Table of Contents

Introduction and Overview

Chromium is a stream-processing framework that implements the OpenGL API (see the Chromium website at Sourceforge for more information). Simplistically, Chromium is intended to be used to drive multiple graphics servers as a single logical device. With Chromium, you can have a garden-variety OpenGL program make use of a tiled array of projectors, where each of the different displays may be a single node in a PC cluster, or a different graphics pipe in a large SMP system.

OpenRM Scene Graph provides scene graph services to applications. These services implement high-level capabilities not present in the graphics API. In addition, scene graph services encapsulate many details of graphics API programming. In addition to providing features not present in the graphics API, like high level semi-procedural primitves, OpenRM provides a large number of features that are simply not present in any graphics API, like multipass rendering, application callbacks, selective rendering and data management services.

The main motivation for the OpenRM+Chromium work is to provide the means to have a parallel application make use of a scene graph system that is parallel-aware. While it is possible to take nearly any serial OpenGL program and have it drive an arbitrary number of graphics servers using Chromium, it is still a serial graphics application. In many cases, it is desireable to parallelize the graphics application itself in order to realize performance improvements. Scientific visualization is one such example, where large datasets are loaded and processed in parallel. Not only is I/O bandwidth ammortized across many processors, but CPU load and memory use is also ammortized. Using parallelization, it is possible to tackle problems that are much too large to fit on a single machine. Enabling processing and visualization of such large problems is the primary motivation of our work.

One troublesome issue when considering support for general purpose parallel programming is the notion of the parallel programming model. There are several different widely-used parallal programming models: the Message Passing Interface, also known as MPI (see http://www-unix.mcs.anl.gov/mpi/, the Parallel Virtual Machine, also known as PVM, (see http://www.csm.ornl.gov/pvm/pvm_home.html), and OpenMP (see http://www.openmp.org/). Generally speaking, applications written to use one parallel programming model must be ported to use a different one. One of our primary design objectives in the OpenRM+Chromium work was to minimize, if not eliminate, the dependency upon any parallel programming environment in order to achieve maximum portability.

Our implementation of "parallel OpenRM" realizes those goals, and is not dependent upon **any** parallel programming framework. To realize that objective, we made design decisions based upon two realizations. The first is that parallel rendering, in general, requires interprocessor synchronization in certain key areas. Such synchronization is accomplished through the use of constructs provided by Chromium. Like OpenRM, Chromium may be used by parallel applications using any parallel processing framework. Chromium's synchronization mechanisms consist of semaphores and barriers. Therefore, the rendering synchronization in a parallel OpenRM+Chromium application occurs during rendering, not in the application - the application is never aware that any synchronization is being performed. "The right thing" just happens.

Second, with a couple of exceptions, the Chromium-enabled implementation of OpenRM does not provide any explicitly parallel scene graph operations. Instead, it is the application developer's responsibility to implement application-level parallelism. In other words, each application process creates its own private scene graph based upon whatever form of spatial data decomposition it deems best for the task at hand. Also, each application process must invoke the OpenRM frame-based renderer, which effectively causes parallel rendering to happen. Sorting and routing of geometry from application process space to one or more graphics servers is provided by Chromium. Any application-level parallelism must be created by the application developer. This approach produces an elegant, compact and extremely flexible scene graph implementation. The parallelism that OpenRM provides to applications is rendering stream synchronization for (1) all framebuffer clear operations, and (2) swapbuffers. These operations and their motivation are described more fully in our 2003 IEEE Visualization PVG paper.

In addition to being able to take advantage of parallel platforms for increased application processing and graphics rendering performance, all the "usual" scene graph tricks work in the parallel environment. Each different application process can use view-dependent operations, like LOD-based model switching and view frustum culling, along with useful OpenRM capabilities like multipass rendering and a rich set of application callbacks to assist in the implemention of application-specific features.

Software You'll Need to Build and Run the OpenRM+Chromium Demonstration Programs.

In order to run these demonstration programs, you will need the following software components:

1. **OpenRM Scene Graph.** openrm-devel-1.6.0.tar.gz from the download page of the OpenRM website at Sourceforge. OpenRM version 1.5.0 and beyond support direct use with Chromium in parallel applications.
2. **OpenRM + Chromium Demo Programs** Source code for the OpenRM+Chromium demonstration programs from the download page of the OpenRM website at Sourceforge. Building the demos, running them and a detailed explanation for each are provided below.
3. **Chromium** You will also need a current version of Chromium. Our development has been performed using v1.8, which is available for download at the Chromium website at Sourceforge. sourceforge.net/projects/chromium
4. **MPI or MPICH** Since the OpenRM + Chromium demo programs are distributed memory parallel and use the Message Passing Interface to implement parallelism, you will need an implementation of MPI. You can grab a copy of MPICH, the freely available and portable implementation of MPI, from http://www-unix.mcs.anl.gov/mpi/mpich/, provided by Argonne National Laboratory.
5. **OpenGL** It is assumed that OpenGL is installed on your machine.

Building Chromium-enabled OpenRM

Building the Chromium-enabled version of OpenRM is no different than building any other version of OpenRM, with the exception that you must use the Chromium-specific build target. The following example shows unpacking the source tarball, which is assumed to have been downloaded into /tmp, into /usr/local/rm160, then building the Chromium-enabled version of OpenRM. You can unpack and build in any directory, but will later need to modify your LD_LIBRARY_PATH variable, which is described in more detail in the next section, "Building the Demos."

Note that the Chromium location is specified in the RM160 configuration file, located in file in /usr/local/rm160/make.cfg (in this example). make.cfg assumes that you have installed Chromium in /usr/local/cr. If your Chromium installation lives elsewhere, you will need to modify make.cfg to point to the root directory of your Chromium installation.

```
# cd /usr/local
# tar xvfz /tmp/openrm-devel-1.6.0-cr.tgz
        ... verbose output omitted ...
# cd /usr/local/rm160
# make linux-cr-debug
        ... compile output omitted...
```

Building the OpenRM+Chromium Demo Programs

First, unpack the demonstration programs into some directory. We'll assume that you'll unpack into /tmp for this example. When these steps are completed, the OpenRM+Chromium demos will be located into /tmp/rmdemoCR

```
% cd /tmp
% tar xvfz /tmp/openrm-demo-CR-1.6.0-2.tar.gz
```

Before building, you'll need to swim through the rmdemoCR/Makefile, and change the pathnames to OpenRM, Chromium and MPI to reflect the actual locations on your machine. There are comments in the Makefile to guide you through this process.

If you are new to building MPI programs, note that most MPI implementations provide commands called "mpiCC" and "mpicc" (or "mpicxx" and "mpicc", depending upon which version of MPICH or MPI you are using. These commands are wrapper scripts for the compilers and linkers. Before you can build the OpenRM+Chromium demo programs, you need to place the path to the mpicc and mpiCC executables in your $PATH environment variable so that "make" will work.

When the paths in the Makefile have been adjusted, just type "make." Note that we have tested ONLY on Linux systems. The Makefile provided with rmdemoCR will need a lot of work to function properly on other platforms due to the vagaries of MPI vs. MPICH, etc. You may need to modify the Makefile to successfully build on your system.

Running the OpenRM+Chromium Demonstration Programs

Preamble: before attempting to run the OpenRM+Chromium demonstration programs, you should first make sure that the demonstration programs included with Chromium work correctly. In addition, you should make sure that MPI is configured correctly, and that the example programs included with the MPICH distribution execute on your platform. You must ensure that both Chromium and MPI are correctly configured and operational before running the OpenRM+Chromium demonstration programs. Chances are that 99% of problems you may encounter while attempting to run the OpenRM+Chromium demonstration programs stem from incorrectly configured Chromium or MPI.

The demonstration programs and four config files included with this distribution allow you to test various combinations of application and Chromium parallelism. At this time, you must launch the mothership using one of the four provided configuration files as the OpenRM implementation obtains the mural size from the mothership in order to set the viewport in the scene graph. If you run the demo programs without first launching a mothership, the programs will run, but the viewport will be nonsense, and you won't see much in the display window, at best, or the program might crash, at worst. In addition, the conf files load the array spu, which is needed to support OpenRM's extensive use of vertex arrays to accelerate rendering performance.

For these examples, "auto-start" is turned on inside the CR-config files. That means you must have your machine set up to accomodate Chromium's auto-start features. Check the CR docs for more information The provided .conf files will auto-launch crservers on the machine 'localhost', so that all parallel tasks are run on a single machine. It is assumed that your MPICH configuration launches all PEs on localhost as well. In our testing, we have used configurations that span many machines, but including those configuration files with the rmdemoCR distribution would be confusing, since those configuration files are entirely site-specific.

The general template for running the one-way parallel demos is as follows:

1. In one window, cd to rmdemoCR/confs, and launch the mothership: e.g., python singleCPUlocalhost.conf
2. In another window, run one of the programs. If you compiled using MPICH, you don't need to use the "mpirun" command to launch a one-way parallel job. You can just invoke the executable from the command line. All demo programs do something reasonable when no command-line arguments are specified. The isosurface demos can take optional command-line arguments, but none are needed.

The template for running four-way parallel jobs is as follows:

1. In one window, cd to rmdemoCR/confs, and launch the mothership: python fourAppNodesFourDisplaysLocalhost.conf
2. Launch one of the demo programs via mpirun as a four-way parallel job:

```
% cd rmdemoCR
% mpirun -np 4 isoBlocksMPI
```

Notes for parallel execution:

1. Set your environment so that it is "complete" when you open a shell. By default, MPICH doesn't propogate your environment when you launch parallel jobs. For the purposes of running these demos, you need to have LD_LIBRARY_PATH and your $PATH set appropriately.
2. Your environment variable LD_LIBRARY_PATH should point to the Chromium libraries, as well as the OpenRM libraries (adjust this line to reflect the actual location of Chromium and OpenRM on your machine):

```
setenv LD_LIBRARY_PATH /usr/local/cr/lib/Linux:/usr/local/rm160/lib
```

3. I put the path to mpirun, mpicc and friends in my .cshrc file so that these executables are always available.
4. (October 31, 2003) All of the OpenRM+Chromium demo Chromium config files (no typos there) use Chromium's "auto start" feature to launch crservers on multiple nodes. Correspondingly, the "crfaker" part of the operation is provided by the parallel launch and run of the parallel application via MPI. One final detail you need to take care of that is needed for "auto start" to work correctly is to create a symlink that points libGL.so.1 to Chromium's "faker" library, the one that presents the OpenGL API to the application. To do so, perform the following steps.

```
% cd /usr/local/cr/lib/Linux
% ln -s libcrfaker.so libGL.so.1
```

   Finally, verify that your LD_LIBRARY_PATH is set correctly, as indicated above, and run the "ldd" command on your application to make sure that the libGL.so.1 in /usr/local/cr/lib/Linux is being used, not the system's libGL.so.1, which is probably located in /usr/lib.

OpenRM + Chromium Demonstration Programs

Detailed information about the OpenRM+Chromium demonstration programs, along with information about the Chromium configuration files included with the rmdemoCR distribution.

This page last modified Thursday, 11-Aug-2005 07:23:26 PDT
Web problem or question? Send email to webmazen at r3vis.com