234Compositor Library

Ver. 0.1

User Manual

Advanced Visualization Research Team

Advanced Institute for Computational Science (AICS), RIKEN

7-1-26 Minatojima-minami-machi

Chuo-ku, Kobe, Hyogo 650-0047, Japan

February 20, 2015

ACKNOWLEDGEMENTS

Part of the results was obtained using the K Computer at RIKEN AICS.

Contents

1. Introduction

234Compositor is a parallel image composition library being developed to perform sort-last image composition on massively parallel environments. The initial goal was to develop a scalable image composition library for K computer, a CPU only supercomputer with tens of thousands of computational nodes, thus only a small subset of image composition functionality has been implemented. In this initial version, some enhancements to the existing Binary-Swap image composition algorithm have been implemented. These enhancements have been focused on these two points:

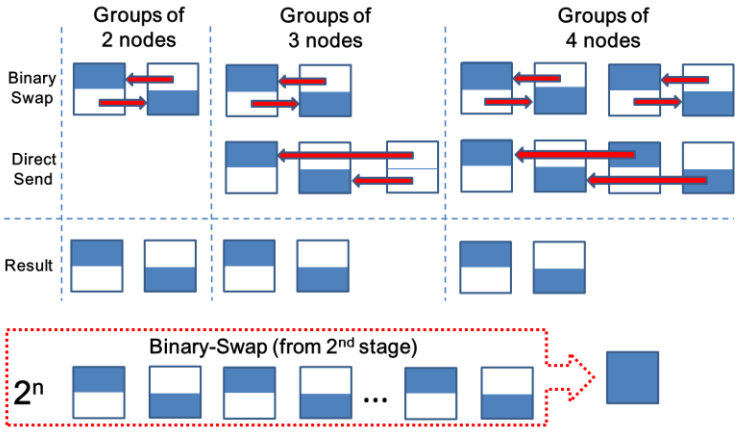
- Enable the use of arbitrary number of nodes

- Scalable final collection of image fragments

Binary-Swap has proven effective on massively parallel environments. However, it works only when the number of composition nodes is exactly a power-of-two. 234Compositor uses the 2-3-4 Decomposition approach for reducing any non-power-of-two number of nodes into the nearest power-of-two which is smaller than the original number of nodes. The 2-3-4 Decomposition performs the decomposing by applying the following rule and create sub-groups of 2 and 3 nodes; only 3 nodes; 3 and 4 nodes. These groups execute independently the image composition equivalent to the first stage of Binary-Swap image composition. This will result in a power-of-two number of nodes, and from this point, Binary-Swap starting from stage 2 can normally be applied.

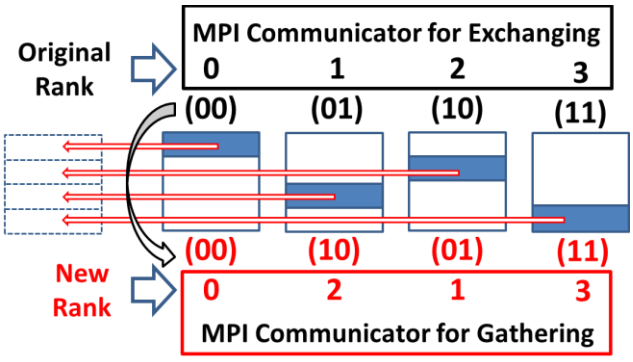
$$2^n < m < 2^{n+1} \qquad \begin{cases} 2-3 : 2^n < m < 2^n + 2^{n-1} \\ 3 \quad : m = 2^n + 2^{n-1} \\ 3-4 : 2^n + 2^{n-1} < m < 2^{n+1} \end{cases}$$

2^{n-1} Groups of 2, 3 or 4 nodes

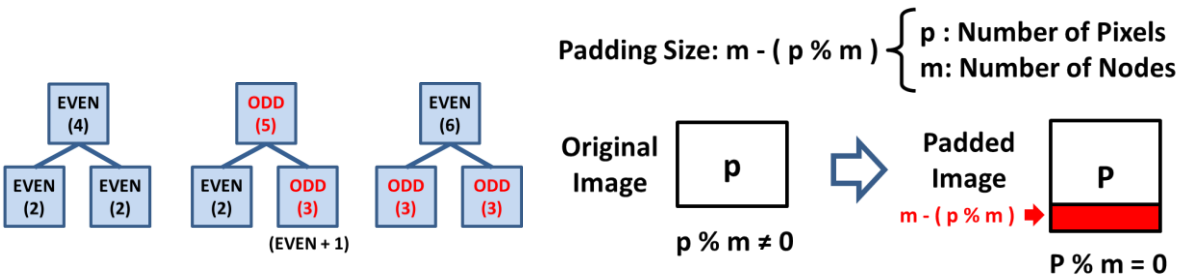


Another enhancement of 234Compositor is the use of MPI_Gather instead of traditional MPI_Gatherv during the final collection of already composited image fragments. On massively parallel environments, where tens of thousands of composition nodes can be involved, MPI_Gatherv can perform poorly compared to MPI_Gather. 234Compositor applies rank reordering and image data padding for enabling the use of MPI_Gather. Although it has shown effective on massively parallel environments, MPI_Gatherv can perform better on small and medium sized parallel environments due to the processing overhead.

The communication pattern of Binary-Swap, which is based on recursive distance doubling algorithm, makes necessary the rank reordering for applying the MPI_Gather since MPI_Gather collects the data fragments in sequential rank order. In order to enable the use of MPI_Gather, the original ranks are reordered using bit-reversal permutation as shown below.

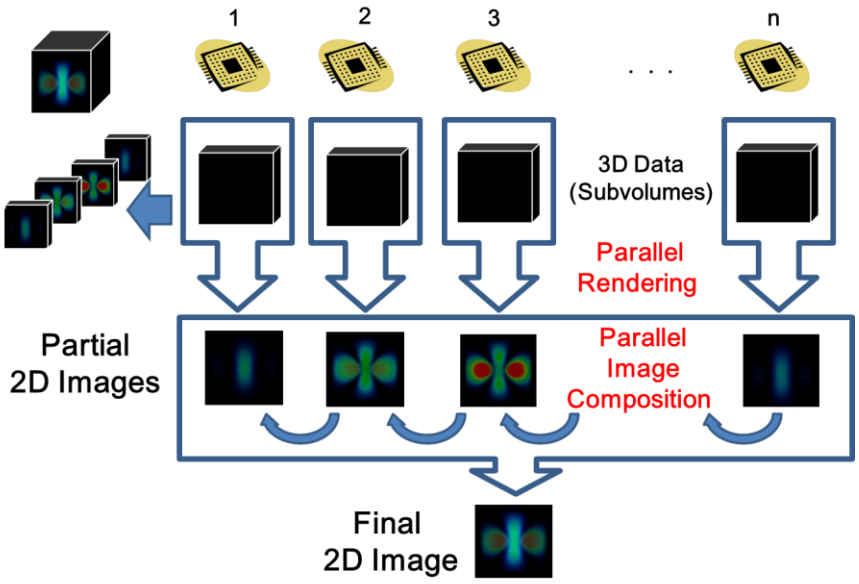


Depending on the number of pixels, the recursive binary subdivision can produce uneven image data distribution as shown below. If one of the sub images possesses odd number of pixels, it will produce unbalanced image division in the subsequent subdivision. Since MPI_Gather works by collecting fixed size data fragments, 234Compositor uses image data padding to avoid the generation of uneven image data distribution.

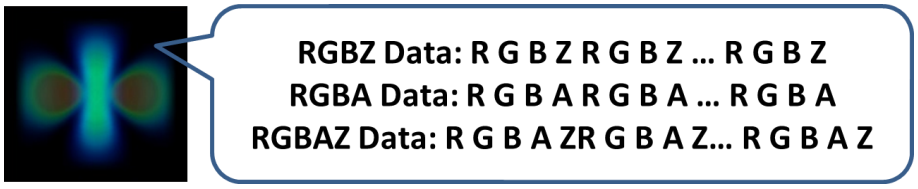


2. Assumptions

234Compositor was designed to assist sort-last parallel rendering applications for combining the rendered images into the final image as shown in the image below. That is, the 234Compositior was designed to be called by the parallel rendering application. Although different kinds of parallel rendering applications exist, only a subset of image composition functionalities has been implemented in this version considering that the initial goal was to development of a scalable image composition library for the K Computer.



The first assumption is that the image data is a sequence of pixels where each pixel can be a combination of color, transparency and depth information.



Although, different kinds of pixels have been used in different environments, only the following four pixels have been implemented in this version of 234Compositor:

- RGBA32 : 32-bit pixel (8-bit RGBA components)
- RGBAZ64: 64-bit pixel (8-bit RGB + 32-bit A components)

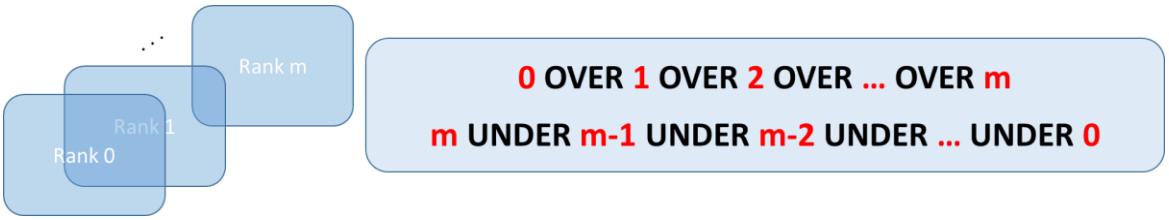
- RGBA128 : 128-bit pixel (32-bit RGBA components)
- RGBAZ160: 160-bit pixel (32-bit RGBAZ components)

The RGBA32 and RGBAZ64 image data are represented as BYTE* (unsigned char*) and the RGBA128 and RGBAZ160 image data are represented as float*. Other image sizes as well as image types including RGBZ are expected to be implemented in the future versions.

In addition to this, there also exist different pixel merging methods depending on the utilized rendering technique. Z-depth sorting for opaque pixels and alpha blending for semi-transparent pixels are probably the most common techniques, and other techniques such as the higher data value selection can be used for instance on MIP (Maximum Intensity Projection) rendering. In this initial version, only the back-to-front alpha blending has been implemented on 234Compositor.

- ALPHA_BtoF

In this case, it is assumed that the images are ordered correctly before calling the 234Compositor.



3. Building 234Compositor Library

234Compositor has been tested on K computer as well as x 86 environments, and is composed of the following files distributed in two subdirectories (src and include):

- src: 234compositor.c, exchange.c, merge.c, misc.c

- include: 234compositor.h, exchange.h, merge.h, misc.h

Through the “make” command it will create a static library named “lib234comp.a” in the lib subdirectory.

4. Basic Usage

4.1. API

234Compositor has a simple API consisting of the following three basic commands. The initialization call prepares the MPI Communicators for 2-3-4 Decomposition as well as for MPI_Gather with bit-reversed rank reordering. After that, until the changing in number of nodes and within the pre-defined image size range, which is explained later, the 234Composition can be called repeatedly without the need to call the initialization again. At the end, the destroy command can be called to release the allocated memory regions and MPI Communicators.

- Init_234Composition

- Do_234Composition

- Destroy_234Composition

In the parallel rendering application side, there is only a need to include the “234compositor.h” in the code and link the “lib234comp.a” together with the math library “-lm”.

API

```
int Init_234Composition  (my_rank, nnodes, \
                          width, height, pixel_ID )
int Do_234Composition  (my_rank, nnodes, width, height, \
                          pixel_ID, merge_ID, \
                          image_data, MPI_COMM )
int Destroy_234Composition ( pixel_ID )
```

- Rank (unsigned int): Rank number of the composition node
- Nodes (unsigned int): Total number of composition nodes
- Width (unsigned int): Image width (in pixels)
- Height (unsigned int): Image height (in pixels)
- pixel_ID (unsigned int): ID of the pixel type
    - RGBA32 : 0 or ID_RGBA32
    - RGBAZ64 : 1 or ID_RGBAZ64
    - RGBA128 : 2 or ID_RGBA128
    - RGBAZ160: 3 or ID_RGBAZ160
- merge_ID (unsigned int): ID of the pixel merging method
    - Back-to-Front Alpha Blending: 0 or ALPHA_BtoF
- image_data (unsigned char* or float*)
- MPI Communicator (MPI_Comm)

4.1. Test Program

A simple test program has been placed in sample subdirectory in order to facilitate the understanding the usage of 234Compositor. It is an offline image compositor which loads pre-generated RGBA images and composites them, and outputs as an jpeg image. The input images to be loaded are placed in the “input_images” subdirectory.

- test_234byte.c (Offline image composition for RGBA32 images)

- test_234float.c (Offline image composition for RGBA128 images)

Both sample codes should be linked with “lib234comp.a” and “-lm -ljpeg” libraries.

The input image generator (generate_image.c) is placed on “input_image” subdirectory and requires GLU and glut graphics libraries (-lGLU –lglut). Both libraries are also required for linking the image data viewer (show_image.c). The usage of these two supporting tools is quite simple:

- generate_image Width Height pixel_ID Number of Images
- show_image Width Height pixel_ID