

#\$+K!

Object Graphics Library 3.0

Julian Smart

September 1998

Contents

[Introduction](#)

[OGLEdit: a sample OGL application](#)

[Class reference](#)

^Contents

^Contents

^browse00001

^K Contents

^DisableButton("Up")

Introduction

Object Graphics Library (OGL) is a C++ library supporting the creation and manipulation of simple and complex graphic images on a canvas.

It can be found in the directory `utils/ogl/src` in the wxWindows distribution. The file `ogl.h` must be included to make use of the library.

Please see [OGL overview](#) for a general description how the object library works. For details, please see the [class reference](#).

File structure

Introduction

topic0

rowse00002

^K Introduction

^DisableButton("Up")

^{##+K!} **OGLEdit: a sample OGL application**

OGLEdit is a sample OGL application that allows the user to draw, edit, save and load a few shapes. It should clarify aspects of OGL usage, and can act as a template for similar applications. OGLEdit can be found in `samples/ogledit` in the OGL distribution.

{bmc ogledit.bmp}

The wxWindows document/view model has been used in OGL, to reduce the amount of housekeeping logic required to get it up and running. OGLEdit also provides a demonstration of the Undo/Redo capability supported by the document/view classes, and how a typical application might implement this feature.

OGLEdit files

How OGLEdit works

Possible enhancements

^oGLEdit: a sample OGL application

^ogledit

^browse00004

^K OGLEdit a sample OGL application

^DisableButton("Up")

Class reference

These are the main OGL classes.

[wxOGLConstraint](#)

[wxBitmapShape](#)

Class reference

classref

rowse00008

Class reference

isableButton("Up")

\$#+K! **File structure**

These are the files that comprise the OGL library.

basic.h Header for basic objects such as wxShape and wxRectangleShape.

basic.cpp Basic objects implementation (1).

basic2.cpp Basic objects implementation (2).

bmpshape.h wxBitmapShape class header.

bmpshape.cpp wxBitmapShape implementation.

canvas.h wxShapeCanvas class header.

canvas.cpp wxShapeCanvas class implementation.

composit.h Composite object class header.

composit.cpp Composite object class implementation.

constrnt.h Constraint classes header.

constrnt.cpp Constraint classes implementation.

divided.h Divided object class header.

divided.cpp Divided object class implementation.

drawn.h Drawn (metafile) object class header.

drawn.cpp Drawn (metafile) object class implementation.

graphics.h Main include file.

lines.h wxLineShape class header.

lines.cpp wxLineShape class implementation.

misc.h Miscellaneous graphics functions header.

misc.cpp Miscellaneous graphics functions implementation.

ogldiag.h wxDiagram class header.

ogldiag.cpp wxDiagram implementation.

F:ile structure

t:opic1

b:rowse00003

K: File structure

E:nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `topic0')")

mfutils.h Metafile utilities header.

mfutils.cpp Metafile utilities implementation.

OGLEdit files

OGLEdit comprises the following source files.

{bmc bullet.bmp} doc.h, doc.cpp: MyDiagram, DiagramDocument, DiagramCommand, MyEvtHandler classes related to diagram functionality and documents.

{bmc bullet.bmp} view.h, view.cpp: MyCanvas, DiagramView classes related to visualisation of the diagram.

{bmc bullet.bmp} ogledit.h, ogledit.cpp: MyFrame, MyApp classes related to the overall application.

{bmc bullet.bmp} palette.h, palette.cpp: EditorToolPalette implementing the shape palette.

^oGLEdit files

^topic2

^browse00005

^K OGLEdit files

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `ogledit')")

How OGLEdit works

OGLEdit defines a DiagramDocument class, each of instance of which holds a MyDiagram member which itself contains the shapes.

In order to implement specific mouse behaviour for shapes, a class MyEvtHandler is defined which is 'plugged into' each shape when it is created, instead of overriding each shape class individually. This event handler class also holds a label string.

The DiagramCommand class is the key to implementing Undo/Redo. Each instance of DiagramCommand stores enough information about an operation (create, delete, change colour etc.) to allow it to carry out (or undo) its command.

Apart from menu commands, another way commands are initiated is by the user left-clicking on the canvas or right-dragging on a node. MyCanvas::OnLeftClick in view.cpp shows how the appropriate wxClassInfo is passed to a DiagramCommand, to allow DiagramCommand::Do to create a new shape given the wxClassInfo.

The MyEvtHandler right-drag methods in doc.cpp implement drawing a line between two shapes, detecting where the right mouse button was released and looking for a second shape. Again, a new DiagramCommand instance is created and passed to the command processor to carry out the command.

DiagramCommand::Do and DiagramCommand::Undo embody much of the interesting interaction with the OGL library. A complication of note when implementing undo is the problem of deleting a node shape which has one or more arcs attached to it. If you delete the node, the arc(s) should be deleted too. But multiple arc deletion represents more information that can be incorporated in the existing DiagramCommand scheme. OGLEdit copes with this by treating each arc deletion as a separate command, and sending Cut commands recursively, providing an undo path. Undoing such a Cut will only undo one command at a time - not a one to one correspondence with the original command - but it's a reasonable compromise and preserves Do/Undo while keeping our DiagramCommand class simple.

How OGLEdit works

topic3

browse00006

^K How OGLEdit works

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('ogl.hlp', `ogledit`)")

^{##+K!}**Possible enhancements**

OGLEdit is very simplistic and does not employ the more advanced features of OGL, such as:

- {bmc bullet.bmp} attachment points (arcs are drawn to particular points on a shape)
- {bmc bullet.bmp} metafile and bitmaps shapes
- {bmc bullet.bmp} divided rectangles
- {bmc bullet.bmp} composite shapes, and constraints
- {bmc bullet.bmp} creating labels in shape regions
- {bmc bullet.bmp} arc labels (OGL has support for three movable labels per arc)
- {bmc bullet.bmp} spline and multiple-segment line arcs
- {bmc bullet.bmp} adding annotations to node and arc shapes
- {bmc bullet.bmp} line-straightening (supported by OGL) and alignment (not supported directly by OGL)

These could be added to OGLEdit, at the risk of making it a less useful example for beginners.

^Possible enhancements

^topic4

^browse00007

^K Possible enhancements

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `ogledit')")

wxOGLConstraint

wxCompositeShape overview

An wxOGLConstraint object helps specify how child shapes are laid out with respect to siblings and parents.

wxheadingDerived from

wxObject

wxheadingSee also

wxCompositeShape

wxheadingMembers

wxOGLConstraint::wxOGLConstraint
wxOGLConstraint::~~wxOGLConstraint
wxOGLConstraint::Equals
wxOGLConstraint::Evaluate
wxOGLConstraint::SetSpacing

^wxOGLConstraint

^wxoglconstraint

^browse00009

^K wxOGLConstraint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

wxBitmapShape

Draws a bitmap (non-resizable).

wxheadingDerived from

wxRectangleShape

wxheadingMembers

wxBitmapShape::wxBitmapShape

wxBitmapShape::~~wxBitmapShape

wxBitmapShape::GetBitmap

wxBitmapShape::GetFilename

^wxBitmapShape

^wxbitmapshape

^browse00015

^K wxBitmapShape

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `classref')")

wxOGLConstraint::wxOGLConstraint

wxOGLConstraint()^K

Default constructor.

wxOGLConstraint(int type, wxShape *constraining, wxList& constrained)^K

Constructor.

wxheadingParameters

docparamconstrainingThe shape which is used as the reference for positioning the *constrained* objects.

docparamconstrainedContains a list of wxShapes which are to be constrained (with respect to *constraining*) using *type*.

docparamtypeCan be one of:

{bmc bullet.bmp} **gyCONSTRAINT_CENTRED_VERTICALLY**: the Y co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same

{bmc bullet.bmp} **gyCONSTRAINT_CENTRED_HORIZONTALLY**: the X co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same

{bmc bullet.bmp} **gyCONSTRAINT_CENTRED_BOTH**: the co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same

{bmc bullet.bmp} **gyCONSTRAINT_LEFT_OF**: the X co-ordinates of the right hand vertical edges of the bounding boxes of the constrained objects will be less than the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_RIGHT_OF**: the X co-ordinates of the left hand vertical edges of the bounding boxes of the constrained objects will be greater than the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_ABOVE**: the Y co-ordinates of the bottom horizontal edges of the bounding boxes of the constrained objects will be less

^wxOGLConstraint::wxOGLConstraint

^wxoglconstraintconstr

^browse00010

^K wxOGLConstraint wxOGLConstraint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(ogl.hlp', `wxoglconstraint')")

^K wxOGLConstraint

^K wxOGLConstraint

than the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_BELOW**: the Y co-ordinates of the top horizontal edges of the bounding boxes of the constrained objects will be greater than the X co-ordinate of the bottom horizontal edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_ALIGNED_TOP**: the Y co-ordinates of the top horizontal edges of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_ALIGNED_BOTTOM**: the Y co-ordinates of the bottom horizontal edges of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the bottom horizontal edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_ALIGNED_LEFT**: the X co-ordinates of the left hand vertical edges of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_ALIGNED_RIGHT**: the X co-ordinates of the right hand vertical edges of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_MIDALIGNED_TOP**: the Y co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_MIDALIGNED_BOTTOM**: the Y co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the bottom horizontal edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_MIDALIGNED_LEFT**: the X co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object

{bmc bullet.bmp} **gyCONSTRAINT_MIDALIGNED_RIGHT**: the X co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object

$\$#+K!$ wxOGLConstraint::~~wxOGLConstraint

\sim wxOGLConstraint()^K

Destructor.

w_x OGLConstraint::~~wxOGLConstraint

t_{opic5}

$b_{rowse00011}$

K wxOGLConstraint ~wxOGLConstraint

E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint'))

K ~wxOGLConstraint

\$#+K! **wxOGLConstraint::Equals**

bool Equals(double x, double y)^K

Returns TRUE if x and y are approximately equal (for the purposes of evaluating the constraint).

wxOGLConstraint::Equals

topic6

browse00012

K wxOGLConstraint Equals

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint')")

K Equals

\$#+K!**wxOGLConstraint::Evaluate**

bool Evaluate()^K

Evaluates this constraint, returning TRUE if anything changed.

^wxOGLConstraint::Evaluate

^topic7

^browse00013

^K wxOGLConstraint Evaluate

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint')")

^K Evaluate

\$#+K! **wxOGLConstraint::SetSpacing**

void SetSpacing(double *x*, double *y*)^K

Sets the horizontal and vertical spacing for the constraint.

wxOGLConstraint::SetSpacing

wxoglconstraintsetspacing

browse00014

K wxOGLConstraint SetSpacing

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxoglconstraint')")

K SetSpacing

`$#+K!` **wxBitmapShape::wxBitmapShape**

wxBitmapShape()^K

Constructor.

`w`xBitmapShape::wxBitmapShape

`t`opic8

`b`rowse00016

`K` wxBitmapShape wxBitmapShape

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp`, `wxbitmapshape`)")

`K` wxBitmapShape

`$#+K!wxBitmapShape::~~wxBitmapShape`

`~wxBitmapShape()`^K

Destructor.

`wxBitmapShape::~~wxBitmapShape`

`topic9`

`browse00017`

`KwxBitmapShape ~wxBitmapShape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")`

`K ~wxBitmapShape`

`$#+K!` **wxBitmapShape::GetBitmap**

`constfuncwxBitmap&GetBitmap`

Returns a reference to the bitmap associated with this shape.

`wxBitmapShape::GetBitmap`

`topic10`

`browse00018`

`K wxBitmapShape GetBitmap`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp',`wxbitmapshape')")`

`wxBitmapShape::GetFilename`

`const wxString GetFilename`

`wxBitmapShape::GetFilename`

`topic11`

`rowse00019`

`wxBitmapShape GetFilename`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`ogl.hlp', `wxbitmapshape')")`

