# CSC 205 - SPRING 2016
# 2D GRAPHICS AND IMAGE PROCESSING
# ASSIGNMENT 1
# UNIVERSITY OF VICTORIA

**Due**: Tuesday, January 26th, 2016 by noon.

## 1 Assignment Overview

The objective of this assignment is to implement rasterization algorithms for a basic vector graphics language. In practice, vector graphics tend to be represented with formats such as PDF or SVG, but these formats are too complicated to be practical for a two week assignment. Instead, this assignment will use an invented vector graphics language called BVG.

Your program must read a text input file describing an image in BVG format (detailed in Section 4) and output a rasterized image in any standard image format with lossless compression (including GIF, PNG, BMP or Targa). A suite of starter code has been provided in Java, C++ and Python. For all three languages, the provided code reads the BVG commands and performs basic syntax checking, then calls functions to render each command. The implementation of the rendering functions has been left blank. The starter code also contains functionality to produce a rasterized image in PNG format. Therefore, it should not be necessary for you to write any code to handle BVG parsing or converting the rasterized array of pixels to an image. The basic functionality of the starter code will be discussed during lectures and labs.

However, you are not required to use the starter code as the basis for your solution. You are permitted to implement your solution in any language that can be run (with publicly available open source tools) on lab machines in ECS 354, ECS 242 or ECS 266 (depending on the operating system required). You are also permitted to use multiple languages, or to use some subset of the starter code rather than all of it.

## 2 Externally-Sourced Code

You are permitted to use appropriately licensed code from an external source, if full attribution is given (including the name of the original author, the source from which the code was obtained and an indication of the terms of the license). Note that using copyrighted material without an appropriate license (such as the GPL, LGPL, BSD License or MIT License) is not permitted. Short fragments of code found on public websites (such as StackOverflow) may be used without an explicit license (but with the usual attribution). If you embed externally sourced code in your own code files, add a citation in your code file and include a README file detailing the sources of all code used. If you use entire, unmodified source files from an external source, a README file is not necessary as long as the file in question contains complete authorship and licensing information. If you submit the work of others without proper attribution, it will be considered plagiarism. Additionally, for assignments in CSC 205, the following two caveats apply to externally sourced code.

- You will not receive any marks for the work of others. That is, if you use externally sourced code to implement the core objectives of the assignment, you will only be marked on the parts of the code that you personally wrote. Therefore, you should only use externally sourced code in a supplementary capacity (such as including a third-party hash table implementation which your code uses to store data, or a third party Gaussian elimination package to solve matrix equations). If you have any doubts about what is considered supplementary, contact your instructor.
- You may not use externally sourced code from another CSC 205 student, or share your code with another CSC 205 student (whether they intend to use it or not), without explicit permission from the instructor.

## 3 Starter Code

Starter code has been provided for C++, Java and Python. For all three languages, the starter code contains a BVG parser (which reads the input file), a wrapper around a PNG writer and a basic command line interface. The C++ and Python versions rely on open source PNG writer modules which are distributed with the starter code (full licensing information is available in the header comments of the relevant source files).

### 3.1 C++

Normally, C++ projects with multiple files are compiled with a build script or Makefile. You may choose to create a Makefile for your project, but the command

```
$ g++ -Wall -o draw_bvg *.cpp
```

can be used to compile all of the C++ source files into an executable called `draw_bvg`. The flag `-Wall` enables all normal compiler warnings, and is recommended to assist with debugging. You may also want to experiment with the `-O3` flag, which enables code optimizations (although it is unlikely that code optimizations will have an appreciable effect on speed in this assignment).

After the executable has been produced, it can be run with the command

```
$ ./draw_bvg input.txt output.png
```

where `input.txt` and `output.png` are the names of the input BVG file and output PNG file, respectively.

It should be possible to implement the entire BVG renderer by completing the implementation of the `BVGRenderer` class in `draw_bvg.cpp`.

### 3.2 Java

The Java starter code can be compiled with the command

```
$ javac DrawBVG.java
```

and run with the command

```
$ java DrawBVG input.txt output.png
```

where `input.txt` and `output.png` are the names of the input BVG file and output PNG file, respectively.

It should be possible to implement the entire BVG renderer by completing the implementation of the `BVGRenderer` class in `BVGRenderer.java`.

### 3.3 Python

The Python starter code can be run with the command

```
$ python draw_bvg.py input.txt output.png
```

where `input.txt` and `output.png` are the names of the input BVG file and output PNG file, respectively.

It should be possible to implement the entire BVG renderer by completing the implementation of the `BVGRenderer` class in `draw_bvg.py`.

## 4 BVG Format

A BVG file consists of a list of BVG commands, with one command per line. Each command consists of a command name (which is the first word on the line) followed by a set of attributes. Command names and attribute names are case-sensitive. Attributes can appear in any order within a given command. There must be at least one space between the command name and the first attribute, and between different attributes. Otherwise, whitespace (including blank lines) is ignored. There is no facility for comments inside BVG files.

The first command in any BVG file must be a `Canvas` command, which defines the size of the drawing area. You may assume that all input files used for marking will be valid (that is, they will contain only valid, well-formatted commands and the parameters to each command will be semantically valid). The commands should be processed in order, so commands appearing later in the file should be drawn after commands appearing earlier. A list of commands appears below.

### 4.1 Canvas

*Example*: `Canvas dimensions=(100,100) background_colour=(255,0,0)`
Constructs a drawing canvas.
Attributes:

- `dimensions` - A pair `(width, height)` which defines the maximum extent of the drawing canvas. Note that the directive `dimensions=(width, height)` should set up a canvas with pixel coordinates from `(0, 0)` to `(width-1, height-1)`
- `background_colour` - An `(r, g, b)` triple containing a background colour. The `Canvas` command should fill every pixel in the newly created drawing canvas with this colour.
- (Optional) `scale_factor` - An integer value $s \geq 1$. If this directive is not present, $s$ defaults to 1. If $s$ is greater than 1, the dimensions of the drawing canvas and all contents should be scaled by a factor of $s$. The implementation of this functionality is optional.

### 4.2 Line

*Example*: `Line from=(5, 90) to=(45,10) colour=(0,255,255)`
Draws a line segment.
Attributes:

- `from`, `to` - `(x,y)` pairs giving the coordinates of the two endpoints of the line segment.
- `colour` - An `(r, g, b)` triple containing the line colour.
- (Optional) `thickness` - An integer value $t \geq 1$ giving the thickness of the line. If this directive is not present, $t$ defaults to 1. The implementation of this functionality is optional.

### 4.3 Circle

*Example*: `Circle center=(10, 10) radius=5 line_colour=(255,128,128) line_thickness=2`
Draws the boundary of a circle.
Attributes:

- `center` - `(x,y)` pairs giving the coordinates of the center of the circle.
- `radius` - An integer radius for the circle.
- `line_colour` - An `(r, g, b)` triple containing the colour of the boundary line.
- (Optional) `line_thickness` - An integer value $t \geq 1$ giving the thickness of the boundary line. If this directive is not present, $t$ defaults to 1. The implementation of this functionality is optional.

### 4.4 FilledCircle

*Example*: `FilledCircle center=(50,50) radius=10 line_colour=(255,255,0) fill_colour=(64,0,64)`
Draws a filled circle.
Attributes:

- `center` - `(x,y)` pairs giving the coordinates of the center of the circle.
- `radius` - An integer radius for the circle.
- `line_colour` - An `(r, g, b)` triple containing the colour of the boundary line.
- (Optional) `line_thickness` - An integer value $t \geq 1$ giving the thickness of the boundary line. If this directive is not present, $t$ defaults to 1. The implementation of this functionality is optional.
- `fill_colour` - An `(r, g, b)` triple containing the fill colour.

### 4.5 Triangle

*Example*: `Triangle point1=(10,10) point2=(25,20) point3 = (90,10) line_colour=(255,255,255) fill_colour=(255,0,0)`
Draws a filled triangle.
Attributes:

- `point1`, `point2`, `point3` - `(x,y)` pairs giving the coordinates of the three vertices $P_1, P_2, P_3$ of the triangle.
- `line_colour` - An `(r, g, b)` triple containing the colour of the boundary line.

- (Optional) `line_thickness` - An integer value $t \geq 1$ giving the thickness of the boundary line. If this directive is not present, $t$ defaults to 1. The implementation of this functionality is optional.
- `fill_colour` - An `(r, g, b)` triple containing the fill colour.

### 4.6 GradientTriangle

*Example*: `GradientTriangle point1=(25,10) colour1=(0,0,0) point2=(60,50) colour2=(0,255,255) point3=(99,1) colour3=(255,0,255) line_colour=(255,128,128)`
Draws a filled triangle where the colour of a pixel $Q = (x, y)$ inside the triangle is determined by the equation

$$C_q = \lambda_1 C_1 + \lambda_2 C_2 + \lambda_3 C_3$$

where $C_1, C_2, C_3$ are colour values (defined below) and $(\lambda_1, \lambda_2, \lambda_3)$ are the barycentric coordinates for the point $Q$ with respect to the vertices $P_1, P_2, P_3$ of the triangle.
Attributes:

- `point1`, `point2`, `point3` - `(x,y)` pairs giving the coordinates of the three vertices $P_1, P_2, P_3$ of the triangle.
- `line_colour` - An `(r, g, b)` triple containing the colour of the boundary line.
- (Optional) `line_thickness` - An integer value $t \geq 1$ giving the thickness of the boundary line. If this directive is not present, $t$ defaults to 1. The implementation of this functionality is optional.
- `colour1`, `colour2`, `colour3` - `(r, g, b)` triples containing the colour values $C_1, C_2, C_3$ associated with the vertices $P_1, P_2, P_3$ respectively.

## 5 Evaluation

Submit all of your code electronically through the Assignments tab on conneX. Your implementation will be marked out of 70 during an interactive demo with an instructor. You may be expected to explain some aspects of your code to the evaluator. Demos must be scheduled in advance (through an electronic system available on conneX). If you do not schedule a demo time, or if you do not attend your scheduled demo, you will receive a mark of zero.

A total of 80 marks are possible on the assignment, but the final assignment mark will be taken out of 70 (so any marks you receive beyond 70 will be treated as bonus). The marks are distributed among the components of the assignment as follows.

| Marks | Feature |
|---|---|
| 15 | The `Line` command correctly renders lines with thickness 1. |
| 5 | The `Line` command does not use floating point operations for any part of its computation. |
| 10 | The `Circle` command correctly renders circles with line thickness 1. |
| 5 | The `Circle` command does not use floating point operations for any part of its computation. |
| 10 | The `FilledCircle` command correctly renders filled circles with line thickness 1. |
| 10 | The `Triangle` command correctly renders solid-filled triangles with line thickness 1. |
| 10 | The `GradientTriangle` command correctly renders gradient-filled triangles with line thickness 1. |
| 5 | The `thickness` and `line_thickness` attributes for all shapes function correctly. |
| 5 | The `scale_factor` attribute of the `Canvas` command functions correctly with scale factors greater than 1. |
| 5 | An antialiasing technique is applied to all rendering operations. |

Additionally, you may receive extra marks (with the total overall mark capped at 80) for other features of your own design, if you have received permission from the instructor before the due date. If you propose your own bonus features, you should be prepared to provide input files which demonstrate those features when you submit your code.

Ensure that all code files needed to compile and run your code in an ECS lab are submitted. Only the files that you submit through conneX will be marked. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.