Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER
UNCLASSIFIED	514903

TITLE

Introduction of the PEX extension for X windows into the RTX graphical user interface

System Number:

Patron Number:

Requester:

Notes:

DSIS Use only:

Deliver to:

This page is left blank

This page is left blank

DEPARTMENT OF NATIONAL DEFENCE
CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT & LAND)

DOR(J&L) RESEARCH NOTE RN 2000/21

INTRODUCTION OF THE PEX EXTENSION FOR X WINDOWS INTO THE RTX GRAPHICAL USER INTERFACE

BY

Mr. John Green
COOP Student

NOVEMBER 2000

OTTAWA, CANADA

National Defence Défense nationale

OPERATIONAL RESEARCH DIVISION

CATEGORIES OF PUBLICATION

ORD Reports are the most authoritative and most carefully considered publications of the DGOR scientific community. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORD Reports are approved personally by DGOR, and are subject to peer review.

ORD Project Reports record the analysis and results of studies conducted for specific sponsors. This Category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorates.

ORD Journal Reprints provide readily available copies of articles published with DGOR approval, by OR researchers in learned journals, open technical publications, proceedings, etc.

ORD Contractor Reports document research done under contract of DGOR agencies by industrial concerns, universities, consultants, other government departments or agencies, etc. The scientific content is the responsibility of the originator but has been reviewed by the scientific authority for the contract and approved for release by DGOR.

National Defence | Défense nationale

National Defence Headquarters
Ottawa, Ontario
K1A 0K2

Quartier général de la Defense nationale
Ottawa (Ontario)
K1A 0K2

3550-3-2 (DOR(J&L))

25 November 2000

Distribution List

DOR(J&L) RESEARCH NOTE RN 2000/21

1. Enclosed is a copy of DOR(J&L) Research Note RN 2000/21 "Introduction of the PEX Extension for X-Windows Into the RTX Graphical User Interface" for your information and retention.

2. It describes the first steps to integrate the PEXlib into RTX in order to evaluate the benefits.

3. Questions on this research note should be addressed to Pierre Ladouceur at (613) 992-4550.

for R.G. Dickinson
Director Operational Research (Joint & Land)
for Director General Operational Research

Distribution List

Mr. John Green
Army Simulation Centre, Kingston
DOR(J&L)
P. S. Ladouceur
ORD Library (2)
DRDCIM (2)
Spares (4)

Canada

DEPARTMENT OF NATIONAL DEFENCE

CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT & LAND)

DOR(J&L) RESEARCH NOTE RN 2000/21

INTRODUCTION OF THE PEX EXTENSION FOR X WINDOWS INTO THE RTX GRAPHICAL USER INTERFACE

by

Mr. John Green
COOP Student

Recommend by: _____
LCol M. Boisvert
RWGT

Approved by: _____
for R.G. Dickinson
DOR(J&L)

OTTAWA, ONTARIO NOVEMBER 2000

ABSTRACT

The PEX extension for X Windows provides an extensive and sometimes complex library of functions and utilities for rendering 3 dimensional graphics in the X Windows environment. This document describes the initial effort to integrate PEX into the RTX Graphical User Interface for the JANUS wargames system. The potential return for this effort is the reduction, simplification and possibly elimination of RTX code, with an overall increase in system performance. By replacing custom RTX code with PEX library functions, and using PEX's ability to store graphics objects on the display server, this first effort to introduce PEX indicates that these objectives could be met.

TABLE OF CONTENTS

PAGE

INTRODUCTION OF THE PEX EXTENSION FOR X WINDOWS INTO THE RTX GRAPHICAL USER INTERFACE

INTRODUCTION

1. JANUS is a multi-user graphical wargames program written in FORTRAN 77. To display graphics over a TCP/IP network, an application program referred to as RTX has been written to make use of the X Windows System. RTX is actually the code that provides the Graphical User Interface (GUI) for the JANUS program. RTX is written in the C programming language and uses the X11 function library to display graphics commands issued by JANUS. The current implementation of RTX consists of 31 modules with associated header files.

2. RTX presents the graphics to the user as a 2 dimensional map. The map contains graphics representing roads, urban and rural areas, and elevation contours. As well, RTX displays a variety of symbols to represent military objects on the map and provides the features for positioning and moving the symbols around the map. It also provides the user with panning and zooming abilities for closer examination of map details.

3. The objective of the work term was to investigate the introduction of PEX into RTX. PEX is a 3 dimensional extension of the X Windows System. With the current implementation using only X Windows, the JANUS / RTX client stores its graphics objects on the client side in C structures called segments. As new graphics objects are added, new memory is continually allocated for segments and graphics primitives, consuming more and more memory resources on the client. PEX allows graphics structures to be maintained on the server. By introducing the PEX extension to RTX, it is hoped that the server can assume more of the memory load by holding graphics objects on the PEX server. PEX also provides numerous utilities for editing and manipulating graphics objects. By using these utilities, much of the complexity of original RTX code

can be reduced and perhaps integrated into JANUS code, simplifying the system with an overall increase in rendering performance.

4. Much of the work term was spent learning the fundamentals of X Windows and PEX, and tracing through the RTX code in an effort to understand its execution logic. By using online tutorials, a PEX reference manual, and the Hewlett Packard integrated debugger, first steps were taken to introduce PEX structures into RTX and make use of PEX structures for storing and rendering graphical objects.

5. This is an ongoing project. Previous students have worked on the RTX code with attempts of introducing the PEX library. This report describes the current activities including a brief description of X Windows and the PEX extension as they related to the work term project.

THE X WINDOWS SYSTEM AND PEX

X Windows and the JANUS / RTX Client

6. The first part of the work term involved learning some of the fundamentals of X Windows programming and the PEX extension. The X Windows System is a network oriented windowing system where client applications can run on a different system other than that which supports the display device. This is normally the case with the JANUS wargame, although with the JANUS system, there is only one client, namely the JANUS / RTX combination. RTX accepts, translates and implements JANUS function commands for rendering. RTX uses the Xlib function library and makes calls to the X server to perform rendering operations. In the X Windows System, the program that supports the display is the server and the program that accesses the Xlib function library is the client.

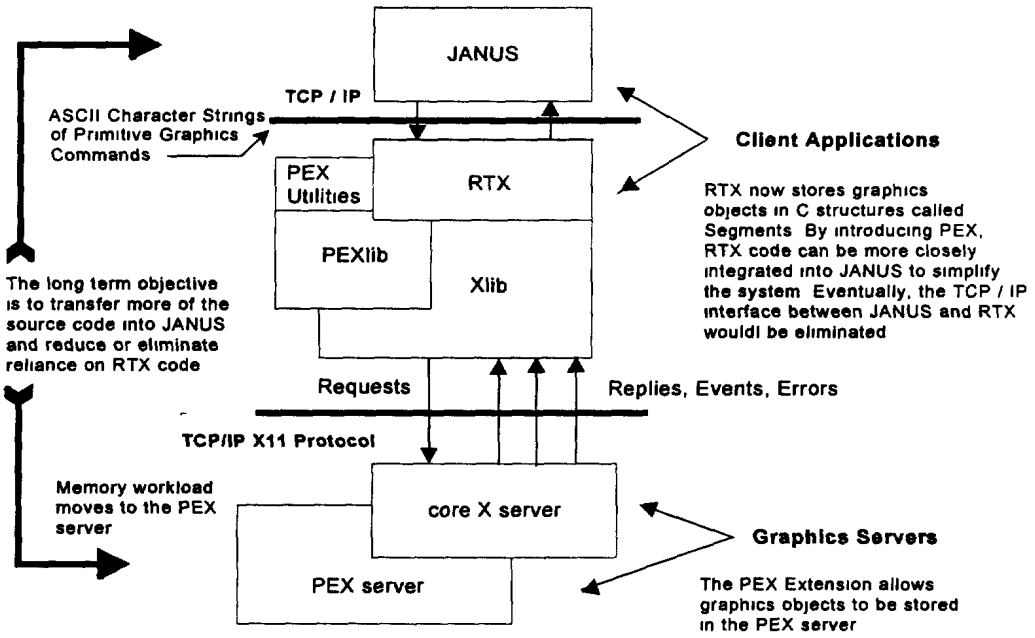

Figure 1 - X Windows / PEX Overview (Reference 1)

The current version of X Windows used by the JANUS system is Version 11 Release 6.

The PEX Extension

7. PEX is a library of functions for 3D rendering on the X Windows System. PEX is similar to X Windows in that the client application accesses the PEXlib function library and sends output commands to the PEX server. It is closely integrated with X Windows. Although the JANUS / RTX client is a 2D GUI, there is one significant feature of the 3D PEX extension that makes it attractive as an addition to RTX. This is the ability of PEX to create models of numerous graphics objects and store these objects on the server side.

8. The PEX extension has 3 rendering modes. Immediate mode, structure mode, and mixed mode that is a combination of immediate and structure mode (Reference 2):

a. **Immediate Mode**. Immediate mode transmits all output commands to the PEX server immediately whenever a change is made to the graphics object or model. This is also what happens with X Windows function calls made to the X server and is the current method of rendering used by RTX. In

immediate mode, no data is stored on the server. All graphics primitives and attributes are sent to the server each time a change is required, and the server renders the graphics on the display device.

b. **Structure Mode.** Structure mode is the mode of interest to the work term efforts. With this mode, graphics primitives, attributes, and transformation matrices all reside in structures and structures can be further held in models or structure networks. When held in structures, these output commands are referred to as structure elements. PEXlib provides extensive utilities for querying, editing and deleting structure elements held on the server. PEXlib also provides features for traversing through the structure networks. The server maintains structure data as static data and only those output commands necessary to modify or edit the structure or model are sent from the client.

c. **Mixed Mode.** This mode combines features of both immediate and structure mode. The geometry is held in a structure on the client and when time to render, the client sends the appropriate commands to send the structure to the server.

RTX and Graphics Segments

9. RTX provides the GUI for JANUS. It consists of some 31 modules and associated header files required for creating, storing, manipulating and rendering graphics commands issued by JANUS. It also provides the code for the initialization of the X server, creation of colormaps suitable for the given display devices, and code required for connecting and communicating with the JANUS process.

10. RTX currently stores graphics objects in a linked list of C structures called segments with each segment pointing to another structure containing an array of opcodes and operands describing graphics primitives. Each segment contains members providing attribute information such as the segment number, pivot point of the graphics object, scaling and rotation information, detectability information, and display priority information.

11. A much simplified segment structure is as follows:

```
struct Segment{
        int segnum;
        Segment *next;          /* the next segment in the linked list */
        int detectability;
        int pivot_x;
        int pivot_y;
          .
          .
          .
        Primlist *primlist;     /* pointer to the structure holding the primitives */
        };

struct Primlist{
        int size;               /* number of bytes held in primlist */
        int primbuf[1024];      /* the array of opcodes and operands */
        } primlist;
```

12. The JANUS process issues low level function calls to the RTX process via 'ct_ ' commands. These are sent in a series to build the desired object. JANUS has commands for lines, curves, rectangles and polygons used to build more complex structures. Additionally, JANUS has commands for setting graphics attributes such as line thickness, line color, polygon fill patterns and so on. RTX keeps a corresponding list of commands, one for each JANUS command.

13. For example, the command "ct_draw, 10, 20 " is a request from the JANUS process to the RTX process to draw a line from the current photon beam position to a point located at x = 10 and y = 20. RTX receives these commands as strings of ASCII characters. After taking the string and extracting the command "ct_draw", RTX parses through its own command list to find the corresponding opcode for this command. This opcode is then stored in the array of primitives followed by the operands 10 and 20.

14. After sending all the graphics primitives required to draw the object, the array of primitives will contain all the opcodes and operands necessary to render this graphics object. At some point, JANUS requests that the segment be closed. RTX then allocates new memory for the segment and copies all the data to the new segment. The original structure is used as a working structure and cleared for the next segment.

15. When it is time to render the object for this segment, the RTX code loops through the array of primitives identifying opcodes. Once identified, the array is passed by reference to the appropriate C function where the operands are collected and the appropriate Xlib function call such as XDrawLine(...) is made. The call is sent over the network to the X server and the graphics object is rendered immediately on the display.

16. For example, a sampling of function calls sent to create map contours are as follows:

```
ct_begin_segment, 22            /* begin a new segment */
ct_set_line_index, 1            /* line color attribute opcode 4 */
ct_set_text_index, 1            /* text color attribute opcode 5 */
ct_move, 8665, 0                /* move to position opcode 0 */
ct_draw, 8665, 1024             /* draw line opcode 1 */
ct_draw, 8618, 2048
ct_draw, 8192, 2818
        .
        .                       /* many additional ct_draw calls */
        .
ct_draw, 301056, 78315
ct_draw, 301620, 78848
ct_end_segment 22
```

17. RTX parses these commands, acquires an integer opcode for each command, and calls an associated C function to store the opcode and operands in the segment structure's primlist.

```
ct_draw_(position_x, position_y)
int *position_x;                /* draw to this x, y position */
int *position_y;

{
        if (new_segment_open || new_graphtext_char_open){
                prim_check();
                add_prim( Op_Draw );
                add_prim(*position_x);
                add_prim(*position_y);
        }

        current_beam_pos_x = *position_x;       /* update global variables */
        current_beam_pos_y = *position_y;
}
```

18. When the segment ends, new memory is allocated on the client, and the segment is copied to it. The primlist array will consist of opcodes and operands such as:

Primlist primlist[4, 1, 5, 1, 0, 8665, 0, 1, 8665, 1024, ..., 1, 301620, 78848]

19. When the object is to be rendered, RTX loops through the segment's primlist extracting opcodes and operands and sets up the appropriate Xlib function call to send to the X server for rendering. What is important to note in this sequence of operations is that the newly allocated memory for each new segment is held on the client where it remains. Any modification of this graphics object requires the entire segment be sent again to the X server.

PEX Structures

20. PEX structures hold Output Commands (OCs) which are graphics primitives and attributes. To render a graphics object stored in a structure, the structure is instanced with the appropiate OC. The structure may be instanced into another structure becoming part of a structure network or model, or it may be instanced directly to the PEX renderer for immediate rendering. PEXlib provides a variety of tools for editing, manipulating and modifying structure elements. By changing the attributes of a graphics object held in a structure, the object may be rendered over and over again with different scale, rotation, colors etc. Along with storing graphics on the server side, some additional advantages of introducing PEX are (Reference 3):

a. By storing Output Commands (OCs) in structures, the amount of network traffic will be reduced each time the graphics object is redrawn. With JANUS, the client and server are not on the same machine. With the original RTX code, redrawing the object requires that segment data be loaded into Xlib commands and the Xlib commands be resent from client to server. (Although the current RTX code makes use of pixmap buffering to reduce network traffic and speed up rendering).

b. A second reason to use structures is to group graphics objects into a model. This was what was done for some of the graphics used by RTX. In particular, the graphics comprising the contours, roads, and urban area

segments of the map terrain were held as structures and instanced into a single structure model for rendering.

INTRODUCING PEX INTO RTX

21. Introducing the PEX extension into the RTX code was the first major task of the work term project. Much of the information to do this was gathered from Hewlett Packard' s online PEX tutorial (Reference 4). According to the tutorial, there are 6 basic steps to using PEX:

a. Open a connection to an X server.
b. Initialize PEXlib.
c. Determine the best visual and colormap for use by PEX.
d. Create a window and map it.
e. Create a PEX Renderer.
f. Draw the picture when the window becomes exposed.

Initializing the PEX Server

22. Steps 1, 3, and 4 are steps common to all X Windows programs and were already accomplished by the RTX code. Steps 2, 5 and 6 were new requirements to the RTX code although an additional standard colormap was introduced for use by PEX. Much of the first half of the work term involved studying sample code and PEX reference material to learn how to do implement these steps. Five new functions were written for the RTX code to implement and setup PEX. The new code was placed in a separate module with the associated header.

```
void
initialize_PEX(display_name, display, win, ix)

char *display_name;     /* the name of the display device */
Display *display;       /* a pointer to the display device */
Window win;             /* the resource ID of the current X window */
int ix;                 /* index into RTX' s server structure */
```

23. This code initializes the connection to PEX. This is the first requirement for using PEXlib. After X Windows opens a connection to the display device, the pointer to the display device is used to initialize the PEX server. This is a prerequisite before any other PEXlib function calls can be made.

```
void
get_standard_colormap( display, vis_info, cmap_info)

Display *display;                /* pointer to the display device */
XVisualInfo *vis_info;           /* pointer to the display's visual information */
XStandardColormap *cmap_info;    /* a pointer to the new colormap structure */
```

24. Code was also written to obtain the appropriate standard colormap. This code was adapted from the O'Reilly sample programs provided with the on-line tutorial. Appropriate references were made to this source in the new code. The code takes a pointer to a display connection, along with information about a particular visual such as visual depth, which is the number of planes used to represent color in a window. The function returns a pointer to a *XStandardColormap* structure. This structure contains the resource ID of the colormap as well as information about red, green and blue pixel values in the colormap. Colormaps are tables where pixel values are used as indices to put colors on the screen.

```
void
set_stdcmap_approx( vis_info, cmap_info, capx_info )

XVisualInfo *vis_info;           /* pointer to the visual information structure */
XStandardColormap *cmap_info;    /* pointer to colormap information structure */
PEXColorApproxEntry *capx_info;/* pointer to the new color approx. structure */
```

25. Again, this code was adapted from the O'Reilly sample programs. Given a pointer to the *XVisualInfo* structure and a pointer to the display, the function sets up the fields in the *PEXColorApproxEntry* structure. This structure describes a color cube used by PEX to approximate colors in red, green, blue triplets. For example, (1,1,0) is yellow and (0,0,0) is black. White is (1,1,1). This structure is necessary for turning RGB triplets into pixel values.

```
void
setup_renderer(renderer, display, win, capx_info, attrs, cmap )

PEXRenderer *renderer;              /* pointer to the new renderer */
Display *display;                   /* pointer to the display device */
Window win;                         /* X window resource ID */
PEXColorApproxEntry *capx_info;/* pointer to the color approximation structure */
PEXRendererAttributes *attrs;       /* the renderer's attributes */
Colormap cmap;                      /* the colormap resource ID */
```

26. The PEX renderer is the central resource in PEX. It controls the process of rendering geometry into pixels (Reference 5). This is referred to as the rendering pipeline. The PEX renderer contains pointers to all the necessary resources required to actually draw graphics objects. This function takes a pointer to the display, a pointer to the *PEXColorApproxEntry* structure, a pointer to the *PEXRendererAttributes* structure and the resource ID's of the colormap and window. It returns a pointer to the resource ID of the newly created renderer.

27. The *PEXRendererAttributes* is a structure defining which renderer attributes are initialized for use. For example, the color lookup table is an indexed table of RGB floating point values in the range of 0 to 1. Because JANUS requests its desired color values from RTX for graphics objects in the terms an index, code was written to set up a color lookup table for the renderer. This involves first creating a lookup table, filling in the desired table entry structures, and sending the entries to the lookup table.

```
PEXLookupTable

init_color_LUT(display, cmap, win)
Display *display;       /* a pointer to the display */
Colormap cmap;          /* the resource ID in the Colormap */
Window win;             /* the resource ID of the X window */
```

Sample code for two of these functions can be found in the annex.

JANUS Graphics Segments as PEX Structures

28. To introduce PEX structures, the idea was to take the JANUS graphics commands and translate these into PEX OCs to be stored in structures as structure elements. This was what was done for the graphics making up the map terrain. The segments composing the map terrain where chosen as the first graphics objects to be held as PEX structures

because they are mostly stable, and change little after first being rendered into the window. The terrain is also the first map feature to be rendered after the X window opens. The map terrain is composed of three graphical objects:

a. A segment of filled polygon primitives describing urban and rural areas on the map.

b. A segment or segments of line primitives describing landscape contours.

c. A segment of filled polygons and lines describing roads and highways on the map.

29. The RTX code was modified so that JANUS requests are monitored for new segments , and when detecting the segment number for roads, contours, or urban / rural areas, PEX structures are created to store the graphics primitives. Using the contour example again, when detecting the command to begin segment 22, a new PEX structure is created:

```
PEXStructure contour_system = PEXCreateStructure( Display *display );
```

contour_system is now the Resource ID of the PEX structure. This ID is used for all further PEXlib commands dealing with this resource. Whether adding graphics attributes or graphic primitives to the new PEX structure, this resource ID is the identifier. Now that the structure exists, graphics primitives are added to the structure and held as structure elements. With the contour segment, as 'ct_draw' commands are received by RTX, the appropriate PEX OC is called and the line primitives are stored.

```
PEXPolyline2D (Display *display,
               PEXStructure contour_system,
               PEXOCStore,
               int num_points,
               PEXCoord2D *points)
```

30. *PEXOCStore* is the PEX request type that stores the graphics primitive *PEXCoord2D* in the structure *contour_system*.

31. The sequence of operations to create PEX structures is summarized:

a. When the desired segment number is detected, create and initialize the new PEX structure.

b. As graphics primitives and attributes are received by RTX from JANUS, store them as structure elements in the structure.

c. When JANUS sends the command to end the structure, instance the structure into a root structure to build a structure network.

d. Render the structure network as a model. Once rendered, the model is held as data on the server.

32. Several functions were introduced into the RTX code to implement these tasks. Currently, code has been developed to build PEX structures made of lines and polygons. As RTX executes, PEX structures are created to hold graphics primitives describing the urban/rural areas and roads segments. PEX structures now exist for the three graphics components making up the map terrain:

```
PEXStructure urban_system
PEXStructure contour_system
PEXStructure road_system
```

33. To build these into single structure, the individual structures are then instanced into a new structure network or model called *terrain*:

```
PEXStructure terrain = PEXCreateStructure( Display *display );

PEXExecuteStructure(Display *display,
                    PEXStructure terrain,
                    PEXOCStore,
                    PEXStructure urban_system);

PEXExecuteStructure(Display *display,
                    PEXStructure terrain,
                    PEXOCStore,
                    PEXStructure contour_system);
```

```
PEXExecuteStructure(Display *display,
                    PEXStructure terrain,
                    PEXOCStore,
                    PEXStructure road_system);

PEXRenderNetwork(Display *display,
                    Window window,
                    PEXRenderer renderer,
                    PEXStructure terrain);
```

34. The terrain structure is referred to root structure and the 3 others are leaf structures. This structure is now held on the server. At any time, the new structure can be edited and instanced to redraw the object. This ability to edit structures on the server is one of the distinguishing features of PEX (Reference 6). For example, the terrain structure can be rotated, scaled, and translated to be redrawn in a different area of the window as desired.

Panning and Zooming with Views

35. New code was also developed to handle panning and zooming into and around the terrain structure. The code was introduced to RTX to make use of PEX views. In PEX, viewing is the operation that determines what and how much of an object the viewer sees. Again, a starting point for this exercise was the HP on-line tutorial and the O'Reilly sample code. The RTX GUI provides features to the user for zooming into the map terrain for closer examination. By storing the terrain as a PEX structure, further features of the PEX extension were utilized to replace existing RTX code. PEX provides a rich set of features for viewing graphical objects. By assigning the new terrain structure to a view table, which is a renderer resource, zooming into map details becomes an efficient and simple exercise.

36. The process which takes geometry from the original JANUS coordinate system to a coordinate system used by the PEX renderer is known as the geometry pipeline.

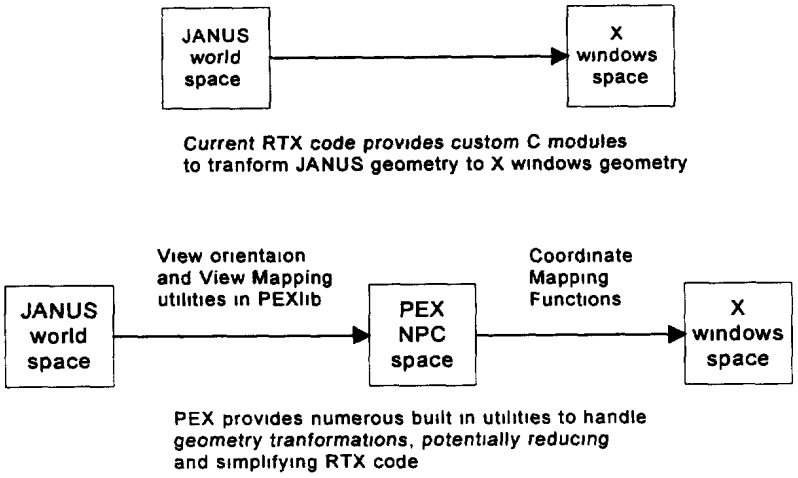

Figure 2 - PEX Geometry Pipeline

37. The JANUS coordinate system uses dimensions of 0 to 307,200 in both x and y-axis to describe the geometry of its graphics primitives. This is the world space. The current RTX code provides several custom modules for taking these coordinates and mapping them into the X window. PEX provides functions to do the same. In PEX, views are held in NPC, or Normalized Projection Coordinate Space, which is a cube from (0, 0, 0) to (1, 1, 1) and provides a common ground between world space and X Windows space. By using PEX utilities for managing views, much of the custom RTX code originally used for zooming can be bypassed.

38. The first are the view orientation utilities which create a matrix that defines the orientation of the model from the viewers perspective. Another set of utilities are the view mapping utilities which define what volume of World space is mapped to NPC space and actually rendered to the X window. To zoom into the terrain, JANUS sends functions to RTX to define new window coordinates to display. The new code takes these coordinates and uses them in a call to the PEX's view mapping matrix function. This creates the desired zoom view that is set into the renderer's view lookup table. Once set, a call is made to render the terrain structure according to the view table, and the desired view is displayed.

CONCLUSION

39. Much of the work term was spent learning the basics of the X Windows system and the PEX extension. As well, learning some of the execution logic of the RTX code was an essential activity before the PEX extension could be introduced. By doing these, first steps were taken to integrate PEX into RTX. The graphics objects comprising the map terrain were successfully stored as PEX structures in a simple model on the PEX server rather than as C structures on the RTX client. Additionally, PEX utilities for viewing graphics objects were used with some success, to make panning and zooming into the terrain model a relatively straightforward exercise and potentially reducing and simplifying RTX code. Development of new code using PEX could continue in the following areas:

a. **Expand the PEX model of the RTX GUI.** By using PEX' s tools and utilities, a much more extensive model of RTX graphics can be built. Along with the map terrain, the model could hold graphics for symbols such as tanks, buildings, and other features used in the war game, and now stored as C structures. This would require making these objects pickable, such that the user can modify and position these objects as desired. PEX provides the necessary utilities to create a structure network where these objects can be made pickable, and the structure network can be traversed to edit symbol's attributes as necessary. By storing the model on the PEX server, only the output commands needed to edit the structure need be sent over the network

b. **Make use of PEX colormap and transformation utilities.** RTX now provides numerous modules and functions for creating and adjusting a colormap to suit JANUS's color requirements. PEX provides several utilities for obtaining a colormap for use by the renderer. It should be investigated whether these utilities can be used to create the appropriate colormap and thus eliminate code now used in RTX.

(1) PEX provides functions for carrying out all necessary coordinate transformations needed to take JANUS coordinates to X Windows coordinates. Some of these where introduced to carry out panning

and zooming but more should be introduced to replace custom code from RTX.

c. **Examine the communication between the JANUS process and RTX process.** The current method of sending graphics function commands between JANUS and RTX involves the transmission of strings of ASCII characters from JANUS to RTX. Once received, RTX uses string functions to break the commands apart and parse them out to the appropriate Xlib graphics functions. This seems like somewhat of a crude way of communicating between the processes. It should be investigated whether a much more efficient binary communication protocol can be introduced into the code to improve performance.

REFERENCES

1. Jan "Yon" Hardenbergy, Building Applications with PEXlib (Englewood Cliffs, New Jersey: Prentice Hall, 1994) 3. Additional input provided by Mr. Pierre Ladouceur.

2. README file for the PEXlib Subset Mode Demo, provided by Hewlett-Packard and located in the directory /opt/graphics/PEX5/examples/hp/SubsetAAModeling/SubsetMode with a default installation.

3. Hardenbergh, 214.

4. Hewlett Packard Online PEX Tutorial, Taken from Chapter 3 of PEXLib Programming Manual, 3D Programming in X by Tom Gaskins (Cambridge, Masschusetts: O'Reilly and Associates, Inc. 1992)

5. Hardenbergh, 214

6. Hardenbergh, 219.

ANNEX A
TO DOR(J&L) RESEARCH NOTE RN 2000/21
NOVEMBER 2000

Sample code: void setup_renderer(renderer, display, win, capx_info, attrs, cmap)

```
/*****************************************************************************************/
* Initialize the PEX renderer attributes and create the renderer resource
*
* Given the colormap information, the color approximation information, the
* function returns a pointer to the new renderer resource
*
*****************************************************************************************/

void
setup_renderer(renderer, display, win, capx_info, attrs, cmap )

PEXRenderer *renderer,                          /* pointer to the new renderer */
Display *display,                               /* pointer to the display device */
Window win,                                     /* X window resource ID */
PEXColorApproxEntry *capx_info,                 /* pointer to the color approximation structure */
PEXRendererAttributes  *attrs;                  /* the renderer's attributes */
Colormap  cmap,                                 /* the colormap resource ID */
{
        unsigned long           mask = 0,
        PEXColorApproxEntry     capx_entry,

        /* Create a color approximation table and set the default */
        /* entry, entry 0, to the colormap approximation specified  */

        mask |= PEXRAColorApproxTable,
        attrs->color_approx_table = PEXCreateLookupTable( display, win, PEXLUTColorApprox ),
        PEXSetTableEntries( display, attrs->color_approx_table, 0, 1, PEXLUTColorApprox, capx_info ),

        /* Initialize the viewport */
        mask |= PEXRAViewport,
        attrs->viewport min x = 0, attrs->viewport min y = 0, attrs->viewport.min z = 0,
        attrs->viewport max x = 0, attrs->viewport max y = 0, attrs->viewport max z = 0,
        attrs->viewport use_drawable = 1,

        mask |= PEXRAViewTable,
        attrs->view_table =
        PEXCreateLookupTable( display, win, PEXLUTView ),

        /* Create a color table for the renderer  */
        mask |= PEXRAColorTable,
        attrs->color_table = init_color_LUT(display, cmap, win),

        /* Create the pick name sets  */

        mask |= PEXRAPickIncl | PEXRAPickExcl,
        attrs->pick_incl = PEXCreateNameSet( display ),
        attrs->pick_excl = PEXCreateNameSet( display ),

        /* Create the renderer */
        *renderer = PEXCreateRenderer( display, win, mask, attrs ),
}
```

Sample code: PEXLookupTable init_color_LUT(display, cmap, win)

```
/*****************************************************************************************
* Create a color lookup table for the PEX renderer and set its entries
* using the standard colormap
*****************************************************************************************/

PEXLookupTable
init_color_LUT(display, cmap, win)
Display *display,               /* a pointer to the display */
Colormap cmap,                  /* the resource ID in the Colormap */
Window win,                     /* the resource ID of the window*/

{
        PEXLookupTable color_lu_table,
        PEXColorEntry entries[256],
        XColor cmap_cells[256],                 /* colorcells used to setup renderer LU table */
        XColor *cell,
        int i,

        /*Create the lookup table resource */
        color_lu_table = PEXCreateLookupTable(display, win, PEXLUTColor),

        /* Fill cmap_cells with cmap's RGB entries */

        cell = &cmap_cells[0],
        for (i = 0 , i < 256, i++)
        {
                cell->pixel = i,
                cell->flags = DoRed | DoBlue | DoGreen,
                cell++,
        }
        XQueryColors(display, cmap, cmap_cells, 256),

        cell = &cmap_cells[0],
        for (i = 0 , i < 256, i++)
        {
                entries[i] type = PEXColorTypeRGB,
                entries[i] value rgb red = (float)cell->red / 0xffff,
                entries[i] value rgb green = (float)cell->green /0xffff,
                entries[i].value rgb blue = (float)cell->blue /0xffff,
                cell++,
        }
        /* Send the entries to the lookup table */

        PEXSetTableEntries(display, color_lu_table,
                0, 256, PEXLUTColor, (char *) &entries),

        return(color_lu_table),
}
```

UNCLASSIFIED
Security Classification of Form
(Highest Classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1 ORIGINATOR (the name and address of the organization preparing the document Organizations for whom the document was prepared e.g. Establishment Sponsoring a contractor's report, or tasking agency, are entered in Section 8)
Operational Research Division
Department of National Defence
Ottawa, Ontario K1A 0K2

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)
UNCLASSIFIED

3 TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title)
Introduction of the PEX Extension for X Windows into the RTX Graphical User Interface

4 AUTHORS (last name, first name, middle initial)
Green, John

5 DATE OF PUBLICATION (month Year of Publication of document)
November 2000

6a. NO OF PAGES (total containing information. Include Annexes, Appendices, etc.) 24

6b. NO OF REFS (total cited in document)
6

7 DESCRIPTIVE NOTES (the category of document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report e g interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)
Research Note

8 SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address).
DOR(J&L)

9a PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written Please specify whether project or grant.)
3550-3-2

9b. CONTRACT NO (if appropriate, the applicable number under which the document was written)

10a ORIGINATOR's document number (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)
DOR(J&L) Research Note RN 2000/21

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)
---

11 DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)
(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors· further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies, further distribution only as approved
( ) Distribution limited to defence departments, further distribution only as approved
( ) Other (please specify)

12 DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11) However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected)

UNCLASSIFIED
Security Classification of Form
(Highest Classification of Title, Abstract, Keywords)

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U) It is not necessary to include here abstracts in both official languages unless the test is bilingual).

The PEX extension for X Windows provides an extensive and sometimes complex library of functions and utilities for rendering 3 dimensional graphics in the X Windows environment. This document describes the initial effort to integrate PEX into the RTX Graphical User Interface for the JANUS wargames system. The potential return for this effort is the reduction, simplification and possibly elimination of RTX code, with an overall increase in system performance. By replacing custom RTX code with PEX library functions, and using PEX's ability to store graphics objects on the display server, this first effort to introduce PEX indicates that these objectives could be met.

14 KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e g Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified . If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

JANUS
RTX

Canada

#514903