1

Introduction to Computer Aided Design

Design is a process to deliver a solution to a specific set of requirements with — as much as possible — an optimum outcome. Typical requirements include inexpensiveness, reliability, performance, safety, maintenance effort, and appearance, while factors typically considered in trying to achieve these requirements include strength, fracture toughness, production rate, styling, weight, assembly, and complexity.

The design starts with a concept, itself a consequence of identifying a gap in what exists versus what is needed: imagine portable telephones before the mobile phone; convenient, rapid personal transport before the automobile and highway system; and information services before the world wide web. The concept is refined, ideally with advice from stakeholders (the customers, colleagues, and so forth) and the direct experience of the designer. Iterating to the final solution, throwing out what doesn't work and including new ideas as necessary in an imperfect process, can be substantially enhanced using computers. This includes the production of examples, using the computer to rapidly command machining equipment to produce concrete examples of the desired design.

This course takes us through various aspects of computer assistance in design, introducing you to the topic in this brief quarter. Rather than teaching you how to use specific software in detail, which would be appropriate for lesser courses and would ignore the simple fact that software changes rapidly, we learn the algorithms underpinning the software: for example, how does a computer-aided drafting program scale an object? Because our time together is brief, we cannot cover how to design gears, plates, pressure vessels, computer chips, aircraft wings, the next iPhone, a replacement hip, and so on. We must make a (course) *design tradeoff*: an educated decision to deliver what is possible, not all that is desired.

What we can do is teach you how to *think* about design and computer-based tools to support it.

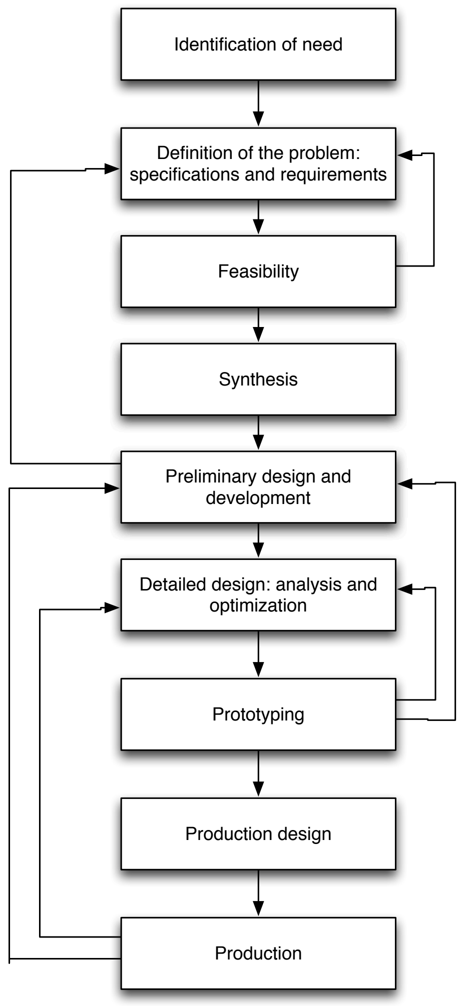
Identification of need
Definition of the problem: specifications and requirements
Feasibility
Synthesis
Preliminary design and development
Detailed design: analysis and optimization
Prototyping
Production design
Production

Figure 1.1: The design process.

An example is appropriate here. Consider how you would follow the design process for a heart valve.

Identifying the need Heart valves are failure prone in older people suffering from congestive heart failure, extended periods of high blood pressure, or genetics. It would be ideal if a replacement valve could be offered to these patients.

Definition of the problem It is important to talk to the stakeholders: the patients, the surgeons asked to install the valves in the patients, and those asked to pay for them. Many issues will be raised, but the key ones are safety, reliability, ease of installation in a difficult location (a beating heart), and utility: can the patient lead a normal life after installation?

Synthesis What form will the valve take? How will it be carried to the installation location and how will it be installed? Sewed in? Will a ball valve with a cage be "better" than a flap valve, and why? (What is better here?) Are the materials non-magnetic? Biocompatible? How do we make the device (in basic form)?

Preliminary design and development We choose a ball valve because the flap valve may embolize (cause a clot). How big should the ball be? How far should it move in its cage to allow sufficient flow, yet avoid making the valve too large to install? What direction will it fit: into the atria, or into the artery? What range of sizes are necessary? What are the flow rates required for most patients? What materials will survive for 20+ years?

Detailed design: analysis and optimization What should the surface texture of the ball be to avoid biofilm formation and engraftment of tissue? How can we improve the design to make it easier for the surgeon to install it? What can we remove to simplify the design and reduce its cost?

Prototyping Three-dimensional models printed from polymer plastic aid trial surgeons in understanding the design: what problems do they anticipate now that they have the prototype? How do we fabricate the titanium cage structure, and does it have the strength and durability we predicted in the design stages? What about the Teflon (PTFE) ball valve? Is the required surface texture fabrication process suitable? Is the entire structure fit for purpose?

Production How do we make the device, and how can we improve the design to improve production rates? How do we maintain sterility, and does the design have manufacturing and assembly steps that prevent effective sterility?

Once these kinds of questions are answered, the process can produce an effective heart valve as a solution to the identified problem. There are many other questions that occur along the way in such a device, indeed any engineering system with even a moderate amount of complexity.

Consider the following examples and think about what you would do in following the outlined design process:

- The next improved iPhone. What does the customer want? Is it really what they *need*? What tradeoffs (design decisions) need to be made?
- You have to design a vehicle that can carry you from your car or home to this class, safely but quickly. What features does it require, and how feasible is it? [1]
- You want to save the largest amount of human existence (= human lives × individual lifetime). Is it enough to save lives? What about quality of life? What is the appropriate measure? Disability adjusted life year (DALY)? What is the single most important problem you can solve to improve the global DALY? The DALY in the local community? What would your design result in?
- You need to design a car you can drive to the beach and park *in the ocean*, because San Diego parking is terrible near the beach in the summer. Is this feasible? What issues arise in the preliminary design? What details in the design are likely to arise?

[1] Is it all about the vehicle? Or are there other considerations at hand (like infrastructure)? Sometimes what seems to be the solution to the problem is not: consider fuel-cell and battery-based cars: it's not about the car.

What you will find in these contrived examples is that you will need help in performing the design, from stakeholders, from information sources, from analysis, from your own judgment as an engineer based on these results. Computers can help you with many of these steps. Alas, they cannot yet provide creativity, good judgment, nor the solution. As trained engineers you'll master these abilities in getting the most out of computers in designing solutions to society's problems.

2

Computer-aided drafting operations

2.1 Introduction

Computer-aided drafting operations are used in software to make it easy for you to make and manipulate models. The modern graphical user interface (GUI) dates from the 1970's, and though initially ridiculed, has become ubiquitous across almost all computers and related devices. This interface hinges upon some basic operations that were originally devised for drafting, and we look at these as an introduction to CAD. Nowadays, these operations tend to be implemented as system-level computer libraries like OpenGL, Vulkan, or Metal.

The objective is for you to understand how complex programs like SolidWorks, AutoCAD, CREO, and others work by looking at some simple *transformations*:

- *Scaling*: change an object's size
- *Translation*: panning or moving an object
- *Rotation about origin*
- *Rotation about an arbitrary point*
- *Reflection*
- *Zooming* (what's the difference between scaling and zooming?)

- *Clipping* (traditionally a **difficult** problem)

These operations tend to be treated mathematically as matrix operations, with the coordinate system assumed to be fixed and orthogonal throughout the operation. Advanced versions of these operations allow for coordinate changes or curvilinear coordinates: coordinates that change depending on your location. Modern software use three-dimensional representations; we will begin with two-dimensional representations for simplicity. You can later build upon these operations to devise your own transformations, and can come to understand even the most complex operations performed on CAD models by modern software.

A three-dimensional CAD model can be described by a set of coordinates $\{x(i), y(i), z(i)\}$ with i denoting the individual points (out of a total N) of the model.[1] An operation from the list above can be made on the model by using a matrix representation of the transformation which is used in multiplying against the coordinates $\{x(i), y(i), z(i)\}$ as $i = \{1, 2, \ldots N\}$.

[1] There must also be information on how these points are connected—or not—but that is for another time.

There are also **many** other operations, including curve fitting using cubic expressions, Hermitian polynomials, and Bezier curves; Boolean operations to add, subtract, and intersect objects[2]; warping and distortion; and a myriad of others. Understanding the basics now will help if you have to learn these advanced operations.

[2] Aristides AG Requicha and Herbert B Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1):30–44, 1985

2.2 *Scaling*

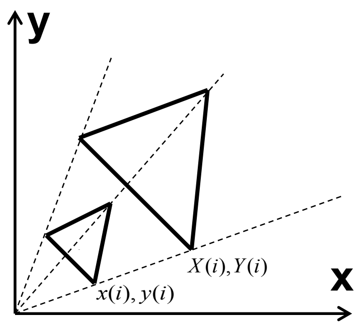

Figure 2.1: Uniformly scaling a triangle shape.

A scaling transformation increases or decreases the size of a drawing. In Fig. 2.1, the triangular shape is scaled uniformly by a scaling factor $S_x = S_y = S$. The original coordinates are denoted by $\{x(i), y(i)\}$, while the coordinates after the transformation are denoted by $\{X(i), Y(i)\}$:

$$\begin{aligned} X(i) &= S_x x(i) \\ Y(i) &= S_y y(i) \end{aligned}, \tag{2.1}$$

where S_x and S_y are the *scaling factors* along the x and y axes, respectively. The scaling in Fig. 2.1 is *uniform scaling*. Note the assumption of x and y axis orientation and orthogonality.

Equation (2.1) can be rewritten in matrix form for each i as

$$\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix}. \tag{2.2}$$

Note how the matrices have an additional equation included; this is for convenience—especially when we have several matrix operations—and denoted as a *homogeneous* coordinate representation.

2.3 *Translation*

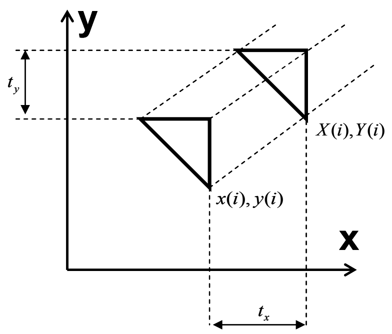

Figure 2.2: Translating the triangle shape.

The translation operation moves an object in space along a linear path without changing its shape or size. We define the distance the object moves along the x and y axes, respectively, by t_x and t_y. Figure 2.2 illustrates this transformation. The new coordinates of the object after translation $\{X(i), Y(i)\}$ are given by

$$\begin{aligned} X(i) &= x(i) + t_x \\ Y(i) &= y(i) + t_y \end{aligned}. \tag{2.3}$$

Rewriting this in matrix notation using homogeneous coordinates again we find, for each i,

$$\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix}. \tag{2.4}$$

With the homogeneous notation, addition of $\{t_x, t_y\}$ to $\{x, y\}$ is made easy via matrix multiplication.

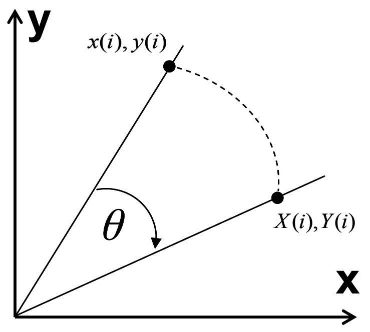

Figure 2.3: Rotating a point $\{x(i), y(i)\}$ to a new point $\{X(i), Y(i)\}$.

2.4 *Rotation about the origin*

Rotating a point $\{x(i), y(i)\}$ about the origin of the (x, y) coordinate system through a clockwise angle θ to a new point $\{X(i), Y(i)\}$ can be written as

$$\begin{aligned} X(i) &= x(i)\cos\theta + y(i)\sin\theta \\ Y(i) &= -x(i)\sin\theta + y(i)\cos\theta \end{aligned} \quad . \tag{2.5}$$

Notice we assumed the angle of rotation for all the points $i = 1, 2, \ldots N$ is the same: θ. Rewriting this in our matrix notation with homogeneous coordinates we find

$$\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix}. \tag{2.6}$$

It is fairly straightforward to find eqn. (2.5), but we need to define a

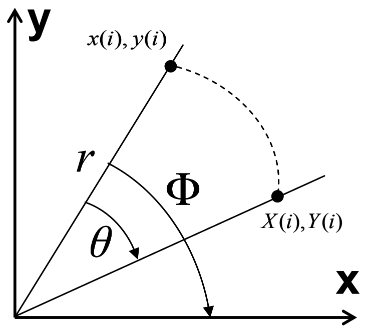

Figure 2.4: Deriving eqn. (2.5) through definition of a second angle, Φ.

second angle Φ from the line $(0,0)$ to $x(i), y(i)$ to the x axis, as shown in Fig. 2.4. Note the line has a length of r. Let's write the destination coordinates (X, Y) in terms of r, θ, and Φ:

$$\begin{aligned} X &= r\cos(\Phi - \theta) \\ Y &= r\sin(\Phi - \theta) \end{aligned} . \tag{2.7}$$

Using a typical trigonometry identity, this can be rewritten as

$$\begin{aligned} X &= \underbrace{r\cos\Phi}_{x}\cos\theta + \underbrace{r\sin\Phi}_{y}\sin\theta \\ Y &= \underbrace{r\sin\Phi}_{y}\cos\theta - \underbrace{r\cos\Phi}_{x}\sin\theta \end{aligned}\quad , \tag{2.8}$$

which gives eqn. (2.5) by eliminating r and Φ.

Note that a simple check is to use $(x,y) = (1,0)$ and then set $\theta = 90$ degrees. After rotation, this will give $(X,Y) = (0,-1)$.

Keep in mind that the *direction* of rotation around the origin for a positive value of θ is important: here we have said it is *clockwise*, but it could be (and sometimes is!) defined positive for a *counter-clockwise* rotation:

$$\begin{aligned} X(i) &= x(i)\cos\theta - y(i)\sin\theta \\ Y(i) &= x(i)\sin\theta + y(i)\cos\theta \end{aligned}\quad , \quad \text{or} \tag{2.9}$$

$$\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix}. \tag{2.10}$$

Throughout this course we will keep θ positive for clockwise rotations.

2.5 Rotation about an arbitrary point

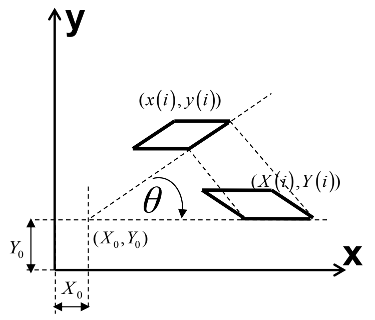

Figure 2.5: Rotation operation about an arbitrary point.

Often we would prefer to rotate an object about any point, not just the origin—for example, when the object is among others and a long way away from the origin. As shown in Fig. 2.5, this ends up being more complex, however, as a culmination of three transformations in sequence:

- The object is temporarily translated so that the desired rotation point (X_0, Y_0) moves to the origin $(0,0)$.

- The object is rotated about this origin.

- The rotated object is then translated back so that the desired rotation point is where it was when we started.

2.5.1 *Translation of rotation point to the origin*

Here is the matrix representation of the first transformation: the translation of the rotation point to the origin:

$$\underbrace{\begin{bmatrix} X'(i) \\ Y'(i) \\ 1 \end{bmatrix}}_{\mathbf{X}'} = \underbrace{\begin{bmatrix} 1 & 0 & -X_0 \\ 0 & 1 & -Y_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}_F} \underbrace{\begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix}}_{\mathbf{x}}. \tag{2.11}$$

Notice how each of the object's points $\{x(i), y(i)\}$ become $\{X'(i), Y'(i)\}$: the primes denote an intermediate location for the object's points on the way to $(X(i), Y(i))$. We define $\mathbf{x}$ as the homogeneous coordinate representation of the point (x, y) for convenience later, and do the same for $\mathbf{X}'$ to represent $(X'(i), Y'(i))$. The matrix $\mathbf{T}_F$ is the *forward translation*. Later we'll use more primes to keep track of these intermediate steps as we build the full transformation set.

2.5.2 *Rotation around the origin*

We've derived this already, and just rewrite it here with the primes indicating an intermediate step...

$$\underbrace{\begin{bmatrix} X''(i) \\ Y''(i) \\ 1 \end{bmatrix}}_{\mathbf{X}''} = \underbrace{\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}} \underbrace{\begin{bmatrix} X'(i) \\ Y'(i) \\ 1 \end{bmatrix}}_{\mathbf{X}'} = \mathbf{R}\mathbf{X}'. \tag{2.12}$$

The matrix $\mathbf{R}$ is the *rotation matrix*.

2.5.3 *Translation of rotation point back to its original location*

Now we need to translate the rotation point from the origin back to its original location (X_0, Y_0):

$$\underbrace{\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} 1 & 0 & X_0 \\ 0 & 1 & Y_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}_B} \underbrace{\begin{bmatrix} X''(i) \\ Y''(i) \\ 1 \end{bmatrix}}_{\mathbf{X}''}. \tag{2.13}$$

This translation for obvious reasons is nicknamed the *back translation*. Because we've completed all the transformations, each of the points $\mathbf{x}_i$ becomes $\mathbf{X}_i$ for all the points of the object $i = \{1, 2, \ldots N\}$. But we have to write all the operations out to make this happen:

$$\mathbf{X} = \mathbf{T}_B\mathbf{X}'' = \mathbf{T}_B\mathbf{R}\mathbf{X}' = \mathbf{T}_B\mathbf{R}\mathbf{T}_F\mathbf{x}. \tag{2.14}$$

2.5.4 *The order of the transformation matters*

We've managed to rotate the object about an arbitrary point through three consecutive transformations $\mathbf{x} \rightarrow \mathbf{T}_F\mathbf{x} \rightarrow \mathbf{R}\mathbf{T}_F\mathbf{x} \rightarrow \mathbf{T}_B\mathbf{R}\mathbf{T}_F\mathbf{x} \rightarrow \mathbf{X}$. *If we switch the order of these transformations, we would not end up with the same result: transformations generally do not commute*.

For example, suppose we swap $\mathbf{R}$ and $\mathbf{T}_F$ in defining $\hat{\mathbf{X}} \equiv \mathbf{T}_B\mathbf{T}_F\mathbf{R}\mathbf{x}$. The operations $\mathbf{T}_F$ and $\mathbf{T}_B$ cancel each other out, and we'd be left with $\hat{\mathbf{X}} = \mathbf{R}\mathbf{x}$, rotation about the origin, not the point (X_0, Y_0); generally $\hat{\mathbf{X}}$ will not equal $\mathbf{X}$.

2.6 *Reflection*

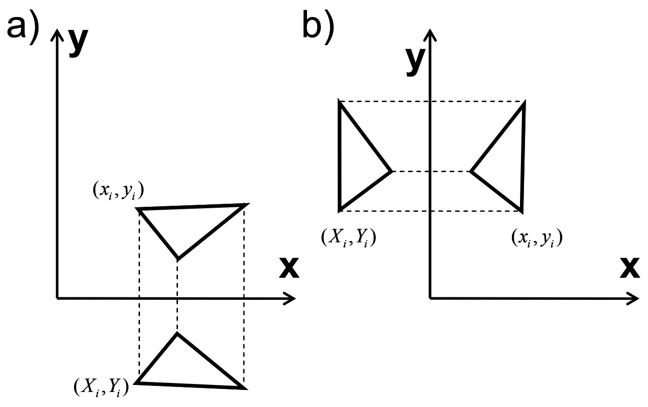

Figure 2.6: Reflecting an object composed of points $(x(i), y(i))$ to new points $(X(i), Y(i))$ via the (a) x and (b) y axes.

Often there is a need to construct new objects via reflection. Here we consider simple reflection through either the x or y axes, as illustrated in Fig. 2.6; there are many other forms of reflection that one could devise, including mirroring through a point, through an arbitrary line, or even curved lines or surfaces. Some of these can produce configurations that destroy the original topology of the object; we'll ignore this for the sake of brevity.

The reflection transformations for the x and y axes are, respectively,

$$\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix} \tag{2.15}$$

and

$$\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix}. \tag{2.16}$$

Notice that these are equivalent to scaling transformations with $\{S_x, S_y\} = \{1, -1\}$ and $\{S_x, S_y\} = \{-1, 1\}$, respectively.

2.7 *Zooming*

The zooming transformation is typical for many GUI-based software programs on computers and particularly smartphones, but is essential in CAD. It combines scaling, translation, and clipping in a series of steps, somewhat similar to rotation about an arbitrary point.

However, we will need to define the size of the screen so that we can relocate the zoom point to the center of the screen.[3] We define the screen size for simplicity's sake in terms of the object's units as L_x, L_y for the length of the screen along the x and y axes.

[3] We will ignore the need to determine the scale of the object in terms of the resolution of the screen or resolution and object dimension-independent units, so-called "dips" or density-independent pixels.

The zooming transformation works in the following way:

- Translate the zoom point (X_0, Y_0), called point P, to the origin, a forward translation transformation.
- Scale the object to the desired size.
- Translate the zoom point $P = (X_0, Y_0)$ to $(L_x/2, L_y/2)$.
- Clip the points lying outside the "window", a rectangular box defined by corners $(0,0)$ and (L_x, L_y). Also, clip the *lines* connecting these points at the window boundaries.

It is very important to note that clipping is complex for two-dimensional objects and was a significant challenge to three-dimensional object representation on computers for decades[4]. A full examination of the topic would require a year or more of effort.

[4] James D Foley, Andries Van Dam, et al. *Fundamentals of interactive computer graphics*, volume 2. Addison-Wesley Reading, MA, 1982; and John F Hughes, Andries Van Dam, James D Foley, and Steven K Feiner. *Computer graphics: principles and practice*. Pearson Education, 2013

The first transformation is essentially similar to the forward translation transformation we have already seen: the translation of the rotation point to the origin:

$$\underbrace{\begin{bmatrix} X'(i) \\ Y'(i) \\ 1 \end{bmatrix}}_{\mathbf{X}'} = \underbrace{\begin{bmatrix} 1 & 0 & -X_0 \\ 0 & 1 & -Y_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}_F} \underbrace{\begin{bmatrix} x(i) \\ y(i) \\ 1 \end{bmatrix}}_{\mathbf{x}}. \tag{2.17}$$

We then scale the object by S_X and S_Y along the x and y axes, respectively:

$$\underbrace{\begin{bmatrix} X''(i) \\ Y''(i) \\ 1 \end{bmatrix}}_{\mathbf{X}''} = \underbrace{\begin{bmatrix} S_X & 0 & 0 \\ 0 & S_Y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} X'(i) \\ Y'(i) \\ 1 \end{bmatrix}}_{\mathbf{X}'}. \tag{2.18}$$

Then the translation of the origin to the center of the window, at

$(L_x/2, L_y/2)$:

$$\underbrace{\begin{bmatrix} X(i) \\ Y(i) \\ 1 \end{bmatrix}}_{\mathbf{X}} = \underbrace{\begin{bmatrix} 1 & 0 & L_x/2 \\ 0 & 1 & L_y/2 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}_{1/2}} \underbrace{\begin{bmatrix} X''(i) \\ Y''(i) \\ 1 \end{bmatrix}}_{\mathbf{X}''}. \tag{2.19}$$

Putting them together via substitution, we have $\mathbf{X} = \mathbf{T}_{1/2}\mathbf{S}\mathbf{T}_F\mathbf{x}$. Again, the order matters.

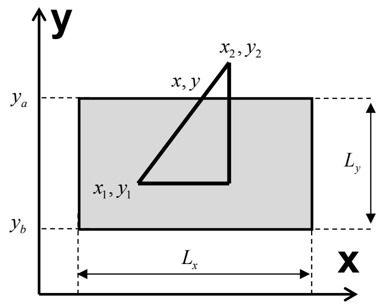

Figure 2.7: Clipping a line from a point (x_1, y_1) of the object in the window to (x_2, y_2) outside it.

The last step is the clipping, which we illustrate here via a simple example in lieu of a more general algorithm[5], which would be difficult to explain in the time available. Figure 2.7 illustrates the determination of where the line from (x_1, y_1) to (x_2, y_2) reaches the edge of the window; the latter point is presumed to be outside the window.

[5] Mike Cyrus and Jay Beck. Generalized two-and three-dimensional clipping. *Computers & Graphics*, 3(1):23–28, 1978

If we look at the slope of the line, $(y - y_1)/(x - x_1) = (y_2 - y_1)/(x_2 - x_1)$, and when $y = y_a$, we are at the edge of the window. The point (x_2, y_2) is replaced by the intersection point defined by $y = y_a$ and, by substitution and solving for x, $x = x_1 + (x_2 - x_1)(y_a - y_1)/(y_2 - y_1)$.

2.8 *Three-dimensional graphics*

Almost universally these types of operations are conducted on fully three-dimensional objects. Simple operations can be represented by expanding the two-dimensional representation to three dimensions by including the z axis. For example, the scaling operation becomes $X = S_x x$, $Y = S_y y$, $Z = S_z z$; in matrix notation this is

$$\begin{bmatrix} X(i) \\ Y(i) \\ Z(i) \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ z(i) \\ 1 \end{bmatrix}. \tag{2.20}$$

Just like before, when $S_x = S_y = S_z = S$ then we have *uniform scaling*.

Translation can likewise be rewritten

$$\begin{bmatrix} X(i) \\ Y(i) \\ Z(i) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ z(i) \\ 1 \end{bmatrix}, \tag{2.21}$$

and reflection is similar:

$$\begin{bmatrix} X(i) \\ Y(i) \\ Z(i) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \\ z(i) \\ 1 \end{bmatrix}, \tag{2.22}$$

for mirroring along the x–z plane (note that we mirror via planes in three dimensions as an analogy to mirroring via a line in two dimensions). The plane represented in the mirroring operation is labeled by the axes that do *not* have the sign reversal.

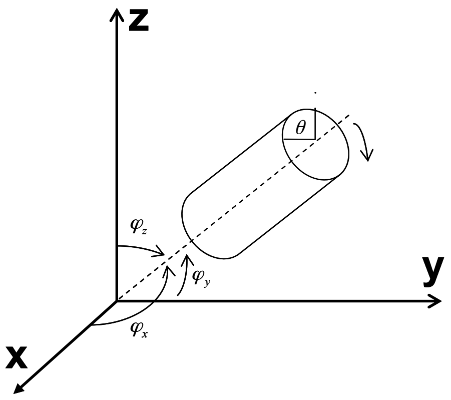

Figure 2.8: Rotating an object about a fully three-dimensional object about an axis that extends through the origin: using direction cosines to define the axis.

Rotation is more complex, and can be defined by several different schemes in three dimensions. *Direction cosines* can be used to uniquely define a rotation of a three-dimensional object as long as the axis of rotation goes through the origin. We define ϕ_x, ϕ_y, ϕ_z as angles from the x, y, and z axes, respectively, to the axis of rotation that we intend to rotate the object about as shown in Fig. 2.8. The *direction cosines* are $n_1 = \cos\phi_x$, $n_2 = \cos\phi_y$, and $n_3 = \cos\phi_z$: we make a small change in notation noting that $x \sim 1$, $y \sim 2$, and $z \sim 3$. It is worth noting that $n_1^2 + n_2^2 + n_3^2 = 1$.

The rotation matrix given these direction cosines can be written as

$$\mathbf{R} = \begin{bmatrix} n_1^2 + (1 - n_1^2)\cos\theta & n_1 n_2(1-\cos\theta) + n_3\sin\theta & n_1 n_3(1-\cos\theta) - n_2\sin\theta & 0 \\ n_1 n_2(1-\cos\theta) - n_3\sin\theta & n_2^2 + (1-n_2^2)\cos\theta & n_2 n_3(1-\cos\theta) + n_1\sin\theta & 0 \\ n_1 n_3(1-\cos\theta) + n_2\sin\theta & n_2 n_3(1-\cos\theta) - n_1\sin\theta & n_3^2 + (1-n_3^2)\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

If the axis of rotation does not go through the origin, it and the objects must first all be translated so that it does, and then the rotation may be made.

Note that if the rotation axis is the z axis, then n_x and n_y are equal to one while n_z is zero, producing a rotation matrix

$$\mathbf{R} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (2.24)$$

analogous to the rotation matrix for the two-dimensional system shown in eqn. (2.6).