# Geometry in 2D

**Jarek Rossignac**
www.gvu.gatech.edu/~jarek

# 1 - Introduction

This chapter reviews basic notions of 2D geometry. We start with the study of vectors and points. Then we discuss coordinate systems (frames) and transformations. Finally, we look at lines, edges, triangles, and circles.

## 1.1 Why are points and vectors important

Points and vectors are the fundamental primitives from which most of the representations and of the geometry processing techniques used in Geometric and Visual Computing (GVC) are constructed. Other geometric primitives are often defined and represented using points, vectors, and scalar values representing various measures (angles, distances). For example, a sphere may be represented by its center (point) and by a radius; a triangle is usually defined by its three vertices (points); a ray, traced by a photon in the absence of obstacles, is conveniently specified by a starting point (maybe the light source) and a travel direction (vector). Hence, much of the geometric processing performed in GVC deals with points and vectors.

- How would you use points, vectors, and scalars to represent an infinite cylinder in 3D?
- How would you use points, vectors, and scalars to represent the force acting on a shape?

## 1.2 Why are initially we assuming that the world is flat

Although much of modeling, animation, and graphic deals with three-dimensional (3D) objects and scenes, initially, we restrict our attention to **planar scenes**. Hence, our points, vectors, and various primitives derived from them (polygons, curves, circles, triangles…) will live in a plane (a page of these notes, the white board, or a projection screen). We do so because many of the constructions discussed in these notes are slightly easier to present in such a two-dimensional (2D) setting. Furthermore, numerous problems that one must solve in 3D are intrinsically two-dimensional. Nevertheless, we will later lift these 2D concepts into the third dimension… and even into four dimensions to discuss some representations and algorithms for 3D animation.

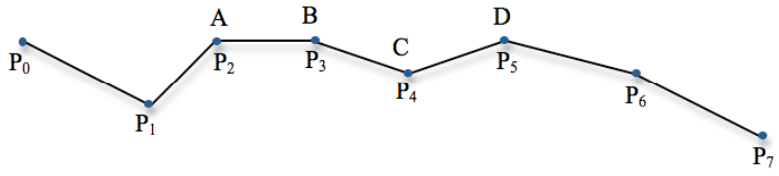
- Planar models are commonly used to represent 3D scenes. Can you think of an example?
- How would you use a 4D space to represent the motion of a point in 3D?

## 1.3 What is a point and how is it denoted and drawn in these notes

A **point** is in fact nothing but a specific **location** (in the plane, since we are initially working in 2D). Examples include the center of a disk, the end of a segment, and the corner of a triangle.

We typically use single letters to refer to points. When we discuss isolated points, we typically use uppercase letters, such as P or Q. When we talk about a point in the context of an infinite set of points, such as a curve, we may use a lower case letter for the point and an uppercase letter for the curve. For example, we may say: "For every point p in curve P…". When the points are ordered, such as the **vertices** of a **polygonal curve**, we typically use subscripts, such as $P_0$, $P_1$, $P_2$…, or $P_i$, $P_{i+1}$… to identify them. However, in these situations, to simplify notation by removing the need for subscripts, we may temporarily rename a short subsequence of such points and say for example: "Given 4 consecutive vertices {A,B,C,D} of polygon P…".

In our illustrations, we show points as dots (small disks) so as to distinguish them from the curves they lie on. Remember however that a point is not the disk, but the location of its center.



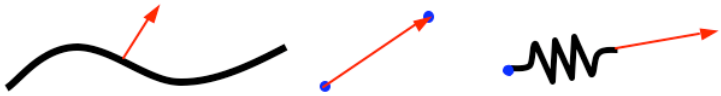
✎ Can you suggest another way of drawing points? Compare the dots used above with your suggestion and discuss their respective advantages and drawbacks.

## 1.4 What is a vector and how is it denoted and drawn in these notes

Most often, a **vector** represents a **direction**, **displacement**, **velocity**, **acceleration**.

We typically use uppercase letters, such as U, V, or W, to denote vectors. We may also use subscripts, such as in $L_1$, $L_2$…, to identify consecutive elements in an ordered list of vectors.

We draw vectors as **arrows** in the figures.



✎ List 3 instances where you have used vectors to represent different entities. For each one, explain exactly what the vector represented.

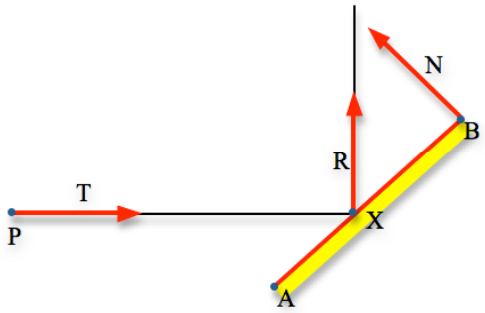
A given vector is defined by the direction and length of the arrow, but is not restricted to any particular location. Hence, a vector may be drawn anywhere on the page, or even replicated at several locations.

✎ Is the force pulling on the handle of a door a vector? Justify your answer.

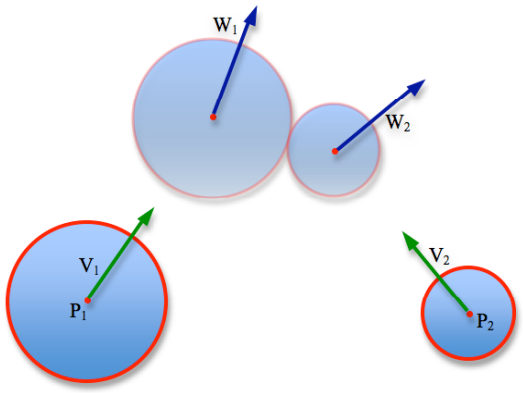
## 1.5 What kind of computations are points and vectors used for

Many algorithms in modeling, animation, and computer graphics take as input combinations of points and vectors and compute scalars, points, or vectors that represent a particular distance, time, location, or orientation. We illustrate these with a couple of classic examples.

*EXAMPLE 1*: Consider a ray from point P with tangent vector T and a flat wall (represented in 2D by an edge between points A and B). Compute the point X where the ray hits the wall and the direction R of the reflected ray, assuming a mirror reflection. This computation is performed in animation when simulating the collision of a ball with a wall. It is also used in photorealistic rendering when simulating the trajectories of photons as they leave the light source, hit surfaces, and bounce off into new directions. Hence, all practitioners in the broad GVC field must be able to quickly invent and implement efficient approaches for computing point X and vector R from the input parameters (P, T, A, and B). Typically, it is easier to present the solution as several construction steps, rather than a single formula. Hence, intermediate variables, such as the normal N defined as the unit vector orthogonal to the wall, are often introduced to simplify the derivation of the solution and its implementation.




*EXAMPLE 2*: Consider two disks one of radius $r_1$ and the other of radius $r_2$. At time t=0, the center of the first disk is at point $P_1$ and the center of the other disk is at point $P_2$. Assume that the first disk travels at constant velocity (vector) $V_1$ and that the second disk travels at constant velocity $V_2$. We use these points and vectors and test whether the two disks collide. If so, we compute the time $t^*$ of collision. We will also use them to compute the position of the two disks at $t^*$ and their respective velocities $W_1$ and $W_2$ after an elastic shock. This is a key component when performing simple animations involving disks in 2D or balls in 3D, such as for instance in a game of pool (billiard) or in molecular simulations.




✎ Suggest another geometric problem that may be defined and solved using points and vectors.

There are often several strategies for solving a typical problem of this kind. You should try to master a variety of them, since one or another may lead to a simpler or more efficient solution.

The **analytic approaches** strive to formulate a series of simultaneous equalities or inequalities and then to solve them to obtain the desired values. Typically, these equalities and inequalities involve ratios of polynomial expressions of scalar parameters, such as time, arc-length, and the coordinates of points and vectors. Although sometimes unavoidable, such formulations are often unnecessarily verbose, tend to obscure the intuitive construction that has lead to their development, and may require solving high-degree polynomials.

Instead, to take advantage of simple shortcuts that reduce labor and computational cost, we advocate **geometric formulations**, which manipulate vectors and points directly, rather than their coordinates. These formulations are based on construction involving the few operators defined below. The advantage of our approach is that the reader is able to really understand and retain a particular construction and to adapt it to a slightly different problem. The drawback is that readers not familiar with operators on points and vectors (such as additions and dot product) must learn them and practice using them.

✎ Consider two disks, one with center $C_1$ and radius $r_1$ and the other with center $C_2$ and radius $r_2$. Propose two solutions for testing whether they interfere. The first one should be formulated using points, distances between points, and scalar values (assume here that the function for computing the distance d(A,B) between point A and B is provided). The second one should be formulated as an algebraic inequality involving coordinates of points (A.x,A.y) and other variables ($r_1$…). Which of these solutions is easier to explain and retain?

### 1.6 How is this chapter organized and why

We first review **vectors**, their representations, their operators, and present few constructions to illustrate their use in graphics, modeling, and animation.

Then, in a separate subsection, we discuss **points**, their representations, their operators, and present a few constructions. This organization stresses the fundamental semantic **difference between vectors and points**.

Then, we discuss linear **transformations** and changes of **coordinate systems** (which we call frames). A **frame** may be specified by an origin (point) and a set of vectors (orthonormal basis) and changes of frames are used to transform points and vectors as a convenient tool for constructing and animating scenes.

Finally, we discuss a few simple geometric **primitives** (lines, edges), which may be represented in terms of points and vectors, and present the computations of their intersections, which are a fundamental tool for many algorithms in modeling (e.g. Boolean operations), animation (e.g. collision), and graphic (e.g. ray-tracing).

## 2 - Vectors

### 2.1 What is the norm of a vector

A vector has a **length**, also called its **norm** or its **magnitude**. The norm of vector V is denoted V.n, n(V) or ||V||. The reader must retain this redundant terminology, since all three terms are used in the literature. The notation n(V) is the most practical for describing an implementation. The abridged object-oriented notation V.n may be preferred by some. The notation ||V|| is often used in mathematical texts.

The norm is a measure. It is expressed in a chosen **unit** (for example inch or meter). Hence, V.n should in principle be qualified with the unit in which it is expressed. However, for convenience, we often assume some unspecified but agreed upon unit. Remember that it is acceptable, and often convenient to select the unit. For example, one may say "Without loss of generality, assume that the unit is chosen so that U.n==1." and then proceed with a given construction in which U.n may be replaced by 1 and hence omitted when it is multiplying another quantity.

A **unit vector** is a vector with norm 1. Many vectors used in graphics are unit. For example, the directions of rays tangent to curves, and the normal's to surfaces in 3D are unit vectors. Knowing that a vector is unit often leads to shortcuts in geometric constructions and in their implementations.

When the norm of a vector is zero, we say that it is the **null vector**, which we represent with a bold **0**. Throughout the remainder of this section and unless specified otherwise, assume that the vectors discussed are not null.

✎ What are the synonyms for the length of a vector V? What are the different notations for it?

✎ We have used above a constraint on the norm to identify two special classes of vectors. List their names and provide the constraint that defines each one of them.

## 2.2 What are the polar coordinates of a vector

A vector may be defined by its length and its "direction". In two dimensions, the "direction" of a vector may be conveniently represented by a signed angle from some reference direction, as discussed below. To avoid this dependency on the reference direction, and to simplify the definition of the "direction" of a vector in 3D, we simply say that a **direction** is a unit vector. As explained below, we use the term "angular coordinate" when referring to the angle of a direction. Hence, you must remember that the term "direction" refers to a unit vector and that the expression "**direction of vector V**" is the unit vector, denoted by the function call U(V), that is obtained by scaling V by 1/V.n, as explained below. We assume of course that V is not null. Hence the direction U(V) of a vector V is parallel to V but has unit length. Note that we use the letter U to remind us that this is the "unit-length" vector (or direction) and an upper-case letter 'U' to remind the reader that the result returned by U(V) is a vector, and not a scalar.

A direction may be used to represent the **tangent** to an oriented curve at a specific point along the curve. In two dimensions, the vector obtained by rotating such a tangent by 90 degrees counterclockwise is called the **normal** to the curve at that point. The noun "**normal**" typically refers to a unit vector (direction) that is perpendicular to some curve (or to some surface in 3D). Hence, we may say "Let N be **the** normal to curve C at point P". Of course, here we assume that such a vector is uniquely defined (i.e., that the curve is smooth at P). The adjective "**normal**" is synonymous to "orthogonal" and to "perpendicular". Hence we may say that "vector V is normal to vector U", or that "vector V is normal to curve C". This does not imply that V is a unit vector.

Remember that the term "**norm**" refers to the length (scalar measure) of a vector. It should not be confused with the term "normal".

A direction (vector) may also be used to represent the direction (orientation of the velocity) in which some hypothetical photon or point may travel.

✎ What is the difference between "normal" and "norm"?

✎ What is the direction of a vector V?

An arbitrary vector V is completely defined by its norm, n(V), and its direction, U(V). The norm is a length and hence may be represented by a scalar (assuming an agreed-upon unit for measuring lengths). Because we are working in two dimensions, the direction may be represented by an angle (denoted angle(V)), with respect to some agreed-upon **base direction** (often the horizontal of the page). The units in which that angle is represented is typically radians, but one may chose to work in degrees or turns and convert to radians (1 turn = 360 degrees = $2\pi$ radians).

n=n(V) is called the **radial coordinate** of vector V. a=angle(V) is called to **angular coordinate** of V. The pair of numbers (r,a) is called the **polar coordinates** of V. Note that (n,a) and (n,a+2k$\pi$), for any integer k, represent the same vector. Hence, for simplicity, we assume that angle(V), when expressed in radians, **is always in [–$\pi$,$\pi$].** Furthermore, n **is never negative**.

✎ The polar coordinates representation of a vector comprises two coordinates. Give their names. Explain what they represent. List the range of values. Discuss the units.

✎ Does the null vector have a unique representation in polar coordinates?

✎ What is the polar coordinates representation of a direction?

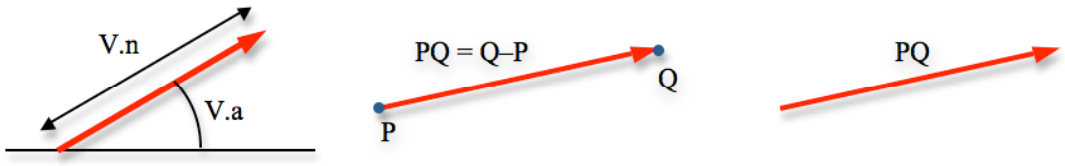
## 2.3 How does a vector represent a displacement

A vector may represent the **displacement** between two points. For example, consider points P and Q. One may define the location Q by saying "Start at P and then move by a displacement V, which has direction D and magnitude n." In other words, "Start at P, face direction D, and walk n units." Such a displacement from P to Q is often denoted PQ. Hence, when you see two letters together, where each one represents a point, the pair defines the vector displacement between them, from the first one, to the second one. Another popular notation for a displacement vector PQ is the difference Q–P between two points. The role of "–" in "Q–P" is at odds with the intuitive notion of the term "minus". What does it mean to subtract a point from another? Nevertheless, this notation is consistent with the rules of the + and – operators in vector and point expressions, because one may say that Q is the result of displacing P by vector PQ, and hence should be written Q=P+PQ. Hence, it may seem natural to move P to the left side of the equation while changing its sign and say that Q–P=PQ.

Also note that QP = – PQ.

Expressions involving vectors written in this two-point displacement style may be reordered and sometimes **simplified**. For example AB+CD–CB can be written AB+BC+CD by replacing –CB by BC and by using the fact that vector addition is commutative. Then, AB+BC+CD may be simplified to AD, since it defines the cumulative displacement from A to B, then from B to C, then from C to D. We started at A and ended at D. Hence, the total displacement is AD. The reader should practice such simplifications.

We draw vectors as **arrows** in our figures. Note that even though a vector may be defined as the displacement PQ, it may be **drawn anywhere** on the plane. Hence, the arrow does not need to start at P. As long as two arrows all have the same polar coordinates (length and angle), they represent the same vector.




✎ Draw an origin O and two points, A and B. Then draw vector AB so that it starts at the origin O.

✎ Explain why any given vector may be drawn anywhere on the page.

✎ What is PQ+QP? Justify your answer.

✎ Draw four arbitrary points A, B, C and D. Now compute the vector V=AB+AC+AD and draw it four times, each time starting at a different point in {A,B,C,D}.

## 2.4 What are the Cartesian coordinates of a vector

Pick two orthogonal directions (unit vectors) on the page. For instance, the *left-to-right horizontal* H and the *upward vertical* U. Remember that H and U are directions, and hence are unit vectors. The displacement defined by a vector V may always be achieved as a sequence of two axis-aligned displacements: one horizontal by an amount V.x and one vertical by an amount V.y. In fact, as discussed below, V = V.x H + V.y U.

These two are the Cartesian coordinates of vector V. We use the notation <V.x,V.y> to represent V by its Cartesian coordinates. It does not matter which axis-aligned displacement comes first (the cumulative displacement vector is the same whether we first move horizontally or first vertically). Nevertheless, by convention, we list first the coordinate V.x that measures the amount of horizontal displacement.

When V.x is positive, the horizontal displacement goes towards the right. When V.x is negative, it goes to the left. Similarly, when V.y is positive, the vertical displacement goes up. (Unless a different convention is chosen, which is the case in some 2D graphic systems, where a positive V.y components indicates a vertical displacement ***downwards***.)

We use parentheses, such as "(n,a)", to denote the polar coordinates of a vector and brackets, "< , >", to denote its Cartesian coordinates. Hence (n,a) denotes a vector V such that n(V)==n and angle(V)==a, while <x,y> denotes a vector V, such that V.x=x and V.y=y. (Of course, this notation (n,a) for a vector should not be confused with the same notation (x,y) for a point. Please use the context to disambiguate.)

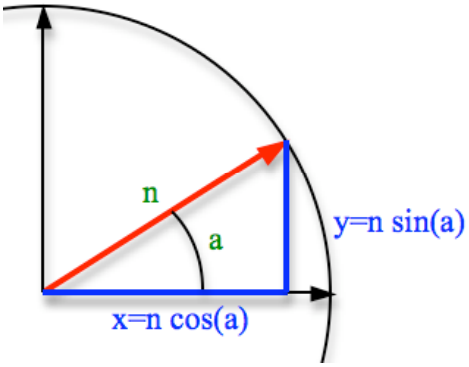
✎ Draw 2 different instances of vector <–2,3>.

✎ What are the different Cartesian representations of the null vector?

## 2.5 How do we convert between polar and Cartesian coordinates

We have studied two different representations of a vector. Some vector operations are simpler to perform in Cartesian coordinates, others in polar. Hence, you may need to switch between them to simplify constructions or calculations. Conversions between Cartesian and polar coordinates follow from the definition of trigonometric functions and from the Pythagorean theorem. For conciseness, assume V=(n,a)=<x,y>.

To convert from Cartesian to polar, use $n=\sqrt{(x^2+y^2)}$ and a=atan2(y,x), where the function atan2 returns an angle in [–π,π]. To convert from polar to Cartesian, use x = n cos(a) and y = n sin(a). You need to understand and remember these formulae.




✎ Let V=(n,a)=<x,y>. Assume x>0 and y>0. Draw V and indicate n, a, x, and y on your figure. Then, write down the four formulae for converting back and forth between polar and Cartesian coordinates. What is the range of angle a?

✎ Let V=(n,a)=<x,y>. Assume x<0 and y<0. Draw V and indicate n, a, x, and y on your figure. Then, write down the four formulae for converting back and forth between polar and Cartesian coordinates. What is the range of angle a? What is the sign of n?

✎ Convert <1,0>, <0,1>, <–1,0>, <1,1>, and <3,4> to polar coordinates.

✎ Convert (1,45°), (2,–45°) and (2,135°) to Cartesian coordinates.

## 2.6 Rotating vectors and measuring angles between them

It is trivial to **rotate** a vector that is expressed by its polar coordinates. The rotated version of V by angle b is the vector (n(V),angle(V)+b). We use the notation R(V,b) to denote the result of such a rotation. We use the mnemonic "R" for rotation and an upper case letter to indicate that we return a vector.

R(V) is a convenient short-cut for R(V,π/2). It returns the vector obtained by rotating V by 90 degrees counterclockwise. It is typically used to obtain a pair of orthogonal vectors for a frame (as discussed later) or for obtaining a normal to a curve, given a tangent.

We use a(U,V) to denote the **angle** from vector U to vector V. We take the convention that a(U,V)==0 when U(U)==U(V) and that a(U,V)>0 when U(V)=R(U(U),b) for some angle b in [0,π]. Hence angle(U,V) returns an angle in [–π,π[.

✎ Let V = (1,a). Write the pseudocode to compute R(V,b). Make sure that the resulting polar form is valid, i.e., that its angle is in [–π,π[.

✍ Assuming that U and V are in polar coordinates, write the pseudocode to compute angle(U,V). Make sure that the resulting angle is in [–π,π[.

⌨ Implement and test angle(U,V).

## 2.7 Comparing and scaling vectors

Two vectors U and V are **equal** when they are both null or when they have the same coordinates (using either polar or Cartesian coordinates). To express the test whether U and V are equal, we write U==V or equal(U,V). Note that, to follow a programming convention, we use "==" to denote logical equality and "=" to denote propositional equality ("A=B" means that A is always equal to B) or an assignment ("x=x+1;"). The context is used to decide whether "=" is a propositional equality and an assignment.

Let s be a positive real number (scalar). The **scalar-vector product** sV denotes the vector V scaled by s. When s==0, sV==**0**. When s is positive, sV is the vector with polar coordinates (s n(V),angle(V). **When s is negative**, sV is the vector with polar coordinates (–s n(V) , angle(V)+π). In other words, it is obtained by scaling the norm of V by the absolute value |s| of s and by rotating the vector 180 degrees (i.e., adding π to its angle and possibly subtracting 2π if necessary to ensure that angle(V) stays in [–π,π]).

Scaling a vector represented in Cartesian form is trivial: s<x,y> = <sx,sy>.

When we wish to **divide** the norm of a vector by s, as a short cut for (1/s)V, we can write V/s.

The vector with norm n(V) and an angle of angle(V)+π is the **opposite** of V and is denoted by –V. Hence, using Cartesian coordinates: –<x,y>=<–x,–y>. We use the call RR(V) to compute –V. The mnemonic is that we have rotated V twice by π/2.

For example, in the figure below, 2V is a vector of the same direction as V, but twice longer. The **opposite** vector, –V, has the same norm as V, but opposite direction.


Two vectors, U and V, are **parallel** when U(U)==U(V). This test or constraint is written U//V or U==kV. The latter notation is a shortcut for "there exists a scalar k, such that U equals kV". When angle(U)==angle(V)+(1+2k)π, we say that U and V are **antiparallel**. We have then U(U)== – U(V).

✎ Let V=<3,4>. Convert it to polar coordinates. Compute the polar coordinates of W= –V by adding π to the angular coordinate of V. Convert W to Cartesian coordinates. Verify that they are <–3,–4>.

✎ Let V=<4,–6>. What is –V/2?

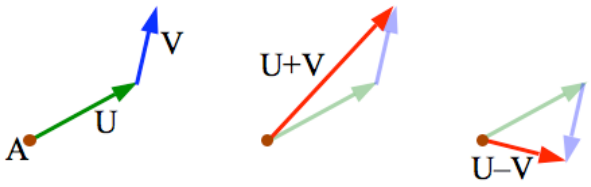
✍ How would you test whether two vectors in Cartesian coordinates are parallel, without computing their angular coordinates?

## 2.8 Adding vectors

Vector **addition** (or vector **sum**) has a simple intuitive meaning when we think of vectors as displacements: Start at some *arbitrary* point A and move by U and then by V. U+V denotes the cumulative effect of the two displacements. Of course, we can add more than two vectors: U+V+W…

To compute U+V, consider our metaphor for Cartesian coordinates and realize that instead of performing a horizontal displacement by U.x, then a vertical displacement by U.y, then a horizontal displacement by V.x, and finally a vertical displacement by V.y, we may perform the two horizontal displacements first, which amounts to a single horizontal displacement by U.x+V.x, and then a vertical displacement by U.y+V.y. Hence U+V=<U.x+V.x, U.y+V.y>.

Similarly, vector **difference** U–V is <U.x–V.x, U.y–V.y>.




Note that U+V = V+U, and that U–V = –(V–U).

More generally, we can write a **weighted sum** (linear combination) of two vectors as sU+tV to denote <sU.x+tV.x, sU.y+tV.y> and extend it to more than two vectors uU+vV+wW… which we conveniently denote using the **weighted average function** `A(u,U,v,V,w,W…)`.

✎ Explain at an intuitive level why U+V=V+U and why U–V≠V–U.

✎ Compute 3(<–1,–2> + 2<3,4> – <4,6>/2).

✎ Let U, V, and W represent forces exerted upon a point mass. U is exerted by a spring. V is gravity. W is the force that you exert by pulling on the mass with a string. You want the mass to stay where it is, hence you want the forces to balance. Compute W.

✎ Let U, V, and W represent forces exerted upon a point mass. They have the same magnitude (radial coordinate). What can you say about their angular coordinates if we assume that the mass is stationary (the 3 forces cancel out)? Given U, use that condition to compute V and W.

## 2.9 Dot product (also called inner product)

The dot product is the most important operator on vectors. It is used as the main tool in most geometric constructions. Hence, you need to fully understand what it computes and retain its formulation, properties, and example applications discussed in this section. You should be able to teach this section to your study group.

The **dot product,** V•U, of two vectors is a **scalar**. It is also called the **inner product** and may be written V∧U. It may be computed as (V.x U.x + V.y U.y) using Cartesian coordinates or as (n(U) n(V) cos(angle(U,V))) using polar coordinates. You need to know these two equivalent formulations, since they justify various fundamental properties of the dot product, which you also need to remember.

To prove the equivalence of these two formulations, rewrite (V.x U.x + V.y U.y) by substituting the Cartesian coordinates with their expressions in terms of polar coordinates to obtain: V.n U.n cos(V.a) cos(U.a) + V.n U.n sin(V.a) sin(U.a). Now, using trigonometric identities, we obtain ½V.nU.n(cos(V.a–U.a)+cos(V.a+U.a)+ cos(V.a–U.a)–cos(V.a+U.a) ), which yields V.n U.n cos(V.a–U.a), and hence the polar form U.n V.n cos(a(U,V)). You do not need to remember this derivation, but make sure that you understand it.

You do need to remember the following three properties, since you will be using them often to simplify expressions.

From the polar formulation and remembering that s(n,a)=(sn,a), we obtain: (sV)•(tU)=st(V•U).

From the symmetry of the Cartesian formulation, we obtain **commutativity**: V•U= U•V.

From the Cartesian formulation, we obtain **distributivity over vector addition**: (V+W)•U= V•U+W•U.

From the polar formulation, we notice that V•U==0 implies that one of the three quantities, U.n, V.n, or cos(a(U,V)) must be zero. Hence, when U and V are not null, V•U==0 implies that U and V are ***orthogonal*** to each other (i.e., that a(U,V) is either π/2 or –π/2). We often use V•U=0 as a *constraint* for computing intersections of linear entities as discussed blow. The use of V•U==0 to indicate orthogonality is arguably the most common application of the dot product.

Also, from the polar formulation, and remembering that U.n and V.n are positive, V•U>0 when |a(U,V)|<π/2. When this is the case, we say that U and V **agree in direction**. (Note that we did not say "have the same direction".) We use the sign of V•U for a variety of tests, including "Is this vertex convex?", "Do these two edges intersect?", and "Is this point inside the triangle?". Hence, the sign of V•U is the other key use of the dot product.

When inventing a solution to a geometric problem, look for statements that imply orthogonality of two vectors and write the corresponding equation. Note that the two vectors may not be given or named in the problem statement. They often must be derived from given points or other vectors.

✎ To practice working with expressions that include vector addition, subtraction, scaling, and dot product, simplify the expression (2U+3V)•(2U–3V).

✎ Test whether <2,3> and <–1,5> agree in direction.

✎ Prove (sV)•U=s(V•U).

✎ Prove V•U= U•V.

✎ Prove (V+W)•U= V•U+W•U.

## 2.10 "Cross" or outer product

The **outer product,** sometimes denoted U∨V is defined as the **determinant** |UV| of the 2×2 matrix ( U V ) having for colums the two column vectors U and V. Hence it may be computed as –U.xV.y+U.yV.x.

It is important to know that U∨V = n(U) n(V) **sin**(angle(U,V)). It may be computed as U•R(V).

Note the similarity with U•V = (n(U) n(V) **cos**(angle(U,V)).

Hence, in our code, we use the call c(U,V) to compute V•U and s(U,V) to compute V∨U. The mnemonics 'c' stands for "cos" and 's' stands for "sin".

Remember that U•V=V•U, but notice that U∨V = –V∨U.

As a short for V•R(U), we abuse the **cross-product notation** V×U. Note that, in this context, the cross-product is a ***scalar***. Beware however that, in three dimensions, the notation V×U defines a ***vector***.

U×V may be used as part of expressions for computing the area of a triangle or the distance between a point and a line.

## 2.11 Summary of dot and cross product notations and expressions

Dot (inner) product:

U•V = U∧V = (U.x V.x + U.y V.y) = n(U) n(V) cos(angle(U,V)) .code: c(U,V)

2D "cross" (outer) product:

U×V = U∨V = |UV| = (–U.x V.y + U.y V.x) = n(U) n(V) sin(angle(U,V)) = U•R(V). code s(U,V)

## 2.12 Geometric product

The **geometric product**, denoted UV, is defined as U∧V+U∨V.

It has been proposed as the fundamental building block for geometric constructions because the inner and outer products may be elegantly defined in terms of the geometric product: U∧V=(UV+VU)/2 and U∨V=(UV–VU)/2.

When U is a unit vector, UV is the sum of the signed magnitudes of the tangential and normal components of V with respect to U. It decomposes the displacement of V into its component along U and along R(U) and returns the sum of these two signed displacements.

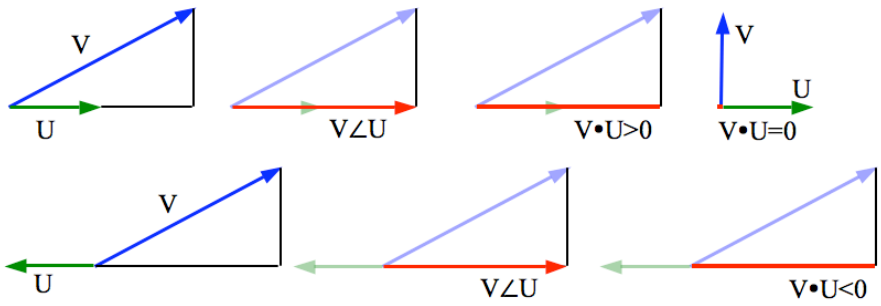
Let U =<1,0> and V=<x,y>, Let z=UV. Provide a simple formulation of the function z(x,y) and explain its shape or plot it.

## 2.13 Tangential and normal components

Now, temporarily assume that U is a unit vector. We will use it as a reference. For example, it may be the tangent direction to a wall. **When U is unit**, the quantity V•U measures the **signed length of the projection** of V onto the direction U. As stated before, V•U is positive when U and V agree in direction. By "projection", we mean the normal projection (or ***shadow***) of U on V.

V×U measures the signed magnitude of the normal component of V with respect to a unit vector U.

Now consider that V represents for example the velocity of a photon upon its impact with the wall. We often wish to decompose V into its tangential and normal components with respect to U. V•U measures the direction (sign) and magnitude of the tangential component of V with respect to U. However, it is a scalar (number), not a vector. We can define the **tangential component** (vector), denoted V∠U, of V with respect to U as (V•U)U, which is the scaling of U. When U and V do not agree in direction, neither do (V•U)U and U.



The **normal component**, V⊥U, of V with respect to U is defined similarly using the orthogonal vector R(U) instead of U: V⊥U=(V×U)R(U). The term "normal" here refers to the direction that is "normal", i.e. orthogonal, to the wall.

Hence, we can express a vector V in terms of its tangential and normal components as V = V∠U + V⊥U. You must know how to express these two components and remember that these formulae for tangential and normal components **assume that U is a unit vector**.

✎ Write V in terms of its tangential and normal component with respect to U and draw all 4 vectors.

✎ You are given W and told that it is V∠U. Does this information suffice to tell whether U and V agree in direction?

✍ Let U be the tangent vector to the beach and let d be the distance to the beach. Your boat moves at constant velocity V. Write an expression for the time t it will take you to reach the beach.

### 2.14 How to use the dot product to compute the reflection vector

To illustrate the power of the dot product, consider the problem of computing the **reflected direction** R of a ray that arrived with direction V upon a wall (edge) between point A and B. You should learn and remember the resulting formula and its derivation.

Define the unit tangent vector U=D(AB) to the wall.

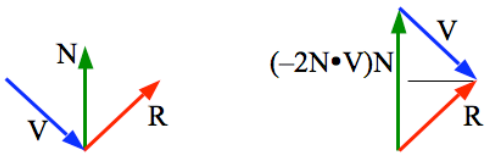
V may be written as the sum V∠U + V⊥U of its tangential V∠U and normal V⊥U components with respect to U.

By definition, R is the mirror image of V with respect to the wall. The **mirror image of a vector is obtained by flipping the sign of its normal component**. Hence, R = V∠U – V⊥U.

Note that R–V = –2(V⊥U). Hence, we obtain a slightly simpler formula: R=V–2(V⊥U).

Often, instead of the tangent direction U, the orientation of the wall is defined by its normal direction N. In 2D, N=R(U). Hence, V⊥U = V∠N = (N•V)N.

This change of variables leads to the popular formula which you should retain: R=V–2(N•V)N.




✎ A billiard ball has constant velocity V and no spin. It collides with a wall that has normal N. What is its velocity R after the elastic shock?

✍ Re-derive the popular formula for the reflection direction and justify each step. Draw all vectors and quantities used in the derivation and the formula.

# 3 - Points

### 3.1 What is the relation between points and vectors

If you pick an origin O, each point $P_i$ is associated with a unique vector $V_i=OP_i$. This trick let's you represent points by the corresponding vectors, and vice versa.

Even though, because of the trick above, both a point and a vector may be represented by the same pair of numbers, it is essential that you **distinguish points from vectors**. Failing to do so may slightly reduce the length of the source code, but will ultimately lead to confusion, wrong solution, coding mistakes, and debugging nightmare.

For example, it you wish to translate, by a displacement V, a ray, which is defined by a starting point P and a tangent direction T, the result is a ray starting at a displaced point P+V, but having the same tangent direction T. Hence, **under translation, points change, but vectors do not**.

You should always clearly specify which of your symbols are points and which are vectors, unless this is obvious from the context. For example if you say "Let P and Q be two points.", then PQ is clearly a vector and you do not

have to say “vector” PQ, although you may. If you say “Let P be a point and V a vector”, then P+V is clearly a point obtained by displacing P by displacement V.

✎ Explain the distinction between points and vectors.

## 3.2 How do we represent points

The Cartesian representation of a point P is two coordinates: (P.x, P.y). Their interpretation assumes an agreed-upon origin O and satisfy OP=<P.x, P.y>. Note that we use parentheses for the Cartesian coordinates of points and brackets for the Cartesian coordinates of vectors.

Remember that we have also used parentheses for the polar coordinates of vectors. Hence (3,4) may either indicate a point or a vector in polar coordinates. The context or explicit annotations should suffice to remove this ambiguity, but when in doubt, assume that this notation refers to a point.

✎ Suggest two good choices of origin when working with points in a window. Discuss their advantages and drawbacks.

## 3.3 How do we translate points

The simplest way to **translate** a point P by a displacement V is to add the displacement vector, hence obtaining P+V.

Sometimes, V is a direction and we wish to translate P along V by a specific distance s. The result may be defined as P+sV and obtained through the call `T(P,s,V)`. Such translation may of course be cascaded: P+sU+tV.

✎ Why may one want to translate a point? Give examples from feature animation or video games.

⌨ Write a simple and efficient implementation of the function for `T(P,s,V)`, assuming that points and vectors are represented by their Cartesian coordinates.

## 3.4 How do we rotate points

The **rotated** version `R(P,a)` of point P by angle A around an agreed-upon origin O is O+R(OP,a). Usually, we assume that O=(0,0). If we wish to use a different origin (fixed point) Q for the rotation, we may compute the rotated point by the call `R(P,a,Q)`. Its implementation is discussed below.

⌨ Write a simple and efficient implementation of the function for `R(P,a)`.

## 3.5 When is it acceptable to write affine combinations of points

In general, one should try not to scale or add points, since they represent locations and since the concept of adding or scaling locations is absurd. Hence, we should not write P+Q or sP, when P and Q are points.

✎ You wish to construct a parallelogram (O,P,Q,R), given the origin O and two points P and Q. You should not compute R as P+Q, which is an illegal operation on points. Provide a legal construction.

We will tolerate two exceptions to this rule (homothety and affine combinations).

A change of units corresponds to a **scaling** (also called *homothety*, *dilation*, and *central similarity*) with respect to a given origin O. The scaling P’ of a point P by a scaling factor s should be written O+sOP, but we will tolerate the form sP and even P/s. A blatant example of such transformations will be discussed when we explore perspective projections that map a point P onto dP/(d+P.z).

The expression uU+vV+wW… is a **linear combination** of vectors U, V, W… with scalar coefficients u, v, w… When u+v+w… = 1, it is called an **affine combination**. An affine combination is called a **convex combination** when none of the coefficients is negative.

By extension, when a+b+c…=1, point O+aOA+bOB+cOC… is an **affine combination of points** A, B, C… and its location is **independent of the choice of the origin** O. As a proof, pick another origin O’. The affine combination O’+aO’A+bO’B+cO’C… may be written as O’+O’O+(a+b+c…)OO’+aO’A+bO’B+cO’C… by adding a the null vector O’O+OO’, which can be written O’O+(a+b+c…)OO’, since a+b+c…=1. The new expression yields O’+O’O+aOO’+aO’A+bOO’+bO’B+cOO’+cO’C…, which simplifies to O+aOA+bOB+cOC.

Given that O+aOA+bOB+cOC… is independent of O, many authors use a shortcut and write this **affine combination** of points simply as aA+bB+cC… This shortcut notation is tolerable and quite handy, but only when a+b+c…=1 (i.e., affine combination).

Note that such point addition follows the same rules as number addition with respect to distributivity of multiplication and division by a scalar. Hence ½A+½B can be written ½(A+B) or simply (A+B)/2.

For example, the midpoint of the line segment (P,Q) may be written as (P+Q)/2, rather than the legal P+PQ/2.

✎ Explain what are linear, affine, and convex combinations of vectors.

✎ What is an affine combination of points? What important property justifies that we do not need to use legal points + vectors expression and may instead use the shortcut?

✎ The center of triangle (A,B,C) may be written as (A+B+C)/3. What is the "legal" form for that location?

### 3.6 When is the path joining three points making a left turn

Consider a path A-B-C made of two straight edges: Edge(A,B) and Edge(B,C). How can we test whether this path makes a left turn at B? As we will see, this test is vital for testing whether a point lies inside a triangle and whether a vertex of a polygon is concave. Many algorithms build on these two functionalities.

The answer, returned by `Left(A,B,C)`, may be formulated using what we have learned so far. `Left(A,B,C)` returns true when R(AB)•BC>0.

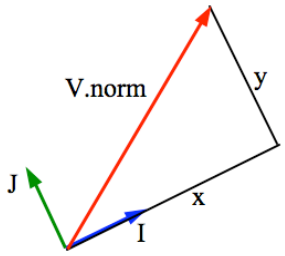
✎ Draw three configurations of the path A-B-C showing a left turn at B, a straight path, and a right turn at B. For each verify that the formula for computing `Left(A,B,C)` is correct.

✎ Recall the formula for computing `Left(A,B,C)` and explain it.

## 4 - Frames

### 4.1 What is a basis

Consider two directions, I and J, such that J=R(I). They define a **basis** denoted [I,J]. The **components** <x,y> of vector V in the basis [I,J] are defined as x=V•I and y=V•J. Note that, given a basis [I,J] and two components <x,y>, the vector V may be expressed as the **weighted vector sum** V=xI+yJ. Note that the norm V.n is the diagonal of a right triangle, hence, by **Pythagoras's theorem**, $x^2+y^2=V.n^2$.




### 4.2 How to change basis

Consider two bases: $[I_1,J_1]$ and $[I_2,J_2]$. Let $<x_1,y_1>$ be the components of a vector V in $[I_1,J_1]$. If we wish to **change the basis**, we must compute the components $<x_2,y_2>$ of V in $[I_2,J_2]$. We do this by first computing $V=x_1I_1+y_1J_1$ and then $x_2=V•I_2$ and $y_2=V•J_2$.

To **implement** scaling, addition, subtraction, projection, and dot-product, we must decide on a **representation** for vectors. Usually, one represents vectors by their components in a **global basis** that is implicit, i.e., not specified. In two dimensions, it is customary to assume that the first basis vector (the X-direction) is horizontal pointing towards the right of the page and that the second basis vector (the Y-direction) is therefore vertical. Mathematicians often assume that it is pointing up, while some graphics packages assume that it points down. We call this a **Cartesian representation** of the vector.

Let <V.x,V.y> be the components of V. Note that if we do not specify a basis, we assume that the components are computed with respect to the global basis. Similarly, let <U.x,U.y> be the components of U. We implement the **operators** discussed above as follows:

sU = < sU.x , sU.y >, –U = < –U.x, –U.y>
U+V = < U.x+V.x , U.y+V.y>, U–V = < U.x–V.x , U.y–V.y>
R(U) = < –U.y , U.x >
V•U = U.xV.x + U.yV.y

$V.n = \sqrt{(V \bullet V)}$

✎ Select values for the coordinates of $[I_1,J_1]$, $[I_2,J_2]$ and for $\langle x_1,y_1\rangle$. Draw these 5 vectors. Compute $\langle x_2,y_2\rangle$ and show them on your drawing.

✍ Express the change of coordinate systems in matrix form.

## 4.3 When are two vectors parallel and when are they orthogonal

Stating that two vectors are parallel or perpendicular implies an equation that can be written as a geometric expression or algebraically as an equation on the coordinates of the vectors. Assume that U.norm≠0 and V.norm≠0. U and V are **perpendicular** when V•U=0 (i.e., when U.xV.x + U.yV.y=0) and **parallel** when V•R(U)=0 (i.e., when U.yV.x = U.xV.y). You should be able to re-derive these equations.

✎ Write the equation satisfied by the coordinates of two parallel vectors.

✎ Write the equation satisfied by the coordinates of two orthogonal vectors.

## 4.4 How are points defined in a coordinate system

**Points** in two-dimensions are associated with locations in the plane. If we assume the existence of an implicit (unspecified) global **origin** G, to each point P of the plane corresponds a vector from G to P. We use the combination GP to denote that vector. Remember that GP=P–G and that GP= –PG.

Consequently, we may use the components <x,y> of GP to represent P. However, we must not confuse points and vectors, which have different semantics and different operators. Consequently, when referring to a point P, we will call x and y its **coordinates** and will denote them using parentheses (x,y), rather than brackets.

Remember that (x,y) is in fact the point G+xI+yJ, where [I,J] is the default basis discussed above.

Let (P.x,P.y) be the coordinates of point P and (Q.x,Q.y) be the coordinates of point Q. The **vector** PQ=Q–P has components <Q.x–P.x,Q.y–P.y>.

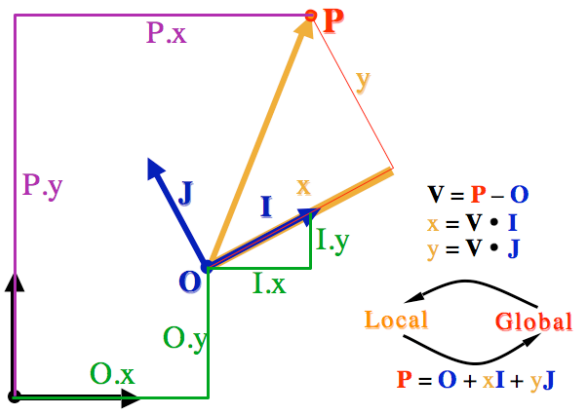
## 4.5 What are rigid transformations

Consider a point P represented by its coordinates (x,y) with respect to a **local coordinate system** in which the point was specified by the user as part of some shape. Now, we want to place the shape in a scene or to animate its motion through a scene. To do so, we will use compositions of two **rigid body transformations**: rotations and translations. The **translation** $T_V(P)$ of P by vector V is P+V. The **rotation** $R_a(P)$ is the point (x cos(a) – y sin(a) , x sin(a) + y cos (a)). We can compose the effects of several transformations.

For example, $T_V(R_a(P))$ will first rotate by a, then translate by V. Similarly, the transformation $T_U(R_b(T_V(R_a(P))$ will first rotate by a, then translate by V, then rotate by b, and finally translate by U. Note that two consecutive translations commute and can be merged: $T_U(T_V(P))=T_V(T_U(P)) =T_{U+V}(P)$. Similarly, consecutive rotations (in 2D only) commute and can be merged: $R_b(R_a(P))=R_a(R_b(P))=R_{a+b}(P)$. However, rotations and translations do not commute: in general $T_V(R_a(P))\neq R_a(T_V(P))$.

In order to provide a fixed size representation of the composition of an arbitrary number of rotations and translations, we can use a **coordinate system** [I,J,O], also called **frame**, which defines an origin O and basis directions [I,J]. Given the coordinates (x,y) of point P in [I,J,O], we compute the point P=(P.x,P.y) in the global coordinate system of the scene as P=O+xI+yJ. In other words, to get to point P, start at the origin O, walk x units along the I direction, then walk y units along the J direction. We assume here that the coordinates of O, I, and J are given in the global coordinate system. The **local coordinates** (x,y) may of course be negative.

Occasionally, we may want to perform the **inverse transformation** and compute the local coordinates (x,y) of some point P defined by its global coordinates (P.x,P.y). For example, the user may wish to select a point on an instance of the object in the scene and all we have is the (P.x,P.y) coordinate of the graphic pointer. To obtain the local coordinates (x,y) of that point with respect to the object, we use the change basis discussed above: x=OP•I, y=OP•J.




Consider now that we have a vector or direction V defined by their components <x,y> in the local coordinate system where the object was designed. For example, that vector may be the tangent or normal to a virve at a given point. How can we compute their components in the global coordinate system? We simply **ignore the translation part** (origin) and perform a change of basis, V=xI+yJ, as explained above. Remember that **vectors are not affected by translations**.

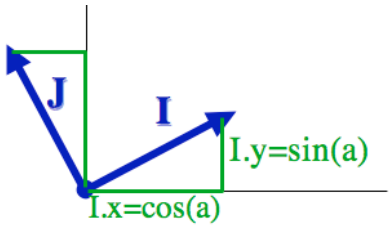
✎ You are given two frames $[O_1,I_1,J_1]$ and $[O_1,I_2,J_2]$ and the local coordinates $(x_1,y_1)$ of a point P in $[O_1,I_1,J_1]$. Explain how to compute the local coordinates $(x_2,y_2)$ of P in $[O_1,I_2,J_2]$. Draw an example, showing P, $[O_1,I_1,J_1]$ and $[O_1,I_2,J_2]$ and indicate in the drawing the 4 coordinates.

## 4.6 Homogeneous matrices

To use matrix multiplications for composing rotations and translations, we represent a transformation that defines a local coordinate system [I,J,O] by a 3×3 **homogeneous matrix** [**I.h J.h O.h**] where the .h operator maps a point (x,y) to its homogeneous counterpart (the homogeneous three-dimensional vector <x,y,1> obtained by **adding a 1** as third coordinate) and maps a vector <x,y> to its homogeneous counterpart (the homogeneous three dimensional vector <x,y,0> obtained by adding a zero as third coordinate). Notice the difference: points are padded with a 1 (so that we take translations into account) and vectors with a 0 (so that we do not take translations into account).

To transform a point in local coordinates (x,y), we perform the matrix-vector multiplication and compute <P.x,P.y,1> as [**I.h J.h O.h**](x,y,1) = x**I.h** + y**J.h** + **O.h**.

$$\begin{pmatrix} P.x \\ P.y \\ 1 \end{pmatrix} = \begin{pmatrix} I.x & J.x & O.x \\ I.y & J.y & O.y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$




Note that the top left 2×2 portion of that homogeneous matrix is the basis [I J], which represents a rotation $R_a$, where I.x = J.y = cos(a) and I.y = –J.x = sin(a). The top two entries of the right-most column represent the translation vector. Hence, given any 3×3 matrix representing such a transformation or the composition of an arbitrary number of transformations, one can compute the **final rotation angle** a=atan2(I.y,I.x) and the **final translation vector** V=<O.x,O.y>. Note that this transformation corresponds to $T_O(R_a(P))$, where the rotation is performed first. Inversely, given angle a and translation vector V, we can trivially compute the coefficients of the matrix directly.

We can similarly **transform a vector** V=<V.x,V.y> by multiplying its homogeneous formulation <V.x,V.y,0> by the homogeneous matrix. Note that the padded zero will multiply—and hence cancel—the translation effect.

Also, note that the **inverse** of $R_a$ is $R_{-a}$, which is obtained from $R_a$ by changing the signs of the I.y and J.x coefficients, since sin(–a) = –sin(a) and cos(–a)=cos(a). Finally, the **inverse** of $T_O(R_a(P))$ is $R_{-a}(T_{-O}(P))$.

✎ Write the matrix form of a transformation that first performs a rotation of 90 degrees and then a translation by <1,2>. Apply this transformation to point (2,3). Show the details using homogeneous coordinates. Plot the point and its image and verify that it is correct.

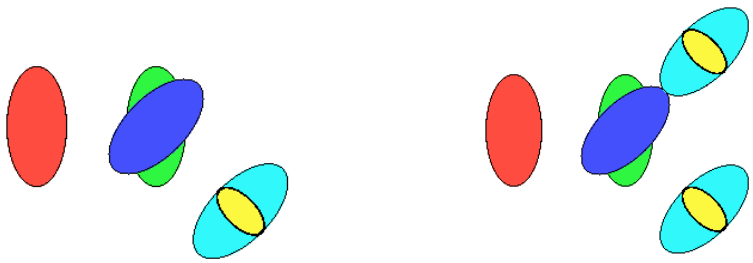
✎ Apply the above transform to the vector <2,3>.

## 4.7 Transformations in graphic libraries

Graphic libraries provide support for these transformations. However, they must be called in **reverse order**. For example, to render the transformed point $T_U(R_b(T_V(R_a(P))$, we would issue the sequence of commands: `translate(U.x,U.y); rotate(b); translate(V.x,V.y); rotate(a); render(P);` To develop an intuitive understanding of this approach, consider that the calls to translate and rotate transform the global coordinate system with respect to which all subsequent transformation and rendering operations will be performed. Hence, in the sequence `{translate(U.x,U.y); rotate(b);}`, the rotation is performed around the new origin, U.

Graphic libraries also provide a **scaling** transformation, which is not a rigid body transformation. Specifically, `{scale(a,b); render(P);}` will render point (aP.x,bP.y). Scaling does not preserve distances, nor vector norms. Furthermore, when a≠b, the scaling operation will not preserve angles. Hence, one should be careful when transforming normal directions and other vectors by a scaling transformation.

Let `paint()` render an ellipse with height 100 and width 50. The sequence `{fill(red); paint(); translate(100,0); fill(green); paint(); rotate(PI/4); fill(blue); paint(); translate(100,0); fill(cyan); paint(); scale(1.0,0.25); fill(yellow); paint(); }` would produce the image below on the left, while the sequence `{fill(red); paint(); translate(100,0); fill(green); paint(); rotate(PI/4); pushMatrix(); fill(blue); paint(); translate(100,0); fill(cyan); paint(); scale(1.0,0.25); fill(yellow); paint(); popMatrix(); translate(0, -100,0); fill(cyan); paint(); scale(1.0,0.25); fill(yellow); paint(); }` would produce the image below on the right. Note that the command `pushMatrix()` pushes the current coordinate system (the one used to paint the blue ellipse) on the stack and that `popMatrix()` restores it. The global x-axis goes right. The y-axis goes down.



In mathematics, a transformation T is said to be **linear** if it preserves vector addition T(V+U)=T(V)+T(U) and multiplication by a scalar T(sV)=sT(V). Hence combinations of rotations, uniform scaling, and even shear (scaling along a single direction) are linear transformations on vectors. Therefore, we also say that any combinations of rotations and shear on points is a linear transformation. A linear transformation in 2D may be represented by a 2×2 matrix.

An **affine** transformation on points is a linear transformation followed by a translation. As shown above, any affine transformation may be represented by a homogeneous matrix. Affine transformations map lines onto lines and preserve affine combinations of points (T(aA+bB+cC…)=aT(A)+bT(B)+cT(C).. when a+b+c…=1).

A **homothety** (sometimes written homotecy) and also called dilation is a combination of a translation with a uniform scaling. It maps any line into a parallel line. The scaling factor s is also called the similitude ratio. When the ratio is negative, the homothety inverts the points with respect to the fixed point Q of the dilation. Hence, the image P' of a point P is defined by QP'=sQP. The homothety is an affine transformation and is also a similarity transformation. When Q is the origin, the homothety is a linear transformation.

✎ Write the code that will draw a spiral made of increasingly larger disks. Discuss the two kind of spirals and write the code for each.

## 4.8 How to rotate a point around a fixed point

What is the **rotation** P' by angle a of point P around center C? We can obtain P by rotating CP by angle a around C and then adding it to C. Hence P'=C+$R_a$(CP).

Note that we can also directly compute it as P'=C + CP.x <cos a , sin a> + CP.y <sin a , cos a>.

### 4.9 How to rotate a portion of the scene around a fixed point

To rotate a portion of a scene by an angle a around a fixed point C, we could transform each vertex as discussed above and rotate simply each vector by a. If the entire scene is completely defined in terms of these vertices and vectors, the result will be correct. However, it may be slow. To take advantage of the GPU hardware acceleration for performing geometric transformations, we may use the graphics API (Processing, OpenGL…) to temporarily change the homogeneous matrices that will be used when transforming points and vertices of all geometric primitives that will be sent to the graphics pipeline. Our solution involves 6 steps:

1) Push the matrix onto the stack to save its previous state

2) Translate by CO, now C is temporarily at the origin

3) Rotate by angle a, which does not affect C

4) Translate by OC, to reverse the effect of 2)

5) Render the desired portion of the scene

6) Pop the matrix to restore its previous state

## 5 - Intersections of lines and edges

In this section, we discuss the computation of intersections between lines and line segments.

### 5.1 What is a parametric form of a point on a line

A given line is an infinite set of points. How can we define all the points of a given line? How can we identify a particular point?

Consider the location A+sAB, where A and B are fixed points, but where s is a variable (parameter). To indicate that the result depends on the value of s, we write it as P(s). Note that P(0)=A+0AB=A, and that P(1)=A+1AB=B. Hence, as s varies from 0 to 1, P(s) traces the line segment from A to B. If we restrict s to be in [0,1], then P(s) is a convex combination of A and B.

Since A+sAB=A+s(B–A)=A+sB–sA=A–sA+sB, we can write P(s) as the affine combination (1–s)A+sB and use the function `I(P,s,Q)` to compute it.

When s is not confined to the interval [0,1], P(s) covers all the points on the line that passes through A and B. Note that if we chose n(AB) as unit of distance, then |s| measures the distance between A and P(s) and its sign indicates whether P(s) is on the same side of A as B along the line.

Because we use a parameter s to identify any given point on the line, we say that P(s)=A+sAB is the **parametric form** of the line. Such a form is often used when computing points on the line that satisfy a given property, which is usually defined by an equation. For example, we may require that P(s) lie on another given line or circle or that it minimizes the distance to a given point Q. Hence, the problem is solved by finding the value or values of s for which the equation is satisfied and by substituting these values (roots) in A+sAB.

You should remember how to derive the parametric expression of the line passing through points A and B.

✎ How would you compute the point P at one third of the distance from A to B? Draw it. Write the corresponding function call, the legal form of the geometric construction, and the affine combination.

✎ Write the parametric expression of the line through points <1,2> and <3,4>. Compute the point where it crosses the X-axis.

### 5.2 Implicit formulation of a line

Sometimes, when formulating the geometric construction of a point, we may want to create auxiliary lines (for example the median of two points). We may define a line by specifying two different points, A and B, that it contains. We create an instance of that line via the call `L=line(A,B)`. We may also define a line that passes through a point R and has normal N. (By that we mean that N is the direction orthogonal to the line.) We may create an instance of such a line by the call `L=line(R,N)`. A point Q lies on L when the vector RQ is parallel to L, and thus when RQ is orthogonal to N. Hence, L is the set of points Q that satisfy equation RQ•N=0. This is the

**implicit equation** of line L. This equality (i.e., constraint) is useful when computing intersections between a line and another curve.

## 5.3 How to compute the intersection of two lines using implicit forms

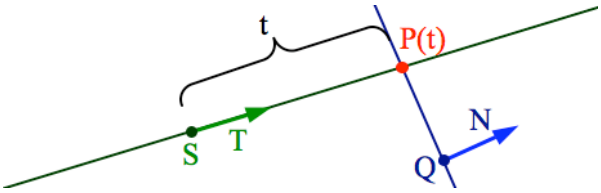
Let $L_1$=line($R_1$,$N_1$) and $L_2$=line($R_2$,$N_2$). Their intersection, $L_1 \cap L_2$, is the set of points Q satisfying simultaneously the two equations: $R_1Q \bullet N_1=0$ and $R_2Q \bullet N_2=0$. Note that these are two linear equations with two variables: the coordinates Q.x and Q.y of Q. When $N_1 \neq N_2$, the intersection point exists and is unique. The coordinates of Q are obtained by solving this linear system.

## 5.4 How to predict the collision time

Consider the **parametric form** of a point P(t) that starts (i.e., when t=0) as S and moves with **constant velocity** T. Hence, P(t)=S+tT. We want to compute the value of time t, when P(t) **collides** with a planar **wall**. We represent the wall by a point Q on its surface and by the direction N of its normal (i.e. the direction orthogonal to the wall). In 2D, this problem is actually one of finding the time t where a particle that travels at constant speed collides with a line that passes through Q and has normal N.

To solve this problem, we need to formulate an **equation** in t that describes the particular configuration between P(t), Q, and N at collision. What is that equation? As seen above, it is QP(t)•N=0. Hence, we obtain the collision time t by solving this equation. We obtain the collision point by substituting the solution for t in S+tT.

How do we solve QP(t)•N=0? First, remember that QP(t) is P(t)–Q. Substituting P(t) with S+tT, QP(t) becomes S+tT–Q, which can be written S–Q+tT, and hence QS+tT. Distributing the dot product over vector addition and scaling, QP(t)•N becomes QS•N+tT•N. Hence, the solution of QP(t)•N=0 is t = – QS•N / T•N or equivalently, using –QS=SQ, t = SQ•N / T•N. You should be able to re-derive this solution.




## 5.5 How to compute the intersection of two lines using implicit and parametric forms

In the previous section, P(t) is the parametric form of a line passing through S with tangent T. Hence, the solution constructed above computes the intersection of two lines, one represented in its parametric form (by point S and tangent T), and the other in its implicit form (by point Q and normal N).

## 5.6 How are half-spaces defined

A line L passing through point Q with normal N partitions the plane into three sets. The line itself is the set of points P satisfying equation QP•N=0. The **interior**, denoted L.interior, is the set of points satisfying equation QP•N<0. The **exterior**, denoted L.exterior, is the set of points satisfying equation QP•N>0. Note that these three sets are pairwise disjoint. L.interior and L.exterior are called **half-spaces**. They are **open** because they do not contain their boundary L. Sometimes, a **closed** half-space is desired. For example, the closed interior is the union of L with L.interior and may be denoted as L.interior.closure. Note that choice of terminology implies that the normal N points towards the exterior. Similarly, assume that you travel in the tangent direction T along L. By convention, we may agree that N=R(T) and hence that L.interior is on your **right**.


We say that a line L **separates** points P and Q when one point is in L.interior and the other in L.exterior.

✎ Write the geometric construction and then the code that will test whether a line passing through points A and B separates points P and Q.

## 5.7 When do edges intersect

An **edge** E, from point A and point B will be denoted Edge(A,B). It is an oriented line segment. It is the set of points E(t)=A+tAB with t∈[0,1]. By rewriting tAB as t(B–A) and then distributing and collecting terms, one

obtains an equivalent weighted sum formulation: E(t)=(1–t)A+tB. Let E.start denote the **starting point** A and E.end the **ending point** B of edge E.

As we will see later, such edges may correspond to the sides of a polygonal region and may be oriented (clockwise) so that the interior of the polygon is on their right.

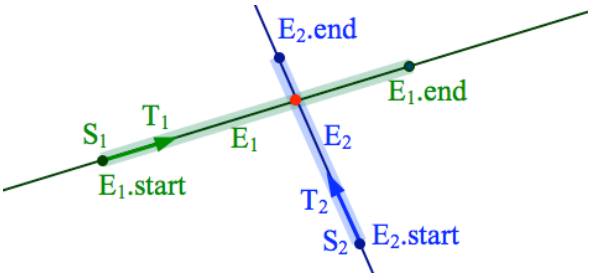
An edge E from A to B is a subset of an oriented line Line(E) that has D(AB) for tangent direction. Hence, we say that point P is on the left of E when it is on the left of Line(E).

When do edges $E_1$ and $E_2$ intersect? An accurate answer to this question is surprisingly complex. For now, assume that the set of 4 vertices bounding these two edges does not contain any triplet of collinear vertices. Such a simplifying assumption is excluding the **singular cases** of collinear edge and even situations when the vertex of one edge lies on the other. When such singularities are excluded, we say that our solution is limited to **general positions**.

In this general case, **$E_1$ and $E_2$ intersect if and only if Line($E_1$) separates the vertices of $E_2$ and vice versa**.

✍ Explain and justify the above result.

Note that this intersection test does not require computing the intersection point. If the test indicates that an intersection exist, and if the intersection point is desired, it may be computed as the intersection of the two supporting lines, as already explained above.




To deal with the singular cases, assuming full accuracy of numeric computations, one may try to enumerate all topological configurations possible and implement a decision tree that identifies the proper configuration. Note that the intersection may be the empty set, a single point, or an edge (where the two collinear edges overlap).

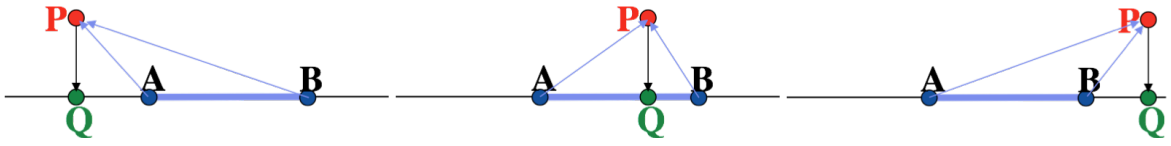
The presence of numeric round-off errors further complicates things. For example, because of round-off errors, one may not be sure whether the vertex of one edge is really to the left of the other. There are two main approaches to deal with this problem. The first one is to eliminate numeric round-off errors by using extended precision rational arithmetic, which typically slows down computation. The second one is to forgo accuracy and simply strive to produce an answer that is correct but for a slightly modified configuration, where for example the vertices have been moved. For example, if we discover that the vertex of one edge is sufficiently close to the line of the other edge, we may decide that it lies on that line, hence favoring a singular case. The difficulty with this approach is to guarantee that these guesses never lead to a logical impossibility. For example, some guesses may indicate that the two vertices of the first edge are on the line of the second edge, but that the reverse is not true.

✎ Write the geometric construction and then the code that will test whether Edge(A,B) and Edge(C,D) intersect, assuming general position.

✎ Assuming that an intersection is found by the above test, explain how to compute the intersection point and provide the geometric construction.

## 5.8 How to compute the closest projection of a point onto an edge

Consider an Edge(A,B) between points A and B and a point P. Let Q be the closest point to P onto Line(A,B). Q may be to the left of A, on A, between A and B, on B, and to the right of B. What test should we use to decide?



An elegant answer simply positions the scalar AP•AB on the real line with respect to 0 and AB•AB. For example, when AP•AB<0, then Q is to the left of A. When 0<AP•AB<AB•AB, then Q lies between A and B. When AP•AB<AB•AB, then Q=B.

# 6 - Triangles

A **triangle**, Tri(A,B,C) interpolates its three vertices, A, B, and C. Note that it is the **area** of the plane between these three points, not just the three edges! Triangles are the predominant geometric primitive for 3D graphics. Here however, we only consider triangles in the plane and study their interactions with points, edges, lines, and other triangles in the plane.

## 6.1 How to test whether a triangle is clockwise

We say that Tri(A,B,C) is clockwise when `left(A,B,C)` is false. It is counterclockwise otherwise.

## 6.2 How to test whether a point lies inside a triangle

How can we use what has been discussed above to devise a simple test that will report whether a point P lies inside Tri(A,B,C)? Again, for simplicity, let us assume that the triangle is not degenerate (i.e., that its vertices are pairwise disjoint and not collinear) and that P is not collinear with any one of the edges of the triangle.

We say that P is in the interior of Tri(A,B,C) when no line that passes through 2 vertices of the triangle separates P from the third vertex.

- Explain why the proposed solution for testing point-in-triangle inclusion works.
- Explain how to modify it to test whether P lies on the border (on the 3 bounding edges) of the triangle.
- Provide the pseudocode for testing whether point P lies inside Tri(A,B,C) by using left(…).
- Provide an algorithm for testing whether an Edge(P,Q) intersects Tri(A,B,C).

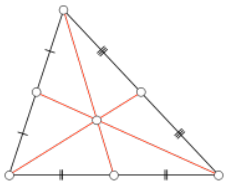
## 6.3 What is the area of a triangle

The area, area(A,B,C), of Tri(A,B,C) is AC•R(AB)/2.

- Verify this formula for a triangle with vertices (0,0), (1,0), and (0,1).
- Justify this formula.

## 6.4 What is the center of mass of a triangle

The center of mass (also called the geocenter) of Tri(A,B,C) lies on the three medians and therefore at the centroid. It is (A+B+C)/3. Note that the 3 medians divide the triangle into 6 smaller triangles of equal area, hence they each divide the triangle into two parts of equal area. However, no other lines that passes through the centroid has that property. Nevertheless, the triangle will stay in equilibrium if positioned on a line that passes through its center of mass.



## 6.5 What is the barycentric coordinates of a point with respect to a triangle

Given a non-degenerate triangle, T=Tri(A,B,C), any point P may be uniquely expressed as aA+bB+cC, with a+b+c=1. The triplet (a,b,c) is called the barycentric coordinates of P in T.

Barycentric coordinates are convenient for computing a variety of special points of triangles, see for example http://mathworld.wolfram.com/BarycentricCoordinates.html

## 6.6 How to compute the barycentric coordinates of a point with respect to a triangle

How can we compute the barycentric coordinates of a point P with respect to a non-degenerate triangle, T=Tri(A,B,C)? A simple, although possibly not the most efficient formulation uses areas:

The barycentric coordinate a of P is area(P,B,C)/area(A,B,C). Similar formula define b and c.

- Provide the pseudocode for computing the barycentric coordinates of a point P with respect to Tri(A,B,C).

# 7 - Circles

## 7.1 What is the implicit equation of a circle

A circle Circ(C,r) of center C and radius r is the set of all points P at distance r from C. Hence, the implicit formulation of Circ(C,r) is {P: ||PC||=r}. In other words, all points P on Circ(C,r) must satisfy the implicit equation ||PC||=r. Note that in practice, we often use an equation where the two terms are squared: $PC \bullet PC = r^2$.

✎ Provide the simple geometric form of the quadratic equation satisfied by all points on a circle of a given center and radius.

✎ Expand this equation for the case where C=(0,0) and verify that this is the usual equation of a circle of radius r around the origin.

## 7.2 What is the parametric equation of a circle

A point P on Circ(C,r) may be expressed parametrically as C + r cos(t) I + r sin(t) J with t varying between 0 and 2π. Note that we often replace cos(t) with $(1-u^2)/(1+u^2)$, and sin(t) with $2u/(1+u^2)$, where u=tan(t/2), in hope to produce an implicit polynomial equation in u.

✎ Verify this formula for a triangle with vertices (0,0), (1,0), and (0,1).

## 7.3 What is the circumcircle of a triangle and how to compute it

The circumcircle Cir(C,r) of Tri(A,B,C) is the circle passing through all 3 vertices. Its center C has barycentric coordinates ( $a^2(b^2+c^2-a^2)$, $b^2(c^2+a^2-b^2)$, $c^2(a^2+b^2-c^2)$ ), where a=||BC||, b=||ca||, and c=||AB||. Its radius r is abc/(4w), where w is the total area of the triangle.

## 7.4 How to compute the intersection of a circle with a line

Several approaches are possible. For example, one may plug in the parametric equation of the line into the implicit equation of the circle. This generates a quadratic equation. Its real roots, when they exist, define the intersection point.

✎ Provide a test that establishes whether Line(A,B) and Circ(C,r) intersect.

✍ Write an algorithm that computes the number of intersection points between Line(A,B) and Circ(C,r).

✍ Provide an algorithm, with the detailed geometric constructions, that computes the intersection points between Line(A,B) and Circ(C,r), assuming that the line does intersect the circle.

✍ Write an algorithm that computes the number of intersection points between Edge(A,B) and Circ(C,r).

# Notation

We provide here a summary of some of the notations and symbols used in the chapter. Note that calls to software methods or functions are indicated using a `different font`.

| | |
|---|---|
| A, B, C, D, P, Q, R | points |
| $P_0$, $P_1$, $P_2$, $P_j$ | points in an ordered sequence |
| `P.x`, `P.y` | Cartesian coordinates of a point P |
| L, U, V, W | vectors |
| N, T, U | directions (unit vectors indicating normals, tangents…) |
| `V.x`, `V.y` | Cartesian coordinates of a vector V |
| V.n, `n(V)`, or ||V||. | the norm (also called length, magnitude, and radial coordinate) of vector V |
| V.a, `a(V)` | angle (also called the angular coordinate) of vector V |
| V.U or `U(V)` | the direction (unit vector) V/V.n of V |
| a(U,V) or `angle(U,V)` | angle in [-π,π] between vectors U and V |
| PQ, Q–P, or `V(P,Q)` | vector (displacement) from P to Q |
| `equal(A,B)`, A==B | points A and B define the same location |

| | |
|---|---|
| `equal(U,V)`, U==V | vectors U and V define the same displacement |
| `R(V,b)` | V rotated by angle b: (V.n,V.a+b) |
| `R(V)` | V rotated by 90 degrees: (V.n,V.a+π/2) |
| sV, `S(s,V)` | scaled vector: **0** when s==0; (sV.n,V.a) if s>0, (\|s\|V.n,V.a+π) if s<0 |
| U//V, U==kV | vectors U and V are parallel |
| –V | opposite vector with radial coordinate V.n and angular coordinate V.a+π |
| sU+tV | weighted sum <sU.x+tV.x, sU.y+tV.y> of two vectors U and V |
| `W(u,U,v,V,w,W…)` | weighted sum uU+vV+wW… of three or more vectors |
| U•V or `c(U,V)` | dot (interior) product U•V = U.xV.x+U.yV.y = U.n V.n cos(a(U,V)) |
| U×V or `s(U,V)` | cross (exterior) product U×V = –U.xV.y+U.yV.x = U.n V.n sin(a(U,V)) |
| U•V==0 | vectors U and V orthogonal to each other |
| V∠U | Tangential component (V•U)U of vector V with respect to unit vector U |
| V⊥U | Normal component (V•R(U))R(U) of vector V with respect to unit vector U |
| UV | Geometric product U•V+U×V of two vectors |
| `T(P,s,V)` | Point P+sV obtained by translating P by the scaled vector sV |
| `I(P,s,Q)` | Point P+sPQ=(1–s)A+sB, affine combination of points P and Q |
| `Line(P,N)` | Line through P with normal N |
| `Line(P,Q)` | Line through P and Q, oriented from P to Q |
| `Edge(P,Q)` | Edge from P to Q |
| `Circ(C,r)` | Circle of center C and radius r |
| `Left(A,B,C)` | C is on the left of Line(A,B). Equivalently, A-B-C makes a left turn at B. |

## References

David Hestenes: *New Foundations for Classical Mechanics* (2nd Edition), Kluwer, in Fundamental Theories in Physics, Academic Pub. Offers a nice discussion of the geometric product.

## Resources

The concepts presented here are discussed on various web sites. These include:

MathWorld http://mathworld.wolfram.com/DotProduct.html

AlgebraLab http://www.algebralab.org/lessons

The Virginia tech Math Emporium has interactive practice tools for learning about vectors and matrices in the Math 1114 module http://www.emporium.vt.edu/math1114/index.html and on Vector Geometry in the Math 1224 module accessible from at http://www.emporium.vt.edu/math1224/index.html.

## Exercises

1) Let V=<3,4>. Compute –V, 2V, V.norm, V.direction, V.left.
2) Are vectors <3,4> and <–9,–25> parallel?
3) Let U=<1,–2> and V=<3,4>. Compute U+V, U–V, U•V, V•U, V∠U, and U∠V.
4) A ball arrives at speed V=<3,–4>. What it its speed U after an elastic chock with the ground (no gravity)?

5) Compute a coordinate system $(O_1,I_1,J_1)$ that would be the result of a translation by <3,4> followed by a rotation of 90 degrees counterclockwise. Compute a second coordinate system $(O_2,I_2,J_2)$ that would be the result of a rotation of 90 degrees counterclockwise followed by translation by <3,4>.
6) Provide two expressions for the center of mass of a triangle with vertices A, B, and C: one as a weighted combination of points and one using proper operators on points and vectors.
7) Let $(x_1,y_1)$ be the coordinates of a point in the system $(O_1,I_1,J_1)$. Let $(x_2,y_2)$ be the coordinates of a point in the system $(O_2,I_2,J_2)$. Provide closed-form conversion expression for $(x_2,y_2)$ in terms of all the other variables.
8) Let U and V be two non-null vectors. Provide simple equations for testing whether U and V are parallel and whether U and V are orthogonal in terms of their coordinates <V.x,V.y> and <U.x,U.y>.
9) You are looking on an architectural drawing. There is a short wall between points A and B. There is a long wall between points C and D. There is a light source at E.
10) Write a test that will determine whether Edge(A,B) intersects Edge(C,D).
11) Given an edge E=Edge(A,B) and a point C, provide the pseudocode for computing the point D that is the point of E and closest to C. Then, compute the parameter t for D in the parametric representation of E.
12) Write the homogeneous matrix form representing a translation by <100,0> followed by a rotation of 90 degrees. What is the total translation of that transformation? What is its inverse matrix? Use the inverse matrix to identify its inverse translation and rotation parameters.
13) Write a test that will determine whether Edge(A,B) intersects Circ(C,r).

## Class projects

1) Implement classes for points and vectors and provide methods for the operators discussed above.
2) Implement a program that tracks the position C of the cursor as you move the mouse and computes and draws (as a small red disk) the closest point D to C on Edge(A,B). Make sure that the user can edit points A, B, and C.
3) Compute the intersection points between two edges, as their vertices are moved by the user.
4) Compute the intersection points between Edge(A,B) and Circ(C,r), as A, B, and C are moved by the user.
5) Compute the points of collision of a disk with an edge. Assume that the disk moves with constant velocity between collisions with the edge and with walls and that the collisions are elastic. Compute the new velocity after each collision. Add an interface for giving the disk an initial speed and for dragging the edge end-points.

## Research topics

1) Compute the cost (measured as the number of arithmetic operations used) in the point-in-triangle test discussed above. Assume that you will be doing this test many times for the same triangle, (A,B,C), but for different candidate points P. Modify or improve the approach to reduce the expected cost when half of the points would be in the triangle.