Translucent Windows in X

Keith Packard
XFree86 Core Team, SuSE Inc.
keithp@suse.com

Abstract

The X Window System allows multiple windows to occupy the same coordinates on the screen. The core protocol defines which portions of each window are visible and which are occluded by overlapping windows, but the overlapping windows are always completely opaque.

Various techniques can be used to simulate non-opaque windows in controlled environments. The Shape Extension can be used to make areas of the window transparent. A background of “None” can be used to inherit the contents of the screen in the region occupied by the window when it is first mapped. Where available, hardware overlays can be used which expose a transparent pixel value.

None of these techniques can be used for translucency in a general way; hardware overlays and the Shape Extension can only provide transparency and cannot blend the pixel colors together. A background of “None” cannot be used when the occluding windows are to be reconfigured or when the occluded region contents are expected to change.

The X Translucent Window Extension is described which solves the general translucency problem by assigning alpha values for pixels in occluding windows. These values are used to blend the occluding window contents with the occluded region for display. The details of managing translucent window hierarchies, reparenting translucent windows and X visual differences between blended pixels are discussed.

1 Translucency

Physically, translucent objects absorb some, but not all, of the light passing through them. The color of the object affects which wavelengths are most strongly absorbed. Visually, translucent objects appear to affect the color and brightness of objects beyond them.

The effect of light on a translucent object can be simulated by blending the color of the translucent object with that of objects beyond it. When dealing with computer images, translucency can be described as a mathematical operation on the color data of a collection of images. This mathematical composition of images was formally defined by Thomas Porter and Tom Duff in 1984 [PD84]. They define formulae which use color data in conjunction with a per-pixel opacity value called “alpha”. With these formulae many intuitive image manipulations can be performed.

1.1 Image Compositing Operators

Each of the operators defined by Porter and Duff operate independently on each of the color channels in each pixel. The equations are abbreviated to show the operation on a single channel of a single pixel.

A common compositing operation is to place one image over another. Transparent areas of the overlying image allow the underlying image to show through. Opaque areas hide the underlying image while translucent areas blend the two images together. By defining the “alpha” of a pixel as a number from 0 to 1 measuring its opacity, a simple equation combines two pixel colors together:

$$C_{result} = C_{under} \cdot (1 - \alpha_{over}) + C_{over} \cdot \alpha_{over}$$

Porter and Duff call this the “over” operator.

Another common operation is to mask an image with another; transparent areas in the mask are removed from the image while opaque areas of the mask leave the image visible.

$$C_{result} = \alpha_{mask} \cdot C_{image}$$

This is the “in” operator. They provide a complete compositing algebra including other operations; only these two are needed for this extension.

One important aspect of this model is that it creates a new image description which attaches another value “alpha” to each pixel. This value measures the “opacity” of the pixel and can be operated on by the rendering functions along with the color components.

1.2 Destination Alpha

Sometimes it is useful to create composite images which are themselves translucent, in other words, contain alpha values. This effect can be achieved by augmenting the operators with an operation which produces a composite alpha value along with the color values. For the “over” operator, the composite alpha value is defined as:

$$\alpha_{result} = \alpha_{under} \cdot (1 - \alpha_{result}) + \alpha_{over}$$

The “in” operator composite alpha value is:

$$\alpha_{result} = \alpha_{mask} \cdot \alpha_{image}$$

The resulting images can now be used in additional rendering operations.

1.3 Premultiplied Alpha

Visible in the above equations for computing the “over” operator is the asymmetry in the computation of alpha and the color components:

$$\alpha_{result} = \alpha_{under} \cdot (1 - \alpha_{over}) + \alpha_{over}$$

$$C_{result} = C_{under} \cdot (1 - \alpha_{over}) + C_{over} \cdot \alpha_{over}$$

This is “fixed” by respecifying the image data as being “premultiplied by alpha”. Each color component in the image is replaced by that component multiplied by the associated alpha value. Blinn [Bli94] notes that premultiplied images easily provide the correct results when run through long sequences of operations, while non-premultiplied images involve awkward computations.

2 Uses For Translucent Windows

As user interface design moves forward with increasing graphics performance, things formerly passed over as computationally intractable are now quite reasonable. Here are a couple of ideas how translucent windows could be used.

2.1 Window Management Effects

It is possible to simulate translucency during some window manipulation operations to provide additional feedback for the user. This can be implemented by capturing a static image of the window and the desktop, blending them, and updating the display.

This means that any changes that occur to the window contents during the operation cannot be reflected dynamically on the screen, limiting this to window movement operations. Moving the blending operations into the window system allows for the dynamic composition of application and underlying images.

Applications should not be required to cooperate to provide these effects. This requires an external control over window hierarchy translucence.

2.2 Transient Data

User interface elements which appear transiently over application windows such as menus and dialog boxes often occlude information useful in the operation of the transient action. By making the background of the transient window transparent and using translucency to highlight dialog elements, an effective user interface element can be usable and yet still allow interpretation of occluded application data.

While this can be effected by eliminating windowing for the transient elements and rendering them directly to the application window, the ability to continue to use window management and other windowing metaphors for these elements provides a strong incentive to incorporate these semantics into the window system.

2.3 Overlays and Annotation

Applications with complex image displays frequently present the ability for users to overlay associated information or annotate the image without affecting the underlying image data. These needs are satisfied today only with graphics hardware that supports overlays. But even where supported, the annotations are usually limited to fewer color planes than the main image and cannot incorporate translucence in the image, only opacity and transparency.

Again, these effects can be implemented by the application, however, the underlying graphics are often expensive to redisplay. Moving these operations into the window system provides the ability to use overlay hardware where available and gracefully fallback to software when necessary.

Both transient windows and annotations require per-pixel translucency that tracks rendering operations.

3 The X Rendering Extension

The X Window System [SG92] is a networked, extensible window system providing hierarchical windows. It was designed to operate on a wide variety of graphics hardware, from simple frame buffers to high-end graphics systems with overlays, underlays, accumulation buffers, z-buffers, double buffering, etc. Over the last 12 years, it has become the standard window system for Unix and Unix-like systems.

The X Rendering Extension [Pac00] provides a new rendering architecture for X applications including image compositing, sub-pixel positioned geometric primitives and application management of glyphs. The X Translucent Window Extension builds on the image composition ideas and mechanisms provided by the X Rendering Extension.

3.1 New Objects

The "PictFormat" object holds information needed to translate X pixel values into color and alpha data. For TrueColor visuals, the color data are extracted directly from the pixel while pseudo color visuals use a separate Colormap. The PictFormat references the appropriate Colormap in that case. It also indicates the portion of the pixel which contains alpha information (if any).

To encapsulate rendering state and color information, X Drawables (pixmaps and windows) are wrapped inside a new "Picture" object. An external pixmap Picture containing alpha data can be associated with the Picture. This external alpha data overrides any embedded alpha data.

3.2 Rendering Operators

The X Rendering Extension uses a modified version of the Plan 9 rendering primitive [Pik00] as the basis for image composition:

$$C_{result} = (C_{image} \text{ IN } C_{mask}) \text{ OP } C_{result}$$

In the Plan 9 window system, OP is always OVER. The extension allows any of the operators defined by Porter and Duff along with a special operator designed for drawing anti-aliased graphics adapted from OpenGL.

Using this basic rendering primitive, the extension defines geometric operations by specifying the construction of an implicit mask which is then used in the general primitive above. Anti-aliased graphics can be simulated by generating implicit masks with partial opacity along the edges.

3.3 Color Management

The core X protocol defines all rendering primitives in terms of pixel values. While this works when the rendering is done with boolean operations, it's color-based rendering must have a color interpretation for each pixel value.

The PictFormat object contains the information necessary to translate a pixel value into a color. The converted color can then be composited with other colors to generate the displayed color. Once a final color is computed, the extension converts it back to a pixel value using a fixed palette with optional dithering to improve color fidelity. Pseudo color displays use a fixed palette generated by the server for all images; the flexibility of a dynamic palette was discarded in favor of reduced colormap flashing and simpler code.

4 Windowing Semantics

X provides a hierarchical window system. Windows provide a view onto a document or scene. Windows are stacked over or under their peers and contain subwindows. The top of the window hierarchy is called the root and is distinguished from other windows by being contained in no window.

The visible area of a subwindow is confined to the visible area of the containing window. The visible area of a window is occluded by all subwindows. A window along with all subwindows stack together with respect to that windows peers.

Given the above semantics, the visible portion of a window can be found by computing the portion of the window within the visible portion of its including window and subtracting the areas of any subwindows and overlying peer windows. The visible portions of each window are combined to form the final displayed image. As the visible portions of each window form a partition of the root window area, each pixel on the screen belongs to the visible portion of precisely one window.

4.1 Transparent Windows

Graphics hardware frequently provides the ability to "color-key" one frame buffer over another frame buffer. This compositing is done in hardware, and allows either the underlay or overlay to be display, but not a combination of the two.

For hardware with this capability, X lists pairs of visuals for each screen, one as the underlay and another as the overlay. Transparent areas in windows created in the overlay visual show through contents of windows in the underlay. The transparent areas are typically specified with a special pixel value.

The semantics of this mechanism are meant to expose the underlying hardware abilities, rather than match the windowing model and can generate surprising results. One such surprise was that transparent pixels in the overlay visual would unconditionally show through to the nearest occluded window in the underlay, even if intervening windows existed in the overlay.

The X Shape Extension [Pac89] provides a mechanism for altering the visible region of a window. This affects graphical output as well as pointer input: areas outside of the shape do not receive pointer events. The Shape Extension can be used to implement partial window transparency, but the effect on input is usually not desirable. Additionally, the regular X semantics fail to ensure that the occluded window contents will be preserved by the X server for redisplay. A large class of applications fail to perform acceptably when their contents are damaged, making shaped windows unsuitable.

The final problem is that the Shape Extension is implemented by modifying the window clipping regions within the server. These regions are represented as lists of rectangles. Complex bitmap shapes generate clip lists of thousands of rectangles, slowing the server unacceptably.

4.2 Windowing as Compositing

An informal description of typical window system semantics is that of overlapping pieces of paper on a desktop. This characterizes the original intent as developed at Xerox, but seems far removed from the formal X semantics described above.

Reinterpreting those in terms of image composition provides a more transparent description.

A window is composed of an image and zero or more stacked subwindows. The visible image of a window is generated by composing the window with the visible image of its subwindows using the *over* operator. The alpha value of a window is 1 inside the shape of the window and 0 outside.

5 Translucent Windowing Semantics

Existing systems provide two interpretations for translucency. A reasonable system will provide mechanisms for both and also allow hardware acceleration when possible.

Systems based on hardware overlays provide a special pixel value that exposes data in the underlying window. These embed transparency in the pixel value itself. Each pixel can be either opaque or transparent, but not translucent.

Systems designed to animate window operations provide an external opacity value that controls the blending of a window to the desktop. The data within the window needn't contain opacity information, rather that is applied by the external window management agent to affect the display of the resulting image.

6 Prototype

To provide a framework for exploring window compositing, a simple prototype was constructed that allows for arbitrary compositing between windows.

As seen above, windowing can be described in terms of compositing images in layers. The displayed image of a window is formed by compositing the window image data along with the displayed images of each inferior window.

This suggests a relatively straightforward, if somewhat inefficient, implementation of windowing. Instead of providing window layering by clipping each rendering operation to a single shared image, window layering can be implemented by compositing separately rendered images.

6.1 Prototype Implementation

The image data for each window is kept in an off-screen image buffer. The complete displayed image is formed by recursively compositing these images together.

A separate displayed image buffer is used to render the composite image of the window and subwindows. The window image data are copied to that displayed image buffer. Finally, the displayed image for each inferior is generated and composited to the displayed image buffer. The root window uses the frame buffer itself for the displayed image.

As all rendering operations now occur off screen, the X server must update the displayed image for the root window whenever visible changes occur. The prototype keeps a single region which encloses any damage. Each rendering command updates that region. When the server is about to wait for additional X requests, it recurses through the window hierarchy updating the damaged areas of each displayed image. Finally, the damaged region is emptied.

6.2 Prototype Results

The initial prototype provides an alternate implementation of X windowing. By itself, that is not terribly useful. To demonstrate the capabilities of the architecture, the server was modified to mask each “override redirect” window with a constant alpha value of 2/3. This makes most menus appear translucent.

The prototype is minimally functional. Performance is poor, enough for a demonstration but not for real applications. Nonetheless, it is interesting to see the effect of translucency on real applications and gauge the usability of various user interface ideas. For example, making menus entirely translucent leads to readability problems. The text should probably be opaque and outlined in a contrasting color while the background of the menu remains translucent. The Phillips TiVo, a Linux-based video storage device, displays text in this manner.

The prototype can be extended to support the Translucent Window Extension; adding semantics for different compositing operators is quite easy once the entire contents of every window is available.

7 Improving the Design

While the prototype demonstrates an easy intuitive architecture for window compositing, it uses memory for window image data which is not used to generate the final display. It also recomposites window images at each level of the hierarchy, making deep window trees perform poorly. The prototype has been useful, but architectural changes are needed for a production system.

The final design should be equivalent to the existing X windowing system in the absence of translucency.

7.1 Opaque Windows

Windows with a constant alpha value of 1 (those without an alpha channel or mask) can be directly rendered to the enclosing window’s image buffer. This is a generalization of the standard X rendering model in which all windows share the frame buffer for image data. Similar rules apply here: the enclosing window and any occluded windows must clip rendering out of that area. When compositing the displayed image, any occluded areas from other windows must not touch the occluding pixels.

7.2 Occluded Window Regions

Sections of windows occluded by opaque windows are not needed to generate the final displayed image on the screen and so need not be contained in any image. Existing X clipping operations can be used to modify rendering operations in this case.

For the root window, this will allow direct rendering to the frame buffer, entirely avoiding the cost of compositing windows while ensuring acceleration of rendering operations with any display hardware.

7.3 Translucent Subwindows

The prototype stores the entire window image off screen so that the image can be composited with subwindows. Instead, only the portion of a window covered by translucent windows need be stored in a separate buffer with the remainder rendered directly to the displayed image buffer.

This gives each window two regions, one containing the area rendered directly to the displayed image buffer and a second containing the area covered by translucent subwindows which must be rendered to an off-screen image buffer.

7.4 Multiple Frame Buffers

The prototype works only for a homogeneous display; all windows must be true color at the same depth. To extend this for hardware with overlays, the implementation must allow for windows of different depths and visual classes.

For opaque windows, providing separate displayed image buffers where needed in the hierarchy is sufficient. Translucent windows must be blended together to be displayed. For windows not using a true color visual, the pixel values must be converted to color values, blended together then converted back to pixel values and stored in the displayed image buffer of an appropriate format. The semantics for this conversion are described in detail by the Rendering Extension.

8 X Translucent Window Extension

The X Translucent Window Extension exposes semantics for window translucency to applications, allowing them to manipulate the composition of window data to the screen. The Translucent Window Extension uses the Rendering Extension compositing primitive with a fixed OVER operator.

$$C_{result} = (C_{image} \text{ IN } C_{mask}) \text{ OVER } C_{result}$$

This single operator combines the masking of the "in" operator with the blending of the "over" operator. The Plan 9 window system uses this primitive for all graphics operations. Blinn suggests that for image composition, the OVER operator is sufficient for nearly every operation. This extended primitive incorporates the ability to render geometric objects, text and images with external alpha channels with a single simple operation.

8.1 Embedded Alpha Values

To allow applications to control opacity while rendering, pixel values must map to opacity values. Overlay hardware frequently uses special pixel values, but is usually limited to either transparent or opaque pixels. A more general solution associates a deeper alpha value with each pixel allowing it to be rendered along with the pixel values.

The Rendering Extension describes pixel formats including alpha; those resulting alpha values may be directly used as window opacity values when that is appropriate. For visuals without embedded alpha values, an external pixmap will contain the alpha values. For windows without embedded or separate alpha values, the extension uses a constant alpha value of 1.

8.2 External Opacity Control

This operations is frequently used to take an existing window and blend it over the screen. Menus, cursors and dialogs are examples where access to underlying information is desired even when the dialog may occlude that information. The essential requirement is for a separate alpha channel that can cause the window contents to go from opaque to transparent without changing the contents of the window. The "mask" operand in the Plan 9 primitive provides such control.

The Rendering Extension allows that mask to be "tiled" over the operation, repeating the mask in both directions to cover the area. By creating a 1x1 mask, the resulting single value controls the blending of the entire window.

9 Directions

There are several lines of work related to this extension. While the X Rendering Extension has a preliminary specification, there remains significant implementation to be completed which may affect that specification.

The server windowing infrastructure needs to be modified to support translucent windows. There are several other potential enhancements to the X Window System which would be able to take advantage of these architectural changes. Among them are an improved save-under system, a mechanism for changing the hardware accelerated visuals on each screen, changes to the backing store system and translucent multi-color cursors.

Finally, the Translucent Window Extension protocol must be formally defined and implemented. With the architectural infrastructure in place, this should be a relatively small effort.

10 Conclusion

Image compositing forms the basis of many modern rendering systems, from PostScript and PDF to GL and the Quartz windowing system. Extending the X windowing semantics to allow for window-level compositing provides a powerful new tool for application development.

Acknowledgments

I thank SuSE for encouraging me to work full time on XFree86. I'd also like to thank the ALS organizers, Usenix and VA Linux for supporting the second XFree86 Hothouse.

References

[Bli94] Jim Blinn. Compositing theory. *IEEE Computer Graphics and Applications*, September 1994. Republished in [Bli98].

[Bli98] Jim Blinn. *Jim Blinn's Corner: Dirty Pixels*. Morgan Kaufmann, 1998.

[Pac89] Keith Packard. Shape Extension Protocol, Version 1.0. X consortium standard, X Version 11 Release 6.4, 1989.

[Pac00] Keith Packard. A New Rendering Model for X. In *FREENIX Track, 2000 Usenix Annual Technical Conference*, pages 279–284, San Diego, CA, June 2000. USENIX.

[PD84] Thomas Porter and Tom Duff. Compositing Digital Images. *Computer Graphics*, 18(3):253–259, July 1984.

[Pik00] Rob Pike. *draw - screen graphics*. Bell Laboratories, 2000. Plan 9 Manual Page Entry.

[SG92] Robert W. Scheifler and James Gettys. *X Window System*. Digital Press, third edition, 1992.