Programming 3 - Advanced

Interoperability - Project 2

Original assignment description

Your task is to create an interactive procedural image generator using a native library written in C++. Let's start with an...

Example:

The user can enter a *generating command*:

- Generate `n width height`

The program creates `n` images whose size is `width`x`height` pixels and calls an external library to fill them with random patters.

The result of the command can then be piped into *processing commands* such as

- Blur `width height`
- Output `filename_prefix`

that modify it or output the result into a file:

```
Generate 10 1024 1024 | Blur 4 4 | Output Image
```

The following chain of commands generates 10 1024x1024 images, blurs them with radius 4 vertically and horizontally and saves them as "Image1.jpeg", "Image2.jpeg", ... , "Image10.jpeg"

Command chains

The user enters a *generating command* and its arguments, followed by 0 or more *processing commands* and their respective arguments. Commands are separated by the `|` (pipe) character.

```
generating_command arg1 arg2 | processing_command1 | processing_command2 arg1 arg2
```

Error checking

Detect syntax errors. It's sufficient to determine one has occurred. In such case don't start the execution of the entered command chain.

Progress reporting

Every library function accepts a progress-reporting callback that is called every 1% of the work:

C++

```
bool TryReportCallback(float)
```

- The argument passed to the callback is the fraction of the work of the current function.
- If the callback returns `false`, the library function returns early.

Show execution progress of each image in the chain separately. Separate the progress of each command in the chain visually (segments of roughly equal size is sufficient).

Example output:

```
[######|######|###---|------|------] 50%
[######|######|##----|------|------] 47%
[######|######|###---|------|------] 50%
[######|######|######|------|------] 60%
```

Abort

When a command chain is being processed, the user can press `x`. All chains should then safely stop. Remember to free resources!

Tip: The progress-reporting callback returns a boolean. In order to stop, return false.

Commands

Implement at least 3 *generating commands* using the `GenerateImage_Custom` library function and 3 *processing commands* using the `ProcessPixels_Custom` library function. Get creative!

The following should be available in your project:

Special commands:

- Help
  - List available commands together with argument descriptions. Does nothing if called in a command chain. List the functions you came up with at the start.

Generating commands:

- Input `filename`
  - Load an image from the disk. (to be implemented)
- Generate `n width height`
  - Create `n` images whose size is `width`x`height` pixels (to be implemented) and fill them with random patterns (implemented in the library).
- *Your 3 commands* (to be implemented using `GenerateImage_Custom`)

Processing commands:

- Output `filename_prefix`
  - Save images to the disk (to be implemented).
- Blur `w h`
  - Apply a `w`x`h` blur (implemented in the library).
- RandomCircles `n r`
  - Add `n` circles of radius `r` placed randomly on the images (to be implemented using `DrawCircles`).
- Room `x1 y1 x2 y2`
  - Draw a filled rectangle with the given coordinates. The coordinates range from 0 to 1 (to be implemented using `ProcessPixels_Custom`).
- ColorCorrection `red green blue`
  - Apply color correction (implemented in the library).
- GammaCorrection `gamma`
  - Apply Gamma correction (implemented in the library).
- *Your 3 commands* (to be implemented using `ProcessPixels_Custom`)