Introduction to Visual Computing

Assignment# 7

Data Visualization

April 10, 2018

Description

In this session you will add a few data visualization views to your game application.

Objectives

- Visualize the top view of the game
- Display the live score
- Visualize the score over time, with a bar chart

Specific Challenges

To arrange the visualization views as well as the main game view on the display area.

Preliminary steps

Complete week 5 assignment, if not already done, as this week's assigmenent builds on your previous work on the game.

Part I

Multiple Drawing Surfaces

Up to now, in this project, you have been drawing everything to the primary display window. However, all of the drawing features available in the display window can be applied to an off-screen drawing surface and then drawn back into the display window as an "image". You can imagine a program as a stack of layers similar to the ones used typically in image editing software. Likewise, surfaces in Processing can be moved around, drawn using blending effects and transparency, and drawn in different orders to change how the layers combine.

In the following, we use the concept of multiple layers, to create the visualization views, such that each view becomes an image that you place onto the primary display window.

Step 1 – PGraphics

Each drawing surface in Processing is an instance of the `PGraphics` class. The display window is the default surface to draw to, and this is why to draw a rectangle you only need to run `rect()` function. To draw the same shape into a new surface, you need to first create a `PGraphics` object, and then draw onto it, by typing the name of the surface object and dot operator before the name of the function. The given code below is a simple example:

```
PGraphics mySurface;
void settings() {
  size(400, 400, P2D);
}

void setup() {
  mySurface = createGraphics(200, 200, P2D);
}

void draw() {
  background(200, 0, 0);
  drawMySurface();
  image(mySurface, 10, 190);
}

void drawMySurface() {
  mySurface.beginDraw();
  mySurface.background(0);
  mySurface.ellipse(50, 50, 25, 25);
  mySurface.endDraw();
}
```

The `createGraphics()` function makes a surface with the size defined by the parameters. `beginDraw()` and `endDraw()` are needed each time you want to edit this surface. Finally, the `image()` function puts the surface on the display window, at the position defined by its parameters.

In practice, `PGraphics` layers are often declared as global, created inside `setup()` and modified in `draw()`.

Note

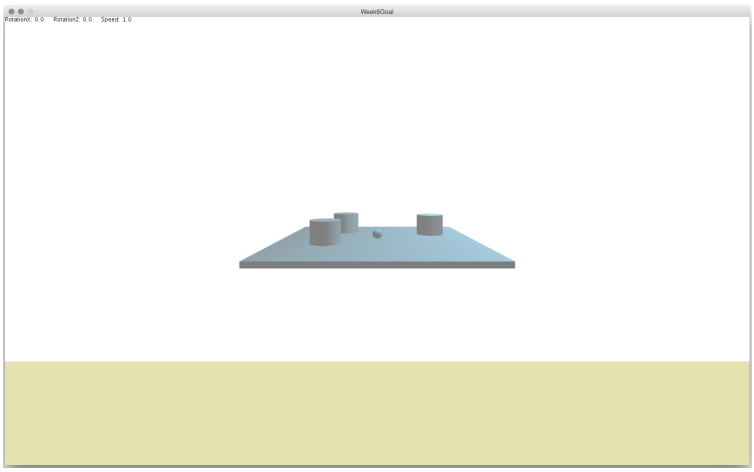
Be careful that to create a new graphics context, as suggested by Processing, you should use only the `createGraphics()` function and *not* `new PGraphics()`. This is also the reason why in the following steps we are not going to make each surface as a class that extends `PGraphics`, even though it would have been more modular.

Step 2 – A surface for data visualization background

For the main game part, you will defined a `P3D` PGraphics in which you will have your previsouly implemented game. For the data visualization part, however, you will make 3 new surfaces. The first one is simply a rectangle at the bottom of the display window. This would be the background for your data visualization area. In this step you should create the background bar as illustrated in the figure below. In the next parts, you will get more information about the other three surfaces, each being one visualization view.

Note

The rendering parameters of a surface such as `stroke` and `fill` do not reset every time `beginDraw()` is called (when a new frame is drawn). This means that if, for example, you set the `fill` field to 0 with `mySurface.fill(0)`, it remains 0 (black) even in the next frames, until the next time it is set to a different value, ex. `mySurface.fill(255)`.



Change the structure of your code, so that you draw your game on a new surface instead of the default surface. More concretely, you need to:

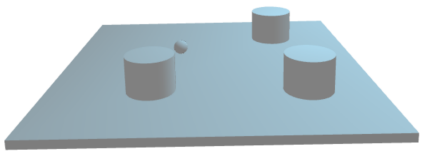
1. Declare a `PGrahics` (let's call it `gameSurface`).
2. Generate the surface with `createGraphics(width, height-300)` (`height-300` makes an empty space at the bottom of the display window)
3. Create a new function (called `drawGame()`), cut and past the content of your current draw function in it, and add `gameSurface.` before each "drawing" function.
4. In the `draw()` function, call `drawGame()` and `image(gameSurface, 0, 0)`
5. Create another PGraphics surface to serve as the background for your scoreboard and draw it similarly to the gameSurface.

Part II

Top View

In this part you should make a new surface (let's call it `topView`). As illustrated in the figure below (the square on the left bottom corner), this surface should show a 2D view of the game from top. The most straightforward way for doing this is to simulate the plate with a rectangle, the cylinders with large and the sphere with small ellipses (with the correct relative sizes). (You do not need to implement any 3D solution such as changing the camera position.)

Please note that, the objective of this week's assignment is to examine different data visualization styles, rather than solving the technical challenge of, for example, making multiple cameras. Likewise, in the next parts the tasks could be merely to extract some interesting values and display them, in such a way that can facilitate the game play and increase user awareness.



Note

For the top view you can use the P2D rendering mode. However, it is important to consider the renderer used with `createGraphics()` in relation to the main renderer specified in `size()`. For example, it's only possible to use P2D or P3D with `createGraphics()` when one of them is defined in `size()`.

Part III

Score Board

In this part you should make a simple score board on another surface and put it just next to the top view, as shown in the figure below.

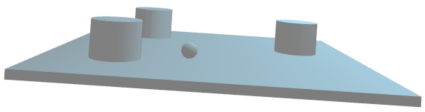
Step 1 – Collecting and losing points

Follow the rules below to calculate the scores:

- Every time the sphere hits a cylinder, the user gains points at the amount of velocity.
- Every time the sphere hits one of the plate edges, the user loses points at the amount of velocity.

Step 2 – Display the score

Create a new surface called `scoreboard`, and display the current score, current velocity magnitude of the sphere, as well as the points that the user achieved in the last (hitting) event. To draw text on the screen, you need to use the function `text()`. (`https://processing.org/reference/text_.html`)



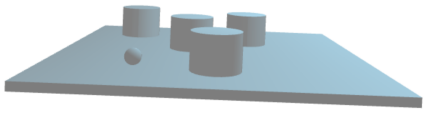
Part IV

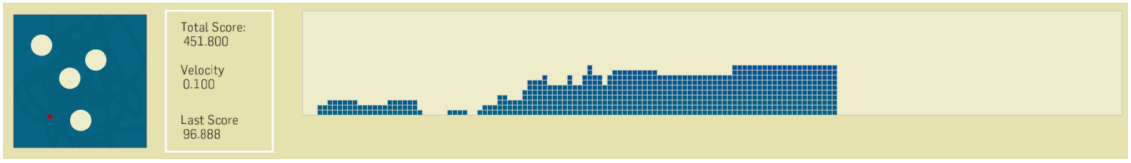
Score Chart

Step 1 – Bar Chart

Make a new surface and call it `barChart`. This surface should show, similar to the figure below, the total score as time goes on. The score is represented with a number of stacked-up tiny rectangles.

> **Note**
>
> Normally when visualizing data, you use functions that are made available by the tool in hand. Nevertheless, in this assignment you should make the chart yourself. The advantage of crafting the chart from scratch, beside the pedagogical reasons, is that you have the full control on how you would like to represent your data.



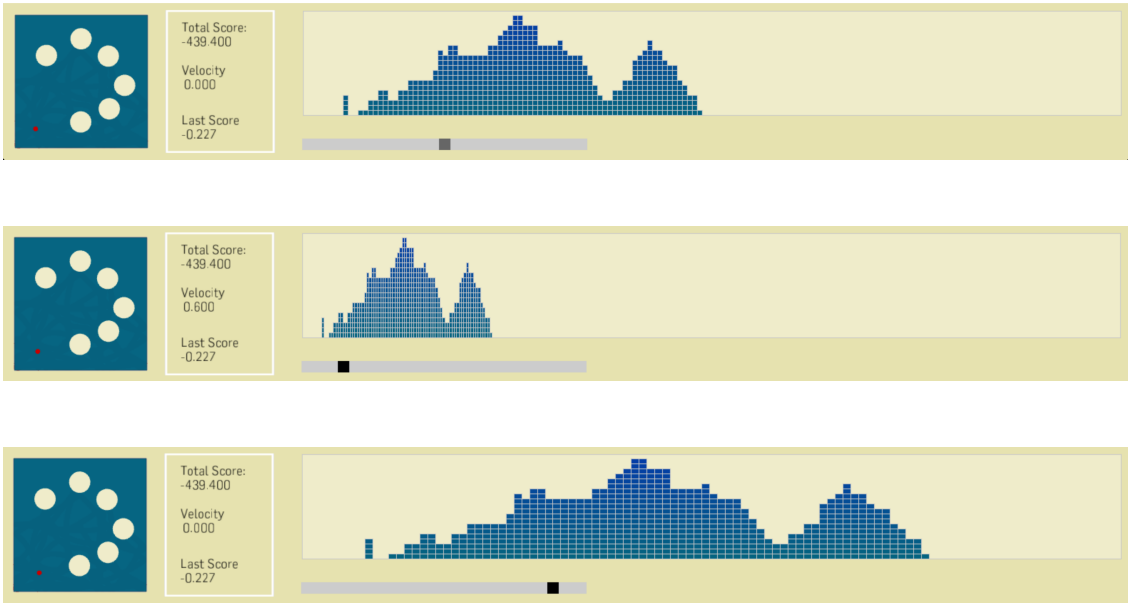


Step 2 – Horizontal Scroll bar

In this step you should make a scroll bar, using which the user can compress or expand the bar chart horizontally. This can be done by changing the width of the tiny rectangles in the bar chart, as depicted in the three figures below.

An example of how scroll bar could be implemented in Processing is provided below. You only need to copy the `HScrollbar` class in a new tab, instantiate it in your `setup()` function, call its `update()` and `display()` in your `draw()` function, and use its `getPos()` to get the current position of the scroll, which is normalized to get a value between 0 and 1. The second piece of code given below is an example of how `HScrollbar` should be used. The `HScrollbar` class is also available on Moodle for downloading.

As you may have noticed `HScrollbar` is not compatible with the concept of multiple layers, as its `display()` functions draws only on the default surface. To simplify this assignment, you can use the `HScrollbar` as it is. You could, however, try modifying `HScrollbar` so that it can become part of a layer (*OPTIONAL*).




```
HScrollbar hs;

void settings() {
  size(400, 200, P2D);
}

void setup() {
  hs = new HScrollbar(50, 90, 300, 20);
}

void draw() {
  background(255);
  hs.update();
  hs.display();
  println(hs.getPos());
}
```

```
class HScrollbar {
  float barWidth;  //Bar's width in pixels
```

```
float barHeight; //Bar's height in pixels
float xPosition;  //Bar's x position in pixels
float yPosition;  //Bar's y position in pixels

float sliderPosition, newSliderPosition;    //Position of slider
float sliderPositionMin, sliderPositionMax; //Max and min values of slider

boolean mouseOver;  //Is the mouse over the slider?
boolean locked;     //Is the mouse clicking and dragging the slider now?

/**
 * @brief Creates a new horizontal scrollbar
 *
 * @param x The x position of the top left corner of the bar in pixels
 * @param y The y position of the top left corner of the bar in pixels
 * @param w The width of the bar in pixels
 * @param h The height of the bar in pixels
 */
HScrollbar (float x, float y, float w, float h) {
  barWidth = w;
  barHeight = h;
  xPosition = x;
  yPosition = y;

  sliderPosition = xPosition + barWidth/2 - barHeight/2;
  newSliderPosition = sliderPosition;

  sliderPositionMin = xPosition;
  sliderPositionMax = xPosition + barWidth - barHeight;
}

/**
 * @brief Updates the state of the scrollbar according to the mouse movement
 */
void update() {
  if (isMouseOver()) {
    mouseOver = true;
  }
  else {
    mouseOver = false;
  }
  if (mousePressed && mouseOver) {
    locked = true;
  }
  if (!mousePressed) {
    locked = false;
  }
  if (locked) {
    newSliderPosition = constrain(mouseX - barHeight/2, sliderPositionMin, sliderPositionMax);
  }
  if (abs(newSliderPosition - sliderPosition) > 1) {
    sliderPosition = sliderPosition + (newSliderPosition - sliderPosition);
  }
}

/**
 * @brief Clamps the value into the interval
 *
 * @param val The value to be clamped
```

```
   * @param minVal Smallest value possible
   * @param maxVal Largest value possible
   *
   * @return val clamped into the interval [minVal, maxVal]
   */
  float constrain(float val, float minVal, float maxVal) {
    return min(max(val, minVal), maxVal);
  }

  /**
   * @brief Gets whether the mouse is hovering the scrollbar
   *
   * @return Whether the mouse is hovering the scrollbar
   */
  boolean isMouseOver() {
    if (mouseX > xPosition && mouseX < xPosition+barWidth &&
      mouseY > yPosition && mouseY < yPosition+barHeight) {
      return true;
    }
    else {
      return false;
    }
  }

  /**
   * @brief Draws the scrollbar in its current state
   */
  void display() {
    noStroke();
    fill(204);
    rect(xPosition, yPosition, barWidth, barHeight);
    if (mouseOver || locked) {
      fill(0, 0, 0);
    }
    else {
      fill(102, 102, 102);
    }
    rect(sliderPosition, yPosition, barHeight, barHeight);
  }

  /**
   * @brief Gets the slider position
   *
   * @return The slider position in the interval [0,1]
   * corresponding to [leftmost position, rightmost position]
   */
  float getPos() {
    return (sliderPosition - xPosition)/(barWidth - barHeight);
  }
}
```