Media ECAD. CLIP tutorial 1.

Configuration, showing, modifying and interacting with pictures

John Robinson

1.1 Getting CLIP running

To get CLIP installed and running on your Windows, Linux or Mac OS-X machine you need:

- a directory/folder with a few megabytes of disk space
- the CLIP zip file http://www.intuac.com/userport/john/clip/clip_1.04.zip which is most easily downloaded via a link from the CLIP 1.04 homepage, http://www.intuac.com/userport/john/clip104.html.

The CLIP zip file contains picture.h, which is the header file containing CLIP itself, a set of test pictures, some useful utilities, all the programs included in these tutorials plus other example programs, and a README file to help you configure CLIP. The rest of this section copies and expands on material in the README file, and similar information also appears on the CLIP homepage.

CLIP adopts the Unix software tools philosophy: "Let each program do one thing well". CLIP programs are therefore most suited to running in a command-line environment. In MS Windows I strongly recommend using cygwin, though you could instead use the MSDOS command prompt. In Linux or on OS-X you will run CLIP programs via a shell in a terminal window. In what follows, it is assumed that whether in Linux, Mac-OS or cygwin, you are running the bash shell.

I recommend that you download clip_1.03.zip into your home directory rather than a subdirectory. When you then expand it, you will get a directory structure with the following levels:

clip - *the top level subdirectory containing configuration scripts and the following subdirectories:*

- bin - *where, by default, all CLIP executables live*
  - src - *sources for CLIP utilities and other programs used as tools in these tutorials*
- include - *containing picture.h and a file called usage.h that several of the utilities use.*
- examples - *example programs*
- tutorial - *containing the programs listed in these tutorials so you don't have to type them in again.*
- pics - *some famous and infamous test images*
- sequence - *a short image sequence*
- progs - *initially empty. It is recommended that this is where you develop CLIP programs.*

More details about the directory structure are given in section 1.2 below.

CLIP programs can be efficiently compiled with cpic - a batch file / shell script that lives in bin. There are various versions; one of these is selected when you run the appropriate makeall script as described below. The important thing to note about cpic is that it puts the compiled executable in the bin directory not in the same directory as the source. This is so that CLIP programs are accessible from anywhere once compiled. However, if you prefer to keep executable and source together, you can easily edit the appropriate cpic script in bin. cpic takes the name of the program you are compiling without its extension.

Overleaf is the step-by-step guide to configuring CLIP and its utilities.

1. Set up your command line interface. On Mac and Linux there is nothing to do. Just open a terminal window running bash and you are set. OnWindows, this is the trickiest part of the configuration. You have to ensure that your command line interface inherits the environment variables that will allow it to run your compiler.

Here are the different situations I've observed:

(i) When running Visual C++ version 6 on Windows 2000, the DOS command prompt and therefore the cygwin shell inherit the environment correctly. I think this is because Windows 2000 still uses autoexec.bat and config.sys, so it's likely that this will also work with other compilers.

(ii) When running Visual Studio .NET (version 7) on Windows XP, you can use the DOS prompt as your command line interface if you run it via the start menu item Visual Studio .NET Tools. The shell that starts up will have all the paths set correctly. If you want to run cygwin in this situation (which is what I do) you need to find the batch file that is pointed to by that start-menu Tools item, copy it to the cygwin startup directory, then merge it with the normal cygwin.bat startup program. A copy of the resulting file for my system is included in this distribution as cygwin_example.bat.

(iii) When running Visual C++ version 6 on Windows XP, you can search down through Program Files, through the Visual Studio Folders, down to a folder called Tools. Inside there will be a .bat file with a name like vsvars32.bat. This is the one to use to start up your DOS prompt or to merge with the cygwin.bat file as under (ii) above.

If you are running cygwin, you could use the gcc compiler instead of Visual C++ or any other made-for-Windows compiler. In that case, you do not have to worry about setting the paths. However, if you have Visual C++ I recommend using it, as gcc does not have support for video input.

One final cygwin issue: Once you have set things up, you can check they are in place by type "which cl" at the cygwin prompt. The response should be something like:

/cygdrive/c/Program Files/Microsoft Visual Studio .NET 2003/VC7/BIN/cl

But then also try "which link". If it finds link in /usr/bin instead of the Visual Studio link, I'm afraid you have to deal with that. I fix the problem by renaming /usr/bin/link.exe to /usr/bin/link.bak. Not very elegant, but it works.

2. Get the CLIP distribution and unzip it. Start up a command prompt or terminal window for the system you are using (e.g. an xterm in Linux, a cygwin terminal window or MSDOS command prompt in MS Windows). Change directory to the clip directory that you have just unpacked.

3. Decide how you are going to run CLIP:

If on a Mac:

If you want image display, you need to run X, and follow step 4(a) below.

If you're happy without image display, you can run under Carbon (the normal Mac window system), and follow step 4(b) below.

If on Linux:

If running X (you probably are), follow step 4(a) below.

If not running X, follow step 4(b) below.

If on a PC:

If you have MS Visual C++ or .NET with C++ and you wish to use that compiler:

If you want to develop and compile using the cygwin command line, follow step 4(c) below.

If you want to develop and compile using the MSDOS command line, follow step 4(d) below.

If you have cygwin, gcc and Xfree installed, follow step 4(e) below.

If you have cygwin and gcc but no Xfree and no other compiler, you will not be able to display images. Follow step 4(a) below.

4(a) Copy makeall.xwin to makeall, then run makeall:

```
$ cp makeall.xwin makeall
$ makeall
```

(b) Copy makeall.nowin to makeall, then run makeall:

```
$ cp makeall.nowin makeall
$ makeall
```

(c) Copy makeall.cygwin_vc to makeall, then run makeall:

```
$ cp makeall.cygwin_vc makeall
$ makeall
```

(d) Run the makeall batchfile:

```
C:\some_directory_list\clip> makeall
```

(e) Copy makeall.cygwin_gcc to makeall, then run makeall:

```
$ cp makeall.cygwin_gcc makeall
$ makeall
```

5. Wait a few minutes. If you are using gcc (i.e. compiling on Linux or Mac or under gcc in cygwin), there will be no feedback while the package compiles (unless it is going wrong). Don't interrupt makeall or else not everything will get made!

makeall sets up cpic in the bin directory so that it is suitable for your chosen configuration. If you decide to change the way you use CLIP in future, you should reconfigure everything by running the appropriate makeall and recompiling all your existing CLIP programs with the new cpic.

When makeall finishes running it asks you to edit a configuration file so that the bin directory is in your "PATH": i.e. so that the command interpreter can find CLIP programs. You must make this edit and then start up a new shell or command prompt so that the changes take effect.

6. You can now test your CLIP installation, e.g. by changing directory to the pics subdirectory and typing "di baboon512". If all has gone well you should be looking at a brightly-coloured picture of a baboon or mandrill.

1.2 What's where

In this and the following sections of the tutorial I assume that you are running a bash shell. If instead you're running under Windows with the DOS command prompt you will have to change some of the commands I refer to: e.g. instead of using "ls" to list files, you will use "dir".

Starting from your home directory, you can get to CLIP by typing "cd clip". (If you've just done step 5 in section 1.1 above and viewed baboon512, do "cd .." to move up a directory from clip/pics.) The clip directory has several subdirectories:

progs When installed, this directory is empty. The idea is that you develop your own CLIP programs here. You don't have to – you can develop them anywhere, but the examples in this tutorial assume that you are writing programs in the progs directory.

bin This is where CLIP's own executable programs live. The edit that you made to your .bashrc file means that whichever directory you are in, the shell will check for programs in *this* directory. "cpic" adds your own executable (i.e. compiled) CLIP programs to the bin directory, and they will then be runnable from any directory. cpic itself is also in bin – it is not a compiled executable but rather a shell script. You can "cd bin" then "cat cpic" to read this script. "cpic" is what allows you compile CLIP programs anywhere on your system, by incorporating references to the clip include directory in the compilation command. If you do "ls" in the bin directory you will see many other files including versions of cpic that are appropriate for other configurations (e.g. cpic.noos which compiles CLIP programs without display or camera support and so is portable to other operating systems).

bin/src Most of the executable programs in the bin directory are compiled versions of CLIP programs. The sources for these programs are in the src subdirectory below bin. These can be browsed as examples of CLIP programming.

include The include directory has all the header files that CLIP uses. The key one is picture.h which provides the complete implementation of CLIP. There is also a header file, usage.h, that is not part of CLIP, but is used in some of the programs in the bin/src directory. You don't need to use usage.h. All you need is picture.h and the first line in your CLIP programs should be:
#include "picture.h"

pics The CLIP distribution includes a variety of standard and non-standard test images in monochrome and colour. All of these are in the raw PNM format (i.e. PGM for monochrome and PPM for colour).

sequences The CLIP distribution also includes a test sequence as a series of PNM files.

1.3 Your first CLIP programs

One of the directories that unzip created when you unpacked the CLIP archive was clip/progs. I suggest you "cd ~/clip/progs" to start developing programs. (The ~ in the cd command is interpreted by the shell as your home directory.)

Hello World

Start up kedit or another editor and type in the following program:

```
#include "picture.h"                          // The CLIP header file
int main() {
        colour_picture inpic("../pics/hello320");     // Instantiate a colour_picture object called inpic
                                                      // from the file hello320 in the directory pics which
                                                      // is a subdirectory of the parent of the current
                                                      // directory.
        inpic.show("Hello World");                    // Show the picture in a window on the screen
        while(inpic.closerequested() == 0)            // Until the user clicks the window close button…
                ;                                     // Do nothing
        }
```

You don't need to type in the comments.

Save the program as file hello.cpp and close the editor, then compile the program with "cpic hello". If all goes well the program will compile and be saved with the name hello (i.e. the source file name without the .cpp extension). Try the program by typing "hello". You should see a courteous greeting which you can close by clicking on the usual window close button. If you see the greeting but can't close the window, or if the program hangs in some other way, type ^C (control-C – i.e. hold down the control key and type C) to abort it. ^C works on all CLIP programs – CLIP does not inhibit interrupts.

What could go wrong? If you have a compilation error, you have probably made a typing mistake. Ensure you have semicolons in the right places and no missing or extra brackets or braces. If nothing is displayed when you run the program, and an error message is printed then you are probably not in the progs subdirectory (i.e. ../pics/hello320 does not get you to the hello320 picture in the pics subdirectory). If you can't seem to close the window, then you may have a typo in the while(…) line. A lot could go wrong but probably it won't, so you have your first CLIP program.

Showing pictures

hello.cpp isn't very general-purpose, so let's adapt it to make a program that will display any picture we like. Copy hello.cpp to show.cpp ("cp hello.cpp show.cpp"), then edit show.cpp to give:

```
#include "picture.h"                            // The CLIP header file
int main(int argc, char *argv[]) {
        if (argc != 2) {
                cout << "Usage: show picname\n";
                return –1;
                }
        colour_picture inpic(argv[1]);          // Instantiate a colour_picture object called inpic
                                                // from the file given by the first argument
        inpic.show(argv[1]);                    // Show the picture in a window on the screen
                                                // with an appropriate title
        while(inpic.closerequested() == 0)      // Until the user clicks the window close button…
                ;                               // Do nothing
        }
```

Compile the program to generate the executable file show. Now you can look at any picture you want, "show ../pics/goldhill512" for example. Note that monochrome pictures like goldhill512 are handled as well as colour pictures like baboon512 ("show ../pics/baboon512"). We often want to know which the picture is, as illustrated in the next program.

Copy show.cpp to inspect.cpp. We are going to add facilities to give the user more information about the picture being viewed. Edit inspect.cpp (mainly with changes near the end of the program) to get the following:

```
#include "picture.h"                            // The CLIP header file
int main(int argc, char *argv[]) {
        if (argc != 2) {
                cout << "Usage: inspect picname\n";
                return –1;
                }
        colour_picture inpic(argv[1]);          // Instantiate a colour_picture object called inpic
                                                // from the file given by the first argument
        inpic.show(argv[1]);                    // Show the picture in a window on the screen
                                                // with an appropriate title
        cout << "Picture " << argv[1] << " is ";        // Print some information about the picture
        if (inpic.ismonochrome())
                cout << "monochrome";
        else
                cout << "colour";
        cout << " with dimensions " << inpic.nrows();
        cout << " rows by " << inpic.ncols() << " columns\n";
        cout << "Maximum value is " << inpic.max();
        cout << ". Minimum value is " << inpic.min() << ".\n";
        int x = 0;                              // Current coordinates
        int y = 0;
        while(inpic.closerequested() == 0)      // Until the user clicks the window close button…
                {
                if (inpic.pointleftbutton() == 1)       // If the left pointer (i.e. mouse) button is pressed…
                        {
                        if ((inpic.pointx() != x)||(inpic.pointy() != y))           // And it has moved
                                {
```

```
                    x = inpic.pointx();
                    y = inpic.pointy();
                    cout << "Value at (" << x << "," << y << ") is ";
                    if (inpic.ismonochrome())
                        cout << inpic.mono[y][x];
                    else {
                        cout << "(" << inpic.red[y][x] << ",";
                        cout << inpic.green[y][x] << ",";
                        cout << inpic.blue[y][x] << ")";
                        }
                    cout << "\n";
                    }
                }
            }
        }
```

This program introduces many new features of the colour_picture object. As well as the closerequested() member function, CLIP offers a range of functions to find out what the user is pointing and clicking at in a picture that is being show()n (i.e. when the show()member function has already been called to display the picture on the screen). pointleftbutton() tells whether the left mouse button is up or down (0 or 1). There is a corresponding function pointrightbutton(). pointx() and pointy() return the current mouse coordinates relative to the top corner of the window where the picture is displayed (provided a button is being pressed). Calling one of these functions on a picture that is not being show()n doesn't make sense, so they just return –1 to signal an error in that case.

The program inspect.cpp also introduces functions that are to do with the picture itself (rather than the user's activity). These work whether or not the picture is being show()n. nrows() and ncols() return the picture's vertical and horizontal sizes, max()and min() return its highest and lowest values, and ismonochrome() returns 0 for true colour pictures and 1 for monochrome pictures. Later in the above program we use ismonochrome() to decide which components of colour_picture to access to print out value information.

colour_picture actually includes four picture_of_int objects. The picture_of_int class is the basic monochrome picture type in CLIP. Its name is a clue that CLIP was designed for *integer*-valued pictures. Every picture element (pixel or pel) value in a picture_of_int is an integer. This is a good thing, as we will see repeatedly in future weeks. (CLIP does support pictures of other data types than integer, but these are not discussed in these tutorials.) The four picture_of_int objects inside a colour_picture object are called mono, red, green and blue. If the colour_picture was constructed (i.e. created) with a true colour image (e.g. by reading in a PPM colour picture) then only the red, green and blue components are used. If it was constructed with a monochrome image (e.g. by reading in a PGM monochrome picture) then only the mono component is used. Nothing that you do after creating a colour_picture can change whether it is a colour or a monochrome picture.

The other new concept included in inspect.cpp is indexing a picture through square brackets. For example in the line

```
                        cout << inpic.green[y][x] << ",";
```

the value of the green pel at coordinates (x,y) is being printed out. This suggests that a picture_of_int (and remember that inpic.green is a picture_of_int) looks like a two-dimensional array when it is indexed by integers. This is indeed the case, but picture_of_int's can also be indexed by other data types (doubles and iranges) as we shall see later. A picture_of_int is *not* the same as a two-dimensional array of integers – it just looks like one when indexed by integers.

If you compile and try the program inspect on a variety of images, you will see that it gives you a lot of information. This is why the CLIP program di is based on inspect. It is more sophisticated – it allows you to view several images at a time; it shows a magnifying-glass inspection view. But the functionality of di is achieved in the same way as the above inspect program. This ability to snoop around an image is so useful

that CLIP provides a library function inspect() that you can include in your programs. If you copy show.cpp to inspect2.cpp, then edit inspect2.cpp to the following program, you will see that using the member function inspect() is a compact way to do some of the things we did explicitly in inspect.cpp.

```
#include "picture.h"                        // The CLIP header file
int main(int argc, char *argv[]) {
        if (argc != 2) {
                cout << "Usage: inspect2 picname\n";
                return –1;
                }
        colour_picture inpic(argv[1]);      // Instantiate a colour_picture object called inpic
                                            // from the file given by the first argument
        inpic.show(argv[1]);                // Show the picture in a window on the screen
                                            // with an appropriate title
        inpic.inspect();                    // Until the user right-clicks, provide info on
                                            // mouse movement with the left button down
        }
```

The difference between inspect2.cpp and inspect.cpp is that inspect2 writes its information to the header of the display window, and the way to get out is to click the right button in the window rather than clicking the close button. The member function inspect() is implemented this way because often you want to inspect an image, right-click to get the program going again, reshow the image after doing some more processing, then inspect it again. You do not necessarily want to close the window.

In inspect.cpp we saw that colour_picture objects include picture_of_int objects. Many clip programs are implemented with colour_pictures and will read in both colour and monochrome images and process them appropriately. Some programs are implemented with picture_of_ints. These too will read in both colour and monochrome images (converting all to monochrome). Programs that just use picture_of_ints are those for which colour does not really make sense. A typical edge detection program will work on the luminance channel only, so the use of picture_of_int makes this explicit. Finally, in this section, we develop two programs that use picture_of_ints.

Copy show.cpp to showcomponents.cpp then edit to get:

```
#include "picture.h"                        // The CLIP header file
int main(int argc, char *argv[]) {
        if (argc != 2) {
                cout << "Usage: showcomponents picname\n";
                return –1;
                }
        colour_picture inpic(argv[1]);      // Instantiate a colour_picture object called inpic
                                            // from the file given by the first argument
        inpic.show(argv[1]);                // Show the picture in a window on the screen
                                            // with an appropriate title
        picture_of_int r(inpic.red);        // Instantiate a picture_of_int object from the
                                            // red channel of inpic
        picture_of_int g(inpic.green);      // Similarly green
        picture_of_int b(inpic.blue);       // Similarly blue
        r.show("Red component");
        g.show("Green component");
        b.show("Blue component");
        while(inpic.closerequested() == 0)  // Until the user clicks the window close button…
                ;                           // Do nothing
        }
```

Compile and test this program on colour and monochrome images. Note that clicking in the close buttons of any of the individual components does not close the window. The only close request that the program is looking for is in the inpic (i.e. the colour picture) window. When this is received, the program terminates so all the windows close. Note also that the member function show() works for picture_of_ints just as for colour_pictures. The vast majority of member functions are common to both picture_of_int, for monochrome pictures, and colour_picture, for colour pictures.

Finally, type in this program and save as bouncethresh.cpp.

```
#include "picture.h"
#include "stdio.h"                                  // To be able to use sprintf below
int main(int argc, char *argv[]) {
        if (argc != 2) {
                cout << "Usage: bouncethresh picname\n";
                return –1;
                }
        picture_of_int inpic(argv[1]);              // If picture is colour, this converts to monochrome
        picture_of_int outpic(inpic.nrows(),inpic.ncols());     // Instantiates a zeroed (black) pictures of
                                                                // the same size as inpic
        outpic.show();
        int threshold = 0;
        int increment = 1;
        while (!outpic.closerequested()) {
                for (int r = 0; r < inpic.nrows(); r++) {
                        for (int c = 0; c < inpic.ncols(); c++) {
                                if (inpic[r][c] < threshold)
                                        outpic[r][c] = 0;
                                else
                                        outpic[r][c] = 255;
                                }
                        }
                threshold += increment;
                if (threshold == 255)
                        increment = -1;
                else if (threshold == 0)
                        increment = 1;
                char comment[256];
                sprintf(comment, "Threshold = %d", threshold);      // Formats an informative string
                                                                    // into the comment buffer…
                outpic.reshow(comment);                             // … which we use to relabel the
                                                                    // window as we show the newly
                                                                    // processed picture.
                }
        }
```

bouncethresh.cpp uses the reshow() member function to refresh the display of the image after processing. Typically, if you want to monitor the execution of a CLIP program, you first put the output image on the screen with show(), then update the view periodically with reshow() between steps of the processing.

You may wonder why CLIP makes you use both show() and reshow(); why not a single function that either starts displaying a picture or refreshes it? As it is, you can't reshow() a picture that hasn't yet been show()n, and you can't show() a picture that is already being show()n. The answer is simply to help catch programming errors: often, while developing an algorithm, you show() then repeatedly reshow() the result, but when you have the algorithm running correctly, you no longer want to see it working. You therefore remove all the show()s and reshow()s from this part of the program. Insisting on first-show()-then-reshow()

makes it easy to remove all the associated display functions from part of a program, and catch any that are mistakenly left behind.

Running the bouncethresh program illustrates the speed of processing and display on CLIP. I use a similar program to compare the performance of different CLIP windowing libraries. bouncethresh.cpp also illustrates indexing of picture_of_ints by integers for both reading and writing. Relatively few operations are done in this way in CLIP, but it is always possible to get at the individual pels if necessary.

1.4 Interactive Display Programs in the CLIP package

You have already encountered the program di. As mentioned previously it can display multiple images. You may try, for example, di ~/clip/pics/* to display all the pictures in the pics directory. di displays a symbolic menu along the top of the displayed pictures. Clicking on symbols with the left mouse button has the following effects: '+', '-' increase and reduce the size of the display window by a factor of two in both horizontal and vertical directions; 'o' restores the window to the size of the image itself; 'i' turns on an inspector window that shows pel values near the cursor as it is moved over the image. 'i' also shows colour histograms. 'r', 'g' and 'b' turn on and off the respective colour channels. You can also use di to display input from a connected webcam. When you do this and open an inspector window with 'i', you will see that the histograms change as the input image changes. Clicking with the right button in an image window closes that window (as does clicking on the usual close box). If the right mouse button is clicked in any image window while the shift key is held down, then *all* image windows are closed and di terminates.

The CLIP program anim plays back a series of picture files in sequence. You can use it either for animating an image sequence, or as a slide show mechanism. To control the framerate, anim provides a parameter picture. Clicking/dragging on this is like moving a slider. Clicking with the left button in the image window pauses the sequence then each successive left-click steps forward one frame. The sequence can be reanimated by clicking in the slider window.

The CLIP program diseq reads all the images in a sequence into memory before starting to display, so it should only be used for relatively short sequences (less than 60 frames). Once started, the left button allows inspection of pel values in the currently-displayed frame, while dragging the right button across the window moves through the sequence of frames.

The CLIP program fiji represents a picture as a 3D surface which you can move interactively to view at different angles. Try, for example, "fiji cameraman256" then drag the pointer around inside the window. Left button does surface rotation, right button does translation. Rather than have the surface displayed as an illuminated surface, you can map a texture onto it. A good example to try is "fiji circlepic cameraman256" (but do "di circlepic cameraman256" first so you can see the different roles of the picture treated as a surface or depthmap (in this case circlepic) and the picture treated as a texture map (in this case cameraman256). The pics directory includes three examples of genuine depthmap colourmap pairs. These may be explored with fiji, though you will have to increase the depthmap gain by using the –g option. Type "fiji" without arguments to see what all the options are, and try "fiji –g 80 face.depthmap face.colmap" for an example.

1.5 Exercises

At least Exercises 1 and 2 should be documented in your logbook.

1. Write a CLIP program that reads in a colour image then allows the user to paint on top of it with the mouse. You may save the result by using the colour_picture member function write("filename");

2. Adapt bouncethresh.cpp to allow interactive thresholding so that, instead of moving the threshold up and down automatically, the threshold is set by the user moving the mouse pointer up and down the picture.

3. You can set every pel in a picture_of_int or a colour_picture to a single value simply using the '=' assignment operator. E.g. picture_of_int a; a = 43; sets every pel in the picture to 43. Adapt showcomponents.cpp so that instead of showing the red, green and blue components as monochrome images, it shows three colour images: withoutred, withoutgreen, withoutblue, where the appropriate colour component is set to zero.

4. Browse the online library documentation for CLIP at http://www.intuac.com/userport/john/clip/doc/index.html. Observe how CLIP is really a template library. In all the tutorials, we deal just with picture_of_int and colour_picture data types. If you ever need to refer to the documentation, you can read clip::picture_of_<PEL> as just picture_of_int. View picture.h in an editor to work out why this is possible (Hint: Search for picture_of_int in picture.h).