# Data structures & Algorithms for Geometry
# Assignment #4 (BSP tree)
# Due on 12/1/2007

In this assignment you will implement *and test* a BSP tree. This tree structure will store polygons with the intention of accelerating ray-polygon intersection tests.

- Implement a `build_tree` method.
  - This method will be passed a `vector` of pointers to `triangle` structures. It will create a tree from that data.
  - A more sophisticated mesh structure will be needed to implement the polygon splitting routine.
  - The split-plane selection need not be automatic. See the lecture notes from 11/10 for some other suggestions.
- Implement an `intersect_ray` method. This method will return the first point of intersection with the specified ray.
  - If there are multiple intersections, the intersection closest to the origin of the ray is considered to be the first.
- Implement a split-plane selection method based on the least-split and blanced-cut methods with most-split as a tie breaker. Make the weighing of least-split vs. blanced-cut a tunable parameter when the tree is created (i.e., a parameter to the tree constructor).
- Implement subtree-nodes.

Using the convex hull generator from assignment #3 will provide a useful basis for testing the BSP tree. Generate a convex hull form a random set of points and bulid a BSP tree form the hull surfaces. Select a point inside the hull, $p_i$, and a point on the hull, $p_h$. $p_i$ is the origin of the test ray and $p_h - p_i$ is the direction. The intersection of this ray and the hull should be very close to 1.0.

Extra credit will be awarded for implementing *compacted complete subtrees* or *fused linear nodes.*

| **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).