Tutorial Outline

- *Introduction (20%)*
- *Object system (20%)*
- *Resolution-independence (15%)*
- *Graphical embedding (40%)*
- *Interoperability with other toolkits (5%)*
- *Script interpreter (5%)*
- *Conclusions*

Introduction

- *What is Fresco?*
- *What new technology is in Fresco?*
- *What GUI(s) will Fresco provide?*
- *How does Fresco relate to*
  Xlib? Xt? Motif? InterViews? ...?
- *Will Fresco run on*
  MS-Windows? MacOS?
- *What is the schedule?*

What is Fresco?

- User interface system
- Broad spectrum architecture
  - Graphics <–> Widgets <-> Embedded apps
- API covers Xlib, Xt functionality
  - + Structured graphics
  - + Application embedding
- X Consortium standards effort
- Specification, sample implementation

What is the new technology?

- Standard object model
  - Natural C++ usage
- Distributed objects
- MT-aware (supports concurrency)
- Resolution-independence
- Graphical embedding

Standard object model

- *OMG's CORBA*
- *IDL for defining object interfaces*
- *Choice of implementation languages*
- *Object references instead of pointers*
- *Address space/network transparency*

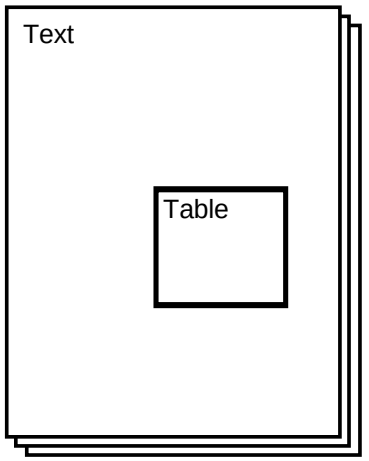
Resolution independence

- *Floating-point coordinates*
- *Stencil/paint drawing operations*
- *Printing (generation of PostScript)*
- *Device-independent graphics state*

Graphical embedding

- *Natural blend of structured graphics and application embedding*
- *Embedded objects may be*
  - *Arbitrary shape (with holes)*
  - *Graphically transformed*
  - *Drawn in 3D*
- *Advance over*
  - *ATK, NextStep, OLE 2.0, OpenDoc*

Embedding example



Why graphical embedding?

- *Allow natural, 3D appearance*
- *Composition in 2D or 3D scenes:*
  - *Editing rotated text in drawing*
  - *Touching up image in virtual gallery*
  - *Setting 3D clock in virtual room*
  - *City buildings in fly-through browser*
- *Simpler, broader architecture*

How does it work?

- *From structured graphics:*
  - *Damage-based update traversal*
  - *Shared state during traversal*
- *From application embedding:*
  - *Input handling, focus, translations*
  - *Menubar and/or palette sharing*
- *From both:*
  - *Attribute propagation*
  - *Transcription*

Distributed embedding

- *Fresco viewers can be remote objects "inside" a window*
- *Applications become objects*
- *User composes objects*
- *Objects may be implemented in different address spaces or on different machines*

What GUI(s) will Fresco provide?

- *Fresco architecture can support multiple GUIs*
- *Sample widgets have Motif L&F*
- *Future widgets will probably be Motif and/or Chicago*

How does Fresco relate to

- Xlib
  - Replaces all but low-level functionality
  - Adds resolution-independence
  - Sample implementation uses Xlib
- Xt
  - Replaces most if not all functionality
  - Adds structured graphics, embedding
  - Provides gateways for interoperability
    - Fresco in XtViewer
    - XtWidget in Fresco

How does Fresco relate to

- Motif
  - Allows re-implementation of widgets for graphical embedding
  - Interoperability at Xt level
- InterViews
  - Replaces object model
  - Evolves glyphs, structured graphics
  - Adds embedding

Will Fresco run on

- *Any X platform – Yes*
- *Win32 (Chicago, NT) – Yes (soon)*
- *Win32S (16 bit GDI) – Probably not*
- *MacOS – Probably not*
- *DOS - No way*

What is the schedule?

- *Fresco94.0 (May) on X11R6*
  - *Structured graphics, ixx, dish, fdraw*
- *Fresco94.1 (Oct)*
  - *Menus, dialogs, field editor*
  - *Dynamic loading*
  - *Native Win32*
- *Fresco95.0 (Jan) Embedding*
- *Fresco95.1 (Jul) Media editing*

Status

- Core defines about 25 interfaces
- Library implementation in C++
  - libFresco (27k lines)
    - stubs (10k)
    - impls (12k)
    - Ox (5k)
  - X11/libfdisplay (9k)
  - Win32/libfdisplay (7k)
- Separate redisplay thread

Status (cont'd)

- Kits
  - dialogs
  - figures
  - layouts
  - menus
  - rasters
  - widgets

Status (cont'd)

- *Custom IDL-to-C++ translator (Ix)*
- *Graphics editor (fdraw)*
- *Script interpreter (dish)*

Fresco Summary

- *Advanced technology*
  - *Graphical embedding*
  - *Distributed objects*
- *Open standards effort (honest!)*
- *Implementation work on-going*

Object system

- *Defining object interfaces*
- *Transparent object references*
- *Ix – Interface translator*

IDL: Interface Definition Language

- *Defined by CORBA*
- *More abstract than C++*
  (no implementation details)
- *Can be bound to C, C++, ...*
- *Supports distribution*
  (can't use inherently local constructs)

IDL example

```
interface Window {
   void map();
   void unmap();
   boolean is_mapped();
   void raise();
   void lower();
};
```


C++ binding

Typedefs, structs, enums, constants

Interfaces –> abstract classes

Operations –> virtual functions

Attributes –> set/get function pairs

Exceptions –> exceptions or extra param

C++ example

```
class Window : public BaseObject {
protected:
   Window();
   virtual ~Window();
public:
   virtual void map();
   virtual void unmap();
   virtual Boolean is_mapped();

   virtual void raise();
   virtual void lower();
};
```


Parameter passing

IDL defines **in**, **out**, **inout**

- *in – pass value*
- *out – return value*
- *inout – pass and return value*

For C++

- *in – copy or const ref*
- *out – ref*
- *inout – ref*
- *return - value*

Object references

Like pointers, but (potentially) across address spaces

Interface type is always reference
void set_transient(in Window w)
–> void set_transient(WindowRef w);

ORB tracks object location

All operations are virtual function calls
vtbl points to implementation or stubs

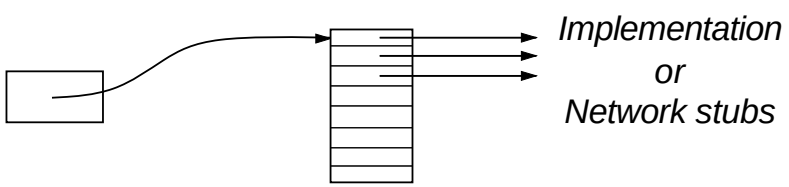
Transparency

Local or remote machine

Direct memory access or inter-process communication

	Local	*Remote*
Memory	*Function call*	*Page over network*
IPC	*Context switch*	*Network message*

Transparent object references



Ix: Sample interface translator

- *Generates*
  - *C++ or C header declarations*
  - *Client, server stubs*
  - *Dynamic type information*
- *Filters implementations*
- *Not full IDL*
  - *(contexts, any)*
- *Usage encapsulated in Imake macros*
  - *MakeHeader, EditImplHeader, EditImpl*

Generating a header file

Runs cpp to process #include's

Use -i option to generate #include's

Write out file.h, mark read-only

MakeHeaderFlags(display,-i '<Fresco/drawing.h>')

Example: Interfaces/screen.idl

```
interface Screen {
  attribute Coord dpi;

  Display display_ptr();

  Coord width();
  Coord height();

  PixelCoord to_pixels(in Coord c);
  Coord to_coord(in PixelCoord c);
  Coord to_pixels_coord(in Coord c);

  void move_pointer(in Coord x, in Coord y);

  Window application(in Viewer v);
  Window top_level(in Viewer v, in Window group);
  Window transient(in Viewer v, in Window trans);
  Window popup(in Viewer v);
  Window icon(in Viewer v);
};
```

Generated screen.h

```
/* DO NOT EDIT – Automatically generated ... */

#ifndef Interfaces_screen_h
#define Interfaces_screen_h

#include <Fresco/enter-scope.h>
#include <Fresco/drawing.h>
#include <Fresco/types.h>

class Screen : _Ix_public FrescoObject {
public:
  _Ix_InterfaceOps(Screen_ptr)

  virtual void dpi(Coord);
  virtual Coord dpi();
  virtual Display_ptr display_ptr();
  virtual Coord width();
  virtual Coord height();
```


class ScreenObj (cont'd)

```
  virtual PixelCoord to_pixels(Coord c);
  virtual Coord to_coord(PixelCoord p);
  virtual Coord to_pixels_coord(Coord c);
  virtual void move_pointer(Coord x, Coord y);
  virtual Window_ptr application(Viewer_ptr v);
  virtual Window_ptr top_level(
    Viewer_ptr v, Window_ptr group
  );
  virtual Window_ptr transient(
    Viewer_ptr v, Window_ptr trans
  );
  virtual Window_ptr popup(Viewer_ptr v);
  virtual Window_ptr icon(Viewer_ptr v);

  _Ix_MainOps(Screen)
};
```

Defining an implementation

- *Subclass from generated class or from another implementation class*
- *Edit subclass source with ix -filter*
  - *EditImplHeader(Xdisplay,$(I)/display.idl)*
  - *EditImpl(Xdisplay,$(I)/display.idl $(I)/viewer.idl)*

Use annotations to specify filtering

```
//+ Interface::Operation
//+ Interface::Attribute[=?]
//+ Subclass(Interface::Operation)
//+ Subclass(Interface::Attribute[=?])
//+ Interface::%init,type-funcs
//+
//+ Interface::%client,server
//+
//+ Subclass : Interface
//+
```

ScreenObj example

```
class ScreenImpl : public Screen {
public:
  ScreenImpl(DisplayImpl*, long n);
  ~ScreenImpl();

  //+ Screen::*
  /* FrescoObject */
  Long ref__(Long references);
    ...
  /* Screen */
  void dpi(Coord);
  Coord dpi();
  Display_ptr display_ptr();
  Coord width();
  Coord height();
  PixelCoord to_pixels(Coord c);
  Coord to_coord(PixelCoord p);
  Coord to_pixels_coord(Coord c);
    ...
  //+
```


ScreenObj example (cont'd)

```
protected:
  SharedFrescoObjectImpl object_;
  DisplayImpl* display_;
  long number_;
  XWindow root_;
  ScreenVisualList* visuals_;
  ScreenImpl::VisualInfo* default_visual_;
    ...
};
```

Xdisplay.cxx

```
#include <Fresco/defs/display.h>
#include <Fresco/defs/drawing.h>
   ...
//+ ScreenImpl(Screen::dpi=c)
void ScreenImpl::dpi(Coord c) {
   if (!Math::equal(c, float(0), float(1e-4))) {
      pixels_ = 72.0 / c;
      points_ = 1 / pixels_;
}

//+ ScreenImpl(Screen::dpi?)
Coord ScreenImpl::dpi() {
   return 72.0 / pixels_;
}

//+ ScreenImpl(Screen::application)
Window_ptr ScreenImpl::application(Viewer_ptr v) {
   return new ApplicationWindow(display_, this, v);
}
```


Stubs

```
//+ Screen::%init,type-funcs,client,server
_Ix_ctdt_0(Screen,6)
void* Screen::_this_ptr() { return this; }
_Ix_ImplementField(Screen,Screen,_Ix_NullParameter)

_Ix_extern_tid(Screen);
_Ix_ExternProxy(Screen);
_Ix_ImplementGetTid(Screen,_Ix__tid(Screen))
_Ix_ImplementNarrow(
   Screen_ptr,Screen,_Ix_NullParameter,Screen,
   &_Ix_CreateProxy(Screen)
)
_Ix_ImplementDuplicate(
   Screen_ptr,Screen,_Ix_NullParameter,
   _Ix__tid(Screen),&_Ix_CreateProxy(Screen)
)
   ...
```

Stubs (cont'd)

```
_Ix_MarshalBuffer::ArgDesc
   _Screen_move_pointer_pdesc[4] = {
      3, 4, 49, 49
};
_Ix_MarshalBuffer::ArgInfo
   _Screen_move_pointer_pinfo = {
      _Screen_tid_, 8,
      _Screen_move_pointer_pdesc, 0
};

void Screen::move_pointer(Coord x, Coord y) {
   _Ix_MarshalBuffer _b;
   _Ix_MarshalBuffer::ArgValue _arg[3];
   _arg[1].u_float = x;
   _arg[2].u_float = y;
   _b.invoke(this, _Screen_move_pointer_pinfo, _arg);
}
```


Creating objects

- *Constructors are global functions*
  - *Specify implementation class*
  - *Imply address space*
  - *Inappropriate for interfaces*
- *Can use operation on another object*
- *Kits create certain kinds of objects*
  - *DialogKit, FigureKit, LayoutKit, MenuKit, WidgetKit*
- *Access kits through naming context*

Example: Hello, world

```
#include <Fresco/defs/fresco.h>
#include <Fresco/figures/figures.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
  FrescoContext* fc = FrescoLib::init_default_context(
    “Test”, argc, argv
  );
  FigureKit_var figures = FigureKit::_narrow(
    FrescoLib::resolve_string(fc, "impls/figures")
  );
  FrescoLib::run(nil,
    Glyph_var(figures->label(
      FigureStyle_var(figures->default_style()), hello
    ))
  );
  CORBA::release(fc);
  return 0;
}
```


Object system summary

- *IDL for interfaces*
  - *No data members, inlines*
  - *No overloading, templates, const member functions*
- *C++ for implementations*
  - *Parameters are abstract*
- *Create objects with kits*
- *Cleaner, more extensible design*
- *Allow distibution, multiple languages*

Resolution independence

Coordinates and vertices

Regions

Display management

Drawing and printing

Coordinates

```
enum Axis { X_axis, Y_axis, Z_axis };
typedef float Coord;
typedef long PixelCoord;
struct Vertex { Coord x, y, z; };
typedef float Alignment;
```

Transformations

- *Objects assume real-world units (inch, mm, pt, ...)*
- *Units may be scaled by parent*
- *Relative coordinates – object-defined*
- *Absolute coordinates – fully-transformed*

Regions

- *Describe an arbitrary shape with origin*
- *May be only bounding box*
- *May be full solid geometry*

Region interface

```
interface Region {
   struct BoundingSpan {
      Coord begin, end;
      Coord origin, length;
      Alignment align;
   };
   boolean contains(in Vertex v);
   boolean contains_plane(in Vertex v, in Axis a);
   boolean intersects(in Region r);
   void copy(in Region r);
   void merge_intersect(in Region r);
   void merge_union(in Region r);
   void subtract(in Region r);
   void transform(in TransformObj t);
   void bounds(out Vertex lower, out Vertex upper);
   void origin(out Vertex v);
   void span(in Axis a, out Region::BoundingSpan s);
};
```


Display management

Display – keyboard/mouse/screens

Screen – output area

Window – opaque portion of screen

Drawing and Printing

- *Brush – line width and pattern*
- *Color – maps to device RGB*
- *Font – character shapes*
- *Raster – bitmaps and images*
- *Painter – 2d stencil/paint operations*
- *DrawingKit – create paint objects*

Drawing Kit interface

```
interface DrawingKit {
  typedef sequence<octet> Data8;
  typedef sequence<long> Data32;

  StyleContext style();

  Brush simple_brush(in Coord width);
  Brush dither_brush(in Coord width, in long p);
  Brush patterned_brush(
    in Coord width, in DrawingKit::Data32 p
  );

  Color color_rgb(
    in Color::Intensity r, in Color::Intensity g,
    in Color::Intensity b
  );
  Color find_color(in CharString name);

  Font find_font(in CharString fullname);
```

DrawingKit interface (cont'd)

```
  Raster raster_open(in CharString n);
  Raster bitmap_from_data(
    in Data8 data,
    in Raster::Index rows, in Raster::Index columns,
    in Raster::Index origin_row,
    in Raster::Index origin_column
  );
  Raster bitmap_from_char(in Font f, in CharCode c);

  Transform identity_transform();
  Transform transform_matrix(in Transform::Matrix m);

  Painter printer(in CharString output);
};
```


Painter interface

```
interface Painter {
  Coord to_coord(in PixelCoord p, in Axis a);
  PixelCoord to_pixels(in Coord c, in Axis a);
  Coord to_pixels_coord(in Coord c, in Axis a);

  void begin_path();
  void move_to(in Coord x, in Coord y);
  void line_to(in Coord x, in Coord y);
  void curve_to(
    in Coord x, in Coord y, in Coord x1, in Coord y1,
    in Coord x2, in Coord y2
  );
  void close_path();

  attribute Brush current_brush;
  attribute Color current_color;
  attribute Font current_font;

  void fill();
  void stroke();
```

Painter interface (cont'd)

```
  void character(
    in Font f, in CharCode ch, in Coord width,
    in Coord x, in Coord y
  );
  void image(in Raster r, in Coord x, in Coord y);
  void stencil(in Raster r, in Coord x, in Coord y);

  attribute Transform current_matrix;
  void push_matrix();
  void pop_matrix();
  void premultiply(in Transform t);
  void clip(
    in Coord x0, in Coord y0, in Coord x1, in Coord y1
  );
  void push_clipping();
  void pop_clipping();
  boolean is_visible(in Region r);

  void comment(in CharString s);
  void page_number(in CharString s);
};
```


Example: Bevelled rectangle

```
void Beveler::fill_path(
  PainterObjRef p, Vertex* v, long n
) {
  p->begin_path();
  p->move_to(v[0].x, v[0].y);
  for (long i = 1; i < n; i++) {
    p->line_to(v[i].x, v[i].y);
  }
  p->close_path();
  p->fill();
}
```

Example (cont'd)

```
// Bevelled rectangle
Coord left_inside = left + thickness;
Vertex v[10];
/* left edge */
set_color(p, light);
v[0].x = left; v[0].y = bottom;
v[1].x = left; v[1].y = top;
v[2].x = left_inside; v[2].y = top;
v[3].x = left_inside; v[3].y = bottom;
fill_path(p, v, 4);
```


Summary

Coordinate units

Regions, vertices

Display, Screen, Window

Drawing and printing

Brush
Color
Cursor
DisplayObj
DrawingKit
Font
PainterObj
Raster
Region
ScreenObj
TransformObj
Window
WindowStyle

Graphical embedding

- Natural blend of
  - structured graphics and
  - application embedding
- Embed a UI application object in
  - a graphical scene
- Scene may involve
  - Arbitrary shapes (including holes)
  - Graphical rotatation or zooming
  - 3D

Structured graphics

- Graphical objects
  - Shapes, properties, transforms
  - Overlays, transparency
  - Double-buffering
  - Sharing instances
- Draw, pick, bounding box
- Grouping into composites
- Attribute propagation
- Transcription – save to file

Application Embedding

- *Graphical output*
- *Geometry management*
- *Input handling*
- *Attribute propagation*
- *Focus management*
- *Menubar, palette sharing*
- *Transcription – save to a file*

Common framework

- **Viewer** – *user interface to data*
  - *Subtype of structured graphics object*
  - *Input handling, focus management*
  - *Menubar, palette sharing*
- **Glyph** – *graphical output*
  - *Geometry management*
  - *Screen update with instancing*
- **StyleContext** – *attribute propagation*

Advantages of common framework

- *Simplicity – fewer overall concepts*
- *Scalability – span range of granularity*
- *Interoperability – mix and match*
- *Flexibility – 3d friendly*
- *Power – editing rotated text in drawing*

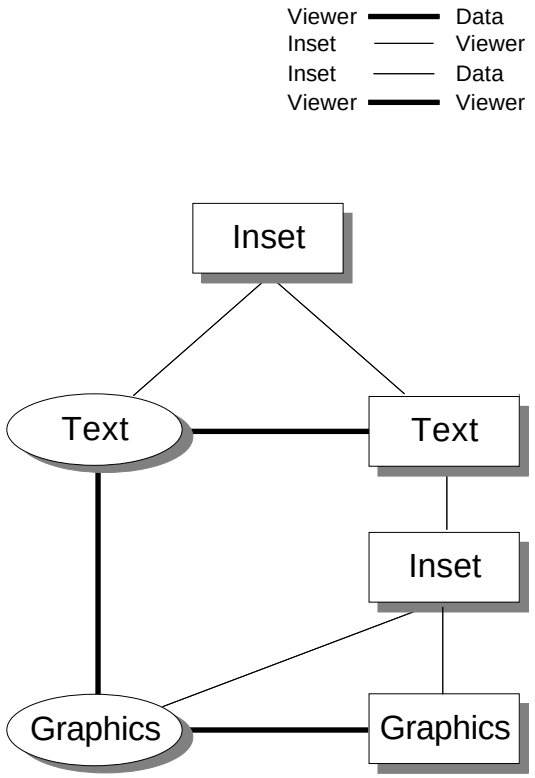
Viewers - UI to data

- *Present and possibly edit*
- *Like ATK views, OLE edit-in-place, Xt widgets (with extensions)*
- *Wide range of sizes and functionality*
  - *Button, Hypertext link, Field editor*
  - *Movie order form, Table editor*
  - *Image viewer, Drawing editor, Builder*
  - *Word processor*
  - *Interactive encyclopedia*

Viewer hierarchy

- *Viewer children correspond to embedded data*
- *Embedded “inset” contains data and default viewer information*
- *“Main” viewer at root implements focus, sharing menubars, palettes*

Inset example



Viewer interface

```
interface Viewer : Glyph {
  Viewer parent_viewer();
  Viewer next_viewer();
  Viewer prev_viewer();
  Viewer first_viewer();
  Viewer last_viewer();

  void append_viewer(in Viewer v);
  void prepend_viewer(in Viewer v);
  void insert_viewer(in Viewer v);
  void replace_viewer(in Viewer v);
  void remove_viewer();
  void set_viewer_links(
    in Viewer parent, in Viewer prev, in Viewer next
  );
  void set_first_viewer(in Viewer v);
  void set_last_viewer(in Viewer v);
```


Viewer interface (cont'd)

```
  Focus request_focus(
    in Viewer v, in boolean temporary
  );
  boolean receive_focus(
    in Focus f, in boolean primary
  );
  void lose_focus(in boolean temporary);
  boolean first_focus();
  boolean last_focus();
  boolean next_focus();
  boolean prev_focus();

  boolean handle(in GlyphTraversal t, in Event e);
  void close();
};
```

Focus interface

```
interface Focus : FrescoObject {
  void add_focus_interest(in Viewer v);
  void receive_focus_below(
    in Viewer v, in boolean temporary
  );
  void lose_focus_below(
    in Viewer v, in boolean temporary
  );
  void map_keystroke(
    in Event::KeySym k, in Action a
  );
  void map_keychord(
    in Event::KeyChord k, in Action a
  );
};
```


Implementation classes

#include <Fresco/Impls/viewers.h>

ViewerImpl

implements hierarchy, input operations

move, press, drag, release, double_click, other

ActiveViewer

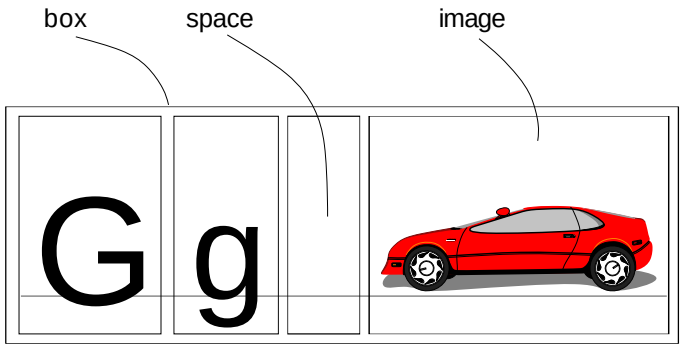
enter, leave

MainViewer

implements focus, damage

Glyphs - objects that draw

Think of a glyph as a character



Glyph composition

- *Directed acyclic graph (dag)*
- *Glyphs are nodes*
- *GlyphOffsets are edges*
- *Offset implementations knows about parent implementation*
- *GlyphTraversal*
  - *Shared state during a traversal*
  - *Path (“trail”) from root to current node*

Glyph interface

```
interface Glyph : FrescoObject {
  struct Requirement {
    boolean defined;
    Coord natural, maximum, minimum;
    Alignment alignment;
  };
  struct Requisition {
    Requirement x, y, z;
    boolean preserve_aspect;
  };
  struct AllocationInfo {
    Region allocation;
    Transform transform;
    Damagj damage;
  };
  typedef sequence<AllocationInfo> AllocationInfoList;

  attribute StyleContext style;
  Transform transform_ptr();
```


Glyph interface (cont'd)

```
  void request(out Glyph::Requisition r);
  void extension(
    in Glyph::AllocationInfo a, in Region r
  );
  void shape(in Region r);

  void traverse(in GlyphTraversal t);
  void draw(in GlyphTraversal t);
  void pick(in GlyphTraversal t);

  attribute Glyph body;

  GlyphOffset append(in Glyph g);
  GlyphOffset prepend(in Glyph g);

  Tag add_parent(in GlyphOffset parent_offset);
  void remove_parent(in Tag add_tag);

  void visit_children(in GlyphVisitor v);
  void visit_children_reversed(in GlyphVisitor v);
  void visit_parents(in GlyphVisitor v);
```

Glyph interface (cont'd.)

```
    // screen update
    void allocations(out Glyph::AllocationInfoList a);
    void need_redraw();
    void need_redraw_region(in Region r);
    void need_resize();
};
```


Geometry management

Requisition - what a glyph wants

- *Requirement per axis*
- *Returned by Glyph::request*

Allocation - region a glyph gets

- *BoundingSpan per axis*
- *Passed as part of GlyphTraversal*

Extension - bounds of glyph output

- *Computed by Glyph::extension*
- *Useful for update, culling*

Traversals

```
enum Operation {
   draw, pick_top, pick_any, pick_all, other
};

Operation op();
Operation swap_op(in Operation op);

void begin_trail(in Viewer v);
void end_trail();

void traverse_child(
   in GlyphOffset o, in Region allocation
);
void visit();

GlyphTraversal trail();
Viewer current_viewer();
Glyph current_glyph();
GlyphOffset offset();
void forward();
void backward();
```


Traversals (cont’d)

```
   attribute Painter current_painter;
   Display current_display();
   Screen current_screen();

   Region allocation();
   boolean bounds(
      out Vertex lower, out Vertex upper, out Vertex origin
   );

   Damage damaged();

   void hit();
   attribute long hit_info;
   GlyphTraversal picked();
   void clear();
};
```

Glyphs are lightweight

- *Simple set of operations*
- *No required base class storage*
- *Pass context during traversal*
- *Can be shared*

Of course, they may be heavyweight

Screen update

need_redraw

- *Damage all allocations for glyph*

need_resize

- *Damage old/new allocations for glyph*

Root object is drawn to repair damage

Output is clipped to damage

Objects may check if they are visible

- *If not then avoid drawing children*

Advantages

- *Objects can share area*
  - *e.g., a label on top of an image*
- *Output may be double-buffered*
  - *Supports smooth interaction, animation*
  - *Sample implementation uses pixmaps*
- *Objects control granularity of update*
  - *Cheap objects just draw*

Attribute management

- *Separate for independent use*
- *Loosely-coupled to structured graphics*
- *Tightly-coupled to embedding*
- *StyleContext – inherited name-value pairs*
- *StyleValue – associated value*

FrescoObject

```
// reference counting

// observability
Tag attach(in FrescoObject observer);
void detach(in Tag attach_tag);
void disconnect();

// notication
void notify_observers();
void update();
```


Callbacks

```
interface Action {
   void execute();
   boolean reversible();
   void unexecute();
};
```

Example

```
#include <Fresco/defs/fresco.h>
#include <Fresco/figures/figures.h>
#include <Fresco/layouts/layouts.h>
#include <Fresco/widgets/widgets.h>
#include <Fresco/impls/action.h>
#include <stdio.h>

class App {
public:
   App() { }
   ~App() { }

   int main(int argc, char** argv);
private:
   void push_me();
};

declareActionCallback(App)
implementActionCallback(App)

void App::push_me() { printf(“pushed\n”); }
```


Example (cont’d)

```
int App::main(int argc, char** argv) {
   FrescoContext* fc = FrescoLib::init_default_context(
      “Test”, argc, argv
   );
   FigureKit_var figures = FigureKit::_narrow(
      FrescoLib::resolve_string(fc, "impls/figures")
   );
   WidgetKit_var widgets = ...;
   LayoutKit_var layouts = ...;
   Button_ptr b = widgets->push_button(
      figures->label(
         figures->default_style(),
         FrescoLib::string_ref(“Push me”)
      ),
      new ActionCallback(App)(this, &App::push_me)
   );
   f->run(b, layouts->margin(b, 10.0));
   return 0;
}
```

Graphical embedding summary

- *Unify structured graphics and embedding*
- *Glyphs*
  - *Graphics, layout, lightweight,*
  - *Shared, damage-based update*
- *Viewers*
  - *User interface to data*
  - *Input handling and focus management*
  - *Menubar, palette sharing*
- *Distributed embedding “for free”*

Interfaces

Action
Damage
Event
Focus
Glyph
GlyphOffset
GlyphTraversal
GlyphVisitor
StyleContext
StyleValue
StyleVisitor
Viewer

Interoperability with other toolkits

- *Structured graphics may be used inside another toolkit*
- *Viewers may be implemented with X subwindows, use Xlib drawing*
- *Partial functionality – embedding without full graphics support*
- *Gateway objects to/from Xt*

XtfviewerWidgetClass example

```
#include <Fresco/Xtf/XtIntrinsic.h>
#include <Fresco/Xtf/Xdefs.h>
#include <X11/StringDefs.h>
  ...
#include <Fresco/Xtf/Xundefs.h>
#include <Fresco/figures/figures.h>
#include <Fresco/layouts/layouts.h>
#include <Fresco/Xtf/XtfFresco.h>
#include <Fresco/Xtf/XtfViewer.h>

static void exit_callback(Widget, XtPointer, XtPointer) {
  exit(0);
}

int main(int argc, char** argv) {
  XtToolkitInitialize();
  XtAppContext app = XtCreateApplicationContext();
  XDisplay* xdisplay = XtOpenDisplay(
    app, NULL, “xtcircle”, “XtCircle”, NULL, 0,
    &argc, argv
  );
```

Xtf example (cont'd)

```
Arg arglist[2];
XtSetArg(arglist[0], XtNallowShellResize, FALSE);
Widget toplevel = XtAppCreateShell(
   "XtTest", NULL, applicationShellWidgetClass,
   xdisplay, arglist, 1
);

Widget paned = XtCreateManagedWidget(
   "paned", panedWidgetClass, toplevel, NULL, 0
);
Widget buttons = XtCreateManagedWidget(
   "buttons", panedWidgetClass, paned, NULL, 0
);
Widget quit_button = XtCreateManagedWidget(
   "quit", commandWidgetClass, buttons, NULL, 0
);
XtfFrescoContext* f = XtfFresco_init(
   app, xdisplay, "XtTest", argc, argv
);
```


Xtf example (cont'd)

```
   FigureKit_var figures = ...;
   Glyph_var g = figures->figure_root(
      figures->circle(
         FigureKit::fill, figures->default_style(),
         0.0, 0.0, 100.0
      )
   );

   Widget fw = XtCreateManagedWidget(
      "circle", XtfviewerWidgetClass, paned, nil, 0
   );
   f->init_widget_glyph(fw, g);

   XtRealizeWidget(toplevel);
   XtMapWidget(fw);
   XtAppMainLoop(app);
   Fresco::unref(f);
   return 0;
}
```

Script interpretation

- *CORBA Dynamic invocation*
- *Tcl scripting language*
- *Dish interpreter*

CORBA Dynamic invocation

- *Create request object*
- *Fill in operation name (as a string)*
- *Fill in parameters*
- *Call invoke on request*
- *Examine result*

Tcl scripting language

- *Shell-like: cmd args ...*
- *Everything is a string*
- *Variables (set, $var)*
- *Procedures, if, for, while*
- *Nested function calls*
- *Commands may be compiled functions*

Dish: Dynamic invocation shell

- *Standard Tcl interpreter*
- *Interactive or batch script*
- *Fresco class operations builtin*
- *Unknown cmd uses dynamic invocation to convert parameters, invoke method, and convert back results (including out parameters)*
- *Tcl command registry is automatic*

Example: Hello, world

```
main 0 [
   figure_kit label [figure_kit default_style] “Hello, world”
]
```

Another example: circle

```
set style [figure_kit default_style]
main 0 [
   figure_kit figure_root [
      figure_kit circle 0 style 0 0 100
   ]
]
```


Button example

```
proc push_me {message} {
   puts $message
   flush stdout
}

set button {
   widget_kit push_button [
      figure_kit label [figure_kit default_style] “Push me”] [
      action push_me “pushed”
   ]
]

main button [layout_kit margin button 10.0]
```

Conclusions

- *Fresco is a new user interface system*
- *Broad spectrum of functionality*
- *Technology advances:*
  - *Graphical embedding*
  - *CORBA-based*
  - *Scripting access is automatic*
  - *Multithreading*
- *Working toward X Consortium standard*
  - *Sample implementation on R6*

What to do now?

- *XCI member:*
  - *Get current snapshot*
  - *Send feedback to mailing list*
- *Non-member:*
  - *Join XCI, or*
  - *Contact me*