COMP612 Computer Graphics Programming

Semester 2, 2024

Assignment 1

Snow Scene

This is an **individual** assignment. All work you submit must be entirely your own. The assignment is worth 30% and marked out of 50.

- You **must** work from the provided `animation.c` template.
- You **must** complete the assignment in procedural C and freeGLUT (**NOT** C++).
- You **must not** use any external libraries other than freeGLUT.
- You **must** use Visual Studio 2022 and build/release an x86 (Win32) project.
- You **must** adhere to the guidelines for use of the template provided below.
- Your final submitted code **must** compile, have graphical output, and run.
- **All the above criteria** must be met for the assignment to be graded.

It is expected that you will work consistently on this assignment, from hand out to due date. Time will be allocated in classes each week for you to ask questions, get help, and work on your assignment. Please be aware that this is not an assignment that can be completed at the last minute. Each class you will be exposed to new concepts, and as we work through these concepts you will be able to progress your assignment. It is estimated that this assignment will take 20 hours to complete.

Generative AI Policy:

You **must not use generative AI** (e.g. ChatGPT) **for any part of this assignment**. This includes research, code, ideas, text, design notes and creative assets such as images and icons.

Where your implementation idea or inspiration has been taken or adapted from other published sources (e.g. a tutorial on drawing teardrops) those sources **must** be acknowledged appropriately in the C file header comments and detailed fully in your developer's logbook.

Assignment Due Date: **Monday 19th August, 10am** (week 6)

Submission:

Your submission must compile and run for it to be graded. Any submissions that cannot compile and run will be awarded a mark of zero.

You must submit your assignment via the link provided on canvas. Your submission must contain the following:

- A single zip file of:
  - Your Visual Studio solution, including all files necessary to build and run the animation.
  - Your logbook entries, either as
    - A single PDF file of the pages scanned from your handwritten logbook, or
    - Your digital logbook saved as a PDF file.

Late Policy:

Late assignments, without an approved extension, will be subject to a deduction of 5% (one grade e.g. from C+ to C) of the total mark available for each 24-hour period, or part thereof, up to a maximum of five calendar days. Assignments over five days late will not normally be accepted or marked, and students will receive a DNC (Did Not Complete) for that assessment.

Unauthorised Collaboration

Unauthorised collaboration means joint effort between students or students and others, in preparing material submitted for assessment, except where this has been pre-approved by the paper programme. Students are encouraged to discuss matters covered in classes, but the expression of ideas and arguments must be the student's own work.

Plagiarism:

Please be aware that any piece of your assessment may be tested with plagiarism prevention software.

Assignment Objectives:

- Basic 2D animation programming
- Keyboard interaction
- Bitmap font rendering
- Simple Particle Systems
- By-Vertex attributes
- Alpha blending

Getting started:

Download the animation template from Canvas (`animation.c`). This provides the FPS timing that is needed to avoid overworking your CPUs and GPUs (this is especially important if your machine is a gaming laptop – without it you will use one full core of the CPU and ~90%+ of your GPU). You **must** work from this template.

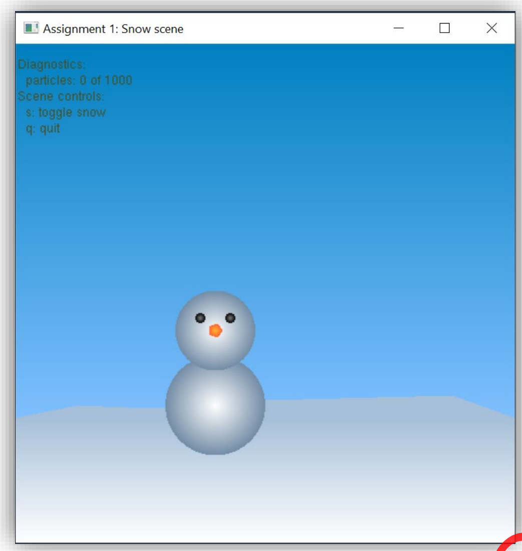
On Canvas there is a video of my basic Snow Scene. It shows how your snow particle system should behave.

Using the Template

Details of how to use the template for this assignment are given in the video lecture "8- Animation timing with GLUT" from around ~ 35.5 in the 40-minute video. We will also go through this process in class using a demo project. Don't forget to update an animate variable each frame and multiply this by FRAME_TIME_SEC to ensure that your frame rate is controlled at the fixed rate (e.g. 60FPS).

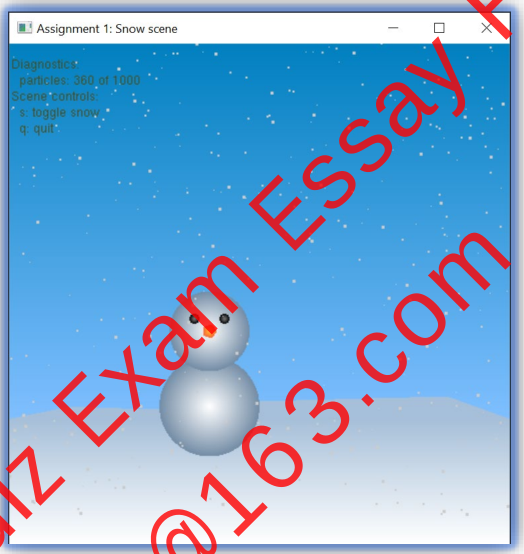
2D scene – Snow Scene

Below are screen shots from my basic demo scene. You will add more detail and features to your scene. You are free to choose any colours, snowman design, etc. that you wish to. My snowman is incomplete and very sad as he has no stick arms.



Assignment 1: Snow scene

```
Diagnostics:
  particles: 0 of 1000
Scene controls:
  s: toggle snow
  q: quit
```



Assignment 1: Snow scene

```
Diagnostics:
  particles: 360 of 1000
Scene controls:
  s: toggle snow
  q: quit
```

1. Ground [3 marks]

The ground in your scene should be formed by an **irregular** polygon. Each time the program is run the vertices that from the top of the ground should be randomly generated. You should use by-vertex colour to give your ground some perspective.

2. Sky [2 marks]

Add a sky to your scene. Make use of by-vertex colours, rendering order, and transparency to get the look you want. (Don't use the GL clear colour).

3. Snowman [5 marks]

To draw a snowman, you will write a utility function to draw a 2D circle, the algorithm to do this is given in the basic geometries section of this document. You may not use GLUT or GLU libraries to do this but instead must use the basic 2D primitives from the GL library. Using by-vertex colouring you can get a nice 3D effect. Add things to your snowman (eyes, nose, stick arms, a scarf etc. – the design is up to you. You may want to have a snowman made up of three snowballs rather than two for example.

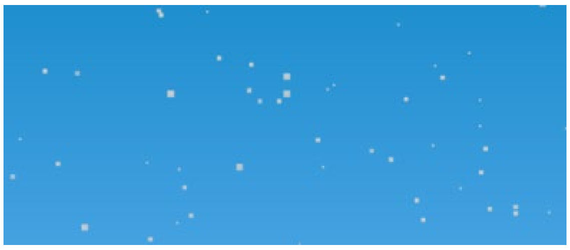
4. Snow [10 marks]

Develop a particle system to simulate snow. You will find that the number of particles you require is finite and you can reuse existing particles for efficiency. If you do not either kill and remove or recycle the particles you will eventually either run of memory or run out of space in your particle system's data structure. Each time a particle reaches the ground and it is recycled, it should be randomised again. If you want to leave the particle on the ground for a while, then you will still eventually need to remove or recycle it.

For snow the particles are generated so that they appear in random locations on the x axis. The particles are also fed into the system slowly over time until the maximum number of flakes has been reached.

Each snowflake (or particle) is rendered as a single 2d point that moves downwards. For more realistic snow you <u>must</u> randomly determine at the time you spawn a particle its:

- ❄ Size (within a visually acceptable range)
- ❄ Transparency (within a visually acceptable range)
- ❄ Fall Speed (based in some way on size and gravity)



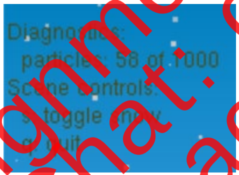
Other possible improvements that you may like to include are:

- ❄ Adding a small shake (brief x position shift)
- ❄ Adding the effect of wind drift
- ❄ Varying the colour of flakes depending on their sizes
- ❄ Slowly making the flake more transparent (or smaller) as it falls
- ❄ Draw some snow behind the scenes' objects and some in front (in my demo it is all drawn after/in front of the snowman).

5. Animation control: [6 marks]

Allow the user to turn snow on and off by pressing the "s" key. When snow is turned off the snow should not immediately disappear but instead should gradually and slowly die out until there are no active particles left. The "q" key should allow the user to exit the application.

6. Displaying Information [2 marks]



```
Diagnostics:
  particles: 58 of 1000
Scene controls:
  s: toggle snow
  q: quit
```

Use GLUT bitmap fonts to display the following application information:

- ❄ Diagnostic: the number of active particles and the maximum number (e.g. "120 of 500")
- ❄ Scene Controls (keys used for interaction (in part 5))
  - ○ "s" to toggle snow (on/off)
  - ○ "q" for quit
- ❄ Any keys required for your additional features.

You may place this diagnostic information anywhere in the window and in any layout that suits your scene, but it must contain the required content. If you want to be able to turn diagnostics off, then add a key event for the “d” key that toggles rendering of diagnostics off or on. When you first start your scene the diagnostic and scene control information **must** be displayed.

7. Two Additional Features [14 marks]

Add **two** new features to your snow scene. If you need help with how to best realise your ideas, then please discuss your proposal with the class instructor in class or by email**.**

Use your imagination. Features are graded according to technical difficulty, novelty, and visual effect. For example, more marks will be allocated for animated effects or algorithmically difficult effects. You are expected to investigate possible ways to implement your features and to evaluate and employ the best solution. This investigation should be presented in your logbook in detail and include relevant references to resources you used to help you design and implement your features.

8. Logbook [8 marks]

You **must** hand in your logbook as it forms part of your proof of authorship. Remember, the onus is on you to prove you are the creator of your product; thorough record keeping is essential to this process.

Your logbook should record dates, time spent, a record of bugs and fixes, design details, additional feature ideas, and details as to how you realised those features.

You must provide a small statement in the last entry that critically evaluates what you did well, what you might do differently next time, and identifies any shortcomings of your application.

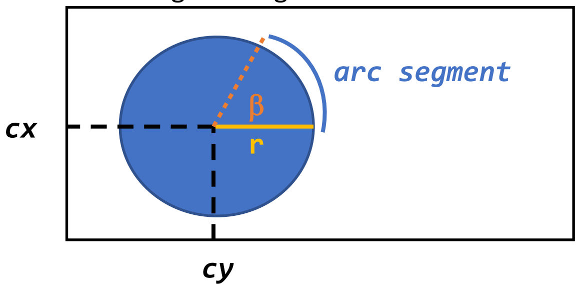
Basic Geometries

To draw a 2D circle, you will need to use the circle equation and draw the circle as a polygon or a triangle fan. The basic algorithm is given below.

```
Loop for $\beta = 0$ until $\beta = 2\pi$ for each loop increment $\beta$ (angle of rotation) by the arc segment length you wish to use ($i$). Smaller arc segments give nicer results.

For each iteration of the loop $i$:

    x = cx + (sin($\beta_i$)*r)
    y = cy + (cos($\beta_i$)*r)
```



```
where:

r is the circle’s radius and (cx,cy) is the location of the centre of the circle relative to the drawing window.

Note that the coordinates of the circle should be stored in an array IF the circle needs to be animated or if the circle location is initially set to a random location.
```