Design and Implementation of a desktop capturing application in c++

By Jonas Nilsson

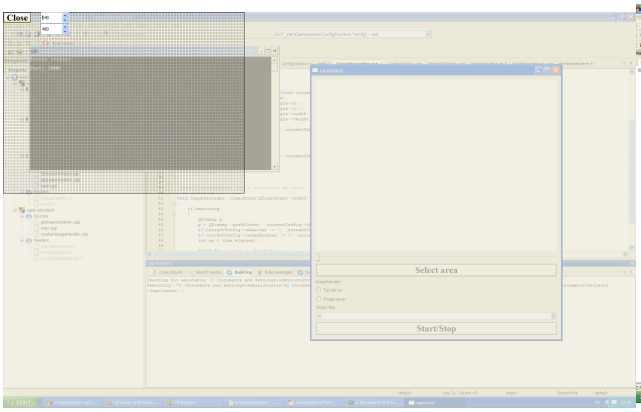
jonas@kajon.se

Introduction:

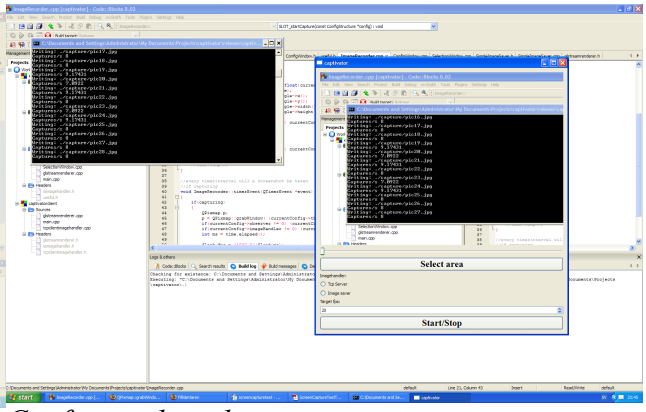
I started by thinking how I would like to share my screen and concluded that the whole screen rarely had to be shared. I would mostly like share an area of the screen where some application is running. I also wanted to develop a design which was easy to extend with new functionality as I didn't really know all possibilities when I started the project. I choose Qt as framework since I had previous experience in it and like the amount of functionality it presents. The program is compiled under Windows using Code::blocks as IDE.

Gui

There exists one main window which contains all options and controls. A selection window can be started from there which enables selection of which area of the desktop to capture. The main window contains an OpenGl renderer which renders the captured images on a quad. The renderer buffer every image which then can be viewed by the use of a slider. A selection can be made between the different image handlers (two at the moment) and the screen capture is started and stopped with a button.

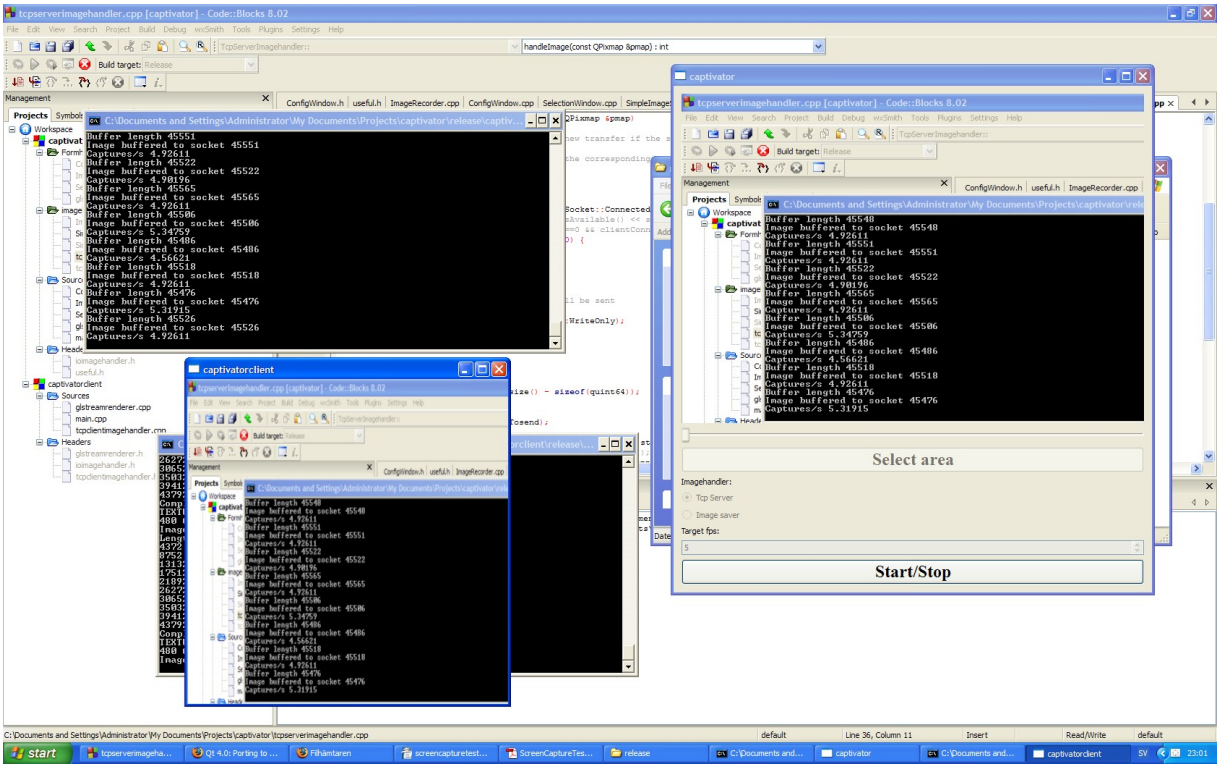


Selection window selecting capture area



Config window showing capture

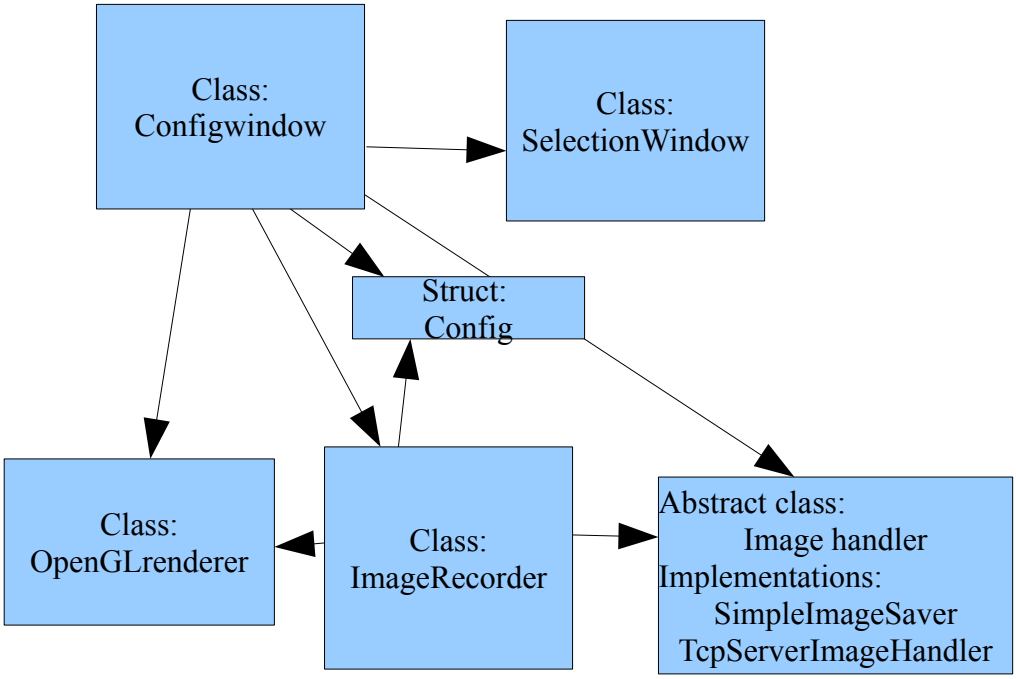
The client's gui is simply the OpenGl renderer without controls at the moment.



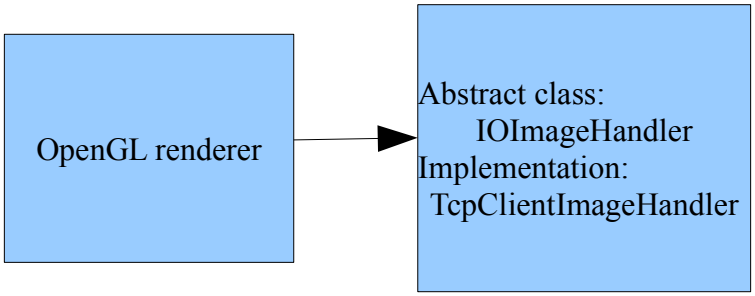
Server and client running on local host

Design

The design of the servers classes looks sort of similar to this, where the arrows correspond to what other classes one class use.



The clients design is more simple and only contains the renderer (same as in the server) and a tcp client.



Classes and heritage

ConfigWindow : QWidget
Displays the configuration gui and create implementations of the other classes. The server sends a pointer with the config structure to the ImageRecorder. The class access the main Qt application which to get a pointer to a QdesktopWidget.

SelectionWindow : QWidget
The selection window displays a fullscreen image with a picture of the desktop. A rectangle (QRect) is rendered on top of the image and can be moved and resized with the mouse. A pointer to the rectangle can be emitted in a signal or returned by a function.

ImageRecorder : QObject
Uses the pointers in the structure ConfigWindow sends to access the functions which will be used for capturing. It then starts a timer which will execute the capturing function until the the class receives a stop signal.
The capturing is handled by accessing functions in QdesktopWidget which returns a Qpixmap. The image handler handleImage function is then called with the Qpixmap as argument. The imagehandler converts the pixmap to a QImage which is faster to use in IO operations.

SimpleImageSaver : (abstract)ImageHandler
This class handleImage function saves an image with the specified filename and format to the harddrive using the Qimage “save” function.

TcpServerImageHandler : (abstract)ImageHandler , QObject
This class starts an QtcpServer which listens to connections on a specified port. The servers ImageHandler function do nothing when no client is connected to the tcp socket. When a client is connected will the server stream the image (and a qint64 containing the size of the image) to Qdatastream buffer. The image is converted to PNG by Qt when streamed to a Qdatastream. The buffer is then written to the tcp socket. The server waits for a packet with one byte to receive on the socket which indicates that the client has read the image and is ready for one more.

GLStreamRenderer : QGLWidget
The OpenGl renderer renders a textured quad which fills the whole view port. It contains some functions to create a new textureid and update the currently used texture id. One uses a Qdatastream(to be connected with the tcpclent) and one other use a Qpixmap directly. The texture id is pushed back in a vector when one of the functions is called. Another function is used to set the current texture to an entry in the vector and thereby enable a replay of saved screen captures.

TcpClientImageHandler : (abstract)IOImageHandler , Qobject
The client is used by the OpenGL renderer which connects its stream pointer with it and waits for a complete signal to be written. The client creates a QTcpSocket which connects to a specified IP and port. It will then send one byte (“N”) to request a new image. Each download is started with a qint64 which contains the size of the image. The client checks the stream each time data is written to the socket to see if the whole image is downloaded, in which case a signal is emitted and captured by the OpenGL renderer which updates the texture using the stream.

Performance

When capturing my desktop of 1650x1050 pixels can I save 2 jpg images/second to the hard drive. I tried to save the uncompressed Qimages to a buffer (list) in the memory instead. It could then capture 3.3 frames/second but the application swallowed 500mb memory after 15 seconds so I quickly stepped away from that path. I also tried to store the encoded images to the buffer (with use of the QIODevice class) but it was not faster than writing to the hard drive directly and caused an extra delay when the capturing stopped to save the images to the disc.
I tried the client with a friend over the Internet and could send him nearly one image per second. But the performance is of course better when capturing smaller parts of the screen.

Causes for performance loss

According to the Qt api the conversion for Qpixmap to Qimage is costly, and will affect the performance significantly as it is used several times per second. Qts function grabWindow used on the desktopWidget is also very expensive and will also affect the performance significantly. Qimage have a large memory overhead as it contains a lot of extra data which i a problem when many Qimages needs to be saved or copied (although I use references or pointers in all functions). Jpg or png encoding is a costly process, especially for large images.

Future updates

Future performance updates:

These are the main updates which might be possible to make to improve the performance.

- Resize captured images to smaller sizes when sending with tcp.
- Check for unnecessary copy functions in existing code.
- Use a video encoder when sending images and converting stored files to video.
- Use a another method than Qt functionality for capturing

Future functionality updates:

If I got time an energy I would like to add the following functionality to the application:

- Replay capability with OpenGL renderer on client side.
- Render FPS in OpenGl renderer.
- Improve the selectionRectangle interaction.
- Enable multiple Tcpclients to connect to the server.
- Use several imageHandlers when capturing the image to save to disc and send to clients simultaneous.
- Make it possible to select a window to capture.