```
// MLWPolylineCreator.cpp: implementation of the MLWPolylineCreator class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "mlwpolylinecreator.h"
#include "mcommandid.h"
#include "../maris/command.h"
#include "../maris/mdoc.h"
#include "../muires/marismsg.h"
#include "../mmath/gmath.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

namespace maris
{

/*
Notes:

Lwpolyline may contain line segments or arc segments. Each segment may have
a start width and/or end width.

Command allows user to
- change segment type between line and arc
- give the start and end widths

If segment is an arc segment, command allows user to create it using three points;
begin and end of segment, and the middle point of the arc.

Command starts by asking the first point. After it has been given, user has
the choices of
- giving the next point and produce a line segment
- change segment type to arc
- specify start and/or end widths.
*/


//////////////////////////////////////////////////////////////////////
// MLWPolylineCreator
//////////////////////////////////////////////////////////////////////

// Command factory object.
static command::factory<MLWPolylineCreator> cmdFactory;


MLWPolylineCreator::MLWPolylineCreator()
:   MGeometryCreateCommand(),
    _pPolyline(NULL),
    _currentSegment(),
    _state(FIRSTPOINT),
    _segmenttype(LINE),
    _segmentsdisabled(false),
                        _redoData()
{
}

MLWPolylineCreator::~MLWPolylineCreator()
{
    // if we own the polyline, delete it
    if ( _pPolyline )
    {
        delete _pPolyline;
        _pPolyline = NULL;
    }
}

void MLWPolylineCreator::disableSegments(void)
{
    _segmentsdisabled = true;
}
```

```
/** Returns the ID of the command */
muint MLWPolylineCreator::getId(void) const
{
		return MCID_CREATE_LWPOLYLINE;
}

/**
Return the pointer to the geometry.

Depending on the state of the command, returned pointer may point to:
- lwpolyline owned by this command in the middle of the construction
process
- lwpolyline that is fully finished and still owned by this command
(lwpolyline is not stored to document)
- lwpolyline that is fully finished and is added to the document.
*/
MGeometryPointer MLWPolylineCreator::getGeometry(void)
{
		// if we have a polyline pointer, return it.
		if ( _pPolyline )
			return MGeometryPointer(_pPolyline);

		// polyline is added to the document. Ask the pointer
		// from the base class.
		return MGeometryCreateCommand::getObjectPointer();
}


/**
Starts the command.
*/
MCEndResult MLWPolylineCreator::start(void)
{
   // Do base class start first
   MCEndResult result = MGeometryCreateCommand::start();

		if ( result == SUCCESS )
		{
			// add command to command stack.
			MUICommand::stackCommand();

      // set main command state to ACTIVE
			setState(ACTIVE);

			// Connect to mouse, keyboard and input
			connect();

			// Show name of the command
      showName();

      // add prompts
      if ( _segmentsdisabled )
      {
         addPrompt(PROMPT_GIVE_FIRST_POINT);        // 0
         addPrompt(PROMPT_GIVE_NEXTPOINT);          // 1
      }
      else
      {
         addPrompt(PROMPT_GIVE_FIRST_POINT);        // 0
         addPrompt(PROMPT_LWPOLYLINE_LINE_NEXT);    // 1
         addPrompt(PROMPT_LWPOLYLINE_STARTWIDTH);   // 2
         addPrompt(PROMPT_LWPOLYLINE_ENDWIDTH);     // 3
         addPrompt(PROMPT_ARC_POINT);               // 4
         addPrompt(PROMPT_LWPOLYLINE_ARC_NEXT);     // 5
      }

      // Start with a simple line segment.
      _currentSegment.bulge = _currentSegment.startWidth = _currentSegment.endWidth = 0.0f;
      _segmenttype = LINE;
      setInputState(FIRSTPOINT);
		}

		return result;
}
```

```
/**
Aborts the command
*/
void MLWPolylineCreator::abort(void)
{
                        end(ABORT, false);
}

/**
Called when the command is activated.
*/
void MLWPolylineCreator::activate(MCEndResult previous_command_result)
{
                        MGeometryCreateCommand::activate(previous_command_result);

    // Check command state and set input state.
                        setInputState(_state);
}

void MLWPolylineCreator::end(MCEndResult callingresult,mbool bUpdateDocChanged /*=true*/,muint32 viewUpdateFlags
/*=VIEW_UPDATE_NORMAL*/)
{
                        disconnect();
    removeDrawables(_pPolyline);

    MCEndResult result(callingresult);
    if ( result == SUCCESS )
    {
        muint index = getCurrentSegment();
                            if ( index >= 0 )
                            {
                                        if ( isSaveToDocument() )
                                        {
                                                    getDocument()->add(_pPolyline);
                                                    MGeometryCreateCommand::setObjectPointer(_pPolyline);
                                                    _pPolyline = NULL;
                                        }
        }
        else
            result = FAILURE;
    }

    if ( result != SUCCESS && _pPolyline )
    {
                                    delete _pPolyline;
                                    _pPolyline = NULL;
    }

                        // Don't need redo data anymore
                        _redoData.clear();

    MGeometryCreateCommand::end(result, bUpdateDocChanged, viewUpdateFlags);
}

mbool MLWPolylineCreator::isLastUndoable() const
{
                        if( _pPolyline )
                                    return _pPolyline->getCount() > 0;

                        return false;
}

mbool MLWPolylineCreator::isLastRedoable() const
{
                        return !_redoData.empty();
}

mbool MLWPolylineCreator::undoLast(void)
{
                        if( !_pPolyline )
                                    return false;

                        if( _pPolyline->getCount() <= 2 )
                        {
        // Start with a simple line segment.
        _currentSegment.bulge = _currentSegment.startWidth = _currentSegment.endWidth = 0.0f;
```

```
	_segmenttype = LINE;
	setInputState(FIRSTPOINT);

                                        if( _pPolyline->getCount() > 0 )
                                        {
                                                LWPLSegData lwplsdata = _pPolyline->getSegmentData(0);
                                                Point2d point = _pPolyline->getPoint(0);
                                                _redoData.push_back( std::make_pair(point,lwplsdata) );
                                        }

                                        removeDrawables(_pPolyline);

                                        delete _pPolyline;
                                        _pPolyline = NULL;

                                        update(VIEW_UPDATE_DRAWABLES);

                                        return true;
                        }
                        else if( _pPolyline->getCount() > 2 )
                        {
                                        Point2d point = _pPolyline->getPoint(_pPolyline->getCount()-2);
                                        LWPLSegData lwplsdata = _pPolyline->getSegmentData(_pPolyline->getSegmentCount()-2);

                                        _redoData.push_back( std::make_pair(point,lwplsdata) );

                                        // Segment starting point
                                        _points[0] = _points[1] = _pPolyline->getPoint(_pPolyline->getCount()-3);

                                        _pPolyline->remove( _pPolyline->getCount()-1 );
                                        lwplsdata = _pPolyline->getSegmentData(_pPolyline->getSegmentCount()-1);
                                        _currentSegment = lwplsdata;

                                        _segmenttype = isZero( _currentSegment.bulge ) ? LINE : ARC;

                                        setInputState(SEGMENT);

                                        update(VIEW_UPDATE_DRAWABLES);

                                        return true;
                        }
                        return false;
}

mbool MLWPolylineCreator::redoLast(void)
{
                        if( !_redoData.empty() )
                        {
                                        const std::pair<Point2d,LWPLSegData> redo_data = _redoData.back();
                                        _redoData.pop_back();

                                        _currentSegment = redo_data.second;

                                        if( _pPolyline )
                                                        modifySegment(getCurrentSegment(), NULL, NULL, &_currentSegment);

                                        _segmenttype = LINE;

                                        point(this,Point3d(redo_data.first));

                                        _segmenttype = isZero( _currentSegment.bulge ) ? LINE : ARC;

                                        // Force segment state
                                        setInputState(SEGMENT);

                                        return true;
                        }

                        return false;
}

////////////////////////////////////////////////////////////////////////////////
// input event handlers


void MLWPolylineCreator::move(void* pSrc,const Point3d& wcsPosition,muint keys)
```

```
{
                    if ( _pPolyline )
   {
      // Calculate only when drawing a segment or arc midpoint
      // But need to update the view in measure - state too, to update
      // the measure vector.
      if (_state == SEGMENT || _state == ARCPOINT)
      {
         Point2d point(wcsPosition);
         if ( _state == SEGMENT )
            // Store end point of segment.
            _points[1] = point;

         if ( _segmenttype == ARC )
            // Update arc segment.
            updateArcSegmentValues(point);
         else
            // Set last point of the lwpolyline
            modifySegment(getCurrentSegment(), NULL, &point, NULL);
      }

                                       // Update view.
                                       update(VIEW_UPDATE_DRAWABLES);
   }
}

/**
Called when user has entered a point.
*/
void MLWPolylineCreator::point(void* pSrc, const Point3d& point3d)
{
   // Make sure we have the object
   createObject();

   Point2d point(point3d);

   // If first point, add first segment
   if ( _state == FIRSTPOINT )
   {
      // update points.
      _points[0] = _points[0] = point;

      // add segment
      addSegment(point);

      // Can start to draw now
      startVisualClues();

      // Set segment state and change prompt.
      setInputState(SEGMENT);
   }
   else if ( _state == SEGMENT )
   {
      // Don't accecpt dual points
      if ( point != _points[0] )
      {
         _points[1] = point;

         // if segment type is an arc, read arc's middlepoint
         if ( _segmenttype == ARC )
         {
            // Set the end point of the segment.
            modifySegment(getCurrentSegment(), NULL, &point, NULL);

            setInputState(ARCPOINT);
         }
         else
         {
            // Add next segment.
            addSegment(point);

                                                                  if( pSrc != this )
                                                                        _redoData.clear();

            // Start of next is the end of previous
            _points[0] = point;
```

```
            }
        }
    }
    else if ( _state == ARCPOINT )
    {
        // Can't use end points
        if ( point != _points[0] && point != _points[1] )
        {
                                                        updateArcSegmentValues(point);

            // User has given the middle point of the arc.
            // Add next segment.
            addSegment(point);

                                                        _redoData.clear();

            // Start of next is the end of previous
            _points[0] = _points[1];

            // next segment
            setInputState(SEGMENT);
        }
    }

    update(VIEW_UPDATE_DRAWABLES);
}

/**
Called when user has entered a value. Used for setting the start/end
widths of an segment.
*/
void MLWPolylineCreator::measure(void* pSrc, mdouble value)
{
    massert(_state == BEGINWIDTH || _state == ENDWIDTH);

    // Note. Measure value may come negative, if they are given using the mouse.
    // Just convert them to positive.
    value = fabs(value);

    if ( _state == BEGINWIDTH )
    {
        // store width to the current segment data
        _currentSegment.startWidth = (mfloat) value;
    }
    else if ( _state == ENDWIDTH )
    {
        _currentSegment.endWidth = (mfloat) value;
    }

    // Update the lwpolylines segment
                    if( _pPolyline )
                    {
                                    modifySegment(getCurrentSegment(), NULL, NULL, &_currentSegment);

                                    // return to the segment state.
                                    setInputState(SEGMENT);
                    }
                    else
                                    setInputState(FIRSTPOINT);
}

/**
Handles the keyboard event.
*/
imKeyboard::Result MLWPolylineCreator::key(void* pSrc, muint key, muint repeats, muint flags)
{
    imKeyboard::Result result = imKeyboard::NotUsed;
    if ( _segmentsdisabled ) return result;

    // User can change the mode while drawing arc or line segment.
    if ( _state == FIRSTPOINT || _state == SEGMENT )
    {
        // Try finding the option. Scan the prompt for option selection character.
        // There's different prompts for line segment and arc segment.
        MCHAR str[2] = {key, 0};
        MString prompt;
```

```
        if ( _segmenttype == LINE )
            prompt = maris::loadString(PROMPT_LWPOLYLINE_LINE_NEXT);
        else
            prompt = maris::loadString(PROMPT_LWPOLYLINE_ARC_NEXT);

        mint option = getOption(prompt, str);

        // If found, option is the zero-based index of the option in the prompt.
        if ( option >= 0 )
        {
            // new state
            STATE state = _state;
            switch (option)
            {
                case 0: // line or arc option. Swicthes form one to another.
                    if ( _segmenttype == LINE )
                        _segmenttype = ARC;
                    else
                    {
                        _segmenttype = LINE;

                        // Set bulge to zero to force a line segment.
                        _currentSegment.bulge = 0.0f;
                    }
                    break;
                case 1: // Query begin width of the next segment
                    state = BEGINWIDTH;
                    break;
                case 2: // Query end width of the next segment
                    state = ENDWIDTH;
                    break;
            }

            // Set new input state and change prompt
            setInputState(state);

            // We used the key; no need to deliver event further.
            result = imKeyboard::Used;
        }
    }

                        return result;
}

/**
Aborts the command.
*/
void MLWPolylineCreator::cancel(void* pSrc, mbool TotalCancel)
{
    // If cancel comes during measure - input, don't
    // abort but continue with segment drawing.
    if ( _state == BEGINWIDTH || _state == ENDWIDTH )
        setInputState(SEGMENT);
    else
                            abort();
}

/**
Cancellation with the mouse. If polyline is valid, ends the command successfully,
else aborts it.
*/
void MLWPolylineCreator::cancelMouse(void* pSrc,const Point3d& wcsPosition,muint keys)
{
    // If cancel comes during measure - input, don't
    // abort but continue with segment drawing.
    if ( _state == BEGINWIDTH || _state == ENDWIDTH )
        setInputState(SEGMENT);
    else
    {
        // poistetaan objektista jo tässä vaiheessa ylimääräinen piste, koska peritty luokka voi käyttää objektia ennen
        // tämän luokan end:iä, jossa piste poistettiin ennen tätä muutosta. jsu 15.4.09
        muint index = getCurrentSegment();
        if ( index > 0 )
        {
            // Remove the extra point first
            modifyPolyline(index+1, false, NULL);
```

```
            end(SUCCESS);
        }
        else
            abort();
    }
}


////////////////////////////////////////////////////////////////////
// Helper methods.

/**
Creates the lwpolyline member object.
*/
void MLWPolylineCreator::createObject(void)
{
                        if ( !_pPolyline )
    {
        _pPolyline = new MLWPolyline;

        // set default values, color etc.
        setCurrentValues(_pPolyline);
    }
}

/**
Sets the input state according to command state
*/
void MLWPolylineCreator::setInputState(STATE newstate)
{
    // Main state of the command.
    if ( getState() == ACTIVE)
    {
        // Set state.
        _state = newstate;

        muint prompt = 0;   // First point - prompt.

        switch ( _state )
        {
            case FIRSTPOINT:        // first point of the lwpolyline
            case SEGMENT:           // segment, either line or arc.
            case ARCPOINT:          // Arc's middle point.
                // Ask points for both line and arc.
                setCoordinateInput();

                if ( _state != FIRSTPOINT )
                {
                    prompt = 1;     // Segment & line or next point
                    if ( _state == SEGMENT )
                    {
                        if ( _segmenttype == ARC )
                            prompt = 5;
                    }
                    else if ( _state == ARCPOINT )
                        prompt = 4;
                }
                break;

            case BEGINWIDTH:        // begin width of the segment
            case ENDWIDTH:          // end width of the segment
                // Set measure input and give a reference point.
                setMeasureInput( (_state == BEGINWIDTH) ? _points[0] : _points[1]);

                prompt = 2;
                if ( _state == ENDWIDTH )
                    prompt = 3;
                break;
        }

        // Set prompt
        showPrompt(getPrompt(prompt));
    }
}

/**
```

```
Starts showing the lwpolyline. Connects command to view and adds lwpolyline
to drawables.
*/
void MLWPolylineCreator::startVisualClues(void)
{
                        // Connect to view
                        connectViews();
    addDrawables(_pPolyline);
}

/**
When the current segment is an arc, updates the segment.
Parameter 'point' varies according to input state:
- SEGMENT: point is the end point of the segment
- ARCPOINT: point is the middle point of the arc segment.
*/
void MLWPolylineCreator::updateArcSegmentValues(const Point2d& point)
{
    massert(_segmenttype == ARC);
    massert(_state == SEGMENT || _state == ARCPOINT);

    // Get the points and segment data from lwpolyline.
    muint index = getCurrentSegment();
    LWPLSegData data(_currentSegment);

    // if end point, set bulge to zero and point to end point
    if ( _state == SEGMENT )
    {
        data.bulge = 0.0;
        // Set last point of the lwpolyline and data
        modifySegment(index, NULL, &point, &data);
    }
    else
    {
        // Calculate an arc using three points and start - point - end and
        // get the bulge value from it.
        data.bulge = calculateCurrentBulge(point);
        // no need to update the points.
        modifySegment(index, NULL, NULL, &data);
    }

}

/**
Returns the index of the current segment
*/
muint MLWPolylineCreator::getCurrentSegment(void)
{
    if ( _pPolyline && _pPolyline->getSegmentCount() > 0 )
        return _pPolyline->getSegmentCount() - 1;

    return 0;
}

/**
Adds a new segment. Point is the start point of the new segment.
Segment values are taken from _currentSegment member.
*/
void MLWPolylineCreator::addSegment(const Point2d& point)
{
    massert(_pPolyline);

    if ( _state == ARCPOINT )
        // Modify the current segment. 'point' is the point in the arc.
        _currentSegment.bulge = calculateCurrentBulge(point);

    if ( _state == FIRSTPOINT )
    {
        // add first segment.
        _pPolyline->append(&point, 1, &_currentSegment);

        // add extra point, so that polyline can be drawn.
        modifyPolyline(0, true, &point);
    }
    else
    {
```

```
        muint index = getCurrentSegment();

        // Remove the extra point first
        modifyPolyline(index + 1, false, NULL);

                                            // ssu 18.4.2012, next segment will start with same width as the previous ended
                                            _currentSegment.startWidth = _currentSegment.endWidth;

        // Add a new segment.
        _pPolyline->append(&_points[1], 1, &_currentSegment);

        // and add an extra point
        modifyPolyline(0, true, &_points[1]);
    }
}

/**
Modifies the segment 'index'. If a pointer != NULL, that value is updated.
*/
void MLWPolylineCreator::modifySegment(muint index, const Point2d* pStart, const Point2d* pEnd,
                                       LWPLSegData* pSegData)
{
    massert(_pPolyline);

                        // points ?
    if ( pStart || pEnd )
    {
        Point2d* p1 = NULL;
        Point2d* p2 = NULL;
        mbool ok = _pPolyline->getSegment(&p1, &p2, index);
                                            massert(p1);
                                            massert(p2);
        if ( ok )
        {
            if ( pStart )
                *p1 = *pStart;
            if ( pEnd )
                *p2 = *pEnd;

                                                                _pPolyline->postGeometricChange(false);
        }
#ifdef _DEGUG
        else
            massert(false);
#endif
    }

    // data ?
    if ( pSegData )
        _pPolyline->setSegmentData(*pSegData, index);
}

/**
Calculates the bulge for current (arc) segment.
*/
mfloat MLWPolylineCreator::calculateCurrentBulge(const Point2d& pointInArc)
{
    // Calculate an arc using three points and start - point - end and
    // get the bulge value from it.
    mArc2d arc;
    mArc2d::createCircularArc3(arc, _points[0], _points[1], pointInArc);
    mfloat bulge = (mfloat) arc.getBulge();

    // arc seems to calculate bulge as positive. Need to modify the
    // sign, so arc can be drawn "both ways".
    mint side = GMath::getSide(_points[0], _points[1], pointInArc);
    if ( side == 0 )
        bulge = -bulge;

    return bulge;
}

/**
Modifies the polyline underlying the lwpolyline by adding an extra point
or removing it.
Adding extra point is done so that lwpolyline can be drawn during mouse events.
```

```
This extra point must be removed.
*/
void MLWPolylineCreator::modifyPolyline(muint index, mbool add, const Point2d* pPoint)
{
    // NOTE. We use a raw cast from const mLWPolyline2d to mPolyline2d. Normally
    // there's no need to modify mPolyline2d direclty, so MLWPolyline does not
    // provide methods to do that. It has a method that gives a const reference
    // to the 2d geometry object mLWPolyline2d.

    const mLWPolyline2d& lwp = _pPolyline->getLWPolyline();

    // make sure mLWPolyline2d is mPolyline2d
#ifdef _DEGUG
    try {
        massert(dynamic_cast<mPolyline2d*>(&lwp) );
    }
    catch (... )
    {
        massert(false);
    }
#endif

    mPolyline2d& pl = (mPolyline2d&) lwp;

    if ( add )
    {
        massert(pPoint);
        pl.append(pPoint, 1);

    }
    else
        pl.remove(index);
}

} // namespace
```