

CS 314: Preliminaries, Image Basics

Robert Bridson

September 2, 2008

1 Preliminaries

The course web-page is at: <http://www.ugrad.cs.ubc.ca/~cs314>. Details on the instructor, teach assistants, lectures, office hours, labs, assignments and more will be available there.

Computer graphics is a topic defined by what is done, not how: loosely speaking, graphics encompasses everything involved in a computer producing visual output. Some people in graphics even blur the boundaries and work with other sensory output, such as sound or touch. To make things work, graphics can involve a multitude of other disciplines: hardware design (from circuits to robots and more), psychology, data structures, computational physics, geometry, photography, art history, machine learning, networking, computer vision, optics, scientific computing, anatomy, architecture, and much, much more. This course necessarily isn't going to delve deeply into any one aspect, and will ignore many, but will try to cover what many consider the core—or at least, a good representative slice—of the field: the **three-dimensional rendering pipeline**. That is, how to go from a somewhat abstract description of a three-dimensional scene to an image on the screen, preferable quickly, and preferably producing a beautiful, realistic image.

This pipeline is of critical importance to many applications you probably are already aware of: visual effects in films, 3D computer games, scientific visualization of experimental measurements. However, much of what we will study also plays an important foundational role in other graphics applications, such as computer-aided design (CAD), abstract information visualization, font rendering and other core two-dimensional operations in use all the time in modern user interfaces, animation, image processing and more. There are two follow-on undergraduate courses offered at UBC that get deeper into graphics topics, offered in alternate years: 424 (offered this academic year) on geometric modeling, and 426 (offered next academic year) on animation and advanced rendering. For the truly intrepid, there may be possibilities for joining graduate-level courses in the department on further advanced topics.

Graphics as a whole, this course included, has become a fairly mathematical subject. Concepts and methods from linear algebra play a prominent role in particular; if you're not comfortable with the following, now is the time to review (Chapter 2, "Miscellaneous Math" in the textbook by Peter Shirley is a great place to read up on these):

- vectors and matrices
- vector operations: addition, scaling, dot products, cross products in 3D, ...
- matrix operations: addition, scaling, matrix-vector multiplication, matrix-matrix multiplication, determinants, inverting 2×2 and 3×3 matrices, ...
- basis vectors: linear independence, orthogonality, orthonormal bases, changing bases, ...

Some calculus will be needed too: derivatives, integrals, gradients (partial derivatives). Geometry and a few other mathematical topics are just as important, but there we'll cover most of what we need as we go. Many of the bugs that graphics programmers run into tend to be "math bugs"; finding, analyzing and fixing them will require geometric intuition and comfort with linear algebra.

Much of the programming work in this course, though not all of it, will involve a particular API named **OpenGL**. This is the cross-platform standard for fast, typically hardware-accelerated, 2D and 3D graphics. Later assignments and the final project will involve a fair amount of OpenGL programming. However, the API itself will not be taught in lectures: it's expected that you will mostly learn it on your own, with help from the "red book" reference text, labs, office hours, the web, and each other. Lecture time will be spent on core concepts and underlying algorithms, studying OpenGL from a high level along the way, alongside other rendering approaches.

2 Image Basics

With that out of the way, let's take a look at the output we will be dealing with for the rest of the course, images. You probably already have a good idea of what this is, say from experience with digital cameras: a 2D array of **pixels**, each of which stores a colour.

What is a colour? This isn't an entirely easy question to answer, and one we will come back to in more detail later in the course. From a purely physical standpoint, colour could be characterized as a total description of how much energy is in each frequency band in the visible electromagnetic spectrum: light waves shorter than infra-red and longer than ultraviolet. However, humans are not capable of distinguishing all of this vast amount of information: our eyes boil it down essentially to three averaged

quantities, corresponding to the three types of **colour cones** in our retinas. Roughly speaking, one type of cone measures the amount of reddish-yellowish (long wavelength) photons coming into our eye, another cone measures the amount of greenish (medium wavelength) photons coming in, and another the amount of blue-ish (short wavelength) photons coming in. We can then characterize every possible human colour **perception** with just three numbers.¹ Thus the colours in images are often specified as **RGB** values: one number for the amount of **R**ed, one number for **G**reen, and one number for **B**lue. This of course corresponds to most display technology, such as LCD panels and CRT screens, which have separate elements for red, green, and blue which can average together to fool the eye into seeing most colours.² Often people will refer to these as **colour channels**: our normal representation will have a red channel, a green channel, and a blue channel; each channel contains part of the information of the full image.

To summarize: at its simplest an image is a 2D array of pixels, each of which contains three numbers indicating RGB values. If the dimension of the image is m horizontally and n vertically, it will take $3mn$ numbers, which we commonly lay out sequentially in memory. There are of course many different orders to lay out all of these numbers; the convenient standard we will follow for the most part in this course (though not necessarily the best from a performance perspective) is to store the image as follows. Each horizontal **scanline** (i.e. a horizontal slice of pixels through the image, the set of all pixels with a given height) is stored one after the other, with the bottom-most scanline stored first, to which we will give the y coordinate 0. Scanline 1 follows, then scanline 2, and so on up to scanline $n - 1$. Inside each scanline the pixels are stored starting from x coordinate 0 on the far left, going up to $m - 1$ on the far right. Inside each pixel the red value is stored first, followed by the green and then the blue. We will typically use a 32-bit floating point value (usually `float` in C++) for each of these RGB values, though in hardware and in some image storage formats 8-bit integers or more exotic data types are commonly used. We'll assume that the usual range for each colour value is between 0 and 1, with 0 being as dark as possible and 1 as light as possible: the RGB value $(0, 0, 0)$ represents the darkest black the display can manage and the RGB value $(1, 1, 1)$ represents the brightest white the display can manage.³

It should be pointed out that in some contexts people flip the coordinate system vertically, so that $y = 0$ is at the top of the display and y increases as you go downwards; in this course (and indeed, with virtually all 3D computer graphics work) we will always take the bottom to be $y = 0$.

¹This isn't completely true: a poorly understood mutation gives some women a fourth type of cone, enabling them to see differences in colours nobody else can—search the web for “tetrachromacy” for more on this now if you're curious. Other species than humans can have different numbers of cones too, and perceive colours completely differently.

²These displays, and in fact just about any display that's ever been built, can't quite produce all the colours our eyes can see, just most of them; we'll come back to this point later in the course.

³This assumption is very simplistic too, as it obviously is closely tied to a particular display's capability; topics such as **High Dynamic Range** and **Colour Matching** deal with being smarter about this.

While the details of a particular display usually aren't quite as simple as this, our basic mental model of an image is a rectangle with pixels centered on the integer lattice points from $(0, 0)$ to $(m - 1, n - 1)$. Each pixel conceptually covers a unit square: pixel (i, j) actually has a physical extent from $(i - 0.5, j - 0.5)$ to $(i + 0.5, j + 0.5)$, since of course real displays show light in a whole region of the display, not just at a single point. In the next topic, **rasterization**, it will be important to keep this in mind.