COMP37111: Advanced Computer Graphics

Coursework: Particle Systems

Toby Howard and Steve Pettifer

Academic year 2019-20

Department of Computer Science
The University of Manchester

Contents

1 Introduction

Computer graphics is about computationally creating visual effects. Although you need to have mastered the basics of a system such as OpenGL to get the right-coloured pixels to appear in the right places on-screen, many of the the real challenges in this field are about understanding how to simulate the phenomenon you're trying to render: and that's as much about 3D geometry, the interactions between light and matter, and how solid and flexible objects behave in the real world as it is the details of any graphics library.

In this exercise you're going to be creating a dynamic effect called a **particle system**[W], which will be introduced in Lecture 2. To create a basic particle system you won't need any knowledge of OpenGL beyond what you will have already encountered in COMP27112; but this exercise will get you thinking about the data structures and algorithms necessary to create a real-time effect, and about how to turn a visual concept into an on-screen reality. Unlike the second year graphics laboratory exercises, which were quite prescriptive in their nature, this exercise is left deliberately open-ended; the type of particle system and the level of sophistication is up to you.

Figure 1 shows some examples produced by the reference implementation of the exercise, and is meant as an illustration only – your solution will almost certainly look quite different.

You are expected to spend no more than 30 hours in total on the exercise.

WARNING #1. TIME MANAGEMENT. Do not spend longer than the 30 hours on this work. If you find the work taking longer than anticipated, you should concentrate on completing the key/basic elements of the work. Your Third Year is a crucial year of study for you, and you should take care to balance your workload. We don't want the time you spend on this course unit's practical work to adversely affect other aspects of your 3rd year studies.

WARNING #2. The purpose of this exercise is not to create a fully-featured particle system modelling suite. Don't get carried away, unless you are absolutely sure you can spare the time.

WARNING #3. Don't plunge in and start hacking code. Do some design on paper first – you'll save a lot of time.

WARNING #4. Read the whole of this script (not the papers, just the description of the exercises and the marking scheme) before starting to do anything – you'll get a much better idea of what's expected that way, and be able to optimise your effort to get the best marks for your work.
WARNING #5. Really, don't plunge in – read all the script, make sure you

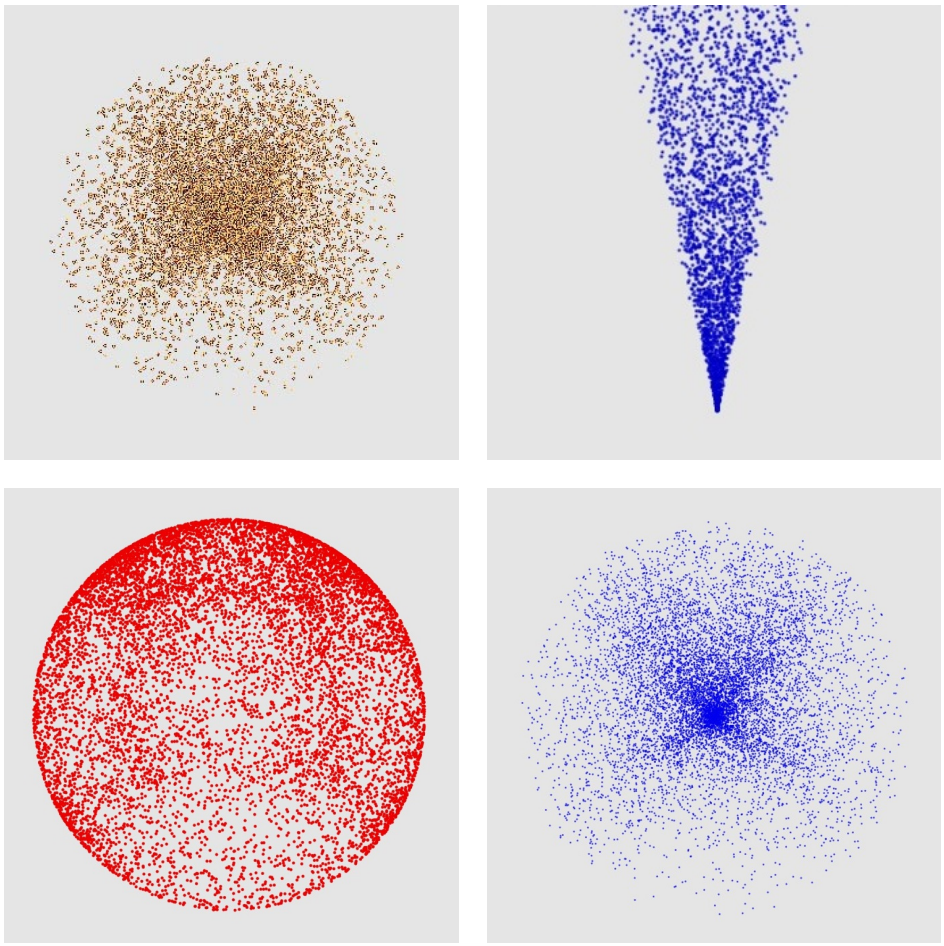

Figure 1: Some examples produced by the reference solution to this exercise.

understand the marking scheme so there aren't any horrible surprises, and think about the design of your code and any experiments you're going to run before putting finger to keyboard.

2 Aims

This lab exercise aims to:

- Support the taught course material, in particular the topics of **procedural modelling**[w] and **real-time rendering**[w].
- Expand your experience of applying computer graphics to a specific problem.
- Expose issues of making graphical systems perform reliably in real-time.
- Develop your graphics programming skills.

3 Timetable, organisation and support

There are 8 weeks scheduled for the lab (excluding Reading Week). It starts on **Monday 30 September (Week 2)** and the submission deadline (using `submit`) is **18:00 on Friday 6 December 2019 (end of Week 11)**. This is a hard deadline, and there are no extensions. Work submitted after the deadline will be subject to the standard lateness eanalty. Face-to-face marking and feedback will take place in **week 12** (full details in Section 7, below).

There is one hour of scheduled lab time per week, on **Tuesday 16:00-17:00 in Room 1.8**. Attendance is optional. The course lecturers will not normally attend these sessions, but there are two 'lab clinics' during the semester (see Blackboard for thedates), when the lecturers will attend the lab. There are no GTAs assigned to the lab, and there is no attendance recording.

4 Reading and resources for the lab

We'll be covering particle systems in Lecture 2 of COMP37111 (week 2). As well as the lecture notes, you'll also need your OpenGL manual from COMP27112 (spare copies are available from the Student Resource Centre, and also online from the COMP37111 Blackboard page).

You should read W. T. Reeves, 1983 and William T. Reeves and Blau, 1985, and if you're really interested possibly Kolb, Latta, and Rezk-Salama, 2004 too. Links to these are in the References section at the end of this script. Note that the links can only be accessed from on-campus or via VPN.

5 Software and machines

The exercise is to be implemented using OpenGL with C/C++, or WebGL. You can use any version of OpenGL you wish – either the fixed pipeline version you met in COMP27112 (and installed on the Department PCs), or the programmable pipeline (shader) version. We'll provide a simple bit of C skeleton code (in GitLab, in Week 2 of the course) that uses GLUT to get you going, but you are welcome to use any GUI toolkit you like. You are also free to use whatever third party frameworks or libraries you wish; we can't promise to provide help on those though – you'll have to figure out the details yourself.

If you do use any third party code you must (a) make sure that you understand any licensing implications and be able to demonstrate that you have used the code in line with its license, and (b) clearly acknowledge this in your code (even if the license doesn't require it); when we meet you to mark your work, you must explain how and why you have used them, so that it is clear which parts of the project are your own work, and which parts are from other people's code.

You can develop your work on whatever computer suits you (Linux, Mac or Windows, in the Department or at home) as long as you are able to demonstrate your program working live in the Department, either on a PC in Room 1.8 or bringing in your own laptop/PC if necessary.

6 The exercise task by task

We suggest you approach the exercise in a number of staged tasks, as described below, but it's really up to you: if you'd prefer to tackle it differently, that's fine. The task list here is presented only as a suggested route for satisfying the deliverables.

TASK 1 : DESIGN YOUR SIMULATION [4 HOURS]

Decide what you will be simulating (take a look at the marking scheme now to convince yourself that whatever you've chosen provides scope for demonstrating all the different properties that will attract marks). Particle systems can be used to simulate lots of different phenomena; many of these are 'real world' things like fireworks, explosions, waterfalls, smoke, rain, fire or snow. In these cases you'll want to create behaviours for the particles that mimic 'classical physics' at some level. There are plenty of other types of systems that could be simulated too that follow different laws; you might want to think about modelling something at the subatomic level, or at the other end of the scale-spectrum simulate the behaviour of matter swirling into a black hole. You could even decide to model something like

a **Strange Attractor**[W], or something completely abstract (be warned however; in these cases to get a good mark you will need to explain what 'laws' you have implemented, and demonstrate that they are of comparable or greater complexity than those associated with classic Newtonian physics). Task 1 is a brain-and-paper task. Do some reading, do some research, think, sketch, plan. Do not code. In particular, make sure you understand the behaviour you're going to model, and have thought through any performance implications (e.g. for example, are there any interactions *between* particles in your system?) Be clear too about what 'extras' you might implement if you have time, and try to make these meaningful to your scenario (for example, if you've implemented 'snow', perhaps you could get it to settle in clumps on some objects in your world; if you're implementing a firework-like system, you'll probably want to make sure you can have lots of fireworks firing at once to get a nice effect).

Here are some things you should definitely think through before you start to write any code:

- The data structure that represents an individual particle. How are you going to represent this in code? What properties does a particle have, and how are you going to represent these efficiently?
- How are you going to represent a collection of particles? What matters most here: is it the time or space complexity? Think through how your program is going to access this data structure (e.g. will it iterate from start to finish, will it be randomly accessing it?) Are you going to have the concept of more than one 'emitter' and if so does your design lend itself to this?
- What are you going to do when a particle dies? Are you going to recycle data structures or throw them away and make new ones?
- How are you going to represent the physics of your simulation? How does this interact with the data structures you've designed so far?
- How are you going to interact with your simulation to test out different effects? Do you need to be able to do this interactively via a GUI or key-presses? Or would it be better to have parameters you can change on the command line? Or do you need both? How will changes you make to parameters interact with the data structures and physics you've designed?
- How are you going to explore the performance of your finished system? What are the different things you want to measure, and make sure your design allows you to measure the things that matter. How are you going to demonstrate that you've measured sensible things?

TASK 2 : DRAW ONE PARTICLE AND MAKE IT MOVE [2 HOURS]

Get going by drawing a single particle that moves over time. Render your particle however you like. It might be best to start with a simple `GL_POINT`. This will be a bit small to see, so scale it up a bit with `glPointSize()` (this function isn't documented in the OpenGL manual, but you can easily find it online).

You'll need to have a loop in your program that implements 'ticks' of time. At each tick, work out the new position of the particle, and draw it. You can give your particle an initial position, velocity and acceleration. Its movement should conform to the equation of motion under gravity. Use the space key to reset time to 0 and trigger the system to start to emit particles.

TASK 3 : DRAW A SYSTEM OF PARTICLES [4 HOURS]

Now create a whole collection of particles. Use the concept of an 'emitter' – a source of particles. Two common types of emitter are single-point emitters, and geometry emitters. With a single-point emitter, all the particles are emitted from one point in space; in a geometry emitter, particles are emitted from multiple sources arranged in some geometrical pattern (a line, for example, might model a waterfall).

You'll need to choose an appropriate data structure for your chosen emitter. Then take the technique you developed in Task 2, and apply it to each of the particles produced by the emitter, in turn.

This time, when assigning initial parameters to each of the particles, you'll want to apply a bit of randomness, so they all behave slightly differently. Our skeleton program includes a function `myRandom()`, which returns a pseudo-random number r in the range $0.0 < r \leq 1.0$.

TASK 4 : CONTROL OF PARTICLE PARAMETERS [4 HOURS]

Use GLUT (or whatever GUI you are using) to implement simple interactive control of the parameters that influence your particles, including:

- initial velocity
- initial colour
- intensity of gravity
- lifetime of particle
- number of particles in system

TASK 5 : IMPROVED/ALTERNATIVE RENDERING [8 HOURS]

So far you'll have rendered the particles as `GL_POINT`s, and this should have been enough to give you confidence that your system is behaving sensibly – but depending on your chosen example, they probably don't look very realistic. One of the interesting things about `GL_POINT`s is because they don't need the same mathematical treatment as other primitives, they often use different paths in the graphics hardware and software to things like polygons. In the olden days this almost always meant they were faster to draw than anything else, but modern graphics cards are so heavily optimised for drawing the kinds of objects and effects you see in games etc. that this isn't necessarily the case. Try drawing your particles as something else, and note whether this makes your simulation faster or slower (if you are developing on more than one system, you may even find it's faster on some hardware and slower on others; it doesn't matter which way round it is, but make sure you're clear about what's going on in your particular case).

Here are some things you might want to consider as alternatives to `GL_POINT`. Implement at least one of these and think about the issues of visual fidelity versus performance. You're welcome to implement more, but only do so if there's a demonstrable benefit to your application, or you think it's going to tell you something interesting about the performance.

1. Particle trails. This is often used for rendering natural forms like grass. Join the positions of a particle at time T and time $T + 1$ with a line. Try different styles of line.

2. Render a particle as a little **billboarded sprite**[W] using `GL_QUADs` – that is, a quadrilateral centred on the particle's position, and rotated so that it is facing the camera. For simplicity, you can assume the camera is fixed, and at a convenient position (as it is already, in the skeleton program). Experiment with different sizes of quads.

3. As (3) but each quad will have an alpha value, to give a level of transparency.

4. As (3 and/or 4) but each quad will be textured. If you want to do this, we suggest you use simple hard-coded texture patterns; it is of course possible to read textures in from files, but this can be fiddly and time-consuming, so beware. If you feel you do have the time for this, be sure to use an existing library to read textures from image files.

5. Some other rendering technique you've researched, or thought of.

TASK 6 : CONDUCT EXPERIMENTS TO EXPLORE PERFORMANCE [6 HOURS]

In this task you will conduct experiments to see how well the performance of your system copes with large numbers of particles, and what the effects are of enabling or disabling different aspects of your code. Increase the numbers of particles from (say) 100 to 1,000 to 10,000 to 100,000 to 1,000,000 (and perhaps more). What effect does this have on performance? Is it linear? Are there any interesting performance effects at certain numbers of particles? Try enabling and disabling different parts of your system to try to understand how they contribute to the overall performance. For example, can you measure the performance of the 'physics' versus the 'rendering' and vice versa?

TASK 7 : SOPHISTICATION AND FLAIR [2 HOURS]

This part of the lab is very open ended; it can be very rewarding, but you do need to think very carefully about the cost/benefit of anything you decide to implement here; please don't go over the allocated time for this lab (not because we don't want you to play with graphics programming, but because it could have a detrimental effect on your other studies!). Here we're looking for any features or effects that enhance your particle system and show off your graphical prowess. There are only 2 marks available, and in terms of allocating the marks for things in this section, we'll be quite generous with the first, and quite stingy with the second – so to get some marks you only have to do something quite simple, but to the full marks you'll have to do some really impressive (and you should really only attempt to do something impressive if you have loads of spare time on your hands).

Examples of the kinds of things we have in mind, starting at the very simple and getting increasingly ambitious are:

- Multiple particle sources
- Supporting different viewpoints of the world, or being able to fly around the scene
- Textured particles
- Interactions between particles
- Animated 'fly-throughs' of the scene (search for 'fireworks filmed with a drone' for a video of how beautiful this can look with real-world fireworks).
- Interactions between particles and other objects in the world (e.g. water bouncing in a fountain, snow settling on houses, Deathstar ray beams destroying planets).
- GPU-based approaches to particle-rendering

7 Deliverables and assessment

The lab forms 25% of the overall assessment for COMP37111. The lab itself is marked out of 20. Marks are awarded based on your demonstration in week 12, of a working system, designed and implemented by you. If you have used code from elsewhere this must be clearly acknowledged in your program, and mentioned during your demonstration.

1. Multiple particles appearing on-screen, moving in three dimensions [2 marks]. RUBRIC: 1 mark for multiple particles appearing; 1 mark for multiple particles moving.

2. Some form of rendering beyond `GL_POINT` [2 marks]. RUBRIC: 2 marks for any kind of rendering which is not GL_POINT. So, either 0 marks or 2 marks [note: if youve just joined up random points with lines or quads etc to make a big fizzing scribbly mess, this doesnt count].

3. Ability to interactively control the system to change properties of your world and see appropriate effects of the changes. RUBRIC: 2 marks for being able to interactively change at least 2 different particle properties such as time-to-live or velocity; 2 marks for being able to change any world property such as gravity or wind.

4. Analysis of the fidelity of your chosen laws of motion – how well do they simulate the behaviour you have chosen to model? Fidelity of chosen laws [3 marks]. RUBRIC: 1 mark for basic description, i.e. can describe what the laws of motion are intended to be, and demonstrate that the particles roughly follow those; 1 mark for ability to describe informally (eg without captured data) how accurately the laws of motion are simulated; 1 mark for ability to describe with specific data/examples how accurately the laws of motion are simulated.

5. Efficiency of your approach to implementing your laws of motion – how efficient is your computation of particle position between frames? RUBRIC: 1 mark for any evidence that efficiency has been thought of; 1 mark if there is evidence of thinking about efficiency and what measures could be taken to improve it, or a solid argument as to why there is no possible way of improving it; 1 mark for demonstrating that specific measures to improve or maximise efficiency have been implemented.

6. Analysis of overall performance / rendering speed, and discussion of efficiencies implemented. Here we'll be expecting you to be able to comment on the performance of your system; is the performance limited by the CPU? the GPU? How do you know that? Are there any

particular bottlenecks in your design, and if so, how could you (hypothetically) improve the performance? This is as much about understanding as it is about implementation so if your system is performing poorly compared to your peers', but you know why that's the case and what could be done about it, then you will be able to get some of the marks for this task. For full marks we would expect evidence of experimentation such as graphs of particle numbers versus framerates [4 marks]. RUBRIC: 1 mark for sensible discussion of how performance is bound by the limitations of data structures/CPU/use of GPU/transfer of data between CPU-GPU; 1 mark for evidence of exploring performance/rendering bounds by performing experiments; 1 mark for analysis/discussion supported by some data; 1 mark for a rigorous performance analysis supported by graphs or other data visualisations.

7. Sophistication and flair. Here were looking for any features or effects beyond a basic particle system, as described in Task 6 above [2 marks]. RUBRIC: 2 marks for demonstration of 2 different things (which could be taken from the list in Task 7, p10, or could be other things); or 2 marks for one super-thing that is impressive and required significant design/implementation.

8 Submission and assessment

Submission is via GitLab. See https://wiki.cs.manchester.ac.uk/index.php?title=UGHandbook19:C
The submission deadline is **18:00 on Friday 6 December 2019 (end of Week 11)**. This is a hard deadline, and there are no extensions. Face-to-face assessment and feedback will take place in Week 12, and we'll organise a timetable.

9 Acknowledgements

This exercise was written by Toby Howard, with tweaks by Steve Pettifer. The reference lab implementation was written by Arturs Bekasovs, an undergraduate student who'd just completed his 1st year, and was employed by the Department as a vacation student.

10 License

References

Kolb, A., L. Latta, and C. Rezk-Salama (2004). "Hardware-based simulation and collision detection for large particle systems". In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. HWWS '04. Grenoble, France: ACM, pp. 123–131. ISBN: 3-905673-15-0. DOI: 10.1145/1058129.1058147.

Reeves, W. T. (1983). "Particle Systems – a Technique for Modeling a Class of Fuzzy Objects". In: *ACM Trans. Graph.* 2.2, pp. 91–108. ISSN: 0730-0301. DOI: 10.1145/357318.357320.

Reeves, William T. and Ricki Blau (1985). "Approximate and probabilistic algorithms for shading and rendering structured particle systems". In: *SIGGRAPH Comput. Graph.* 19.3, pp. 313–322. ISSN: 0097-8930. DOI: 10.1145/325165.325250.