Processing: Initiation

Jarek Rossignac
www.gvu.gatech.edu/~jarek

1 - Introduction

A key objective of this course is to enable you to write interactive graphic programs that display shapes, images, and animations and that update them in realtime according to user's input. To help the TA to grade your projects and to help you build your online portfolio, you will be asked to post your programming assignments online as interactive applets with accompanying PDF reports.

Most of your programming assignments will be written in **Processing**, a language and environment developed at MIT by Casey Reas and Ben Fry. People familiar with Java or C will find it easy to learn Processing. In fact, Java calls may be used in Processing. Furthermore, the graphics commands used in Processing are closely related to OpenGL ones. Hence, those of you familiar with OpenGL will find it easy to adapt. In fact, you can also use OpenGL calls in Processing. This is particularly useful when you need advanced graphic functions that are not directly available in Processing.

If you never programmed in Java and never used OpenGL nor any other graphics API, do not despair, Processing will make it easy and fun for you to learn how to do interactive graphics.

You may ask "Why Processing?". Several years ago, Mark Luffel, as he was an undergraduate student in my graphics class, sent me his assignment as a Processing applet. I was curious about the language. I downloaded Processing, opened Mark's program, run it, made some changes just to see how it worked. Then I noticed the "export" button. I pressed it, et voila, I had a folder with an interactive applet that I could post on the web. I was hooked—and now, it is your turn to enjoy the simplicity of Processing.

Savvy programmers may be frustrated by the rudimentary level of the Processing editor and by the lack of debugging tools.

The former issue may be addressed by using an "external" editor (such as for instance Eclipse: processing.org/learning/tutorials/eclipse/ and www.learningprocessing.com/tutorials/processing-in-eclipse/).

The latter is a bigger issue and is discussed throughout this course: How to write and debug complex graphics programs? Good debugging tools may help a bit, but they are not sufficient. Through the course, you will learn how to scaffold your modules on solid, debugged foundations. You will learn that you need to write a lot of additional code just to help you debug these foundations. You will learn that printing out symbols and numbers does not help much when debugging graphic programs. You will learn how to develop graphic debugging tools. You will learn never to debug two features simultaneously…

This chapter explains how to install the Processing environment, how to use it to create interactive graphics programs, and how to post them as applets on the web. It also presents the structure of a Processing application, a few programming constructs that are common in graphics programs, and a few simple graphic input and rendering commands and techniques. It also lays out a structure for the assignments and the core of the classes for points, vectors, transformations, and polygons. Becoming familiar with—and following—this structure and using these classes will save you time and facilitate grading. But, if you are an expert programmer and prefer to develop your own set of classes and tools and to use your own programming style, please feel free to do so. But if you decide to use my framework and classes, you will have to put up with my non-standard formatting and programming style. I value conciseness and elegance of software, as they may me more productive.

You may want to start by reading the small tutorial at http://processing.org/learning/gettingstarted/ and then go through the various modules of http://processing.org/learning/ .

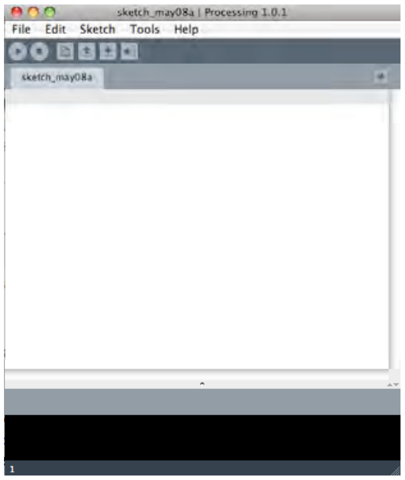
2 - Installation and operations

In this section, I explain how to install processing; how to write, run, and save an application; and how to post it on the web.

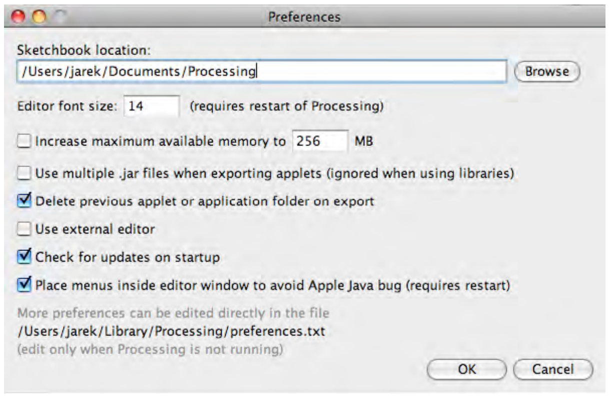
2.1 How to install or upgrade your personal copy of Processing

From your personal computer, point your browser to http://processing.org/download/ and download the latest version of Processing for your operating system. You may want to move it to your application folder.

Start Processing. You should see the following window.



In the Processing menu bar at the top of your screen (not the top of the processing window), you may select "Preferences" and use the pop-up pane (shown below) to set some preferences, such as the font size or the default folder where Processing will be saving your programs (called "sketches").


Preferences

Sketchbook location:
/Users/jarek/Documents/Processing [Browse]

Editor font size: 14 (requires restart of Processing)

☐ Increase maximum available memory to 256 MB

☐ Use multiple .jar files when exporting applets (ignored when using libraries)

☑ Delete previous applet or application folder on export

☐ Use external editor

☑ Check for updates on startup

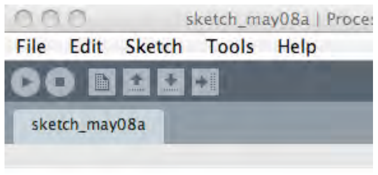
☑ Place menus inside editor window to avoid Apple Java bug (requires restart)

More preferences can be edited directly in the file
/Users/jarek/Library/Processing/preferences.txt
(edit only when Processing is not running)

[OK] [Cancel]

2.2 How to run your first program

At the top of the Processing window, there is a menu bar.

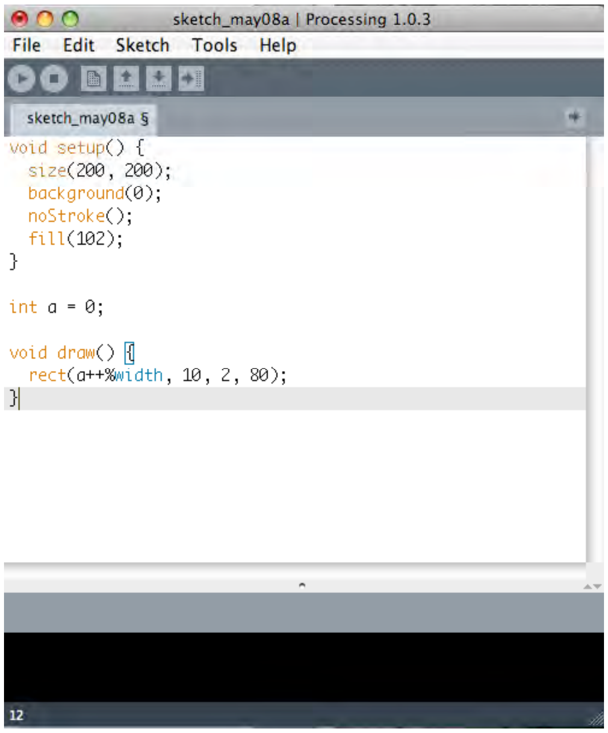


Under "Help", select "Reference". You see the *Abridged* list of Processing commands. Later, you may want to start using the *Extended* version.

Find the entry for "setup()" and click on it.

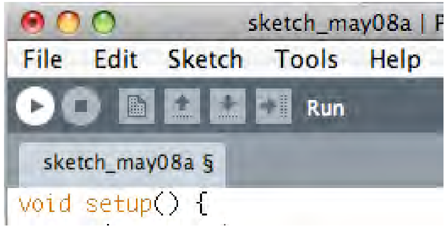
Cut the small processing program:

```
void setup() {
  size(200, 200);
  background(0);
  noStroke();
  fill(102);
}

int a = 0;

void draw() {
  rect(a++%width, 10, 2, 80);
}
```
and paste it into the white pane in the Processing window.

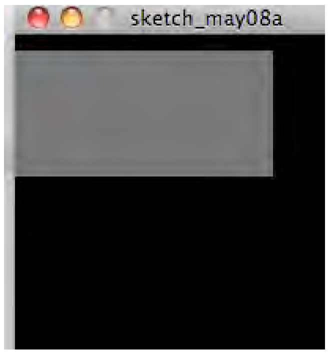


```
void setup() {
  size(200, 200);
  background(0);
  noStroke();
  fill(102);
}

int a = 0;

void draw() {
  rect(a++%width, 10, 2, 80);
}
```

To run this program, press the left-most button (just below "File") at the top of the Processing window.



After a couple of seconds, a small window pops up and the program starts. You should see a grey rectangle grow from left to right.

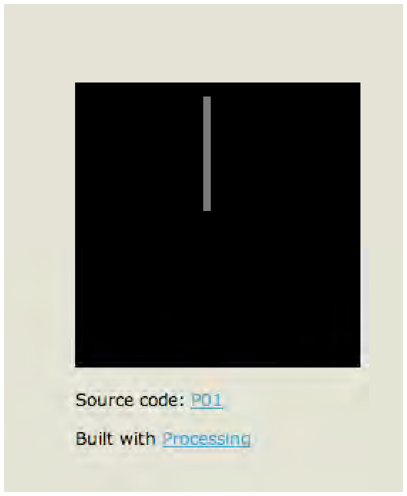


To stop the program, press on the square button next to the run button.

2.3 How to save and export your program

To save your program, from the "File" pull down menu, select "Save as", select a location for your folder and type a name, say "P01" (sans quotes). That name is used to create a *sketch* folder P01. In that folder, your program is saved as fine P01.pde. Processing expects this name coherence: In a sketch folder there is a file with the same name and extension .pde.

Now, you may want to create an online applet with your program. To do so, you must first export it. Simply press the last of the six buttons at the top of the Processing pane, or select "File" and "Export". This action creates an "applet" folder in your sketch folder. Double clicking on the index.html file in that applet folder will start the applet.



Source code: P01

Built with Processing

You may want to edit the index.html file with your favorite editor.

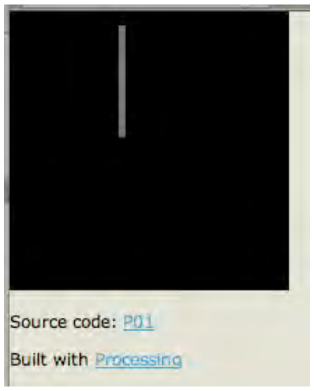
For example, I usually change

```
body {
  margin: 60px 0px 0px 55px;
```

to

```
body {
  margin: 0px 0px 0px 0px;
```

which aligns the applet graphics window with the top-let corner of the web page. This is important when you are using a large graphic window.



Source code: P01

Built with Processing

You can also change a title of your web page:

```
<title>Project P01 for CS3451</title>
```

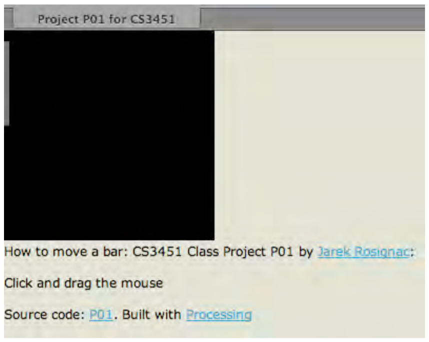
and change the text. For example, replace

```
    </div>

    <p>

    </p>

    <p>
    Source code: <a href="P01.pde">P01</a>
    </p>

    <p>
    Built with <a href="http://processing.org" title="Processing.org">Processing</a>
    </p>
</div>
```

by

```
    </div>
How to move a bar: CS3451 Class Project P01 by <a href="http://www.gvu.gatech.edu/%7Ejarek/">Jarek Rosignac</a>:<br><br>
Click and drag the mouse<br><br>
Source code: <a href="P01.pde">P01</a>.
Built with <a href="http://processing.org" title="Processing.org">Processing</a>
  </div>
```

to obtain:



Project P01 for CS3451

How to move a bar: CS3451 Class Project P01 by Jarek Rosignac:

Click and drag the mouse

Source code: P01. Built with Processing

Then, copy the application folder to your web server and give it a proper name (maybe CS3451P01).

Also, you may want to grab an image of the window of your program and save it as loading.gif in the applet folder. The image will be displayed in the browser as the program is loading.

When saving more complex sketches that have several tabs and data files, you may want to use the "Archive Sketch" option to produce a .zip file that would be linked from your index.html page.

Very important: The export action does not create a data folder. For more complex sketches, you will have a data folder containing fonts, images, data files. **You must COPY** (not move) **this data folder into the applet folder**.

Read http://processing.org/reference/environment/export.html for additional information on the exporting options.

3 - How to structure and edit your sketch

In this section, I explain how a Processing sketch is structured and how you can edit it.

3.1 How is a processing program structured

Your programs will have two main functions: setup and draw.

Setup is executed once at the beginning of the program. Typically, you use it to open a graphic window, set some graphic states, read data from files, load images or fonts, build data structures. The actions performed in the setup of our example are explained in the comments below.

```
void setup() {
  size(200, 200); // opens a small window of 200×200 pixels
  background(0); // erases the screen by painting it grey with level 0 out of
256 (which means black)
  noStroke(); // prevents line drawing
  fill(102); // enables area filling with a gray level 102 out of 256
}
```

Draw is executed at each frame. Typically, in draw you call functions that check the status of input devices (keys pressed, cursor position), change drawing attributes (color, view), update your model to respond to user actions or to continue animation, and redisplay the model. In our example, the model is represented by variable a, which represents the state of the model (it is the horizontal position, i.e., the x-coordinate of a rectangle that is painted on the screen).

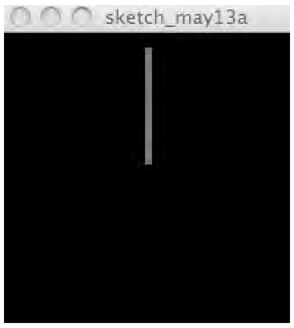
3.2 How to modify your program

Edit the example as follows. In the setup procedure, use the mouse to select the line containing background(0). If you want the line including the line break, then click the beginning of the line and drag the mouse down, or click the beginning of the line and press ⇧🡇 (keep shift pressed and then press the down arrow). Then cut it (on a mac press ⌘c) and paste it at the beginning of the draw procedure (on a mac press ⌘+v). Now your code should look like this:

```
void setup() { size(200, 200);  background(0);  noStroke();  fill(102);}
int a = 0;
void draw() {  background(0);  rect(a++%width, 10, 2, 80); }
```

and produce a moving rectangle.

Look up the description of rect and change it to make the rectangle a bit thicker (as shown below). To look up a key word (these are colored red in the Processing pane), double click on one and then in the Help pull-down menu select "find in reference" or use the ⌘F (⌘⇧f) shortcut.



You should check http://processing.org/reference/environment/ for more details on what you can do in the Processing Development Environment (PDE).

4 - How to make interactive sketches

In this section, I explain how track user actions and make interactive/reactive Processing sketches.

4.1 How to track the mouse

Let us now change the program so that it the horizontal position of the rectangle tracks the mouse movements, but only when the mouse button is pressed.

```
int a = 0;
void setup() { size(200, 200);  background(0);  noStroke();  fill(102);}
void draw() {
  background(0);
  if(mousePressed) a+=mouseX-pmouseX; // increment a by mouse displacement when
mouse pressed
  rect(a, 10, 5, 80);
  }
```

The Boolean variable mousePressed is true when a mouse button is currently pressed.

When used inside draw(), the system variable mouseX always contains the current horizontal coordinate of the mouse, while pmouseX contains the mouse coordinate in the previous frame. (Same for mouseY and pmouseY.)

Therefore, mouseX-pmouseX returns the amount of horizontal displacement of the mouse.

Note that moving the mouse when the buttons are not pressed leaves the rectangle in place. Pressing the mouse button without moving the mouse will also leave the rectangle in place. The rectangle only moves when the mouse button is pressed and the mouse is moving.

How would you change the behavior so that the rectangle jumps to the mouse position as soon as the mouse button is pressed and then tracks stays aligned on the vertical of the mouse? Hint: You only need to delete a few characters.

It is somewhat inelegant to be probing at each frame whether the mouse is pressed. Furthermore, when the mouse is pressed, we are updating variable a even when the mouse has not moved. To avoid these, you may want to use the mouse event function mouseDrag, which is executed each time the mouse is moved while one of its buttons is pressed.

```
// P02 tracking the mouse
int a = 0;
void setup() { size(200, 200);  background(0);  noStroke();  fill(102);}
void draw() { background(0);  rect(a, 10, 5, 80);  }
void mouseDragged() {a+=mouseX-pmouseX;}
```

Now, instead of drawing a rectangle, we can draw an edge (line segment) from point (pmouseX,pmouseY) to point (mouseX,mouseY), each time the mouse is dragged and a button is pressed. To do so, we simply use the line() call. Note that we do not want to erase the screen each time, so we have removed the background() call from draw(). I prefer the background to be white, so that when I include images in papers or notes, they blend better and use less ink to print. Hence, I set the background color to 255, which is white. I also increased the window size.

4.2 How to react to key actions

Finally, you can call strokeWeight() to change the width of the line. To give the user control over the width, I provide my with-changing code in keyPressed(), which is called every time a key is pressed. In there, I access the system variable key, which contains the ASCII value of the most recently pressed key. I convert it to an integer, subtract 48, and store it in variable w. I check that w lies between 0 and 10. If so, I set the stroke width to w.

```
// P03 drawing with mouseDragged and change stroke weight using keys
void setup() { size(400, 400);  background(255);   }
void draw() {  }
void mouseDragged() {line(pmouseX, pmouseY, mouseX, mouseY); }
void keyPressed() {int w = int(key)-48; if(0<w && w<10) strokeWeight(w);}
```

Now, let us add the option to erase, when the space bar is held down. Also, I use smooth() and strokeCap to increase the quality of the drawing. (You may want to turn these off to increase performance when rendering very complex drawings).

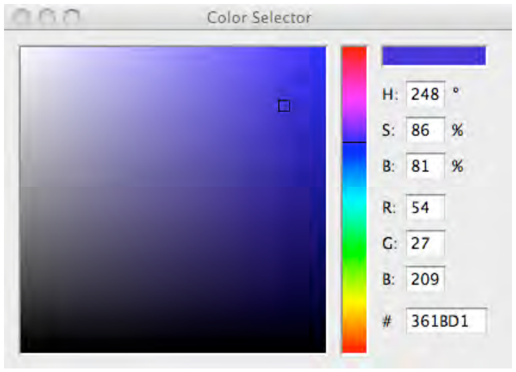
```
// P04 drawing and erasing with mouseDragged and change stroke weight using
keys
int w=1; // stroke width
void setup() { size(400, 400);  background(255); smooth(); strokeCap(ROUND); }
void draw() {}
void mouseDragged() {
   if(keyPressed && key==' ') stroke(255); else stroke(0);
   line(pmouseX, pmouseY, mouseX, mouseY); }
void keyPressed() { if(48<int(key) && int(key)<58) w=int(key)-48;
strokeWeight(w);}
```

5 - How to change colors and use buttons

In this section, I explain how change colors and support buttons in your GUI (Graphic User Interaction).

5.1 How to change color

Suppose that, instead of drawing in black, you want a nice blue color. In the "Tools" pull down menu, pick the "color Selector", move the horizontal black mark to where you like it along the vertical rainbow ramp, and then select the color you want on the large square.



In your code, replace stroke(0) with stroke(54,27,209), or whatever the red (R), green (G), and blue (B) values are shown in the Color Selector for the color you like. Your new sketch should look like this

```
// P05 drawing and erasing with mouseDragged and change stroke weight using
keys
void setup() { size(400, 400);  background(255); smooth(); strokeCap(ROUND); }
void draw() {}
void mouseDragged() {if(keyPressed && key==' ') stroke(255); else
stroke(54,27,209); line(pmouseX, pmouseY, mouseX, mouseY); }
void keyPressed() {if(48<int(key) && int(key)<58) strokeWeight(int(key)-48);}
```

As an exercise, try to improve this sketch so that the user can change the drawing color by pressing 'r', 'g', or 'b' to get red, green or blue.

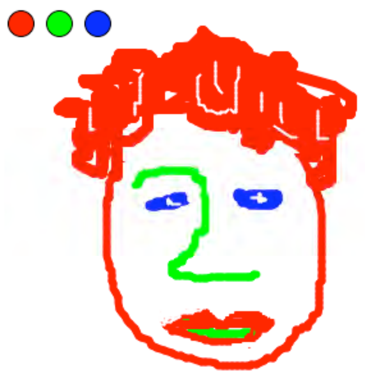
5.2 How to make a class of buttons

Instead of pressing keys, you may want to show colored buttons on your screen and program them so that the user can press on any one of them to change the drawing color. We will make a simple class for these buttons.

Each button (i.e., each object of the Button class) has 4 internal variables: the x and y coordinates of its center, the radius r of the disk, and the color col. You can create a new button with a desired position and color by calling new Button(x,y,c). The radius is kept fixed here for simplicity. There are two methods for buttons. show() draws a button as a properly positioned and colored disk. press checks whether the mouse is over the button, and if so, sets the global color variable c to the color of the pressed button. It also temporarily stops drawing by setting the global Boolean pick to false. In mouseDrag, we test pick before drawing a new stroke. Pick is reset by the mouseReleased function, which is called when the mouse is released.

```
// P05 drawing, erasing, and changing colors
class Button {float x=100, y=100, r=10; color col=color(255,0,0);
  Button (float px, float py, color pcol) {x=px; y=py; col=pcol;}
  void press() {if (sq(mouseX-x)+sq(mouseY-y)<sq(r)) {c=col; pick=true;}}
  void show() {fill(col); ellipse(x,y,2*r,2*r);}
  }
color c=color(100);
Boolean pick=false;
Button[] But = {new Button(20,20,color(255,0,0)), new
Button(20+30,20,color(0,255,0)), new Button(20+30+30,20,color(0,0,255)) };
void setup() { size(400, 400);  background(255); smooth(); strokeCap(ROUND);
for(int i=0; i<3; i++) But[i].show();}
void draw() {}
void mouseDragged() {if(keyPressed && key==' ') stroke(255); else stroke(c);
if(!pick) line(pmouseX, pmouseY, mouseX, mouseY); }
void keyPressed() {if(48<int(key) && int(key)<58) strokeWeight(int(key)-48);}
void mousePressed() {for(int i=0; i<3; i++) But[i].press();}
void mouseReleased() {pick=false;}
```

You should be able to make a nice drawing such as the one below. For practice, add a few buttons with new colors.



6 - How to save and load images

In this section, I explain how change colors and support buttons in your GUI (Graphic User Interaction).

6.1 How to save an image of the graphics screen

The call `saveFrame()` may be used to save the current content of the graphic window into a file. You can specify the file name and also the path. You can also number the images you save (variable pic) which helps for making movies. I have set it up so that all of this is indicated when you press key 'X'. In my examples, the images are saved into a folder data/Pictures and are numbered f000.tif, f001.tif, and so on. This is particularly important when you want to make movies out of a large set of such saved images and you want to ensure that image 2 comes before image 11. Hence you call them 002 and 011.

```
// P06 drawing, erasing, changing colors, saving and loading images
color c=color(100); // current drawing color
Boolean pick=false; // we just picked a color. No drawing until mouse released
int pic=0; // picture number for saving sequences of pictures of making movies
Button[] But = {new Button(20,20,color(255,0,0)), new
Button(20+30,20,color(0,255,0)), new Button(20+30+30,20,color(0,0,255)) };
void setup() { // executed once when the program starts
  size(400, 400);  // opens window
  background(255);  // paints a white background
  smooth(); strokeCap(ROUND); // sets status so that lines are drawn with
antialiasing and rounded ends
  for(int i=0; i<But.length; i++) But[i].show(); // shows all buttons
  }
void draw() {}
// executed at each frame. Here if does nothing: we are not erasing the screen
void mouseDragged() {
  if(keyPressed && key==' ') stroke(255); else stroke(c); // sets drawing color
(or white if SPACE_BAR is pressed)
  if(!pick) line(pmouseX, pmouseY, mouseX, mouseY); }
void keyPressed() {
  if(48<int(key) && int(key)<58) strokeWeight(int(key)-48);
       // changes stroke width if key '1' through '9' is pressed
  if (key=='X') {saveFrame("data/Pictures/f"+Format0(pic++,3)+".jpg"); }
        // saves graphic window as image to file
  }
void mousePressed() {for(int i=0; i<3; i++) But[i].press();}
    // it any button was pressed, we will change color
void mouseReleased() {pick=false;} // resume drawing
class Button {float x=100, y=100, r=10; color col=color(255,0,0);
    // Buttons for changing colors
  Button (float px, float py, color pcol) {x=px; y=py; col=pcol;}
     // method for creating a button from parameters
  void press() {if (sq(mouseX-x)+sq(mouseY-y)<sq(r)) {c=col; pick=true;}}
    // if picked, change c and set pick to prevent drawing
  void show() {fill(col); ellipse(x,y,2*r,2*r);} // show button as a disk of
the proper color
  }
String  Format0(int v, int n) {
     // returns an n-charcter string of the value v left-padded with 0's
   String s=str(v); String spaces = "00000000000000000000";
   int L = max(0,n-s.length());
   String front = spaces.substring(0, L);
   return(front+s);
  };
```

6.2 How to load an image and show it as background

Now that you can save your art, you may want to let the user of your program load such saved images, and other image of the correct window format. To do so, you simply attach the following code to the 'x' key. It defines an image variable I, loads it from a file data/Pictures/f000.jpg and displays it aligned at the top left corner (0,0) of the window.

```
if (key=='x') {
      PImage I;
      I = loadImage("data/Pictures/f"+Format0(picx++,3)+".jpg"); // load next image
      image(I, 0, 0);} // show it
```

It increments the loaded image counter picx, so that next time, you will load data/Pictures/f001.jpg

The entire source code for an applet that lets you change colors and stroke width, draw strokes and erase the screen, and save and load images is included below.

```
// P07 drawing, erasing, changing colors, saving and loading images
color c=color(100); // current drawing color
Boolean pick=false; // we picked color: no rawing till mouse released
int picX=0; // picture number for saving sequences of pictures of making movies
int picx=0; // picture number for loading sequences of pictures of making movies
Button[] But = {new Button(20,20,color(255,0,0)),
  new Button(20+30,20,color(0,255,0)), new Button(20+30+30,20,color(0,0,255)) };
void setup() { // executed once when the program starts
      size(400, 400);  // opens window
      background(255);  // paints a white background
      smooth(); strokeCap(ROUND); // lines drawn with antialiasing and rounded ends
      for(int i=0; i<But.length; i++) But[i].show(); // shows all buttons
      }
void draw() {} // executed at each frame.
void mouseDragged() {
  if(keyPressed && key==' ') stroke(255); else stroke(c); // drawing color
  if(!pick) line(pmouseX, pmouseY, mouseX, mouseY); }
```

```
void keyPressed() {
  if(48<int(key) && int(key)<58) strokeWeight(int(key)-48);
      // changes stroke width if key '1' through '9' is pressed
  if (key=='X') {saveFrame("data/Pictures/f"+Format0(picX++,3)+".jpg"); }
      // saves graphic window as image to file
  if (key=='x') {PImage I; // load next image
    I = loadImage("data/Pictures/f"+Format0(picx++,3)+".jpg"); image(I, 0, 0);}
  if (key=='z') {background(255); noStroke();// erase and redraw buttons
      for(int i=0; i<But.length; i++) But[i].show(); stroke(c);};  }
void mousePressed() {for(int i=0; i<3; i++) But[i].press();} // change color
void mouseReleased() {pick=false;} // resume drawing

// Buttons for changing colors
class Button {float x=100, y=100, r=10; color col=color(255,0,0);
  Button (float px, float py, color pcol) {x=px; y=py; col=pcol;}  // constructor
  void press() {if (sq(mouseX-x)+sq(mouseY-y)<sq(r)) {c=col; pick=true;}}
     // if picked, change c and set pick to prevent drawing
  void show() {fill(col); ellipse(x,y,2*r,2*r);}
    // show button as a disk of the propoer color
  } // end class

// returns an n-character string of the value v left-padded with 0's
String  Format0(int v, int n) {
     String s=str(v); String spaces = "00000000000000000000";
     int L = max(0,n-s.length());
     String front = spaces.substring(0, L);
     return(front+s);
     };
```

Note that here, for simplicity, I am not checking whether the images exist. Hence, if you keep pressing ‘x’, the program will eventually run out of image files and crash.

It is very important that you remember including the image folder in the data folder when you export your sketch and post it online. Otherwise the applet will simply crash.

7 - How to work in a retained mode

So far, the result of the user interaction was the image. Our program offered facilities for erasing, changing, saving, and loading the image. Although some popular image editing programs work this way, most graphics programs and animation represent shapes or animations and render them at each frame using display parameters and view parameters controlled by the user. In this section, we will transform our drawing program to such a mode and discuss the overall structure of such applets.

7.1 How to work with a list of strokes

Instead of drawing each mouse stroke on the screen and forgetting about it, we are going to store it. Specifically, we store the (x,y) coordinates of the starting and ending points, the color, and the stroke width. To simplify the program, I have made a class Edge with these internal variables and have made a constructor and a display method.

To make this work, I had to store the current color (`c`) and width (`ww`) and change the button actions to update these variables, rather than to change the graphic state.

I have created an array of `maxne` (=20000) edges and have initialized them all. Then, each time the user moves the mouse when the mouse button or space bar are pressed, increment the edge counter ne and set the next edge in that array to the geometry and attribute (color, width) of the last edge drawn by the user.

In `draw()` I erase the screen and then render the buttons and then all the ne edges, each one with its attributes.

When the user presses ‘z’, I simply set `ne` to 0.

```
     // P08 working with a list of edges
     class Edge{
```

```
    float x0=0; float y0=0; float x1=100; float y1=100;
    color col=color(255,0,0); int w=1;
  Edge() {} // creates standard edge
  void setTo(float px0, float py0, float px1, float py1, color pcol, int pw)
       {x0=px0; y0=py0; x1=px1; y1=py1; col=pcol; w=pw;}
  void show() {strokeWeight(w); stroke(col); line(x0,y0,x1,y1);} // draws edge
  } // end class

Boolean pick=false; // flag indicating that we just picked a color and shold
not be drawing until the mouse button is released
int picX=0; // picture number for saving sequences of pictures of making movies
int picx=0; // picture number for loading sequences of pictures of making
movies
Button[] But = {new Button(20,20,color(255,0,0)), new
Button(20+30,20,color(0,255,0)), new Button(20+30+30,20,color(0,0,255)) };
PImage I;
boolean loadedI=false; // did not load an image
color c=color(100); // current drawing color
int ww=1; // current stroke weight
int ne=0;  // current edge-count;
int maxne=20000;  // max edge-count;
Edge[] E=new Edge[maxne]; // array of edges

void setup() { // executed once when the program starts
  size(400, 400);  // opens window
  background(255);  // paints a white background
  smooth(); strokeCap(ROUND); // lines drawn antialiased and rounded ends
  for(int i=0; i<But.length; i++) But[i].show(); // shows all buttons
  for(int i=0; i<E.length; i++) E[i]=new Edge();
  }

void draw() {
  background(255); // clear screen
  if(loadedI) image(I, 0, 0); // show loaded image
  noStroke(); for(int i=0; i<But.length; i++) But[i].show(); // draw buttons
  for(int i=0; i<ne; i++) E[i].show(); // draws all edges
  }

void mouseDragged() {
  color cc;
  if(keyPressed && key==' ') cc=color(255); else cc=c; // sets drawing color
  if(!pick) E[ne++].setTo(pmouseX, pmouseY, mouseX, mouseY,cc,ww);
  }

void keyPressed() {
  if(48<int(key) && int(key)<58) ww=int(key)-48;  // changes stroke width
  if (key=='X') {saveFrame("data/Pictures/f"+Format0(picX++,3)+".jpg"); }
       // saves graphic window as image to file
   if (key=='x') {    // load next image
    I = loadImage("data/Pictures/f"+Format0(picx++,3)+".jpg"); loadedI=true;}
  if (key=='z') {ne=0;}; // erase all and redraw buttons
  }

void mousePressed() {for(int i=0; i<3; i++) But[i].press();}
       // it any button was pressed, we will change color

void mouseReleased() {pick=false;} // resume drawing

class Button {float x=100, y=100, r=10; color col=color(255,0,0);
```

```
Button (float px, float py, color pcol) {x=px; y=py; col=pcol;}
void press() {if (sq(mouseX-x)+sq(mouseY-y)<sq(r)) {c=col; pick=true;}}
void show() {fill(col); ellipse(x,y,2*r,2*r);}
}

String  Format0(int v, int n) {
   String s=str(v); String spaces = "00000000000000000000";
   int L = max(0,n-s.length());
   String front = spaces.substring(0, L);
   return(front+s);
  };
```

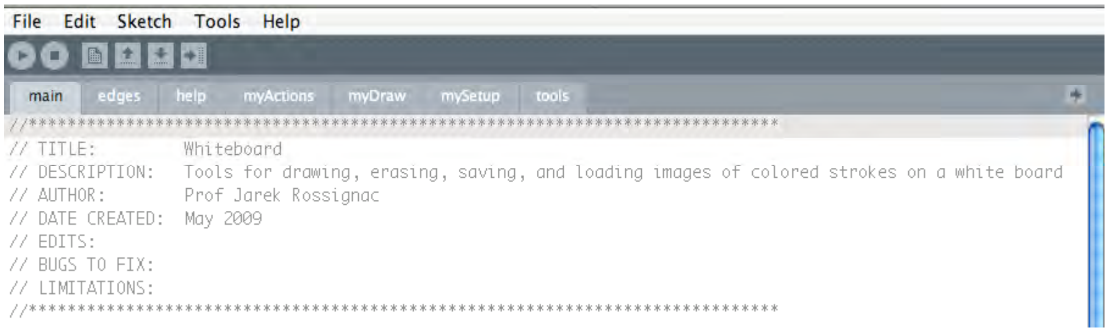
7.2 How to organize your sketch

Although our 53 lines program is relatively short for what it does, it has become difficult to read and work with. Here, we discuss how to structure it so as to facilitate improvements and how to add a user's help. The result of restructuring the code is posted on http://www.gvu.gatech.edu/~jarek/demos/whiteboard . Play with it first to understand what it does. Then, download the zipped folder, unzip it. Make a copy of the entire folder. Load it into Processing by double clicking the main.pde file that is in the main folder. Make sure that you can run it locally.

Notice that in the data folder, we have an image (me.jpg), a font (Courier-14.vlw) and a Pictures folder with two .jpg images.

Notice that it has several tabs, one per PDE file, listed on the top. The "main" tab is listed first because its name matches the name of the folder (they do not have to be called main, but we may as well adopt a convention). The other tabs are listed alphabetically. At the right, there is an arrow. You may use it to create new tabs or rename them.

Open the "main" tab. You should see this.



```
//************************************************************************************
// TITLE:          Whiteboard
// DESCRIPTION:    Tools for drawing, erasing, saving, and loading images of colored strokes on a white board
// AUTHOR:         Prof Jarek Rossignac
// DATE CREATED:   May 2009
// EDITS:
// BUGS TO FIX:
// LIMITATIONS:
//************************************************************************************
```

Edit the header by putting your name as co-author next to mine. Then look at the structure of the program.

I declare some variables and then define setup and draw.

One of the variable is used to toggle whether we are showing the help screen or not.

The other one is to print something in the bottom pane for debugging. It is set by a key action and reset at the end of draw to avoid printing things more than once.

```
boolean showHelpText=true;       // toggled by keys to show/hide help text
boolean printIt=false;            // temporarily set when key '?' is pressed and
used to print some debugging values
PImage me; // picture of student's face
```

Setup() loads a font that we use to draw text on the screen at a desired location. It also loads an image of me. You should replace that file (me.jpg) with an image of yourself. Headshot please: it helps me recognize you in class. That picture will be displayed in the help screen.

We then set some modes, open a window, and call mySetup, which is detailed in a separate tab.

```
void setup() {
  PFont font = loadFont("Courier-14.vlw"); textFont(font, 14);      // load
font
  me = loadImage("data/me.jpg");
  smooth(); strokeJoin(ROUND); strokeCap(ROUND);  rectMode(CENTER); // set up
window and drawing modes
  size(600, 600); // open large window
  mySetup();       // initialize
  }
```

`draw()` sets a yellow background and stroke width and either calls `showHelp` (detailed in the "help" tab) or "myDraw" detailed in the tab of that name. It resets `printIt` to avoid printing more than once when debugging.

```
void draw() { background(252,219,150);  strokeWeight(1);
// sets background
  if (showHelpText) showHelp(); else myDraw(); // show text help or draw
  printIt=false;
  };  // end of draw
```

Now, open the "help" tab. Edit the "Author: " line to add your name.

Find the descriptions of "`text`", "`popMatrix`", and "`translate`", study them and understand how the `showHelp` procedure works. Run the program as you study it to compare the code and what you see.

Note that the color for the "text" function is set by "`fill`" and not by "`stroke`". Here I set it to dark blue.

An improved version of this program is available at http://www.gvu.gatech.edu/~jarek/demos/paint/

7.3 How to load and display SVG sketches

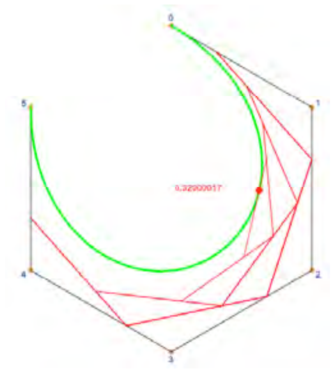
The loadShape() command is used to read simple SVG (Scalable Vector Graphics) files into a Processing sketch. It works for SVG files created from Adobe Illustrator, but may fail for anything else. Here's how you can use it.

```
PShape bot;

void setup() {
  size(640, 360);
  smooth();
  // The file "bot1.svg" must be in the data folder
  // of the current sketch to load successfully
  bot = loadShape("bot1.svg");
  noLoop(); // Only run draw() once
}
void draw(){
  background(102);
  shape(bot, 110, 90, 100, 100);  // Draw at (10, 10) at size 100 x 100
  shape(bot, 280, 40);            // Draw at (70, 60) at the default size
}
```

For detailed explanations, see: http://processing.org/learning/basics/loaddisplayshape.html

8 - The structure of an interactive graphic program

Since several of the projects deal include interactive manipulation of shapes and animations, we discuss here a few tools for editing an array P[] of control points and for drawing the corresponding Bezier curve. A screen shot is presented below. T applet:may edit the control points (move, insert, delete), may transform the whole curve (translate, scale), and may edit the parameter value t for which the construction is shown.



We look at the various tabs of an example applet.

8.1 Main

The "main" tab contains the usual header, declarations of some of the variables, and code for the `setup()` and `draw()`.

```
//*****************************************************************************
// TITLE:          PGS (PLANAR GEOMETRY SANDBOX)
// DESCRIPTION:    Template to demo manipulation of points and rendering of curves
// AUTHOR:         Prof Jarek Rossignac
// DATE CREATED:   August 2009
// EDITS:
//*****************************************************************************
boolean showHelpText=true;        // toggled by keys to show/hide help text
boolean printIt=false;
// temporarily set when key '?' is pressed. Used to print some debugging values

void setup() {
  size(600, 600); smooth();// set up window and drawing modes
  strokeJoin(ROUND); strokeCap(ROUND);
  setColors(); // calls my set-colors function
  PFont font = loadFont("ArialMT-24.vlw"); textFont(font, 12); // load font
  declarePoints(); resetPoints();  // initialize points in P[]
  }

void draw() {
  background(121);  strokeWeight(1); // sets background
  if (showHelpText) showHelp(); else myActions();
  printIt=false;
  };  // end of draw Rigid body transformations
```

8.2 UI

The UI tab contains most of the key-board interaction, some of the mouse actions, and the help display.

When keys 'm', 'i', 'd', 't', or 'z' keys are kept pressed, they modify the actions of `clickPolygon()` or `dragPolygon()`, which are defined in the "polygon" tab. The empty entries for these keys are listed here to remind the developer not to assign these keys to other actions.

```
void keyPressed() {
  if (key==' ') showHelpText=!showHelpText ;
  if (key=='m') {}  // used to move when mouse pressed
  if (key=='i') {}  // used to insert when mouse pressed
  if (key=='d') {}  // used to delete when mouse pressed
  if (key=='O') loopIsClosed=!loopIsClosed;
  if (key=='#') showVertexIds=!showVertexIds ;
  if (key=='V') showVertices=!showVertices;
  if (key=='S') {refine(0.5); coarsen();};
  if (key=='R') refine(0.5);
  if (key=='C') coarsen();
  if (key=='W') savePts();
  if (key=='G') loadPts();
  if (key=='X') {String S="images/P####.tif"; saveFrame(S);};   ;
  if (key=='?') printIt=true;  // toggle debug mode
  if (key==',') {n--; n=max(n,1); resetPoints(); println("n="+n); };
  if (key=='.') {n++; n=min(n,P.length); println("n="+n);resetPoints();};
  };
```
The `showHelp` procedure prints the help text on the graphic window when `showHelpText` is true.
```
void showHelp() {
    fill(dblue); pushMatrix(); translate(20,20);
     text("PGS (PLANAR GEOMETRY SANDBOX) by Jarek Rossignac",0,0); translate(0,20);
     text("  ",0,0); translate(0,20);
     text("First click in the window to activate it ",0,0); translate(0,20);
     text("Press SPACE to show/hide this help text",0,0); translate(0,20);
     …
 popMatrix(); noFill();
   }
```
The `mousePressed` is called each time the mouse button is pressed. It checks whether the mouse is in the window by calling the `isInWindow` method of the class `pt` on a point created at the current location of the cursor returned by `Mouse()`. Then, if a key is pressed, it calls `clickPolygon`, which is defined in the “polygon” tab.
```
void mousePressed() {if (Mouse().isInWindow()) if (keyPressed) clickPolygon();}
```

8.3 actions

The action tab declares and initializes global variables for the current parameter t selecting the point on the curve for which the construction is shown and the toggles that control whether the vertices and their IDs should be shown.
```
float t=0.5; // parameter for displaying a point (red dot) on the curve
 boolean showVertexIds=true, showVertices=true;
```
Then, in `myActions` it checks whether the user wants to edit the parameter t and if so increments it by the latest horizontal mouse displacement (difference between the current and the previous mouse coordinate), and then, depending on the display toggles, sets some rendering states (stroke color, fill color) and calls the desired display functions (defined in the “polygon” tab).
```
 void myActions() { // actions to be executed at each frame
    if(mousePressed&&keyPressed&&(key=='x')) t+=float(mouseX-pmouseX)/width;
    if ((mousePressed)&&(keyPressed)) dragPolygon();
    if (showVertices) {stroke(orange); fill(orange); showPoints(3); noFill(); };
    if (showVertexIds) {fill(dblue); drawPointIDs(); noFill(); };
    noFill(); stroke(black); drawPolygon(false);
    stroke(green); strokeWeight(3);
    beginShape(); for(float s=0; s<=1; s+=0.01) bez(0,s,n-1).v(); endShape();
    strokeWeight(1); fill(red); stroke(red);
    pt Q=bezD(0,t,n-1); show(Q,4); label(P(width/2,height/2),str(t));
    };
```
The line `beginShape(); for(float s=0; s<=1; s+=0.01) bez(0,s,n-1).v(); endShape();`

renders the Bezier curve by advancing the value of the s parameter, by calling `bez()` to compute the corresponding point on the Bezier curve, and by calling the `.v()` method, which is used for drawing polygonal curves, when executed between `beginShape()` and `endShape()`. Note that `bez` uses the global array `P[]` of control points declared in “polygon”.

8.4 Bezier

This part deals explicitly with the evaluation of the points on the Bezier curve and with the display of the recursive construction process.

`bez` evaluates a point on a Bezier curve of degree r for parameter value t that is defined by control polygons P[i], P[i+1]…P[i+r-1]. Note that it performs a linear interpolation (`L()`) between points on two Bezier curves of a lower degree, but with different control points (shifted by one along the control polygon).
```
pt bez(int i, float t, int r) {
  if(r==0) return P[i]; // zero degree Bezier curve returns the control point
  return L(bez(i,t,r-1),t,bez(i+1,t,r-1));}
```

The function `bezD` is a modified version of bez to display the construction lines for the desired point.

```
pt bezD(int i, float t, int r) { // to show the construction
  if(r==0) {stroke(red); return P[i];};
  if(r>1) show(bezD(i,t,r-1),bezD(i+1,t,r-1)); // draw line used in interpolation
  return L(bezD(i,t,r-1),t,bezD(i+1,t,r-1));}
```

8.5 polygons

The "polygon" tab contains the code for processing points, in the global array `P[]` and for drawing and processing the polygonal curve they represent. I have deliberately set this as a set of functions operating on a global array `P[]` rather than as a polygon class, so as to facilitate prototyping. However, in more advanced applets, I prefer to use a polygon class, which has functions for copying and blending between polygons.

```
boolean loopIsClosed = false;  // states whether the polygon curve is a closed loop
```

Global variable `n` denotes the current number of control points. Note that I declare a large array to simplify insertion of additional control points. This may be avoided by dynamically extending the array.

```
int n=6;                        // current number of control points
pt [] P = new pt[1000];            // declares an array of 1000 vertices
```

To keep track of which control point was selected when the mouse was most recently pressed, I use the global variable `p`.

```
int p=0;                    // index to the currently selected vertex being dragged
```

When accessing neighbors around a closed-loop control polygon, I use the `next` and `prev` functions.

```
int next(int j) {if(j==n-1) {return (0);} else{return(j+1);}}; // next in loop
int prev(int j) {if (j==0) {return (n-1);} else{return(j-1);}  }; // previous
```

The array `P` will contain references to points (instances of class `pt`) each point must be created using a new constructor of the `P()` function discussed in the "geo2D" tab.

```
void declarePoints() {for (int i=0; i<P.length; i++) P[i]=new pt();}
```

The procedure `resetPoints` is used to set, or reset, the control points evenly spaced around a circle.

```
void resetPoints() {  // init the points to be on a circle
for (int i=0; i<n; i++) {
   P[i]=new pt(width/2,height/7.); P[i].rotateBy(-2.*PI*i/n, screenCenter());}; }
```

When reading from file or building the control points one by one, we may use `appendPoint`.

```
void appendPoint(pt Q)  { P[n++].setTo(Q);  }; // add point at end of list
```

To edit the set of control points, we use the following pair of procedures, which test which key is being held down and call the appropriate actions. Remember that `clickPolygon` is invoked in the "UI" tab each time the mouse button is pressed. When 'm' or 'd' keys is pressed, the mouse click results in picking the closest point to the mouse, which is recorded in the variable `p`. When the 'i' keys is pressed, instead of finding the closest point, I call my `insertPoint` procedure, which finds the closest edge and inserts a new control points between the end-points of that edge.

```
void clickPolygon() {
   if (key=='m') pickClosestPoint();
   if (key=='i') insertPoint();
   if (key=='d') {pickClosestPoint(); deletePoint();};     }
```

When the mouse button is pressed and a key is pressed, in `myAction`, I call `dragPolygon`. It either drags the selected (or inserted and automatically selected) control points or performs a global translation or rotation of all the points. Instead of changing the points, it may be desired to use a viewing transformation.

```
void dragPolygon() {
   if (key=='i') dragPoint();
   if (key=='m') dragPoint();
   if (key=='t') translatePoints(mouseDrag());
   if (key=='z') scalePoints(-dot(mouseDrag(),Mouse().makeVecToCenter())/10000.);
   };
```

Here are two variations of procedures that compute the index `p` of the closest point to the mouse:

```
void pickClosestPoint(pt M) {if(mouseIsInWindow()) { p=0; for (int i=1; i<n; i++)
if (M.disTo(P[i])<M.disTo(P[p])) p=i; };}
void pickClosestPoint() {pt M = Mouse(); p=0; for (int i=1; i<n; i++) if
(M.disTo(P[i])<M.disTo(P[p])) p=i;}
```

Here are two variations of utilities for dragging the picked point `P[p]` with the mouse:

```
void dragPoint() { P[p].moveWithMouse(); P[p].clipToWindow(); }
      // moves selected point (index p) by amount mouse moved recently
void dragPoint(vec V) {P[p].translateBy(V);}
```

Point deletion is performed by copying the portion of P[p+1…P[n-1] to P[p]…P[n-2]: and decrementing n.

```
void deletePoint() {
      for (int i=p; i<n-1; i++) P[i].setTo(P[next(i)]); n--; p=prev(p);}
```

To insert a point, we first compute the closest edge and insert it after the first point on that edge, which requires copying the following points one notch further in `P`.

```
void insertPoint() {  // adds vertex at closest edge
     pt M = Mouse();
     p=0; for (int i=1; i<n; i++) if (M.disTo(P[i])<M.disTo(P[p])) p=i;
     int e=-1;
     float d = M.disTo(P[p]);
     for (int i=0; i<n; i++)
        if ( (0.25<M.ratioOfProjectionBetween(P[i],P[next(i)])) &&
             (M.ratioOfProjectionBetween(P[i],P[next(i)])<0.75) &&
             (M.disToLine(P[i],P[next(i)])<d))
          {e=i; d=M.disToLine(P[e],P[next(e)]);};
        if (e!=-1){for (int i=n-1; i>e; i--) P[i+1].setTo(P[i]);
        n++; p=next(e); P[p].setToMouse();
        };         }
```

The following procedures are used to transform all control points

```
void translatePoints(vec V) {for (int i=0; i<n; i++) P[i].translateBy(V); };
void scalePoints(float s) {
   for (int i=0; i<n; i++) P[i].translateTowards(s,screenCenter());};
```

The following procedures are used to draw the vertices or the edges of the polygon, as an open or closed loop.

```
void showPoints() {for (int i=0; i<n; i++) P[i].show();}
void showPoints(int r) {for (int i=0; i<n; i++) P[i].show(r);}
void drawPointIDs() {
  for (int i=0; i<n; i++) {
    vec V=P[i].makeVecToAverage(P[prev(i)],P[next(i)]);
    V.normalize(); V.scaleBy(-10);
    V.add(-3,5);
    P[i].showLabel(str(i),V); };};
void drawPolygon() {
    beginShape();  for (int i=0; i<n; i++) P[i].v(); endShape(CLOSE); }
void drawPolygon(boolean closed) {
    beginShape();
    for (int i=0; i<n; i++) P[i].v();
    if(closed) endShape(CLOSE); else endShape();}
```

The following procedure computes the arc-length of the polygon.

```
float length () {float L=0;
    if (!loopIsClosed) for (int i=2; i<n-3; i++) L+=P[i].disTo(P[next(i)]);
    else for (int i=0; i<n; i++) L+=P[i].disTo(P[next(i)]);
    return(L); }
```

The following procedure computes the closest projection of a point M onto the polygon.

```
pt projectionOfPoint(pt M) {
     int v=0; for (int i=1; i<n; i++) if (M.disTo(P[i])<M.disTo(P[v])) v=i;
     int e=-1;
     float d = M.disTo(P[v]);
     for (int i=0; i<n; i++)
```

```
        if(M.projectsBetween(P[i],P[next(i)])&&(M.disToLine(P[i],P[next(i)])<d))
            {e=i; d=M.disToLine(P[e],P[next(e)]);};
    if(e!=-1)return(M.makeProjectionOnLine(P[e],P[next(e)]));
    else return(P[v].makeClone()); }
```

The following procedure computes the distance from a point M to its closest projection onto the polygon.

```
 float distanceToPoint(pt M) {return(M.disTo(projectionOfPoint(M)));}
```

The following procedure establishes whether point M is contained in the interior region bounded by the closed loop polygon. Does it require that the polygon be oriented in a particular clockwise or counterclockwise manner?

```
boolean polygonContains(pt M) {boolean isIn=false;
      for (int i=1; i<n-1; i++)
            if (M.isInTriangle(P[0],P[i],P[next(i)])) isIn=!isIn; return(isIn); }
```

The following procedure refines the polygon by doubling the number of its vertices. It assumes that the polygon is a closed loop. It uses the `b()` and `f()` masks to compute the positions of the new vertices. These may be implemented to support the four-point subdivision, the quintic Bezier subdivision, and a whole family of intermediate schemes, including the cubic Bezier subdivision and the Jarek subdivision.

```
void refine(float s) {
      pt[] Q = new pt [2*n];
      for (int i=0; i<n; i++) {
            Q[2*i]=b(P[prev(i)],P[i],P[next(i)],s);
            Q[2*i+1]=f(P[prev(i)],P[i],P[next(i)],P[next(next(i))],s); };
      n*=2;
      for (int i=0; i<n; i++) P[i].setTo(Q[i]);          }
```

The following procedure removes every other point from `P`.

```
void coarsen() {n/=2; for (int i=0; i<n; i++) P[i].setTo(P[2*i+1]); }
```

The following procedures return true if the closed loop polygon is stabbed by edge (A,B). The second one ignores edge j in the test and is useful for testing whether a polygon is self-intersecting.

```
boolean stabbed(pt A, pt B) {
      boolean stab=false;
      for  (int  i=0;  i<n;  i++)  if(edgesIntersect(A,B,P[i],P[next(i)]))  stab=true;
      return stab; }
boolean stabbed(pt A, pt B, int j) {
      boolean stab=false;
      for(int i=0; i<n; i++)
         if((i!=j)&&(edgesIntersect(A,B,P[i],P[next(i)]))) stab=true;
      return stab; }
```

The following procedures are used to save the control points on file and to load them back.

```
void savePts() {savePts("P.pts");}
void savePts(String fn) {
      String [] inppts = new String [n+1];
      int s=0;
      inppts[s++]=str(n);
      for (int i=0; i<n; i++) {inppts[s++]=str(P[i].x)+","+str(P[i].y);};
      saveStrings(fn,inppts);  };
void loadPts() {loadPts("P.pts");}
void loadPts(String fn) {
      String [] ss = loadStrings(fn);
      String subpts;
      int s=0; int comma; n = int(ss[s]);
      for(int i=0;i<n; i++) { comma=ss[++s].indexOf(',');
      P[i]=new pt (float(ss[s].substring(0, comma)),
                   float(ss[s].substring(comma+1, ss[s].length()))); }; };
```

8.6 colors

The "colors" tab declares and initializes a set of useful color names and sets the color scheme to HSB with a range of 121.

```
color red, yellow, green, cyan, blue, magenta, dred, dyellow, dgreen, dcyan, dblue,
dmagenta, white, black, orange, grey, metal;  // declares color names
void setColors() {
  colorMode(HSB,121);
  red = color(0, 120, 120); green = color(40, 120, 120);
   …
```

8.7 geo2D

The “geo2D” tab contains functions and classes for points and vectors, and a few other useful geometric tools things. It is too long to include here. I only include the most common calls for points and vectors.

The following functions and procedures deal with points in the plane.

```
// create
pt P(float x, float y) {return new pt(x,y); }; // make point (x,y)
pt P() {return P(0,0); };// make point (0,0)
pt P(pt P) {return P(P.x,P.y); }; // make copy of point P
```

```
// combine
pt S(pt A, pt B) {return new pt(A.x+B.x,A.y+B.y); };// Weighted sum: A+B
pt S(pt A, pt B, pt C) {return S(A,S(B,C)); }; // Weighted sum: A+B+C
pt S(pt A, pt B, pt C, pt D) {return S(S(A,B),S(C,D)); };// Weighted sum: A+B+C+D
pt S(float s, pt A) {return new pt(s*A.x,s*A.y); }; // Weighted sum: sA
pt S(pt A, float s, pt B) {return S(A,S(s,B)); }; // Weighted sum: A+sB
pt L(pt A, float s, pt B) {return P(A.x+s*(B.x-A.x),A.y+s*(B.y-A.y)); }; // A+sAB
pt S(float a, pt A, float b, pt B) {return S(S(a,A),S(b,B));} //: aA+bB
pt S(float a, pt A, float b, pt B, float c, pt C) {return S(S(a,A),S(b,B),S(c,C));}
pt S(float a, pt A, float b, pt B, float c, pt C, float d, pt D){
      return A(S(a,A,b,B),S(c,C,d,D));}   // Weighted sum: aA+bB+cC+dD
pt A(pt A, pt B) {return P((A.x+B.x)/2.0,(A.y+B.y)/2.0); }; // Average: (A+B)/2
pt A(pt A, pt B, pt C){return P((A.x+B.x+C.x)/3.0,(A.y+B.y+C.y)/3.0);};// (A+B+C)/3
```

```
// transform
pt R(pt Q, float a) {float dx=Q.x, dy=Q.y, c=cos(a), s=sin(a);
  return new pt(c*dx+s*dy, -s*dx+c*dy); }; // Rotated Q by a around origin
pt R(pt Q, float a, pt P) {float dx=Q.x-P.x, dy=Q.y-P.y, c=cos(a), s=sin(a);
  return P(P.x+c*dx-s*dy, P.y+s*dx+c*dy); }; // Rotated Q by a around P
pt T(pt P, vec V) {return P(P.x + V.x, P.y + V.y); } // Translate: P+V
pt T(pt P, float s, vec V) {return T(P,S(s,V)); }  // Translate: P+sV
pt T(pt A, float s, pt B) { return T(A,s,U(V(A,B))); };
                  // Translate: P by s towards Q: P+sU(PQ)
```

```
// measure
boolean isSame(pt A, pt B) {return (A.x==B.x)&&(A.y==B.y) ;} // Equality: A==B
boolean isSame(pt A, pt B, float e) {return ((abs(A.x-B.x)<e)&&(abs(A.y-B.y)<e));}
                  // Equality: ||A-B||<e
float d(pt P, pt Q) {return sqrt(d2(P,Q));  };// Distance: ||AB||
float d2(pt P, pt Q) {return sq(Q.x-P.x)+sq(Q.y-P.y); };// Distance squared: AB*AB
```

```
// render
void v(pt P) {vertex(P.x,P.y);}; // next point when drawing polygons
void cross(pt P, float r) {line(P.x-r,P.y,P.x+r,P.y); line(P.x,P.y-r,P.x,P.y+r);};
                   // shows P as cross of length r
void cross(pt P) {cross(P,2);}; // shows P as small cross
void show(pt P, float r) {ellipse(P.x,P.y,2*r,2*r);}; // draws circle of radius r
void show(pt P) {ellipse(P.x, P.y, 4,4);}; // draws small circle around point
void show(pt P, pt Q) {line(P.x,P.y,Q.x,Q.y); };// draws edge (P,Q)
void arrow(pt P, pt Q) {arrow(P,V(P,Q)); } // draws arrow from P to Q
void label(pt P, String S) {text(S, P.x+5,P.y+4); } // writes S next to P
```

```
// mouse & screen
pt Mouse() {return P(mouseX,mouseY);}; // current mouse location
pt Pmouse() {return P(pmouseX,pmouseY);}; // previous mouse location
vec mouseDrag() {return V(mouseX-pmouseX,mouseY-pmouseY);}; // vector of mouse-drag
```

```
pt mouseInWindow() {float x=mouseX, y=mouseY; x=max(x,0); y=max(y,0);
 x=min(x,height); y=min(y,height); return P(x,y);}; // clips mouse to square window
pt screenCenter() {return P(height/2,height/2);} // point in center of screen
boolean mouseIsInWindow()// if mouse is in square window
     {return(((mouseX>0)&&(mouseX<height)&&(mouseY>0)&&(mouseY<height)));};
```

The following functions and procedures deal with two dimensional vectors.

```
// create
vec V(vec V) {return new vec(V.x,V.y); }; // make copy of vector V
vec V(float x, float y) {return new vec(x,y); };// make vector (x,y)
vec V(pt P, pt Q) {return new vec(Q.x-P.x,Q.y-P.y);}; // returns PQ = Q-P
vec MouseDrag() {return new vec(mouseX-pmouseX,mouseY-pmouseY);};
                                   // vector representing recent mouse displacement
// combine
vec S(vec U, vec V) {return new vec(U.x+V.x,U.y+V.y);} // U+V
vec S(float s,vec V) {return new vec(s*V.x,s*V.y);}; // sV
vec S(float a, vec U, float b, vec V) {return S(S(a,U),S(b,V));} // aU+bV
vec S(vec U,float s,vec V) {return new vec(U.x+s*V.x,U.y+s*V.y);}; // U+sV
vec A(vec U, vec V) {return new vec((U.x+V.x)/2.0,(U.y+V.y)/2.0); };// (U+V)/2
vec L(vec U,float s,vec V) {return new vec(U.x+s*(V.x-U.x),U.y+s*(V.y-U.y));};
                               // Linear interpolation between vectors: (1-s)U+sV
// transform
vec U(vec V) {float n = n(V); // Unit vector: V/||V||
   if (n==0) return new vec(0,0); else return new vec(V.x/n,V.y/n);};
vec R(vec V) {return new vec(-V.y,V.x);}; // V turned right 90 degrees (on screen)
vec R(vec U, float a) {vec W = U.makeRotatedBy(a); return W ;}; // U rotated by a
vec R(vec U, float s, vec V) {float a = angle(U,V); vec W = U.makeRotatedBy(s*a);
    float u = n(U); float v=n(V); S((u+s*(v-u))/u,W); return W ; };
            // Rotate: interpolation (angle and length) between U and V
// Measure
float n(vec V) {return sqrt(sq(V.x)+sq(V.y));}; // ||V||
float n2(vec V) {return sq(V.x)+sq(V.y);}; // V*
float dot(vec U, vec V) {return U.x*V.x+U.y*V.y; }; //U*V
boolean parallel (vec U, vec V) {return dot(U,R(V))==0; }; // true if parallel
// render
void show(pt P, vec V) {line(P.x,P.y,P.x+V.x,P.y+V.y);} // show V (line) from P
void show(pt P, float s, vec V) {show(P,S(s,V));} // show sV from P
void arrow(pt P, vec V) {();          // show arrow for V starting at P
  show(P,V);  float n=n(V); float s=max(min(0.2,20./n),6./n
  pt Q=T(P,V); vec U = S(-s,V); vec W = R(S(.3,U));
  beginShape(); v(T(T(Q,U),W)); v(Q); v(T(T(Q,U),-1,W)); endShape(CLOSE);};
void arrow(pt P, float s, vec V) {arrow(P,S(s,V));} // show arrow from P along sV
```

The following functions and procedure deal with angles.

```
float angle (vec U, vec V) {return atan2(dot(R(U),V),dot(U,V)); };
                                   // angle <U,V> between -PI and PI
float angle(vec V) {return(atan2(V.y,V.x)); };// angle <<1,0>,V> between -PI and PI
float angle(pt A, pt B, pt C) {return  angle(V(B,A),V(B,C)); } // angle <BA,BC>
float turnAngle(pt A, pt B, pt C) {return  angle(V(A,B),V(B,C)); }
                 // angle <AB,BC> (positive when right turn as seen on screen)
int toDeg(float a) {return int(a*180/PI);}
float toRad(float a) {return(a*PI/180);}
float positive(float a) { if(a<0) return a+TWO_PI; else return a;}
```

The following function tests whether edge(A,B) intersects edge(C,D).

```
boolean edgesIntersect(pt A, pt B, pt C, pt D) {boolean hit=true;
    if (isRightTurn(A,B,C)==isRightTurn(A,B,D)) hit=false;
    if (isRightTurn(C,D,A)==isRightTurn(C,D,B)) hit=false;
    return hit; }
```

The following function tests whether the polyline (A,B,C) makes a right turn at B (as seen on the screen).

```
boolean isRightTurn(pt A, pt B, pt C) {
  return dot( R(V(A,B)),V(B,C)) > 0 ; };
```

References

A short online tutorial: http://processing.org/learning/gettingstarted/

Resources

I strongly suggest that you get the MIT Press book “Processing A Programming Handbook for Visual Designers and Artists” by Casey Reas and Ben Fry. It teaches the basics of Processing and explains many of its extensions (to images, videos, audio, web…).