PROLOGUE

PROCEDURAL MODELLING EXAMPLE

Example: cosine function

A function that oscillates

\[
\begin{align*}
\cos(0) &= 1 \\
\cos(90) &= 0 \\
\cos(180) &= -1 \\
\cos(270) &= 0 \\
\cos(360) &= 1
\end{align*}
\]

We could use a for loop and calculate \( \cos \) of different values and plot these. Example of a parametric equation in 2D.

```java
for (int i = 0; i < 360; i += 10)
{
    float j = 100 * \cos(i);
    ellipse(i, j, 5, 5);
}
```

Above code is slightly modified for simplicity – in practice you will have to make minor adjustment to the parameters to draw the curve at the scale illustrated.

PROCEDURAL ANIMATION EXAMPLE

```java
void draw()
{
    background(255);
    translate(70, height/2);
    scale(1, -1);
    fill(255, 0, 0);
    phase += 1;    //update this each frame
    for (int i = 0; i < 360; i += 10)
    {
        float j = 100 * \cos(i + phase);
        ellipse(i, j, 5, 5);
    }
}
```

This is just the cosine function with a different phase each time.

Above code is slightly modified for simplicity – in practice you will have to make minor adjustment to the parameters to draw the curve at the scale illustrated.

PROCEDURAL 3D MODELLING AND ANIMATION

Surface points (obtained by parametric function below):

\[
x = T_1; \\
y = \cos(T_1) + \sin(T_2); \\
z = T_2;
\]

Normals (obtained by differentiating the parametric equations)

\[
x_n = \sin(T_1); \\
y_n = 1; \\
z_n = -\cos(T_2);
\]

Procedural animation of this parametric equation:

Change the phase and recalculate normals and surface points; apply lighting based on normals.

GRAPHICS PROCESSES

Computational process comprise step-by-step instructions to generate images, models, animations

For example:

- Drawing Objects: Lines, Curves, Fills / Shading, Illumination
- Creating Models: Parameterising surface equations
- Transforming Models: Translate, Rotate, Scale
- Animating Objects: Particles, Physics, Behavioural Animation
- Camera operations + projections
- User input and interaction
PROCESSING

A free language and environment for writing simple programs with images, animation and sound.

- Uses a relatively simple syntax (similar to JavaScript/Java)
- Can create programs that range from very quick and simple to highly complex.

Available on multiple platforms with export options to Java applet or Windows/macOS executables.

Website: [http://www.processing.org](http://www.processing.org)

Outline of Capabilities:

- Load, manipulate and display images
- Draw objects in 2D (and 3D) using points, lines and geometric objects
- Animation and User Input

Programs can be compiled as Java applets for use on web pages OR as standalone executable programs in multiple platforms.

Rich collection of user contributed libraries for processing various interactive digital media including 3D models, sound, video, computer vision, computer graphics etc.

Extensions to Arduino / Wiring, Android

GETTING STARTED

Should be already installed but if not (or if you want to install in your own system), get the program from:

- [http://www.processing.org/download/](http://www.processing.org/download/)
- Includes Windows, Mac and Linux versions

Unzip to your computer (In the lab, I recommend somewhere on the D: drive)

Double click on processing.exe

- This will open the Interactive Development Environment (IDE)

BOOKS (OPTIONAL)


By Casey Reas and Ben Fry.

Published December 2014, The MIT Press.

720 Pages. Hardcover.

**Processing: Creative Coding and Computational Art**

By Ira Greenberg.

PDE: THE PROCESSING DEVELOPMENT ENVIRONMENT

Menu
- File
- Edit
- View
- Window
- Help

Toolbar

Tabs

Text Editor

Message Area

Console

Some run-time error messages might appear here.
**Processes / Procedures**

Basically, a set of instructions to follow.

Essentially all programs involve procedures or indeed ARE procedures.

In general their purpose is to generate/alter data.

**Data**

Variables hold information.

Usually this is a value that may change over time.

Common variables in imaging and animation:

- Colour e.g. RGB
- Positions: 3D vectors e.g. <2.5, -2.5, 5>
- Directions
- Time, Velocity, Acceleration

In practice we must first declare a variable stating that we need storage in computer memory for some data of a certain type. e.g.

```
int population;
```

We later put values in the variable or retrieve what's stored in it.

```
population = 6525486603;
```

**Basic Types**

In Processing we must decide and state explicitly what TYPE of variable we are storing!

Commonly used data types:

- `char k;` // a character (letter, number or symbol) e.g. a or 0 or @
- `int I;` // a whole number typically used for COUNTING; can be negative e.g. -3 or 2
- `float d;` // a real number typically used for MEASURING; with a decimal point e.g. 3.14159
- `color c;` // this is actually a vector of 3 integers to represent RGB color

With regard to the `color` data type, in Processing:

- `Built in functions red(), green(), and blue()` can be used to extract individual values from a `color` object
- `e.g. float r = red (c);`
- Conversely you can create a `color` from three scalars using the color constructor:
- `e = color (255, 0, 125);`

**Type Casting**

If you declare a variable as one type, you can't normally give it a value of another type.

```
int I;
I = 1.6; // this will give an error!
```

If for some reason you need to convert from one type to another you do it with a cast:

```
I = (int) 1.6; // this converts 1.6 to the nearest integer
// and stores it in the variable I
```

```
int i;
I = (int) (6 * 4 / 3);
println(i);
```
**ARRAYS**

Very often variables, are grouped into chunks of homogeneous data.

Large collections of data are usually stored in Arrays. Procedures can be applied "iteratively" to individual elements in the array.

Commonly used Arrays in Visual Computing
- An Image is a collection of pixels.
- An array is a collection of images.
- A polygon is a collection of vertices (and edges).
- A 3D object is a collection of polygons.

For graphical objects, we need arrays of at least 100 elements for things to be recognizable.

Most real-world objects have tens of thousands or more elements.

But, when it comes to arrays, dealing with 10 elements... is not much different from dealing with 1000000.

**BASIC ARRAYS**

Declaring arrays in processing:
- `int[] numbers = new int[3];`

Accessing arrays (pretty much like c/c++/java):
- `numbers[0] = 90;`
- `numbers[1] = 150;`
- `numbers[2] = 30;`

Alternatively:
- `int[] numbers = { 90, 150, 30 };`
- `int a = numbers[0] + numbers[1]; // Sets variable a to 240`
- `int b = numbers[1] + numbers[2]; // Sets variable b to 180`

**WORKING WITH PROCEDURES**

Two main issues in "writing" procedures
- Algorithm: Logical set of steps to follow
- Syntax: Specific formal language for mapping out steps

Important to note (if you’re not used to coding already):
- Program code cannot be ambiguous
- N.B. order, spelling, "punctuation" must be exact

**BASIC ELEMENTS OF SYNTAX**

<table>
<thead>
<tr>
<th>Comments</th>
<th>Expressions / Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>// Two forward slashes denote a single line comment. // Anything after the slashes, on a line of text, is ignored // and won’t be executed. // This is for inserting text for the benefit of the reader</td>
<td></td>
</tr>
<tr>
<td>/* If you need several lines of text the &quot;slash star&quot; comment allows you to comment out multiple lines. Everything between the slash-star and star-slash is ignored */</td>
<td></td>
</tr>
</tbody>
</table>

Comments are useful for leaving notes, explaining code also omitting a bit of code you’ve written. Use them well and they can make your life (and those of others looking at your code) much easier.

**STEPS**

"A journey of a thousand miles begins with a single step." — old proverb

Every “step” needs to be clearly defined in terms of basic operations that will be executed by the computer ...

... eventually filtering down to simple circuits in hardware.

Thankfully — reuse is an important strategy in computing so we don’t have to do everything ourselves.
ITERATION

A.k.a. repeating a set of instructions/steps

Repetition is what computers are good at.

ITERATION: while loop

```
int x = 0;
while (x<10)
{
    x = x + 1;
    print ("A");
}
```

ITERATION: for loop

```
for ( int x = 0; x < 10; x=x+1 )
{
    print ("A");
}
```

ITERATION: DRAW A LINE

```
for ( int x=0; x<100; x=x+1)
{
    y = y +2;
    point (x, y);
}
```

ITERATION: PAINTING AN IMAGE

E.g. paint a bitmap black ...

For every pixel set the pixel colour to {0, 0, 0} i.e. black

We'll talk more about images later.

ITERATION: ANIMATION

E.g. animate a moving ball:

- Ball starts at {0,0}
- Every second, move the ball by (vx,vy)
**BRANCH/CONDITIONS**

if / else

Allow us to write procedures that can "choose" to do one action or another ... based on certain things being true.

*E.g.*
- if user presses a key then play a sound
- if ball hits the wall then make it bounce back

**ADVANCED DATA: IMAGES**

An image is an array of pixels

- Each pixel is a variable which has an RGB value.
- In an image file, we usually also store the width and height of the image.

Processing has a built in class `PImage` to deal with images

```java
PImage img;
img = loadImage( "picture.gif");
image (img, 0, 0);
```

**PROCESSING IMAGES**

We change an image by applying a Procedure to some or all of it's pixels.

Examples:
- Greyscale the Image
- Clear Image
- Threshold Image
- Mirror the Image

**VECTOR DATA**

Processing also supports the creation of vector objects in 2D and 3D using a number of commonly used graphical primitives:

```java
beginShape(...)
  POINTS, LINES, QUADS, TRIANGLES,
  TRIANGLE_STRIP, TRIANGLE_FAN,
  QUAD_STRIP
endShape()
```

More on these in a later lecture!

**SOME BASIC COMMANDS**

```java
size ( 100, 100 ) ; //Set window size to 100 x 100 pixels
background (255, 255, 255) ; //Set background colour to White
fill (255, 0, 0) ; //Set fill colour to Red
stroke (0, 255, 0) ; //Set line colour to Green
point (205, 90) ; //Draw a point at (205, 90)
line (20, 20, 50, 50) ; //Draw a line from (20, 20) to (50, 50)
```

**EVENT DRIVEN PROGRAMMING**
DIRECT VS EVENT-DRIVEN MODE

If your program isn’t highly interactive e.g. simple image processing, simply state your instructions step by step: this is called Direct Mode.

But processing allows to work in a simple Event-driven mode, similar to most windows programs.

Event-driven programs loop infinitely checking to see if some event (e.g. user input) has occurred.

AN EVENT LOOP

Event driven programs loop infinitely checking to see if some event (e.g. user input) has occurred.

```
while (TRUE) {
    e = get_next_event();
    switch (e) {
        case MOUSE_PRESSED_EVENT:
            call registered MouseFunction
            break;
        case KEY_PRESSED_EVENT:
            call registered KeyboardFunction
            break;
        ...
    }
}
```

INTERACTIVE PROGRAMS

- **setup()** function is where you state operations you want performed once when your program begins.
- **draw()** function gets called repetitively (unless you stop it).
- **noLoop()** stops redrawing loop() restarts it.
- Several mouse callbacks exist including mousePressed(), mouseReleased(), mouseMoved(), mouseDragged() .
- Mouse positions are returned in mouseX, mouseY.
- The keyPressed() function gets called when you press a key. You can get the key pressed using the global variable keyPressed.

SUMMARY

- **Data**
  - Arrays (Lists)
  - Images, Polygons, Vertices, Frames, Transforms, Pixels
- **Procedures**
  - Algorithm + Syntax
  - Iteration / Event driven Loop
  - Conditions

SOME TIPS

Try to keep your code neat and meaningfully indented. Balance out matching brackets. This really helps when you start writing bigger programs.

In the Menu-Bar clicking on Sketch> Show Sketch Folder will open up the folder in which the current sketch is stored. This is the best place for placing any relevant files e.g. Images, textures and models used by your program.

Useful Key-board Short Cuts:
- **CTRL-R** runs your program.
- **CTRL-T** autoformats your code. It doesn’t change or add any text but by inserting tabs and aligning matching brackets makes the code easier to read, understand and debug.
RECOMMENDED READING

The "Tutorials" section on processing.org has some useful tips:
https://processing.org/tutorials/

I suggest you read: "Getting Started" by Casey Reas and ben Fry

Also see the reference manual online: http://www.processing.org/reference/