Description

The repo contains a code that adds skirt to the mesh form *.stl file. It's supposed mesh is topologically correct, contains at least one open boundary loop (a closed chain of edges that bound only one triangle).

The code is implemented from scratch using C++ language (Visual Studio 17 project). It contains several low-level things like:

Connected mesh (Mesh.h) – my version representation of connected mesh. It's my service class to simplify work with triangulated mesh. The speed and simplicity is achieved by several excessive entities and fields. In contrast with the most effective QEdge representation edges knows about the second vertex (can be omitted), vertices store the list of all edges, also there are faces that store 3 edges. In addition, the set of "orphan" edges i.e edges that are contacting with the only one face is preserved as well. It allows to read stl files on the fast manner and add new triangles to the mesh without full search of contacts along the edge list (in assumption of the topological correctness.) Even though I haven't implemented any visualization the class contains a function that exports information into collections compatible with vertex list OpenGL 5.0 representation so can be more or less easily added to any renderer.

Vector.h and TransMat.h – auxiliary classes. Vector contains a template vector3 class (in future I'm going to play with GMP on this platform). TransMat – is a rigid transformation matrix that allows to define rotational transformation (by a new basis) or shift. For both only necessary functions are implemented.

StlImport.h – functionality of import and export for stl files. I used an opensource code for the stl import since it doesn't matter for the geometrical algorithm. The program can read either text or binary representation but exports the text representation only (it's the longest part during the program work).

I'm going to have additional extensive development for these classes in order to share them later as an open-source project MeshSandbox to allow everybody (and me outside any organization eco-system) to have a base functionality for experiments with algorithms without usage of any large size libraries.

The rest of the file set is a business logic.

Skirt_Win.cpp is a main function. Technically it's a command line application with call like

Skirt_Win.exe f:\Data\example.stl **or**
Skirt_Win.exe f:\Data\example.stl -opt:OFF

The rest of settings like technological settings is hardcoded in the structure:

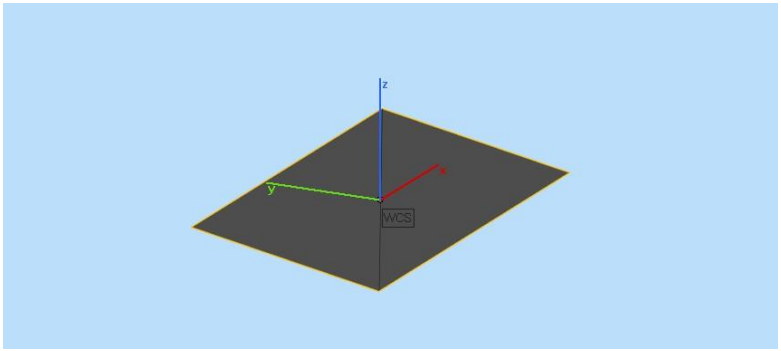
```
SkirtBuilder::Data data;
```

If it were solution for the real work I would add some yaml or json file with settings but I don't want either use huge general-case parser or write lousy mine here. So, the assumption is the technological process parameters are predefined. The result (if any) or error dump will be saved to the same folder with initial .stl file. Result will have an "_out.stl" part after the initial filename.

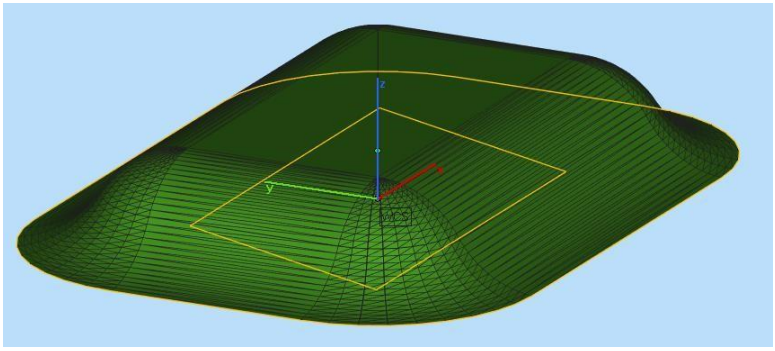
PositionOptimizer.h is a primitive prototype of the functionality that optimizes the mesh orientation. The approach here is the source mesh can be produced by traditional stamping i.e all faces have normal

in the same hemisphere. So, the functionality simply gets the mean normal and orients one along z axis. Also, it has a simple check of the geometry and topology correctness. The “orphan” edges chains must be closed and all faces on boundary must have normal that have positive dot product with the z axis. In case of fail it dumps results – temporary mesh in the stl file and other stl file with problematic face. In such a case the optimizer can be switched off by -opt: OFF setting. The user is in response for the correct orientation in such a case.

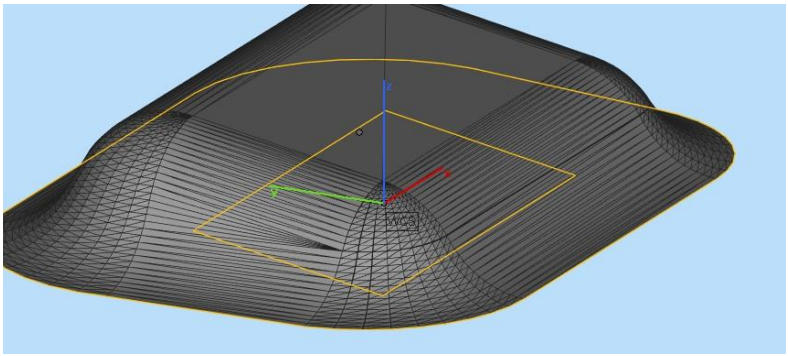
The simple example of the work of optimizer for the inclined plane.



Source



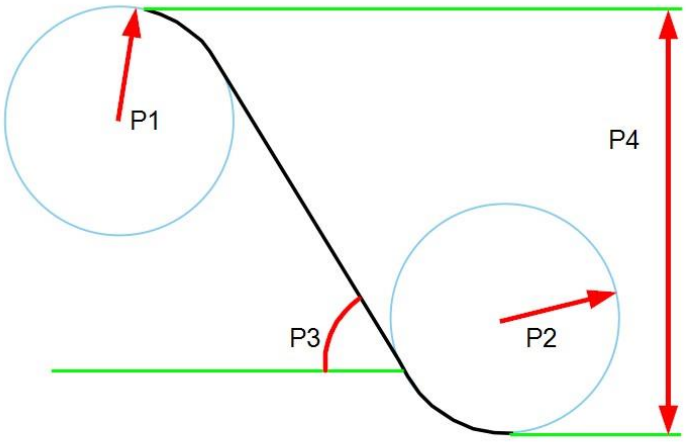
Optimizer is on



Optimizer is off

For the real application optimizer can be enhanced to provide the most technologically reliable orientation and the lower material consumption as well (by the way, despite the stupidity the current algorithm is quite optimal from the material consumption point of view).

SkirtBuilder.h – functionality that builds a skirt for the properly oriented mesh. As it's visible from previous pictures skirt contains several parts – upper blending that connects the mesh surface with straight slop part on the smooth manner (it's an arc of circle so it isn't C2 surface), straight part and lower blending area that smoothly connects the straight part with a horizontal plane.



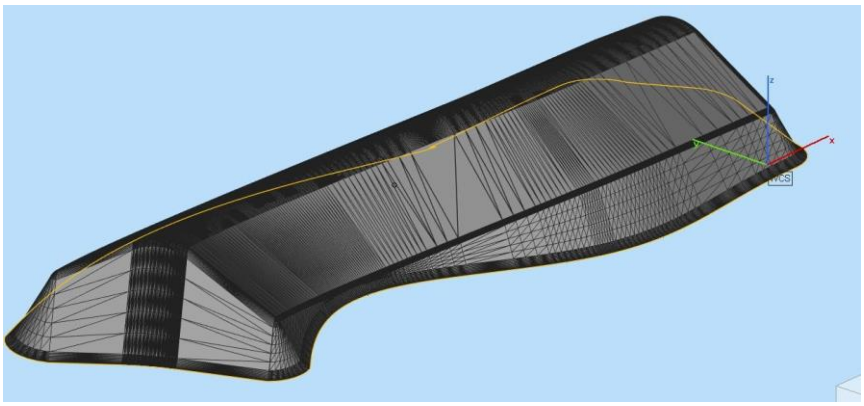
The parameters are defined by the
`SkirtBuilder::Data data;` structure
where:

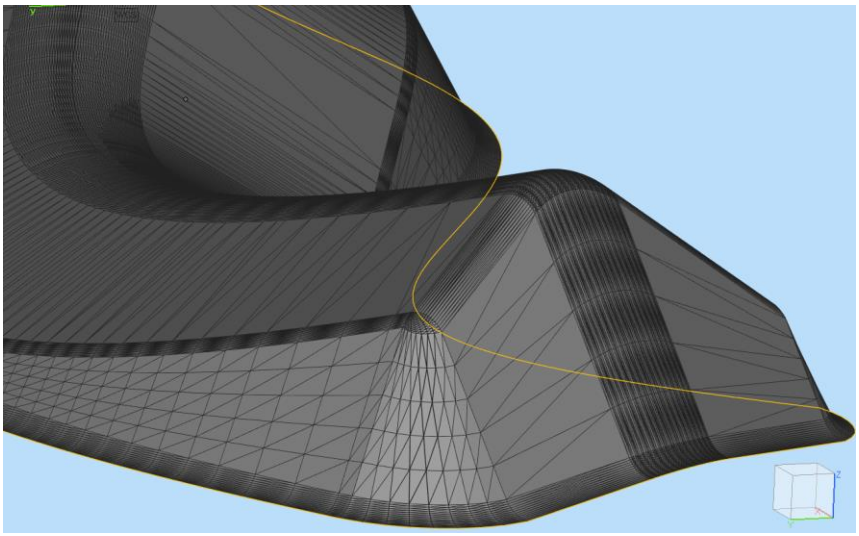
```
struct Data   {     double upperSmoothR = 5.;   // upper
smoothing radii P1     double lowerSmoothR = 5.;   // lower
smoothing radii P2
    double minimalHeight = 0.1;  //minimal distance from any point to the bottom P4
double slopeAngle = PI / 3; //slope angle (from horizontal plane in radians) P3   };
```

All parameters are in units of stl file except the slope angle (radians). The data structure is hardcoded but, again, any parameter can be change.

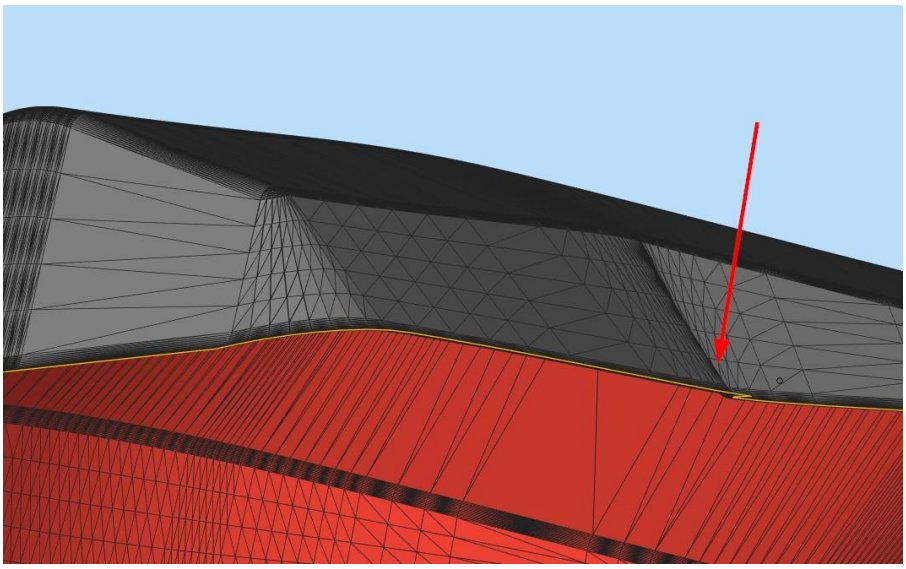
The physical sense of minimal height is a minimal distance of any point of the initial mesh to bottom. If both blending parts exceeds this height the real minimal height will be evaluated as a sum of two arcs for the point of the initial mesh boundary that is closest to the bottom plane. For such a point the straight part will be generated (since I implemented only trivial meshing functionality that preserves number of triangles for any "vertical" strip of skirt) but it will have a minimal enabled length.

NOTE: the problem I haven't solved is degeneracies of mesh due to folds. It needs significant labor and I'm not ready to do it in my free time on the fast manner.





Part that was proposed for the challenge with the skirt



Fold

There are two possible ways to solve it.

1) Optimizer functionality. It can be enhanced to search positions without folds. The result is not guaranteed
2) “Quasi” Boolean operation. For any triangle we seek intersection with others (except topological neighbors). After that they are used to split the triangle onto several parts. Such a parts are used as a “polygon soup” where only “external” triangles are preserved. The triangles that are closed

   by the “external” ones are extracted. The time for such a development is 2-3 weeks in case of full workday load so I’d prefer not to do it in at my free time.

I also added an .exe file. It’s built for 64 bit windows system and can work as is, but it might need VS17 redistribution pack (I cannot check it since all my computers have one). The size of the exe is 99 kb and the most time-consumptive operation for the proposed part is an export to the stl file. (That’s why I don’t use Python)

Also you can find the stl files I used for the debugging and some output in the ‘Data” folder.

Future of the project: since I spent significant part of my free time for this project I’m planning to enhance the “low level” part. To introduce rational arithmetic, to add a unit tests and, later, to add a QT based viewer. Finally, the idea is to make an opensource MeshSandbox functionality to allow everybody to have toolkit for some experiments and mesh debugging. The skirt *.stl example as an yours intellectual property will never be shared by me.

Maxim Prut