Cart_2D Manual

Table of Contents

1.0 Package Content

This package contains VC++ projects and source code for two programs: Cart_2D (directly in the main folder) and Convert_to_gmsh (under to_gmsh folder). As well as some sample input file to run the program out of the box (nested_circles.msh, adaptive_conds.txt, and parameters.txt).

Simply run:

```
Cart_2D nested_circles.msh adaptive_conds.txt output_circles.msh parameters.txt
Convert_to_gmsh output_circles.msh output_circles_gmsh.msh
```

To get the output mesh files output_circles.msh and output_circles_gmsh.msh

List of source files:

```
Cart_2d.cpp  ADT_2d.h  ADT_2d.cc  basic_types_2d.h  basic_types_2d.cc
Polyhedron_2d.h  Polyhedron_2d.cc  to_gmsh/to_gmsh.cc
```

2.0 Building Instructions

Instructions for building binaries from the source files; note that binaries are readily available under the main folder (Cart_2D.exe and Convert_to_gmsh.exe)

2.1 Windows

Open the projects with Visual Studio (Cart_2D.vcxproj, and to_gmsh/ Convert_to_gmsh.vcxproj). Simply hit F7 to build, resulting binary will be located under the /debug folder within each respective folder.
If you're using an alternative compiler or build system, see next section (2.2 Unix instructions).

2.2 Unix

These programs do not need external libraries, any standard c++ compiler will work. I tested with gcc on Mac OS X 10.6.8 and Ubuntu 10.04, 11.10.
Compile and link the source as follows:

```
g++ -c Cart_2d.cpp
g++ -c ADT_2d.cc
g++ -c basic_types_2d.cc
g++ -c Polyhedron_2d.cc
g++ -o Card_2D Cart_2d.o ADT_2d.o basic_types_2d.o Polyhedron_2d.o
```

will produce the binary Card_2D, you may delete the object files with rm *.o

And for Convert_to_gmsh:

```
g++ -o Convert_to_gmsh to_gmsh.cc
```

will produce the binary Convert_to_gmsh

3.0 Running Instructions

Instruction for running the programs, such as input parameters and format. See sample input files formatting requirements.

3.1 Cart_2D

This program generates a mesh from the input parameters.
Usage: Cart_2D InputGmshFile InputRefinementFile OutputMeshFile InputParameters
(all must be present and in this order):
InputMeshFile (GMSH .msh format)
InputRefinementFile (text file, format depend on generation methods)
OutputMeshFile (name of the mesh to be generated)
InputParameters (text file, format see example)

Required fields of input files (all must be present and in this order):
InputRefinementFile:
MAX_LVL_REF, integer, the absolute maximum level of refinement allowed
ALPHA, real, the angular threshold to decide whether a cell should be divided (radians)
InputParameters:
GBOX_X_MIN, real, dimensions of the bounding box, lower bound of x
GBOX_X_MAX, real, higher bound of x
GBOX_Y_MIN, real, lower bound of y
GBOX_Y_MAX, real, higher bound of y
INIT_MIN, real, the size of the initial coarse cell to generate
PROGRAM_GENE_TYPE, integer, 0 is MANUAL, 1 is MIN_SIZE, 2 is ADATPIVE
INTERNAL, integer, 0 is false, 1 is true; whether or not to generate internal mesh

3.2 Convert_to_gmsh

This program converts the output of Cart_2D into a GMSH format for visualization.
Usage: Convert_to_gmsh InputFileName OutputFileName
InputFileName (name of mesh file generated by Cart_2D)
OutputFileName (name of GMSH file to be generated)
will produce the GMSH file named OutputFileName

4.0 Program Step Through

A walkthrough of the execution flow of the Cart_2D

4.1 Alternating Digital Tree (ADT)

A binary search tree that subdivides the items to be searched (geometries) in alternative orthogonal axes; with the median item attached to each node at that level. A key

characteristic of this tree is that a node will only contain a geometry bounded by its region field, which is necessarily inside its parent node's region field. Note that the region associated with a node is NOT the same as the region associated with the geometry on that node.

ADTNode fields:

```
Polyhedron* poly; //the geometry attached to the node
pt4D *region; //the region that this node marks
int depth; //depth of this node in the main tree
ADTNode* L_branch; //children nodes
ADTNode* R_branch;
```

4.2 Execution Flow

See Appendix A for flowcharts, the program starts by check the number of parameters given. If the number is too few it will exit with a usage message. If everything is normal, the program attempts to load the input parameters (box dimensions, initial mesh size, refinement conditions, and whether the mesh is internal).

The geometry is read and stored in an ADT structure, the goal being that this will give a faster search time on the geometry. A dummy mesh is also generated and cropped to the proper dimensions; this dummy mesh is also stored using an ADT structure.

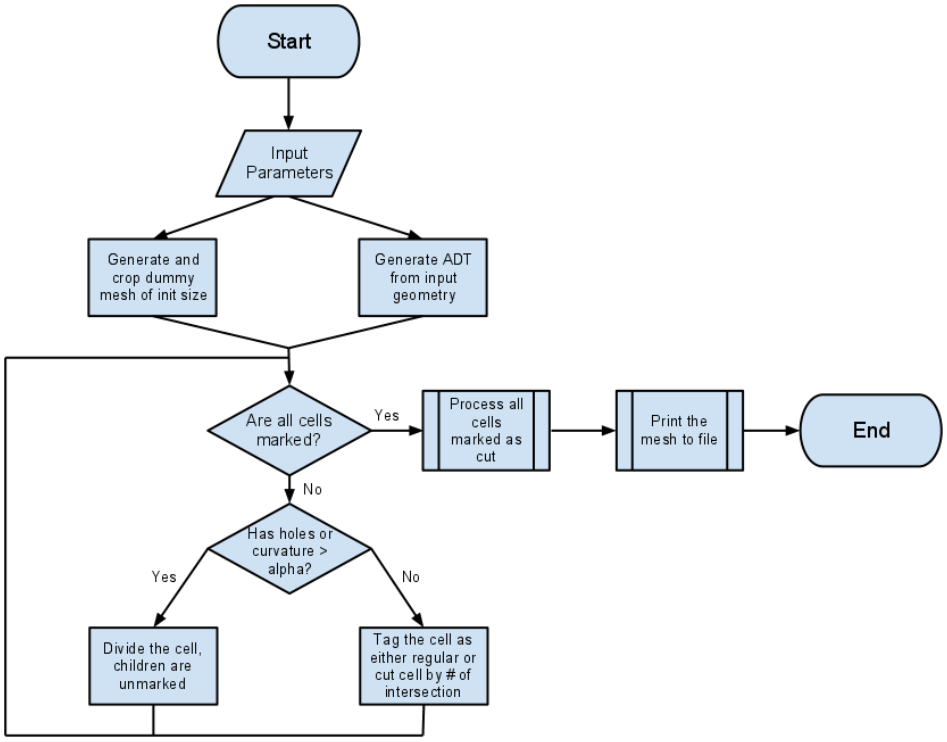
The initial cells in the dummy mesh are unmarked by default, and program begins to process all unmarked cells. Unmarked cells will go through a rough intersection finding function to generate a list of intersecting candidates; the cell is marked as regular if it has no intersecting candidates. If a unmarked non-regular cell has a hole or has an inside curvature greater than alpha, the cell will be divided with its children unmarked; otherwise it is marked as a cut cell.

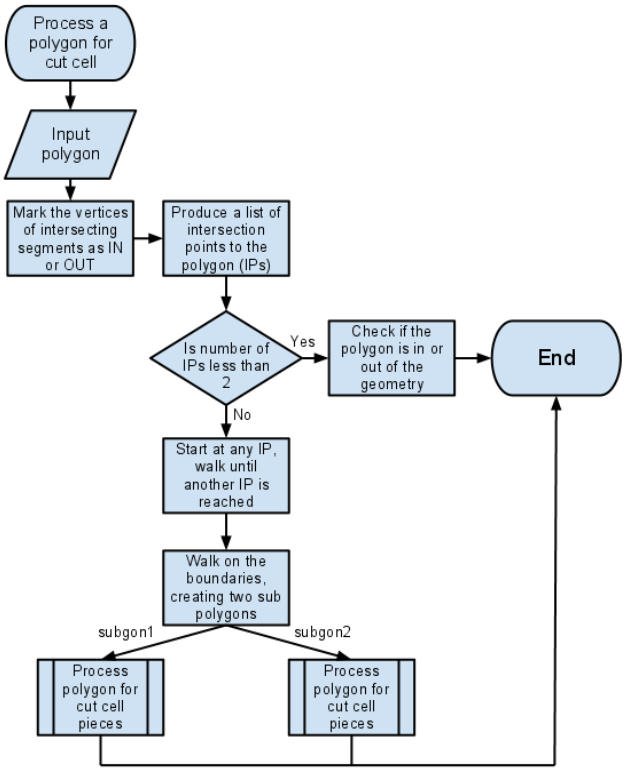
Once a list of all cut cell (candidates) is produced, the program begin to process each candidate and convert them to actual cut cells (tossing out the "inside" portion). This is done by first filtering the candidate list to find real intersecting line segments, and converting all its vertices and edges to the sub-class version (vert_intersection and body_edge), which contains more information (i.e. in/out, on boundary or no, etc). Then performing walks to find two pieces of this cell, this procedure is perform on these pieces until there's no pair of intersection points to further divide the polygon. Each piece will be checked for inside/outside, and unneeded pieces are discarded.

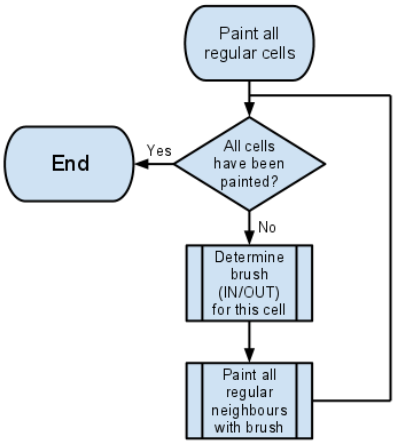
After all the cut cells have been processed, the program performs a painting algorithm on the regular cells. The painting algorithm is simply check a cell for inside/outside, then recursive painting all regular neighbours (then neighbours of these neighbours) as the same status; until there are no more unpainted cells.

And Finally, the mesh is printed with name given in the parameter.

Appendix A: Flowcharts







Appendix B: Key Functions

A brief description of the mechanics of a few key functions in Cart_2D

Intersection Finding (Rough)

This function only attempts to find intersecting candidates of a given region. Since the geometry is stored in an ADT, this can be done very efficiently. Due to the unique property of the ADT, this function checks if the region associated with the geometry in the node overlap with the input region; if yes, it is a candidate and its children are checked for whether or not their associated regions (not their geometry's regions) overlap with the input; the one that does is queued for evaluation. Goto and stack are used to prevent using recursion and causing stack overflow.

Boundary Walk

There are two types of walks available, in-cell walk and cell-edge walk.

The in-cell walk algorithm starts on a vertex that is on the boundary of a polygon, and finds the next vertex inside the polygon that is connected to itself by an edge. The function is then recursively called on the newly found vertex until a vertex on the boundary is found, tracing a path that divides the polygon.

The cell-edge walk is basically the same algorithm, but walks using the edges of the polygon rather than the segments that intersect it. The terminating condition for this walk is if a given vertex is reached.

This two walks performed together in this order will divide a polygon (cell) in two and produce the two children polygons.

Curvature Checking

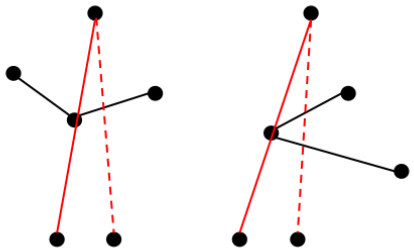
This function checks for the cell's internal curvature, this is done by finding all the in-cell walks of the cell; evaluating the curvature of each walk, and terminate once all walk are exhaust or a offending walk is found.

Each line segment in the walk will have their angles to the x and y axes measured, the curvature of the walk is defined as the deviation of largest angle in the walk and the smallest.

Inside/outside Determination

This function determines of a point (works for regular cell, use any vertex) is inside or outside of the geometry. The ray casting algorithm is used to achieve this, the number of

intersections of a line segment connecting the vertex in question and a vertex outside of the geometry (a corner of the global box) is evaluated. If this number is odd, the point is inside, and even otherwise. This works fine besides a special case where this test line segment intersects two lines in geometry at their point of intersection, which generates a special case that could not be definitively determined. The solution used in the program is to shift the used point (outside the geometry) until the test line has no special case.



Examples of special case scenario being inside and outside (solid) with solutions to the special case problem (dotted)