COMP37111 - Advanced Computer Graphics Notes

Sam Littlefair

October 8, 2018

1 Course Unit Structure

- 10 credit module. Online exam 75%.
- 30 hour individual lab marked out of 20, due Friday 7 Dec 2018.
  - Design and implement a simulation with particle systems.
  - Model real-time behaviour of a particle system.
  - Give analysis of performance, real-time interaction with varying number of particles.
- The course is split into 4 main sections, the first 2 by Toby Howard:
  1. *Generative Modelling:* Creating 3D models and textures from sets of rules. (Particle Systems, Fractals..)
  2. *Modelling and model acquisition:* Where to get meshes. (Laser scanning, Triangulation..)
- And the last 2 by Steve Pettifer:
  3. *Global illumination:* The rendering equation, ray tracing, luminosity..
  4. *Real-time rendering:* Maximising performance (i.e. frame-rate) using methods like culling.

1.1 Resources

- Real-Time Rendering book. (Tomas Möller, Eric Haines, Naty Hoffman)
- Real-Time Rendering website.

2 Generative Modelling

For most things we can use polygons, but there are a lot of phenomena that are not fundamentally polygonal - Liquids, fire, smoke, trees, bushes, hair etc. For these we use generative (aka procedural) modelling techniques:

2.1 Particle Systems

Each particle has a set of properties which change over time:

- Size, shape, colour, opacity.
- Position, velocity, acceleration.
- Age, lifetime.

2.1.1 Particle Attributes

For some property P of the particle, we compute its initial value using:

$$P_{initial} = P_{mean} + (Random() \times P_{variance}) \tag{1}$$

2.1.2 Computing particle position

You can compute the position of a particle at time t using standard equations of motion:

$$x_p = v_p t cos\beta_p \tag{2}$$

$$y_p = v_p t sin\beta_p - \frac{1}{2} g t^2 \tag{3}$$

- x_p = Particle Velocity in m^2
- β_p = Particle Angle in degrees
- g = Acceleration of Gravity in ms^2
- t = Time in s

For particle bounces, check if $y = y_{ground}$, if so reset the particle position and dampen the velocity. Bounces are a special case of particle/polygon intersection, we need fast intersection algorithms.

TODO: Add efficiency and scalability section here.

2.1.3 Particle System Algorithm

For every frame:

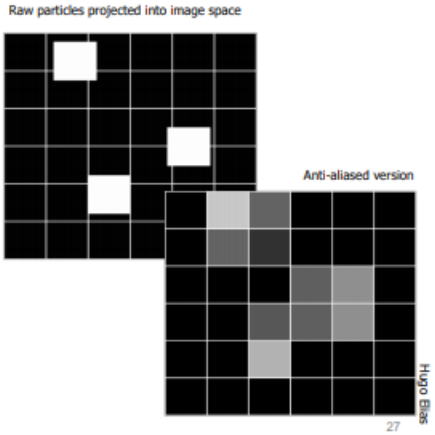
1. Remove any particles past their lifetime.
2. Create new particles and initialise them.
3. Update attributes for each particle.
    - Position:
        - Detect collisions of particles with environment.
        - Detect inter-particle collisions.
    - Colour, acceleration, etc.
4. Render current particles.

2.1.4 Particle System Algorithm

Particle systems engines are often driven by an external textual script which gives: Laws, initial values, limits, etc. The engine reads script, sets parameters, then executes rules.

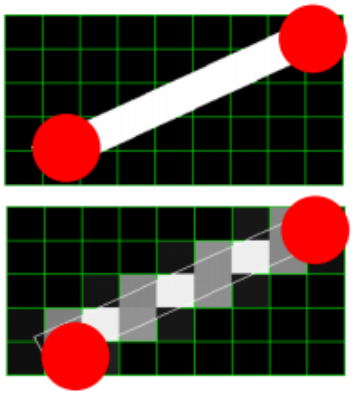
2.1.5 Rendering Ideas - 1

Particles rendered simply as pixels/groups of pixels, raw or anti-aliased.



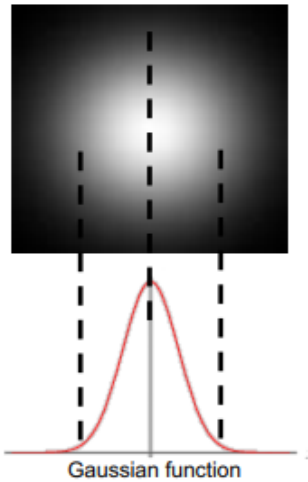
2.1.6 Rendering Ideas - 2

Connect old and new particles with a line segment.



2.1.7 Rendering Ideas - 3

Particles rendered as Gaussian Kernals aka splats.



2.1.8 Rendering Ideas - 1

Particles rendered as small alpha-textured (partially transparent) quads. Like billboards, always aligned to screen.

2.1.9 Rendering Grass

We can make the particles path draw the shape of the grass.

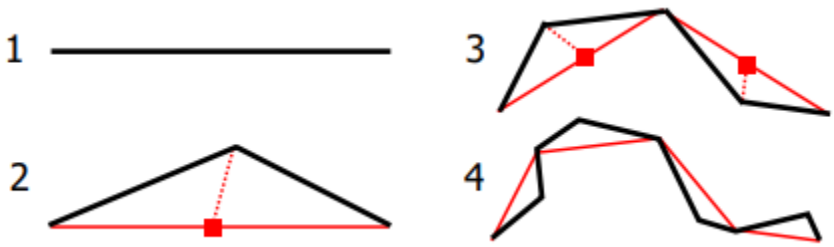
2.2 Fractal Modelling

A lot of natural things are **self-similar** meaning they have the same shape at different scales.

2.2.1 Modelling fractal curves

We can use **recursive subdivision**:

- Take mid-point of line, displace it randomly
- Replace first line with two new lines
- Repeat recursively until required level of detail.



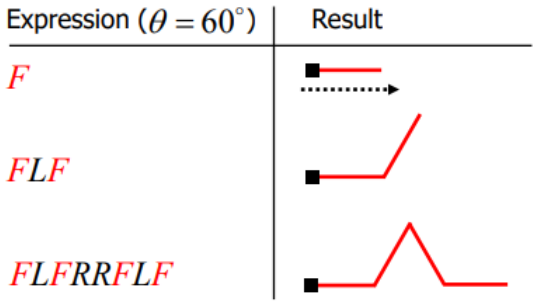
2.2.2 3D terrain generation

```
splitPoly (vertices) {
    if (!detailedEnough (vertices) ) {
        splitPoints = computeSplitPoints(vertices)
        foreach splitPoints as s {
            splitPoly (s);
        }
    }
}
```

2.3 Grammar-based modelling

We can define objects using simple grammar production rules, trace out path using commands like:

- F = move 1 unit forwards
- L = turn left through angle θ
- R = turn right through angle θ



We can apply rules recursively, for example:

$$F \rightarrow FLFRRFLF$$

2.4 L-Systems

L-systems add **push** and **[pop]** commands, which permit more complex structures:

$$F \rightarrow F[\mathbf{R}F]F[\mathbf{L}F]F$$