Nesting Factory
Developer's Guide

info@algomate.com, www.algomate.com

Introduction

Nesting Factory (NF) is a software module for the automatic nesting.
It integrates into a partner CAM system (host) transparently to an end user.
The integration is done on the source code level at the compile time. Certain files are added to the installer package of the host. This tutorial covers these steps.

You can contact Algomate representative with the questions regarding Nesting Factory via e-mail:

support@algomate.com

Algomate

Integration

The overall work flow consists of the following steps -

1. Collect the geometry of the input items
2. Define the auxiliary parameters
3. Create a nesting task
4. Provide a callback function to be invoked upon task completion
5. Launch the task computation.

Step 4 above may be skipped if a synchronous mode is intended.

Geometric contour.
We consider a contour as a sequence of line segments and circle arcs. Each fragment is encoded as a triple (**x**,**y**,**b**), where
x,**y** – 2D coordinates of the beginning of a fragment, and
b – is a bulge notation, analogous to DXF standard. b is zero for a segment.

This functionality is implemented in *amCATInputContour* class.

Input Item
Input item consists of at least one geometric contour (*amCATInputContour*). An item has the following properties -

ID – a unique ID which this item can be distinguished with,
Quantity – the number of copies of this item that are required to be nested,
Reflection – a boolean flag defining reflection permission for the item, the default value is True
Rotation – a rotation step value, currently four values are supported: none, Pi, Half-Pi and free rotation. The default value is 'free'.

This functionality is implemented in *amCATInputItem* class.

Input Domain
Input domain refers a geometric region where the input items are intended to be nested. Any input item can be used as a domain. We provide simple API to create rectangular domains instantly. Each domain has -

ID – a unique ID which this domain can be distinguished with,
Quantity – the number of copies of this domain that can be utilized during nesting.
Length and width – two real numbers defining the geometric properties.
Domains of different sizes can be used in one task.

This functionality is implemented in *amCATInputDomain* class.

Nesting Task
Input items and domains are aggregated in Nesting Task. Two additional parameters are required to compute a task - Item-to-Item and Item-to-Domain distances. This two real numbers define the certain gaps.

This functionality is implemented in *amCATTask* class.

Facade

After the input data is prepared, it is provided to a Facade controlling object. This object launches the actual computation and invokes a callback function when the computation is done. The computation can be performed either asynchronously or sequentially.

This functionality is implemented in *amCATFacade* class.

No explicit deletion of the input objects is required at the end of the work.

Results

Nesting results are provided as series of geometric transformations of input items.

This series are packed into containers on a per-sheet basis, one *amCATNestingResult* object for a consumed sheet. Iterating through a Result container retrieves a Result item (*amCATNestedItemInstance* class). Each result item contains -

ID - the unique ID of the original Input Item that the host provided,

Instance ID - a per-item counter to distinguish between the nested copies of the same input item,

reflection - flag saying whether a reflection of the input item was performed for this instance during the nesting, two values are supported: CAT_NO_REFLECTION and CAT_Y_AXIS_REFLECTION

rotation - real number, counter-clockwise direction, degrees, saying how an instance of an input item was rotated,

transfer - two real numbers (a 2D vector), saying how an instance of an input item was moved.

The reflection and the rotation values agree with the permissions given for the input item. It's important to apply the transformation in the order mentioned above.

Please take a look on the following source code for the further explanations.

```
#include <amCATFacade.h>
#include <iostream>
//---------------------------------------------------------------------------
// In the this sample we are going to pack 3 frames and 2 circles.
// Frame has two square contours, 4 vertices each.
// Circle has one contour, diameter is 30.
// The vertex encoding is similar to DXF's polyline.
// Namely (x,y) pair and a bulge

// Some constants for easier notation
// We have 3 contours that combine 2 items.
#define NUMBER_OF_CONTOURS 3
#define CONTOUR_MAX_LENGTH 4*3
// 4 - the maximal number of vertices per contour in our example
// 3 - x,y,bulge triple
```

```
// Some junk values are in the buffers. Just to please the compiler.
#define NUMBER_OF_ITEMS    2
#define MAX_CONTOURS_PER_ITEM 2

//--------------------------------------------------------------------------
// This is a callback function that is invoked when the processing is finished.
// The function iterates through the results and prints the positioning
// transformation. The order of the transformation application is important.
// Namely: reflection, rotation, transfer.
void __cdecl TaskIsComplete( void* pT )
{
        amCATFacade* pFacade = (amCATFacade*) pT;
        int i = 0;
        const vector<amCATNestingResult>& Nestings = pFacade->getResults();
        vector<amCATNestingResult>::const_iterator iterCurrSheet =
                                        Nestings.begin();
        for( ; iterCurrSheet != Nestings.end(); ++iterCurrSheet )
        {
                const vector<amCATNestedItemInstance>& CurrSheet = *iterCurrSheet;
                vector<amCATNestedItemInstance>::const_iterator iterCurrItem =
                                        CurrSheet.begin();
                cout << "Sheet No. " << i++ << endl;
                for( ; iterCurrItem != CurrSheet.end(); ++iterCurrItem )
                {
                      cout << " Origin Item ID "
                           << iterCurrItem->nOriginID_ << endl ;
                     cout << " Packed Instance ID "
                           << iterCurrItem->nInstanceID_ << endl;
                     cout << " Reflection: (Y-axis) "
                           << (iterCurrItem->Transform_.Reflection_== CAT_Y_AXIS_REFLECTION)<< endl;
                     cout << " Rotation: (CCW, Degrees) "
                           << iterCurrItem->Transform_.CCWRotationDegrees_ << endl;
                     cout << " Translation: (2D Vector) "
                           << iterCurrItem->Transform_.Vector_[0]  << " "
                           << iterCurrItem->Transform_.Vector_[1] << endl;
                }
        }
}
```

```
//--------------------------------------------------------------------
// The entry point
int main()
{
	// The geometry row source data
	double pAllContours[NUMBER_OF_CONTOURS][CONTOUR_MAX_LENGTH]
			= { {  0,  0, 0,  30,  0, 0, 30, 30, 0,  0, 30, 0 },
			    // Outer contour comes first
			   { 10, 10, 0,  20, 10, 0, 20, 20, 0, 10, 20, 0 },
			    // Inner contour. CCW-ness isn't important
			   {  0,  0, 1,  30, 0,  1, -1,-1,-1,-1,-1,-1  } };
			    // Bulges are 1-s since this is a circle, '-1' just to please the compiler

	// The number of triples, not array lengths
	int	pContourLengths[NUMBER_OF_CONTOURS] = { 4, 4, 2 };

	// The 1st and the 2nd contour combines the frame
	// The 3rd contour is the circle
	int	pItemContoursIndeces[NUMBER_OF_ITEMS][MAX_CONTOURS_PER_ITEM] =
				{  {0, 1}, {2, -1} };
				   // No such ^^ Contour index
				   // Just to please the compiler
	// actual quantities of contours per item
	int	pNumberOfContoursInItem[NUMBER_OF_ITEMS] = { 2, 1 };

	// We want 3 frames and 2 circles
	int	pItemsQuantity[NUMBER_OF_ITEMS]		= { 3, 2 };

	// Create the task
	amCATTask* pSampleTask = new amCATTask;

	// Unique ID
	long nID = 0;
```

```
        for( int i = 0; i < NUMBER_OF_ITEMS; ++i )
        {
                // Create nesting item, and set the quantity
                amCATInputItem CurrentItem( nID++, pItemsQuantity[i] );
                for( int j = 0; j < pNumberOfContoursInItem[i]; ++j )
                {
                   int k = pItemContoursIndeces[i][j];
                   // Create a contour
                   amCATInputContour CurrentContour( pAllContours[ k ],
                                                     pContourLengths[ k ] );
                   // Add the contour to the item
                   CurrentItem.addContour( CurrentContour );
                }
                // Add item to the task
                pSampleTask->addInputItem( CurrentItem );
        }

        // Create the domain - a rectangular, length = 120, width = 100
        amCATInputDomain Domain( nID++, 120, 100 );
        // Add Domain
        pSampleTask->addInputDomain( Domain );

        // set the offset distances
        pSampleTask->setItem2DomainDist( 5 );
        pSampleTask->setItem2ItemDist( 5 );

        // Finally create the Facade object
        amCATFacade* pFacade = new amCATFacade( pSampleTask );
        UponCompleteFnc pFnc = TaskIsComplete;
        // Asynchronious call example - launch and "forget"
        pFacade->registerCallback( pFnc );
        pFacade->start();
        // suspend the main thread before exiting for 3 seconds
        Sleep( 3000 );
        return 0;
}
```

Installation and Deployment

A set of files should be added to the installer package. These files combines the core runtime environment of Nesting Factory. Additionally, a ***certificate*** file is sent to user separately to guarantee the uniqueness of the acquired copy.
After the installation, user is required to activate a fresh copy of NF.

Activation
CATInit.exe is required to be launched as a post-install trigger. This executable asks the user for a certificate file. The executable then generates a unique ***ID-file*** and exchanges it with Algomate server for a ***license*** file. If no Internet connectivity is present then the user is asked to send the certificate and id files via e-mail. The license is sent back later. The license file is verified every time a nesting is launched.

Certificate
Certificate file is issued by Algomate. A partner gets a certificate which is distributed to the customers. A certificate has a predefined number of permitted installations and an expiration date. Several certificates may be acquired by one partner.

For a local setup scenario two more API functions are provided

amCATIsValidLicense – returns an integer number with the following meanings

- 0 – the license is valid
- 1 – license is expired
- 2 – inappropriate hardware, i.e. the license was obtained for another computer
- 0xFF – no license file is found

amCATDaysLeftForLicense – returns an integer number with the following meanings

- 0 – the license is expired
- -1 – the license is unlimited
- some other positive number – approximate number of days (1 month "has" 30 days)