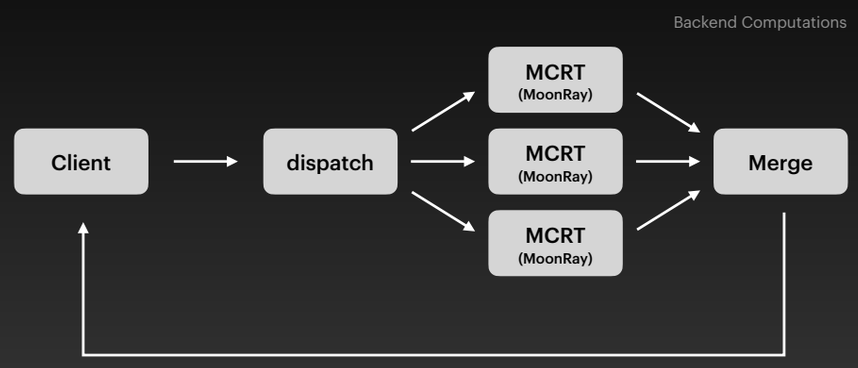
I’m Toshi Kato Principal Engineer of Moonray Team @ Dreamworks animation
and mainly work on distributed moonray.

I want to explain about multi-machine rendering for moonray today.
Moonray’s multi-machine rendering is mainly focusing on interactive lighting sessions.

Typical Configuration



MOONRAY

This is a typical configuration of a multi-machine moonray session.
Client sends scene to dispatch
Dispatch copies of scene to the all MCRT.
We send the entire scene to each backend MCRT computation.
MCRT computation runs exactly the same moonray code of moonray and moonray_gui internally.
Merge node merges the image and creates the whole final image and send back to client

Characteristic

- Multiplex Pixel Distribution
- ProgressiveFrame message

MOONRAY

I want to briefly explain 2 fundamental characteristics of multi-machine moonray.
They are
	Multiplex pixel distribution
and
	ProgressiveFrame Message

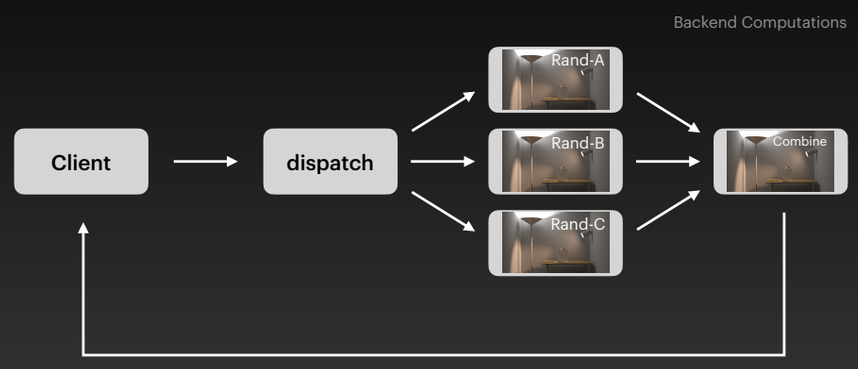
Characteristic

- Multiplex Pixel Distribution
- ProgressiveFrame message

MOONRAY

Multiplex Pixel Distribution

Not tile-based task dispatch



MOONRAY

One of the widely used solutions for multi-machine rendering is tile-based workload dispatch.
Each node tries to get the next tile to each other. this works well.
Actually,
We don’t use tile-based task dispatch. Instead, each mcrt computation tries to render the whole image.
But the important point is, each MCRT uses different random sequences.
Then, image is combined at merge node and sent back to the client. Very simple idea but it works very well.

Multiplex Pixel Distribution

- Almost no communication between mcrt-node : good scalability
-

MOONRAY

It’s almost no communication between mcrt-node during rendering.
This is very important for scalability in terms of the total number of machines

Multiplex Pixel Distribution

- Almost no communication between mcrt-node : good scalability
- Good load-balancing
-

MOONRAY

It’s good load balancing even if we don’t have any special work dispatch or work stealing logic.
Each node simply can run at full speed independently
Logic is pretty simple and robust.
High CPU utilization is always important and this solution has advantage for this.

Multiplex Pixel Distribution

- Almost no communication between mcrt-node : good scalability
- Good load-balancing
- Can work well under mixed different spec machine conditions
-

MOONRAY

This solution can work very well under mixed different spec machines.
we don’t need to use exactly the same spec machines for the backend
Each machine contributes to the final rendering according to its own performance.
And, This gives us various choices of configurations of backend engines.

Multiplex Pixel Distribution

- Almost no communication between mcrt-node : good scalability
- Good load-balancing
- Can work well under mixed different spec machine conditions
- No required sync start timing of MCRT phase
-

MOONRAY

We don’t need to sync the start timing of all MCRT rendering.
This is important because boot and start render timing is depending on each host’s environment.
Some hosts can quickly boot and start rendering quickly.
And another host might slow boot and start rendering at a very delayed time. But this is OK.
Finally, each host can contribute to the final rendering by their own timing.

Multiplex Pixel Distribution

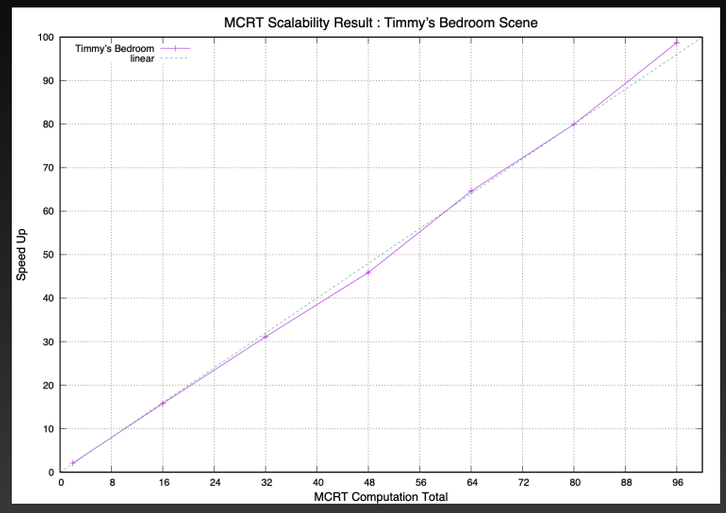
- Almost no communication between mcrt-node : good scalability
- Good load-balancing
- Can work well under mixed different spec machine conditions
- No required sync start timing of MCRT phase
- Robustness from unexpected accident

MOONRAY

You can relatively easily build a potentially robust environment for rendering session.
Because each MCRT node does not need to trackdown other hosts progress.
In terms of rendering progress, the system itself does not care which host is working and which host is dead.
If one host hangs or crashes, this is also fine. rest of the hosts try to finish the rendering.

Multiplex Pixel Distribution

Scalability



Client :
Glendale DWA

Backend :
Azure : westus
128GByte 64HTcores

MOONRAY

We had a chance to do a scalability deep test on Azure cloud.
In this case, The client was running on Glendale DreamWorks and all backend engine is on WestUS Azure cloud.
Actually, the biggest bottleneck was bandwidth between WestUS and Glendale. But multi-machine moonray adjusts bandwidth dynamically even if we uses 96 machines.
96 hosts uses more than 6000 cores. and result is pretty much the linear.

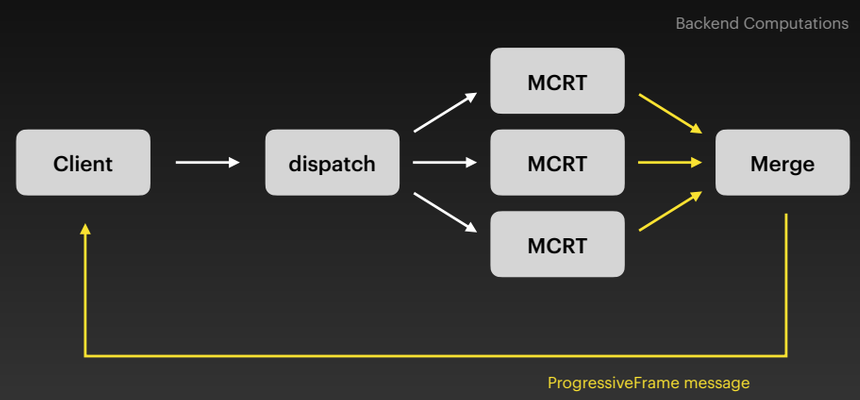
Characteristic

- Multiplex Pixel Distribution
- ProgressiveFrame message

MOONRAY

The next topic is progressiveFrame message

ProgressiveFrame message



MOONRAY

We are using ProgressiveFrame message for transferring image data between computations and clients.
This message is using special image data encoding format.

ProgressiveFrame message

- Heavily delta coded (PackTile codec)
-

MOONRAY

The fundamental idea is delta coding of progressively updated rendering images.
We only send the updated pixels to the destination.
We call this is PackTile codec and it’s lossless coding.

ProgressiveFrame message

- Heavily delta coded (PackTile codec)
- Selectable data precision (UC8, H16, F32)
-

MOONRAY

There are 3 different pixel precision 8bit, 16bit, and 32bit.
This is useful to reduce message size.
For example, very early stage of rendering, we don’t need color accuracy and chose low precision to make data size small and try to achieve low latency. But change to the original full 32bit precision for the later stage of rendering.

ProgressiveFrame message

- Heavily delta coded (PackTile codec)
- Selectable data precision (UC8, H16, F32)
- Multi image blocks (AOVs)
-

MOONRAY

We can send multiple layers of images for AOVs
Typically, the production uses lots of AOVs and ProgressiveFrame can handle them.
We have various different support logic to reduce message size regarding AOVs like selectively sending AOVs and partially sending Images.

ProgressiveFrame message

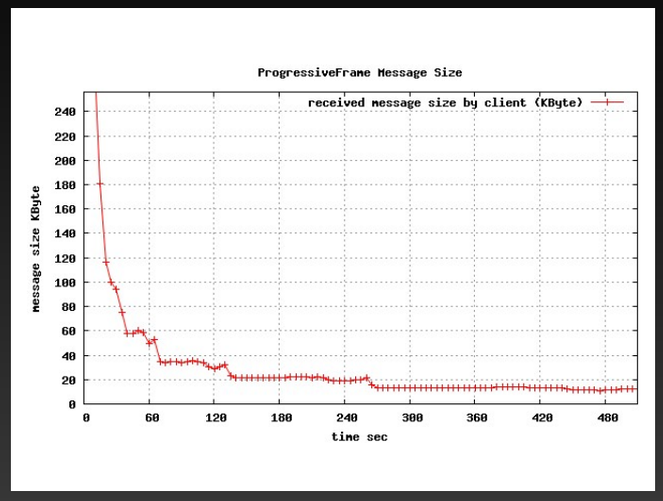
- Heavily delta coded (PackTile codec)
- Selectable data precision (UC8, H16, F32)
- Multi image blocks (AOV)
- Can send statistical info (InfoCodec)

MOONRAY

We can send non-image data as well. We call this as InfoCodec.
For example, the backend engines are sending lots of telemetry data to the client by progressFrame message.
This includes CPU usage, Memory usage, network bandwidth info, snapshot timing, and so on.
They are used for input parameters for dynamically control logic and profiling / debugging purposes.

ProgressiveFrame message

Message Size Transition



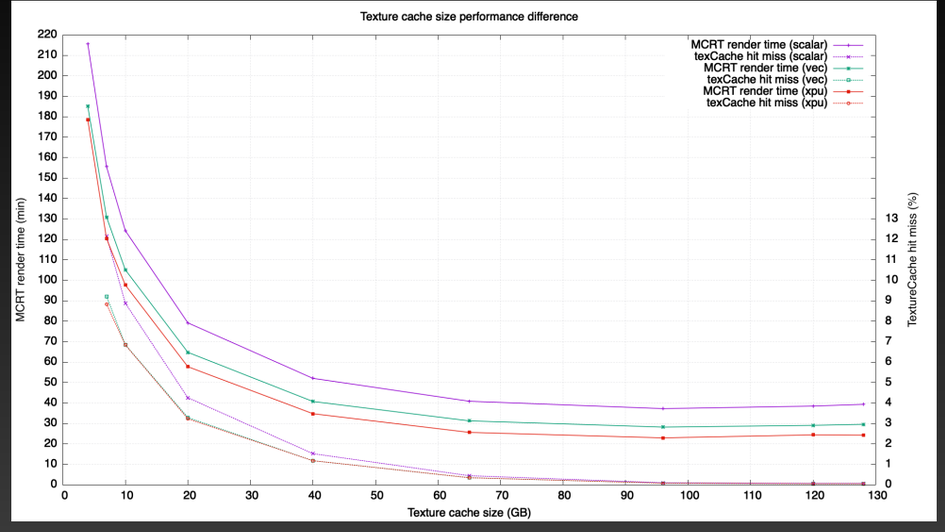
This is the chart of how message size is changing during rendering.
In this case, message size is quickly reduced when rendering is in progress.
Because the image is drastically updated in the beginning and data size is big.
But render focuses local region at the later stage, as a result, delta info gets smaller.

I want to show some demo using ALab scene.

ALab 2.0.1 is pretty good for testing texture-heavy scenes.
Pretty complex scenes and the original rdlb size is 2.6GB.
Lots of 4K textures and it required around 100G texture cache for good interaction.
We can run only using 4G of texture cache but it is slow. This is a trade-off of space and speed.

Texture Cache Size

ALab 2.0.1 Render Time Transition



MOONRAY

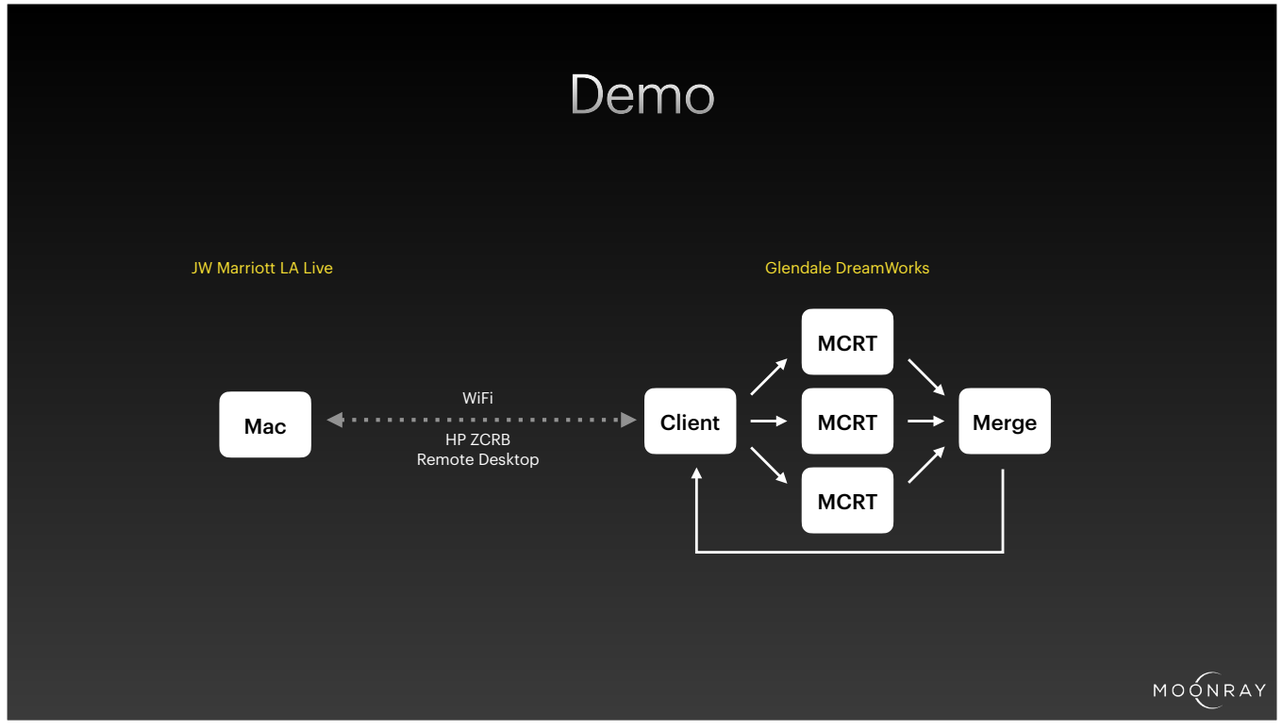
This graph shows the performance change of ALab 2.0.1 regarding texture cache size.
Actually, We can render ALab with only 4G texture cache but it is slow. and
We can easily get around 6x speed up if we can assign around 100G texture cache.

This graph also shows performance differences between Scalar, Vector, and XPU modes of moonray.
Vector is up to 1.3 x faster than scalar and XPU is up to 1.6x faster than scalar.
Moonray’s vector/XPU architecture is pretty strong for the texture-heavy scene because bundled architecture maximizes memory access coherency.

Texture Cache Size

ALab 2.0.1 XPU breakdown

This is a MCRT task breakdown of XPU mode by render_profile_viewer.
The purple block is texturing and it is dominant when using the small texture cache and drastically deduced if we use big texture cache.



This is an environment for today’s demo.
multi-machine configuration for the demo is running on Glendale DreakWorks.
And I’m using several different machines for the backend.
The client is also running on Glendale.
I’m going to connect from here to Glendale using the Remote Desktop program via WiFi connection

Questions?

Questions?