OpenGL API Introduction

Agenda

- Structure of Implementation
- Paths and Setup
- API, Data Types & Function Name Conventions
- GLUT
- Buffers & Colours

OpenGL: An API, Not a Language

- OpenGL is not a programming language; it is an application programming interface (API).
- An OpenGL application is a program written in some programming language (such as C or C++) that makes calls to one or more of the OpenGL libraries.
- Does not mean program uses OpenGL exclusively to do drawing:
    - It might combine the best features of two different graphics packages.
    - Or it might use OpenGL for only a few specific tasks and environment-specific graphics (such as the Windows GDI) for others.

Implementations

- Although OpenGL is a “standard” programming library, this library has many implementations and versions.
- On Microsoft Windows the implementation is in the opengl32.dll dynamic link library, located in the Windows system directory.
- The OpenGL library is usually accompanied by the OpenGL utility library (GLU), which on Windows is in glu32.dll,.
- GLU is a set of utility functions that perform common tasks, such as special matrix calculations, or provide support for common types of curves and surfaces.
- On Mac OS X, OpenGL and the GLU libraries are both included in the OpenGL Framework.
- The steps for setting up your compiler tools to use the correct OpenGL headers and to link to the correct OpenGL libraries vary from tool to tool and from platform to platform.

Include Paths

- On all platforms, the prototypes for all OpenGL functions, types, and macros are contained (by convention) in the header file gl.h.
- The utility library functions are prototyped in a different file, glu.h.
- These files are usually located in a special directory in your include path, set up automatically when you install your development tools.

```
#include<windows.h>
#include<gl/gl.h>
#include<gl/glu.h>
```

```
#include <Carbon/Carbon.h>
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
```

Variations on Header setup

```
// Bring in OpenGL
// Windows
#ifdef WIN32
#include <windows.h>       // Must have for Windows platform builds
#include "glee.h"          // OpenGL Extension "autoloader"
#include <gl\gl.h>         // Microsoft OpenGL headers
#include <gl\glu.h>        // OpenGL Utilities
#include "glut.h"          // Glut (Free-Glut on Windows)
#endif

// Mac OS X
#ifdef __APPLE__
#include <Carbon/Carbon.h> // Brings in most Apple specific stuff
#include "glee.h"           // OpenGL Extension "autoloader"
#include <OpenGL/gl.h>      // Apple OpenGL haders (version depends
                             //on OS X SDK version)
#include <OpenGL/glu.h>     // OpenGL Utilities
#include <Glut/glut.h>      // Apples Implementation of GLUT

// Just ignore sleep on Apple
#define Sleep(x)

#endif

#ifdef linux
#include "GLee.h"
#include <GL/gl.h>
#include <GL/glu.h>
#include <glut.h>
#include <stdlib.h>
#endif
```

API Specifics

- OpenGL applies some standard rules to the way functions were named and variables were declared.
- The API is simple and clean and easy for vendors to extend. OpenGL tries to avoid as much “policy” as possible.
    - Policy: assumptions that the designers make about how programmers will use the API.
- Eg:
    - assuming that you always specify vertex data as floating-point values,
    - assuming that fog is always enabled before any rendering occurs,
    - assuming that all objects in a scene are affected by the same lighting parameters

API Specifics

- This philosophy has contributed to the longevity and evolution of OpenGL.
- OpenGL’s basic API has shown surprising resilience to new unanticipated features.
- The ability to compile ten-year-old source code with little to no changes is a substantial advantage to application developers, and OpenGL has managed for years to add new features with as little impact on old code as possible.
- Some versions of OpenGL offer “lean and mean” profiles, where some older features and models may eventually be dropped - eg OpenGL ES

Data Types

- To make it easier to port OpenGL code from one platform to another, OpenGL defines its own data types.
- These data types map to normal C/C++ data types that you can use instead, if you want.
- The various compilers and environments, however, have their own rules for the size and memory layout of various variable types.
- By using the OpenGL defined variable types, you can insulate your code from these types of changes.

Data Types

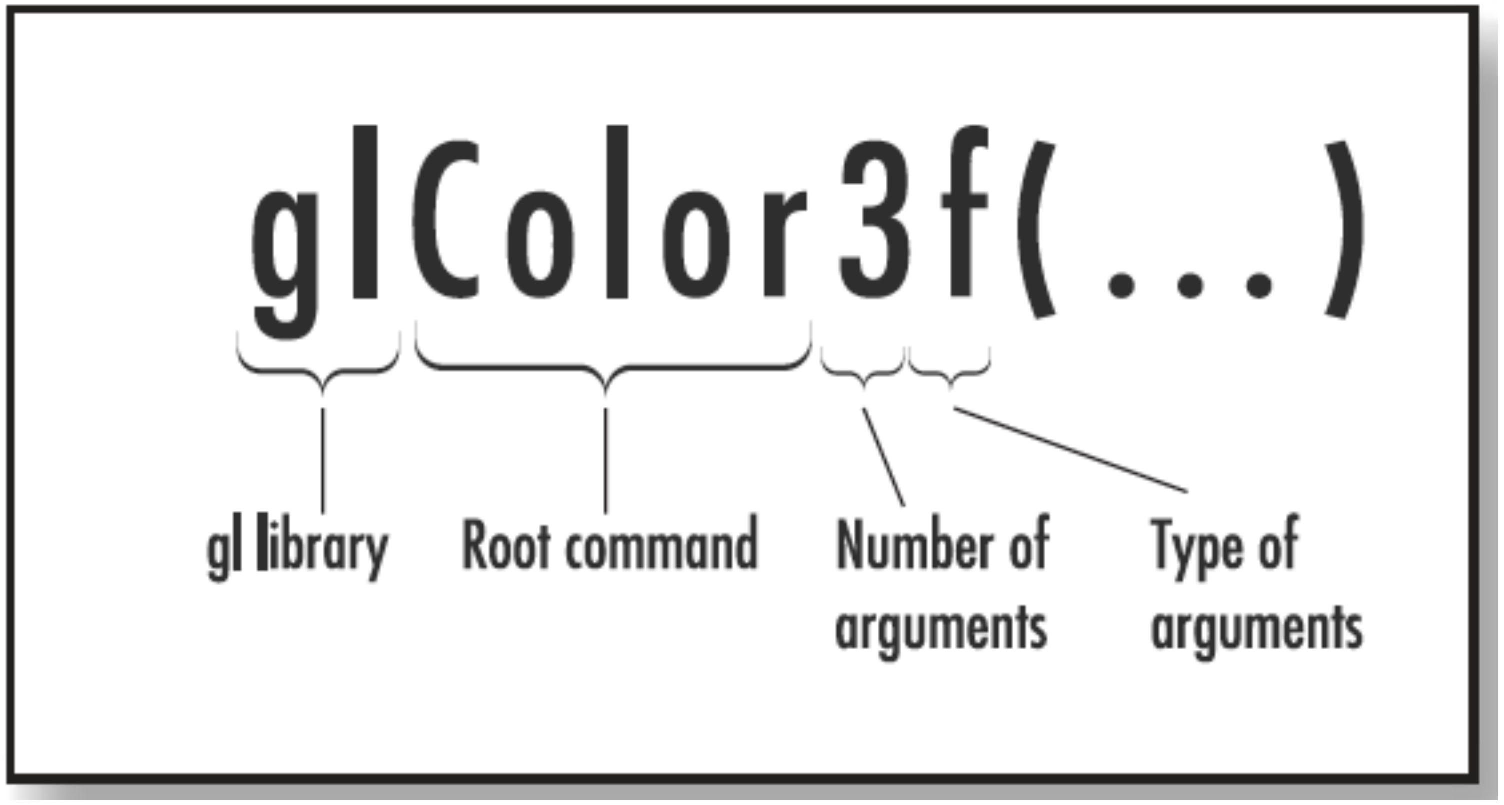
OpenGL Data Type	Internal Representation	Defined as C Type	C Literal Suffix
`GLbyte`	8-bit integer	`signed char`	`b`
`GLshort`	16-bit integer	`short`	`s`
`GLint`, `GLsizei`	32-bit integer	`long`	`l`
`GLfloat`, `GLclampf`	32-bit floating point	`float`	`f`
`GLdouble`, `GLclampd`	64-bit floating point	`double`	`d`
`GLubyte`, `GLboolean`	8-bit unsigned integer	`unsigned char`	`ub`
`GLushort`	16-bit unsigned integer	`unsigned short`	`us`
`GLuint`, `GLenum`, `GLbitfield`	32-bit unsigned integer	`unsigned long`	`ui`
`GLchar`	8-bit character	`char`	None
`GLsizeiptr`, `GLintptr`	native pointer	`ptrdiff_t`	None

Denoting Data Types

- All data types start with a GL to denote OpenGL.
- Most are followed by their corresponding C data types (byte,short,int,float, and so on).
- Some have a u first to denote an unsigned data type, such as ubyte to denote an unsigned byte.
- For some uses, a more descriptive name is given, such as size to denote a value of length or depth.
- The clamp designation is a hint that the value is expected to be “clamped” to the range 0.0–1.0. T

Function-Naming Conventions

- OpenGL functions follow a naming convention that tells you which library the function is from and how many and what types of arguments the function takes.



floats & doubles

- Any conformant C/C++ compiler will assume that any floating-point literal value is of type double unless explicitly told otherwise via the suffix mechanism.

- When you’re using literals for floating-point arguments, if you don’t specify that these arguments are of type float instead of double, many compilers will issue a warning while compiling because it detects that you are passing a double to a function defined to accept only floats,

- This results in a possible loss of precision, not to mention a costly runtime conversion from double to float.

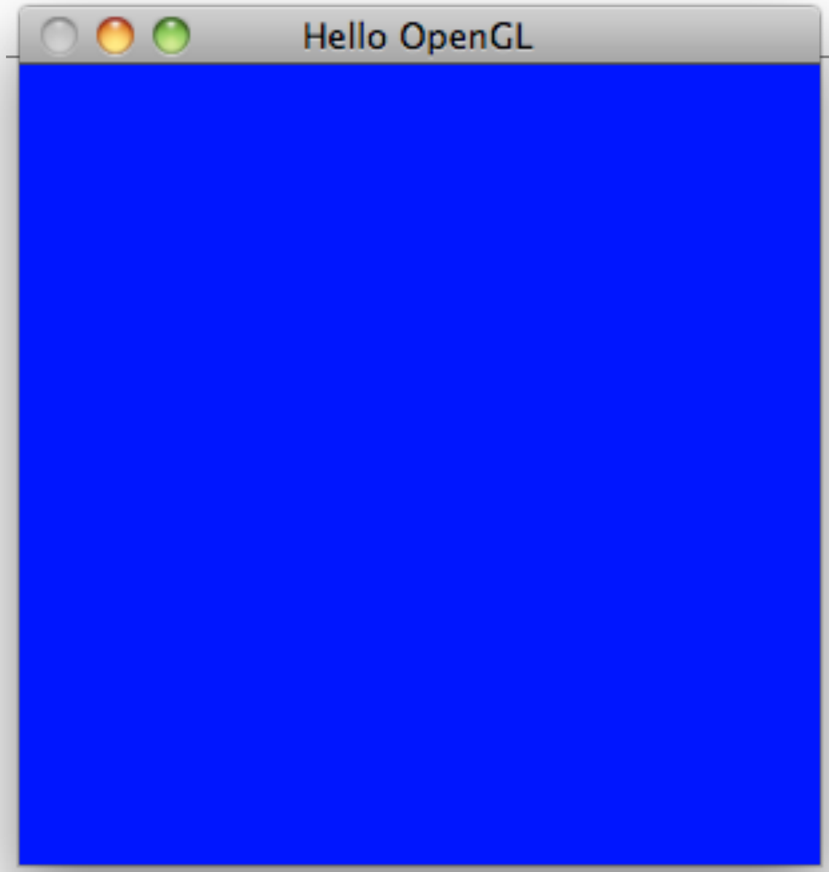
Platform Independence

- OpenGL is a powerful and sophisticated API for creating 3D graphics:
  - > 300 functions that cover everything from setting material colors and reflective properties to doing rotations and complex coordinate transformations.
- Does not address
  - to window or screen management.
  - keyboard input or mouse interaction.
- Primary goals was for OpenGL to be a platform independent abstraction of graphics hardware.
  - Creating and managing windows and polling for user input are inherently operating system related tasks

Using GLUT

- GLUT stands for OpenGL utility toolkit (not to be confused with the standard GLU—OpenGL utility library).
  - GLUT is widely available on most UNIX distributions (including Linux), and is natively supported by Mac OS X, where Apple maintains and extends the library.
  - On Windows, it has been replaced with Freeglut
- GLUT eliminates the need to know and understand basic GUI programming on any specific platform

First Program



- This simple program contains four GLUT library functions (prefixed with glut) and three“real” OpenGL functions (prefixed with gl).

```
#include "libopengl.h"

void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glFlush();
}

void setupRC(void)
{
  glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
  glutCreateWindow("Hello OpenGL");
  glutDisplayFunc(renderScene);

  setupRC();

  glutMainLoop();

  return 0;
}
```

Header

```
#include "libopengl.h"
```

```
#ifdef WIN32
#include <windows.h>
#include <gl\glu.h>
#include "glut.h"
#endif

#ifdef __APPLE__
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#include <Glut/glut.h>
#endif
```

glutInit()

- Console-mode C and C++ programs always start execution with the function main.

- The first line of code in main is a call to glutInit, which simply passes along the command-line parameters and initializes the GLUT library.

```
//..

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  //...
}
```

glutInitDisplayMode()

```
//..

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
  //...
}
```

- The flags here tell it to use a single-buffered window (GLUT_SINGLE) and to use RGBA colormode (GLUT_RGBA).

- A single-buffered window means that all drawing commands are performed on the window displayed.

- An alternative is a double-buffered window, where the drawing commands are actually executed on an offscreen buffer and then quickly swapped into view on the window.

- RGBA color mode means that you specify colors by supplying separate intensities of red, green, blue, and alpha components.

glutCreateWindow()

```
//..

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
  glutCreateWindow("Hello OpenGL");
  //...
}
```

- Call to the GLUT library actually creates the window on the screen.
- The single argument to glutCreateWindow is the caption for the window's title bar.

glutDisplayFunc()

```
//..

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
  glutCreateWindow("Hello OpenGL");
  glutDisplayFunc(renderScene);
  //...
}
```

```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glFlush();
}
```

- Establishes the previously defined function renderScene as the display callback function.
- This means that GLUT calls the function pointed to here whenever the window needs to be drawn.
- This call occurs when the window is first displayed or when the window is resized or uncovered..
- This is the place to put OpenGL rendering function calls.

Set up Rendering Context()

```
//..

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
  glutCreateWindow("Hello OpenGL");
  glutDisplayFunc(renderScene);
  setupRC();
  //...
}
```

```
void setupRC(void)
{
  glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}
```

- Do any OpenGL initialization that should be performed before render-ing.
- Many of the OpenGL states need to be set only once and do not need to be reset every time you render a frame (a screen full of graphics).

glutMainLoop()

```
//..

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
  glutCreateWindow("Hello OpenGL");
  glutDisplayFunc(renderScene);
  setupRC();
  glutMainLoop();
  //...
}
```

- This function starts the GLUT framework running.
- glutMainLoop never returns after it is called until the main window is closed
- Needs to be called only once from an application.
- It processes all the operating system–specific messages,keystrokes, and so on until you terminate the program.

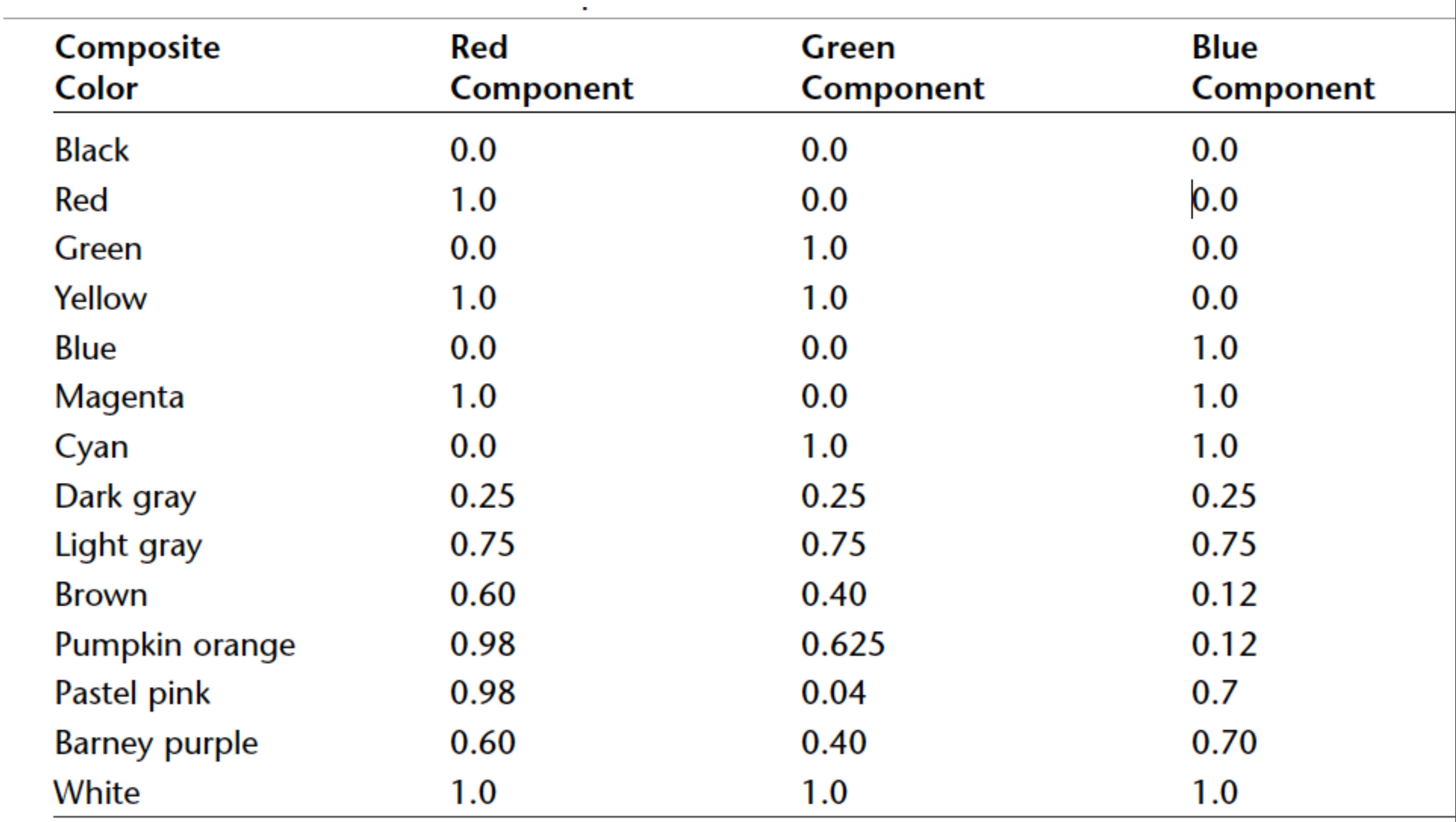
OpenGL Graphics Calls -glClearColor

```
void setupRC(void)
{
  glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}
```

```
void glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
```

- glClearColor sets the color used for clearing the window
- GLclampf is defined as a float.
- A single color is represented as a mixture of red, green, and blue components.
- The range for each component can vary from 0.0 to 1.0. thus yielding a virtually infinite number of potential colors.
- Practically speaking, color output is limited on most devices to 24 bits (16 million colors) total.
- OpenGL takes this color value and converts it internally to the nearest possible exact match with the available video hardware

Common Colour Values



Composite Color	Red Component	Green Component	Blue Component
Black	0.0	0.0	0.0
Red	1.0	0.0	0.0
Green	0.0	1.0	0.0
Yellow	1.0	1.0	0.0
Blue	0.0	0.0	1.0
Magenta	1.0	0.0	1.0
Cyan	0.0	1.0	1.0
Dark gray	0.25	0.25	0.25
Light gray	0.75	0.75	0.75
Brown	0.60	0.40	0.12
Pumpkin orange	0.98	0.625	0.12
Pastel pink	0.98	0.04	0.7
Barney purple	0.60	0.40	0.70
White	1.0	1.0	1.0

aplha parameter

```
void glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
```

- The last argument to glClearColor is the alpha component, which is used for blending and special effects such as transparency.
- Transparency refers to an object's capability to allow light to pass through it.
- E.g. Suppose you would like to create a piece of red stained glass, and a blue light happens to be shining behind it. The blue light affects the appearance of the red in the glass (blue + red = purple).
- You can use the alpha component value to generate a red color that is semitransparent so that it works like a sheet of glass—an object behind it shows through.

Clearing the Color Buffer

```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glFlush();
}
```

- A buffer is a storage area for image information. glClearfunction clears a particular buffer or combination of buffers.
- The red, green, and blue components of a drawing are usually collectively referred to as the color buffer or pixel buffer.
- The Colour buffer is the place where the displayed image is stored internally and that clearing the buffer with glClear removes the last drawing from the window. Y
- More than one kind of buffer (color, depth, stencil, and accumulation) is available inOpenGL
- The term framebuffer refers to all these buffers collectively since they work in tandem.

glFlush()

```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glFlush();
}
```

- Causes any unexecuted OpenGL commands to be executed.
- OpenGL uses a rendering pipeline that processes commands sequentially.
- OpenGL commands and statements often are queued up until the OpenGL driver processes several “commands” at once.
- This design improves performance because communication with hardware is inherently slow.
- Making one trip to the hardware with a comprehensive dataset is much faster than making several smaller trips for each command or instruction.