Processing Tutorial

How to make a really cool spaceshooter in Processing

Nikolaus Gradwohl, local-guru.net <guru@local-guru.net>

Revision History

Revision 0.5 2009-06-19

First published beta

Table of Contents

intro

Processing is an awsome little programming language. In Processing one get's really great result with little effort. So I decided to lower the entry bar a little bit by writing this processing tutorial.

the current version of Processing and it's install instructions can be found at *http://www.processing.org/*

The current version of this tutorial can be found in my blog at *http://www.local-guru.net/*

make a starfield

every cool spaceshooter has some stars scrolling by in the background. So lets start and make one. Start processing and make a new sketch. add the following codde and hit the play button (don't worry if you don't understand the code yet - i'll explain it in a moment)

```
Star stars[];
int STARS = 100;

void setup() {
  size(640,480);
  stars = new Star[STARS];
  for ( int i =0; i < STARS; i++) {
    stars[i] = new Star( random( width ), random( height ), random( 10 ));
  }
  frameRate( 25 );
  smooth();
}

void draw() {
  background(0);

  strokeWeight( 2 );
  for ( int i =0; i < STARS; i++) {
    stroke( stars[i].z * 25 );
    point( stars[i].x, stars[i].y );

    stars[i].x -= stars[i].z;
    if (stars[i].x < 0) {
      stars[i] = new Star( width, random( height ), sqrt(random( 100 )));
    }
  }
}

class Star {
  float x, y, z;
  Star( float x, float y, float z ) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
}
```

every processing sketch has 2 methods setup() and draw(). The code in the setup gets executed once when the programm is startet. In the code above we set the size of the window to 640x480 pixesl and initialize a Array full of Star objects. then the frameRate is set to 25 and we tell processing to smooth the edges of our drawings.

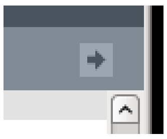
the code in draw gets executed every frame and is used to refresh the content of our window. first we set the background to black. Then we set the line weight to 2 pixel and draw a dot for every star in our array. The nearer a star is the brighter it will be drawn. stroke sets the line color, point draws a point on the screen.

after the point is drawn to the screen, the position of the star gets updated. We want stars that are nearer to move faster. So we simply subtract the content of the z coordinate from the x coordinate. If a star falls out of the left edge of our window we replace it with a new one on the right side on at a random height and at a random distance.

because nearer stars fall of faster then the ones who are further away we don't simply set the z coordinate to a random value but set the range of the random to 100 and take the squareroot. this makes it more likely that 'near' stars are generated.

make a starfield class

now that we have the starfield up and running, lets clean up the code a bit. first generate a new tab using this button



and name it "starfield". In the new text editor panel add the following code

```
public class Starfield {
  private Star stars[];
  private int count;

  public Starfield( int count ) {
    this.count = count;
    stars = new Star[count];
    for ( int i =0; i < count; i++) {
      stars[i] = new Star( random( width ), random( height ), random( 10 ));
    }
  }

  public void draw() {
    strokeWeight( 2 );
    for ( int i =0; i < count; i++) {
      stroke( stars[i].z * 25 );
      point( stars[i].x, stars[i].y );

      stars[i].x -= stars[i].z;
      if (stars[i].x < 0) {
        stars[i] = new Star( width, random( height ), sqrt(random( 100 )));
      }
    }
  }
}

class Star {
  float x, y, z;
  Star( float x, float y, float z ) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
}
```

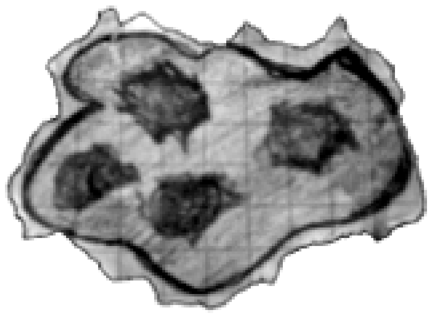
now switch to the first panel and change your main-programm to this:

```
Starfield starfield;

void setup() {
  size(640,480);
  starfield = new Starfield( 100 );
  frameRate( 25 );
  smooth();
}

void draw() {
  background(0);
  starfield.draw();
}
```

our programm now doesn't need to anything about starfields any more, it just instanciates one and calls the draw method of the starfield in every frame. This makes it much easyer to add code to the main programm and keeps it readable (= maintainable - every code that gets written has to be changed someday - beleve me)

add a comet

now that our starfield is in place and our code is tidy, we gonna add something to shoot at - a comet. Add the image of a comet to the sketch by selecting "Add file ..." from the "Sketch menu, and add a great image of a comet like this one



Then change the main program to look like this.

```
Starfield starfield;
PImage comet;
int pos;

void setup() {
  size(640,480);
  starfield = new Starfield( 100 );

  comet = loadImage( "comet.png" );
  pos = width + comet.width/2;

  frameRate( 25 );
  smooth();
}

void draw() {
  background(0);
  starfield.draw();

  pos -= 3;
  if ( pos < -comet.width/2 ) {
      pos = width + comet.width/2;
  }

  pushMatrix();
  translate ( pos, 240 );
  image( comet, -comet.width/2, -comet.height/2 );
  popMatrix();
}
```

we add a variable to save the image of the comet, and a second one to save the current position of the comet. In the setup method the position is set to a value right of the viewable area. in the draw method the position is decremented by 3. Then we check if the comet is still visible, if not the position will be set beond the right border of the window again. Then the image of the comet will be drawn to the screen.

The pushMatrix and popMatrix commands are used to generate/restore a new coordinate system which is moved by translate - sounds complicated but it's really usefull, just use it for a while it comes clear what these commands do after a while. Make sure that pushMatrix and popMatrix always come in pairs.

You can imagine pushMatrix/popMatrix like adding a transparent layer on top of a drawing, which can be rotated or scaled independent of the paper it self. whatever you draw on the paper won't be affected by the rotating or scaling, but whatever is drawn on the transparent layer is rotated. so everything that is drawn outside the pushMatrix/popMatrix pair is not affected by our rotate, scale or translate - every thing that is drawn inside the pushMatrix/popMatrix pair (like our comet) is affected by rotate, scale or translate.

make it rotate

now we refactor our comet to another class - because our program doesn't care about drawing comets either. We also want our comet to rotate. so click on 'new tab' again and name the tab "comet" and enter the following code

```
public class Comet {
  int x;
  int y;
  int speed;
  PImage img;
  float alph;

  public Comet( int x, int y, int speed, PImage img ) {
    this.x = x;
    this.y = y;
    this.speed = speed;
    this.img = img;
    this.alph = 0;
  }

  public void update() {
    alph-= 0.1;
    x -= speed;
    if ( x < - img.width/2 ) {
      x = width + img.width/2;
      y = int( random( height - img.height )  + img.height/2 );
    }
  }

  public void draw() {
    pushMatrix();
    translate ( x, y );
    rotate( alph );
    image( img, -img.width/2, -img.height/2 );
    popMatrix();
  }
}
```

in this class we seperated the updating of the comets position and rotation angle, and the drawing stuff into 2 methods update and draw. the draw method contains a rotate command which does the rotation magic. notice that the rotate comes after the translate stuff, otherwise the comet would wirrl around very wierdly and would be most of the time out of the viewable area.

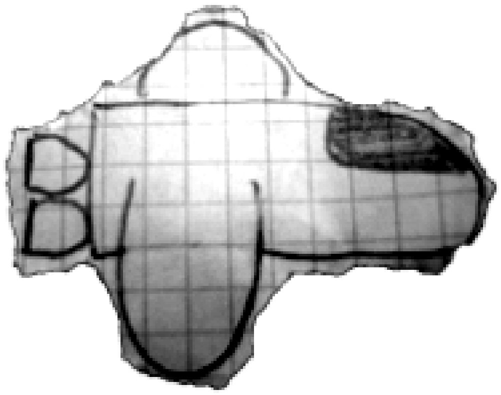
now change your main program to

```
Starfield starfield;
Comet comet;

void setup() {
  size(640,480);
  starfield = new Starfield( 100 );

  PImage img = loadImage( "comet.png" );
  comet = new Comet( width + img.width/2, height/2, 3, img );

  frameRate( 25 );
  smooth();
}

void draw() {
  background(0);
  starfield.draw();
  comet.update();
  comet.draw();
}
```

we define a new variable for the comet like we did for our starfield. and initialize it in the setup method. in the draw method update and draw get called.

add a ship

now that we have somenthing to shoot at we need someone who shoots - so lets add a ship. Again use "Add file ..." from the sketch menu to add an image like this one



this time we already know how to make a class for showing sprites so we don't fiddle around in the main program and clean up later but start with making a class. so click on "new tab" and name it "ship". then add the following code

```
public class Ship {
  PImage img;

  int x;
  int y;

  public Ship( int x, int y, PImage img ) {
    this.x = x;
    this.y = y;
    this.img = img;
  }

  void draw() {
    pushMatrix();
    translate ( x, y );
    image( img, -img.width/2, -img.height/2 );
    popMatrix();
  }
}
```

this is a simplified version of the class we wrote for our comet, since our ship doesn't rotate and it is not moving yet. then change your main programm to look like this

```
Starfield starfield;
Comet comet;
Ship ship;

void setup() {
  size(640,480);
  starfield = new Starfield( 100 );

  PImage cimg = loadImage( "comet.png" );
  comet = new Comet( width + cimg.width/2, height/2, 3, cimg );

  PImage simg = loadImage( "ship.png" );
  ship = new Ship( 100, height/2, simg );

  frameRate( 25 );
  smooth();
}

void draw() {
  background(0);
  starfield.draw();
  comet.update();
  comet.draw();
  ship.draw();
}
```

make it move

now we wan't our ship to move up and down when we press the cursor keys to move out of the way (well - at least as long as we can't shoot back ;-)). so change your "ship" class to

```
public class Ship {
  PImage img;
  int speed;
  int x;
  int y;

  public Ship( int x, int y, int speed, PImage img ) {
    this.speed = speed;
    this.x = x;
    this.y = y;
    this.img = img;
  }

  void draw() {
    pushMatrix();
    translate ( x, y );
    image( img, -img.width/2, -img.height/2 );
    popMatrix();
  }

  void up() {
    y -= speed;
    if ( y < img.height/2 ) { y = img.height/2; }
  }

  void down() {
    y += speed;
    if ( y > height - img.height/2 ) { y = height - img.height/2; }
  }

}
```

we added 2 new methodes up and down which will be called if the user presses the cursor buttons. since noone calls them yet lets change our main program to this

```
Starfield starfield;
Comet comet;
Ship ship;

void setup() {
  size(640,480);
  starfield = new Starfield( 100 );

  PImage cimg = loadImage( "comet.png" );
  comet = new Comet( width + cimg.width/2, height/2, 4, cimg );

  PImage simg = loadImage( "ship.png" );
  ship = new Ship( 100, height/2, 3, simg );
  frameRate( 25 );
  smooth();
}

void draw() {
  background(0);
  starfield.draw();
  comet.update();
  comet.draw();
  if ( keyPressed == true && key == CODED ) {
     if ( keyCode == UP ) {
       ship.up();
     } else if ( keyCode == DOWN ) {
       ship.down();
     }
  }
  ship.draw();
}
```

in the draw method we are checking if a key is pressed and if the key is 'CODED'. in progressing the pressed keys are stored in the key variable so if anyone presses an 'a' keyPressed is set to 'true' and key gets the value 'a'. but if any of the keys is pressed that isn't associated with an character like the cursor keys or the function keys, the value of key is set to 'CODED' and the code of the key is stored in the keyCode variable. So we check if the user has pressed the up or down key on the cursor block and move our ship.

let them collide

now that our ship is moving, let's build in collision detection and make our ship explode if it is hit by the comet. first we define a new class named box. Make a new tab and name it 'box'. Enter the following code

```
public class Box {
  int x1, x2;
  int y1, y2;

  Box( int x1, int y1, int x2, int y2 ) {
    this.x1 = x1;
    this.y1 = y1;

    this.x2 = x2;
    this.y2 = y2;
  }

  boolean isOverlap( Box b ) {
    if ((( x1 <= b.x1 && b.x1 <= x2 ) || ( x1 <= b.x2 && b.x2 <= x2 ))
     && (( y1 <= b.y1 && b.y1 <= y2 ) || ( y1 <= b.y2 && b.y2 <= y2 ))) {
      return true;
    }
    return false;
  }
}
```

this class defines a box by defining the coordinates of its endpoints. it also provides a method to check if two of them overlap. now we change our ship class and our comet class to return their bounding boxes.

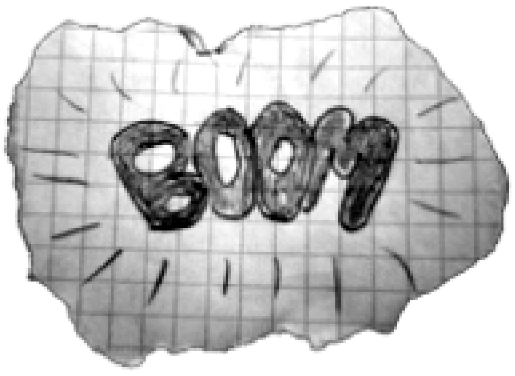
the ship class should look like this

```
public class Ship {
  PImage img;
  int speed;
  int x;
  int y;

  public Ship( int x, int y, int speed, PImage img ) {
    this.speed = speed;
    this.x = x;
    this.y = y;
    this.img = img;
  }

  void draw() {
    pushMatrix();
    translate ( x, y );
    image( img, -img.width/2, -img.height/2 );
    popMatrix();
  }

  void up() {
    y -= speed;
    if ( y < img.height/2 ) { y = img.height/2; }
  }

  void down() {
    y += speed;
    if ( y > height - img.height/2 ) { y = height - img.height/2; }
  }

  public Box getBox() {
    return new Box( x - img.width/2, y-img.height/2, x+img.height/2, y+img.heig
  }
}
```

and the comet class should look like this

```
public class Comet {
  int x;
  int y;
  int speed;
  PImage img;
  float alph;

  public Comet( int x, int y, int speed, PImage img ) {
    this.x = x;
    this.y = y;
    this.speed = speed;
    this.img = img;
    this.alph = 0;
  }

  public void update() {
    alph-= 0.1;
    x -= speed;
    if ( x < - img.width/2 ) {
      x = width + img.width/2;
      y = int( random( height - img.width ) + img.width/2);
    }
  }

  public void draw() {
    pushMatrix();
    translate ( x, y );
    rotate( alph );
    image( img, -img.width/2, -img.height/2 );
    popMatrix();
  }

  public Box getBox() {
    return new Box( x - img.width/2, y-img.height/2, x+img.height/2, y+img.heig
  }
}
```

now let's add an image for the explosion



and change the main code to look like this

```
Starfield starfield;
Comet comet;
Ship ship;

PImage boom;

void setup() {
  size(640,480);
  starfield = new Starfield( 100 );

  PImage cimg = loadImage( "comet.png" );
  comet = new Comet( width + cimg.width/2, height/2, 4, cimg );
  PImage simg = loadImage( "ship.png" );
  ship = new Ship( 100, height/2, 3, simg );

  boom = loadImage( "boom.png" );

  frameRate( 25 );
  smooth();
}

int boom_count = 0;

void draw() {
  background(0);
  starfield.draw();
  comet.update();
  if ( boom_count == 0 ) {
    if ( keyPressed == true && key == CODED ) {
       if ( keyCode == UP ) {
         ship.up();
       } else if ( keyCode == DOWN ) {
         ship.down();
       }
    }
    ship.draw();
  } else {
    image( boom, ship.getBox().x1, ship.getBox().y1 );
    boom_count--;
  }

  if ( ship.getBox().isOverlap( comet.getBox())) {
    boom_count = 25;
  }
  comet.draw();
}
```

here we introduce a new variable boomcount that is set to 25 when the bounding box of the ship and the comet overlap. if the variable is not null we show the boom image instead of the ship and decrement the boomcount. this ensures that the explosion is visible at least for 25 frames (=1 second since we have set the framerate to 25 frames per second)

start shooting back

now let's add a torpedo that can be shot by the ship. Add a new tab, name it "torpedo" and add the following code

```
public class Torpedo {
  PImage img;
  int x;
  int y;
  boolean active;

  public Torpedo(int s, int x, int y) {
    img = makeTexture( s );
    this.x = x;
    this.y = y;
    this.active = true;
  }

  void update() {
    if (active) {
      x+=8;
      if (x > width) {
        active = false;
      }
    }
  }

  void draw() {
    if (active) {
      blend( img, 0, 0, img.width, img.height, int(x) - img.width/2, int(y) - i
    }
  }

  public Box getBox() {
    return new Box( x - 20, y-20, x+20, y+20);
  }

  PGraphics makeTexture( int r ) {
    PGraphics res = createGraphics( r * 6, r * 6, P2D);
    res.beginDraw();
    res.loadPixels();
      for ( int x = 0; x < res.width; x++) {
        for( int y = 0; y < res.height; y++ ) {
          float d = min( 512, 50*  sq( r / sqrt( sq( x - 3 * r) + sq( y - 3 * r
          res.pixels[y * res.width + x] = color( min(255,d), min(255, d*0.8), d
        }
      }
    res.updatePixels();
    res.endDraw();
    return res;
  }

}
```

This time we don't add an image but render it from code. The makeTexture function generates an image and renders a glowing orb. the function loadPixels makes all the pixels of an image accessible throu an array named pixels we calculate the distance of every pixel from the middle of the image and make the color an inverse function of distance. when we are finished with changing the pixels we call updatePixels

the getBox function doesn't return the actial size of the image, because the image is larger than the actual orb because of it't glowing aura.

we also added a boolean variable named active and only draw the image if active is true.

add a new method named reset to the comet class so it looks like this

```
public class Comet {
  int x;
  int y;
  int speed;
  PImage img;
  float alph;

  public Comet( int x, int y, int speed, PImage img ) {
    this.x = x;
    this.y = y;
    this.speed = speed;
    this.img = img;
    this.alph = 0;
  }

  public void update() {
    alph-= 0.1;
    x -= speed;
    if ( x < - img.width/2 ) {
      reset();
    }
  }

  public void reset() {
      x = width + img.width/2;
      y = int( random( height - img.width ) + img.width/2);
  }

  public void draw() {
    pushMatrix();
    translate ( x, y );
    rotate( alph );
    image( img, -img.width/2, -img.height/2 );
    popMatrix();
  }

  public Box getBox() {
    return new Box( x-img.width/2, y-img.height/2, x+img.width/2 , y+img.height
  }
}
```

and now change the main class to look like this

```
Starfield starfield;
Comet comet;
Ship ship;
Torpedo torpedo;

PImage boom;

void setup() {
  size(640,480);
  starfield = new Starfield( 100 );

  PImage cimg = loadImage( "comet.png" );
  comet = new Comet( width + cimg.width/2, height/2, 4, cimg );

  PImage simg = loadImage( "ship.png" );
  ship = new Ship( 100, height/2, 3, simg );
  boom = loadImage( "boom.png" );
  torpedo = new Torpedo(20,10,240);
  torpedo.active = false;

  frameRate( 25 );
  smooth();
}

int boom_count = 0;
int comet_boom_count = 0;

void draw() {
  background(0);
  starfield.draw();
  torpedo.update();
  torpedo.draw();

  if ( boom_count == 0 ) {
    if ( keyPressed == true && key == CODED ) {
       if ( keyCode == UP ) {
         ship.up();
       } else if ( keyCode == DOWN ) {
         ship.down();
       }
    }
    ship.draw();
  } else {
    image( boom, ship.getBox().x1, ship.getBox().y1 );
    boom_count--;
  }

  if ( ship.getBox().isOverlap( comet.getBox())) {
    boom_count = 25;
  }

  if ( torpedo.active && comet.getBox().isOverlap( torpedo.getBox())) {
    comet_boom_count = 25;
    torpedo.active = false;
  }

  if ( comet_boom_count == 0 ) {
    comet.update();
    comet.draw();
  } else if ( comet_boom_count == 1 ) {
    comet.reset();
    comet_boom_count--;
  } else {
    image( boom, comet.getBox().x1, comet.getBox().y1 );
    comet_boom_count--;
  }
}

void keyPressed() {
  if ( key == ' ' && !torpedo.active ) {
    torpedo.x = ship.x + ship.img.width/2 - 20;
    torpedo.y = ship.y;
    torpedo.active = true;
  }
}
```

we use a callback function here to react to the keypressed event. if the key is the spacebar we set the start coordinates of our torpedo to the front of the ship and activate it. then it runs to the right. if it hits the comet our boom code is displayed at the location of the comet and we decrement a second counter. if the counter is nearly finished the comet gets reset to a location right of the border again.

make some noise

TBD

count the score

TBD