CS 211 – Programming Practicum – Summer 2020
Programming Project 2

Due: Monday May 25th at 12:00 noon. This assignment is to be handed in via Gradescope.

For this project you are to write a **C program (NOT C++)** to maintain a collection of points in 3D space, and certain properties of the collection as a whole. Then you will use this information to test whether a given separate point intersects with the collection of points or not.

Background

- In computer graphics one important task is to maintain information about points in 3D space, a.k.a. vertices. It is also important to maintain groups of these points, such as the vertices comprising a particular 3D object, and certain properties of the group of vertices as a whole.
- One such property of the group is the extent or size of the group. This is often maintained as the dimensions of a "bounding box" – the smallest box with sides parallel to the axes that would encompass all of the points within the group. This is kept as the minimum and maximum values of the X, Y, and Z coordinates of any vertex within the group.
- A simple but effective collision detection mechanism for determining if a given point intersects with a collection of points is to see if the given point is inside the bounding box of the collection. (This is often done as a first quick collision test, followed by more rigorous checks if the point is found to be within the bounding box of the collection.)

Point3d Data Type

Employ **struct** and **typedef** to create a data type called ***Point3d***, having the following properties:

- int ID – A unique integer serial number for each vertex. These may be used later for specifying which vertices make up a polygon for example.
- double X, Y, Z – These are the coordinates of the point in 3D space. (**<u>Alternative:</u>** You may declare an array of 3 doubles named "coordinates" instead of X, Y, Z if you wish.)
- double luminosity – A measure of the "brightness" of the vertex, in the range 0.0 to 1.0. (In practice computer graphics keep track of a LOT more information at every vertex.)

PointCloud Data Type

Employ **struct** and **typedef** to create a data type called ***PointCloud***, having the following properties:

- Point3d * points – This is a dynamically allocated array holding the points in this particular PointCloud. There is no specified limit on the number of points in a cloud, so this will have to grow dynamically as was done in project 1.
- int allocated – This keeps track of how much space has been allocated in the points array.
- int nPoints – Keeps track of how many points currently exist in the points array.
- double luminosity – This is the average luminosity of the points in the point cloud. (If the cloud has no points, then this should be zero.)
- double xMin, xMax, yMin, yMax, zMin, zMax – These keep track of the bounds of the bounding box of the point cloud. (If the cloud has no points, then set these all to zero.) (**<u>Alternative:</u>** You may use arrays here instead of 6 fields if you wish. If so, you will have to choose between two arrays (lower bounds[3] and upper bounds[3]) or 3 arrays(xBounds[2], etc.))

Command Line Arguments

- Your program **must** check for the presence of "-d" as a command line argument, and if present it should put the program into ***debug mode***. (See below.)
- Your program **may** accept a file name as a command line argument. See optional enhancements.

Debug Mode

- If the program is in ***debug mode***, then additional information should be printed out as the program runs, tracing operations.
- Ordinarily global variables are evil, and should be avoided at all costs. In this case, however, you will be allowed to create a single global boolean "DebugMode", and set it to True if the "-d" flag is present as a command-line argument, or False otherwise. Then check this variable when deciding whether to print tracing information or not.

Program Operations

- The program must first read in data for a series of points, normally from standard input. For each point read in, a new Point3d should be created, and added to a PointCloud. This continues until a point with negative luminosity is encountered (which should not be stored.)
- The program needs to determine and store the upper and lower bounds of the bounding box, and the average luminosity of the point cloud, and store these in the PointCloud structure. This can be done either while reading in the input, or as a separate step after all the point data has been read in.
- Then the program needs to read in a second set of points, and for each of these points report whether this point intersects with the PointCloud or not. (Any point that is exactly on a boundary of the PointCloud should be considered as intersecting the PointCloud.) This continues until a point with negative luminosity is encountered (which should not be processed.)

Program Input

- The input to this program is normally read in from standard input, which can be redirected from a file as in project 1. The format specifier for a double in scanf is `%lf` instead of `%d`.
- Each point will be represented by four floating-point numbers, representing the X, Y, and Z coordinates of the point, followed by its luminosity.
- Each point should be assigned a unique integer ID, starting at 0 for the first point and increasing by 1 for each consecutive point. (These will correspond to their array locations initially, but will remain unchanged if the points are reorganized into different groupings or storage structures.)
- Each set of points will be terminated by a point with a negative luminosity, and this last point should be ignored. (Normal values for luminosity are 0.0 to 1.0 inclusive.)
- There are no guarantees regarding how many points will be included in the input, or how the data is split among lines. (The data for a point may span multiple lines, or there may be data for more than one point on a single line. There may or may not be blank lines anywhere in the input.)

Program Output

- The program should report the number of points in the point cloud, the bounds of the bounding box, and the average luminosity of the cloud.
- For each point in the second set of points, the program should echo the point coordinates and report whether it intersects with the cloud or not.

Required Methods

At a minimum the following methods will be required:

- int addPointToCloud(PointCloud * cloud, Point3d point); - Return value is 0 if the point is successfully added to the cloud, or -1 otherwise.
- bool intersection(PointCloud cloud, Point3d point); - Return False if the cloud has no points. Otherwise check if the coordinates of the point are within (or equal to) the bounds of the cloud, and return True if there is an intersection or False otherwise.

Other Notes

- **No global variables are allowed**, with the unusual exception of the DebugMode boolean. In particular the PointCloud must be declared in main and passed around as needed. (The definition of the data struct **types** must be global, so they are known to all functions, but the variables themselves are all local variables.)
- All good programming practices should be exercised, including proper commenting, variable naming, loop indentation, use of functions, and so on.
- **All dynamically allocated memory must be freed before exiting the program.** This last point will be tested using valgrind when evaluating your program. (And for all future projects.)
- malloc( ) returns NULL when it is unable to allocate the memory requested. It is good practice to check the return value from malloc before using it, and exiting with an error condition if malloc returns NULL.

Optional Enhancements

Optional enhancements allow students to take assignments a little bit farther, generally on their own without instruction or guidance. Optional enhancements can never raise a score above 100, but may compensate for deficiencies in other areas. It should also be possible to earn 100% credit for an assignment without doing any optional enhancements, **IF** everything else about the program is **perfect**. Here are a few ideas for this assignment:

- If a file name is provided as a command line argument, read data input from this file **instead of** reading from standard input.
- Calculate and report the center of gravity of the PointCloud, defined as the average of the X, Y, and Z coordinates of the points in the PointCloud. Store and report the value. (This is a pretty trivial enhancement.)
- Create a second PointCloud from the second set of points read in, and write a method to detect the intersection of two PointClouds. Report the results.
- Calculate and store the center of the bounding box. Write a method to calculate the distance between a given point and the center of the box, and report this information when doing intersection testing.
- Identify the point in the point cloud closest to a given point when doing intersection testing, and report this point and the distance between them when doing intersection testing.
- Determine a bounding **sphere** for the PointCloud in addition to the bounding box, defined as the center and radius of the smallest sphere that would contain all the points of the PointCloud. This is actually much harder than it sounds, as the center of the sphere is not necessarily the same as the center of the bounding box. Use this sphere to detect and report intersection testing, and compare the results to the bounding box approach.