C++ Capstone Project

Random Dot Stereogram Generator (RDS)

README

Building/running the project

The compilation instructions are basically the same as for theConcurrent Traffic Simulation project of the current c++ NanoDegree program.
It uses the same cmake and make files, and uses OpenCV.
Additionally, it uses cvui, to build a simple graphical user interface.
Dependencies for Running Locally
* cmake >= 2.8
   * All OSes: [click here for installation instructions](https://cmake.org/install/)
* make >= 4.1 (Linux, Mac), 3.81 (Windows)
   * Linux: make is installed by default on most Linux distros
   * Mac: [install Xcode command line tools to get make](https://developer.apple.com/xcode/features/)
   * Windows: [Click here for installation instructions](http://gnuwin32.sourceforge.net/packages/make.htm)
* OpenCV >= 4.1
* CVUI installation instructions can be fould here: https://dovyski.github.io/cvui/usage/ (You will need to copy cvui.h to your src folder)
   * The OpenCV 4.1.0 source code can be found [here](https://github.com/opencv/opencv/tree/4.1https://github.com/opencv/opencv/tree/4.1.0.0)
* gcc/g++ >= 5.4
   * Linux: gcc / g++ is installed by default on most Linux distros
   * Mac: same deal as make - [install Xcode command line tools](https://developer.apple.com/xcode/features/)
   * Windows: recommend using [MinGW](http://www.mingw.org/)

## Basic Build Instructions

1. Clone this repo.
2. Make a build directory in the top level directory: `mkdir build`
3. In the build directory, make a directory called ‘outputimages’ - this will be the default output directory where the images are saved
4. Copy the ‘RDS.ini’ and ‘SamsungGalaxyS7.csv’ files to the ‘build’ directory.
3. Compile with cmake: enter the build directory (‘cd build’) and there type ‘cmake..’
3. Compile: ‘make’
4. Run it: `./RDS`

Aim and scope of the project

The program generates a single or a series of stereo images, whose size and parameters can be adjusted in the input files to work properly on any mobile device. The main point is to generate and draw Random Dot Stereogram images.

- A short theoretical background:

Random Dot Stereograms were invented by Béla Julesz, who proved that human stereopsis is able to work based merely on disparity (seeing in 3D based on the two sightly different images projected to the two eyes), without any cues gained from prior shape recognition. (More information e.g.: https://en.wikipedia.org/wiki/Random_dot_stereogram)

- A short history:

Since I am an assistant professor at a university, cognitive psychology department, my aim was to create a cheap (almost free) test for screening a larger sample of people with lack or deficit of stereopsis (usually e.g. people with strabismus have some stereo vision deficit) in order to further investigate their perception and perceptual learning skills in later research. So I wrote the original version of this program in Borland Delphi in 2017, without any threads, any custom classes, any knowledge of memory management, no pointers - as I learned programming only on my own as a hobby - so the aim then was only to get the images ready somehow, so that my students can use them on their own mobile devices for their mini research projects.

The main challange then was to devise an algorithm to generate the random dot stereo images of different shapes, such as triangle, or circle (Julesz used square only, which is a bit more simple to implement. I did not have any sample code even for that, so I had to find it all out for myself - I did not even know GitHub.). In the present project, I have further refined the algorithm and I also applied the programming features learnt during the course (such as threads, classes, etc) to make the program much more effective than the original one used to be.

Each student copied and used the images (generated by my original, quite lumbersome and slow Delphi program on Windows PC) on his/her own mobile device - that is why the image size parameters had to be adjusted to suit any mobile device screens (that is why sizes are given in mm or 0.1 mm in the input files - sizes had to be roughly the same on all devices).

The students used Google Cardboard (instructions for making your own Google cardboard: https://arvr.google.com/intl/hu_hu/cardboard/get-cardboard/) and Quickpick image gallery app (that worked on iPhone and on Android as well in timed slideshow mode) to present the images for their subjects. The students had to write down their subjects' responses for each image and evaluate the correctness of the subjects' responses.

The task: The subjects had to tell for each image which shape was perceived as closest to them, in the middle and farthest, which merely depended on the shape's disparity.

Description of the program:

The input files for generating the images are:

- the *.ini file (the default ini file is RDS.ini) for generating a single stereo image. It can be edited in any text editor.

- the *.csv file for generating image series (for screening subjects' stereopsis) - the default csv file is 'SamsungGalaxyS7.csv'. This can be also edited in any spreadsheet editor, like Excel or OpenOffice, opened as a comma delimited csv file. (Thus it is easy to modify the parameters for the given device without modifying image parameters, by copying the new device parameters in the corresponding columns pulling them down over rows.)

Device parameters:

DisplaySize;DisplayWidth;DisplayHeight; : The display size of the device should be provided in inch, and the display width (the longer side) and the display height should be provided in pixels.

Image parameters:

Each image contains 3 shape pairs (circle, triangle, square). The disparity of each can be adjusted with 0.1 mm precision (which serves as the 3D cue when viewed on the mobile device placed in a Google cardboard). These are: SquareDisparMM;TriangleDisparMM;CircleDisparMM.

There are 4 image types (imageType), that involve 4 levels of difficulty:

- simple: homogeneous background, homogeneous shapes, sharp edges (0)
- RDSFilledShape: random dot background, filled shape (1)
- RDSDiffColors: random dot background and shape, in different colors (2)
- RDSclassic: random dot background and shape, same colors: the only cue is the disparity (3)

Image types, size of the shapes (ShapeWidthMM;ShapeHeightMM: 0.1 mm precision), their overall horizontal distance (TotalHorizontalDisparMM), vertical disparity (TotalVerticalMM - vertical shift between left and right groups of shapes - by default it is 0, but it might be adjusted for strabismic patients), the vertical order of the shapes (orderSquare;orderTriangle;orderCircle, with valuses top = -1, middle = 0, bottom=1) can also be set in the input files.

The colors of the background and shapes (ShapeColorIndex;BackgroundColorIndex;) can be adjusted by color indexes taken from the Delphi GUI element color indexes, they can be between 9 and 15, and 0 (eg. 9 is red, 11 is yellow, etc) - this is in order to be able to use the old input files.

The JuleszPixelSize (JuleszPixSizeMM) is the size of the random dots, and this should be given in 0.1 mm in the input files.

From these device and image data, the program calculates all sizes in pixels for the given device, so that the sizes provided in mm in the input files remain roughly constant on all devices, and generates the images, and also generates a solution key in a textfile, which contains the correct answers for each image, in order to help the experimenter to evaluate the correctness of the subject’s responses.

How to use the program

The GUI of the program consists of 4 buttons and 3 checkboxes. By means of the checkboxes, the user can select whether he/she wishes to use the default input files (either *.ini and *.csv), and the default output directory for saving the image series. It can also be selected whether the user wants to generate a solution key textfile for the image series.

If the user does not select the default input files or the default output directory, he/she will be asked to type in the wished input file name / output directory name in the terminal. The program checks whether the typed path exists. If not, it ask the user again to type it in until an existing path is typed. If only the file extension is missing in case of the input file, the program also accepts it, since it completes it inside the code.

To read data from the *ini file and calculate the data for generating the image, hit the “Create image matrix” button.

To display the raw (base) image, which contains the shapes in a black-white image in the appropriate size and positions, hit the “Show base image” button.This image is the input for generating the final image of the required type and color.

To display the final, stereo image, hit the ‘Show RDS’ image button.

To save this single image, hit the ‘Save RDS image’ button. The image will be saved as RDS.bmp in the directory of the program.

To generate the image series of 40 stereo images based on the input *.csv file (either the default or the one provided by the user depending on the state of the corresponding checkbox - and the same applies for the output directory, as described above), hit the ‘Save RDS series’ button.

When ready, you can copy the images on your mobile device in a directory, and with an appropriate gallery application (such as QuickPic), you can launch a timed slide show (we set 10 s for each image), put it in the cardboard, and try to tell which shape is closest and farthest to you in each image.

In the default csv file, there are 10 of each image type, with a gradually increasing difficulty. If you have a perfect stereposis, you will see the shapes in depth even in the Type 3 images (classical Random Dot Stereograms) where no cue is available other than disparity.

(Personal note: I do not see the shapes on Type 3 images, and for me it is very hard to see the depth in Type 2: I have stereo deficit due to slight strabismus, but the very most of our subjects with normal vision did. On the other hand, seriously strabismic people did not even see type 0 images in depth).

<u>**File and class structure**</u>

Classes:

BaseImage (.cpp and .h) class: contains all parameters that are related to the image as a whole: width and height of the image, pixelsize calculations of the given display, calculations of the random dot sizes (JuleszPixelSize), horizontal and vertical midlines of the images, image proportions of the given display.

Shape (.cpp and .h) class: inherits from BaseImage, and also contains all the data and calculations that are common to all the three shapes. Such are the overall horizontal distance of the left and right groups of shapes, their vertical disparity, their size (same for all shapes), vertical distance between shapes, etc. This class also contains a virtual Draw fuction, which is overridden by its three child classes, Triangle, Square and Circle.

Triangle, Square and Circle classes (.cpp and .h): they inherit from Shape. Besides the common properties inherited from Shape and BaseImage, they contain the parameters and calculations that are unique to the given shape. Such are the vertical order of the shape, the disparity of the shape, or in case of circle, the radius or the centre points, in case of the triangle, the three rook points, and the related functions to calculate these values. These three classes implement their own overridden Draw function. Their constructor gets all the parameters that are necessary for the calculations. Their own private variable is initialized by means of initializer list, and their inherited parameters are initialized inside the constructor. Their constructor is called in the InitClasses function in Main.cpp, embedded into unique pointers. Their calculation functions are called also from InitClasses, passed into threads so that the parameters of the three shapes can be calculated parallel to each other. Their overridden Draw functions are called also from there, after joining the threads of the calculations. They all draw to the BaseImg, which is a shared pointer to a cv::Mat image and serves as the input for drawing the final RDS image.

Inheritance here is public in all cases, since the RDS class will need access to the parameters defined in base classes. The instances of square, circle and triangle are moved to the RDS class constructor wrapped in unique pointers, so that the RDS class instance will access the public data of the shapes.

RDS class: this class does not inherit from the above classes, but this one contains the functions and parameters that are necessary to draw the final stereo image - Random Dot Stereogram. The instances of the three shapes are moved to its constructor, along with the color codes and the image type read from the input files in the parse functions, and checked for valitity by the lambdas in InitClasses function in Main.cpp. After calling the constructor, InitClasses calls the Draw function of the RDS instance, which is also passed to a thread, along with its argument, that is the shared ptr to the BaseImage, on which the three shapes have already been drawn. The reason for starting this RDS::Draw function in a separate thread is that it involves reading a high resolution image (the BaseImage) pixel by pixel, and also modifying many pixels there, pixels by pixels, thus involving a lot of work.

The program is controlled by Main.cpp, in which the CVUI GUI is implemented (Main.cpp, Main function, from line 461 on).

If the subject hits the Create image matrix button, the data read from the ini file are moved to the init classes function (I used move semantics because it is a lot of data, I did not want to make a copy of them), along with the instance of Images created here, so that the shared pointers to the images will be accessible from the main function. A nullptr is also passed to the Filename attribute of InitClasses, since here, when creating the single image, we do not want to save the output image automatically, no file name is necessary.

However, a non-empty ptr to the filename is passed when the image series is generated by the ParseCSV function - there filenames are automatically generated and their pointers are passed to Initclasses. For this filename string parameter, I considered it unnecessary to use the heap, because it is not very data-consuming, but it might be more data to copy to pass the string itself, than its address, that's why I used a raw pointer here.

So both ParseIniFile and ParseCSV file move data to InitClasses, to initialize the instances of classes to calculate all parameters for drawing the images. ParseIniFile moves the data in its return value, ParseCSV file moves it line by line read from the csv, in the same vector of pairs format as an r value, and it also copies its return value to a local variable, in order to be able to save the solution file at the end.

InitClasses then checks the validity of all the data using async tasks and futures to return possible exceptions in lambda functions.

If data are checked, it creates the instances of the shapes (square, triangle and circle) on the heap, using unique pointers for them (because they contain larger amunt of data, that are needed troughout drawing the image, but not in Main.cpp anymore, so they can be moved to the RDS class constructor). After creating the instances of the three shapes, their calculation functions are called, which are passed to three threads, so the calculations for the square, triangle and circle can be done paralel to each other. When the calculations are done, join is called on each, and their overridden Draw functions (which is a virtual function in Shape class) is called one after the other, so that the shapes are drawn on the BaseImage.

The BaseImage is a cv::Mat type high resolution image, created as a shared pointer. The heap is used for it because it contains a large amount of data (high resolution image), and it has to be accessed from several functions of many classes. It is created in InitClasses, and it is passed to the Draw function of the triangle, circle and square instances, so all the three shape pairs will be drawn on the base image. (This image can be displayed by hitting the Show Base Image button (after creating the image matrix. If it is not created before hitting the Show Base Image button, the program warns the user on the terminal and asks him/her to hit the Create Image matrix button first)).

Finally, the unique pointers to the instances of the three shapes are moved to the constructor of the RDS class, and the image type and color indexes are also passed there. This constructor is passed into a thread, because it involves tedious work (as mentioned already). The shared pointers to the baseImage and the RDS image cv matrixes are copied into a struct defined at the beginning of Main.cpp, because the Main function will need access to these images. The BaseImage is passed to the GenerateSolutionKey in the RDS class, to calculate and return the right solution for the given image copied to a local string, SolutionKey. The reset function is then called on the shared pointers, becaue they will have to point to the next image in the series.

The InitClasses function returns the SolutionKey for the given image, so that the ParseCSVFile can add it to a vector and save the solution for the image series to a textfile.

The rds instance converts the color indexes to b, r and g values (TColorToBGR), sets the stimulus type, and creates and initializes its shared ptr to a cv::Mat by the BaseImage passed to its constructor. Based on the image type, its Draw function, called from InitClasses in Main.cpp in a thread, calls the drawing functions corresponding to the image type, and draws the final image. When this thread is joined in Main, the shared pointer is copied to the struct Images in Main, and the original shared pointer is reset. (Without copying it to the Images struct, the GUI elements in the Main function could not access it).

Finally, the Main function in Main.cpp has access to the Images struct, containing the shared pointers to the base and RDS images drawn by the program. Now the user can display or save them by hitting the corresponding buttons in the GUI, or the image series can be automatically saved to the default (or user-given) output directory by hitting the Save RDS series button.

Addressed rubrics

Loops, Functions, I/O

1. *The project demonstrates an understanding of C++ functions and control structures*: the project is organized into functions and uses a variety of control structures, numerous for loops (both with iterators - e.g. in RDS::DrawShapePair or DrawRDSBackround - and with vector elements - e.g. in Main.cpp, ParseCSVFile, line 444, when writing the vector to the textfile), while loops - e.g. when reading input files, listening to button events in the GUI in Main, if-then-else statements, etc.)

2. *The project reads data from a file and process the data, or the program writes data to a file.*

The ParseIniFile and ParseCSVFile functions in Main reads in input data from these files, and at the end of ParseIniFile.csv, the solutionKey is written in a textfile. The cv::Mat images are also saved to *.bmp files.

3. *The project accepts user input and processes the input*.: The program has a GUI, the user controls it by hitting buttons and checking checkboxes. If the user wishes to use other input files than the default, or other output directory, the program asks the user to type the wished path to the terminal, and the program uses that input. (ParseIniFile and ParseCSVFile funtions in Main.cpp. The PathExist function checks the validity of the user-typed paths.)

Object Oriented Programming

1. *The project uses Object Oriented Programming techniques*. :The project code is organized into classes with class attributes to hold the data, and class methods to perform tasks - as described above.

2. *Classes use appropriate access specifiers for class members:* All class data members are explicitly specified as public, protected, or private.

3. *Class constructors utilize member initialization lists*. : NOT all, but the three shapes' own private variables are initialized from member initialization lists (so the technique is utilized in the project), while inherited variables are initialized within the constructor.

4. *Derived class functions override virtual base class functions*.: The Draw functions of Square, Triangle and Circle override the virtual Draw function of their base class's (Shape.h, line 54) Draw function.

Memory Management

1. *The project makes use of references in function declarations*. Functions called from the Main function take references as arguments, e.g. ParseCSVFile in line 290 in Main.cpp, ParseIniFile in line 221 or PathExists in line 215 in Main.cpp.

2. *The project uses move semantics to move data, instead of copying it, where possible*. The instances of the 3 shapes are moved to the RDS class wrapped in unique pointers. Also, the data read from the input files (ParseIniFile, ParseCSVFile) are moved to InitClasses as r values in Main.cpp.

3. *The project uses smart pointers instead of raw pointers*. The high resolution images (BaseImag, RDSImage) are wrapped in shared pointers, while the three shapes are used with unique pointers. I have one raw pointer, it is a pointer to a filename string that is passed as an argument within Main.cpp (ParseCSVFile line 446 passes the address of the filename string to line 44, InitClasses). As mentioned above, I used a raw pointer here because it is not a large data structure, which would make it necessary to use the heap, but the address to the string is usually less data to copy than the string itself.

Concurrency

1. *The project uses multithreading*. The calculations of parameters of the three shapes are launched as threads to make it faster, and also the Draw function of the RDS class is called in a thread in InitClasses in Main.cpp.

2. *A promise and future is used in the project*. The exception handling for checking the various data read from the input files in InitClasses are executed in async tasks in lambda functions in InitClasses, using futures to return the caught exceptions.