Data structures & Algorithms for Geometry
Assignment #2 (BVH frustum culling)
Due on 10/27/2007

For this assignment you will implement *and test* a bounding volume hierarchy class. The class must adhere to a predefined interface. Code for the base class and several supporting classes (`frustum`, `idr_vec4`, and `idr_mat4`) will be provided.

- Implement a subclass of the `bounding_volume_hierarchy` class.
- The bounding volume hierarchy subclass will require the existence of a bounding volume class. Any bounding volume type that we have studied in this course may be used.
- For this class you must implement the contructor, the `transform` method, and the `intersect` method for frustums. You do *not* need to implement BV-BV intersection tests or BVH-BVH intersection tests.
  - Any of top-down, bottom-up, or insertion methods may be used to create the bounding volume hierarchy.
  - Implement *two* different criteria for selecting node subdivision (for top-down), node merging (for bottom-up), or node code (for insertion).
  - Analyze the performance of the resulting BHV for some (random) data sets. The analysis does not need to be arduous. Simply state, with numbers, which performed better with some brief thoughts as to why.
- Create a set of tests for the BVH-frustum intersection test. Please provide, as comments in the code, explanation of the test cases.

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).