Application Redirection: Hosting Windows Applications in 3D

Maarten van Dantzich,
George Robertson
Microsoft Research
One Microsoft Way
Redmond, WA 98052
maartenv@microsoft.com

Vadim Gorokhovsky
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052

ABSTRACT

We present *Application Redirection*, a novel architecture that lets unmodified Windows applications be hosted in a 3D virtual environment. The result is a platform for experimentation in 3D Information Visualization in which the user retains all familiar productivity tools. This paper describes the implementation of Application Redirection, using the Task Gallery to illustrate how it is used.

Keywords

Window Managers, 3D User Interfaces, 3D Window Managers, Information Visualization.

1. INTRODUCTION

Researchers exploring 3D visualizations currently choose between two approaches: create a 3D environment with new applications, or host 3D elements inside 2D applications. Neither is ideal: any new 3D environment will have few productivity tools available for it, while keeping 3D visualizations inside a 2D window limits the range of designs that can be pursued.

We have developed a novel way to bring existing Windows applications into a 3D environment. The result is a platform for experimentation in 3D Information Visualization in which the user retains all familiar productivity tools. We present *Application Redirection*, an architecture that lets unmodified Windows applications be hosted in a 3D virtual environment.

Other potential uses for Application Redirection include easy thumbnail generation for browse histories or document visualization, scalable user interfaces, and other ways to redisplay applications.

The Task Gallery [7] is a 3D Window Manager that uses Application Redirection to host applications by redirecting both their input (events) and output (visuals). This paper describes the implementation of Application Redirection, using the Task Gallery to illustrate how it is used.

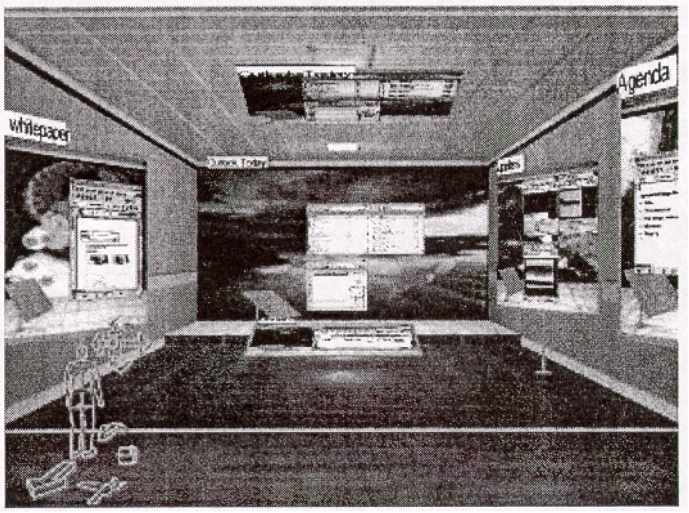

Figure 1. The Task Gallery.

2. RELATED WORK

Most 3D environments have required the implementation of new client applications. For example, the 3D Rooms system [5] was built as an information workspace that used 3D virtual environments to extend the ideas of Rooms [3]. 3D Rooms was not actually a window manager; abstract information visualizations replaced windows. Hence there was no need for a mechanism like Application Redirection.

Feiner et al. [2] modified an X server to put 2D windows into a 3D augmented reality. Regions of a virtual display surface were overlaid on the 3D scene view, possibly anchored to the camera or to objects in the scene. However, windows were always aligned with the film plane, and were not first-class 3D citizen. Our work differs in several ways: we can redirect individual windows, rather than an entire desktop. The windows become texture maps, and thus can be displayed under any 3D transformation. We have made the windows fully interactive by providing input redirection, which Feiner et al. do not address. We also address the performance issues Feiner mentions. Elmqvist et al. [1] have a more complete X Windows solution currently under development, using a modified version of the Xvfb virtual frame buffer.

NPIVM 99 Kansas City Mo USA

3. TASK GALLERY DESIGN

Figure 1 shows a view of the Task Gallery, a 3D window manager designed to provide direct support for task management and document comparison. A task is a collection of documents and applications organized around a particular user activity. User tasks appear as artwork hung on the walls of a virtual art gallery, with the selected task on a stage, which contains opened documents and running applications for that task. The user switches to a new task by clicking on it, which moves it to the stage. Viewing multiple windows within a task simultaneously is done with a button click, using automatic layout and movement in the 3D space to provide uniform and intuitive scaling.

To view other tasks, the user backs up to see more of the gallery, as in Figure 1. The gallery is composed of a sequence of rooms, with only one closed end; more rooms are revealed without limit as the user moves back. The user is allowed to place tasks wherever desired with a dragging movement, constrained to remain on walls, floor, or ceiling.

Users (especially non-gamers) tend to get lost in many systems that require them to navigate. We avoid this problem by keeping the space simple (a linear hallway), and by constraining the navigation. Thus, we provide a few simple controls rather than a general egocentric navigation mechanism: jump back, jump forward, and return to active task; these are presented as buttons surrounding a human figure overlaid on the 3D view. Our studies showed that users did not become disoriented in the 3D space when using these controls.

The active task workspace offers a few novel ways of rearranging windows belonging to that task. The user maintains overlapped windows in much the same way the current desktop metaphor does in the 'loose stack' of windows. Documents that are not in active use but desirable to have at hand can be stacked up on a podium that maintains an automatically tidied stack of windows. Selected windows are brought to a region closer to the user, so that they are more easily read; this corresponds to maximizing a window when the user wants to focus on a particular document. Multiple windows can be selected and are scaled automatically so that they appear side-by-side for easy comparison or cut and paste operations.

The Start Palette is a Data Mountain [6] for application icons, document thumbnails, and command buttons; it is used to create tasks and to populate them. The Start Palette is carried relative to the virtual camera and is accessed by a glancing operation [4].

4. IMPLEMENTATION ISSUES

The current implementation runs on PCs with a 400 MHz Intel Pentium II processor, 128 MB of RAM, and AGP support. We used the NVIDIA Riva TNT2 graphics accelerator with 32MB of memory and AGP texturing support. We achieve frame rates above 20 frames per second (fps).

While we intend the Task Gallery to become usable as a primary visual shell, at this early stage of development it runs as a full-screen application on top of the standard Windows 2000 shell. Applications started by the Task Gallery appear redirected into its environment.

All Task Gallery code was implemented in C++, using the Win32 and Direct3D APIs. The software architecture is divided into five major components as follows:

1. objects to simplify and abstract Win32 platform functionality like window creation, devices and events;
2. management of redirected windows;
3. a simple 3D renderer that abstracts the Direct3D interface and performs render state maintenance and texture management;
4. a library for scene graph management, animation, and basic graphical object types; and
5. the high-level objects that implement behavior and visual representations of Rooms, Tasks, and Windows.

4.1 Redirection

The key technical challenge in building a 3D window manager like the Task Gallery is to get existing applications to work in the 3D environment without changing or recompiling them. This requires both output and input redirection facilities in the operating system. Output redirection causes applications to render to off-screen bitmaps instead of the screen, provides access to those bitmaps so they can be used as textures in the 3D environment, and sends notification whenever an application has updated its visual display in any way. Input redirection causes mouse and keyboard events to be received by an application rather than the 3D environment's main window, but with mouse coordinates translated from 3D to 2D. The next two sections describes these facilities.

4.1.1 Output Redirection

The Window Manager in Windows 2000 provides a new top-level window style called Layered Window. Layered windows encompass two different concepts – layering, the ability of windows to exhibit sprite-like behavior, and redirection, the ability of the system to redirect the drawing of legacy windows into an off-screen bitmap. The sprite support for layered windows includes flicker-free animation, and transparent and non-rectangular windows with per-pixel alpha information. Client applications can make explicit use of Layered Windows with source code changes, but this does require recompilation.

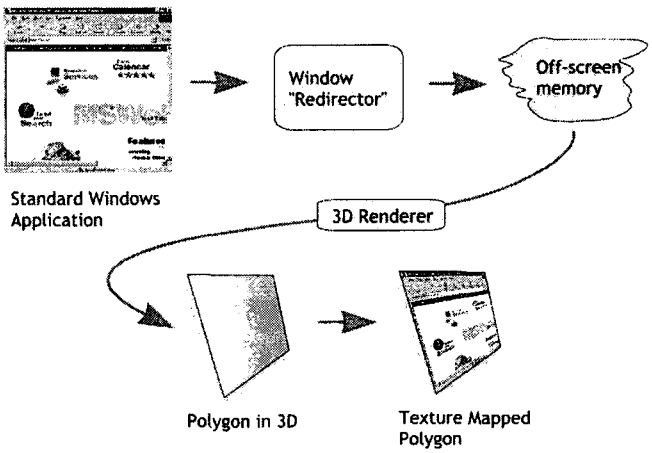

Figure 2 Output Redirection

Since Windows 2000 already supports painting redirection as an integral part of the layering support, the Window Manager was

modified to expose redirection as a separate feature via a "Redirected" style bit. The Redirected style bit can be set on a window at any time after its creation, enabling the redirection functionality for that window. The system removes the window from the desktop, creates an off-screen bitmap, and reroutes further drawing operations applied to the window into that bitmap. Since the redirected application is not aware of this change, the initiator of the redirection (called the redirection host) is responsible for propagating changes in the application's visible appearance to the screen. To enable this, the system fires a new event ("redirected paint") whenever the application has finished a visual update. By requesting a hook on this event, the redirection host obtains notifications of painting updates, including which window was updated and the affected region within the window.

In the Task Gallery, we handle this event in a lazy fashion: the update region is composed with the existing cached dirty region for a window, and the texture object associated with the window is marked dirty. When the next frame is rendered, we copy the accumulated dirty region from the window's off-screen bitmap to the texture surface.

Of course visual updates should happen as promptly as possible to ensure that the user receives good visual feedback. Currently, our system always re-renders the whole scene, as is common in 3D systems. This is not a problem since the frame rate remains above 20 fps.

The redirection host must also notice when a redirected application creates a new window, so that it will appear inside the 3D environment. This includes the creation of new document windows, dialog boxes, and even drop-down menus. When windows are destructed, the host should free resources such as the associated texture. To do this, our redirection host installs the hooks to receive the window show and hide notification events for each application thread that creates windows.

The host must make intelligent decisions about the positioning of such new windows: drop-down menus must be positioned very precisely in relation to a parent window, whereas a dialog that otherwise appeared in the center of the screen should now be positioned relative to another of the application's windows.

Finally, some handling of special cases is necessary. Some applications put up a splash screen, then fork off another process which creates the actual main window. We have addressed this on a case-by-case basis so far, but are implementing a module that will track process creation so that we can correctly track window ownership.

4.1.2 Input Redirection

The Task Gallery displays application windows using a texture map. However, we want the texture map to appear as if it is the live application, responding correctly to user input. Since the operating system does not recognize the textured polygon as a window object, some work has to be done to route input events to the proper destination. The steps needed to accomplish this are: ensure that mouse messages are delivered to redirected application windows where they belong; make the redirected application believe that the mouse cursor is in the right 2D location over a redirected window even though it is over the redirection host window; ensure no impact to proper system performance. We describe each of these in turn.

If nothing is done, the redirection host will receive all mouse messages, since in reality the user is interacting with the host's window. Thus, the host can examine each incoming event, decide which logical window it was meant for, and repost the event to the correct window's event queue. This has two disadvantages. First, each of these events will be posted twice. Since some event types (such as mouse motion) can occur very frequently, this can become a performance bottleneck. Second, it does not address the issue of the mouse being captured by redirected applications by calling SetCapture. Once the capture is set to the redirected window, the redirection host may stop receiving mouse messages. In addition, even if the redirection host was able to intercept them, the mouse messages distributed under the mouse capture may contain a point in screen coordinates as opposed to window coordinates without the redirection host being aware of it. In general, it would be extremely difficult for the redirection host to imitate correctly the behavior of the system function that distributes messages to applications.

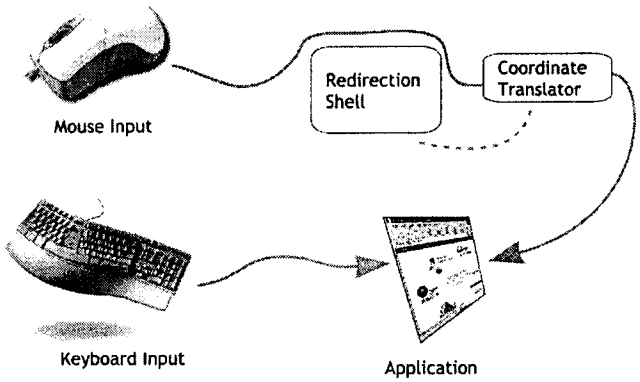

Figure 3 Input Redirection

Our system employs a different tactic. In current versions of the Windows operating system, the Window Manager is responsible for distributing device events from the system event queue to the appropriate window message queue. Keyboard messages go to the focus window of the application currently in the foreground. Mouse messages generally go to the window under the mouse cursor. However, one can intervene in this mapping by installing a low-level hit test hook. This allows the redirection host to change the mouse position of a hit tested point.

Thus, whenever the pointer is over the main window of the redirection host, we examine these hit test hook messages. If the pointer is on top of a pixel that is part of a redirected window's texture map in the 3D scene, we adjust the mouse coordinates and the target window handle in the hit test structure so that the pointer will appear to be in the proper location over the redirected window in the Windows 2D desktop coordinates. This causes the Window Manager to post the mouse message to the right application and with the desired coordinates. In turn, the application that the user wants to interact with will become the foreground application and update its visuals and behavior accordingly. Keyboard messages are now automatically sent to the redirected application. Thus, neither mouse nor keyboard messages actually reach the host's window event queue.

The mouse position must be adjusted in two situations: when the hit test result is adjusted, and when a redirected application

inspects the mouse position directly. Windows applications do this through the Win32 API call GetCursorPos. A new hook type was added to the Window Manager so that our redirection host can catch the occurrence of a call to GetCursorPos and forge the mouse coordinates as we do for the hit test.

Deciding if the pointer is over a pixel belonging to a redirected window is simple. By limiting the hit test to the list of geometry objects known to represent application windows, we can avoid walking the entire scene graph. Our 3D scene manager supports a standard 3D hit test using hierarchical bounding boxes which returns a 2D normalized coordinate on the face of the bounding box that was hit. We construct the "forged" 2D mouse coordinates by converting this normalized 2D coordinate into an appropriate pixel coordinate in the window's coordinate system.

Each low-level mouse message generated in the system requires inspection, and the 3D hit test we employ is non-trivial. Our current system limits the impact on system responsiveness by only forging mouse input to the active application, thus we only test against one bounding box, but we allow that window to be transformed in the scene arbitrarily: it does not have to be camera-aligned. In the future, we will keep a map of the screen locations of transformed 3D objects representing windows, so a simple point-in-polygon test will determine whether the mouse is over that window. By inverting and caching the 3D camera transform, the transformation to 2D forged coordinates becomes much cheaper than the ray-plane intersection we currently do against the bounding box.

Note that the redirection host never touches keyboard messages: these are sent to the foreground application by the operating system. This implies that our environment maintains the existing notions and policies of window activation and keyboard focus that Windows implements. Thus, a click on a window object causes the corresponding application to receive activation and focus; a click on the 3D scene background causes the redirection host to receive focus. When window manager functionality is accessed through keyboard commands, the user has to explicitly direct keyboard focus to the shell to obtain the desired effect. This is true in current window managers as well; usually special modifier keys are employed by the window manager as a workaround. In the Task Gallery, we avoid this by providing on-screen iconic controls for all window manager functionality, so that keyboard input is not needed.

4.1.3 Generality

The low-level components of this system were implemented by the USER and GDI groups of the Windows 2000 team (primarily one of the authors, VG). Some components use standard features of the operating system, such as the window hide/show messages. Others were added to a private build of Windows 2000 for this work: the ability to turn on the redirection for a window at runtime, the event generated when a window is updated by the application, the low level hit test hook, and the CBT hook event generated for calls to GetCursorPos. These new features work without recompiling the application because their implementation consists of changes to libraries which all applications load dynamically at runtime as well as the Windows 2000 driver that contains the Window Manager and the Graphics subsystem.

The details of this solution are specific to the Windows 2000 operating system. However, the components needed will be similar on other operating systems. We believe that similar changes are possible for any OS that uses the X Window System, as long as the X server runs on the same machine as the client applications and the window manager so that bitmap sharing is efficient (e.g., Feiner [2]).

X makes it easy to replace the window manager: many have been created by independent developers, including so-called "virtual desktops" which dynamically position certain windows off-screen. Many X servers can already cache bitmaps of windows for fast handling of window expose events, providing the low-level equivalent of redirected windows; these could be placed in shared memory accessible to the 3D window manager. There is already a placement negotiation mechanism between window managers and applications. X performs cursor hit tests in the server process based on a "window map"; this would be modified to involve the window manager. The source code for the X window system is widely available, making such modifications possible.

4.2 Texture Management

Another key problem is managing textures. Keeping a texture for each window and each task can quickly fill any texture memory available. In addition, our textures represent running applications and thus are updated often, requiring high texture download bandwidth. AGP graphics cards make it possible to texture from main memory with little performance penalty. This helps enormously, but careful management of textures is still required.

For example, in the Task Gallery, only the windows in the current task need have their textures updated when window change events occur. Those change events describe what region of the window changed, hence only that region of the texture need be changed (assuming the graphics card allows the modification of portions of textures).

The task snapshots are reduced in resolution (from the original 1024x768 frame buffer size to 256x256 pixels) to avoid overwhelming the texture map engine. However, we do not reduce the textures that represent application windows, in order to maintain as much visual detail and text legibility as possible.

5. CONCLUSION

Application Redirection is a novel approach to allowing existing, unmodified Windows applications to live within a 3D virtual environment. This allows 3D visualization users to take advantage of familiar productivity tools. The Task Gallery is a 3D window manager that illustrates how we can explore alternative novel user interfaces for application environments by building on top of the application redirection technology.

6. ACKNOWLEDGMENTS

Corneliu Lupu and Andrew Goossen of the Windows 2000 USER and GDI teams were instrumental in architecting appropriate low-level support for this work.

7. REFERENCES

[1] Elmqvist, N. et al, 3Dwm. Project information at http://www.medialab.chalmers.se/projects/3dwm/index.html

[2] Feiner, S., MacIntyre, B., Haupt, M., and Solomon, E., Windows on the world: 2D windows for 3D augmented reality, in Proceedings of ACM UIST '93 Symposium

on User Interface Software & Technology, November 1993, pp. 145-155.

[3] Henderson, A., Card, S. K., Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface, ACM Transactions on Graphics 5, 3, (1986), pp. 211-243.

[4] Pierce, J., Conway, M., van Dantzich, M., and Robertson, G., Toolspaces and glances: storing, accessing, and retrieving objects in 3D desktop applications, in Proceedings of Symposium on Interactive 3D Graphics, April 1999, pp. 163-168.

[5] Robertson, G., Card, S., and Mackinlay, J., Information visualization using 3D interactive animation, CACM, 36, 4, (1993), pp. 57-71.

[6] Robertson, G., Czerwinski, M., Larson, K., Robbins, D., Thiel, D. & van Dantzich, M.. Data Mountain: Using Spatial Memory for Document Management, in Proceedings of ACM UIST '98 Symposium on User Interface Software & Technology, November 1998, pp. 153-162.

[7] Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Risden, K., & Gorokhovsky, V. The Task Gallery: A 3D Window Manager, to be presented at ACM CHI 2000.