2D Drawing Tutorial

In this tutorial, you will start working with he T3D graphics engine, and write code to draw lines and circles.

You should be able to get by with your knowledge of C from previous units, but it would be better to have started Self Study: Maths Refresher.

Getting Started with T3D

1. Create a new folder in your home directory called Tutorial 1.
2. Go to the T3D GitHub page and download the zip folder (Green Clone or Download button) to this directory and unzip it (alternative
3. Unzip "T3D" and open the resulting folder.
4. Find the "T3D.sln" file and double click to launch Microsoft Visual Studio and open the T3D project.
5. Browse and open project files using the "Solution Explorer" panel.
6. At the bottom of the Solution Explorer, you should see a file called "Main.cpp". Open this file and observe that it is the main entry poin
7. Run T3D now by clicking on the play button (green right-facing triangle) in the toolbar at the top of the window (Ignore any warnings th
8. Modify Main.cpp so that instead of running T3DTest it instead runs the Tutorial1 application.

Overview of the Tutorial1 application

The application places a 2D overlay over the entire screen, and starts a Task that can be modified to draw onto that overlay.

The overlay is called drawArea and is of type Texture. Texture has a plotPixel method, and that's all we will be using in this tutorial. The screen (and texture) size is currently hard coded at 1024x640, but that may change in the future.

The task is called DrawTask. This class has an init method that currently clears the screen and draws a line using the drawDDALine method. The drawDDALine method works for these parameters, but not for other choices of start and end point. There is also a drawBresLine method that does nothing.

The update method has a commented line to clear the draw area (this can be uncommented for animation), and renderer call to reload the texture so that any new data is uploaded to OpenGL.

Note: This is not a recommended way to do 2D drawing with OpenGL, it just a simple method of testing our drawing algorithms

Drawing Lines

Review the Lecture Content

The DDA algorithm was covered in Lecture 02 2D Drawing.pptx.

Make sure that you are familiar with these concepts before continuing. You may also be able use the code in the lecture slides as a startin

Setting Up a Test Case

A great way to start with many programming tasks is to set up a test case. So, let's do that for our drawDDALine algorithm.

1. Add some drawDDALine calls to the *init* method to test all possible configurations of parameters.
   - e.g. first point at the bottom left with shallow slope, first point at the bottom left with steep slope, etc.
   - There are at least 12 different arrangements that you should test.
2. Modify your 12 lines so that they are all draw with a different colour (so that you can see which ones are working correctly and which
3. Confirm that most lines are NOT drawn correctly.

Fixing the DDA Code

1. Using the notes from the lectures, you should be able to work out how to modify the drawDDALine method, so that all tests pass.
    - There are two approaches to ensuring that all 12 cases are covered
    - The easiest approach requires selecting between four different 'for' loops
    - But it can be done with just one or two 'for' loops
2. When you have it working, check with your tutor to make sure that you haven't missed anything.

Bresenham's Algorithm

1. Change all of your drawDDALine method calls to drawBresLine calls with the same parameters.
    - When you run this nothing should be drawn
2. Use your lecture notes (and maybe a bit of Googling) to try and get Bresenham's algorithm working.

Remember!

Bresenham's Algorithm should not use any floating point variables.

Drawing Circles

Review the Lecture Content

Drawing circles was also covered in Lecture 02 2D Drawing.pptx.

You should be aware of the trigonometry method, and the method that uses Pythagoras.

With Trigonometry

1. Add a drawCircle method that takes a centre, radius and colour as parameters.
2. Copy the code from the lectures for drawing a circle using trigonometry
3. Experiment with different radii and different step sizes
    - Is there an ideal step size?

Optimisation

1. Modify your code so that you mirror each quadrant to make the algorithm 4x faster.
    - If you write a loop to draw many random circles, you should be able to see the speed-up.
2. Modify your code so that you mirror each quadrant to make the algorithm 8x faster!

With Pythagoras

1. Modify your code so that you use Pythagoras instead of trigonometry.
    - Can you observe any further speedup?

Bresenham's Circle Algorithm

If you still have time, you could try searching for Bresenham's circle algorithm, and try implementing that to see if you can get any further imp