

CRUT: The Cluster Rendering Utility Toolkit for Chromium

Dale Beermann

December 17, 2002

1 Abstract

As the demands for rendering power keep increasing, clusters of workstations are becoming popular as an alternative “big iron” solutions. Clusters offer a cost-effective approach to dealing with the problem of limited resources regarding graphical complexity. With this increasing popularity it is becoming necessary to create tools to aid this growth. Chromium has recently been introduced as one system to allow the manipulation of streams of graphics API commands on clusters of workstations [1]. We define CRUT, a toolkit for Chromium, to facilitate event processing using this form of distributed environment. Without CRUT, Chromium is unable to allow the end user to interact with applications running on a cluster of workstations. CRUT makes this possible by extending the functionality of Chromium to handle application-defined callbacks executed upon receiving events sent over Chromiums network layer.

2 Introduction

Chromium provides a general mechanism for enabling interactive graphics on clusters [1]. However, interactivity, when talked about in the computer graphics community, does not always mean user-interactivity. Chromium makes it possible to visualize datasets and run applications that would not be able to run on a single workstation, but as of yet users are not able to interact with the data by sending events to the application. Because of the distributed form of computing that Chromium presents, user-interaction is decoupled from the application. In most cases parallel applications will be running on different worksta-

tions than the ones displaying the results of the application. For this reason, even if an application was written to receive events such as mouse or keyboard events, the user is still interacting with a different window than the one put up by the application. As of yet, Chromium has no way to return user events on a display node back to the application.

With Chromium, a Directed Acyclic Graph, or DAG, can be defined that describes the flow of data through the system. Each node in the graph represents a workstation in the cluster. Each workstation can perform multiple operations on a stream of OpenGL commands through Stream Processing Units, or SPUs. In a local configuration, Chromium is able to render back into the original application window, but in a distributed application, user-interaction cannot be accomplished. Any interaction must be built into the application, making it possible to perform walkthroughs and to display rotating datasets, but the user is unable to perform operations like simple pan and zoom.

CRUT makes it possible to program a user-interactive application to run on Chromium. It is modeled after GLUT (the OpenGL Utility Toolkit) so that it is intuitive and easy to learn [3]. This will also allow applications programmed using GLUT to be easily migrated to run under Chromium. An application using CRUT is able to register callbacks with the client library that will be called every time a corresponding event is generated. A separate program, the `crutserver`, creates a window that Chromium’s `renderspu` can render into. The `crutserver` uses GLUT to capture events and only those registered to be received by the application will be sent across the network. Chromium’s networking layer is used

to communicate user events back to the `crutclient` library. Upon receiving these events the client library calls the registered callbacks.

3 The CRUT Architecture

We define an API that is the basis of CRUT. On the server side, the API allows custom GUIs to send events back to the application. The `crutserver` is an example implementation of an event server using the CRUT API and GLUT to capture events. The `crutserver` is a standalone program that puts up a window on the display node into which the Chromium `renderspu` can render. It receives events and sends those asked for by the application across the network to be handled by the application using the CRUT client API. The client API is the library that applications use to create a context and to register event callbacks. We also implement a proxy server that is responsible for receiving events from an event server and distributing them to the clients.

3.1 The CRUT API

The CRUT API is responsible for sending events across a network to be received by CRUT clients. We realize that GLUT does not provide ideal functionality for many applications so we would like the functionality of CRUT to exist for other toolkits as well. By creating an API for CRUT we make it possible to build a custom GUI into which the application can render. Using the API, the GUI can send events it has caught across the network so that callback functions can be executed on the application side. This way we do not rule out the possibility of using toolkits such as Tk or Motif.

One of our design considerations in implementing CRUT was to create something that was entirely detached from rendering operations. CRUT is solely responsible for handling events and distributing them throughout the system. The only communication needed between CRUT and the Chromium `renderspu` is that the `renderspu` must be notified of the window that the event server has created. In doing this, we provide a level of separation such

that event generation need not be dependent on any graphical context.

3.2 THE CRUT Client API

The CRUT client API is the library used by applications to register event callbacks and receive events. The syntax is much like that used for GLUT. In this way, for those familiar with GLUT, the library is easy to get to know. As stated before, this should also make it easy to port existing applications using GLUT to run under Chromium. There are some differences however. Notably, it is not necessary to create a window as it is with GLUT. For example, an application could use `crutCreateContext` to create a rendering context. This in turn uses the `glCreateContextCR` extension in Chromium. This is beneficial since most applications running under Chromium will have no need to put up a window.

Once the user has initialized the `crutclient` library and registered callbacks, `crutMainLoop` can be called to let the program run. The `crutMainLoop` function is responsible for telling the Chromium mother-ship what events the application will be expecting. This will in turn be used by the `crutserver` so that unnecessary events are not sent across the network. When called, `crutMainLoop` continuously checks for messages received from a `crutserver`, and executes registered callback functions based on the events it receives.

We recognize that giving up control of the application to CRUT can detract from the functionality of an application. For this reason, we also provide functionality to poll for events that have occurred. This may be important for applications which need to have complete control over their environment. Much like programming with Xlib, an application can be written that checks for events whenever it wants; it does not require the use of `crutMainLoop`. CRUT buffers all messages received so that they can be retrieved whenever possible. Using `crutCheckEvent`, an application can poll for events and perform any operation based on them.

3.3 The crutserver

The `crutserver` is the standalone program that takes care of sending events back to the original application. When the `crutserver` starts up, its first job is to create a window using the parameters specified in the configuration file (e.g. window dimensions and position). Once a window is created by the `crutserver`, the window id is registered with the mothership. The Chromium `renderspu` already has the ability to render into the original window put up by the application, so it is fairly straightforward to extend Chromium to render into a different window. When the `renderspu` is initialized, all it needs to do is retrieve the window id from the mothership for the `crutserver` window and make current to the window. As far as the underlying operation of Chromium is concerned, it can keep executing as normal. Once the `crutserver` has created a window and registered its id with the mothership, it must connect to the application in some manner. CRUT provides a few different options for network configurations which will be addressed in section 2.3.

The number of events it is possible to generate may pose a problem with CRUT. Since events are being sent across a network, it is important to send only those necessary. It is the job of the client library to register needed events with the mothership so that the `crutserver` is able to find out what events it needs to send. Once the `crutserver` has the necessary network connections and it knows what events to send, all it has to do is wait for events and send them when they are generated.

3.4 CRUT Network Structure

Using the Chromium configuration files, certain application nodes can be specified to receive events from the `crutserver`. As shown in figures 1 and 2, the network graph created by the `crutserver` and its CRUT clients is a DAG which can have different configurations. Figure 1 is an example of the CRUT network in a fan-out configuration. Here the `crutserver` simply sends generated events to each of the clients defined in the configuration file. Figure 2 shows a ring network configuration where the `crutserver` sends the

events to one client and events are cascaded to the remaining clients. CRUT also provides the ability to define different network configurations such as a tree-based approach. Thus, different applications can deal with events as they deem necessary.

The ring network configuration is an attempt to alleviate problems that may arise from sending multiple events to multiple nodes. In the future, Chromium will be capable of remote rendering, in which case another problem arises. Most nodes in the cluster will not be accessible outside the high-speed network on which they reside. For this reason, we have also created a `crutproxy` server which will handle incoming events and send them to clients in the cluster. The purpose is that the `crutproxy` server will be running on a computer that is connected to both an outside network and to the network for the cluster. The `crutproxy` server can then perform the same functions as the `crutserver` in the configurations mentioned above, receiving events from a remote `crutserver`.

4 Future Work

Currently CRUT does not support the use of menus as GLUT does. We are currently developing a way to define the structure of menus so that they can be defined as a parameter on the mothership. One preliminary thought would be to create an XML structure to describe the hierarchy of menus. We expect that this functionality will be available in the near future.

We also need to analyze the number of events it is possible to generate on the user-end of the system. Since events can be generated very rapidly, we may need to look into consolidating messages passed between nodes. One option is using something similar to the command packing functionality Chromium uses for OpenGL commands. However, since ideally events should arrive at the application soon after they are generated, few events would be packed together and sent at a time. Another option is to simply drop some events. Depending on the application, we could also make it possible to define different actions to take.

5 Conclusions

We have defined CRUT, the Cluster Rendering Utility Toolkit for Chromium. CRUT enables applications to register callbacks for user-generated events. We have taken many circumstances into consideration when designing CRUT. Because the nature of programs written for Chromium is much different than most OpenGL applications, we have provided a flexible structure that does not force an application to give up its control. This allows applications to be customized for specific needs while still taking advantage of the functionality CRUT provides. We accomplish this by providing an API that custom applications can use to send events as well as the possibility of event queuing. CRUT also attempts to provide solutions for bandwidth problems that may arise from generating multiple events in a small amount of time. This is done by providing different network configurations, as well as a proxy server, to receive and disperse events as needed. By adding these elements to Chromium, we have added another level of interactivity that will provide many new uses for Chromium.

References

- [1] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, J. T. Klosowski. *Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters*. In Proceedings of SIGGRAPH 2002, pages 129-14, 2002.
- [2] B. Johanson, A. Fox, P. Hanrahan, and T. Winograd. *The event heap: An enabling infrastructure for interactive workspaces*. Technical Report CS-2001-02, Stanford Computer Science Department, Stanford, CA, 2001.
- [3] M. J. Kilgard. *The OpenGL Utility Toolkit (GLUT) Programming Interface, API Version 3*. Silicon Graphics Inc., 1996.