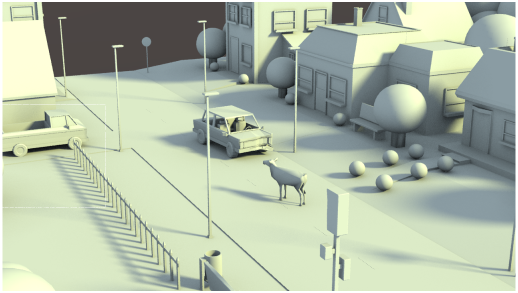


Scene 2

A 256x256 tile

Scene 2

Final image composited from all tiles

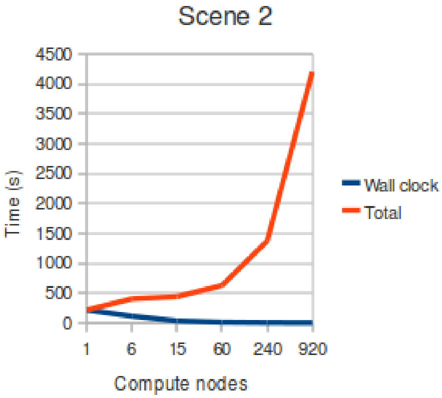


Render time statistics for Scene 2, in seconds

Tile size/count	Min	Max	Avg	StdDev	Total
Original					219.143
512/6	23.663	115.793	63.666	29.849	405.810
256/15	17.283	35.541	27.944	5.232	442.589
128/60	4.769	12.688	10.064	1.793	628.559
64/240	4.239	6.410	5.610	0.536	1373.658
32/920	4.124	4.772	4.506	0.122	4202.137

Wall clock is 10-15s more than the max tile time.

Scene 2 render times



Amdahl's law

If P is the proportion of a program that can be made parallel (i.e. benefit from parallelization), and (1 - P) is the proportion that cannot be parallelized (remains serial), then the maximum speedup that can be achieved by using N processors is

$$\frac{1}{(1-P)+\frac{P}{N}}$$

Takeaways

- Massive parallelization will hit Amdahl's law
- Brute force parallelization is unproductive
- Understanding the computational workflow enables us to separate the parallelizable non-parallelizable components, which in turn can enable us to make better decisions about parallelization strategy

Future work

- get more results on more complex scenes
- integration of pomset_render into the Blender rendering workflow
- apply same approach to other aspects of Blender's content creation suite– effects, animation, etc.

Questions?

- Michael J Pan - mjpan@{pomsets.org|nephosity.com}
- pomsets webpage - http://pomsets.org