GEOMETRY FOR COMPUTER APPLICATIONS
COURSE NOTES
MATH 431- FALL 2022
UNIVERSITY OF MARYLAND

This course develops the theory of data types for computer applications. Specifically we develop algebraic data types for computer graphics, computer vision and robotics. If we take points, lines, lengths, angles, areas, etc. as the (extremely basic) building blocks of graphics, then an over arching theme is to discover ways in which to represent these geometric objects in a computer. Simultaneously, we aim to develop ways in which to represent in a computer the manner in which these geometric objects transform.

1. Imagine designing a video game!

Your avatar is flying a spaceship through a hazardous jungle populated by wild monsters, evil dinosaurs and poisonous plants, with dangerous objects zipping by. Throw in a few earthquakes, tsunamis, hurricanes and tornados, too, just for fun.

Date: September 10, 2024.

> Of course enemies are chasing you, shooting rockets and subjecting your craft to waves of treacherous force fields. In addition to steering your vehicle, you need to be able to change your viewpoints and perspectives in order to fully gauge your direction and speed. Your instruments need to sense all the awful perils your adversary has aimed at you. Your survival, and the lives of millions of other people, depends on being able to manipulate — reliably, quickly, and in real time — huge amounts of graphical data by many types of geometric transformations: rotations, dilations, translations, reflections, and changes of perspective.

2. Goals

- Due to total size of the graphical data, the data types must be compact and efficiently designed.
- Due to the demands of interactive use, the computations must be as fast as possible.
- Due to the demands of ever-changing technology, the code must be easy to debug, maintain, and update. Thus the data types and the manipulation routines must be readable, succinct and comprehensible to other programmers.
- The data will ultimately be vectors and matrices, and the mathematical routines basically linear algebra. Matrix operations are cheap, efficient and easy to implement. The compelling advantage of linear transformations is that —by using coordinates on a vector space defined by a basis — the geometric information is encoded in a *finite set* of numbers. It's only how they are manipulated which varies by their context.

Here are some examples of how abstraction and mathematical elegance are both means and end in regards to computer applications.

3. Data types in plane geometry

First consider the familiar case is that of points in the plane. Points are described uniquely by an ordered pair of numbers. Vector operations enable us to compute geometric relations (such as distance) between points. Furthermore transformations of the plane are conveniently described by matrices. The calculations are cheap to implement on a computer, easy to understand.

Trying to do the same for lines in the plane is more difficult and more interesting. However, programming a video game may require you to

transform lines and, eventually, more complicated graphical objects, in a similar way. The reality is that lines in the plane are not as easy to parametrize as points in the plane: the set of lines does not admit a coordinate system as easy as just the (x, y)-coordinates which uniquely describe arbitrary points.

Here are some ways we describe lines in the plane. As you can see, none of them are as convenient as parametrizing points in the plane.

(1) Given two distinct points p_1 and p_2 (represented as 2-vectors), the line $\overleftrightarrow{p_1p_2}$ joining them is described by equations

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

or, in parametric form,

$$x = x_1 + t(x_2 - x_1)$$
$$y = y_1 + t(y_2 - y_1)$$

This parametrization has the following drawbacks:

- The points p_1, p_2 are not unique and may not be so efficient to find; there are many pairs of points which determine a given line.
- Determining when two different pairs (p_1, p_2) determine the same line may be unnecesarily time-consuming.
- The initial data requires that $p_1 \neq p_2$. Checking this each time is necessary (and time-consuming).

(2) Given a slope $m \in \mathbb{R}$ and $b \in \mathbb{R}$, the line with slope m and y-intercept b has *slope-intercept form*

$$y = mx + b.$$

This efficiently parametrizes all *non-vertical* lines uniquely by the pair $(m, b) \in \mathbb{R}^2$. However, vertical lines have "infinite slope" ($m = \infty$) and don't fit nicely into this parametrization. However they are parametrized by the x-intercept $a \in \mathbb{R}$: the vertical line corresponding to $a \in \mathbb{R}$ is given by $x = a$.

(3) One can remove (or, more accurately, "hide") the difficulty with infinite slope by replacing the slope m by the *angle of inclination,* that is, the angle θ the line makes with the x-axis. These two parameters relate by:

$$m = \tan(\theta)$$

However, like all angles, θ is only defined up to multiples of π. One can restrict θ to lie in an interval, say, $0 \leq \theta < \pi$, but this line does not vary continuously as $\theta \nearrow \pi$.

(4) Lines may also be parametrized by their *closest-points* as follows. A line L contains a unique point p which is closest to the origin O. If $p \neq O$, then this point determines L uniquely. However, the case $p = O$ (that is, when $L \ni O$) has to be handled separately, like the vertical lines in the slope-intercept parametrization.

3.1. **Topology is the villain.** The problem cannot be solved easily, since it reflects a fundamental fact, involving the *topology* of the set of lines in the plane. Unlike the points in the plane, which form a tractable algebraic object (a *vector space*), the lines in the plane form a space which is inherently more complicated. "Topology" refers to how the elements of the set are organized, and even for simple familiar objects, the topologies are very subtle.

Angles are illustrative of this phenomenon. Since the set of angles "closes up" —- when you go around a full 360^o — the set cannot be identified with a set of numbers or vectors in a completely satisfactory way. One has to introduce special cases to handle exceptions, and there is no way to get around this.

Other situations, like sets of lines in the plane or 3-space, lead to even more complicated topologies. For these two cases, the set is *nonorientable,* like a Möbius band, and this is particularly difficult to coordinatize.

Projective geometry began by the efforts of Renaissance architects and artists to deal with perspective. *Projective space* enlarges our usual space by introducing new points (called *ideal points*) which is where parallel lines eventually meet. (Imagine an aerial view of railroad tracks converging in the horizon.)

Projective space also has a very complicated topology; for example, the projective plane is nonorientable. It enjoys a set of *homogeneous coordinates,* which are only unique up to scaling — and to do calculations in projective geometry one has to work only in pieces of the space which are manageable and do admit vector coordinates. The geometric calculations all reduce to matrix operations in linear algebra, but to specify an arbitrary point in two-dimensional projective space, one needs *three* coordinates.

3.2. **Transformations and data types.** Certain types of transformations work better in certain coordinate systems. For example, polar coordinates behave very well under rotations, but they can be extraordinarily awkward in others. Try writing down the expression for a translation in polar coordinates and compare it to the expression in rectilinear coordinates.

Transformations preserving some special geometric properties may be suitable for special data types, which can by more succinct and more efficient. For example, angle-preserving transformations can be written very elegantly in terms of complex numbers: if $z \in \mathbb{C}$ is a complex number, then the *affine transformation* $z \longmapsto \lambda z + \tau$, where $\lambda \in \mathbb{C} \setminus \{0\}$ and $\tau \in \mathbb{C}$ is the most general orientation-preserving transformation which preserves angles. Whereas general affine transformations of the plane need six numbers, angle-preserving transformations depend only on two *complex numbers,* which is equivalent to four (real) numbers. This represents a significant improvement in the storage of data.

In 3 dimensions, rotations are more complicated, but they can be represented very elegantly using *quaternions,* the four-dimensional generalization of complex numbers. A linear rotation of $\mathbb{R}^3$ admits a very nice description in terms of quaternions, generalizing the remarkable formula

$$e^{i\theta} = \cos(\theta) + i \sin(\theta)$$

Quaternions are more complicated than complex numbers, largely due to the fact that $AB \neq BA$ in general. However, they are easily implemented in terms of standard vector operations. Discovered in 1843 by W. R. Hamilton (long before the advent of computer graphics) they are now a standard component of graphical hardware.

4. Overview

In this course we will discuss various data types for geometric objects and their transformations. After a review of linear algebra, vectors and matrices, we discuss Euclidean plane geometry via complex numbers. From that we discuss projective geometry (the geometry of perspective) in dimension two. From there we move to 3-dimensional Euclidean geometry, using quaternions to elegantly represent rigid motions (isometries) of Euclidean 3-space. Finally we bring all this together, giving parametrizations of points, lines and planes in projective 3-space.