The Math of Space

1 Overview

- A *space* is a set of *points*, typically characterized as a *tuple* $(t_1, t_2, \ldots, t_n)$ in terms of that space. We frequently speak about objects in space; such an object in a space S, can be viewed as a subset of points X such that $X \subseteq S$.
- Some examples of spaces are:
  - $\mathbb{R}^2$ is a space and $\{(x, y) \mid x^2 + y^2 = 25\}$ is an object in that space.
  - $\mathbb{R}^3$ is a space and $\{(x, y, z) \mid x^2 + y^2 + z^2 \leq 6\}$ is an object in that space.
  - $\hat{\mathbb{C}}$ is a space and
    $$F(f) = \{z \mid \langle f^n(z) \rangle_{n=0}^{\infty} \text{ is bounded}\}$$
    is an object in that space. The particular one above is a fractal called the "filled Julia set" of f.
- In addition to pure geometry, we attach other attributes — color, texture, material, lighting — to these objects in order to represent a complete scene.
- A *Euclidean space* is a space $\mathbb{R}^n$ where $n \in \mathbb{N}, n \geq 0$. n is the *dimension* of the space. 2D graphics typically work in $\mathbb{R}^2$ and 3D works in $\mathbb{R}^3$. By convention, for 2D, positive x goes to the right and positive y goes up. In 3D, we add a z axis to this setup; if positive z points "toward" us, we are using a *right-handed* convention, and if positive z points "away," then we are *left-handed* (see Figure 1).
- A lot of computer graphics involves translation *among* spaces, specifically "world space" and "device space." World space refers to the coordinate system used to represent the model, or the core data, of a 3D graphics application. Device space refers to the coordinate system of the destination medium: a window on the screen, a piece of paper. World space is typically $\mathbb{R}^3$ while device space is a finite integer subset of $\mathbb{R}^2$ with the positive y axis pointing "down," $x \geq 0$, and $y \geq 0$.

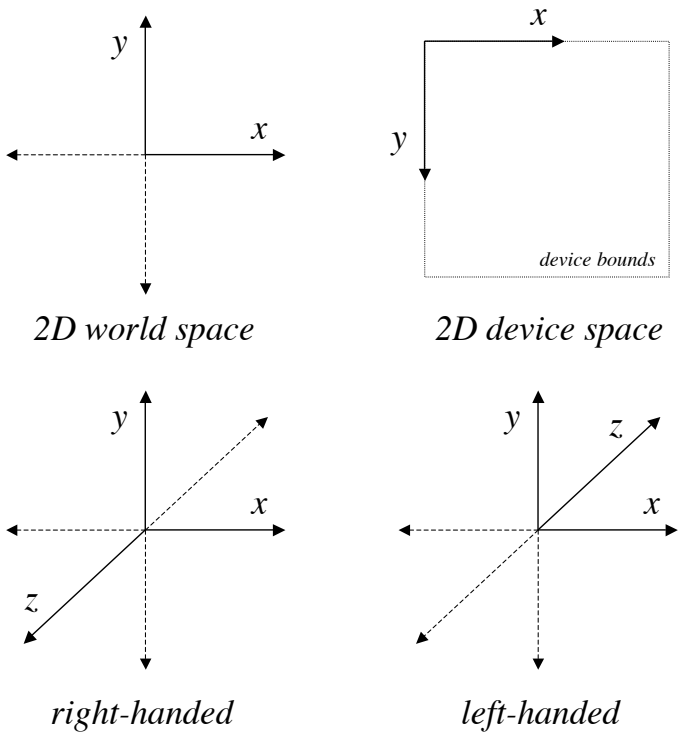

Figure 1: Conventions for representing Euclidean spaces: 2D world, 2D device, right-handed 3D, left-handed 3D. Solid axes indicate positive values.

2 Scalars

- A scalar can be thought of as a special case of a point in 1-dimensional space, but really that is overkill. A scalar is a single element of $\mathbb{R}$ that can engage in the standard arithmetic operations.
- When a set of scalars S is associated with the two operations *addition* and *multiplication* such that these operations have the properties of *closure*, *associativity*, *commutativity*, and *inverse*, then S is said to form a *scalar field*.
- Unfortunately, deeper discussion of fields and the related phenomena of groups and rings leads us to the realm of abstract algebra, which is (sob) beyond the scope of this class.

3 Points

- Points are pure locations; they have neither size nor shape.
- The distance δ between two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ in $\mathbb{R}^2$ is a scalar:

$$\delta(P, Q) = \sqrt{(x_Q - x_P)^2 + (y_Q - y_P)^2}$$

- The distance δ between two points $P = (x_P, y_P, z_P)$ and $Q = (x_Q, y_Q, z_Q)$ in $\mathbb{R}^3$ is a scalar:

$$\delta(P, Q) = \sqrt{(x_Q - x_P)^2 + (y_Q - y_P)^2 + (z_Q - z_P)^2}$$

  ... and so on for n dimensions.
- A frequently-used synonym for "point" is *vertex*, plural *vertices*. In particular, this is OpenGL's term of choice. But be careful — "vertex" sounds a lot like "vector."

4 Vectors

- Intuitively a *vector* is any quantity that has direction and magnitude. A vector does not represent a location in space — that is a point. However, vectors are represented in a very similar fashion, as an n-tuple where n is the dimension of the space we are using.
- In our notation, we distinguish between points and vectors by using parentheses () for points and angled brackets $\langle\rangle$ for vectors.
- Thus, a 2D vector u can be represented as an ordered pair $\langle x_u, y_u \rangle$. x_u represents the number of units along the x axis over which u "moves," and y_u does the same for the y axis.

- Similarly, 3D vector v can be represented as an ordered triple $\langle x_v, y_v, z_v \rangle$.
- The term *directed line segment* can be viewed as a synonym for a vector. But it's just too long to use regularly, so most of the time we'll stick with "vector."

4.1 Vector Operations

- The magnitude component of a vector can be represented by a scalar multiplier on the vector. Thus, if vector v has the same direction as u but is twice as long (i.e. has twice the magnitude), we can say:

$$2u = v$$

$$2\langle x_u, y_u, z_u \rangle = \langle 2x_u, 2y_u, 2z_u \rangle = \langle x_v, y_v, z_v \rangle$$

- If we connect vectors "head-to-tail" — that is, we "move" along multiple vectors in consecutive order — the resulting vector is the *sum* of the connected vectors. Thus, "going along u" then "going along v" in 3D translates mathematically into:

$$u + v = \langle x_u, y_u, z_u \rangle + \langle x_v, y_v, z_v \rangle = \langle x_u + x_v, y_u + y_v, z_u + z_v \rangle$$

  Figure 4.3 in the Angel textbook illustrates these concepts.

- A vector v can also be added to a point P, with the result leading to a new point Q. This *point-vector addition* is:

$$Q = P + v$$

$$(x_Q, y_Q, z_Q) = (x_P + x_v, y_P + y_v, z_P + z_v)$$

  (Figure 4.5 in Angel)

- Since the sum of a point and a vector is a point, then the *difference between two points* is a vector:

$$v = Q - P$$

$$\langle x_v, y_v, z_v \rangle = (x_Q - x_P, y_Q - y_P, z_Q - z_P)$$

- The *zero vector* (in any number of dimensions) is the a vector whose magnitude is 0. The direction of such a vector is undefined.
- Relative to vector addition, the zero vector is the *identity vector* — the vector I such that for any vector u, $u + I = u$.
- Given any vector u, the vector v such that $u + v = I$ (the zero vector) is the *additive inverse* of u, written as $-u$ (Figure 4.4 in Angel).

4.2 More Vector Definitions

- The magnitude component of a vector v is written as $|v|$.
- The magnitude of a scalar-vector product αv, $|\alpha v|$, equals the product of the scalar's absolute value and the original vector's magnitude:

$$|\alpha v| = |\alpha||v|$$

- A vector v whose magnitude is 1 — i.e. $|v| = 1$ — is a *unit vector*.
- The direction of a scalar-vector product αv is the same as the direction of the original vector v if $\alpha > 0$; it is the opposite of v if $\alpha < 0$.
- Every non-zero vector v has a corresponding unit vector U_v that has the same direction as v. U_v is easily derived from v, and this process is called the *normalization* of v:

$$U_v = \frac{v}{|v|}$$

- Given three points P, Q, and R, define three vectors, one of which is the sum of the other two relative to a specific point. For example, if R is viewed as the "destination point:"

$$u = Q - P$$
$$v = R - Q$$
$$w = R - P$$
$$u + v = w$$
$$(Q - P) + (R - Q) = R - P$$

  This is illustrated in Figure 4.9 in the Angel textbook.
- Two vectors u and v are *parallel* — $u \parallel v$ — if and only if $\exists\ \alpha \neq 0 \mid u = \alpha v$.
- Vectors $v_1 \ldots v_n$ are *linearly independent* if no vector v_k in that set can be written in terms of the others using scalar-vector addition.

4.3 The Dot Product

- The *dot product* of two vectors u and v — $u \cdot v$ — is a scalar defined as:

$$u \cdot v = x_u x_v + y_u y_v + z_u z_v$$

- The dot product is distributive over vector addition:

$$u \cdot (v + w) = (u \cdot v) + (u \cdot w)$$

  (you can prove that, right?)

- We use the dot product to quantitatively define the magnitude of a vector v ($|v|$):

$$|v| = \sqrt{v \cdot v} = \sqrt{x_v^2 + y_v^2 + z_v^2}$$

- The dot product is also the basis for determining the angle between two vectors:

$$\cos\theta = \frac{u \cdot v}{|u||v|} \tag{1}$$

(fun exercise: how is this derived?)

- Thus, it follows that two vectors u and v are *perpendicular* or *orthogonal* — $u \perp v$ — if and only if $u \cdot v = 0$.

- Also due to (1), the *orthogonal projection of u onto v* (as illustrated in Figure 4.14 of Angel) has magnitude:

$$|u| \cos\theta = \frac{u \cdot v}{|v|} \tag{2}$$

- In vector form, the projection of u onto v is just v with the magnitude calculated in (2). Thus, this vector is:

$$\left(\frac{u \cdot v}{|v|} \right) U_v = \left(u \cdot \frac{v}{|v|} \right) U_v = (u \cdot U_v)\, U_v$$

Are you loving this stuff yet?

4.4 The Cross Product

- The *cross product* of two three-dimensional vectors u and v — written as $u \times v$ — yields a vector w that is perpendicular to both u and v, i.e. $u \cdot w = v \cdot w = 0$:

$$u \times v = \langle y_u z_v - z_u y_v, z_u x_v - x_u z_v, x_u y_v - y_u x_v \rangle$$

It's easy to mix the combinations up! A mnemonic is shown in Figure 2. Multiply down the dotted lines, then subtract the product from the opposing dotted line.

- The resulting perpendicular vector follows the convention of a right-handed coordinate system.

- Note that the cross product is not commutative! The vectors yielded by $u \times v$ and $v \times u$ are not the same — though of course, by definition, they are both perpendicular to u and v.

- A vector that is perpendicular to a given set of vectors is called a *normal* to those vectors. Yes, there is another potential point of confusion here — "normalization" means the derivation of a unit vector, and "normal" means a vector that is perpendicular to other vectors.

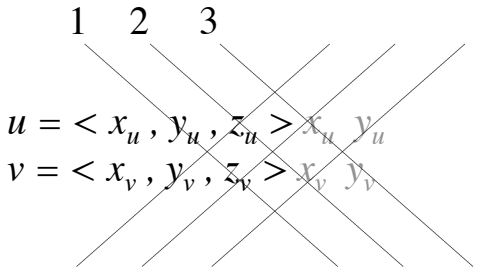

Figure 2: A visual way to remember the cross product of two 3D vectors u and v.

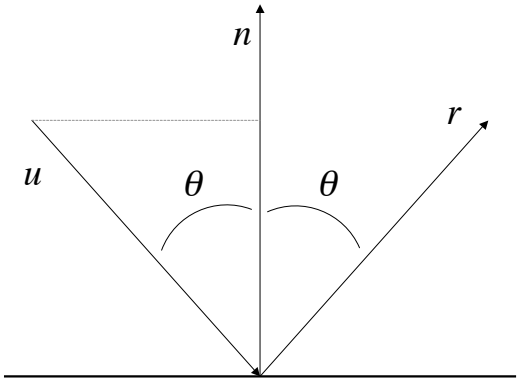

Figure 3: Reflection of u about a normal vector n.

- The *reflection* r of a vector u along some presumed normal n (see Figure 3) can be derived as follows:

$$u = r + \text{projection}(u, n) + \text{projection}(u, n)$$

$$r = u - 2\ \text{projection}(u, n)$$

$$r = u - 2\ (u \cdot U_n)\ U_n$$

  Typically, n is derived from a cross product of two other vectors that define the plane from which u is being reflected.

5 Lines

- We all know the drill: "Two points define a line." Let's state that mathematically, describing line L given points P and Q as a function of some scalar α:

$$L(\alpha) = P + \alpha(Q - P) \tag{3}$$

  Alternatively, this can also be written as:

$$L(\alpha) = \alpha P + (1 - \alpha)Q$$

  ... notice how P and Q "switch roles" in this format.

- (3) is known as the *parametric* form for describing a line, where α is the parameter in question. Notice how $L(0.0) = P$ and $L(1.0) = Q$. Also, $L(0.5)$ is the midpoint between P and Q. This is called *linear interpolation* — by traversing α from 0.0 to 1.0, you can calculate every point in the line segment defined by P and Q.[1]

- If we break P, Q, and L into their scalar coordinate components, then in 2 dimensions:

$$(\ x_L(\alpha), y_L(\alpha)\ ) = (\ x_P + \alpha(x_Q - x_P), y_P + \alpha(y_Q - y_P)\ ) \tag{4}$$

- Since we know that the difference between two points is a vector, then we can rewrite (3) and (4) as follows, replacing $Q - P$ with any vector v:

$$L(\alpha) = P + \alpha v$$

$$(\ x_L(\alpha), y_L(\alpha)\ ) = (\ x_P + \alpha x_v, y_P + \alpha y_v\ )$$

  This is useful in cases when you don't want some reference point Q to determine your line, but a pure vector v (such as a velocity, for example).

- Two lines L_1 and L_2 are *parallel* if $\exists\ P_1 \in L_1,\ \exists\ P_2 \in L_2,\ \exists\ Q_1 \in L_1,\ \exists\ Q_2 \in L_2,\ P_1 \neq Q_1,\ P_2 \neq Q_2 \mid (P_1 - Q_1) \parallel (P_2 - Q_2)$. Since we have defined parallel vectors, that latter condition needs no further exposition.

- Two lines L_1 and L_2 are *perpendicular* if, for the same P_1, P_2, Q_1, Q_2 above, $(P_1 - Q_1) \perp (P_2 - Q_2)$. Ditto with perpendicular vectors.

- The distance between a point P and a line L is defined as the distance between that P and the point Q on L such that P and Q define a line that is perpendicular to L.

[1] You know how some games have settings for "bilinear interpolation" and "trilinear interpolation?" Yes, those are related to (3).

6 Planes

- Just as "two points define a line," then "three non-collinear points define a plane." Mathematically, the plane F (for "face") that is defined by three non-collinear points P, Q, and R is a function of two parameters α and β such that:

$$F(\alpha, \beta) = P + \alpha(Q - P) + \beta(R - P) \tag{5}$$

- Again, as with lines, this can be rewritten in terms of a point P and two vectors u and v:

$$F(\alpha, \beta) = P + \alpha u + \beta v$$

- Representing a plane or face in terms of two vectors is useful because you can then easily calculate the *normal vector* n to the plane as the cross product of u and v. The normal vectors of planes or faces are crucial for determining the "front" and "back" of a face, and also for calculating how a face is lit.

- Note how, just as with lines, having $0.0 \leq \alpha, \beta \leq 1.0$ defines the triangle that is formed by P, Q, and R. You guessed it, this is *bilinear interpolation*.

- Again, analogous to points and lines, the distance between a point P and a plane F is defined as the distance between P and the point Q on F such that P and Q fall along the normal vector n of F. This is useful when figuring out which mouse click(s) hit which objects in a 3D graphics program!

7 Spatial Algebra API

The manipulation of points, vectors, scalars, and lines form a spatial algebra, which in turn can be expressed as a programming interface. This programming interface would consist of these types:

Boolean The usual true/false value.

Scalar A straightforward numerical type; floating-point and/or double-precision may sometimes be required.

Point An abstraction for a point.

Vector An abstraction for a vector.

Line An abstraction for a line.

Plane An abstraction for a plane.

Clearly, the specific programmatic definitions of Point, Vector, Line, and Plane depend on the number of dimensions of the space for which this interface is being implemented.

Given our basic types, the following functions or methods can be derived (and there can very well be more):

```
constructor: Point(Scalar*)
constructor: Vector(Scalar*)
constructor: Vector(Point, Point)
constructor: Line(Point, Point)
constructor: Line(Point, Vector)
constructor: Plane(Point, Point, Point)
constructor: Plane(Point, Vector)

distance(Point, Point) : Scalar
distance(Point, Line) : Scalar
distance(Point, Plane) : Scalar
subtract(Point, Point) : Vector
add(Point, Vector) : Point
add(Vector, Vector) : Vector
dotProduct(Vector, Vector) : Scalar
crossProduct(Vector, Vector) : Vector
times(Scalar, Vector) : Vector
negate(Vector) : Vector
unit(Vector) : Vector
isPerpendicular(Vector, Vector) : Boolean
isParallel(Vector, Vector) : Boolean
cosine(Vector, Vector) : Scalar
projection(Vector, Vector) : Vector
reflection(Vector, Vector) : Vector
reflection(Vector, Plane) : Vector
magnitude(Vector) : Scalar
```

You will likely want some subset of this "spatial algebra toolkit" for your graphics work from here on. If your language of choice supports generics, you might even be able to implement this API for an arbitrary number of dimensions!

8 Why Should You Care?

Why indeed...

- Virtually all interactive graphics programs ultimately boil down to some manipulation of points, vectors, lines, and planes. It pays to have a solid theoretical understanding of these entities and their behavior when figuring out how to make your airplane crash, how to make your ball bounce of a wall, or deciding if a rocket will hit a target.

- A computer scientist's view of computer graphics involves not only how to use a graphics API like OpenGL, but also understanding how to build one. The math of space is fundamental to doing this well, both creatively and correctly.

- Mathematical entities, once defined, analyzed, and proven, are forever true and consistent. While programmers may have learned Pascal in one decade then Java in the next, the behavior of mathematical entities is eternal and unchanging. You may not know what language you'll be using ten years from now, but you can be sure that you'll still be manipulating points, vectors, lines, and planes in the same way then as we do now.