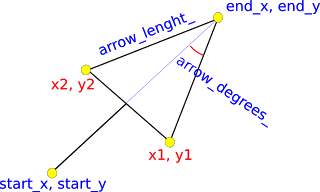
# 箭头绘制原理

## Drawing arrows with Cairo

For my ggredit project I need to draw connectors between objects, connectors are simple lines that can have an arrow head at the begin and/or at the end.  
But how to draw an arrowed line? We have to come back at school days, when we learn trigonometry.

**Basics**

With the function [atan2](http://en.wikipedia.org/wiki/Atan2) we can obtain the angle of the source line. Adding to this angle a defined sub-angle (arrow\_degrees\_) we can get the left and right angle of the arrow arms. With this two angles and using a defined lenght for the arrow arms (arrow\_lenght\_) we can finally achieve two points to use for perform drawing operations.  
  
Now return to the future...we need to transform this on c++ code.

void calcVertexes(double start\_x, double start\_y, double end\_x, double end\_y, double& x1, double& y1, double& x2, double& y2)

{

double angle = atan2 (end\_y - start\_y, end\_x - start\_x) + M\_PI;

x1 = end\_x + arrow\_lenght\_ \* cos(angle - arrow\_degrees\_);

y1 = end\_y + arrow\_lenght\_ \* sin(angle - arrow\_degrees\_);

x2 = end\_x + arrow\_lenght\_ \* cos(angle + arrow\_degrees\_);

y2 = end\_y + arrow\_lenght\_ \* sin(angle + arrow\_degrees\_);

}

**Effective arrow object**

We want to offer many kinds of arrows to the user, like on the wikipedia page for the[symbol Arrow](http://en.wikipedia.org/wiki/Arrow_(symbol)). I start to define this kind of arrows:

enum ArrowStyle

{

ARROW\_OPEN,

ARROW\_SOLID,

ARROW\_SOLID\_FILLED,

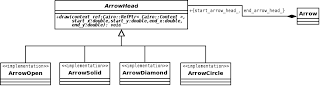
ARROW\_DIAMOND,

ARROW\_DIAMOND\_FILLED,

ARROW\_CIRCLE,

ARROW\_CIRCLE\_FILLED

};

But we do not like to put an [if-then-else](http://en.wikipedia.org/wiki/If-then-else) evry time we need to draw the arrow. Is better to define an ArrowHead base class with an abstract method draw, and implements all different styles inheriting from the ArrowHead class.  
  
The ArrowHead class is the base class.

class ArrowHead

{

public:

enum ArrowStyle

{

ARROW\_OPEN,

ARROW\_SOLID,

ARROW\_SOLID\_FILLED,

ARROW\_DIAMOND,

ARROW\_DIAMOND\_FILLED,

ARROW\_CIRCLE,

ARROW\_CIRCLE\_FILLED

};

ArrowHead() :

arrow\_lenght\_( 15 ),

arrow\_degrees\_( 0.5 )

{

}

virtual ~ArrowHead()

{

}

void calcVertexes(double start\_x, double start\_y, double end\_x, double end\_y, double& x1, double& y1, double& x2, double& y2)

{

double angle = atan2 (end\_y - start\_y, end\_x - start\_x) + M\_PI;

x1 = end\_x + arrow\_lenght\_ \* cos(angle - arrow\_degrees\_);

y1 = end\_y + arrow\_lenght\_ \* sin(angle - arrow\_degrees\_);

x2 = end\_x + arrow\_lenght\_ \* cos(angle + arrow\_degrees\_);

y2 = end\_y + arrow\_lenght\_ \* sin(angle + arrow\_degrees\_);

}

virtual void draw(Cairo::RefPtr< Cairo::Context > context\_ref, double start\_x, double start\_y, double end\_x, double end\_y) = 0;

protected:

double arrow\_lenght\_;

double arrow\_degrees\_;

};

The Arrow object has two arrow head, for draw the begin and the end arrow.

std::auto\_ptr< ArrowHead > start\_arrow\_head\_ptr\_;

std::auto\_ptr< ArrowHead > end\_arrow\_head\_ptr\_;

we can use this object for draw the arrow:

start\_arrow\_head\_ptr\_->draw (context\_ref, x2, y2, x1, y1);

end\_arrow\_head\_ptr\_->draw (context\_ref, x1, y1, x2, y2);

Now we need to implements all classes for the effective renderering. We can start with the ARROW\_OPEN, like this:  
  
We simply need to draw two lines, the first line from end\_x, end\_y to x1, y1 the second line from end\_x, end\_y to x2, y2.

class ArrowOpen : public ArrowHead

{

public:

ArrowOpen() :

ArrowHead(),

line\_color\_( "black" )

{

}

virtual ~ArrowOpen()

{

}

void draw(Cairo::RefPtr< Cairo::Context > context\_ref, double start\_x, double start\_y, double end\_x, double end\_y)

{

double x1;

double y1;

double x2;

double y2;

calcVertexes (start\_x, start\_y, end\_x, end\_y, x1, y1, x2, y2);

context\_ref->set\_source\_rgb (line\_color\_.get\_red\_p(), line\_color\_.get\_blue\_p(), line\_color\_.get\_green\_p());

context\_ref->move\_to (end\_x, end\_y);

context\_ref->line\_to (x1, y1);

context\_ref->stroke();

context\_ref->move\_to (end\_x, end\_y);

context\_ref->line\_to (x2, y2);

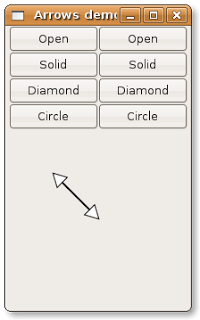
context\_ref->stroke();

}

protected:

Gdk::Color line\_color\_;

};

We can implement the ARROW\_SOLID, like this:  
  
In this case we need to draw a triangle using the three vertexes: end\_x, end\_y; x1, y1 and x2, y2.

class ArrowSolid : public ArrowHead

{

public:

ArrowSolid() :

ArrowHead(),

line\_color\_( "black" ),

fill\_color\_( "white" )

{

}

virtual ~ArrowSolid()

{

}

void draw(Cairo::RefPtr< Cairo::Context > context\_ref, double start\_x, double start\_y, double end\_x, double end\_y)

{

double x1;

double y1;

double x2;

double y2;

calcVertexes (start\_x, start\_y, end\_x, end\_y, x1, y1, x2, y2);

context\_ref->move\_to (end\_x, end\_y);

context\_ref->line\_to (x1, y1);

context\_ref->line\_to (x2, y2);

context\_ref->close\_path();

context\_ref->set\_source\_rgb (line\_color\_.get\_red\_p(), line\_color\_.get\_blue\_p(), line\_color\_.get\_green\_p());

context\_ref->stroke\_preserve();

context\_ref->set\_source\_rgb (fill\_color\_.get\_red\_p(), fill\_color\_.get\_blue\_p(), fill\_color\_.get\_green\_p());

context\_ref->fill();

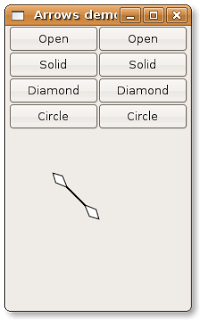
}

protected:

Gdk::Color line\_color\_;

Gdk::Color fill\_color\_;

};

We can implement the ARROW\_DIAMOND, like this:  
  
In this case we need four vertexes, but for now we have only three vertexes. The fourth vertex is also a point on the source line (so for the same angle of the source line), I use this method:

class ArrowDiamond : public ArrowHead

{

public:

ArrowDiamond() :

ArrowHead(),

line\_color\_( "black" ),

fill\_color\_( "white" )

{

arrow\_lenght\_ = 10.0;

}

virtual ~ArrowDiamond()

{

}

void draw(Cairo::RefPtr< Cairo::Context > context\_ref, double start\_x, double start\_y, double end\_x, double end\_y)

{

double angle = atan2 (end\_y - start\_y, end\_x - start\_x) + M\_PI;

double x1 = end\_x + arrow\_lenght\_ \* cos(angle - arrow\_degrees\_);

double y1 = end\_y + arrow\_lenght\_ \* sin(angle - arrow\_degrees\_);

double x2 = end\_x + arrow\_lenght\_ \* cos(angle + arrow\_degrees\_);

double y2 = end\_y + arrow\_lenght\_ \* sin(angle + arrow\_degrees\_);

double x3 = end\_x + arrow\_lenght\_ \* 2 \* cos(angle);

double y3 = end\_y + arrow\_lenght\_ \* 2 \* sin(angle);

context\_ref->move\_to (end\_x, end\_y);

context\_ref->line\_to (x1, y1);

context\_ref->line\_to (x3, y3);

context\_ref->line\_to (x2, y2);

context\_ref->close\_path();

context\_ref->set\_source\_rgb (line\_color\_.get\_red\_p(), line\_color\_.get\_blue\_p(), line\_color\_.get\_green\_p());

context\_ref->stroke\_preserve();

context\_ref->set\_source\_rgb (fill\_color\_.get\_red\_p(), fill\_color\_.get\_blue\_p(), fill\_color\_.get\_green\_p());

context\_ref->fill();

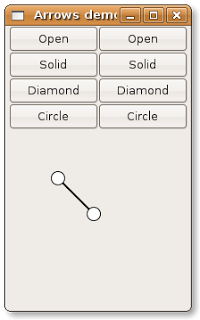
}

protected:

Gdk::Color line\_color\_;

Gdk::Color fill\_color\_;

};

The last implementation is the ARROW\_CIRCLE, like this:  
  
For draw a circle we need to have the center and the radius. We can define a radius (using arrow\_lenght\_) and then put the center on the source line.

class ArrowCircle : public ArrowHead

{

public:

ArrowCircle() :

ArrowHead(),

line\_color\_( "black" ),

fill\_color\_( "white" )

{

arrow\_lenght\_ = 7.0;

}

virtual ~ArrowCircle()

{

}

void draw(Cairo::RefPtr< Cairo::Context > context\_ref, double start\_x, double start\_y, double end\_x, double end\_y)

{

double angle = atan2 (end\_y - start\_y, end\_x - start\_x) + M\_PI;

double xc = end\_x + arrow\_lenght\_ \* cos(angle);

double yc = end\_y + arrow\_lenght\_ \* sin(angle);

context\_ref->arc (xc, yc, arrow\_lenght\_, 0.0, 2 \* M\_PI);

context\_ref->set\_source\_rgb (line\_color\_.get\_red\_p(), line\_color\_.get\_blue\_p(), line\_color\_.get\_green\_p());

context\_ref->stroke\_preserve();

context\_ref->set\_source\_rgb (fill\_color\_.get\_red\_p(), fill\_color\_.get\_blue\_p(), fill\_color\_.get\_green\_p());

context\_ref->fill();

}

protected:

Gdk::Color line\_color\_;

Gdk::Color fill\_color\_;

};

Now we need only two functions for set the begin and end arrow style:

void CairoArea::setStartArrowStyle(ArrowHead::ArrowStyle style)

{

switch (style)

{

case ArrowHead::ARROW\_OPEN:

start\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowOpen() );

break;

case ArrowHead::ARROW\_SOLID\_FILLED:

start\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowSolid() );

break;

case ArrowHead::ARROW\_DIAMOND\_FILLED:

start\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowDiamond() );

break;

case ArrowHead::ARROW\_CIRCLE\_FILLED:

start\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowCircle() );

break;

}

}

void CairoArea::setEndArrowStyle(ArrowHead::ArrowStyle style)

{

switch (style)

{

case ArrowHead::ARROW\_OPEN:

end\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowOpen() );

break;

case ArrowHead::ARROW\_SOLID\_FILLED:

end\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowSolid() );

break;

case ArrowHead::ARROW\_DIAMOND\_FILLED:

end\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowDiamond() );

break;

case ArrowHead::ARROW\_CIRCLE\_FILLED:

end\_arrow\_head\_ptr\_ = std::auto\_ptr< ArrowHead >( new ArrowCircle() );

break;

}

}

## 《vc++实例大全》这本书----第7章 文本输出与屏幕绘图---实例160——绘制带有箭头的线条

void CDrawArrowDemoView::DrawArrow(CPoint p1,CPoint p2,double theta,int length)  
{  
theta=3.1415926\*theta/180;//转换为弧度  
double Px,Py,P1x,P1y,P2x,P2y;  
//以P2为原点得到向量P2P1（P）  
Px=p1.x-p2.x;  
Py=p1.y-p2.y;  
//向量P旋转theta角得到向量P1  
P1x=Px\*cos(theta)-Py\*sin(theta);  
P1y=Px\*sin(theta)+Py\*cos(theta);  
//向量P旋转-theta角得到向量P2  
P2x=Px\*cos(-theta)-Py\*sin(-theta);  
P2y=Px\*sin(-theta)+Py\*cos(-theta);  
//伸缩向量至制定长度  
double x1,x2;  
x1=sqrt(P1x\*P1x+P1y\*P1y);  
P1x=P1x\*length/x1;  
P1y=P1y\*length/x1;  
x2=sqrt(P2x\*P2x+P2y\*P2y);  
P2x=P2x\*length/x2;  
P2y=P2y\*length/x2;  
//平移变量到直线的末端  
P1x=P1x+p2.x;  
P1y=P1y+p2.y;  
P2x=P2x+p2.x;  
P2y=P2y+p2.y;  
CClientDC dc(this);//获取客户窗口DC  
CPen pen,pen1,\*oldpen;  
pen.CreatePen(PS\_SOLID, 2, RGB(0, 0, 0));  
pen1.CreatePen(PS\_SOLID, 2, RGB(0, 0, 255));  
oldpen=dc.SelectObject(&pen);  
dc.MoveTo(p1.x,p1.y);  
dc.LineTo(p2.x,p2.y);  
dc.SelectObject(&pen1);  
dc.MoveTo(p2.x,p2.y);  
dc.LineTo(P1x,P1y);  
dc.MoveTo(p2.x,p2.y);  
dc.LineTo(P2x,P2y);  
dc.SelectObject(oldpen);  
}  
lz可以试试，我用过，没问题!

## DrawArrow

void CDrawArrowDemoView::DrawArrow(CPoint p1,CPoint p2,double theta,int length)

{

    theta=3.1415926\*theta/180;//转换为弧度

    double Px,Py,P1x,P1y,P2x,P2y;

    //以P2为原点得到向量P2P1（P）

    Px=p1.x-p2.x;

    Py=p1.y-p2.y;

    //向量P旋转theta角得到向量P1

    P1x=Px\*cos(theta)-Py\*sin(theta);

    P1y=Px\*sin(theta)+Py\*cos(theta);

    //向量P旋转-theta角得到向量P2

    P2x=Px\*cos(-theta)-Py\*sin(-theta);

    P2y=Px\*sin(-theta)+Py\*cos(-theta);

    //伸缩向量至制定长度

    double x1,x2;

    x1=sqrt(P1x\*P1x+P1y\*P1y);

    P1x=P1x\*length/x1;

    P1y=P1y\*length/x1;

    x2=sqrt(P2x\*P2x+P2y\*P2y);

    P2x=P2x\*length/x2;

    P2y=P2y\*length/x2;

    //平移变量到直线的末端

    P1x=P1x+p2.x;

    P1y=P1y+p2.y;

    P2x=P2x+p2.x;

    P2y=P2y+p2.y;

    CClientDC dc(this);//获取客户窗口DC

    CPen pen,pen1,\*oldpen;

    pen.CreatePen(PS\_SOLID, 2, RGB(0, 0, 0));

    pen1.CreatePen(PS\_SOLID, 2, RGB(0, 0, 255));

    oldpen=dc.SelectObject(&pen);

    dc.MoveTo(p1.x,p1.y);

    dc.LineTo(p2.x,p2.y);

    dc.SelectObject(&pen1);

    dc.MoveTo(p2.x,p2.y);

    dc.LineTo(P1x,P1y);

    dc.MoveTo(p2.x,p2.y);

    dc.LineTo(P2x,P2y);

    dc.SelectObject(oldpen);

}