CMSC 425: Lecture 5
More on Geometry and Geometric Programming

More Geometric Programming: In this lecture we continue the discussion of basic geometric programming from the previous lecture. We will discuss the cross-product, orientation testing, and homogeneous coordinates.

Cross Product: The cross product is an important vector operation in 3-space. You are given two vectors and you want to find a third vector that is orthogonal to these two. This is handy in constructing coordinate frames with orthogonal bases. There is a nice operator in 3-space, which does this for us, called the *cross product*.

The cross product is usually defined in standard linear 3-space (since it applies to vectors, not points). So we will ignore the homogeneous coordinate here. Given two vectors in 3-space, $\vec{u}$ and $\vec{v}$, their *cross product* is defined as follows (see Fig. 1(a)):

$$\vec{u} \times \vec{v} = \begin{pmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{pmatrix}.$$

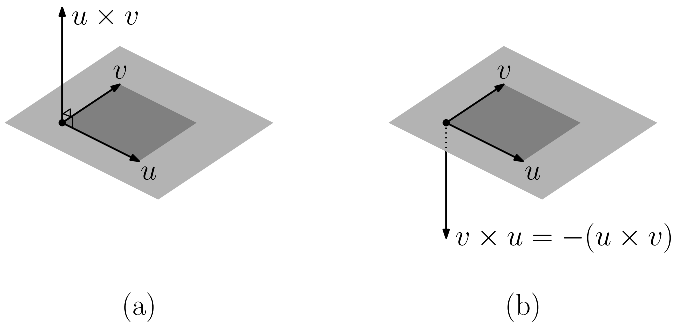


Fig. 1: Cross product.

A nice mnemonic device for remembering this formula, is to express it in terms of the following symbolic determinant:

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{e}_x & \vec{e}_y & \vec{e}_z \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}.$$

Here $\vec{e}_x$, $\vec{e}_y$, and $\vec{e}_z$ are the three coordinate unit vectors for the standard basis. Note that the cross product is only defined for a pair of free vectors and only in 3-space. Furthermore, we ignore the homogeneous coordinate here. The cross product has the following important properties:

Skew symmetric: $\vec{u} \times \vec{v} = -(\vec{v} \times \vec{u})$ (see Fig. 3(b)). It follows immediately that $\vec{u} \times \vec{u} = 0$ (since it is equal to its own negation).

Nonassociative: Unlike most other products that arise in algebra, the cross product is *not* associative. That is

$$(\vec{u} \times \vec{v}) \times \vec{w} \neq \vec{u} \times (\vec{v} \times \vec{w}).$$

Bilinear: The cross product is linear in both arguments. For example:

$$\begin{aligned}\vec{u} \times (\alpha\vec{v}) &= \alpha(\vec{u} \times \vec{v}),\\ \vec{u} \times (\vec{v} + \vec{w}) &= (\vec{u} \times \vec{v}) + (\vec{u} \times \vec{w}).\end{aligned}$$

Perpendicular: If $\vec{u}$ and $\vec{v}$ are not linearly dependent, then $\vec{u} \times \vec{v}$ is perpendicular to $\vec{u}$ and $\vec{v}$, and is directed according the right-hand rule.

Angle and Area: The length of the cross product vector is related to the lengths of and angle between the vectors. In particular:

$$|\vec{u} \times \vec{v}| = |u||v| \sin\theta,$$

where θ is the angle between $\vec{u}$ and $\vec{v}$. The cross product is usually not used for computing angles because the dot product can be used to compute the cosine of the angle (in any dimension) and it can be computed more efficiently. This length is also equal to the area of the parallelogram whose sides are given by $\vec{u}$ and $\vec{v}$. This is often useful.

The cross product is commonly used in computer graphics for generating coordinate frames. Given two basis vectors for a frame, it is useful to generate a third vector that is orthogonal to the first two. The cross product does exactly this. It is also useful for generating surface normals. Given two tangent vectors for a surface, the cross product generate a vector that is normal to the surface.

Example–Tiny-planet frame: Inspired by tiny-planet photos (see Fig. 2(a)), let us consider how to construct a local coordinate system for a player object standing on the sphere. Let c denote the sphere's center point, and let p denote the point on the sphere where the player object is standing (see Fig. 2(b)). In order to indicate the direction in which the player is facing, a second point $q \neq p$ is given on the surface of the sphere. These two points define a great-circle on the sphere. The player's *up axis* $\vec{u}$ is directed along a ray from c through p, the *forward axis* $\vec{f}$ is tangent to the minor great-circle arc from p to q, and the *right axis* $\vec{r}$ is orthogonal to these two and is directed to the player's right (see Fig. 2(c)).

Question: Given c, p, and q, how can we construct the vectors $\vec{u}$, $\vec{f}$, and $\vec{r}$ of the player's local coordinate frame?

Answer: First, the player's up-vector $\vec{u}$ is just the normalization of the vector from c through p, that is

$$\vec{u} \leftarrow \text{normalize}(p - c) = \frac{p - c}{\|p - c\|}.$$

(Recall that the length of a vector $\vec{v}$ can be computed as $\|\vec{v}\| \leftarrow \sqrt{\vec{v} \cdot \vec{v}}$.)

Next, to compute the player's right-vector $\vec{r}$, we observe that it must be perpendicular to the equatorial plane containing p and q, or equivalently, it must be perpendicular to both of the up-vector $\vec{u}$ and $\vec{w} = q - c$. Using the standard right-handed cross product, we have

$$\vec{r} \leftarrow \text{normalize}(\vec{w} \times \vec{u}), \quad \text{where } \vec{w} \leftarrow q - c.$$

You might wonder why $\vec{r}$ is not coming out of c. Recall that these are *free* vectors, and hence they are not associated with any particular location in space. (Note that the

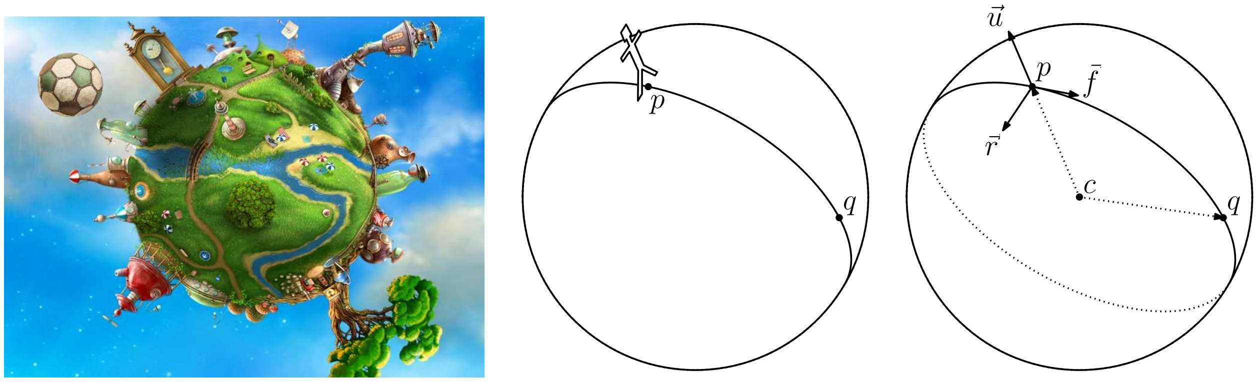


Fig. 2: Tiny-planet coordinate frame.

normalization step will fail if $\vec{w} \times \vec{u}$ is the zero vector. What must be true of p, c, and q for this to happen?)

Finally, to compute the forward vector $\vec{f}$, we observe that this vector is perpendicular to both $\vec{u}$ and $\vec{r}$. Given the relative orientations of the vectors, we have

$$\vec{f} \leftarrow \vec{u} \times \vec{r}.$$

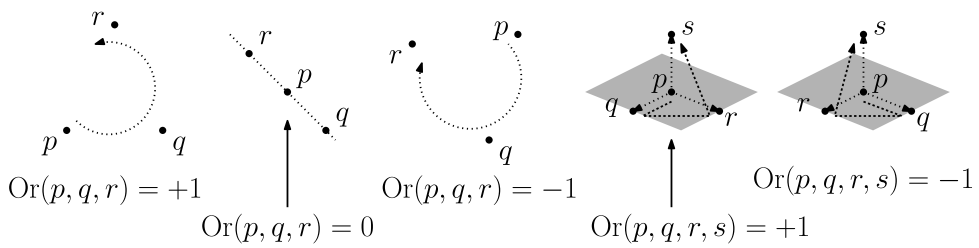
Note that there is no need to perform a normalization here. Since $\vec{u}$ and $\vec{r}$ are already of unit length and perpendicular to each other, it follows that the above cross product returns a vector of unit length.

Orientation: A fundamental concept in the design of numeric conditions is that of comparison. Given two numbers a and b, there are three possibilities: $a < b$, $a = b$, and $a > b$. Can we generalize the notion of comparison to multi-dimensional space?

We can think of a comparison as given by the sign of an *orientation function*, which takes on the values $+1$, 0, or -1 in each of these cases. That is, $\mathrm{Or}_1(p, q) = \mathrm{sign}(q - p)$, where $\mathrm{sign}(x)$ is either -1, 0, or $+1$ depending on whether x is negative, zero, or positive, respectively. What do we mean by the orientation of points in 2D or 3D (or higher dimensions)?

It turns out that the notion of orientation is well defined in all dimensions, but it is not a property of just two points. In general, it is the property of a sequence three points in 2D, four points in 3D, and generally $d + 1$ points in $\mathbb{R}^d$.

What does orientation mean intuitively? Given a three-point sequence $\langle p, q, r\rangle$ in the plane, its orientation is $+1$ if the triangle pqr is oriented counter-clockwise, -1 if clockwise, and 0 if all three points are collinear (see Fig. 3). In 3-space, a positive orientation means that the points follow a right-handed screw, if you visit the points in the order $\langle p, q, r, s\rangle$. A negative orientation means a left-handed screw and zero orientation means that the points are coplanar. Note that the order of the arguments is significant. The orientation of $\langle p, q, r\rangle$ is the negation of the orientation of $\langle p, r, q\rangle$.



r
p
q
$\mathrm{Or}(p,q,r) = +1$
r
p
q
$\mathrm{Or}(p,q,r) = 0$
r
p
q
$\mathrm{Or}(p,q,r) = -1$
s
p
q
r
$\mathrm{Or}(p,q,r,s) = +1$
s
p
r
q
$\mathrm{Or}(p,q,r,s) = -1$

Fig. 3: Orientations in 2 and 3 dimensions.

This intuition can be made mathematically formal. We define the *orientation* of a three-point sequence $\langle p,q,r \rangle$ in $\mathbb{R}^3$, denoted $\mathrm{Or}_2(p,q,r)$, and the orientation of a four-point sequence $\langle p,q,r,s \rangle$ in $\mathbb{R}^4$ as the signs of the following determinants:

$$\mathrm{Or}_2(p,q,r) = \text{sign det} \begin{pmatrix} 1 & 1 & 1 \\ p_x & q_x & r_x \\ p_y & q_y & r_y \end{pmatrix}, \qquad \mathrm{Or}_3(p,q,r,s) = \text{sign det} \begin{pmatrix} 1 & 1 & 1 & 1 \\ p_x & q_x & r_x & s_x \\ p_y & q_y & r_y & s_y \\ p_z & q_z & r_z & s_z \end{pmatrix}.$$

Recall from your linear algebra that swapping of any two columns of a determinant negates its sign, which implies that swapping any two points of the sequence negates the orientation.

Example–Bullet Shooting Query: Orientation testing is a very tool, but it is (surprisingly) not very widely known in the areas of computer game programming and computer graphics. Suppose that we have a bullet path, represented by a line segment $\overline{pq}$. Let $\triangle abc$ be a triangle of a mesh. We want to know whether the bullet hits the triangle (see Fig. 4).

Question: Does the (infinite) line passing through p and q intersect $\triangle abc$?

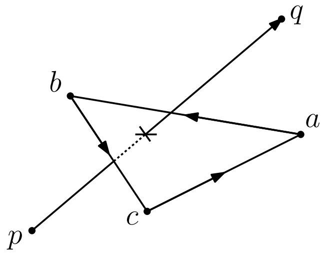


Fig. 4: Using orientation testing to determine line-triangle intersection.

Answer: We can determine this using three orientation tests. To see the connection, consider the three directed edges of the triangle $\overrightarrow{ab}$ $\overrightarrow{bc}$ and $\overrightarrow{ca}$. Suppose that we place an observer along each of these edges, facing the direction of the edge. If the line passes through the triangle, then all three observers will see the directed line $\overrightarrow{pq}$ passing in the same direction relative to their edge (see Fig. 4). (This might take a bit of time to convince yourself of this. To make it easier, imagine that the triangle is on the floor with a, b, and c given in counterclockwise order, and the line is vertical with p below the floor and q above. The line hits the triangle if and only if all three observers, when facing the direction of their respective edges, see the line on their left. If we reverse the roles of p and q, they will all see the line as being on their right. In any case, they all agree.)

It follows that the line passes through the triangle if and only if

$$\mathrm{Or}_3(p, q, a, b) \;=\; \mathrm{Or}_3(p, q, b, c) \;=\; \mathrm{Or}_3(p, q, c, a).$$

(By the way, this tests only whether the infinite line intersects the triangle. To determine whether the segment intersects the triangle, we should also check that p and q lie on opposite sides of the triangle. Can you see how to do this with two additional orientation tests?)

Local and Global Frames of Reference: Last time we introduced the basic elements of affine and Euclidean geometry: points and (free) vectors. However, as of yet we have no mechanism for representing these objects. Recall that points are to be thought of as locations in space and (free) vectors represent direction and magnitude, but are not tied down to a particular location in space. We seek a "frame of reference" from which to describe vectors and points. This is called a *coordinate frame.*

There is a *global coordinate frame* (also called the *world frame*) from which all geometric objects are described. It is convenient in geometric programming to define various *local frames* as well. For example, suppose we have a vehicle driving around a city. We might attach a local frame to this vehicle in order to describe the relative positions of objects and characters within the vehicle. The position of the vehicle itself is then described relative to the global frame. This raises the question of how to convert between the *local coordinates* used to define objects within the vehicle to their *global coordinates.*

Bases, Vectors, and Coordinates: The first question is how to represent points and vectors in affine space. We will begin by recalling how to do this in linear algebra, and generalize from there. We know from linear algebra that if we have 2-linearly independent vectors, $\vec{u}_0$ and $\vec{u}_1$ in 2-space, then we can represent *any* other vector in 2-space uniquely as a *linear combination* of these two vectors (see Fig. 5(a)):

$$\vec{v} = \alpha_0\vec{u}_0 + \alpha_1\vec{u}_1,$$

for some choice of scalars α_0, α_1.

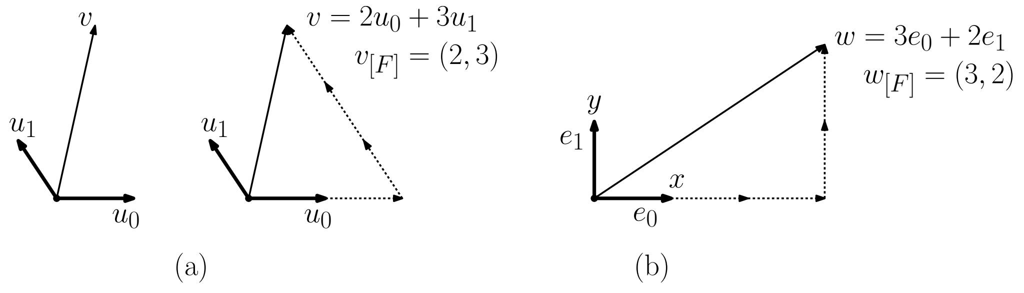


Fig. 5: Bases and linear combinations in linear algebra (a) and the standard basis (b).

Thus, given any such vectors, we can use them to represent any vector in terms of a pair of scalars (α_0, α_1). In general d linearly independent vectors in dimension d is called a *basis.*

The most convenient basis to work with consists of two vectors, each of unit length, that are orthogonal to each other. Such a collection of vectors is said to be *orthonormal*. The *standard basis* consisting of the x- and y-unit vectors is an example of such a basis (see Fig. 5(b)).

Note that we are using the term "vector" in two different senses here, one as a geometric entity and the other as a sequence of numbers, given in the form of a row or column. The first is the object of interest (i.e., the abstract data type, in computer science terminology), and the latter is a representation. As is common in object oriented programming, we should "think" in terms of the abstract object, even though in our programming we will have to get dirty and work with the representation itself.

Coordinate Frames and Coordinates: Now let us turn from linear algebra to affine geometry. Again, let us consider just 2-dimensional space. To define a coordinate frame for an affine space we would like to find some way to represent any object (point or vector) as a sequence of scalars. Thus, it seems natural to generalize the notion of a basis in linear algebra to define a basis in affine space. Note that free vectors alone are not enough to define a point (since we cannot define a point by any combination of vector operations). To specify position, we will designate an *arbitrary* point, denoted O, to serve as the *origin* of our coordinate frame. Let $\vec{u}_0$ and $\vec{u}_1$ be a pair of linearly independent vectors. We already know that we can represent any vector uniquely as a linear combination of these two basis vectors. We can represent any point p by adding a vector to O (in particular, the vector $p - O$). It follows that we can represent any point p in the following form:

$$p = \alpha_0\vec{u}_0 + \alpha_1\vec{u}_1 + O,$$

for some pair of scalars α_0 and α_1. This suggests the following definition.

Definition: A *coordinate frame* for a d-dimensional affine space consists of a point (which we will denote O), called the *origin* of the frame, and a set of d linearly independent *basis vectors*.

Given the above definition, we now have a convenient way to express both points and vectors. As with linear algebra, the most natural type of basis is orthonormal. Given an orthonormal basis consisting of origin O and unit vectors $\vec{e}_0$ and $\vec{e}_1$, we can express any point p and any vector $\vec{v}$ as:

$$p \;=\; \alpha_0 \cdot \vec{e}_0 + \alpha_1 \cdot \vec{e}_1 + O \qquad \text{and} \qquad \vec{v} \;=\; \beta_0 \cdot \vec{e}_0 + \beta_1 \cdot \vec{e}_1$$

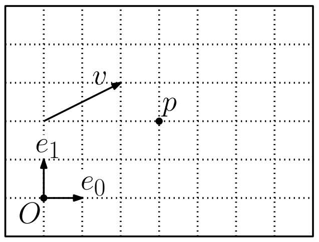
for scalars α_0, α_1, β_0, and β_1.

In order to convert this into a coordinate system, let us entertain the following "notational convention." Define $1 \cdot O = O$ and $0 \cdot O = \vec{0}$ (the zero vector). Note that these two expressions are blatantly illegal by the rules of affine geometry, but this convention makes it possible to express the above equations in a common (homogeneous) form (see Fig. 6):

$$p \;=\; \alpha_0 \cdot \vec{e}_0 + \alpha_1 \cdot \vec{e}_1 + 1 \cdot O \qquad \text{and} \qquad \vec{v} \;=\; \beta_0 \cdot \vec{e}_0 + \beta_1 \cdot \vec{e}_1 + 0 \cdot O.$$

This suggests a nice method for expressing both points and vectors using a common notation. For the given coordinate frame $F = (\vec{e}_0, \vec{e}_1, O)$ we can express the point p and the vector $\vec{v}$ as

$$p_{[F]} \;=\; (\alpha_0, \alpha_1, 1) \qquad \text{and} \qquad \vec{v}_{[F]} \;=\; (\beta_0, \beta_1, 0)$$



$$p = 3 \cdot \vec{e}_0 + 2 \cdot \vec{e}_1 + 1 \cdot O$$
$$\Rightarrow p_{[F]} = (3, 2, 1)$$

$$v = 2 \cdot \vec{e}_0 + 1 \cdot \vec{e}_1 + 0 \cdot O$$
$$\Rightarrow v_{[F]} = (2, 1, 0)$$

Fig. 6: Coordinate frames and (affine) homogeneous coordinates.

(see Fig. 6).

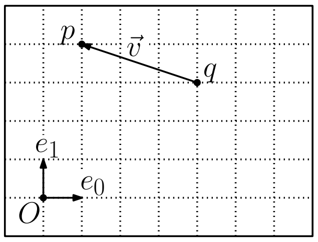
These are called *(affine) homogeneous coordinates.* In summary, to represent points and vectors in d-space, we will use coordinate vectors of length $d+1$. Points have a last coordinate[1] of 1, and vectors have a last coordinate of 0.

Properties of homogeneous coordinates: The choice of appending a 1 for points and a 0 for vectors may seem to be a rather arbitrary choice. Why not just reverse them or use some other scalar values? The reason is that this particular choice has a number of nice properties with respect to geometric operations.

For example, consider two points p and q whose coordinate representations relative to some frame F are $p_{[F]} = (1, 4, 1)$ and $q_{[F]} = (4, 3, 1)$, respectively (see Fig. 7). Consider the vector

$$\vec{v} = p - q.$$

If we apply the difference rule that we defined last time for points, and then convert this vector into it coordinates relative to frame F, we find that $\vec{v}_{[F]} = (-3, 1, 0)$. Thus, to compute the coordinates of $p - q$ we simply take the component-wise difference of the coordinate vectors for p and q. The 1-components nicely cancel out, to give a vector result.



$$p_{[F]} = (1, 4, 1)$$
$$q_{[F]} = (4, 3, 1)$$
$$\vec{v}_{[F]} = (1 - 4, 4 - 3, 1 - 1)$$
$$= (-3, 1, 0)$$

Fig. 7: Coordinate arithmetic.

In general, a nice feature of this representation is the last coordinate behaves exactly as it should. Let u and v be either points or vectors. After a number of operations of the forms $u + v$ or $u - v$ or αu (when applied to the coordinates) we have:

[1]Some conventions place the homogenizing coordinate first rather than last. There are actually good reasons for doing this, as we will see below in our discussion of orientation testing. But we will stick with standard engineering conventions and place it last.

- If the last coordinate is 1, then the result is a *point*.
- If the last coordinate is 0, then the result is a *vector*.
- Otherwise, this is not a legal affine operation.

This fact can be proved rigorously, but we won't worry about doing so.