Procedural Generation

“Computing instead of modelling”

Procedural Modelling
Procedural Animation
Physically-based Animation
Main Points

- **Definition and Examples**
  - Procedural Textures/Models
  - Physically Based Animation
  - Particle Simulation
  - Behavioural Animation

- **Basic Principles**

- **Practical**
**Definition**

- **In the field of computer images, procedures are used..**
  - to generate pixels,
  - to model geometry
  - to animate the scene

- **Procedural graphics refers to graphics which are generated on-the-fly by a fixed set of rules rather than a modelling pre-process**
  - Simulating reality (e.g. physically based modelling) falls loosely in this category.
A desirable goal is for high complexity in images, which comes from

Great Processes OR Great data
Procedural graphics

- All computer imagery is generated by a combination of
  - Human input
  - Automated computational work

- We define procedural graphics as any type of graphics that is generated by a relatively large portion of computation in relation to manually entered data

- Motivations
  - Cost/Lack of human manpower
  - Increased complexity (lots of data from less volume of work)
  - When there are well known and well defined rules, computational procedures can sometimes do a better job than humans
    ... at least as far as complexity, accuracy, realism are concerned
Procedural Modelling

Where procedures count: modelling a full city would require a large amount of manpower – alternatively we can define some rules to generate a plausible city with logical variations in building sizes and styles. Putting some clever effort into establishing rules for building cities allows us to reduce the modelling workload.

CityEngine: A system for generating large complex cities based on relatively simple input
Procedural Modelling
Procedural Modelling
Procedural Modelling
Procedural Modelling
Procedural generation in the game, SPORE
© Electronic Arts
Spore

- Spore is a simulation that "ranges from the cellular level to the galactic level". In his GDC talk, Will Wright likened the style of game-play of each stage to an existing game:
  - Microbial stage, similar to Pac-Man
  - Creature stage, Diablo
  - Tribal stage, Populous
  - City stage, SimCity
  - Civilization stage, Risk and Civilization
  - Spacefaring stage, (a.k.a. UFO stage or Invasion), with some elements reminiscent of Destroy All Humans!
  - Galactic stage, which is a giant sandbox game
A height field is a black and white image from which we get the heights of points on the map. Bright bits are high ground; dark bits are low. This can be used to generate landscapes (a.k.a. terrain). N.B. here some additional work is done to add nice colours.
Procedural terrain in minecraft

http://www.jewe.org/software/ri_height.png
life: 100

ammo: 272

kkrieger: 96kb  half-life: 600 000+ kb
Fractals

Benoît Mandelbrot in 1975
Example: Sierpinski Gasket

Wacław Sierpiński (1882-1969)
Mandelbrot Set

Example in processing: https://processing.org/examples/mandelbrot.html
Interpolation

\[ x = t; \quad y = 0.5 t \]

```c
void lineDDA (int x1, int y1, int x2, int y2) {
    int dx = x2 - x1, dy = y2 - y1, steps, k;
    float xIncr, yIncr, x = x1, y = y1;

    if ( abs(dx) > abs(dy) ) steps = abs(dx);
    else steps = abs(dy);
    xIncr = dx / (float) steps;
    yIncr = dy / (float) steps;

    point ( (int) x, (int) y ); //draw pixel
    for (k=0; k<steps; k++)
    {
        x = x + xIncr;
        y = y + yIncr;
        point( (int) x, (int) y ); //draw pixel
    }
}
```
Extrapolation

- Generate output based on function.
- Take one or more input parameters (or control variables), not necessarily end points, to compute new shapes or graphical objects.
Most 3D Studio MAX objects are parametric surfaces.
Furthermore, operations such as lathe, extrude, sweep employ parameterization.
Procedural Noise

- **PERLIN NOISE:**
  - Pseudo-randomness
  - Controllable synthetic texture with detail

KEN PERLIN TUTORIAL: http://www.noisemachine.com/talk1/
Procedural Terrain
Procedural Animation Examples

- Spore
- Evolving virtual creatures
- Rag-doll physics
- Fluids
- Particles
Non-procedural animation examples

- **Scripted**
  - Direct scripting
  - Key-frame interpolation

- **Performance driven**
  - Motion capture
Physically based animation

- Use laws of physics to control behaviour of objects
- E.g.:
  - Impulses/Forces
  - Gravity
  - Friction
  - Buoyancy
Particle System Simulations

Star Trek Wrath of Khan: One of the earliest effective uses of particle systems is an example of procedural animation.
Behavioural Simulation
Flocking

Alignment

Separation

Cohesion

Path Following

Leader following

Sensing

Obstacle avoidance
Evolving Virtual Creatures
Karl Sims – Siggraph 94
Summary

- **Procedural generation**
  - Increased detail without human effort
  - Increased detail without large amounts of data
  - Increased realism (sometimes)
  - Emergent behaviours
Basic Building Blocks
Procedural Generation

- Automated generation of detail
- Usually a trade off between
  - Control: constraints or specific requirements
  - Power: needs to create data so that author doesn’t have to
  - Also affordances: degrees of freedom without compromising power or control
- Some recurring Mechanisms
  - Extrapolation: create more and more stuff
  - Interpolation: fill in details in between
  - Randomness: create diversity
Extrapolation

- Start with ..
  - some initial conditions
  - a pattern to follow
- Repeat pattern to generate novel output
Interpolation

- Start with...
  - two end conditions
- Given a point between the two ends... calculate what lies at that point
Randomize

```cpp
void setup()
{
    size(800, 400);
    noLoop();
}

void draw()
{
    background(255);
    //strokeWeight(2);
    beginShape();
    vertex(0, height);
    for (int x=0; x<width; x++)
    {
        float y = random(height);
        vertex(x, y);
    }
    vertex(width, height);
    endShape();
}
```
Practical Examples
Even Repetition

```cpp
//draw regular lines evenly across the screen
void setup()
{
    size(600, 600);
}

void draw()
{
    for (int y=0; y<height; y+=10)
    {
        line(0, y, width, y);
    }
}
```
Increasing Repetition

```
void setup()
{
  size(600, 600);
}

void draw()
{
  float step = 1;

  for (int y=0; y<height; y+=step)
  {
    line(0, y, width, y);
    step = step +1;
  }
}

// draw lines increasingly spaced across the screen
```
Evenly Increasing Colour

```java
//Randomize color of each line
void setup()
{
    size(600, 600);
    noLoop();
}

void draw()
{
    float step = 1;
    float oldstep = 0;
    float c;

    for (int y=0; y<height; y++)
    {
        c = (float) y/width;
        //interpolation: starts off at zero and then becomes one
        stroke(c * 255);
        line(0, y, width, y);
    }
}
```
Random Colours

```java
void setup()
{
    size(600, 600);
    noLoop();
}

void draw()
{
    float c;
    for (int y=0; y<height; y++)
    {
        c = random(255);
        stroke(c);
        line(0, y, width, y);
    }
}
```
Random + coherent

- Here the lines are based on the previous

```java
void setup()
{
  size(600, 600);
  noLoop();
}
void draw()
{
  float c = 100;
  float c_change;
  for (int y=0; y<height; y++)
  {
    c_change = random(-4, 4);
    c = c + c_change;
    if (c<0) c=0;
    if (c>255) c=255;
    stroke(c);
    line(0, y, width, y);
  }
}
```
Lab 20
Procedural Generation
Terrain Example
Base code: heightfield terrain

- Copy and run the code in the next page
  - It will setup a 3D camera with a terrain generated using the `terrainMap` function to draw a Minecraft-like terrain based on an input image used as a heightfield

- Download the following image before running it
  https://www.scss.tcd.ie/John.Dingliana/cs7029/randomimg.jpg
PImage img;

void setup()
{
  size(400, 400, P3D);

  img = loadImage("randomimg.jpg");
  //we will replace this line

  img.resize(50, 50);  //shrink image to speedup program
}

void draw()
{
  background(30, 60, 255);
  fill(50, 250, 10);

  //The following lines place and rotate the "terrain"
  translate(width/2, height/2);
  rotateX(radians(rx));
  rotateZ(radians(rz));

  lights();  //basic lighting

  terrainMap(img, 10);  //draw the terrain object
}

void terrainMap(PImage a, float siz)
{
  float z;
  float zs = 0.25;
  translate(siz*-a.width/2, siz*-a.height/2, -zs*127);

  for (int x=0; x<a.width; x++)
    for (int y=0; y<a.height; y++)
      {
      z = zs * brightness( a.get(x, y));
      pushMatrix();
      translate(x*siz, y*siz);
      translate(0, 0, z/2);
      box(siz, siz, z);
      popMatrix();
      }
}

//the following lines to rotate object using the mouse
float rx = 0;
float rz = 0;
void mouseDragged()
{
  rx += mouseY - pmouseY;
  rz += mouseX - pmouseX;
}
Creating an image

Next you need to change the image that it takes as input

- You could do this by loading your own image
- But instead we will do this by creating a procedural image inside the program (see following slides)

You can create an image in processing by using `createImage`

- e.g. `img = createImage( 300, 300, RGB);`

Now in the code from the previous page replace the line

- `img = loadImage("randomimg.jpg")`

With

- `img = randimg(50, 50);`

YOUR OBJECTIVE FOR THIS LAB is to change what goes on in here to do something meaningful
Modifying the terrain

- Your task is to apply some procedural image generation to create the image that will be used for the heightmap.
- You will need a basic loop that goes through all the pixels of the image “img” and change this before passing it to the terrainMap function.
- Suggestions:
  - You can apply filters to the image as in previous lectures e.g. blur the image `img.filter(BLUR)`; //this would smooth out the terrain
  - Simple procedural images can be also built from scratch. Some that you might try are listed in today’s lecture (slides 41 to 45 of)

- Saving an image can be done as follows
  - `img.save(  “myimage.jpg”  );` //save the Pimage object “img”
  - `save(“myimage.jpg”);` // save what is on the screen to an image
  - This is optional for this lab but you may find it easier to have a separate program to save a procedural image
Basic Fractal
Fractal Circles

- The next slide shows a simple example of a fractal generated by RECURSION
- Recursive functions basically REPEAT by calling themselves (instead of a loop)
- What the program does:
  - A function recursiveCircle is defined that takes in a size S
  - IF S is greater than 10  the following is done (note this IF is very important for stopping recursion eventually)
    - a circle of size S is drawn in the current position. A relative position is set by using the function translate() before the recursiveCircle is called
    - Save the current position (we do this using pushMatrix)
    - Move left by translating by size/4 (i.e. a quarter of the size of the first circle)
    - Then draw a smaller recursive circle by calling the recursiveCircle function
    - Go back to the center of the big circle (do this using popMatrix)
    - Move right by translating by size/4
    - Then draw a smaller recursive circle by calling the recursiveCircle function

- WARNING: make sure you save your program before running it (if you get the recursion wrong it may never stop and you could lose your work)
float scaling = 1; //for resizing the circles

void setup()
{
  size(600, 600);
}

void draw()
{
  background(255);
  translate(width/2, height/2);
  drawRecursiveCircle(scaling*600);

  //zooms in or out when mouse is dragged
  void mouseDragged()
  {
    scaling *= 1+((float)(mouseX - pmouseX)/width);
    redraw();
  }
}

void drawRecursiveCircle(float S)
{
  if (S > 10)
  {
    ellipse( 0, 0,    S, S);
    pushMatrix(); //save current position
    translate(-S/4, 0);
    drawRecursiveCircle(S/2);
    popMatrix(); //recall old position

    pushMatrix(); //save current position
    translate(S/4, 0);
    drawRecursiveCircle(S/2);
    popMatrix(); //recall old position
  }
}
Modifying the recursion

- Change the sample program to create your own recursive fractal
- The following suggestions are in increasing order of difficulty (for grading purposes: you need to do at least one, the more complex the higher the mark)
  - Change the appearance of the circle (Change colours, line widths etc.)
  - Change the shape from an ellipse to something else (use rect, or beginShape see previous lectures)
  - Try to draw four circles each time instead of two (add one above the center and another below the center of the old circle)
  - ADVANCED (and OPTIONAL): Try to create a 3D recursive shape (add the PeasyCam code from the previous lab to move the camera so you can look at it). See next slides for a hint
Advanced a 3D Example

Hints on how to build a Sierpinski Cube Fractal
Overview

1. Set starting conditions:
   - translate to centre (width/2, height/2);
   - Set start value of box_size to width/3

2. Create generator function (this will be recursive)
   - Draw a box (or rect) of current box_size
   - From current position translate up, down, left, right, top-left, top-right, bottom-right, bottom left, and for each of these positions, call the draw function to create a “child” box of half the size for each of these
   - Repeat step 2 recursively

3. Make sure you have something to stop recursion
Recursive Part

```c
void sierp(int box_size)
{
    box(box_size);
    pushMatrix();
    translate (0, -box_size); // up
    sierp(box_size/3);
    popMatrix();

    pushMatrix();
    translate (-box_size, -box_size); // up&left
    sierp(box_size/3);
    popMatrix();

    pushMatrix();
    translate ( box_size, -box_size); // left
    sierp(box_size/3);
    popMatrix();
    ...
}
```

- Note we Use `pushMatrix` to save the centre of the current box
- Draw smaller box
- Then retrieve the center using `popMatrix`

---

Note YOU MUST ADD SIMILAR CODE FOR OTHER 5 DIRECTIONS
void sierp(int box_size)
{
    if (box_size >= 1)
    {
        box(box_size)
        pushMatrix();
        translate (0, -box_size); // up
        sierp(box_size/3);
        popMatrix();

        pushMatrix();
        translate (-box_size, -box_size); // up&left
        sierp(box_size/3);
        popMatrix();
        ...
        ...
    }
}

This stops the recursion when the child boxes become too small