Parallel rendering

Juan Hernando Vieites
jhernando@fi.upm.es

ECAR 2012

1. Introduction
2. Techniques for large data sets
3. Parallel rendering
   - Compositing techniques
4. Libraries and frameworks
   - Equalizer

Why do parallel rendering/visualization

Motivations

Whenever we want to:

- Increase the rendering capability:
  - Being able to deal with larger data sets in a reasonable amount of time.
  - Being able to combine more techniques in a single visualization.
- Increase interactivity: The same data sizes but with shorter response time.

we will find bottlenecks in some stage of the visualization pipeline.

Why do parallel rendering/visualization

Low level bottlenecks

These refer to both the CPU and the GPU:

- Compute power
- Memory
- Bandwidth

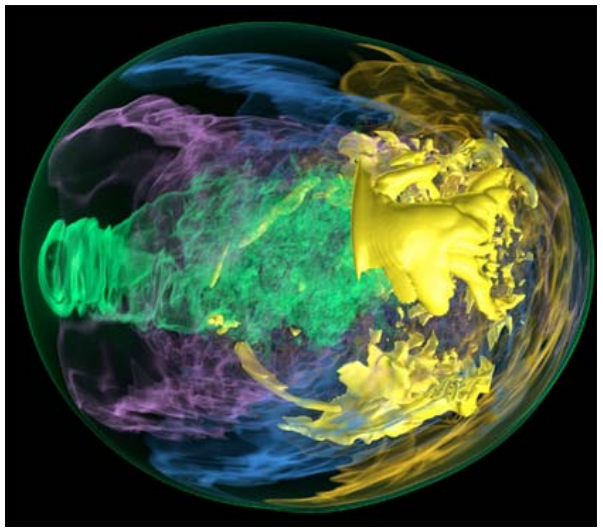
Graphics pipeline bottlenecks

- Update limited (CPU pre-processing)
- Geometry processing (too much vertex processing)
- Fill-rate limited (too much fragment processing)

HPC vs. real-time applications

Parallel rendering in HPC

- To tackle capability problems (massive data).
- Interactivity is interesting in exploratory settings, but it is not deemed as the main goal.
- Latency is not considered an issue.
- Weak scaling is more interesting.
- Software rendering in a supercomputer.



Supernova collapse simulation
(863^3 Scalar field)
Simulation by John Blondin at the (NCSU) and Anthony Mezzacappa (ORNL)
Image by Tom Peterka (ANL) et al

HPC vs. real-time applications

Parallel rendering in real-time/interactive applications

- The main goal is to reduce processing time to increase interactivity.
- The data sizes may not be larger, but we want to render faster.
- Latency is an issue in certain settings, e.g. VR applications
- Strong scaling is more interesting.
- Dedicated cluster with GPUs for rendering.
- Also relates to multi-display and VR facilities.

Large data sets

What is a large data set?

A large data set is one that forces us to apply special techniques to process it due to compute/memory/bandwidth constraints.

Techniques to deal with large data

- Out-of-core
- Sub-setting
- Multi-resolution/compression
- In-situ
- Parallelization

Techniques for large data sets

Out-of-core

- Load data on demand.
- If lowers the memory requirements and enables smaller machines to process the data.
- Still all data is processed (IO will still be a bottleneck)
- We: separable data, mappable from outputs to inputs and result invariant algorithms.

Sub-setting

- Reduce the data set to be visualize.
- The selection can be:
  - Trivial (e.g. slicing)
  - Query driven (manual or semi-automatic)
- Query driven sub-setting requires indexing data structures.

Techniques for large data sets

Multi-resolution/Compression

- HD screens have 2M pixels, but some data sets are much bigger than that in terms of cells.
- In some cases, compression and subsampled representations can be used with the same visual results.
- However, the full data must be preprocessed at least once.

In-situ

- Do not store the bulk of the data. Process the data as it is being generated.
- A complex problem on its own (resiliency, resource allocation, load balancing if parallel, ...)

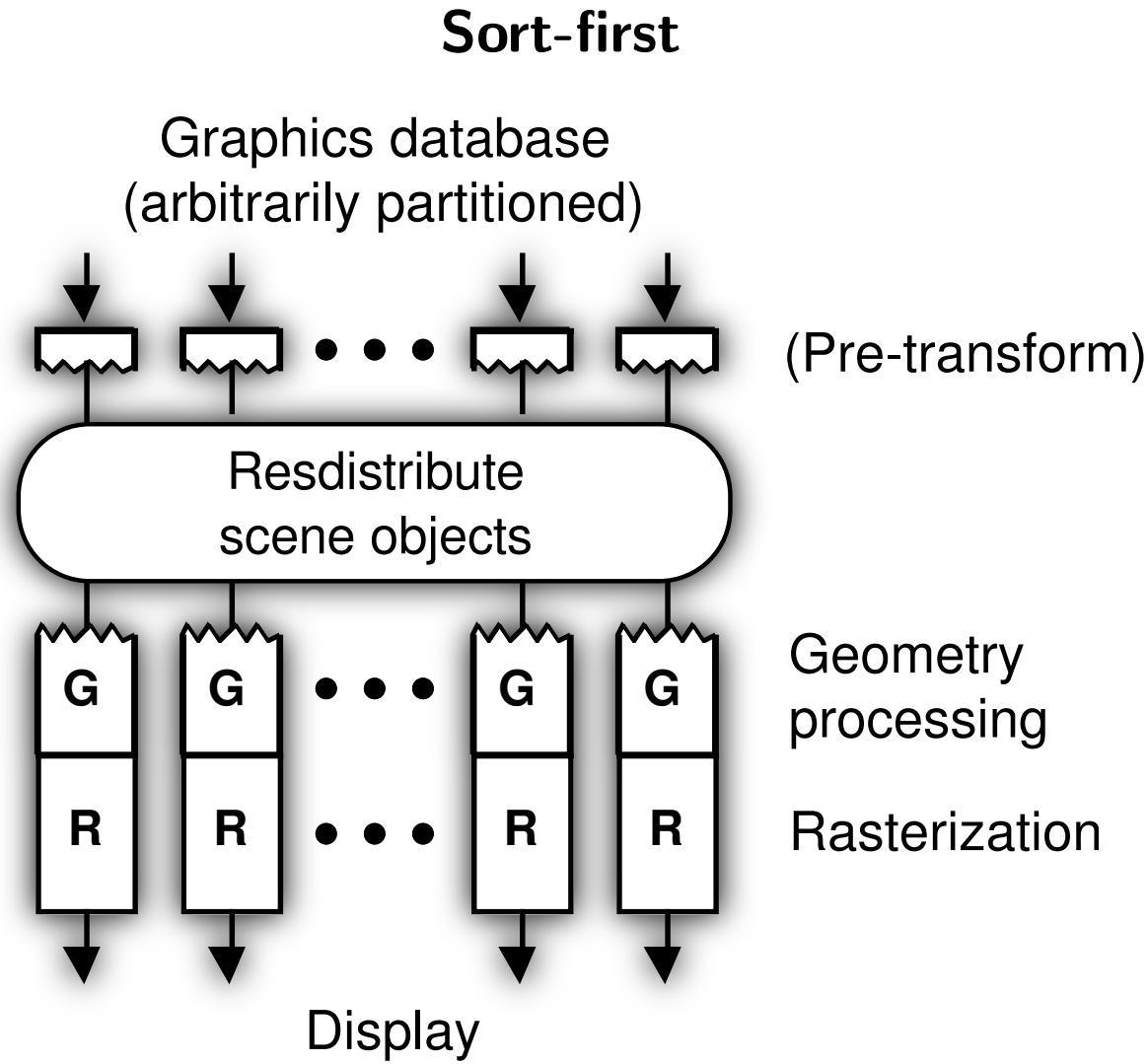
Types of parallel rendering

Classical classification

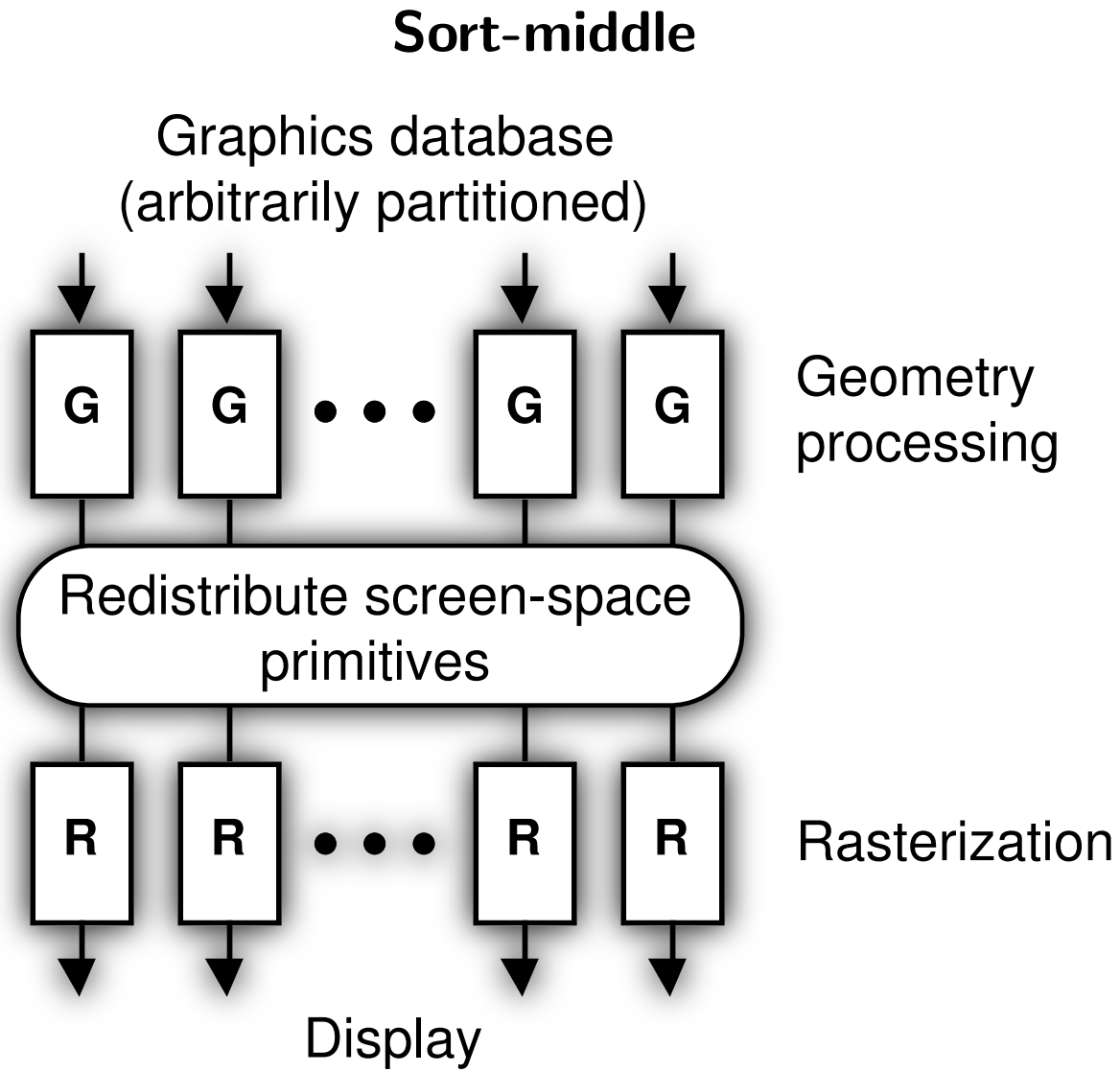
Molnar et al presented the first classification of parallel rendering systems. Originally focused on SMP visualization supercomputers, the terms still remain.

- Sort-first
- Sort-middle
- Sort-last

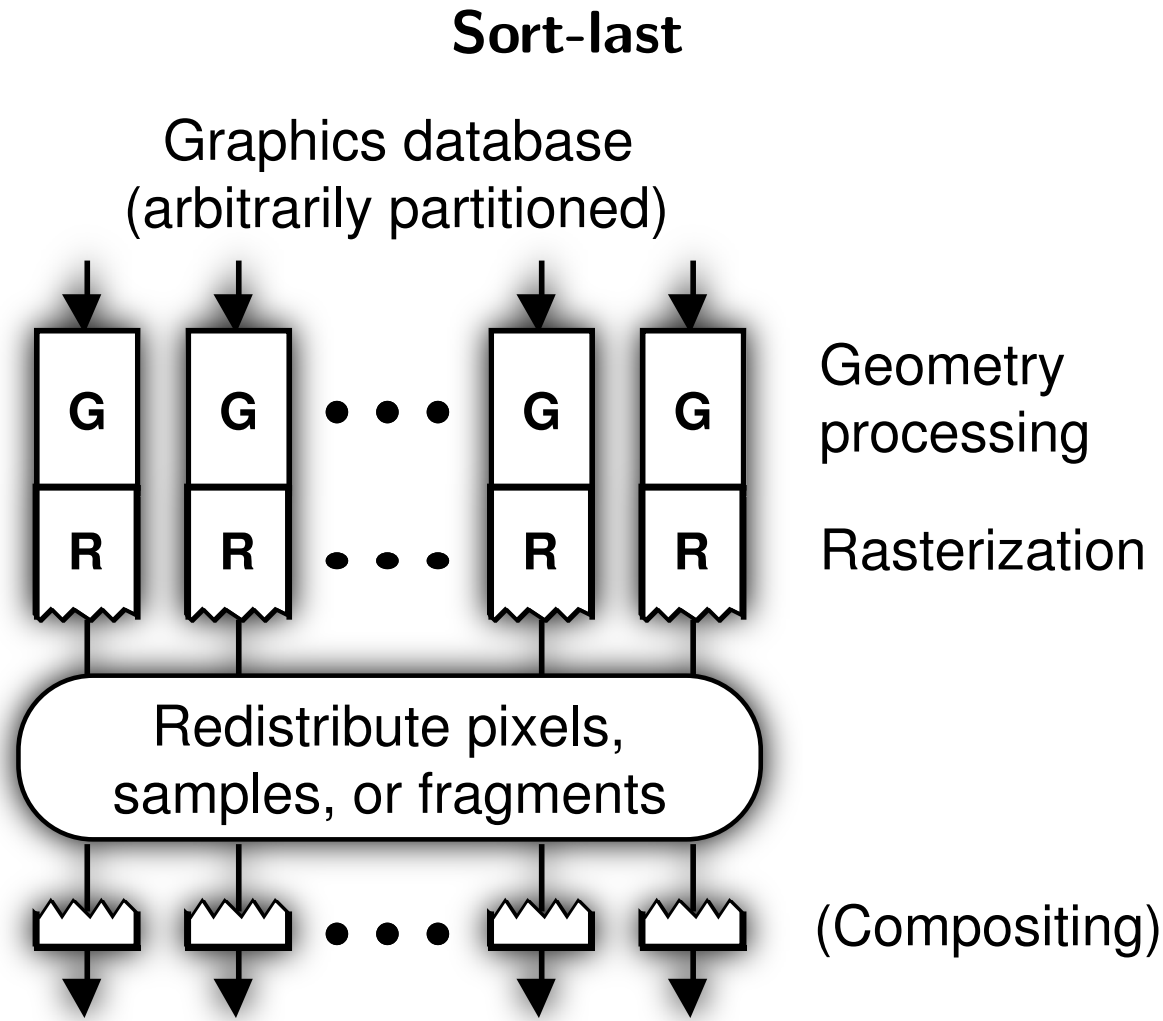
Types of parallel rendering



Types of parallel rendering

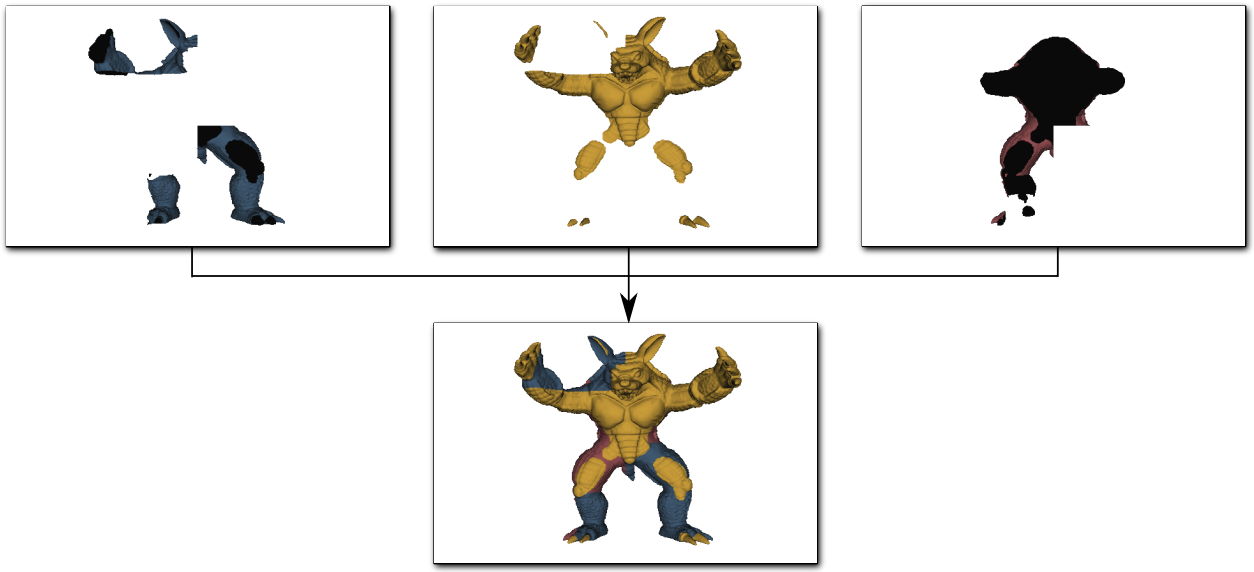


Types of parallel rendering



Database partition

- Divide up the objects between the processors.
- Each processor renders its part of the scene.
- The images are composed (using the z-buffer information) into a final image.
- Equivalent to Molnars's sort-last.



Database partition

Advantages

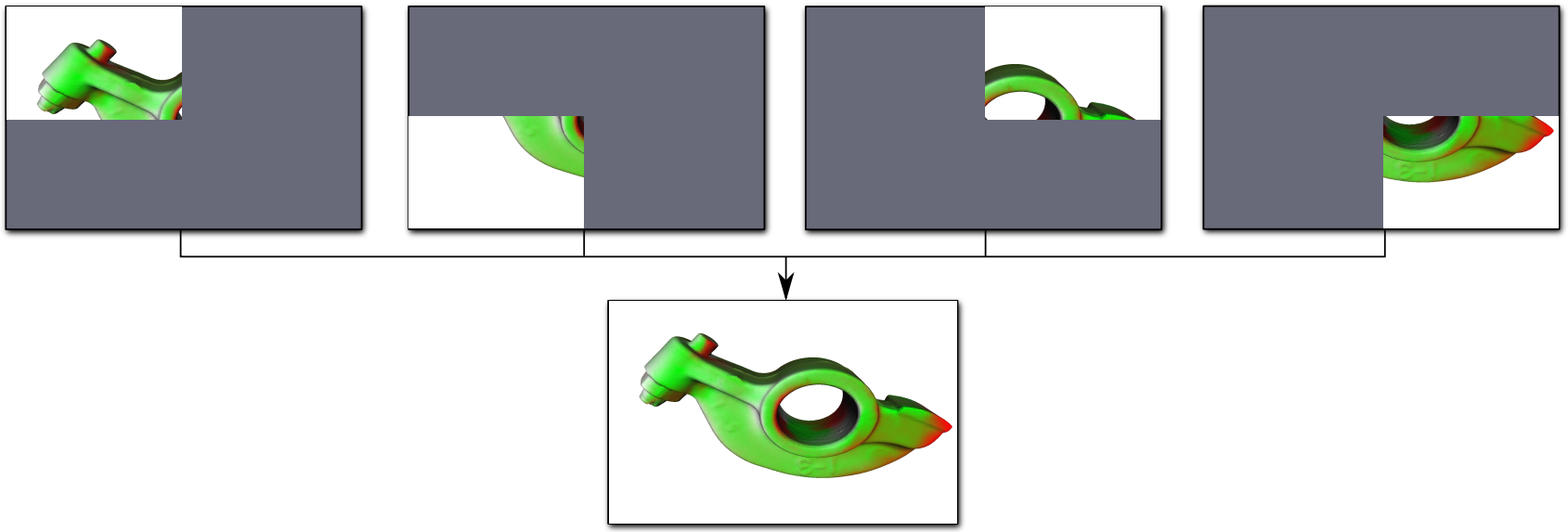
- Static partitions provide good load balancing in many cases.
- Appropriate when the rendering is geometry or memory limited.

Disadvantages

- The compositing step is costly:
  - GPU-CPU read-back
  - Uses more network bandwidth (could use compression or ROI)
  - Transparency is more difficult.
- Scalability is limited by the compositing step.
- Depending on the viewpoint the rendering can be unbalanced.
- Problems with global illumination and anti-aliasing.

Screen space partitions

- Divide the screen in regions and assign each region to a processor.
- A variation divides the scene in tiles and uses a master-slave scheme to dispatch work.
- Equivalent to Molnars's sort-first.



Credit: Stefan Eilemann, Eyescale

Screen space partitions

Advantages

- Ideal for fill limited applications
- Transparency and anti-aliasing are well supported.

Disadvantages

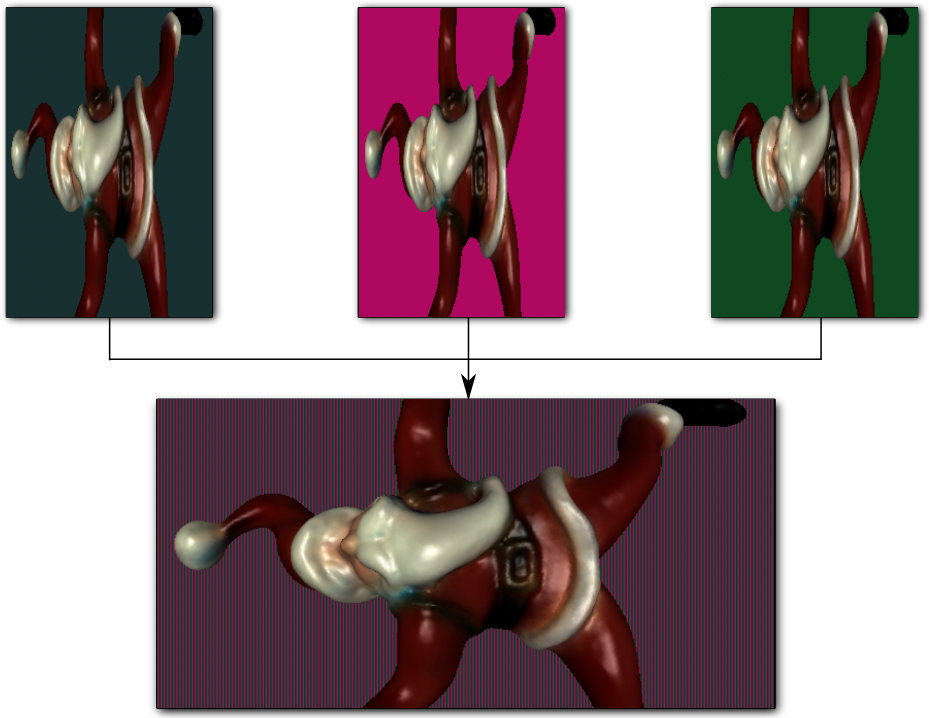
- Memory requirements are not decreased.
- More difficult to balance (needs dynamic load balancing)
- Scalability is limited by the view frustum culling algorithm.
- Problems with global illumination.

Neither sort-first nor sort-last is always better than the other.

Pixel/sample partitions

Pixel partitions

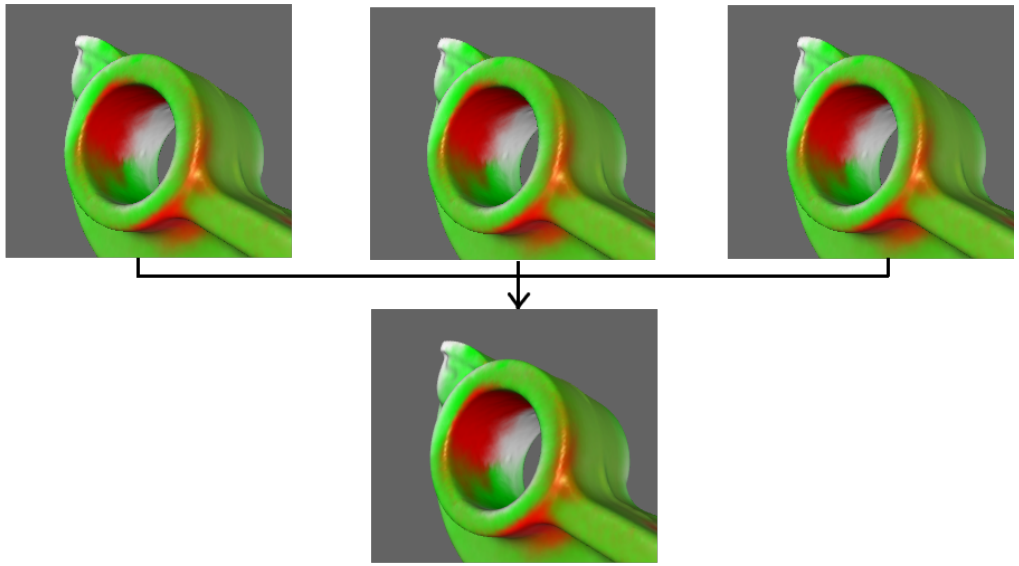
- Distribute pixels over processors in an interleaved fashion.
- The workload of geometry processing is not reduced.
- Only useful for applications that are purely fill-limited.



Credit: Stefan Eilemann, Eyescale

Sample partitions

- Similar properties to pixel partitions but different goal.
- The goal is to increase the rendering quality by reducing aliasing.
- Every processors renders the full image with a small shift in the camera.
- The images are averaged instead of interleaved.



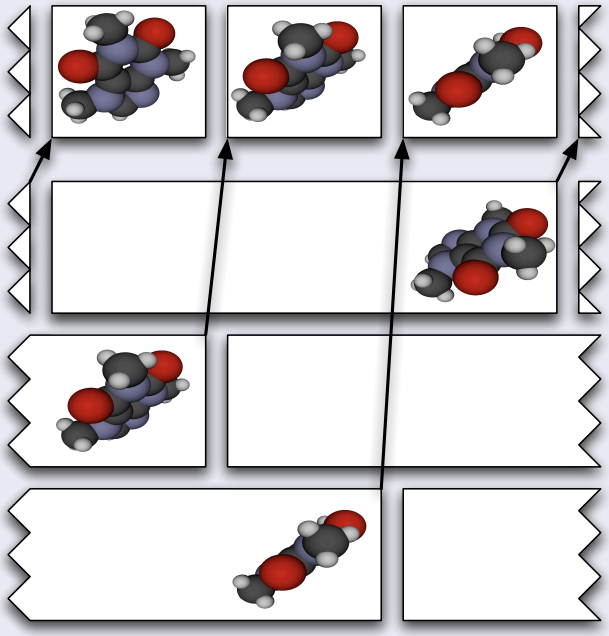
Credit: Stefan Eilemann, Eyescale

Frame distribution

Distribute frames over processors

Subtypes

- Time multiplexing



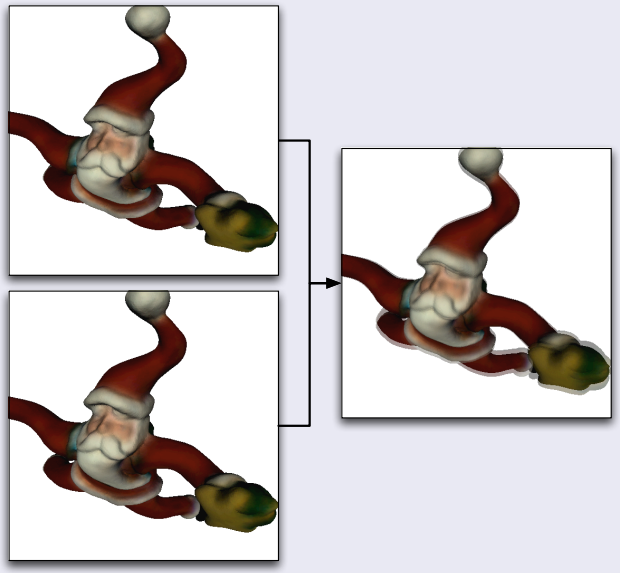
Credit: Stefan Eilemann, Eyescale

Frame distribution

Distribute frames over processors

Subtypes

- Time multiplexing
- Eye decomposition



Credit: Stefan Eilemann, Eyescale

Compositing

- Compositing is needed when the database is partitioned between the processors to merge the images into a single one.
- This step has a non-negligible impact in the final rendering time.

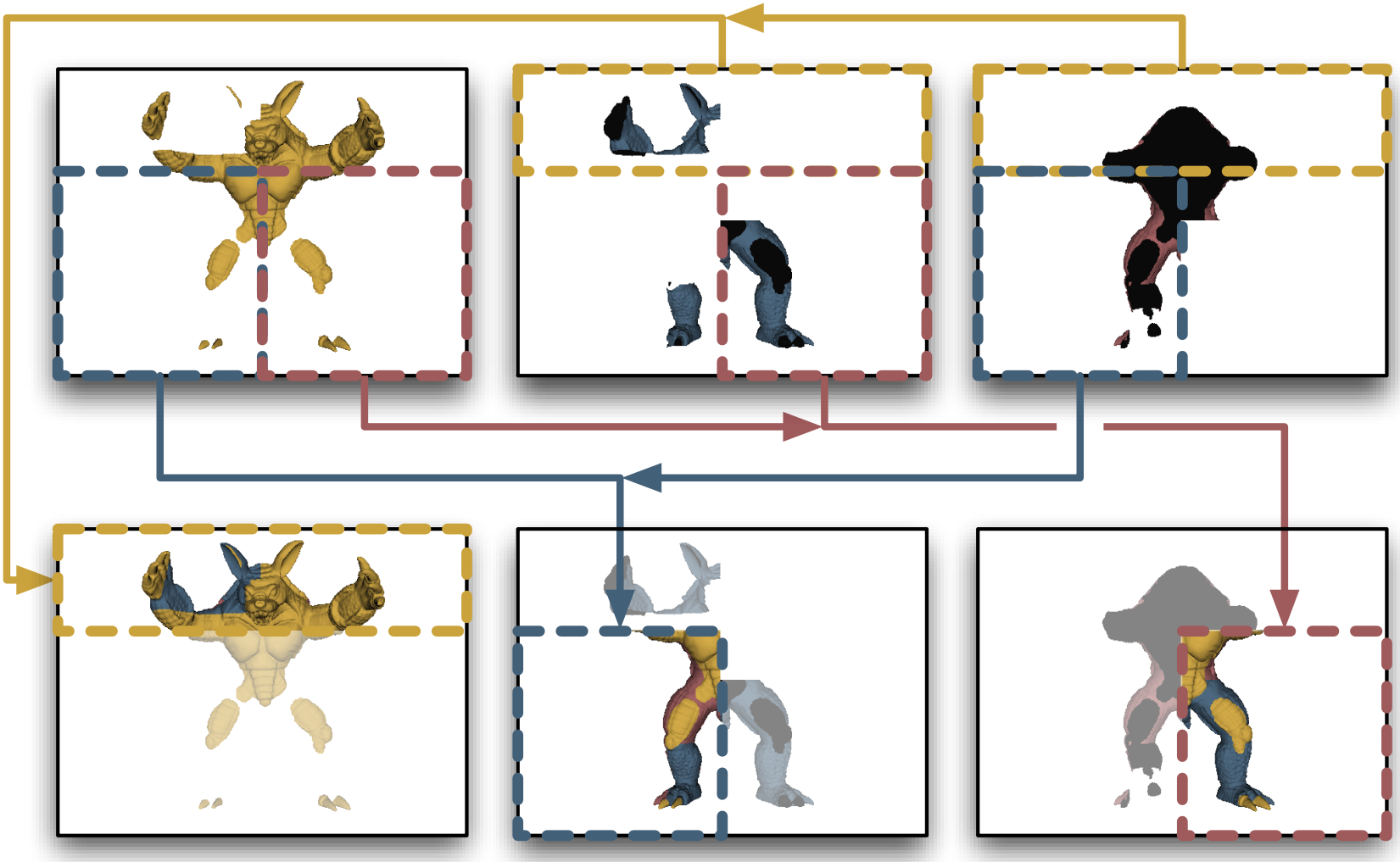
Algorithms

- All-to-one (trivial, unbalanced)
- Direct send
- Binary swap
- 2-3 swap

Direct send

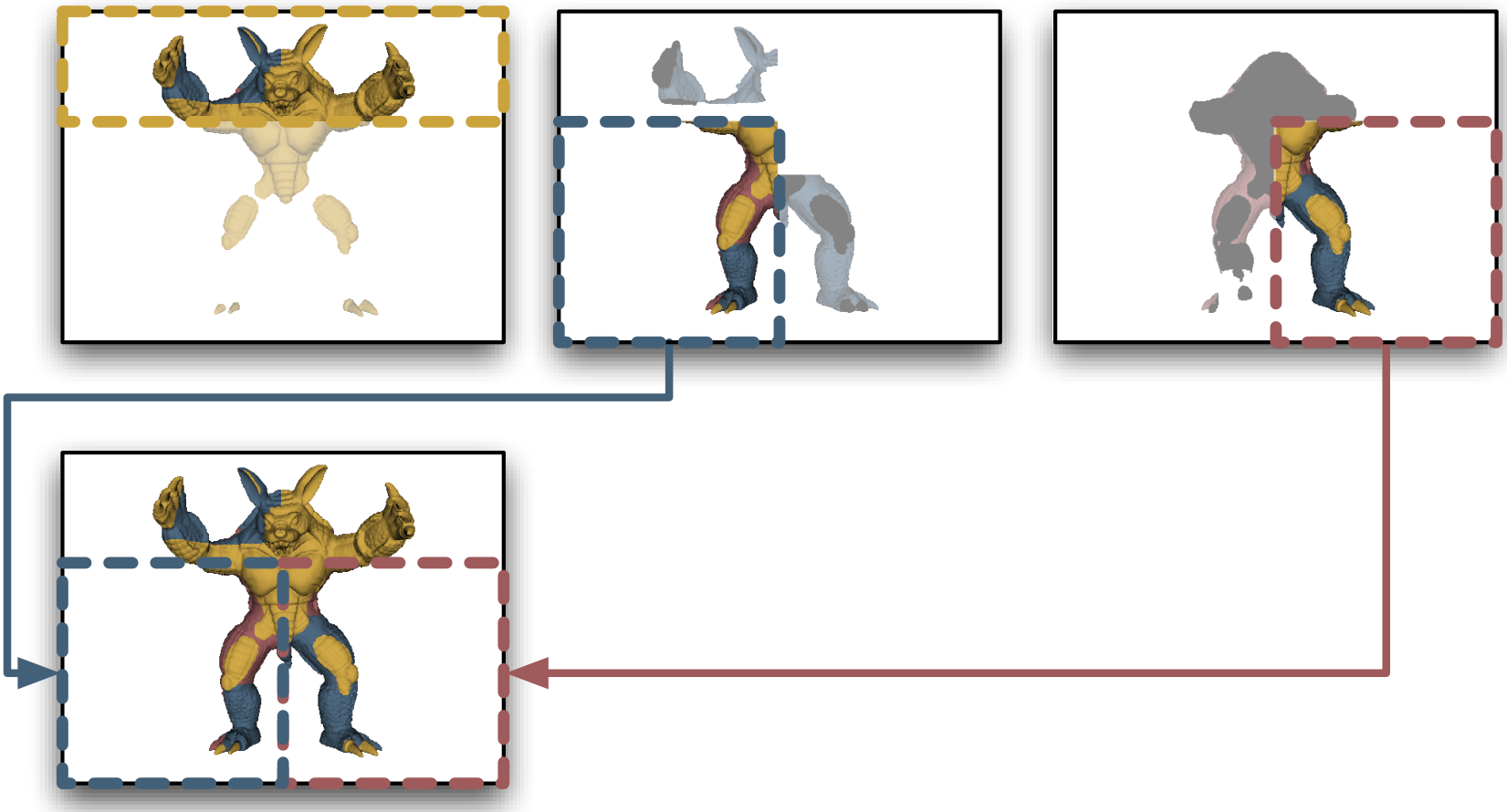
- Make all processors collaborate not only in the rendering but also in the compositing step.
- For n processors, each one is responsible of composing a $1/n^{th}$ of the image.
- Each processors sends to all the others the portion of the image they are responsible to compose
- In total $n(n-1)$ messages are exchanged.

Direct send



Credit: Stefan Eilemann, Eyescale

Direct send

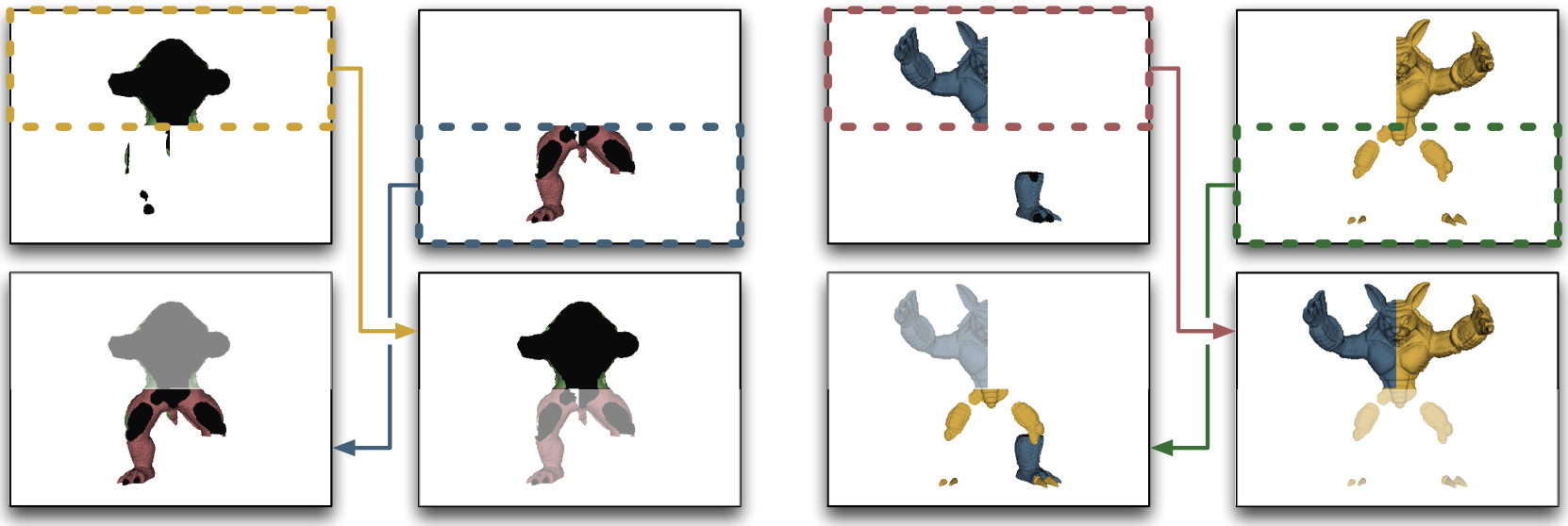


Credit: Stefan Eilemann, Eyescale

Binary swap

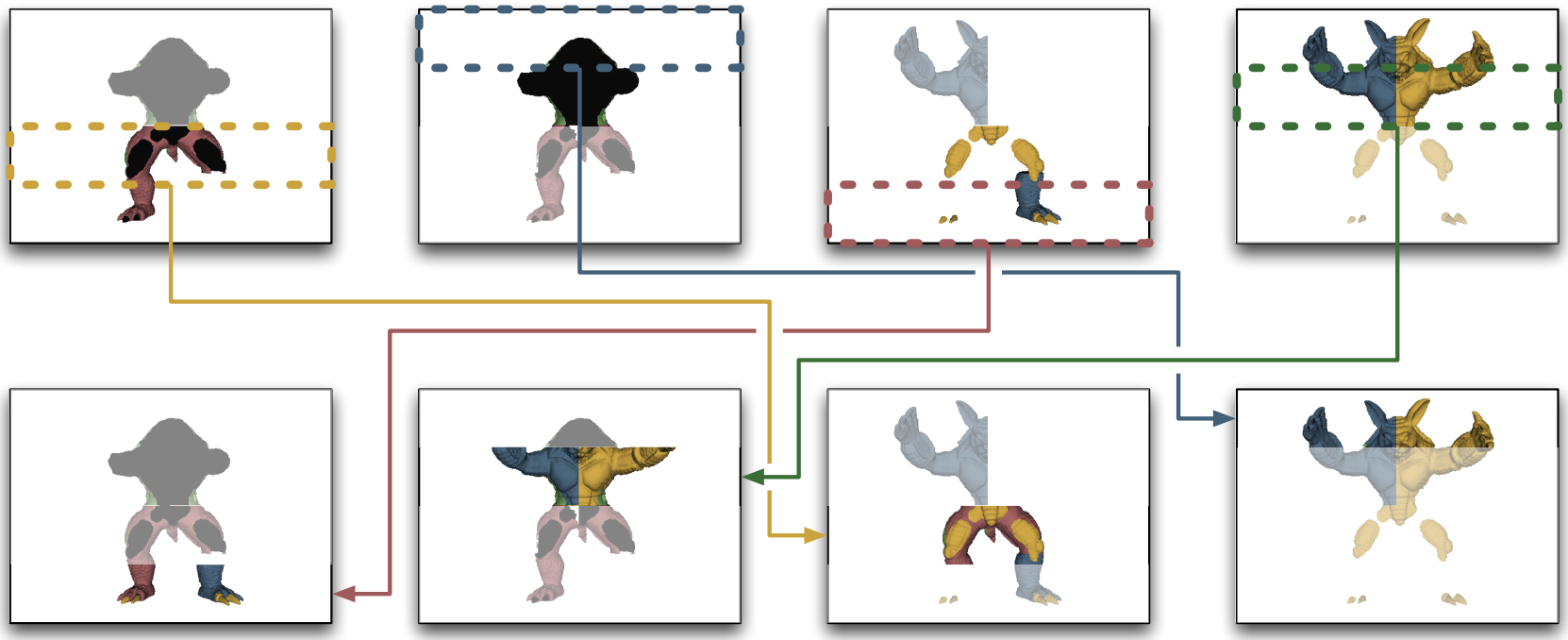
- Iterative algorithm that composes the image with a minimum number of messages.
- The image is composed in $\log_2(n)$ stages.
- At each stage, a processor exchanges data only with another one, and the image portion exchanged is halved each time (in the first step half the image is exchanged).
- At the end each processor has $1/n^{th}$ of the final image.
- More amenable to HPC interconnects than direct send.
- But, the number of processors has to be a power of 2.

Binary swap



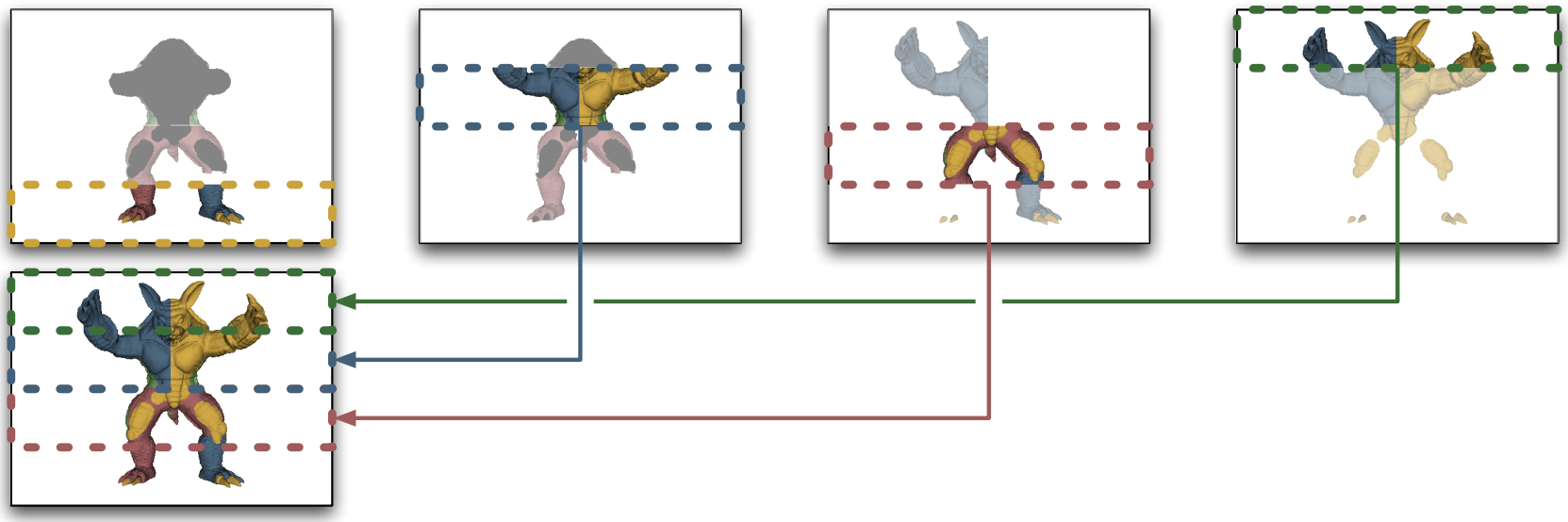
Credit: Stefan Eilemann, Eyescale

Binary swap



Credit: Stefan Eilemann, Eyescale

Binary swap



Credit: Stefan Eilemann, Eyescale

Libraries, frameworks and systems

- Equalizer: Feature rich parallel rendering frameworks to write parallel rendering applications.
- Chromium: library interposed between the application and the OpenGL library. Can parallelize applications transparently
- OpenSG: Distributed scene graph. Discontinued project
- Paracomp: Parallel compositing library by HP.
- IceT: Parallel compositing developed with HPC applications in mind
- Lighting-2, Sepia-2: Hardware compositors

Equalizer

Description

- Open-source framework written in C++ for parallel rendering applications using OpenGL.
- An Equalizer application consists of 3 types of processes (called nodes) that communicate through TCP sockets
  - The Application node: The main application class. It implements the outer methods of the rendering loop.
  - The Server node: start client nodes based on a configuration and connects the application to the rendering clients
  - Rendering clients: The nodes that perform the rendering.

http://www.equalizergraphics.com

Equalizer

Features

- Multiplatform and vendor independent.
- All the rendering parallelizations describe above.
- Hybrid parallelization schemes using *compounds*.
- Dynamic load balancing in sort-first and sort-last compounds.
- Flexible compositing mechanism.
- Zero conf cluster configuration using `gpu_sd`.
- Region of interest for readback operations.
- Plugin system for image compressors
- Stereographic rendering.
- More info:
  `http://www.equalizergraphics.com/features.html`