Programming with InterViews

Mark Linton
Silicon Graphics
2011 N. Shoreline Blvd.
P.O. Box 7311
Mountain View, CA 94039-7311
linton@sgi.com

Outline

Overview

Key features

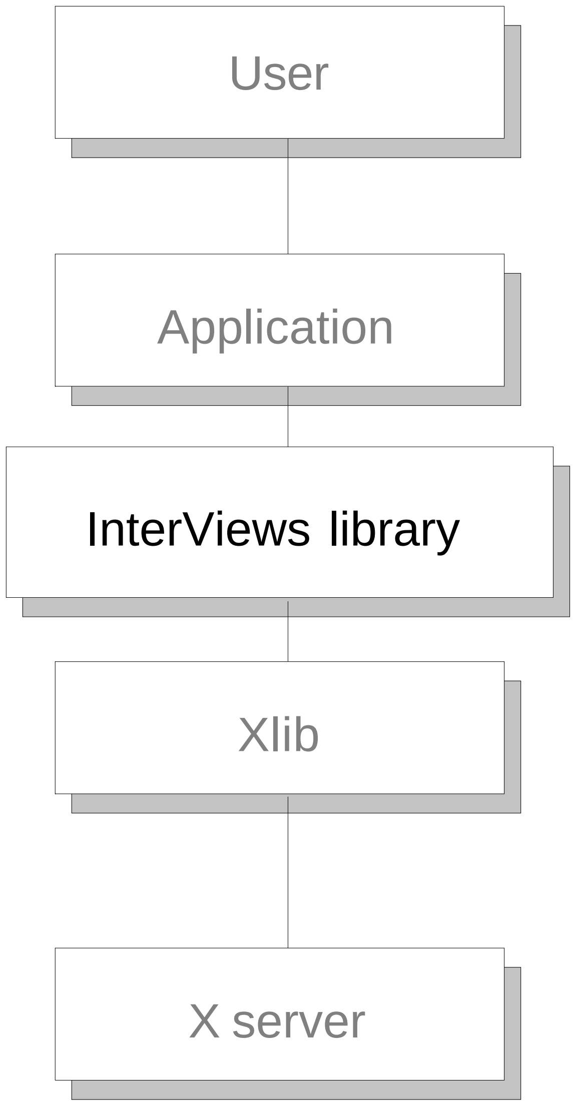
Differences from other toolkits

Portability

Multi-vendor support

Availability

High-level toolkit



InterViews = Interactive **Views**

Interactive objects

- Buttons, menus, scrollbars
- Chooser, dialogs, editors
- Multiple views

Graphics objects

- Immediate and structure mode
- Resolution-independence (printing)
- Transformations
- Color, font objects

Layout objects

- Sophisticated formatting in toolkit

Graphical editing framework

- Connectivity
- Undo/redo

Natural C++ API

Differences from other toolkits

Uniform object model

- Glyphs: lightweight, shareable
- Both widgets and structured graphics

Graphics

- Transformations (including fonts)
- Resolution-independence (printing)
- Direct color
- Structured graphics

Layout mechanisms

- TeX boxes and glue, discretionaries

Unidraw framework

- Connectivity, dataflow, undo/redo

Native C++

- Efficient, object-oriented language

Portability

Runs on all X/Unix platforms

(SGI, HP, Sun, DEC, IBM, ...)

Compatible with most C++ compilers

AT&T 2.0, 3.0, DEC, ...

g++: contact Mike Stump, (mrs@csun.edu)

X and OS dependencies isolated

Planning port to Windows/NT

Multi-vendor support

Support open industry standard
- SGI, Sun, HP, Fujitsu, ...

X Consortium working group

Compatible with OMG object model

Integrate best of InterViews, ATK, Xt

Portable across X, Windows, ...

Optional multimedia support

Availability

Unrestricted copyright (just like X)

Anonymous ftp to sgi.com
- graphics/interviews/3.1.tar.Z

Thousands of users world-wide

Commercial versions
- Quest ObjectViews, HP InterViews Plus

A simple application

Source code for “hi mom!”

Basic types, classes

Building the application

```
#include <IV-look/kit.h>
#include <InterViews/background.h>
#include <InterViews/session.h>
#include <InterViews/window.h>

int main(int argc, char** argv) {
   Session* session = new Session(
      "Himom",argc,argv
   );
   WidgetKit& kit = *WidgetKit::instance();
   return session->run_window(
      new ApplicationWindow(
         new Background(
            kit.label("himom!"),kit.background()
         )
      )
   );
}
```

Notation

```
typedef float Coord;
```

Default units are printer's points
Relative to fonts (typically 75/72)

```
typedef unsigned int boolean;
static const unsigned false = 0;
static const unsigned true = 1;

typedef unsigned int DimensionName;
enum{
   Dimension_X,Dimension_Y,Dimension_Z,
   Dimension_Undefined
};

#include <InterViews/enter-scope.h>
```

Define name to **iv***name*

No public or protected data members!

Basic classes

Session – main loop coordinator

WidgetKit – create buttons et al.

LayoutKit – create layout objects

Glyph – an object that draws

Canvas – a place to draw

Window – a canvas on a display

Display – logical input/output devices

Style – set of <name,value> attributes

Include directories

Intrinsics: <InterViews/*class*.h>

Look+feel: <IV-look/*class*.h>

X-dependent: <IV-X11/*class*.h>

OS-dependent: <OS/*interface*.h>

Dispatching: <Dispatch/*class*.h>

Imakefiles

```
#ifdef  InObjectCodeDir

OBJS = main.o

APP_CCDEFINES=
APP_CCINCLUDES =
APP_CCLDFLAGS =
APP_CCLDLIBS =

Use_libInterViews()
ComplexProgramTarget(himom)

MakeObjectFromSrc(main)

#else

MakeInObjectCodeDir()

#endif
```

Building the application

Default is object files in subdirectory

ivmkmf generates *Makefile*

make Makefiles generates *Makefile*
in subdirectory (SGI)

make depend computes dependencies

make builds *a.out*

make install puts *a.out* in installed bin

LayoutKit

High-level layout specification

Describe format, not position

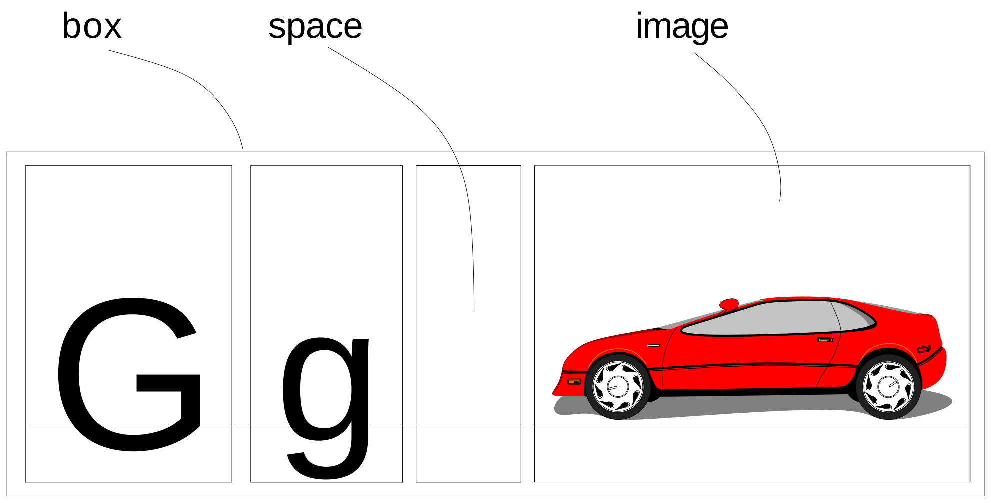
Use document formatting power:
  TeX boxes and glue

Glyphs define “natural size”

Glyphs may “stretch” or “shrink”

Glyphs may define an “alignment”
  Location of origin relative to size

Think of glyphs as characters



Requisition – what a glyph wants

- Requirement per dimension
  - Natural, stretch, shrink, alignment
- Returned by Glyph::request

Allocation – what a glyph gets

- Allotment per dimension
  - Origin, span, alignment
- Passed to Glyph::draw

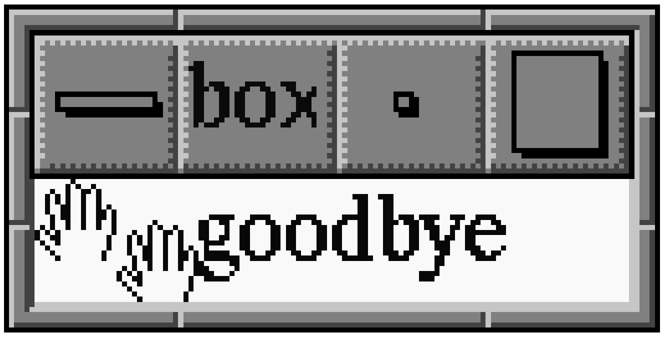
Box Example 1

```
// start with hi mom code
const Font* f = kit.font();
const Color* fg = kit.foreground();
const LayoutKit& layout = *LayoutKit::instance();
return session->run_window(
   new ApplicationWindow(
      new Background(
         layout.hbox(
            new Character(’g’, f, fg),
            new Character(’o’, f, fg),
            new Character(’o’, f, fg),
            new Character(’d’, f, fg),
            new Character(’b’, f, fg),
            new Character(’y’, f, fg),
            new Character(’e’, f, fg)
         ),
         kit.background()
      )
   )
);
```

Box Example 2: Add a stencil (twice)

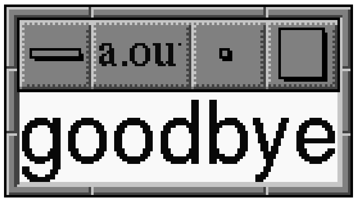
```
#include<InterViews/Bitmaps/hand.bm>
  . . .
  layout.hbox(
    new Stencil(
      new Bitmap(
        hand_bits, hand_width, hand_height
      ),
      fg
    ),
    new Stencil(
      new Bitmap(
        hand_bits, hand_width, hand_height,
        hand_x_hot, hand_y_hot
      ),
      fg
    ),
    . . .
```

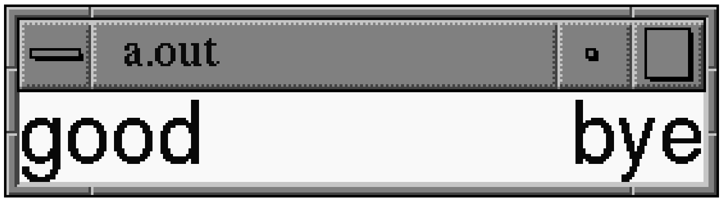
Boxes align origins



Boxes and glue

```
layout.hbox(
   kit.label("good"),
   layout.hglue(),
   kit.label("bye")
)
```





WidgetKit

Coordinating a common look+feel

Motif, OpenLook

Buttons

Push, toggle, radio

Menus

Menubars, toggle items, radio items

Scrollbars, sliders

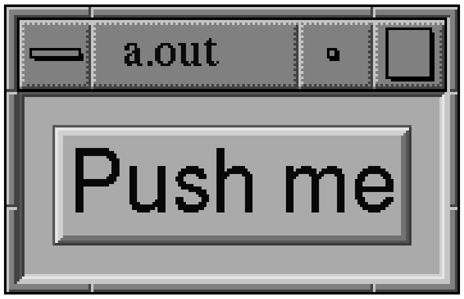
Defining an action callback

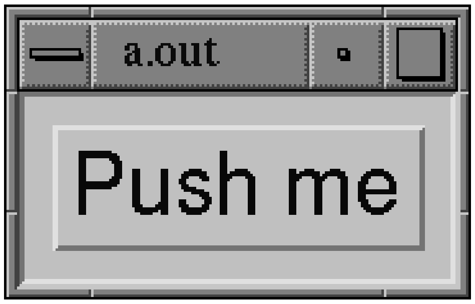
```
class App {
public:
   void msg();
};

declareActionCallback(App)
implementActionCallback(App)

void App::msg() {
   printf("himom!\n");
}

App* a = new App;
   ...
... new ActionCallback(App)(a, &App::msg) ...
```

Creating a push button

```
int main(int argc, char** argv) {
   Session* session = new Session(
      "Himom",argc,argv
   );
   WidgetKit& kit = *WidgetKit::instance();
   const LayoutKit& layout = *LayoutKit::instance();
   App* a = new App;
   return session->run_window(
      new ApplicationWindow(
         kit.inset_frame(
            layout.margin(
               kit.push_button(
                  "Pushme",
                  new ActionCallback(App)(a, &App::msg)
               ), 10.0
            )
         )
      )
   );
}
```

Button looks





Using a different button look

```
Raster* rast = TIFFRaster::load(argv[1]);
if (rast == nil) {
   ... error message ...
}

return session->run_window(
   new ApplicationWindow(
      kit.inset_frame(
         layout.margin(
            kit.push_button(
               new Image(rast),
               new ActionCallback(App)(a, &App::msg)
            ), 10.0
         )
      )
   )
);
```

Check boxes and radio buttons

```
kit.check_box(
   "Checkme",
   new ActionCallback(App)(a, &App:msg)
)

TelltaleGroup* group = new TelltaleGroup;

layout.vbox(
   kit.radio_button(group, "One", nil),
   kit.radio_button(group, "Two", nil)
)
```

Menus

List of items: <glyph, state, action, submenu>

Menu alignment defines submenu position

```
void Menu::append_item(MenuItem*)
```

WidgetKit operations

```
Menu* menubar()
Menu* pulldown()

MenuItem* menubar_item(Glyph*)
MenuItem* menu_item(Glyph*)
MenuItem* check_menu_item(Glyph*)
MenuItem* radio_menu_item(TelltaleGroup*,Glyph*)
MenuItem* menu_item_separator()
```

Menu example

```
struct CmdInfo;

class App {
public:
   void open(), save(), quit();
   void cut(), copy(), paste();
   void black(), red(), green, blue();

   Menu*menubar(
      CmdInfo*, WidgetKit&, const LayoutKit&
   );
private:
   Menu* pulldown(
      CmdInfo*, int, WidgetKit&, const LayoutKit&
   );
};

declareActionCallback(App)
implementActionCallback(App)
```

Menu example (cont’d)

```
struct CmdInfo {
   const char* str;
   ActionMemberFunction(App)* func;
   CmdInfo* submenu;
   int options;
};

CmdInfo filemenu[] = {
   { "Open", &App::open },
   { "Save", &App::save },
   { "", nil },
   { "Quit", &App::quit },
   { nil }
};
 ...
CmdInfo bar[] = {
   { "File", nil, filemenu, 0 },
   { "Edit", nil, editmenu, 1 },
   { "Color", nil, colormenu, 2 },
   { nil }
};
```

Menu example (cont’d)

```
void App::open() { printf("open"); }
void App::save() { printf("save"); }
void App::quit() { Session::instance()->quit(); }

Menu*App::menubar(
   CmdInfo* info, WidgetKit& kit,
   const LayoutKit& layout
) {
   Menu* m = kit.menubar();
   for (CmdInfo* i = info; i->str != nil; i++) {
      MenuItem* mi = kit.menubar_item(kit.label(i->str));
      mi->menu(
         pulldown(i->submenu, i->options, kit, layout)
      );
      m->append_item(mi);
   }
   return m;
};
```

Menu example (cont’d)

```
Menu* App::pulldown(
   CmdInfo* info, int opt, WidgetKit& k,
   const LayoutKit& layout
) {
   Menu* m = k.pulldown();
   TelltaleGroup* group = nil;
   for (CmdInfo* i = info; i->str != nil; i++) {
      if (i->str[0] == ’\0’) {
         m->append_item(k.menu_item_separator());
      } else {
         MenuItem* mi;
         // create the item
         if (i->func == nil && i->submenu != nil) {
            mi->menu(
               pulldown(i->submenu, i->options, k, layout)
            );
         } else {
            mi->action(
               new ActionCallback(App)(this, i->func)
            );
         }
         m->append_item(mi);
      }
```

Menu example (cont’d)

```
// creating the item
Glyph* g = layout.r_margin(
   k.label(i->str), 0.0, fil, 0.0
);
switch (opt) {
case 1:
   mi=k.check_menu_item(g);
   break;
case 2:
   if (group == nil) {
      group = new TelltaleGroup;
   }
   mi = k.radio_menu_item(group, g);
   break;
default:
   mi=ki.menu_item(g);
   break;
}
```

Scrolling

Observable and Observer

Adjustable

Bounded value example

Observable and Observer

```
class Observable {
public:
   virtual void attach(Observer*);
   virtual void detach(Observer*);
   virtual void notify();
};

class Observer {
public:
   virtual void update(Observable*);
   void disconnect(Observable*);
};
```

Mix-in classes

Adjustable

Object that can be scrolled/zoomed

One observable per dimension

Access current placement

```
Coord lower(DimensionName) const
Coord upper(DimensionName) const
Coord length(DimensionName) const
Coord cur_lower(DimensionName) const
Coord cur_upper(DimensionName) const
Coord cur_length(DimensionName) const
```

Requesting an adjustment

```
virtual void scroll_forward(DimensionName)
virtual void scroll_backward(DimensionName)
virtual void page_forward(DimensionName)
virtual void page_backward(DimensionName)
virtual void scroll_to(DimensionName, Coord lower)
virtual void scale_to(DimensionName, float visible)
virtual void zoom_to(float magnification)
```

Bounded value example

```
class BoundedValue : public Adjustable {
public:
   BoundedValue(Coord lower, Coord upper);
   virtual ~BoundedValue();

   virtual void lower_bound(Coord);
   virtual void upper_bound(Coord);
   virtual void current_value(Coord);
   virtual void scroll_incr(Coord);
   virtual void page_incr(Coord);

   virtual Coord lower(DimensionName) const;
   virtual Coord upper(DimensionName) const;
   virtual Coord length(DimensionName) const;
   virtual Coord cur_lower(DimensionName) const;
   virtual Coord cur_upper(DimensionName) const;
   virtual Coord cur_length(DimensionName) const;

   virtual void scroll_to(DimensionName, Coord lower);
   virtual void scroll_forward(DimensionName);
   virtual void scroll_backward(DimensionName);
   virtual void page_forward(DimensionName);
   virtual void page_backward(DimensionName);
```

Bounded value example (cont’d)

```
private:
   Coord curvalue_, lower_, span_;
   Coord scroll_incr_, page_incr_;
};

BoundedValue::BoundedValue(Coord a, Coord b) {
   lower_ = a;
   span_ = b - a;
   scroll_incr_ = span_ * 0.04;
   page_incr_ = span_ * 0.4;
   curvalue_ = (a + b) * 0.5;
}

BoundedValue::~BoundedValue() { }
```

Bounded value example (cont'd)

```
void BoundedValue::current_value(Coord value) {
   curvalue_ = value;
   constrain(Dimension_X,curvalue_);
   notify(Dimension_X);
   notify(Dimension_Y);
}

#define access_function(name,value) \
Coord BoundedValue::name(DimensionName) const {\
   return value;\
}

access_function(lower,lower_)
access_function(upper,lower_ + span_)
access_function(length,span_)
access_function(cur_lower,curvalue_)
access_function(cur_upper,curvalue_)
access_function(cur_length,0)
```

Bounded value example (cont'd)

```
void BoundedValue::scroll_to(
   DimensionName d, Coord position
) {
   Coord p = position;
   constrain(d, p);
   if (p != curvalue_) {
      curvalue_ = p;
      notify(Dimension_X);
      notify(Dimension_Y);
   }
}

#define scroll_function(name,expr) \
void BoundedValue::name(DimensionName d) { \
   scroll_to(d, curvalue_ + expr); \
}

scroll_function(scroll_forward,+scroll_incr_)
scroll_function(scroll_backward,-scroll_incr_)
scroll_function(page_forward,+page_incr_)
scroll_function(page_backward,-page_incr)
```

Bounded value example (cont’d)

```
class App : public Observer {
public:
   App(Adjustable*, TelltaleState*);
   virtual ~App();

   void print_value();

   virtual void update(Observable*);
   virtual void disconnect(Observable*);
private:
   Adjustable* adj_;
   TelltaleState* continuous_;
};

App::App(Adjustable* a, TelltaleState* s) {
   adj_ = a;
   a->attach(Dimension_X, this);
   continuous_ = s;
}
```

Bounded value example (cont’d)

```
App::~App() {
   if (adj_ != nil) {
      adj_->detach(Dimension_X, this);
   }
}

void App::print_value() {
   printf("%.5f\n",adj_->cur_lower(Dimension_X));
}

void App::update(Observable*) {
   if (continuous_->test(TelltaleState::is_chosen)) {
      print_value();
   }
}

void App::disconnect(Observable*) {
   adj_ = nil;
}
```

Bounded value example (cont'd)

```
Button* cb = kit.check_box("Continuous", nil);
BoundedValue* b = new BoundedValue(0.0, 100.0);
App* a = new App(b, cb->state());
b->current_value(50.0);
b->scroll_incr(5.0);
b->page_incr(20.0);
```

Rendering

Glyph operations:
allocate, *draw*, *print*, *undraw*

Canvas and Printer

Screen update

Glyph operations

```
void allocate(
   Canvas*, const Allocation&, Extension&
);
void draw(Canvas*, const Allocation&) const;
void print(Printer*, const Allocation&) const;
void pick(
   Canvas*, const Allocation&, int depth, Hit&
);
void undraw() const;
```

Allocation – given area for layout

Extension – rendering area

- Must be resolution-dependent
- Used for causing update, short-circuiting traversals

Usage

All rendering is done in *draw*

Printer-specific output may be done in *print*

(by default, Glyph::print calls draw)

Extension is computed in *allocate*

(must also compute component allocations)

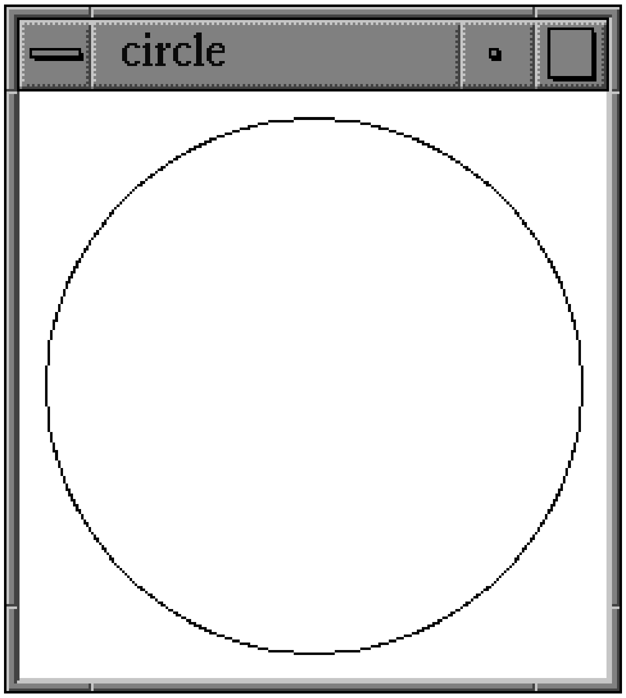
Cached information freed in *undraw*

In the future, draw will compute extension as side effect, so allocate will become optional.

Example: Circle

```
class Circle : public Glyph {
public:
   Circle(Coord radius, const Color*, const Brush*);
   virtual ~Circle();

   virtual void request(Requisition&) const;
   virtual void allocate(
      Canvas*, const Allocation&, Extension&
   );
   virtual void draw(Canvas*, const Allocation&) const;
private:
   Coord radius_;
   const Color* color_;
   const Brush* brush_;
};
```

Circle output



Circle example (cont'd)

```
Circle::Circle(
   Coord r, const Color* c, const Brush* b
) {
   radius_ = r;
   // use reference counting for (potentially)
   // shared objects
   Resource::ref(c);
   color_ = c;
   Resource::ref(b);
   brush_ = b;
}

Circle::~Circle() {
   Resource::unref(color_);
   Resource::unref(brush_);
}
```

Circle example (cont'd)

```
// Natural size is bounding box of given diameter.
// Stretchability and shrinkability are zero.
// Alignment is to put center at glyph origin.

void Circle::request(Requisition& req) const {
   Coord d = radius_ + radius_;
   Requirement rx(d, 0, 0, 0.5);
   Requirement ry(d, 0, 0, 0.5);
   req.require(Dimension_X,rx);
   req.require(Dimension_Y,ry);
}

// Simple assumption: we draw in entire bounding box.
void Circle::allocate(
   Canvas* c, const Allocation& a, Extension& ext
) {
   ext.merge(c, a);
}

// Can't show Circle::draw until we cover Canvas class
```

Canvas and Printer

Canvas draws to screen

Printer subclass generates PostScript

PostScript-like path/stroke/fill

Transformations and clipping

Damage area

Canvas drawing operations

```
void new_path()
void move_to(Coord x, Coord y)
void line_to(Coord x, Coord y)
void curve_to(
   Coord x, Coord y, Coord x1, Coord y1,
   Coord x2, Coord y2
)
void close_path()

void stroke(const Color*, const Brush*)
void fill(const Color*)
```

Canvas drawing operations (cont'd)

```
void line(
   Coord x1, Coord y1, Coord x2, Coord y2,
   const Color*, const Brush*
)
void rect(
   Coord x1, Coord y1, Coord x2, Coord y2,
   const Brush*
)
void fill_rect(
   Coord x1, Coord y1, Coord x2, Coord y2,
   const Color*
)
```

Canvas drawing operations (cont'd)

```
void character(
   const Font*, long code, Coord width,
   const Color*, Coord x, Coord y
)

void stencil(
   const Bitmap*, const Color*, Coord x, Coord y
)

void image(const Raster*, Coord x, Coord y)
```

Drawing a circle

```
void Circle::draw(
   Canvas* c, const Allocation& a
) const {
   const Coord r = radius_, x = a.x(), y = a.y();
   const Coord p0 = 1.00000000*r, p1 = 0.89657547*r;
   const Coord p2 = 0.70710678*r, p3 = 0.51763809*r;
   const Coord p4 = 0.26794919*r;
   c->new_path();
   c->move_to(x+p0,  y);
   c->curve_to(x+p2, y+p2, x+p0, y+p4, x+p1, y+p3);
   c->curve_to(x, y+p0, x+p3, y+p1, x+p4, y+p0);
   c->curve_to(x-p2,  y+p2,  x-p4,  y+p0,  x-p3,  y+p1);
   c->curve_to(x-p0,  y,  x-p1,  y+p3,  x-p0,  y+p4);
   c->curve_to(x-p2,   y-p2,   x-p0,   y-p4,   x-p1,   y-p3);
   c->curve_to(x,   y-p0,   x-p3,   y-p1,   x-p4,   y-p0);
   c->curve_to(x+p2,  y-p2,  x+p4,  y-p0,  x+p3,  y-p1);
   c->curve_to(x+p0,  y,  x+p1,  y-p3,  x+p0,  y-p4);
   c->close_path();
   c->stroke(color_,  brush_);
}
```

Transformations and clipping

```
void transform(const Transformer&)
void push_transform()
void pop_transform()

// Use current path
void clip()
void push_clipping()
void pop_clipping()
void clip_rect(
   Coord x1, Coord y1, Coord x2, Coord y2
)
```

Canvas damage

```
void damage(
  Coord left, Coord bottom, Coord right, Coord top
)

boolean damaged(
  Coord left, Coord bottom, Coord right, Coord top
)
```

Screen update

Damage is accumulated on canvas

Associated window added to repair list

No input pending => repair windows

Draw root glyph for window

Pruning with damage checks

Render to “back buffer” (pixmap)

Move repaired area to front

Flexibility

Full/partial updates as appropriate

Double-buffering

Non-rectangular objects

Overlays, transparencies

Unifies structured graphics, UI objects

Colors

Current support for RGB
  Other color models in the future

Automatically maps to pixels

Can specify alpha value (transparency)
  Partially implemented

For monoplane systems, can specify
  xor as color operator

Can specify X visual to control
  color mapping somewhat

A simple document previewer

Compositions

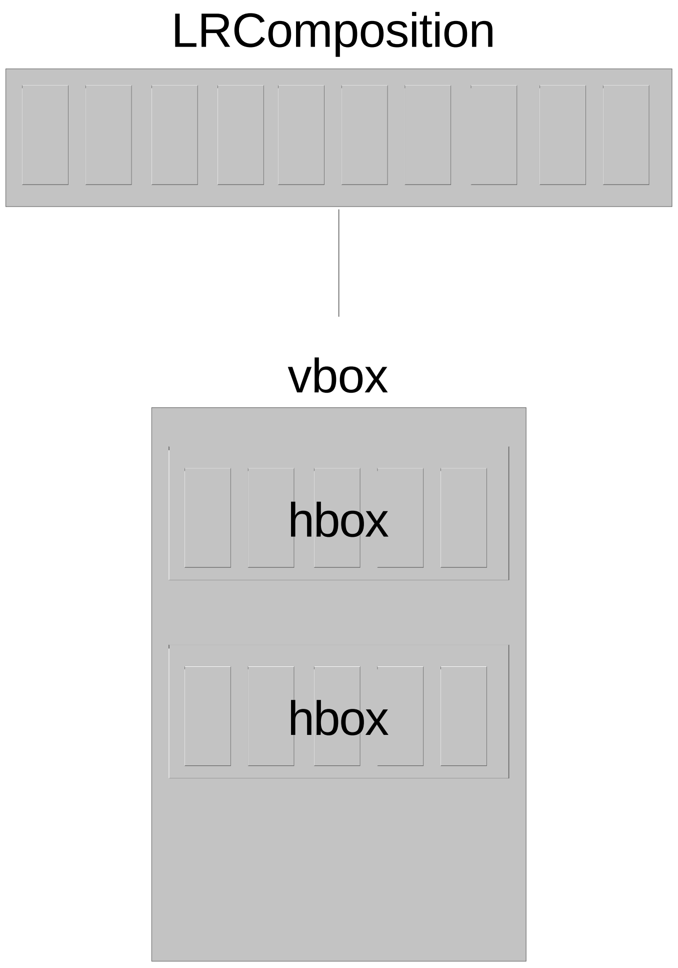
Discretionaries

Compositors

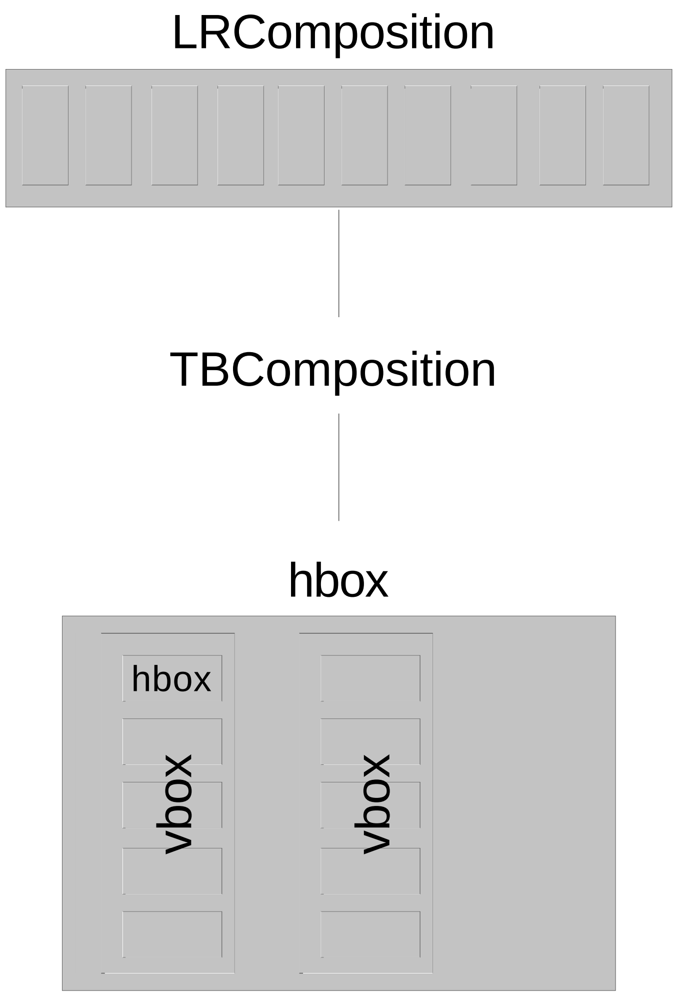
Operating system support

Previewer source

Instance structure



Nestingcompositions



What happens at a break?

- Character space becomes zero width
- Hyphen becomes visible
- Paragraph separator is zero

Different glyph appearance!

Specify appearance before, during, after

Discretionary

```
// constants defined in <InterViews/compositor.h>
const int PenaltyBad = 10000;
const int PenaltyGood = -10000;

layout.discretionary(
   int penalty, Glyph* no_break,
   Glyph* before, Glyph* at_break, Glyph* after
);
```

Simple document viewer

```
class DocumentView : public MonoGlyph {
public:
   DocumentView(
      InputFile*, WidgetKit&, const LayoutKit&
   );
   virtual ~DocumentView();

   virtual Adjustable* adjustable() const;
private:
   Composition* page_;
   ScrollBox*  box_;
   Glyph* begin_par_;
   Glyph* end_par_;
   Glyph* begin_line_;
   Glyph* word_space_;
   Glyph* interline_;
   Glyph* vfil_glue_;

   void add(
      const String&, WidgetKit&, const LayoutKit&
   );
};
```

Operating system classes

Directory – sorted array of filenames

File – read/mmap input file

Host – access hostname

List – parameterized list of objects

Math – min, max, abs, round, equal

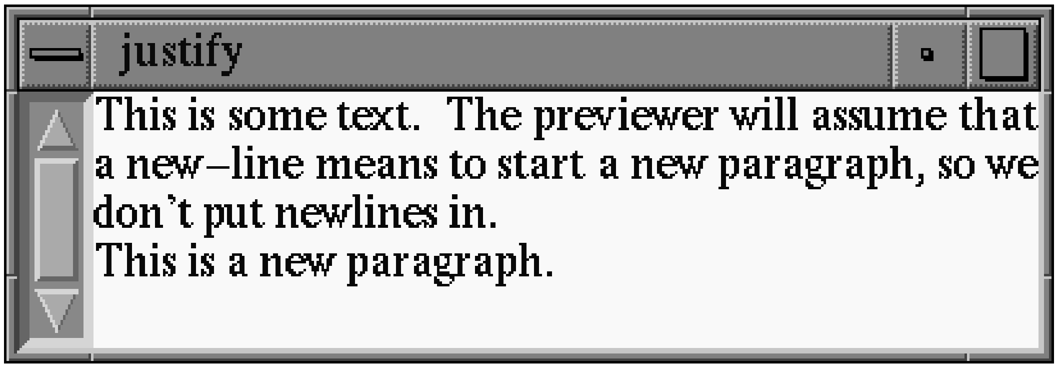
Memory – copy/zero memory

Operating system classes (cont'd)

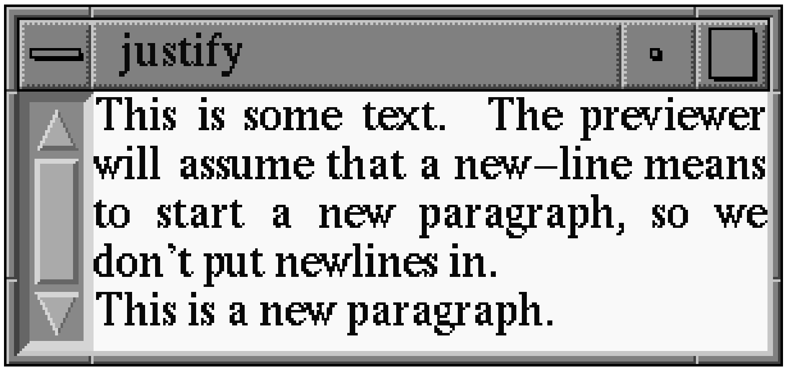
String – character strings
- CopyString – copy data
- NullTerminatedString – guaranteed
- UniqueString – fast comparison

Table – associative store

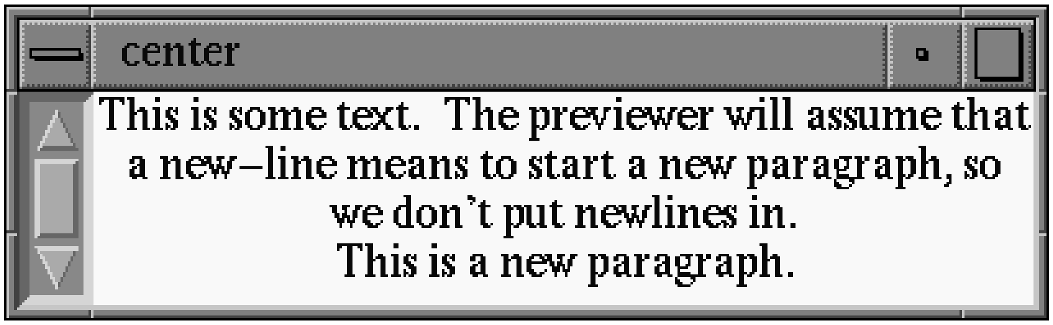
Document viewer



This is some text. The previewer will assume that a new–line means to start a new paragraph, so we don't put newlines in.
This is a new paragraph.



This is some text. The previewer will assume that a new–line means to start a new paragraph, so we don't put newlines in.
This is a new paragraph.



This is some text. The previewer will assume that a new–line means to start a new paragraph, so we don't put newlines in.
This is a new paragraph.

Document viewer (cont’d)

```
int main(int argc, char** argv) {
   Session* session = new Session(
      "Text", argc, argv, options, props
   );
   if (argc != 2) {
      fprintf(stderr, "Usage: %s file\n", argv[0]);
      exit(1);
   }
   WidgetKit& kit = *WidgetKit::instance();
   const LayoutKit& layout = *LayoutKit::instance();
   InputFile* file = InputFile::open(argv[1]);
   if (file == nil) {
      fprintf(stderr, "can’t open %s\n", argv[1]);
      exit(1);
   }
   DocumentView* view = new DocumentView(
      file, kit, layout
   );
```

Document viewer (cont’d)

```
    return session->run_window(
        new ApplicationWindow(
            layout.hbox(
                kit.inset_frame(
                    kit.vscroll_bar(view->adjustable())
                ),
                new Background(
                    layout.variable_span(
                        layout.natural_span(
                            layout.vcenter(view, 1.0), 4*72.0, 6*72.0
                        )
                    ),
                    kit.background()
                )
            )
        )
    );
}
```

Document viewer (cont’d)

```
DocumentView::DocumentView(
   InputFile* file,
   WidgetKit& kit, const LayoutKit& layout
) {
   const Font* f = kit.font();
   const Color* fg = kit.foreground();
   word_space_ = layout.spaces(2, 0.5, f, fg);
   interline_ = layout.vglue();
   vfil_glue_ = layout.vglue();

   String v("justify");
   kit.style()->find_attribute("alignment", v);
   if (v == "left") {
      begin_line_ = layout.vstrut(0);
      end_line_ = layout.strut(f, 0, fil, 0);
      begin_par_ = layout.vstrut(0);
      end_par_ = layout.strut(f, 0, fil, 0);
```

Document viewer (cont’d)

```
   box_ = new TBScrollBox;
   page_ = new LRComposition(
      box_, new TeXCompositor(10), nil, 6*72.0,
      fil, fil, file->length()
   );
   page_->append(begin_par_);
   const char* data;
   for (;;) {
      int len = file->read(data);
      if (len <= 0) {
         break;
      }
      add(String(data, len), kit, layout);
   }
   page_->append(vfil_glue_);
   page_->repair();
   body(page_);
}
```

Document viewer (cont’d)

```
void DocumentView::add(
   const String& data,
   WidgetKit& kit, const LayoutKit& layout
) {
   const char* p = data.string();
   const char* end = p + data.length();
   const Font* f = kit.font();
   const Color* fg = kit.foreground();
   Glyph* g[256];
   for (int i = 0; i < 256; i++) {
      g[i] = new Character(i, f, fg);
   }
```

Document viewer (cont’d)

```
Resource::unref(g[’\n’]);
g[’\n’] = layout.discretionary(
   PenaltyGood, end_par_, end_par_,
   layout.discretionary(
      0, interline_, vfil_glue_, nil, nil
   ),
   begin_par_
);

Resource::unref(g[’’]);
g[’ ’] = layout.discretionary(
   0, word_space_, end_line_,
   layout.discretionary(
      0, interline_, vfil_glue_, nil, nil
   ),
   begin_line_
);

for (; p < end; p++) {
   page_->append(g[*p]);
}
```

Conclusions

Simple, cheap, shareable objects
(glyph ~ character)

Sophisticated layout
TeX boxes and glue, discretionaries

Common widgets
WidgetKit, Style

Resolution-independence
Canvas,Printer

Structured graphics
Transformations,images

Dynamic behavior modification
Monoglyph – pass operations to body

What to do next

Obtain a copy of InterViews 3.1

Build/install InterViews

Read/scan the reference manual

Write a simple application

Use DebugGlyph to find problems

Readcomp.windows.interviews