CG Programming III – Assignment #3 (half-edge data structure)
Due on 03/02/2011

In this assignment you will be required to implement a simple subdivision surface using a half-edge data structure. Skeleton code will be provided in class. In the sample application, when the user presses the "s" key, the function `subdivide_mesh` is called. This function should:

- Create a new mesh structure.
- Iterate over the *polygons* of the old mesh structure. The providied `triangulate_mesh` function in `he_to_patch.cpp` shows a way to do this.
- For each polygon in the old mesh:
  - Make copies of the vertices that made up the old polygon in the new mesh. Each vertex should be copied only once! Each vertex in the old mesh will be shared by multiple polygons, so some extra bookkeeping is needed to prevent adding vertices multiple times.
  - Create new vertices at the midpoints of each edge of the old polygon. Each new vertex should be created only once! Each edge in the old mesh will be shared by tow polygons, so some extra bookkeeping is needed to prevent adding vertices multiple times.
  - Create edges edges connecting the points.
  - Create four new polygons bounded by the new edges.

You should only need to modify code in `tesselate.cpp`, `tesselate.h`, and perhaps `he_to_patch.cpp`. It is likely that you will need to subclass `half_edge`. If you do this, you will also need to subclass `edge_factory`.

Criteria	Excellent	Good	Satisfactory	Unacceptable
Completion	Program correctly implements all required elements in a manner that is readily apparent when the program is executed. User interface is complete and responsive to input. Program documents user interface functionality.	Program implements all required elements, but some elements may not function correctly. User interface is complete and responsive to input.	Program implements most required elements. Some of the implemented elements may not function correctly. User interface is complete and responsive to input.	Many required elements are missing. User interface is incomplete or is not responsive to input.
Correctness	Program executes without errors. Program handles all special cases. Program contains error checking code.	Program executes without errors. Program handles most special cases.	Program executes without errors. Program handles some special cases.	Program does not execute due to errors. Little or no error checking code included.
Efficiency	Program uses solution that is easy to understand and maintain. Programmer has analysed many alternate solutions and has chosen the most efficient. Programmer has included the reasons for the solution chosen.	Program uses an efficient and easy to follow solution (i.e., no confusing tricks). Programmer has considered alternate solution and has chosen the most efficient.	Program uses a logical solution that is easy to follow, but it is not the most efficient. Programmer has considered alternate solutions.	Program uses a difficult and inefficient solution. Programmer has not considered alternate solutions.
Presentation & Organization	Program code is formatted in a consistent manner. Variables, functions, and data structures are named in a logical, consistent manner. Use of white space improves code readability.	Program code is formatted in mostly consistent with occasional inconsistencies. Variables, functions, and data structures are named in a logical, mostly consistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted with multiple styles. Variables, functions, and data structures are named in a logical but inconsistent manner. Use of white space neither helps or hurts code reability.	Program code is formatted in an inconsistent manner. Variables, functions, and data structures are poorly named. Use of white space hurts code reability.
Documentation	Code clearly and effectively documented including descriptions of all global variables and all non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of most global variables and most non-obvious local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted, as are the input requirements and output results.	Code documented including descriptions of the most important global variables and the most important local variables. The specific purpose of each data type is noted. The specific purpose of each function is noted.	No useful documentation exists.

This rubric is based loosely on the "Rubric for the Assessment of Computer Programming" used by Queens University (http://educ.queensu.ca/ compsci/assessment/Bauman.html).