SAMPLE KPM MASTER

K P MANSFIELD

MARCH 14, 2013

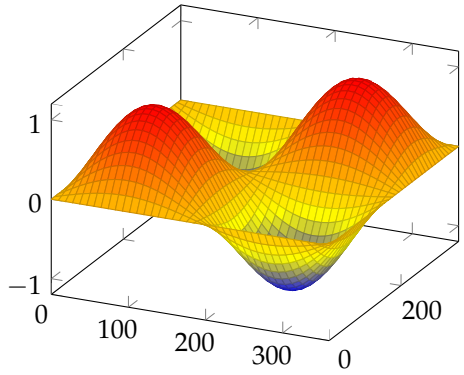
Colour is easy to appreciate, difficult to measure . . . impossible to understand!

As delivered in MSc Session 2011–2012.

Showing some of the capability of the `framed` environment to denote a passage of text.

A very long todonote that certainly will fill more than a single line in the list of todos. Just to make sure let's add some more text . . .

First section—an example pgfplot



Second section—an example of pgfplotstable

dof	error1	info
4	0.25	48
16	$6.25 \cdot 10^{-2}$	25
64	$1.56 \cdot 10^{-2}$	41

Todo list

HERE IS THE WAY that a new thought is expressed in the *tufte-latex* class.

Here is an example of the `marginnote` command (without superscript number) and which allows text styling.

Figures and Tables

Full page width table

Here is an example. Note also the use of the `textcolor` command both in the main text and in `marginnote`. Captions and references to the Table (`label`) are also shown.

The new project is created in *XCode4* from `File > New > Project...` and is named `OpenGLStage2_KM`. A comparison of the file structure of the two projects is shown in Table 1 (green = common; red = unique).

Q. Significance of `MainMenu~.nib` files?

OpenGLStage_KM	OpenGLStage2_KM
build *folder*	—
Controller.h	—
Controller.m	—
DrawingFunctions.h	—
—	~~OpenGLStage2_KM > AppDelegate.h~~
—	~~OpenGLStage2_KM > AppDelegate.m~~
—	OpenGLStage2_KM > en.lproj > Credits.rtf
English.lproj > InfoPlist.strings	OpenGLStage2_KM > en.lproj > InfoPlist.strings
English.lproj > MainMenu.nib	OpenGLStage2_KM > en.lproj > MainMenu.xib
English.lproj > MainMenu~.nib	—
GLView.h	—
GLView.m	—
Help > index.html	—
Info.plist	OpenGLStage2_KM > OpenGLStage2_KM-Info.plist
main.m	OpenGLStage2_KM > main.m
OpenGLStage_KM_Prefix.pch	OpenGLStage2_KM > OpenGLStage2_KM-Prefix.pch
OpenGLStage_KM.xcode.proj	OpenGLStage2_KM.xcodeproj
version.plist	—

Table 1: File listing subset `OpenGLStage_KM` and `OpenGLStage2_KM`

Next . . . continue from here

First subsection

Second subsection

Third subsection

First subsection

Second subsection

Third subsection

Replicating OpenGLStage_KM

File structure comparison

The new project is created in *XCode4* from `File > New > Project...` and is named `OpenGLStage2_KM`. A comparison of the file structure of the two projects is shown in Table 1 (green = common; red = unique). It should be noted that the *root* folder for *OpenGLStage_KM* is `OpenGLStage_KM`; that for *OpenGLStage2_KM* is correspondingly `OpenGLStage2_KM` but the *XCode4* template encapsulates the program files in another folder—`OpenGLStage2_KM`. It is not clear whether this is significant or not.

The two files `AppDelegate.h` and `AppDelegate.m` are *global* in nature (dealing with the *UI*) and it is recommended to use them sparingly or not at all. For the purposes of this exercise they are **removed**. One minor question is the significance of `MainMenu~.nib` in OpenGLStage2_KM.

Delete within *XCode4*.

On `Building` and `Running` *OpenGLStage2_KM*, the product is an app with a menu and an empty window in *Xcode4* but otherwise incomplete.

THE PROCESS TO COMPARE the two projects:

1. compare common files in the two projects (green)
2. add minimal working code to new files in `OpenGLStage2_KM` (red).

Common and files in `OpenGLStage_KM` and `OpenGLStage2_KM`

`InfoPlist.strings` and `InfoPlist.strings`

OpenGLStage_KM was built under a legacy version of *Xcode*—while that of *OpenGLStage2_KM* is built under *XCode4*. Notice that the enclosing folder in *OpenGLStage_KM* here is `English.lproj` compared to `en.lproj` in *OpenGLStage2_KM*.

These files (which are strings files **not** property lists) provide a mechanism for *localizing* user-visible strings in an application's meta-file—which is contained in its `Info.plist` file.

Here is the existing code in *OpenGLStage_KM* and *OpenGLStage2_KM* respectively which is left unchanged:

```
/* Localized versions of Info.plist keys */

CFBundleName = "OpenGLStage_KM";
NSHumanReadableCopyright = "©" __MyCompanyName__, 2007";
```

English.lproj > InfoPList.strings

```
/* Localized versions of Info.plist keys */
```

OpenGLStage2_KM > en.lproj > InfoPList.strings

MainMenu.nib and MainMenu.xib

The .NIB file is a resource file that stores the visual part of the application (windows, views *etc*) and is also used to configure *nonvisual* objects (such as controller objects). The .NIB file is an *InterfaceBuilder* document which creates an object graph. This is archived when you save the file and when the file is loaded the object graph is unarchived *viz*. at runtime. The .NIB file in *OpenGLStage_KM* is the original interface implementation in *InterfaceBuilder*. It seems to be a *compiled* version and cannot be opened in *XCode4*.

object graph: a group of objects forming a network through their relationships with each other either through direct reference to an object or through a chain of intermediate references.

It seems that *InterfaceBuilder* is now accessed **directly** in *XCode4* to provide an improved graphical editor. The new format is the .XIB file (a plain text XML file). The customisation of this new interface can be left to a later stage.

Use only *XCode4* to edit these files directly.

Info.plist and OpenGLStage2_KM-Info.plist

These files are the applications' *information property list* file (Info.plist)(Figures 1 and 2).

Key	Type	Value
Localization native development region	String	English
Executable file	String	OpenGLStage_KM
Icon file	String	
Bundle identifier	String	com.apple.myCocoaApp
InfoDictionary version	String	6.0
Bundle OS Type code	String	APPL
Bundle creator OS Type code	String	????
Bundle version	String	1.0
Main nib file base name	String	MainMenu
Principal class	String	NSApplication

Figure 1: Info.plist

A property list is "a structured data representation used by *Cocoa* and *Core Foundation* as a convenient way to store, organize, and access standard types of data. It is colloquially referred to as a "plist" ". Here is Apple's description of a property list:

> A property list is a representation of a hierarchy of objects that can be stored in the file system and reconstituted later. Property lists give applications a lightweight and portable way to store small amounts of data. They are hierarchies of data made from specific types of objects—they are, in effect, an object graph. Property lists are easy to create programmatically and are even easier to serialize into a representation that is persistent. Applications can later read the static representation back into memory and recreate the original hierarchy of objects.

Key	Type	Value
Localization native development region	String	en
Executable file	String	${EXECUTABLE_NAME}
Icon file	String	
Bundle identifier	String	kpm.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
Minimum system version	String	${MACOSX_DEPLOYMENT_TARGET}
Copyright (human-readable)	String	Copyright © 2012 UCL Bartlett School of Graduate Studies. All rights reserved.
Main nib file base name	String	MainMenu
Principal class	String	NSApplication

Figure 2: OpenGLStage2_KM > OpenGLStage2_KM-Info.plist

Apple's notes about the storage and editing of property lists: "The preferred way to store property lists on Mac OS X and iOS is as an XML file. . . these files have the advantages of being human-readable and in the standards-based XML format. Generally, there is little need to create or edit XML property lists yourself, but if you do, use *Xcode's* built-in property list editor or the *Property List Editor* application (which is part of the tools package). You should not edit the XML data in a text editor unless you are very familiar with XML syntax and the format of property lists.
These files do not need to be edited.

.

`main.m` *and* `main.m`

These files are automatically populated by *XCode4* when the project is created (including the commented section):

```
//
//  main.m
//  OpenGLStage_KM
//
//  Created by Kevin Mansfield on 14/07/2007.
//  Copyright KPM 2007. All rights reserved.
//

#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc,  (const char **) argv);
}
```

`main.m`

```
//
//  main.m
//  OpenGLStage2_KM
//
//  Created by Kevin Mansfield on 15/06/2012.
//  Copyright (c) 2012 UCL Bartlett School of Graduate Studies. All rights
      reserved.
//

#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc, (const char **)argv);
}
```

`OpenGLStage2_KM > main.m`

The job of `main.m` consists of two steps:

1. to set up a core group of objects
2. to turn program control over to those objects.

The *main* function is extremely simple and consists of only one call to `NSApplicationMain`. This function

> creates the application object, sets up an autorelease pool, loads the initial user interface from the main nib file, and runs the application, thereby requesting it to begin handling events received on the main event loop.

To use Frameworks in the project, the framework is added to the project through the Linked Frameworks and Library pane (in *XCode4*) and the top-level header file is included in the source file. The use of the `#import` directive ensures that the same header file is never included more than once. Syntax is

`#import <Framework_name/Header_filename.h>`

e.g. `#import <Cocoa/Cocoa.h>`

viz. here is the folder listing in the two projects...

OpenGLStage_KM

Frameworks > Linked Frameworks > Cocoa.framework

Frameworks > Linked Frameworks > OpenGL.framework

Frameworks > Other Frameworks > Foundation.framework

Frameworks > Other Frameworks > AppKit.framework

OpenGLStage2_KM

Frameworks > Cocoa.framework

Frameworks > Other Frameworks > AppKit.framework

Frameworks > Other Frameworks > CoreData.framework

Frameworks > Other Frameworks > Foundation.framework

Linked Frameworks and Libraries

Cocoa.framework

Standard locations for Frameworks are `/System/Library/Frameworks` or `/Users/kpm/Library/Frameworks`.

`OpenGLStage2_Prefix.h` and `OpenGLStage2_KM-Prefix.pch`

A prefix file is a *header* file that is generated automatically by many *Xcode* project templates—they can include "umbrella frameworks appropriate to the selected type of application".

Here are the Prefix files generated in each of the projects:

```
//
// Prefix header for all source files of the 'OpenGLStage2'
// target in the 'OpenGLStage2' project
//

#ifdef __OBJC__
    #import <Cocoa/Cocoa.h>
#endif
```

`OpenGLStage2_Prefix.h`

```
//
// Prefix header for all source files of the 'OpenGLStage2_KM' target in
     the 'OpenGLStage2_KM' project
//

#ifdef __OBJC__
    #import <Cocoa/Cocoa.h>
#endif
```

`OpenGLStage2_KM > OpenGLStage2_KM-Prefix.pch`

They are identical and are therefore left unchanged.

OpenGLStage2.pbproj and OpenGLStage2_KM.xcodeproj

Here is Apple's definition of an *Xcode* project:

> An Xcode project is a repository for all the files, resources, and information required to build one or more software products. A project contains all the elements used to build your products and maintains the relationships between those elements. It contains one or more targets, which specify how to build products. A project defines default build settings for all the targets in the project (each target can also specify its own build settings, which override the project build settings).

`OpenGLStage2.pbproj` is a legacy *ProjectBuilder* project and cannot be opened in *XCode4*.

Credits.rtf

A generated file in *XCode4* which is self-explanatory.

```
Engineering:
        Some people

Human Interface Design:
        Some other people

Testing:
        Hopefully not nobody

Documentation:
        Whoever

With special thanks to:
        Mom
```

`OpenGLStage2_KM > en.lproj > Credits.rtf`

The MVC *paradigm*

The program is roughly structured using the *Model-View-Controller* paradigm. According to the design notes, there is no *Model* as such. The *View* part (*GLView* class) "handles 3D rendering with the *Controller* class negotiating between the stepper controls and the rendering engine". *OpenGLStage_KM* provides a skeleton example of an *OpenGL* program. Use this as a reference to develop the basic functionality of *OpenGLStage2_KM*. A first step is to add **partial** versions of `GLView.h` and `GLView.m` from *OpenGLStage_KM* to *OpenGLStage2_KM*.

See `Help > Page03.html`.

GLView.h

To create a file in a project, go to `Utilities > Show File Template Library` and drag the appropriate file type into the project (this ensures correct linking *etc*). Here is the original code from *OpenGLStage_KM* placed in *OpenGLStage2_KM*:

```
/* GLView */

#import <Cocoa/Cocoa.h>
```

```
@interface GLView : NSOpenGLView
{
}
@end
```

GLView.h

```
//
//  GLView.h
//  OpenGLStage2_KM
//
//  Created by Kevin Mansfield on 14/06/2012.
//  Copyright (c) 2012 UCL Bartlett School of Graduate Studies. All rights
      reserved.
//

#import <Cocoa/Cocoa.h>

@interface GLView : NSOpenGLView
{
}
@end
```

OpenGLStage2_KM > GLView.h

GLView.m

Similarly, here is **partial** code from *OpenGLStage_KM* placed into *OpenGLStage2_KM* (extra code commented out):

```
//#include compiler directive. Syntax: #include <Framework/Header.h>
#include <OpenGL/gl.h>

#import "GLView.h"
#import "DrawingFunctions.h"

@implementation GLView

//Overriding the view's drawRect method to draw OpenGL content.
//Syntax: - (void) drawRect: (NSRect) aRect
-(void) drawRect: (NSRect) rect

{

        //Specifies red, green, blue and alpha values used
        //(clamped to a range [0, 1]) when color buffers are cleared.
        //Syntax: void glClearColor (GLclampf red, GLclampf green,
        //GLclampf blue, GLclampf alpha).
        glClearColor(0.2,0.2,1.0,0.1);

        //Sets the bitplane area of the window to value previously
        //selected by glClearColor. The argument indicates which
        //buffer(s) are to be cleared viz. the buffer enabled for
        //color writing.
        //Syntax: void glClear (GLbitfield mask)
        glClear(GL_COLOR_BUFFER_BIT);

        //Function defined in DrawingFunctions.h called from main()
        //with appropriate value passed to function.
        //Syntax: FunctionName (parameter_value);
        DrawBoundingBox (0.3);

        //Different GL implementations 'buffer' commands in several
        //different locations (inc. the graphics accelerator itself).
        //glFlush empties all buffers as quickly as the rendering
        //engine allows. Call glFlush when depending on the completion
        //of previously issued commands eg. user input depending upon a
```

```
        //generated image.
        //Syntax: void glFlush(void).
        glFlush();

}
@end
```

GLView.m

```
//#include compiler directive. Syntax: #include <Framework/Header.h>
#include <OpenGL/gl.h>

#import "GLView.h"
//#import "DrawingFunctions.h"

@implementation GLView

//Overriding the view's drawRect method to draw OpenGL content.
//Syntax: - (void) drawRect: (NSRect) aRect
-(void) drawRect: (NSRect) rect

{

        //Specifies red, green, blue and alpha values used
        //(clamped to a range [0, 1]) when color buffers are cleared.
        //Syntax: void glClearColor (GLclampf red, GLclampf green,
        //GLclampf blue, GLclampf alpha).
        glClearColor(0.2,0.2,1.0,0.1);

        //Sets the bitplane area of the window to value previously
        //selected by glClearColor. The argument indicates which
        //buffer(s) are to be cleared viz. the buffer enabled for
        //color writing.
        //Syntax: void glClear (GLbitfield mask)
        glClear(GL_COLOR_BUFFER_BIT);

        //Function defined in DrawingFunctions.h called from main()
        //with appropriate value passed to function.
        //Syntax: FunctionName (parameter_value);
        //DrawBoundingBox (0.3);

        //Different GL implementations 'buffer' commands in several
        //different locations (inc. the graphics accelerator itself).
        //glFlush empties all buffers as quickly as the rendering
        //engine allows. Call glFlush when depending on the completion
        //of previously issued commands eg. user input depending upon a
        //generated image.
        //Syntax: void glFlush(void).
        glFlush();

}
@end
```

OpenGLStage2_KM > GLView.m

Controller.h

Here is the original code from *OpenGLStage_KM* followed by that placed in *OpenGLStage2_KM*:

```
/* Controller */

/* #import automatically includes a
header file only once. Suspect that
<---> represents a 'system' file and
"---" represents a 'user' file.
Q. Why include Cocoa/Cocoa.h and
GLView.h here? */
```

```
#import <Cocoa/Cocoa.h>
#import "GLView.h"

/* @interface introduces the essential
declarations of the interface. Declaration
is ended by @end.
Syntax: @interface ClassName : Superclass */
@interface Controller : NSObject
{
        /* IBOutlet references an
         Interface Builder object in
         an application. Best to statically
         type it as a pointer to an object.
         Syntax: IBOutlet Object *myObject; */
        IBOutlet GLView *glView;
}

/* An IBAction is a specific method when
 a control is triggered. An action method
 takes a single parameter of type (id) and
 returns the type IBAction.
 Syntax: -(IBAction)actionMethod:(id)sender; */
-(IBAction)launchHelp:(id)sender;

@end
```

Controller.h

```
/* Controller */

/* #import automatically includes a
header file only once. Suspect that
<---> represents a 'system' file and
"---" represents a 'user' file.
Q. Why include Cocoa/Cocoa.h and
GLView.h here? */
#import <Cocoa/Cocoa.h>
#import "GLView.h"

/* @interface introduces the essential
declarations of the interface. Declaration
is ended by @end.
Syntax: @interface ClassName : Superclass */
@interface Controller : NSObject
{
        /* IBOutlet references an
         Interface Builder object in
         an application. Best to statically
         type it as a pointer to an object.
         Syntax: IBOutlet Object *myObject; */
        IBOutlet GLView *glView;
}

/* An IBAction is a specific method when
 a control is triggered. An action method
 takes a single parameter of type (id) and
 returns the type IBAction.
 Syntax: -(IBAction)actionMethod:(id)sender; */
-(IBAction)launchHelp:(id)sender;

@end
```

OpenGLStage2_KM > Controller.h

Controller.m

Similarly, is the original code from *OpenGLStage_KM* followed by that placed in *OpenGLStage2_KM*:

```
/* Imports the appropriate header file. */
#import "Controller.h"

/* @implementation declares the definitions of
 the methods. Similar to the interface block it has
 no data section. Declaration
 is ended by @end.
 Syntax: @implementation ClassName */
@implementation Controller

//-(void)awakeFromNib
//{

//}

// TBC.....
-(IBAction)launchHelp:(id)sender
{

        NSBundle * mainBundle = [NSBundle mainBundle];
        NSString * filename = [mainBundle pathForResource: @"index" ofType
             :@"html" inDirectory:@"Help"];
        if(![[NSWorkspace sharedWorkspace] openFile: filename
             withApplication:@"Safari"])
        {
                if(![[NSWorkspace sharedWorkspace] openFile: filename
                     withApplication: @"Internet Explorer"])
                        NSBeep();
        }

}

@end
```

Controller.m

```
/* Imports the appropriate header file. */
#import "Controller.h"

/* @implementation declares the definitions of
 the methods. Similar to the interface block it has
 no data section. Declaration
 is ended by @end.
 Syntax: @implementation ClassName */
@implementation Controller

//-(void)awakeFromNib
//{

//}

// TBC.....
-(IBAction)launchHelp:(id)sender
{

        NSBundle * mainBundle = [NSBundle mainBundle];
        NSString * filename = [mainBundle pathForResource: @"index" ofType
             :@"html" inDirectory:@"Help"];
        if(![[NSWorkspace sharedWorkspace] openFile: filename
             withApplication:@"Safari"])
        {
                if(![[NSWorkspace sharedWorkspace] openFile: filename
                     withApplication: @"Internet Explorer"])
                        NSBeep();
        }

}
```

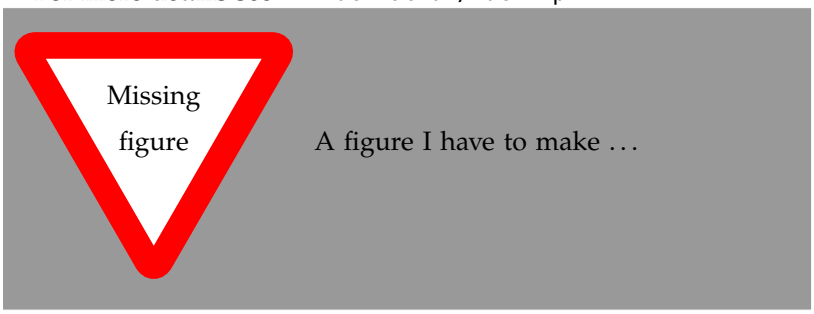
```
@end
```

OpenGLStage2_KM > Controller.m

Introduction

This lecture is divided into the following sections.[1]

For more details see www.ucl.ac.uk/~ucftkpm



[1] Edward R. Tufte. *Beautiful Evidence*. Graphics Press, LLC, first edition, May 2006. ISBN 0-9613921-7-7; Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 2001. ISBN 0-9613921-4-2; and Edward R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990. ISBN 0-9613921-1-8

Colour and the Visual Process

We consider colour vision, adaptation and constancy (see figure 13) and continue with the colour response of rods and cones (figure 11).

Reference is then made to opponent colour theory (see figure 10).

This is a margin note. Notice that there is no number preceding the note, and there is no number in the main text where this note was written.

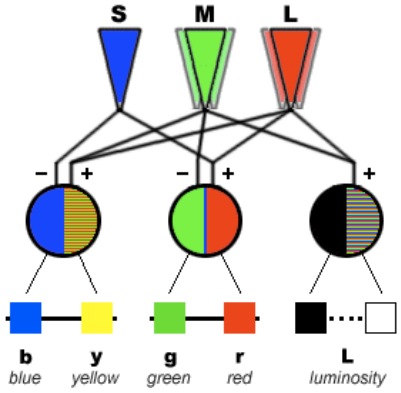


Figure 3: Generic colour appearance model (www.handprint.com/HP/WCL/color2.html)

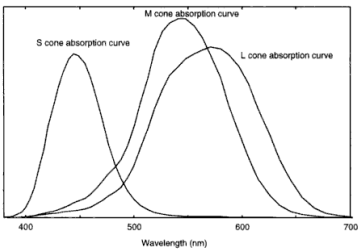


Figure 4: Cone absorption curves (Sangwine S and Home R. *The Colour Image Processing Handbook*. Springer (1998) p12)

Colour Measurement and Quantification

From the studies of Newton we define additive mixing (see figure 12). Grassman derived his additivity law in 1853. We consider red, blue and green mixing and the need for negative values. The introduction of *imaginary* sources allows transition to the CIE system. The features of the CIE system are described including distribution curves, tristimulus values and chromaticity values.

Calculation and measurement within the CIE system are described with reference to the CIE diagram which is used to describe the colours of light sources and objects lit with a particular light source. The mixture of two colours will lie on the straight line between them. Different parts of the diagram have different subjective

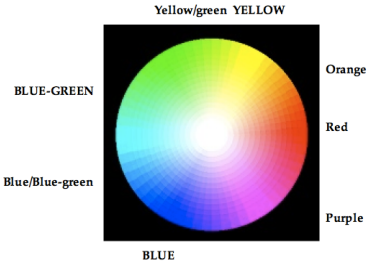


Figure 5: Colour circle (www.realcolorwheel.com/final.htm)

differences. The diagram is further developed in the CIE uniform chromaticity scale (UCS) uv and $u'v'$ diagram.

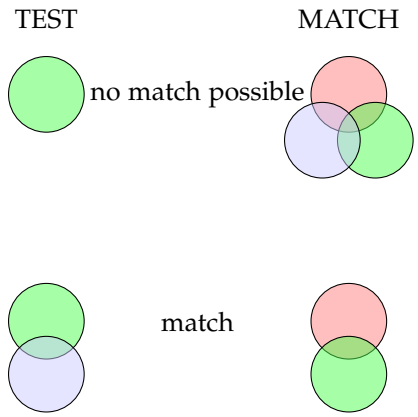


Figure 6: Extension of the definition of mixing. 'Negative' mixing in that adding blue to the spot on the left is equivalent to adding *'negative'* blue on the right

Light Source Colour Performance

We describe the electromagnetic spectrum and various light source spectral distributions before moving on to their measurement. Light source colour description is in terms of *colour appearance* (CIE correlated colour temperature (CCT)) and colour rendering (CIE colour rendering index (CRI) R_a).

Surface Colour Systems

We consider the Munsell Colour System 1905 (refined 1976) with its components Value, Hue and Chroma and the Natural Colour System (NCS).

Table 5 shows the CIE system. Another version is shown in Table 6 and a final version in Table 7.

$$(C) \equiv x(X) + y(Y) + z(Z)$$
where
$$x = \frac{X}{(X+Y+Z)}$$
$$y = \frac{Y}{(X+Y+Z)}$$
$$z = \frac{Z}{(X+Y+Z)}$$
and hence
$$x + y + z = 1$$

Table 2: CIE system

$(C) \equiv x(X) + y(Y) + z(Z)$ where $x = \frac{X}{(X+Y+Z)}$ $y = \frac{Y}{(X+Y+Z)}$ $z = \frac{Z}{(X+Y+Z)}$ and hence $x + y + z = 1$

Table 3: CIE system

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Here is a short section of text where we want multiple references. The best overview of colour is by Kuehni[2]. Simons and Bean[3] provide useful computational techniques and a full discussion of different colour metrics is given in Guo and Houser.[4]

$(C) \equiv x(X) + y(Y) + z(Z)$ where $x = \frac{X}{(X+Y+Z)}$ $y = \frac{Y}{(X+Y+Z)}$ $z = \frac{Z}{(X+Y+Z)}$ and hence $x + y + z = 1$

Table 4: CIE system

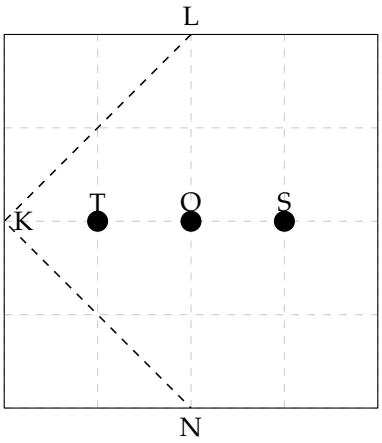
[2] Edward R. Tufte. *Beautiful Evidence*. Graphics Press, LLC, first edition, May 2006. ISBN 0-9613921-7-7

[3] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 2001. ISBN 0-9613921-4-2

[4] Edward R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990. ISBN 0-9613921-1-8

Miscellaneous TikZ Examples

Some interesting TikZ examples from a variety of sources. A graph example is shown in figure 14.



Şekil I

Figure 7: A graph example.

Here is another small TikZ example (see figure 15).

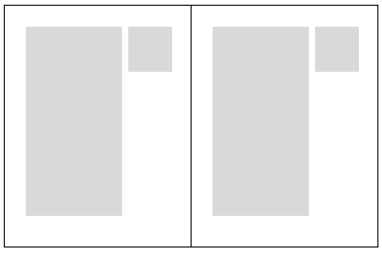


Figure 8: A small example.

Finally a small margin figure in TikZ (see figure 16).

```
//
//  main.m
//  OpenGLStage2
//
//  Created by James De Spears on Sun Feb 16 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

#import <Cocoa/Cocoa.h>

int main(int argc, const char *argv[])
{
    return NSApplicationMain(argc, argv);
}
```

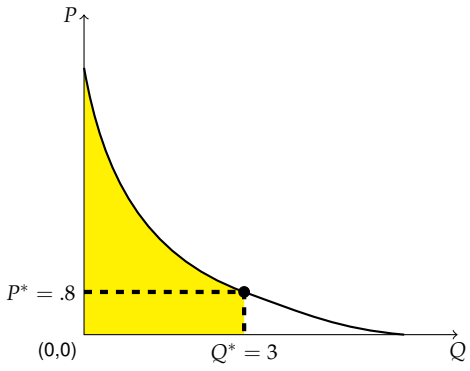


Figure 9: An economics graph.

main.m

```
//
//  main.m
//  OpenGLStage2
//
//  Created by James De Spears on Sun Feb 16 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//

#import <Cocoa/Cocoa.h>

int main(int argc, const char *argv[])
{
    return NSApplicationMain(argc, argv);
}
```

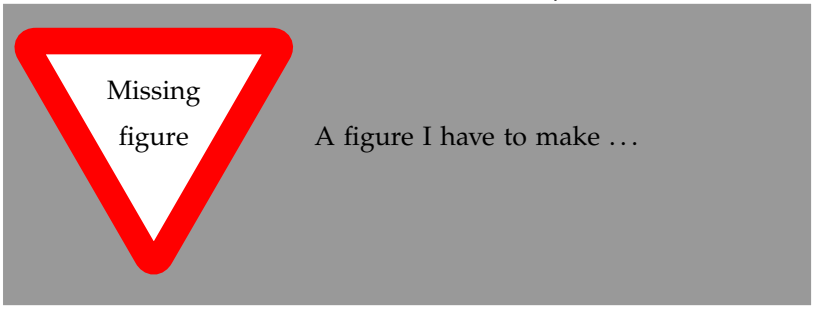
main.m

Showing some of the capability of the `framed` environment to denote a passage of text.

Introduction

This lecture is divided into the following sections.[5]

For more details see `www.ucl.ac.uk/~ucftkpm`



[5] Edward R. Tufte. *Beautiful Evidence*. Graphics Press, LLC, first edition, May 2006. ISBN 0-9613921-7-7; Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 2001. ISBN 0-9613921-4-2; and Edward R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990. ISBN 0-9613921-1-8

Colour and the Visual Process

We consider colour vision, adaptation and constancy (see figure 13) and continue with the colour response of rods and cones (figure 11).

Reference is then made to opponent colour theory (see figure 10).

This is a margin note. Notice that there is no number preceding the note, and there is no number in the main text where this note was written.

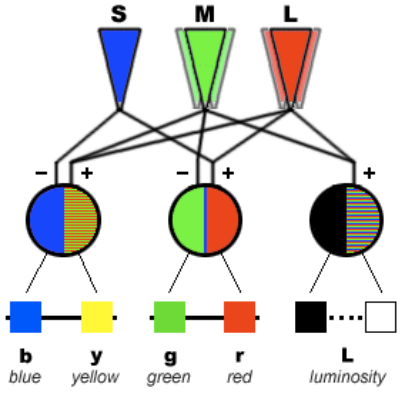


Figure 10: Generic colour appearance model (`www.handprint.com/HP/WCL/color2.html`)

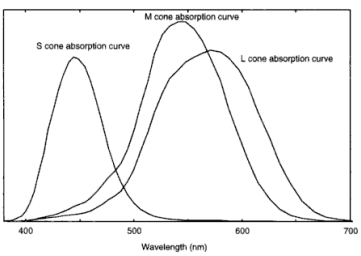


Figure 11: Cone absorption curves (Sangwine S and Home R. *The Colour Image Processing Handbook*. Springer (1998) p12)

Colour Measurement and Quantification

From the studies of Newton we define additive mixing (see figure 12). Grassman derived his additivity law in 1853. We consider red, blue and green mixing and the need for negative values. The introduction of *imaginary* sources allows transition to the CIE system. The features of the CIE system are described including distribution curves, tristimulus values and chromaticity values.

Calculation and measurement within the CIE system are described with reference to the CIE diagram which is used to describe the colours of light sources and objects lit with a particular light source. The mixture of two colours will lie on the straight line between them. Different parts of the diagram have different subjective differences. The diagram is further developed in the CIE uniform chromaticity scale (UCS) uv and $u'v'$ diagram.

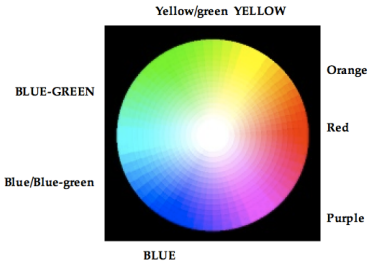


Figure 12: Colour circle (`www.realcolorwheel.com/final.htm`)

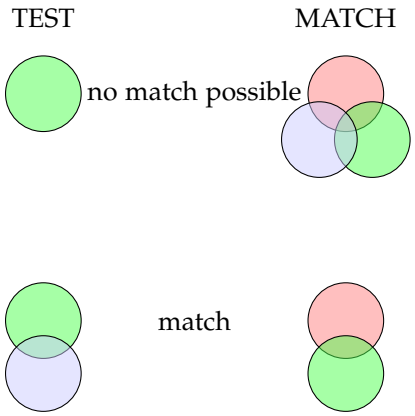


Figure 13: Extension of the definition of mixing. 'Negative' mixing in that adding blue to the spot on the left is equivalent to adding *'negative'* blue on the right

Light Source Colour Performance

We describe the electromagnetic spectrum and various light source spectral distributions before moving on to their measurement. Light source colour description is in terms of *colour appearance* (CIE correlated colour temperature (CCT)) and colour rendering (CIE colour rendering index (CRI) R_a).

Surface Colour Systems

We consider the Munsell Colour System 1905 (refined 1976) with its components Value, Hue and Chroma and the Natural Colour System (NCS).

Table 5 shows the CIE system. Another version is shown in Table 6 and a final version in Table 7.

$(C) \equiv x(X) + y(Y) + z(Z)$
where
$x = \frac{X}{(X+Y+Z)}$
$y = \frac{Y}{(X+Y+Z)}$
$z = \frac{Z}{(X+Y+Z)}$
and hence
$x + y + z = 1$

Table 5: CIE system

Table 6: CIE system

$(C) \equiv x(X) + y(Y) + z(Z)$
where
$x = \frac{X}{(X+Y+Z)}$
$y = \frac{Y}{(X+Y+Z)}$
$z = \frac{Z}{(X+Y+Z)}$
and hence
$x + y + z = 1$

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Here is a short section of text where we want multiple references. The best overview of colour is by Kuehni[6]. Simons and Bean[7] provide useful computational techniques and a full discussion of different colour metrics is given in Guo and Houser.[8]

$(C) \equiv x(X) + y(Y) + z(Z)$
where
$x = \frac{X}{(X+Y+Z)}$
$y = \frac{Y}{(X+Y+Z)}$
$z = \frac{Z}{(X+Y+Z)}$
and hence
$x + y + z = 1$

Table 7: CIE system

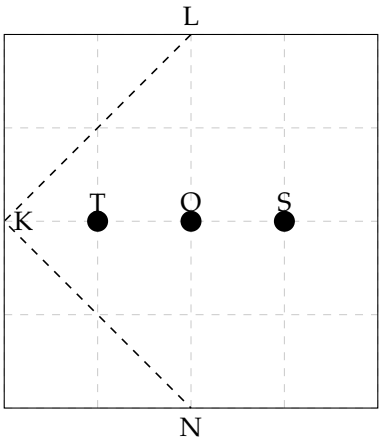
[6] Edward R. Tufte. *Beautiful Evidence*. Graphics Press, LLC, first edition, May 2006. ISBN 0-9613921-7-7

[7] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 2001. ISBN 0-9613921-4-2

[8] Edward R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990. ISBN 0-9613921-1-8

Miscellaneous TikZ Examples

Some interesting TikZ examples from a variety of sources. A graph example is shown in figure 14.



Şekil I

Figure 14: A graph example.

Here is another small TikZ example (see figure 15).

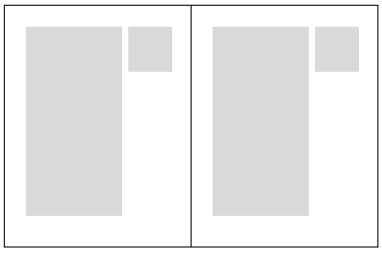


Figure 15: A small example.

Finally a small margin figure in TikZ (see figure 16).
Bibliography example([9]).

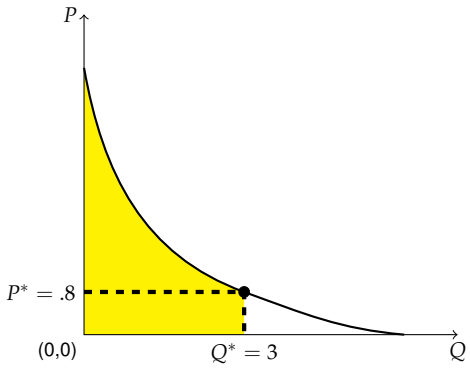


Figure 16: An economics graph.

[9] Edward R. Tufte. *Visual Explanations*. Graphics Press, Cheshire, Connecticut, 1997. ISBN 0-9613921-2-6

References

Edward R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990. ISBN 0-9613921-1-8.

Edward R. Tufte. *Visual Explanations*. Graphics Press, Cheshire, Connecticut, 1997. ISBN 0-9613921-2-6.

Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 2001. ISBN 0-9613921-4-2.

Edward R. Tufte. *Beautiful Evidence*. Graphics Press, LLC, first edition, May 2006. ISBN 0-9613921-7-7.