Lab 3: Rendering Wireframe Images Using Bresenham Algorithm

Topics Covered

- Viewport transform
- Rasterization of line segments
- DDA Line Rasterization Algorithm
- Bresenham Line Algorithm

Prerequisites

- Correct implementation of Lab 2

Implementation Details

1. Implement a function `Pbo::render_linebresenham()` with declaration:

```
struct Pbo {
/*
other declarations here ...
static void render_linedda(int x1, int y1, int x2, int y2, Pbo::Color clr);
*/

  /*
  set all pixels with same color clr on line segment starting at
  point P1(x1, y1) and ending at point P2(x2, y2) using the integer-only
  Bresenham line algorithm.
  The parameters representing 2D vertex coordinates are specified in
  window coordinates.
  Your implementation must not use non-integral constants, variables, or
  expressions.
  */
  static void render_linebresenham(int x1, int y1, int x2, int y2,
                                   Pbo::Color clr);
};
```

2. In `Pbo::render()` , replace the call to `Pbo::render_linedda()` with a call to `Pbo::render_linebresenham()`.
3. The beauty of the Bresenham algorithm comes from its amazingly simple method to replace floating-point operations with integer-only arithmetic. To appreciate its simplicity and beauty, your implementation must not use ***any*** non-integral constants, variables, or expressions. The entire code must be based on integer-only arithmetic. Use the class presentation and handout to understand and implement this seminal algorithm.

4. Amend `Pbo::init()` so that `DPML::parse_obj_mesh()` will parse an OBJ file named `lab3-test.obj`.

Submission Guidelines

1. Check the assignment web page for the submission deadline.
2. ***Only*** files `pbo.h` and `pbo.cpp` must be submitted - additional files will not be used for grading. Changes to `main.cpp`, `app.cpp`, `glslshader.cpp`, and shader source files are not required nor necessary for this submission.
3. Open the rubric page (in Word format or in PDF format) and check the portions that are complete. Sign and date the bottom of the page. The document can be submitted as a PDF or JPEG or Word document. I cannot grade your submission if the signed rubrics page is missing.
4. Copy submission source files and signed rubric page into a submission folder that is named using the following convention: <**student login name**>_<**class**>_<**lab#**>**.zip**. Assuming your login is **foo**, the course obviously is **cs250**, the lab is **3**, then the zipped folder would be named: **foo_cs250_lab3.zip**.

Directory Layout

The number of shader source files and geometry files will increase as the semester progresses. To avoid unnecessary back and forth during the grading process, following conventions laid out in the "Setting up your OpenGL framework" document will make grading simpler to manage. The "Setting up your OpenGL framework" document describes the following directory layout:

1. Let's say `CS250` is the solution folder identified by Visual Studio macro `$(SolutionDir)`. The `CS250.sln` solution file is located in this folder.
2. To avoid duplication, data common to all projects must be located in `$(SolutionDir)`. The directory containing shader source files called `shaders` and the directory containing OBJ files and other mesh data called `meshes` must be located in this directory.
3. Let's say a new project called `Lab1` is required. Make Visual Studio create a new folder called `Lab1` to store all files relevant to just this project: project file `Lab1.vcxproj`, source files in directory `src` and header files in directory `include`.
4. Similarly, make Visual Studio create a new folder, say `Lab`2, to store all files relevant to a second project called `Lab2`.
5. Visual Studio dumps release mode executables `Lab1.exe` and `Lab2.exe` into directory `$(SolutionDir)Release` and corresponding debug mode executables into directory `$(SolutionDir)Debug`. These programs can now be executed by going into these directories and double-clicking the icons, or by using **Ctrl-F5** in Visual Studio, or even from the command line.
6. To access shader source files in `$(SolutionDir)shaders` and OBJ files in `$(SolutionDir)meshes`, release mode executables in `$(SolutionDir)Release` must locate a specific OBJ by moving up from `$(SolutionDir)Release` to base directory `$(SolutionDir)`, change directory to `meshes`, and then access the particular OBJ file. This means that if the executable must open and parse OBJ file `model-to-test.obj`, then C/C++ file I/O functions must be provided string `"../meshes/model-to-test.obj"` as an argument.
7. Similarly, to load a shader source file `basic-texture.frag`, shader I/O functions must be passed string `"../shaders/basic-texture.frag"` as an argument.
8. One final test: copy your executable to the same directory in which the sample is located - both the sample and executable should run.

9. If you wish to follow some other conventions, then all I require is that your submission will somehow manage to compile, link, and execute correctly with the framework described above.