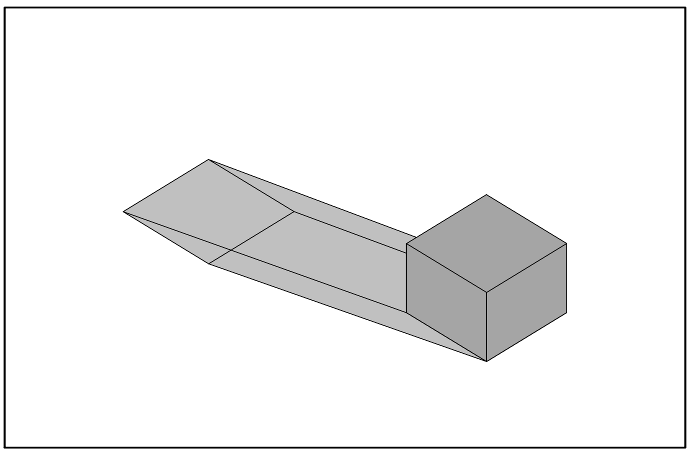
Visual Problem Solving with Maple

A Text for Teachers

Paul Eakin and Carl Eberhart
Department of Mathematics
University of Kentucky



July 16, 1997

Contents

1 Introduction

1.1 A Lament

Teacher: "William, you are not doing those additions correctly."
William: "Thats the way I do them."
Conversation reported by Dr. Don Coleman when relating his experience as an elementary school teacher in 1980

Our Mathematics has been over five millenia in develoment and is unrivaled among human constructions for its unity, consistency, and utility. Nevertheless it is percieved by many as a disconnected collection of arcane formulas, symbols, and processes whose principal use is the production of numbers, formulas, or symbols called 'answers'. A 'mathematics problem' is typically expected to be stated in a telegraphic style, and produce a single correct answer. The object of mathematical study seems to be learning which steps to apply in a particular situation to produce the answer. Significant weight is attached to the speed and accuracy with which one can produce the answer. 'Problems' tend to be viewed as things found at the end of a chapter or section in a text book or on a test. 'Working a problem' is a task that takes at most a few minutes of locating a similar exercise in the text and then matching the steps employed to solve that problem to the current problem. The test of whether one's answer is correct is that it agrees with the table of answers in the back of the book, or the teacher's answer. Getting the same answer as these oracles 75% of the time is as much as could be expected from an 'average' student.

Viewed this way William's approach is altogether reasonable. His algorithm gave the correct sum of three, three digit numbers most of the time. Since mathematics is simply the application of algorithms to symbols there was no obvious reason why the official algorithm is preferable to the one he devined and could apply quickly, consistently, and accurately.

There is no report on whether Dr. Coleman's admonition had an influence on William's view of mathematics but as measured by the percentage of college freshmen with the same outlook, over a decade of exposure to the same message, delivered by highly qualified, motivated teachers has not resonated with many. These students, many of whom have made good grades in algorithmic, 'plug and chug' courses emerge from a dozen year's study of mathematics with problem solving skills not even remotely commensurate with the investment made in their education. Further, they leave with a view of the discipline and the level to which the average person can master it not so far removed from that of ancient Egyptians peasants for whom priests reconstructed the boundaries of their fields after the floods of the Nile. While obviously imposing career limits on adults who hold them, such attitudes are readily passed on to the next generation by many parents who are unable to assist and encourage their children in studies of mathematics beyond elementary arithmetic or algebra.

1.2 Part of the problem and a remedy

William had no objective criterion for assessing whether his solution was better or worse than any other, other than by comparing it with the 'official' answer. He could work so fast that he got more correct answers than most of his peers so reinforcement from them was unlikely to cause him to re-assess his approach. Nor, one suspects, was input from parents. One may note, however, that if William had invented a new fingering for a musical instrument with the same properties as his mathematical algorithm (80 percent correct) he and his peers would have very quickly determined that the approach was in need of further development. Their common understanding of how some things are supposed to sound provides the basis for an objective assessment. Our object here is to bring something akin to that type of reality to the exercise of elementary mathematics, by providing experience in creating and analyzing problems whose statements and/or solutions are extrinsic to formal mathematics and are phrased in terms which can be appreciated and even assessed independently of the mathematics required to model or solve them. The extrinsic character allows students to guage the effacy of their understanding progress toward solutions and allows peers and parents to provide encouragement and criticism in the venaculor of the context, even when of they have no facility with, or even understanding of the mathematics.

An admittedly incomplete, though useful metaphor is music. Parents and colleagues to whom musical notation and theory is as impenetrable as any page of mathematics are nevertheless able to appreciate, encourage, and to some extent criticize student music. Student musicians themselves have a reasonable feeling for their progress toward 'getting it right'. An analog of this in mathematics occurs when a visual object, such as a picture or animation is produced which can be viewed, appreciated, and critiqued by an 'audience'. This is one of the goals of our visual approach in this text which we call Visual Problem Solving.

In Visual Problem Solving, we study the development and analysis of problems within a context in which a spectrum of explorations are not only feasible but natural. Results of explorations must be expressible in a visual format which can be appreciated independently of the underlying mathematics and through which a 'lay' person can assess in a meaningful way the progress it represents toward the exploration's stated objective. While emphasis on the visual serves the objective of introducing what for lack of a better term we forthrightly call a 'performance' dimension, it's more fundamental purpose is the linking of geometric intuition, the most powerful ananlytic tool most people possess, to systematic problem solving.

In this text, we will introduce and develop a number of problem contexts, each of which satisfies the following:

- The context can be modeled and is amenable to serious exploration using elementary arithmetic, algebra, trigonometry, and plane and analytic geometry. We eschew calculus here but are perfectly willing to use formulas for the volume of a cone and such basic notions as position, velocity, acceleration from calculus.

- The contexts, or at least solutions to explorations they engender must be amenable to highly visual explanation or representation which can be appreciated with no understanding of the mathematical details. Such an observer should be able on the basis of the visual presentation to make an informed decision on whether the work presented reprsents progress toward a solution to the problem under investigation.

- The explorations suggested by the context should lie in a broad spectrum from elementary descriptive projects to high quality student research areas.

In general our approach is to introduce a context and develop a mathematical model which contains or is extensible to a family of explorations. Typically, we initiate a primary investigation and ask students to complete or extend parts of it over a fixed period of time. Except for projects done for individual evaluation, students are encouraged to work together in groups. On all projects students are encouraged to confer with faculty frequently on their progress. Modeling the way technical investigations develop and proceed in practice, assistance is readily given at the very general conceptual level and the very specific, technical level. In addition, we provide a literature of investigations which can be used both for general ideas and directions of inquiry.

When the mathematics and vocabulary of a context are well understood, some attention is directed to its further development/extension through application of some general problem development techniques. This introduces the art, and science of problem context development in a hands on process. The general message is that while new, fruitful problem areas may spring from pure inspiration, by far the largest part of them will be systematically be derived from previous efforts and will inherit many of the features that made the source productive. This is particularly true of problem areas ultimately designed for young students - those in which the mathematical prerequisites are very modest. We will find that rather than making the development task more difficult, well-chosen constraints such as our 'no calculus' and the visual requirement for our investigations provide both a focus and a wealth of problems to address.

The final stage in the development cycle of a context is the adaptation of its most recent fruit to the classroom. This process is a prism which attempts to carefully resolve the spectrum of (now) well understood possible investigations into a suite of interrelated tasks suitable for individuals or groups of students of differing levels and interests to pursue with reasonable expectation of success. Student problems will fall into some broad categories:

1.3 Levels of problem solving.

- **Exploratory**: The development of the problem context produces software (in this case Maple worksheets) which provide a visual insight into the problem which can be appreciated independently of the underlying mathematics. These can be developed into student activities which allow them to explore the problem area and gain an appreciation of the question and what can be done with the elementary mathematics they are learning.

- **Descriptive**: At a higher level, students do not simply use the software but modify and extend its geometric content, learning to precisely (matheatically) describe and visually represent objects in the context. (e.g. polygonal objects, filling in escher pictures, etc.)

- **Manipulative**: At a still higher level students not only create visual objects but learn to manipulate them with mathematics and to represent the results visually. (e.g. translation, dialation).

- **Beginning problem solving**: A higher level than "pure" manipulative. Students articulate, solve, and implement visual representation of desired manipulations (e.g. translate/rotate, break apart, etc)

- **Structured elementary problem solving**: These would typically involve a brief sequence of elementary problem solving steps, with the results of one step needed for another. Problems are presented in a structured format, typically as extensions of example investigations.

- Elementary Problem Solving: Problems are presented in an open format but within the current context. These are typically extensions of example investigations which may be achieved through different routes. These would be "straightforward" to the teacher but would require several hours work for a student or student group to formally "solve" and an additional number of hours to complete and prepare a presentation of the solution. These generally require simplifying assumptions which students should identify and note their role in the solution.

- **Intermediate problem solving**: These differ from elementary primarily in the level of presentation to the student. They need not fit solely within the current context but should "map" into it with some thought. They may not have been worked in complete detail by the teacher but the former would have a firm conceptual outline of a solution and know that it is within the ability of the student. The principal difference between an intermediate problem and an elementary problem is the amount of formal direction given with the problem. Intermediate problems should be given with the expectation that even good students will require some assistance. This assistance may in total reduce the problem to an elementary problem had it been given formally, at the outset. However it is given in response to specific student questions which reflect work done on the problem and hence is substantially "generated" by the student(s).

- **Investigative problem solving**: This differs from intermediate problem solving in primarily in the format of the question. Here the teacher knows by experience that the question is reasonable and is confident that he/she can make a contribution. The questions are typically phrased as "What happens if..?", "Can this be done if ..?", etc. Such problems are given to students who have successfuly completed an intermediate project with some distinction and who clearly have a "feel" for the context. Such investigations will often lead to mathematical problems well beyond the confines of the elementary model. The identification and clear statement of such a question, together with numerical/arithmetical or graphical evidence would be an excellent outcome of such an investigation.

- **Advanced problem solving**: Students undertake the investigation of a problem which has emerged from some of the contexts studied which the teacher knows to be difficult and probably does not know how to solve or can do so only partially or by employing advanced mathematics. It will often happen that today's advanced problem is tomorrow's elementary problem - in some sense that conversion is the ultimate goal of the project. Such projects would be undertaken only by students who have successfully completed an investigative problem solving project and would preferably be done in a mentoring situation with a mathematics teacher or professor.

1.4 Use of the Maple language and worksheets.

Our principal means for integrating images and mathematical symbols and communicating them to others is the computer algebra system Maple. We might just as well have chosen the equally powerful Mathematica but for the historical fact that when the University of Kentucky first went looking for a local computer algebra system Maple offered more attractive terms.

In order to follow this program of study the reader will need access to a current version of Maple V. No experience with Maple or any other similar program is assumed. The course materials were developed in Maple V, Release 4 and do take advantage of features such as of that version which are not available in Release 3. *This text is a collection of Release 4 worksheets which will not run under Release 3*, although you may be reading a hardcopy version. Most of their content can be run under Release 3. However this is likely to be very frustrating for the inexperienced user.

The course materials were developed using computers with Intel Pentium processors, with 16 to 32 Mb memory. Most of the activities will run quite well on smaller, slower machines but some of them, particularly the three-dimensional animations may be slow or require more resources than some of those machines have available. In such cases an understanding of the material will readily allow the user to adapt the material to the local environment.

Since the program Maple is central to this presentation, however our approach we begin with a brief *hands-on* introduction to a few Maple commands and concepts. These will allow us to promptly begin to use it for our problem solving and communications purposes. As the course progresses we will introduce additional tools as they are needed. The reader should immediately begin to explore beyond the very limited scope of commands we formally introduce.

Maple material appears in text cells like this one or in in execution cells like the next:

```
This is an execution cell - you can tell by
the ">" prompt.  Maple expects
expressions entered in at the command prompt to be valid Maple
commands - which this certainly isn't.  If you press the "enter" key
with the cursor in this cell then Maple will try to interpret it.
Since this isn't a valid command,
Maple will return an error message.
```

```
Syntax error, missing operator or `;`
```

In the course of this text we will often discuss words or commands which have meaning to Maple and will often provide expressions which can be entered at the prompt. One way we can do this is to provide the input cell with the command already entered, leaving it to the reader to simply press the "enter" key to have Maple execute it. For instance:

```
2+7;
```


Note that in such cases we usually leave an empty command line or two. To execute the command above one simply places the cursor anywhere in the cell and presses "enter". Maple will execute the command and move the cursor to the nect command prompt. If thre are many lines of text intervening then the user has to scroll back, and search for the results of the calculation.

In many cases we will not leave commands sitting at command prompts waiting to be inadvertantly executed or modified but will rather place them in the text where the reader can copy them into a command cell if he/she wishes. Such commands will be recognizable by the font they are in and the fact that they are terminated by a semicolon ";". Thus for instance if plot(sin(3*x),x=-Pi..Pi); is copied and pasted into the command line below, an "enter" will result in a graph of sin(3x) on the interval $[-\pi, \pi]$.

2 Chapter 1 First steps – Drawing boxes

2.1 What is visual Problem solving?

Visual problem solving is concerned with problems whose statements or solutions (hopefully both) have a visual character which can be appreciated and even assessed without reference to the underlying mathematics. This is analogous to the manner in which we can appreciate music without reference to or even understanding of its

creator's tools - **the fact notwithstanding** those with such knowledge can appreciate it at a higher level. In the course of this study we will find that there is an incredible variety of problems of this nature and that there are techniques for developing families of them from basic contexts. In this introduction to the subject we will primarily restrict our attention to problems which can be analyzed with the non-calculus mathematics and basic science studied in middle and high school. However, since part of what we will be doing is generating new problems from old we must be prepared to find ourselves formulating questions in that vocabulary whose solution requries new tools. We will celebrate when we do for the quest for problems not solvable within a science is the most basic process in by which it is extended. problem solving is about marshalling the intellectual tools at our disposal to attack a problem.

The expected result of a serious attack on a problem is a *contribution* to its solution. A contribution may well fall short of a complete solution but it will always present one's *formulation* of the problem, one's approach to *analyzing* the problem, and the *results* of one's analysis. In keeping with our visual approach at least one of these **components will contain elements of a visual nature.** In this course our contributions will always take form on a Maple (release 4) worksheet.

Maple is a computer program (sometimes called a symbolic manipulation program) which can generate and manipulate algebraic, numerical, and geometric objects. It has a language of its own (the Maple language) through which it can be directed to perform virtually any mathematical or geometric operation. More over this language is *extensible* which means that, like any natural language, it can be used to define new words which the program can recognize and new symbols which it can manipulate. A Maple Worksheet is an electronic file containing text and inserted images (which have meaning only to people), Maple commands (words) which direct the program to perform certain operations, and output which is text, images, or other constructs generated by the program in response to the commands. In general the responses (also called the output) are not needed if one has the commands since they can always be recovered from the latter. Such a sequence of commands is typically a small amount of text which can readily be communicated by electronic mail. This is the form in which all work in this course - all contributions - will be made, and the primary form of correspondence among the participants in this course. Since this is the case our first order of business will be to begin to learn how to use Maple as a *mathematical communications* tool. We learn to do this by doing it. We start with the geometric vocabulary.

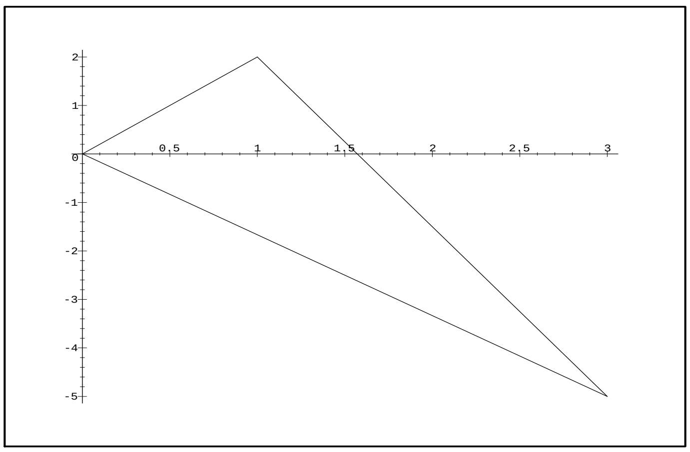
2.2 Mathematical Boxes

We assume that the reader is familiar with the fact that any two mutually perpendicular lines in the plane or three mutually perpendicular number lines in space define a *cartesian coordinate system* relative to which every point in the plane or space has a unique name, called its *coordinates*. Points in the plane or space may have other names such as "X","Lexington", or "Detroit", etc. and for a particular purpose these may be far more descriptive than a list of coordinates. The importance of coordinates is that that algebraic relationships among the coordinates of (collections of) points fully express geometric relationships among the points themselves. This means that we can use algebra to express and investigate geometric relationships (and visa versa). Moreover, since computers can "understand" and "communicate" algebraic relationships, this provides both a path to the physical/geometric/visual worlds we propose to explore and a vehicle take us at least part of the way there. The coordinates of a point are a list so our first task is learn about lists in Maple.

In Maple, lists are sequences of objects enclosed in brackets [ ] . The order of the objects in a list is important. Thus the lists [1,2,3] and [2,1,3] are different and the lists [a,b,c] and [d,e,f] are equal (as lists) only if a=d, b=e, and c=f. Note also that, to Maple, only the first of [1,2,3], {1,2,3}, and (1,2,3) is a list. Our initial Maple command to plot points in space is "plots[polygonplot3d]" which is a command which accepts lists of points and plots the (spatial) polygon with those points as its vertices. Thus if we want to create a visual representation of the spatial triangle with vertices [1,2,3], [4,1,5] and [7,8,3] we enter the following at the command prompt ">" . plots[polygonplot3d]([[1,2,3],[4,1,5],[7,8,3]]);

The analogous command which displays a polygon in the plane is "plots[polygonplot]."

```
plots[polygonplot]([ [ 0,0], [1,2],
[3,-5]]);
```

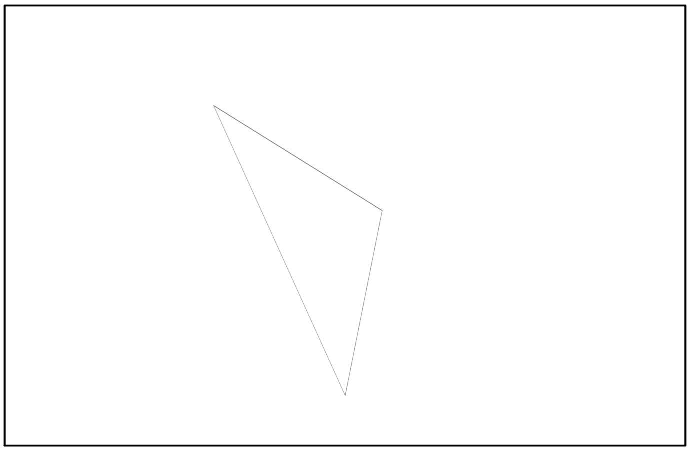


The "plots" part of these command tells Maple to look in the "package" of commands called "plots" for the word polygonplot3d . We can first tell Maple to "load" all of the words in the plots package into the current session by the command with(plots); after which it will recognize only the polygonplot3d part.

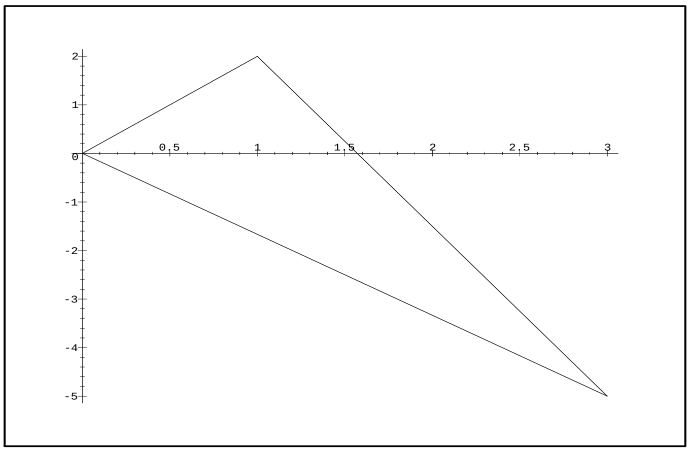
```
with(plots);
```

[*animate*, *animate3d*, *changecoords*, *complexplot*, *complexplot3d*, *conformal*, *contourplot*, *contourplot3d*, *coordplot*, *coordplot3d*, *cylinderplot*, *densityplot*, *display*, *display3d*, *fieldplot*, *fieldplot3d*, *gradplot*, *gradplot3d*, *implicitplot*, *implicitplot3d*, *inequal*, *listcontplot*, *listcontplot3d*, *listdensityplot*, *listplot*, *listplot3d*, *loglogplot*, *logplot*, *matrixplot*, *odeplot*, *pareto*, *pointplot*, *pointplot3d*, *polarplot*, *polygonplot*, *polygonplot3d*, *polyhedraplot*, *replot*, *rootlocus*, *semilogplot*, *setoptions*, *setoptions3d*, *spacecurve*, *sparsematrixplot*, *sphereplot*, *surfdata*, *textplot*, *textplot3d*, *tubeplot*]

```
polygonplot3d([[1,2,3],[4,1,5], [7,8,3]]);
```



```
polygonplot([ [ 0,0], [1,2], [3,-5]]);
```

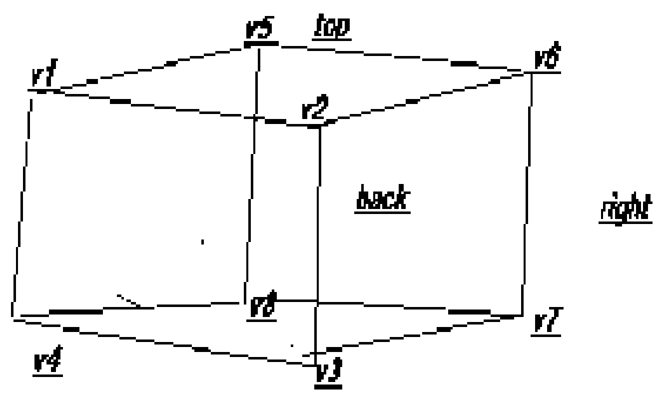


Note the large list of interesting "visual" words which were returned uopn execution of the with(plots); We will visit a number of them in this study. If you are interested in what a word means you can use the Maple "help" system in a number of ways. For the present the command help(polygonplot3d); illustrates the most obvious use of the help system.

```
help(polygonplot3d);
```

Although loading the plots package relieves us of a few keystrokes in writing down our work we will typically forego this for the far greater advantage of portability and readability of our instructions. We will continuously find that we want to go back to old worksheets and reuse pieces of code via "cut and paste" operations. Rather than having to carefully re-read such material to see if perhaps it depends on packages having been loaded earlier in the worksheet.

The ability of Maple to draw polygons in space provides us with the means to have Maple create images of any spatial object which consists of polygons. Lets begin with one of the most familiar objects, a cube. A cube has eight *vertices, and six faces, each defined by a sequence of the vertices. Since Maple can work with symbols we can represent the vertices symbolically, say with symbols v1, v2, .. and the faces (top, bottom, left, right, front, and back) as sequences of the vertices. First we sketch our cube, labeling its vertices and faces with these symbols and then "read off" the lists we need to communicate with Maple. It is very helpful to use a hand-drawn sketch or diagram. (You can draw these with for example "paint" and cut and paste the sketches into the worksheet.)*



Notice that all of this so far is hand drawings and text. We have not communicated anything to the Maple program which "understands" a limited vocabulary and has no ability at all to understand pictures. We enter Maple commands at the command prompt ">". We want to describe to Maple each of the faces of our box using terms that have meaning (the same meaning) to both us and the computer.

2.2.1 *Definition: A* **polygon** *is an ordered list of points lying in a plane in space.*

2.2.2 *Definition: An* **n-gon** *is a list of n different points. Two lists correspond to the same polygon if a circular permutation of one or yields the other or a reverse listing of the other.*

The connection between our usual geometric view of a polygon and this form is that the list which describes it may be thought of as the order in which the verticies are passed as one "walks" along the boundary of the polygon along a path which passes each vertex exactly once. The fact that there are numerous paths which so this corresponds to the fact that there are numerous lists which represent a particular polygon. Thus, for instance, the front face of our sketch is the list [v1, v2, v3, v4]. It is also [v2,v3,v4,v1] and [v3, v2,v1,v4].

2.2.3 *Problem: Exactly how many different lists of vertices corespond to any one n-gon?*

It is posible and convenient for the same object to have more than one name in Maple. Thus for instance, with reference to the above diagram we may want to refer to the ploygon whose vertices are v1,v2,v3,v4 as the "front face" or simply to give it the name "front". Similarly there is a "back", "top", etc, the latter being names for other polygons which are easy to refer to and remember. The way we tell Maple that we want to refer to the list [v1,v2,v3,v4] as "face" is via assigned equality.

```
front:=[v1,v2,v3,v4];
```

$$front := [[2, 0, 3], [2, 1, 3], [2, 1, 2], [2, 0, 2]]$$

```
front;
```

$$[[2, 0, 3], [2, 1, 3], [2, 1, 2], [2, 0, 2]]$$

What we have done is create an alternate name or label for this list. We might communicate this in English with a sentence like "Let "front" designate the list [v1,v2,v3,v4]."

The same object can have any number of names.

```
Front_face_of_the_cube:=front;
```

$$Front\_face\_of\_the\_cube := [[2, 0, 3], [2, 1, 3], [2, 1, 2], [2, 0, 2]]$$

Sometimes longer names such as the above are very useful in the development of more complicated sets of instructions.

We will repeatedly use this idea of assignment for labels: alternate convenient names for certain object which will have the same meaning to us and the computer program. We will, at the same time need to understand the difference between an equation (which uses the symbol "=") and an assigned equality which uses the symbol ":=". Consider the following:

```
joe:=1;
```

$$joe := 1$$

```
sam=10;
```

$$sam = 10$$

The first states that joe has the value 1 - that Joe is simply another name for the number "one". The second is an equation rather than an equality. It does not provide any alternative interpretation for the value of sam.

```
joe;
```

$$1$$

```
sam;
```

$$sam$$

For the moment we simply need to be aware of the fact that an equation is simply two expressions separated by a equal sign. By itself it does not affect the value of either. We can write an equation involving any two experssions.

```
1=2;
```

$$1 = 2$$

Maple will, however, balk at attempts to assign values. For instance the following commands would attempt to redefine the value of 1.

```
1:=joe;
```

```
Syntax error, `:=` unexpected
```

```
1:=2;
```

```
Syntax error, `:=` unexpected
```

We will have numberous opportunities to take advantage of the fact that Maple can solve equations

```
solve(2*x+1=0,{x});
```

$$\{x = \frac{-1}{2}\}$$

If you understand why it hasn't a clue as to what to do with the following then you probably understand the difference between "=" and ":="

```
solve(x+1:=x^2,{x});
```

```
Syntax error, `:=` unexpected
```

A command to Maple is always entered at the command
prompt (">") and terminated with a semicolon (";") or a colon ":".
If a command is terminated with a semicolon then Maple displays the result of its interpretation of the command; if the command is terminated with a colon the Maple output is not displayed.

```
1+2+3+4+5;
```

$$15$$

```
6+7+8+9+10:
```

Multiple commands can be entered at the same prompt:

```
front:=[v1,v2,v3,v4];back:=[v5,v6,v7,v8];left
:=[v4,v8,v5,v1];
right:=[v3,v2,v6,v7];bottom:=[v3,v4,v8,v7];top:=[v1,v2,v6,v5];
```

$$front := [[2, 0, 3], [2, 1, 3], [2, 1, 2], [2, 0, 2]]$$

$$back := [[1,\ 0,\ 3],\ [1,\ 1,\ 3],\ [1,\ 1,\ 2],\ [1,\ 0,\ 2]]$$

$$left := [[2,\ 0,\ 2],\ [1,\ 0,\ 2],\ [1,\ 0,\ 3],\ [2,\ 0,\ 3]]$$

$$right := [[2,\ 1,\ 2],\ [2,\ 1,\ 3],\ [1,\ 1,\ 3],\ [1,\ 1,\ 2]]$$

$$bottom := [[2,\ 1,\ 2],\ [2,\ 0,\ 2],\ [1,\ 0,\ 2],\ [1,\ 1,\ 2]]$$

$$top := [[2,\ 0,\ 3],\ [2,\ 1,\ 3],\ [1,\ 1,\ 3],\ [1,\ 0,\ 3]]$$

Lets get back to our box. We (people) have a concept of an abstract box and we have a sketch which helps us visualize it. We and the computer "know" that there certain symbols, v1...v8 which we (people) call the vertices but this designation has no meaning to the computer. We and the computer know about certain lists of the vertices and have agreed on names for them. We have agreed that certain other lists have the same meaning (when thought of as polygons) but the computer knows only about the specific lists we chose. We can "ask" Maple how it is interpreting a symbol by entering that symbol as a command.

```
top;
```

$$[[2,\ 0,\ 3],\ [2,\ 1,\ 3],\ [1,\ 1,\ 3],\ [1,\ 0,\ 3]]$$

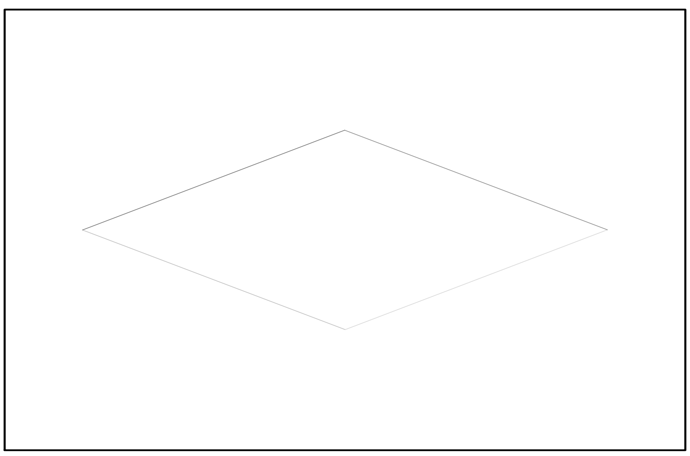
```
TOP;
```

$$TOP$$

Lower case "top" refers to the list of symbols it returned. "Top" is simply a string to which no other meaning has been assigned.

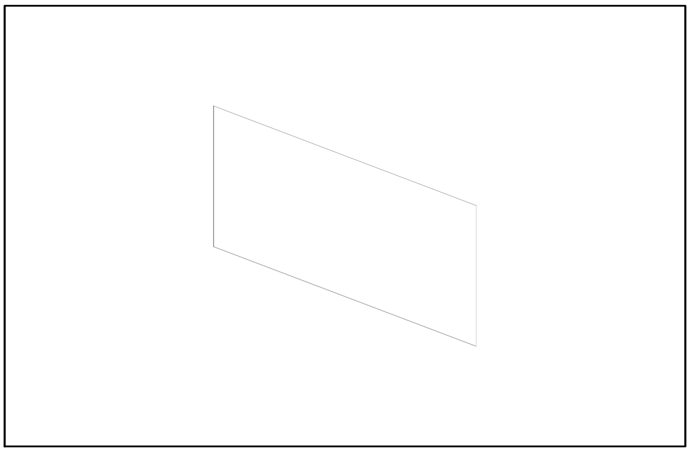
Although we can sketch and work with an abstract figure Maple can draw only draw specific figures which we specify specific completely. The polygonplot3d command works with specific, not general lists of vertices. For instance,

```
plots[polygonplot3d]([[0,0,0],[0,1,0],[1,1,0]
, [1,0,0]]);
```



produces a square but

```
plots[polygonplot3d]([v1,v2,v3,v4]);
```



doesn't know what to do with since the items in the list its given do not represent points in space, i.e. are not themselves lists of three numbers. The solution to this is to tell Maple how to interpret v1, v2, v3, v4 as points in space. This amounts to assigning these names to specific points (or more precisely to sets of coordinates for the points relative to some reference frame). This assignment is totally up to us. We might, for instance have decided that our abstract box is actually the "unit cube" in three dimensions relative to some coordinate system. In that case our sketch might have been.

Plot: ch1box1.eps

This can be interpreted as assigning values to our previously abstract quantities v1, v2, v3, v4, ... – something we now know how to do.

```
v1:=[1,0,1]; v2:=[1,1,1]; v3:=[1,1,0];
```

```
v4:=[1,0,0];
v5:=[0,0,1]; v6:=[0,1,1]; v7:=[0,1,0]; v8:=[0,0,0];
```

$$v1 := [1, 0, 1]$$

$$v2 := [1, 1, 1]$$

$$v3 := [1, 1, 0]$$

$$v4 := [1, 0, 0]$$

$$v5 := [0, 0, 1]$$

$$v6 := [0, 1, 1]$$

$$v7 := [0, 1, 0]$$

$$v8 := [0, 0, 0]$$

The computer will assign these meanings whenever it encounters the symbols v1..v8. Since they occur in our previously defined names for the faces, those names now represent specific polygons in space.

```
top;
```

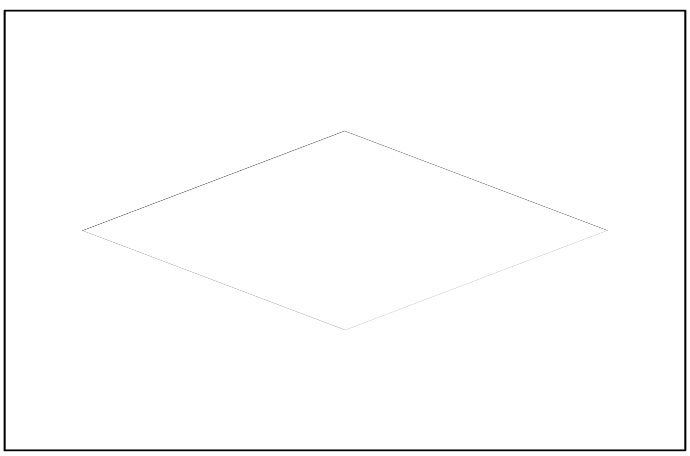
$$[[2, 0, 3], [2, 1, 3], [1, 1, 3], [1, 0, 3]]$$

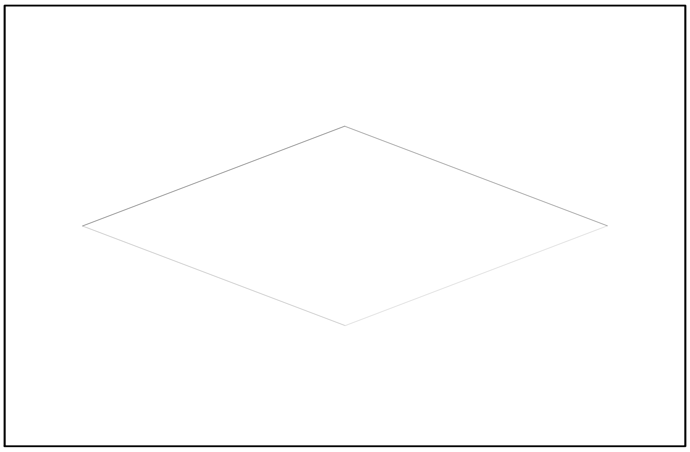
```
bottom;
```

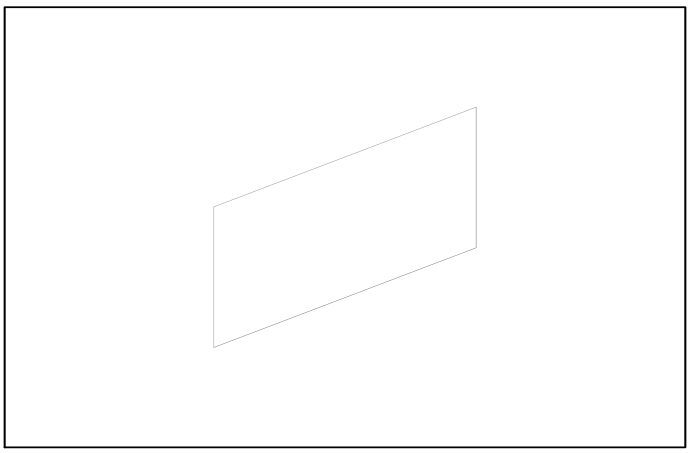
$$[[2, 1, 2], [2, 0, 2], [1, 0, 2], [1, 1, 2]]$$

Now we can ask Maple to render visualizations of these faces:

```
plots[polygonplot3d](top);
plots[polygonplot3d](bottom);plots[polygonplot3d](right);
```

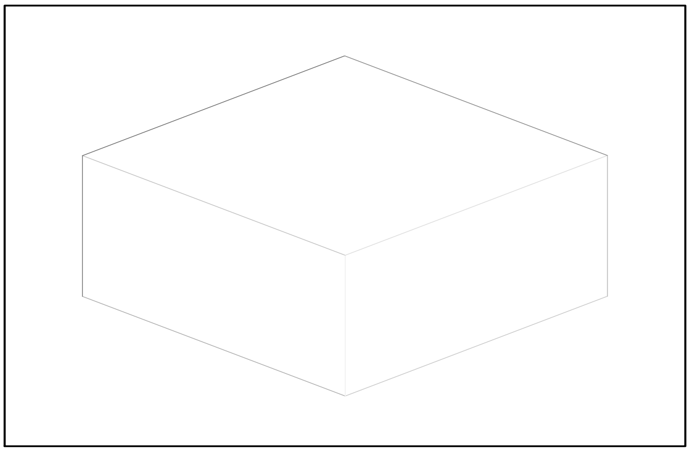






Of course this isn't satisfactory, we don't want to see the faces separately, we want them all together. To do this we simply give the command the complete list of faces which directs the program to render all of them in the same image:

```
plots[polygonplot3d]([top, bottom, left,
right, front, back]);
```



This is better but we can't identify front, back, etc. nor does the picture have the "wireframe" format of our sketch. This can be remedied in by asking for axes and altering the plotting style. Left clicking on the image encloses it in a box and creates a "tool bar" of manipulations of the image. The reader should explore the effect of each of the toolbar buttons and should note that the image can be rotated by draging the cursor across it. In addition there are "command line" options corresponding to each of the buttons on the tool bar. We will now explore several of them: style, scaling, and color which will be of immediate use. Each of these is an *option for the plots[polygonplot3d] command. The way to learn about a command and its options is through the help system. As noted earlier this can be invoked in a number of ways. When we know exactly what we are looking for the command line approach is the most efficient:*

```
?plots[polygonplot3d];
```

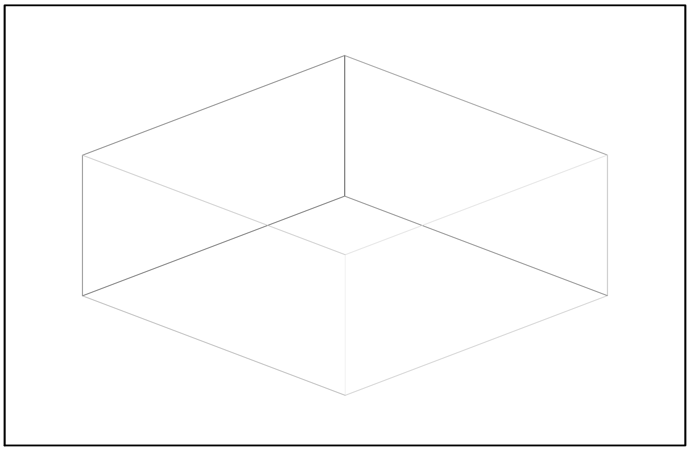
Note that the "help" page this takes us to describes the command and refers us to other help sheets for further information. We are interested in the options which we are told can be reached by the command ?plot3d[options]; or by clicking on that hyperlink at the bottom of the sheet. In either case we reach a page which lists a large number of options. Our main objective at this point is to know that the page exists and how to reach it. A few examples of the use of some of the options generally suffices to give us enough information to work out for ourselves how others are used. Note that the primary `plots[polygonplot3d]` help sheet contains some examples. These can be copied and pasted into our work session and executed to see what they do. For now we will apply some of these options to our example. First lets take advantage of our ability to assign short names:

```
box:=[top, bottom, right, left, front,
back];
```

$$box := [[[2, 0, 3], [2, 1, 3], [1, 1, 3], [1, 0, 3]], [[2, 1, 2], [2, 0, 2], [1, 0, 2], [1, 1, 2]], [[2, 1, 2], [2, 1, 3], [1, 1, 3], [1, 1, 2]], [[2, 0, 2], [1, 0, 2], [1, 0, 3], [2, 0, 3]], [[2, 0, 3], [2, 1, 3], [2, 1, 2], [2, 0, 2]], [[1, 0, 3], [1, 1, 3], [1, 1, 2], [1, 0, 2]]]$$

Now we can simply have the computer draw "box" - but we want to do this with some of the options. The first changes the plotting style from the default "hidden line" to the "wireframe" format indicated in our rough sketch.

```
plots[polygonplot3d](box,
style=wireframe);
```



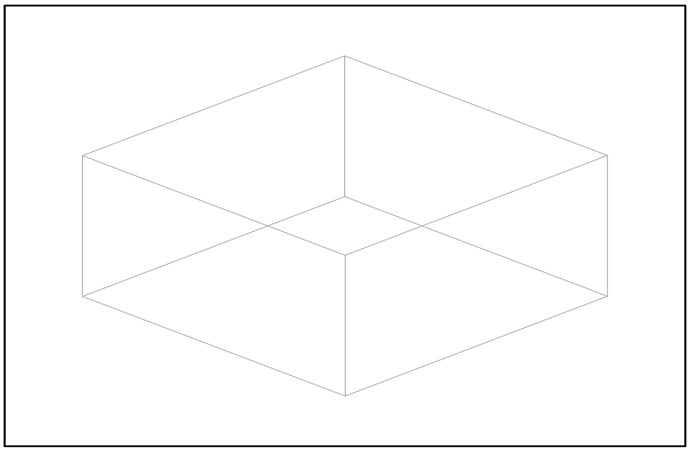
We can color the edges of the frame as we wish. The response to ?plot3d[options];refers us to plot[color]

```
?plot3d[options]
```

```
?plot[color]
```

Which gives us the following list : aquamarine, black , blue, navy, coral, cyan, brown, gold, green, gray, grey, khaki, magenta, maroon, orange, pink, plum, red, sienna, tan, turquoise, violet, wheat, white, yellow - as well as instructions for making our own colors. For now we will stuck with the "off-the-shelf" colors.

```
plots[polygonplot3d](box,
style=wireframe,color = tan);
```



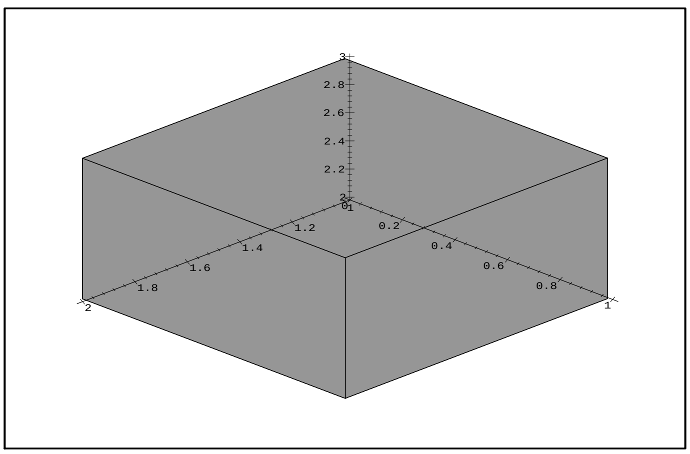
The "patch" style allows us to specify a color for the entire box.

```
?plot3d[options]
```

```
plots[polygonplot3d](box, style=patch,color =
tan);
```

The axes option allows us to include coordinate axes in different ways:

```
plots[polygonplot3d](box, style=patch,color =
green, axes= normal);
```

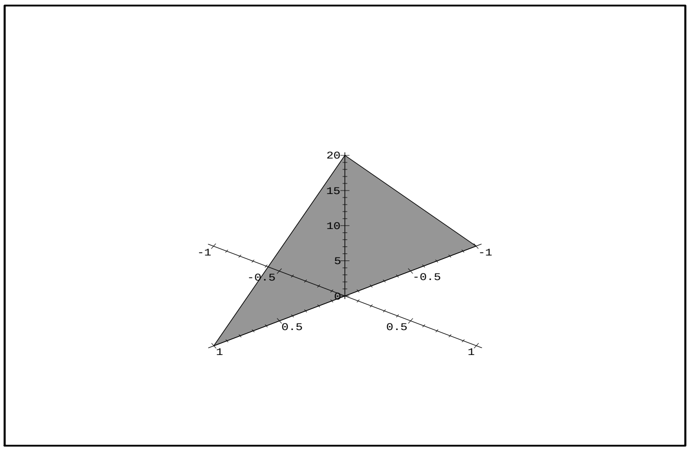


When creating a general plot Maple attempts to do the best job it can of placing the entire picture in the space available. In doing so it often scales the coordinate axes so that the image produced does not appear "to scale". The option which turns this off or on is "scaling" which is "constrained" (i.e. to scale) or "unconstrained". The symmetry of our box makes it a poor choice to illustrate this but we can readily define other objects such as the following

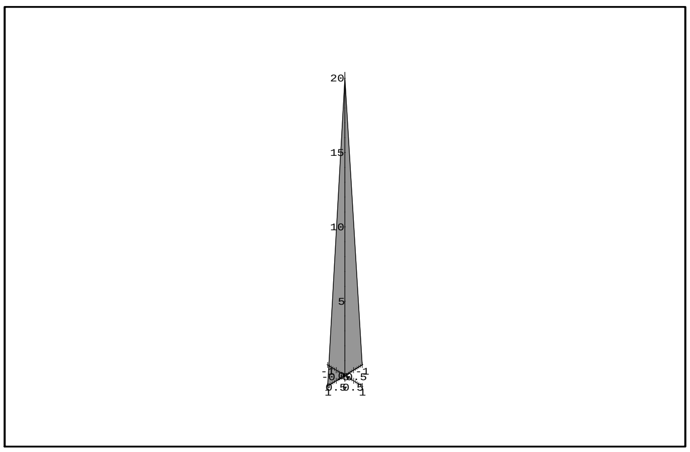
```
skinnytriangle:=[ [-1,0,0],
[1,0,0],[0,0,20]];
```

$$skinnytriangle := [[-1, 0, 0], [1, 0, 0], [0, 0, 20]]$$

```
plots[polygonplot3d](skinnytriangle,
style=patch,color = green, axes= normal);
```




```
plots[polygonplot3d](skinnytriangle,
style=patch,color = green, axes= normal, scaling=constrained);
```



An obvious disadvantage of the coloring option as used on our box to this point is that we can specify only one color in a command and therefore our box can be only one color. The solution to this is to "draw" each of the faces separately with whatever color we want and plot them together. To do this we need another command, plots[display] which displays a list of individual plots in the same picture, relative to the same coordinates. Suppose we want our box to have a blue top, yellow sides, and a purple bottom. We can separately construct the pieces with different colors:

```
bluetop:=plots[polygonplot3d](top,
style=patch,color = blue );
```

```
purplebottom:=plots[polygonplot3d](bottom, style=patch,color =
purple);
yellowsides:=plots[polygonplot3d]([left,right,front,back],
style=patch,
color = yellow);
```

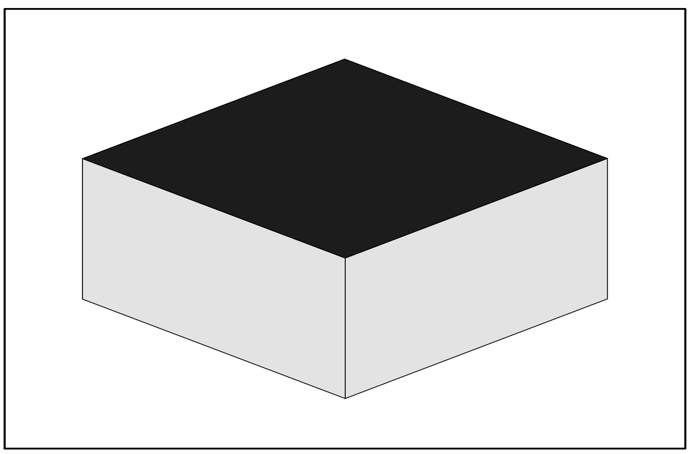
$$bluetop := \mathbf{PLOT3D}(\mathbf{POLYGONS}([[2., 0, 3.], [2., 1., 3.], [1., 1., 3.], [1., 0, 3.]]),\ \mathbf{STYLE}(\boldsymbol{PATCH}),\ \mathbf{COLOUR}(\boldsymbol{RGB}, 0, 0, 1.00000000))$$

$$purplebottom := \mathbf{PLOT3D}(\mathbf{POLYGONS}([[2., 1., 2.], [2., 0, 2.], [1., 0, 2.], [1., 1., 2.]]),\ \mathbf{STYLE}(\boldsymbol{PATCH}))$$

$$yellowsides := \mathbf{PLOT3D}(\mathbf{POLYGONS}([[2., 0, 2.], [1., 0, 2.], [1., 0, 3.], [2., 0, 3.]], [[2., 1., 2.], [2., 1., 3.], [1., 1., 3.], [1., 1., 2.]], [[2., 0, 3.], [2., 1., 3.], [2., 1., 2.], [2., 0, 2.]], [[1., 0, 3.], [1., 1., 3.], [1., 1., 2.], [1., 0, 2.]]),\ \mathbf{STYLE}(\boldsymbol{PATCH}),\ \mathbf{COLOUR}(\boldsymbol{RGB}, 1.00000000, 1.00000000, 0))$$

We have not told Maple to draw these items. What we have done is assigned names to the "plot structures", the sets of information needed by Maple to display the images. The command plots[display] accepts a list of such images and displays them simultaneously:

```
plots[display]([bluetop, purplebottom,
yellowsides]);
```

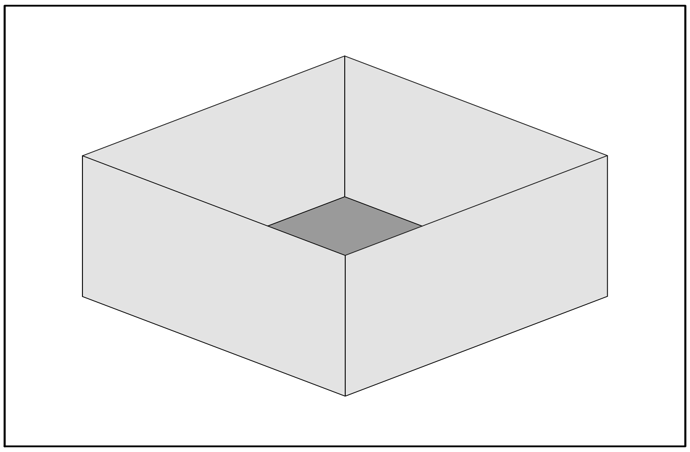


If we would like to be able to "see through" the top of our box we could draw the top in the "wireframe" style.

```
bluetop:=plots[polygonplot3d](top,
style=wireframe,color = blue );
```

$$bluetop := \mathbf{PLOT3D}(\mathbf{POLYGONS}([[2., 0, 3.], [2., 1., 3.], [1., 1., 3.], [1., 0, 3.]]),\ \mathbf{STYLE}(\boldsymbol{LINE}),\ \mathbf{COLOUR}(\boldsymbol{RGB}, 0, 0, 1.00000000))$$

```
plots[display]([bluetop, purplebottom,
yellowsides]);
```



Note that the second assignment of a value to "bluetop" redefines "bluetop" to be a wireframe style plot structure. It does not change the previous image (which still has its solid blue top) since that is an image computed from the previous meaning. However if we now go back and re-evaluate the command which produced that image the computation will be done with the current meaning of "bluetop" and the resulting picture will be identical to the one immediately above.

2.2.4 Exercise (drawing colored cubes): Have Maple draw a cube on which each face has a different color. Have it draw a cube on which each pair of opposite faces have the same color but no two faces which share an edge have the same color.

We have seen that we can make new boxes from old ones by "repainting" the faces of a single box or making some of the faces transparent. These "new" boxes will, however, occupy exactly the same position in space as the original. Suppose we want to "stack" a copy of the original box on top of it, or create a duplicate some distance away. This is extremely easy to do because of the general nature of our approach. Our original sketch could have any box at all - it didn't become a specific box until we assigned values to the vertices v1..v8. Thus if we simply go back and assign different values to the v1..v8 and execute all of the commands which followed we will duplicate everthing we have done for a new box which can then be displayed along side the old one. However at the outset of an investigation it is usually not a good idea to go back and change commands which we have working the way we want. It is MUCH better to copy the working commands that we want into the same or another worksheet and modify the copies. That way when something doesn't work we have a definite point at which to start "debugging". We will use the option of copying the commands we want into a new worksheet within the same Maple session.

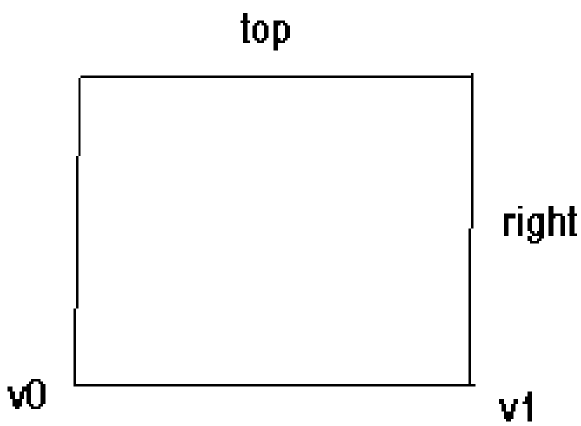
2.3 Introductory Exercises: Drawing in the Plane

The following exercise set is a brief, "get started" introduction to Maple. Follow the instructions in order.

2.3.1 *1. Open a new Maple worksheet by clicking on "new" in the "file menu". Practice "toggeling" between this and the new worksheet by selecting the appropriate worksheet in the "Window" menu. Save the new worksheet using the "Save As" option at the "File" menu with an appropriate name such as "EX_CH1".*

*

2.3.2 *2. "Select" this (entire) exercise set with the mouse and copy and paste it into the new worksheet.*



2.3.3 *3. Use "Paint" or a similar program to draw a figure like the one above. Label the upper right vertex "V2" and the upper left "V3". Label the other edges "left" and "bottom" as appropriate.*

PASTE THE DIAGRAM IN THE SPACE ABOVE

*

2.3.4 *4. Working in the new worksheet, break this exercise into distinct paragraphs (execution groups) by placing the cursor at each asterisk to the left and pressing "F3". This can also be done at at the "Edit" menu at "split or join" execution group.*

*

2.3.5 *5. Delete each of the asterisks.*

*

2.3.6 *6. Open two command lines (execution groups) immediately below the each of your newly created cells by use of the CONTROL J or at the "Insert" Menu or by use of the large ">" (greater than sign) on the toolbar. That is, place the cursor in the cell and then press CONTROL J or use the menu or toolbar (twice). The result should be that each of the cells you created is followed by two new cells (execution groups) containing only a ">" prompt .*

*

2.3.7 *7. Follow each of the following instructions by entering the requested command on the first of the command lines you opened. Commands for "arithmetic": Addition, multiplication, division, exponents, parenthesis, etc. follow the familiar syntax used with most calculators. For instance one would enter 3*(5+3^12)^(2^(11)); in order to calculate* $3\,(5+3^{12})^{(2^{11})}$.

2.3.8 *8. Add the numbers 1,3,17,-21,35,12*

*

2.3.9 *9. Multiply the numbers 1,3,17,-21,35,12*

* The "ifactor" command allows you to factor integers, e.g ifactor(24); should return the prime factorization of 24.

*

2.3.10 *10. factor your social security number; factor your birth year.*

*

2.3.11 *11. Calculate* $2^{(2^j)}+1$ *and factor* $2^{(2^j)}+1$ *for j = 0,1,2,3,4,5,6,7. Do so by entering both commands on the same command line below (each with its own semicolon) and subsequently by modifying the commands, changing the "j" to the next integer. (Until the turn of the century it was thought that all such numbers were prime.)*

*

Maple will add, subtract, and multiply polynomial expressions. Typically a line like (x+y)*(x-y); will return the "unexpanded" version. The "expand" command tells the computer to actually multiply out a polynomial expression in any number of variables, e.g. expand(x*(a+b)); and expand((a+b)^2); will produce just what one would expect. One can do things like assign a name to an expression like joe:=expand((a+b)^2); . The command "factor" does for polynomials what "ifactor" does for integers. Thus following the previous command one could factor(joe);

2.3.12 *12. Have the computer do the polynomial multiplication (1-3*x+5*x^3)*(23-19*x+12*x^2);.*

*

2.3.13 *13. Have the computer "multiply out"* $(a+b)^{99}$*. **

2.3.14 *14. Tell the computer to assign the value "kate" to the result of expanding* $(2\,x+3)^{50}$ *out "completely".*

*

2.3.15 *15. Have the comuter to factor "kate".*

*

2.3.16 *16. Release 4 of Maple will "add" lists (earlier versions will not) and multiply a number times a list 2*[a,4]; returns [2a,8]) . If you have Release 4 or higher , tell Maple to add the lists [1,2,3], [4,5,6], and 7 times [a,b,c].*

*

Maple knows all of the functions found on a graphics calculator and many more. The command plot(sin(3*x),x=-2*Pi..2*Pi); will plot sin(3*x); from -2π to 2π. The command plot({sin(3*x),sin(2*x), sin(x)} ,x=-2*Pi..2*Pi); will calculate the three functions sin(3x), sin(2x) and sin(x) at the same time.

*

2.3.17 *16. Have Maple plot cos(x) from -5 to 3.*

*

2.3.18 *17. Have Maple plot 1, x, 2*x, and 3*x^2 from -2 to 2 at the same time.*

*

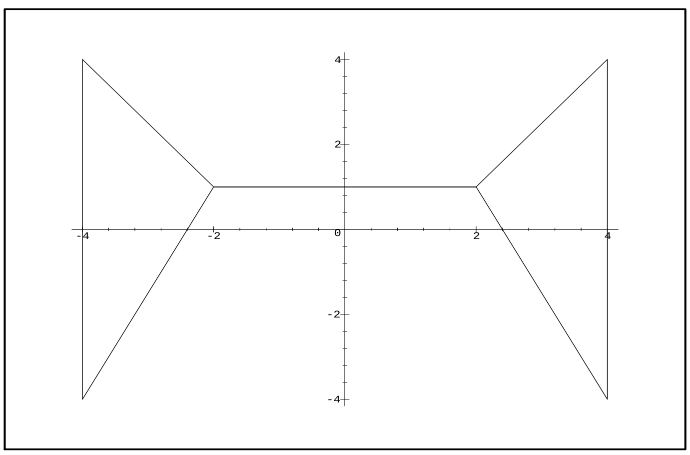
The plots[polygonplot] command will plot lines and polygons in the plane. The command plots[polygonplot]([[1,2],[3,7]]); will draw the line segment connecting the point [1,2] to the point [3,7] while

plots[polygonplot]([[1,2],[3,-4], [4,0]]); will draw a triangle with the specified vertices.

plots[polygonplot]({ [[1,2],[3,7]], [[1,2],[3,-4], [4,0]]}); will draw a line segment and a triangle.

*

2.3.19 *18. Have Maple draw the following figure:*
2.3.20



*

There are many options available to enhance pictures drawn with Maple. Some of them are "axes", "scaling", "color", "thickness" and "style". These and the others are described in the "help" system. Some comon possibilities are:

axes = none, axes=framed, axes=boxed, scaling =constrained,

scaling=unconstrained, color=red, color=tan, color=blue,

style=patch, style=POINT, style=LINE, thickness=2, thickness=3.

Multiple options can be invoked in the same plot. For instance, plots[polygonplot]([[1,2],[3,-4], [4,0]],color=red, style=patch,axes=none, scaling=constrained);

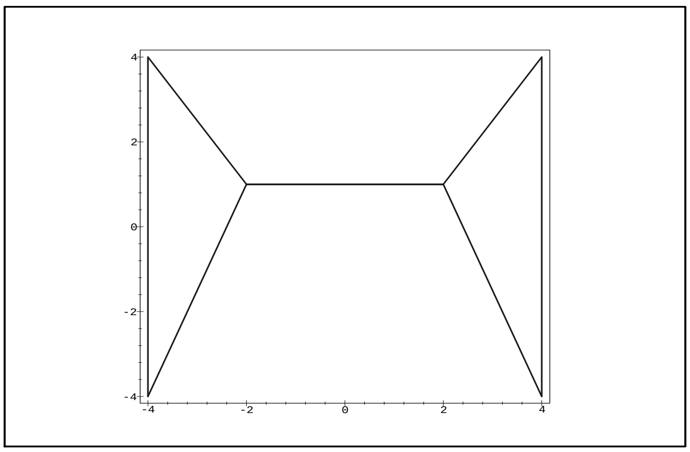
will draw the polynomial "to scale", with no axes, and will "color it in" in red. Specifying a color will automatically set the style to "patch" so if one wants just the edges of a polygon colored one invokes the LINE style.

*

2.3.21 *19. Have Maple draw a "solid" blue triangle with vertices at [0,0], [3,0], and [0,10] - do not specify any scaling (this automatically invokes the default which is "unconstrained"). Draw the same triangle "to scale" with "framed" axes. Draw the same triangle with no axes, do not fill it in, unconstrained, with the edges as "thick" (thickness=3) red lines.*

*

2.3.22 *20. Have Maple draw this picture.*



*

The plot options can also be used with the "plot" command (although in some cases they may have no effect). For instance the following will yield the same results: plot(sin(x), x=-Pi..Pi); and plot(sin(x), x=-Pi..Pi, style = patch); . On the other hand other options will change the graphs.

*

2.3.23 *21. Execute the following, changing the varions options among those listed above and observe the differences:*

2.3.24 *plot({sin(x),cos(x)}, x=-Pi..Pi, style = POINT, scaling=constrained, axes=framed, thickness=2, color=blue); **

2.4 More Introductory Exercises and Drawing in Space

2.4.1 *1. Open a new Maple worksheet by clicking on "new" in the "file menu". Practice "toggeling" between this and the new worksheet by selecting the appropriate worksheet in the "Window" menu.*

*

2.4.2 *2. "Paint" this exercise with the mouse and copy and paste it into the new worksheet.*

*

2.4.3 *3. Use the "paint" program to draw a (square-based) pyramid, label each of its vertices with the symbols v1,..v5, and paste it into your new Maple worksheet in the space below.*

PASTE THE DIAGRAM IN THIS SPACE
*

2.4.4 *4. Working in the new worksheet, break this exercise into distinct paragraphs (execution groups) by placing the cursor at each asterisk to the left and pressing "F3". This can also be done at at the "Edit" menu at "split or join" execution group.*

*

2.4.5 *5. Delete each of the asterisks.*

*

Open two command lines (execution groups) immediately below the each of your newly created cells by use of the CONTROL J or at the "Insert" Menu or by use of the large ">" (greater than sign) on the toolbar. That is, place the cursor in the cell and then press CONTROL J or use the menu or toolbar (twice). The result should be that each of the cells you created is followed by two new cells (execution groups) containing only a ">" prompt .

Follow each of the following instructions by entering the requested command on the first of the command lines you opened.

Maple does arithmetic with integers (whole numbers: no decimal point) or fractions whose numerator and deniminator are integers with no round off. If a single number in such a calculation has a decimal (i.e. is a floating point number) then Maple carries the number of decimal digits specificd by the variable Digits . The default value of digits is 10. The user can change the value of Digits by assigning a (positive integer) value to it.

2.4.6 *6. Have Maple calculate 2^100 and 100!*

*

2.4.7 *7. The Maple command "sqrt" can be used to calculate square roots, To calculate the square root of 19 use sqrt(19); Note the difference between sqrt(19); and sqrt(19.0);*

Have Maple display the first thousand digits of the square root of 2. Start with the command Digits:=1000; When done be sure to reset the digits to 10.

*

2.4.8 *8. Sometimes Maple's accuracy obscures what we really want to know. For instance, is the fraction $\frac{13^{99}}{123^{97}}$ greater than 1 or less than 1? Enter joe:=13^99/123^97; and see if you can tell. In such cases it is convenient to convert such experssions to floating point (decimal) approximations from which we can readily answer such questions. The Maple command evalf (evaluate floating point) does this. Try evalf(joe);*

2.4.9 *9. Maple stores 10000 digits of Pi but if you ask it to return Pi by entering the command Pi; you will just get a symbol. Use "Digits" and "evalf" to have Maple print out 5000 digits of Pi. Note that Pi is what Maple thinks is the ration of the circumference of a circle to its diameter; pi is simply another symbol with no independent interpretation. Note the difference between evalf(Pi); and evalf(pi);*

*

2.4.10 *10. We have seen that Maple can plot functions of one variable with plot . It can plot functions of two variables with plot3d . To plot "sin(x^2-y^2)" for x between -2 and 3 and y between -1 and 2.5 use the command plot3d(sin(x^2-y^2),x=-1..3,y=-1..2.5);*

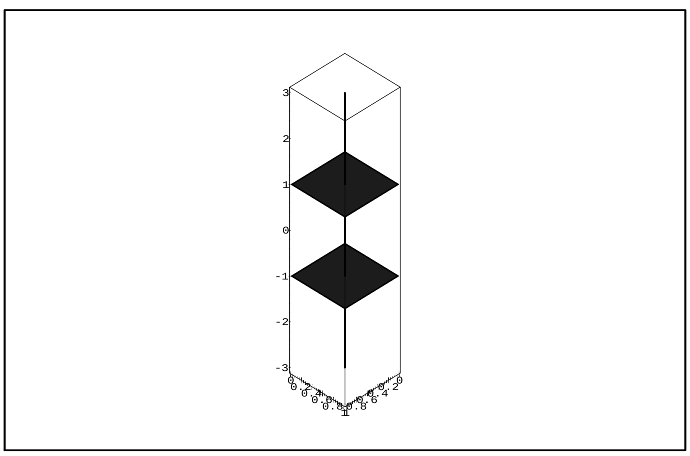
*

The plots package contains many interesting three dimensional plotting tools. Among the most useful is plots[polygonplot3d], the analog of plots[polygonplot] in two dimensions. One must keep in mind that plots[polygonplot] expects lists of points in the plane while plots[polygonplot3d] expects lists of points in space. plots[polygonplot3d] will plot lines and polygons in the plane. The command plots[polygonplot]([[1,2,0],[3,7,0]]); will draw the line segment connecting the point [1,2,0] to the point [3,7,0] while

plots[polygonplot]([[1,2,0], [3,-4,0], [4,0,0]]); will draw a triangle with the specified vertices.

plots[polygonplot]({ [[1,2,0],[3,7,0]], [[1,2,0],[3,-4,0], [4,0,0]]}); will draw a line segment and a triangle.

```
plots[polygonplot3d]({[[0,0,1],[1,0,1],[1,1,
1],[0,1,1]],
[[0,0,-1],[1,0,-1],[1,0,-1],[1,1,-1],[0,1,-1]],[[1/2,1/2,3],[1/2,1/2,-
3]]},
thickness=3,color=blue, scaling=constrained, style=patch,
axes=boxed);
```



2.4.11 *11. Make changes in the options invoked and observe the changes. Note that you can rotate three dimensional plots by "clicking" on them with the mouse, holding the mouse button down and "dragging" the reference box (which represents the figure) around. Double clicking on the figure or selecting "R" for (redraw) at the tool bar will draw the rotated figure.*

2.4.12 Exercise (drawing colored cubes): Have Maple draw a cube on which each face has a different color. Have it draw a cube on which each pair of opposite faces have the same color but no two faces which share an edge have the same color.

3 Translating Geometric Objects

3.1 Duplicating and moving cubes

The following commands, which we have copied from the previous worksheet contain essentially everything we have done to this point to create our cube(s). In this chapter we will begin to make use of the colon ":" terminator for commands to suppress unwanted output. Note that the colon can always be replaced by a semicolon and the command re-executed if we need to look at the output.

```
front:=[v1,v2,v3,v4];
back:=[v5,v6,v7,v8];
left:=[v4,v8,v5,v1];
right:=[v3,v2,v6,v7];
bottom:=[v3,v4,v8,v7];
top:=[v1,v2,v6,v5];

v1:=[1,0,1]:  v2:=[1,1,1]:  v3:=[1,1,0]:
v4:=[1,0,0]:
v5:=[0,0,1]:  v6:=[0,1,1]:  v7:=[0,1,0]:  v8:=[0,0,0]:

box:=[top, bottom, right, left, front, back]:

bluetop:=
plots[polygonplot3d](top, style=patch,
color = blue ):
purplebottom:=plots[polygonplot3d]
(bottom, style=patch,color = purple):
yellowsides:=plots[polygonplot3d]([left,right,front,back],
style=patch,color = yellow):

plots[display]([bluetop, purplebottom,
yellowsides]);
```

It is evident that simply by changing the assigned values of v1..v8 we can create images of different cubes. However, rather than doing exactly that we will make another copy in this sheet which we will modify so that our previously defined objects will continue to have the same meaning. This illustrates a practice that is always good to follow. Copy and paste working code whenever possible (as opposed to re-typing it which almost always introduces mistakes). Since only the precise symbols we have defined have current meaning a slight change creates entirely new symbols and relationships. For instance we may elect to change the names of the vertices to w1,...,w8 and simply add a "w" to each of the previously defined words. (The reader will suspect immediately see that this could get out of hand if we keep doing the same thing - we will soon learn a more systematic approach).

```
frontw:=[w1,w2,w3,w4];backw:=[w5,w6,w7,w8];
leftw:=[w4,w8,w5,w1];rightw:=[w3,w2,w6,w7];
bottomw:=[w3,w4,w8,w7];topw:=[w1,w2,w6,w5];
```

Now we have a "template" for a second cube and need only assign values to the vertices w1...w8 as before. The question is which values to assign.

Let us assume we want to stack a duplicate of the first cube directly on top of it. To "lift" our first cube (which has height 1) straight up so that its base moves to the level of its top requires that the "z-coordinate" of each of its points be increased by exactly 1. That is if [a,b,c] is a point on the cube that that point moves to [a,b,c+1]. We don't need to explicitly move every point,

only the vertices. Moreover we note that with an obvious meaning for addition $[a,\, b,\, c+1] = [a,\, b,\, c] + [0,\, 0,\, 1]$. Maple (Relese 4 and higher) understands this addition so, assuming that the vertex w1 is where v1 goes, w2 is where v2 goes, ..., etc. we can assign values to our vertices w1...w8 as follows:

```
shift:=[0,0,1];

w1:=v1+shift;w2:=v2+shift;w3:=v3+shift;w4:=v4
+shift;w5:=v5+shift;
w6:=v6+shift;w7:=v7+shift;w8:=v8+shift;

boxw:=[topw, bottomw, rightw, leftw, frontw,
backw];

bluetopw:=plots[polygonplot3d](topw,
style=patch,color = blue ):
purplebottomw:=plots[polygonplot3d](bottomw, style=patch,
color = purple):
yellowsidesw:=plots[polygonplot3d]([leftw,rightw,frontw,backw],
style=patch,color = yellow):
```

Let's put in an origin to show the shift. Note a list with one point in it is a polygon.

```
ORIGIN := plots[polygonplot3d]([[0,0,0]]):

plots[display]([ ORIGIN,bluetopw,
purplebottomw, yellowsidesw], axes=normal);
```

Reference to the axes verifies that our new cube is indeed a copy of the old one shifted up exactly one uint. The question now is how to display them at the same time. The answer is that we do exactly what we did before.

```
plots[display]([ plots[display]([bluetop,
purplebottom, yellowsides]),
plots[display]([bluetopw, purplebottomw, yellowsidesw], axes=normal)]
);
```

The command above obviously stacked our cubes although we do see here the need to tell Maple not to scale the image if we want the cubes to appear as such. That is easily remedied. More important is to understand how what the above command did and how to abbreviate it. The plots[display]

command takes in *lists of plot structures and outputs a plot structure. Each of our cubes is such an output and . Thus this command is of the form plots[display]([plotstructure1, plotstructure2], axes=normal). We can make the assignments:*

```
cube1:=plots[display]([bluetop,
purplebottom, yellowsides]):

cube2:=plots[display]([bluetopw,
purplebottomw, yellowsidesw]):
```

Then we can express the command which generated the "stacked" cubes in the form

```
plots[display]([cube1, cube2], axes=normal,
```

```
scaling=constrained);
```

Note that we have added the option scaling=constrained to give our pictures the proper scale.

There is no reason to for both cubes the same color or style, we can give the faces of cube2 any color or style we want. What if we want a cube3? The obvious ways to produce a cube3 are repeat what we did to get cube2 or even to take the tack we avoided then and simply modify our previous code to produce a cube3. Keeping in mind that once we have produced cube2 if we then redefine the vertices w1..w8 we can produce a separate cube3, retaining the *picture of our "original" cube2 but not cube2 itself. We could do the same thing to get a cube4, cube5, etc. If all we want are the pictures this will work but there are problems. For instance, suppose we wanted 100 cubes made them all yellow and then decided that they should be red. We would have to repeat all of these steps "by hand". Fortunately, Maple provides tools to automate this type of manufacturing process. What we need to do is build a new Maple word which takes the original cube and produces a set of "stacked" copies (or a picture of such a set) assembled according to our instructions. We will take up this process in the next chapter but first we want to actually build a few new cubes and assemble them in order to gain a firm grasp of exactly what process we want to automate. This will be a process we will repeat many times in this study. First let us block out what we did to get cube1.*

3.2 The steps in building a cube

1. Define the original, abstract cube "box" as a list of faces where each face was itself a list of vertices v1..v8.

2. Assign specific values to the vertices v1..v8. Doing so automatically caused Maple to interpret the abstract cube, box, as the specific cube with the vertices assigned to v1..v8.

3. Execute a command cube1:=plots[polygonplot3d](box, options) which produced a plot structure representing the current "value" of "box"

Now let us extend these steps to the process which produced cube2 - but this time re-assigning the values of the v1..v8 rather than introducing the new vertices.

4. Assign a specific value to the word "shift" (note there is nothing special about the word "shift", "yzkotqp" would work just as well but would be less intuitive and perhaps harder to remember).

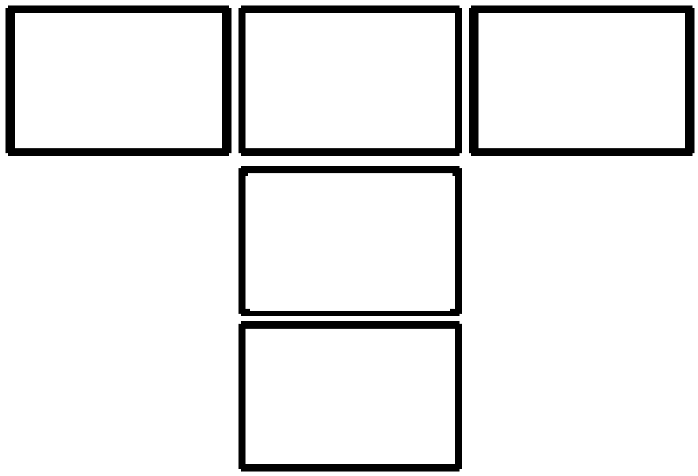
5. Assign new values to v1..v8 by the assignment commands v1:=v1+shift; v2:=v2+shift; ..., which say "the new value of v1 is the current value of v1 plus shift, etc.

6. Execute the command cube2:=plots[polygonplot3d](box, options)

7. go to step 4.

3.3 Building a TEE

Lets create an assemblage of cubes in the shape of a "T" as in the following sketch.



Although we don't really need to we will repeat the initial steps. This time we will use a different notation for the pictures of the faces which doesn't designate their

colors so that we may feel free to vary these as we wish. In addition, we introduce the "shift" immediately and simply let it be [0,0,0] to start.

```
v1:=[1,0,1]:  v2:=[1,1,1]:  v3:=[1,1,0]:
v4:=[1,0,0]:
v5:=[0,0,1]:  v6:=[0,1,1]:  v7:=[0,1,0]:  v8:=[0,0,0]:

shift:=[0,0,0]:

v1:=v1+shift:  v2:=v2+shift:  v3:= v3+shift:
v4:=v4+shift:
v5:=v5+shift:  v6:=v6+shift:  v7:=v7+shift:  v8:=v8+shift:
front:=[v1,v2,v3,v4]:back:=[v5,v6,v7,v8]:left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:bottom:=[v3,v4,v8,v7]:top:=[v1,v2,v6,v5]:

toppic:=plots[polygonplot3d](top,
style=patch,color = blue ):
bottompic:=plots[polygonplot3d](bottom, style=patch,
color = purple):
sidespic:=plots[polygonplot3d]([left,right,front,back], style=patch,
color = yellow):

cube1:=plots[display]([ toppic, bottompic,
sidespic]):
```

Now we have cube1, all we need do now is reset the shift and repeat the second group of commands, changing only the name of the result to "cube2".

```
v1:=[1,0,1]:  v2:=[1,1,1]:  v3:=[1,1,0]:
v4:=[1,0,0]:
v5:=[0,0,1]:  v6:=[0,1,1]:  v7:=[0,1,0]:  v8:=[0,0,0]:

shift:=[0,0,1]:
```

```
v1:=v1+shift:  v2:=v2+shift:  v3:= v3+shift:  v4:=v4+shift:  v5:=v5+shift:  v6:=v6+shift:
v7:=v7+shift:  v8:=v8+shift:  front:=[v1,v2,v3,v4]:back:=[v5,v6,v7,v8]:left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:bottom:=[v3,v4,v8,v7]:top:=[v1,v2,v6,v5]:
```

```
toppic:=plots[polygonplot3d](top,
style=patch,color = blue ):
bottompic:=plots[polygonplot3d](bottom, style=patch,color = purple):
sidespic:=plots[polygonplot3d]([left,right,front,back], style=patch,
color = yellow):

cube2:=plots[display]([ toppic, bottompic,
sidespic]):
```

Now cube3 results from a shift of two units straight up

```
v1:=[1,0,1]:  v2:=[1,1,1]:  v3:=[1,1,0]:
v4:=[1,0,0]:
v5:=[0,0,1]:  v6:=[0,1,1]:  v7:=[0,1,0]:  v8:=[0,0,0]:

shift:=[0,0,2]:


v1:=v1+shift:  v2:=v2+shift:  v3:= v3+shift:
v4:=v4+shift:
v5:=v5+shift:  v6:=v6+shift:  v7:=v7+shift:  v8:=v8+shift:
front:=[v1,v2,v3,v4]:back:=[v5,v6,v7,v8]:left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:bottom:=[v3,v4,v8,v7]:top:=[v1,v2,v6,v5]:

toppic:=plots[polygonplot3d](top,
style=patch,color = blue ):
bottompic:=plots[polygonplot3d](bottom, style=patch,color = purple):
sidespic:=plots[polygonplot3d]([left,right,front,back], style=patch,
color = yellow):

cube3:=plots[display]([ toppic, bottompic,
sidespic]):
```

Cube4 is two up and one to the "right"

```
v1:=[1,0,1]:  v2:=[1,1,1]:  v3:=[1,1,0]:
v4:=[1,0,0]:
v5:=[0,0,1]:  v6:=[0,1,1]:  v7:=[0,1,0]:  v8:=[0,0,0]:

shift:=[-1,0,2]:


v1:=v1+shift:  v2:=v2+shift:  v3:= v3+shift:
v4:=v4+shift:
v5:=v5+shift:  v6:=v6+shift:  v7:=v7+shift:  v8:=v8+shift:
front:=[v1,v2,v3,v4]:back:=[v5,v6,v7,v8]:left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:bottom:=[v3,v4,v8,v7]:top:=[v1,v2,v6,v5]:

toppic:=plots[polygonplot3d](top,
style=patch,color = blue ):
bottompic:=plots[polygonplot3d](bottom, style=patch,color = purple):
```

```
sidespic:=plots[polygonplot3d]([left,right,front,back], style=patch,
color = yellow):

cube4:=plots[display]([ toppic, bottompic,
sidespic]):
```

And cube5 results from a shift of two up and one to the "left"

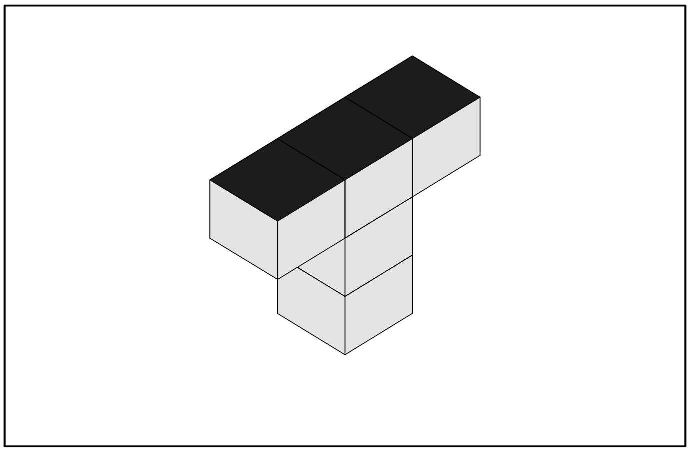
```
v1:=[1,0,1]:  v2:=[1,1,1]:  v3:=[1,1,0]:
v4:=[1,0,0]:
v5:=[0,0,1]:  v6:=[0,1,1]:  v7:=[0,1,0]:  v8:=[0,0,0]:

shift:=[1,0,2]:


v1:=v1+shift:  v2:=v2+shift:  v3:= v3+shift:
v4:=v4+shift:
v5:=v5+shift:  v6:=v6+shift:  v7:=v7+shift:  v8:=v8+shift:
front:=[v1,v2,v3,v4]:back:=[v5,v6,v7,v8]:left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:bottom:=[v3,v4,v8,v7]:top:=[v1,v2,v6,v5]:

toppic:=plots[polygonplot3d](top,
style=patch,color = blue ):
bottompic:=plots[polygonplot3d](bottom, style=patch,color = purple):
sidespic:=plots[polygonplot3d]([left,right,front,back], style=patch,
color = yellow):

cube5:=plots[display]([ toppic, bottompic,
sidespic]):
```

We have not taken the time to change the colors on our various cubes but we could have, simply by changing the options for each of them. Lets make our cubes into a list and display them

```
cubes:=[cube1,cube2,cube3,cube4,cube5]:
```

Finally, we can display our cubes. Since its to be a "T" we'll call it "Tee". The following command will then assign that plot structure to the symbol "Tee". It will not display it. On the other hand, Tee then becomes a Maple word whose value is that plot structure so when Maple receives "Tee" from the command line it will respond return "Tee" in the manner it represents such structures - namely by plotting them.

```
Tee:=plots[display](cubes,
scaling=constrained):

Tee;
```



3.3.1 Exercise: The "T" generated previously is an example of a figure that can be made by five identical cubes such that each of them share a face with at least one other. There is an "L" figure which can be made with 4. Find all of the figures of this type which can be made using either three or four cubes and construct a Maple word for each one which displays it. We regard two figures as the same if they are congruent in the sense of geometry - that is if one could be moved and rotated in space to become coincident with the other. Include a careful explanation of why your list is complete.

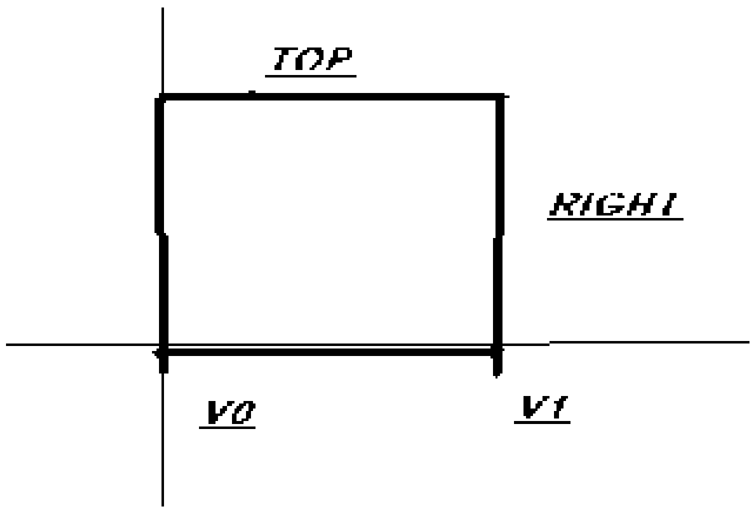
3.4 Exercises: More Pictures in the Plane

Although we have concentrated on "3d" graphics in the text one often wants to work with "2d" versions of the same problems. Not only may the problem be inherently two dimensional but computer resources required for three dimensions may severely limit what we can do on a particular machine. In this exercise we will discuss translating planar boxes.

3.4.1 *0. If you have not read through and executed the main worksheet for this exercise go to the beginning of this worksheet and execute the first few commands.*

3.4.2 *1. Copy this entire exercise set into a new worksheet and begin work in the new worksheet.*

3.4.3 *3.. Use "Paint" or a similar program to draw a figure like this oneLabel the upper right vertex "V2" and the upper left "V3". Label the other edges "left" and "bottom" as appropriate*



PASTE YOUR DIAGRAM IN THIS AREA

3.4.4 *4. Enter the commands* `top; right; left; bottom;`

3.4.5 *at the prompt. If Maple returns anything but these names then they already have assigned values - this will be the case if you have just executed the first few commands in the worksheet. Note that all worksheets open at any one time share a common set of variables. Thus (assuming exercise 1 was done) even though we are in a different worksheet, the value of "top" from the original is shared by this "new" one.*

3.4.6 *We want to assign new values to these symbols for our "plane" work. For instance we will want "bottom" to be the list "[v0,v1]" and to represent the bottom edge of the figure we have drawn. If we are careful in making these assignments then we will simply "overwrite" the previous meanings of the words with new assignments. However were we to forget to actually make the assignment for, say, "left" then subsequent execution errors will occur which can be quite difficult to track down. In addition, when we change the value of "top" in the new worksheet its "new" value will be used in subsequent calculations in other worksheets - but will not affect the current values from earlier calculations. That is, it doesn't work like a spreadsheet, previous calculations are not automatically "updated". In most cases the simplest thing to do is issue the command restart; This resets all variables to their default value which is simply their name. That is what we will do. Issue the command restart;*

3.4.7 *Now re-enter the commands* `top; right;` *left; bottom;*

3.4.8 *5. Assign the list "[v0,v1]" to the name "bottom" with the command* `bottom:=[v0,v1];` *Similarly assign the appropriate lists to "right", "left", and "bottom". Assign the name "box" to the list [bottom, right, top, left]*

3.4.9 *6. The computer is unable graphically interpret quantities that are not completely described, A person can relate the list [v0,v1] to a general line segment but the computer needs to know precisely what v0 and v1 are in order to interpret it as a line segment. Issue the command* `plots[polygonplot]([v0,v1]);` *then issue the command plots[polygonplot]([[0,0],[1,1]]);*

3.4.10 *7. Assign values [0,0] to v0, [1,0[to v1, [1,1] to v2, and [0,1] to v3 and have Maple draw the edges of the square with the command plots[polygonplot] Draw the top and left edges at the same time with plots[polygonplot]([top, left]); Draw the entire box with plots[polygonplot](box);*

NOTE: If/when you subsequently re-define the values of v0,v1,v2,v3 if you do not also "update" the interpretation of "box" (for instance by re-issueing the defining command that defined "box") then the command `plots[polygonplot](box);` will draw the *previously* defined box.

3.4.11 *8.Recall that there are many options available to enhance pictures drawn with Maple. Some of them are "axes", "scaling", "color", "thickness" and "style". These and the others are described in the "help" system. Some comon possibilities are:*

3.4.12

axes = none, axes = framed, axes = boxed,
scaling = constrained , scaling = unconstrained,
color = red, color = tan, color = blue,
style = patch, style = POINT, style = LINE,
thickness = 2, thickness = 4

3.4.13

3.4.14 *Multiple options can be invoked in the same plot. For instance ,*

3.4.15 *plots[polygonplot](box,color=green, style=patch,axes=none, thickness=4); will draw the box with no axes, and will "color it in" in green. Specifying a color will automatically set the style to "patch" so if one wants just the edges of a polygon colored one invokes the LINE style.*

*

3.4.16 *10. If P= [a,b] is a point in the plane and we want the point Q which is the result of shifting P by 1 to the right and up 4 then Q=[a+1,b+4]. Similarly if we want Z which is the result of shifting P by 3 to the left and down 7 then Z=[a -3,b-7]. We can write Q= P+[1,4] and Z = P+[-3,-7]. Beginning with release 4, Maple understands this "list arithmetic" which allows us to instruct Maple to "shift" objects in the plane.*

3.4.17 *If v0,v1,v2,v3 are the vertices of a box which is shifted so that each point of the box move "a" to the right and "b" to the left then if we "shift" be [a,b], the shifted box is [v0+shift,v1+shift, v2+shift,v3+shift]. If "shift" is [0,0] then this is simply the original box.*

3.4.18 *We want to define a new "abstract" box called "sbox" (for "shifted box) which contains the (currently undefined) quantity "shift". However if we do the "obvious" something which might not be expected will happen if we have been doing the previous exercises in the current session. Execute the command sbox:=[v0+shift,v1+shift, v2+shift,v3+shift];*

If you got the error message "adding lists of different length" this was probably because v0...v3 had meaning as lists of length 2 from the previous work while "shift" is just a name with no such meaning. One way out of this is to use symbols other than v0,v1..,v3. Edit the previous command to sbox:=[w0+shift,w1+shift, w2+shift,w3+shift]; If w0..etc had not been previously assigned some meaning and "shift" hasn't acquired one then this will work fine. However it begs the question of what to do if we subsequently need "w0" for another purpose. What one needs to do is "reset" v0,v1, etc (and possibly "shift") to their default values. In principle we could use the command restart but this would reset all variables in all worksheets in this session.

The commands `v0:='v0'; v1:='v1'; v2:='v2';` v3:='v3'; shift:='shift'; assign to each of these varaiables its own name, which is its default value. Once these are executed the definition of "sbox" will work fine.

3.4.19 *11. With "sbox" abstractly defined one can proceed as before to draw general quadralaterals in the plane by specifying (assigning) values for v0...v3 and "shift".*

3.4.20 *Assign v0,v1,v2,v3 the values [0,0], [1,0], [1,1], and [0,1], and "shift" the value [2,3]. The use the command*

3.4.21 *plots[polygonplot](sbox, color = green, style = patch, axes = normal);*

3.4.22 *to draw the shifted box.*

3.4.23 *12. One can assign a name to a plot and subsequently display that plot by invoking its name. For instance joe:=plots[polygonplot](box,color=blue, style=LINE, thickness=4, axes=normal); will display Maple's internal description of the figure, In this case the internal description is very short and interesting to view. In general it may run many, many pages and one does not want to see it. This is a case where the colon delimiter is very useful. Run joe:=plots[polygonplot](box,color=blue, style=LINE, thickness=4, axes=normal): then joe;*

3.4.24 *13. With the ability to make and "store" different pictures we can collect different pictures and display them together, creating more complex images. For instance the following is simply a collection of images of squares made one at a time. Each is a shift of the basic "unit box", (v0=[0,0], v1=[1,0],v2=[1,1],v3=[0,1]), made by a sequence of commands of the form*

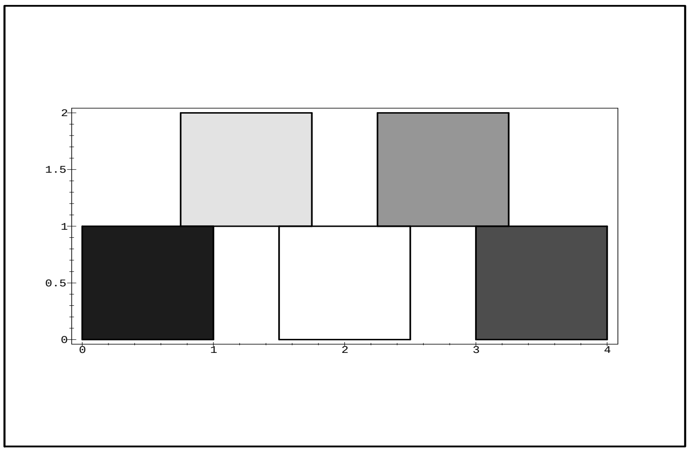
3.4.25 *shift:= [1,3]; sbox:=[v0+shift,v1+shift,v2+shift,v3+shift]; bx1:=plots[polygonplot](sbox,color=green, style=patch,axes=normal*

3.4.26

3.4.27 *The command to display the members of a list a collection of plots images at the same time is plots[display]. Thus for instance if, using the above command sequence one makes images 4 images bx1, bx2, bx3, bx4t. The command to display them all at once would be plots[display([bx1,bx2,bx2,bx3,bx4]); Of course one could make a list bx:=[bx1,bx2,bx3,bx4]; and simply do plots[display](bx);*

3.4.28 *Note that one can use different plot options in each image. One can also include options in plots[display]: plots[display](bx,scaling=constrained);*

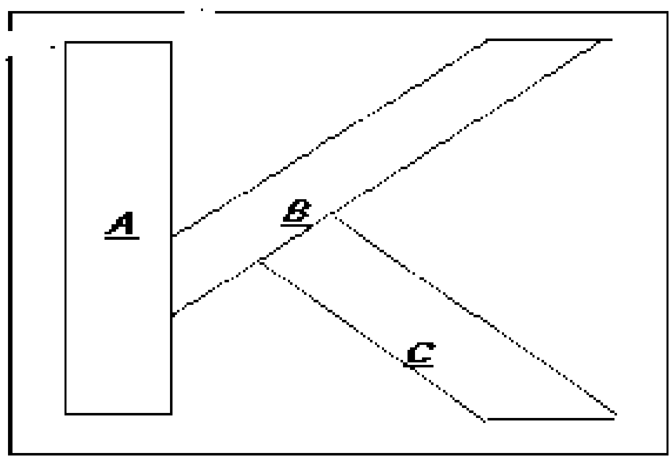
3.4.29 *Have Maple draw this picture.*



.

3.5 Exercises: Making 3d Alphabet Blocks

Introduction: One often finds colored cubes as children's toys and these often have letters painted on them. Lets investigate the possibility of making a white cube with a blue "K" on one face. The white cube is no problem, we've done that before. We need to make the "K". We know how to make polygons so we want a "K" thats made of them.



This one is made of the polygons A,B, and C. All we need to do is draw something like this on a piece of graph paper, read off the coordinates of the vertices, make up each of the polygons and put them together.

```
A:=[[1/10,0,1/10],[1/10,0,9/10],[3/10,0,9/10]
,[3/10,0,1/10]];
```

$$A := [[\frac{1}{10}, 0, \frac{1}{10}], [\frac{1}{10}, 0, \frac{9}{10}], [\frac{3}{10}, 0, \frac{9}{10}], [\frac{3}{10}, 0, \frac{1}{10}]]$$

```
B:=[[3/10,0,4/10],[3/10,0,6/10],[6/10,0,9/10]
,[9/10,0,9/10]];
```

$$B := [[\frac{3}{10}, 0, \frac{2}{5}], [\frac{3}{10}, 0, \frac{3}{5}], [\frac{3}{5}, 0, \frac{9}{10}], [\frac{9}{10}, 0, \frac{9}{10}]]$$

```
C:=[[4/10,0,5/10],[6/10,0,6/10],[9/10,0,1/10]
,[7/10,0,1/10]];
```

$$C := [[\frac{2}{5}, 0, \frac{1}{2}], [\frac{3}{5}, 0, \frac{3}{5}], [\frac{9}{10}, 0, \frac{1}{10}], [\frac{7}{10}, 0, \frac{1}{10}]]$$

```
K:=plots[polygonplot3d]([A,B,C],color=blue,st
yle=patch):
```

Now we need the cube. As before we retrieve our basic box construction

```
v1:=[1,0,1]:  v2:=[1,1,1]:  v3:=[1,1,0]:
v4:=[1,0,0]:
v5:=[0,0,1]:  v6:=[0,1,1]:  v7:=[0,1,0]:  v8:=[0,0,0]:

box:=[top, bottom, right, left, front, back]:

front:=[v1,v2,v3,v4];back:=[v5,v6,v7,v8];left
:=[v4,v8,v5,v1];
right:=[v3,v2,v6,v7];bottom:=[v3,v4,v8,v7];top:=[v1,v2,v6,v5];
```

$$front := [[1, 0, 1], [1, 1, 1], [1, 1, 0], [1, 0, 0]]$$

$$back := [[0, 0, 1], [0, 1, 1], [0, 1, 0], [0, 0, 0]]$$

$$left := [[1, 0, 0], [0, 0, 0], [0, 0, 1], [1, 0, 1]]$$

$$right := [[1, 1, 0], [1, 1, 1], [0, 1, 1], [0, 1, 0]]$$

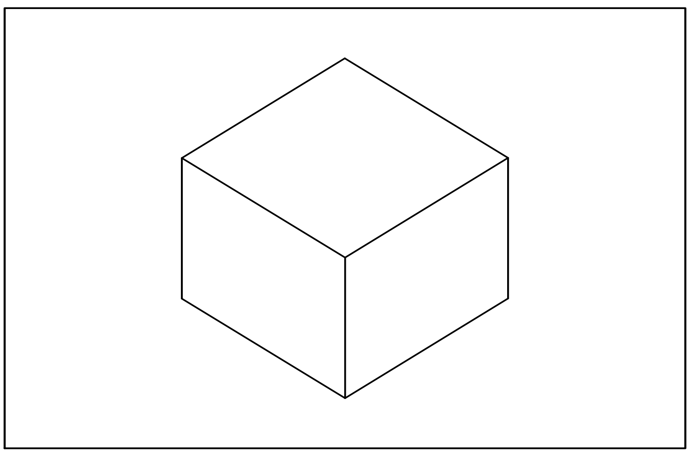
$$bottom := [[1, 1, 0], [1, 0, 0], [0, 0, 0], [0, 1, 0]]$$

$$top := [[1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]]$$

```
whitecube:=plots[polygonplot3d](box,
color=white, style=patch,thickness=3,scaling=constrained);
```

$$\begin{aligned}&whitecube := \mathbf{PLOT3D}(\mathbf{POLYGONS}([[1., 0, 1.], [1., 1., 1.], [0, 1., 1.], [0, 0, 1.]],\\&\quad [[1., 1., 0], [1., 0, 0], [0, 0, 0], [0, 1., 0]], [[1., 1., 0], [1., 1., 1.], [0, 1., 1.], [0, 1., 0]],\\&\quad [[1., 0, 0], [0, 0, 0], [0, 0, 1.], [1., 0, 1.]], [[1., 0, 1.], [1., 1., 1.], [1., 1., 0], [1., 0, 0]],\\&\quad [[0, 0, 1.], [0, 1., 1.], [0, 1., 0], [0, 0, 0]]), \mathbf{STYLE}(\mathit{PATCH}), \mathbf{SCALING}(\mathit{CONSTRAINED}),\\&\quad \mathbf{THICKNESS}(3), \mathbf{COLOUR}(\mathit{RGB}, 1, 1, 1))\end{aligned}$$

```
Kcube:=plots[display]([K,whitecube]):
Kcube;
```



There are two problems here. The first is a very sloppy "K" which results from inaccurate reading of coordinates from a hastily drawn graph. This can be partially addressed by more careful graphing. The second problem is that some views show a two-toned "K". This is what happens when two patch-style pictures are in contact. The patch pictures are made of colored triangles occupying the same positions and the computer doesn't know which onw we really want to see. The solution is to move them slightly apart. In this case its we might have the "K" as being slightly raised above the surface of the cube. That just means altering the definition of K

```
A:=[[1/10,-.01,1/10],[1/10,-.01,9/10],[3/10,-
.01,9/10],[3/10,-.01,1/10]];
```

$$A := [[\frac{1}{10}, -.01, \frac{1}{10}], [\frac{1}{10}, -.01, \frac{9}{10}], [\frac{3}{10}, -.01, \frac{9}{10}], [\frac{3}{10}, -.01, \frac{1}{10}]]$$

```
B:=[[3/10,-.01,4/10],[3/10,-.01,6/10],[6/10,-
.01,9/10],[9/10,-.01,9/10]];
```

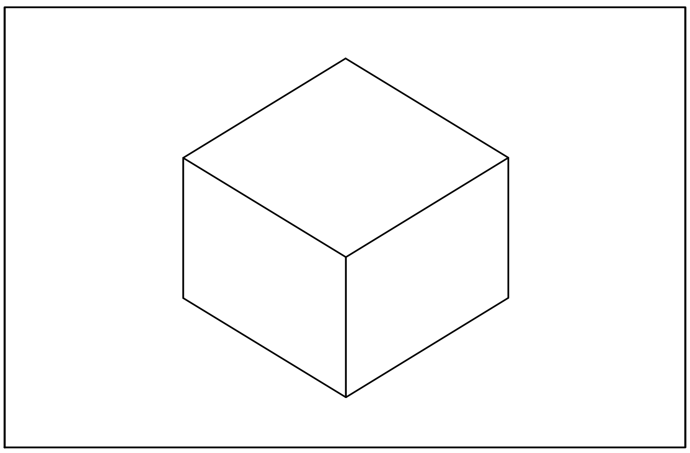
$$B := [[\frac{3}{10}, -.01, \frac{2}{5}], [\frac{3}{10}, -.01, \frac{3}{5}], [\frac{3}{5}, -.01, \frac{9}{10}], [\frac{9}{10}, -.01, \frac{9}{10}]]$$

```
C:=[[4/10,-.01,5/10],[6/10,-.01,6/10],[9/10,-
.01,1/10],[7/10,-.01,1/10]];
```

$$C := [[\frac{2}{5}, -.01, \frac{1}{2}], [\frac{3}{5}, -.01, \frac{3}{5}], [\frac{9}{10}, -.01, \frac{1}{10}], [\frac{7}{10}, -.01, \frac{1}{10}]]$$

```
K:=plots[polygonplot3d]([A,B,C],color=blue,st
yle=patch):
```

```
Kcube:=plots[display]([K,whitecube]):Kcube;
```

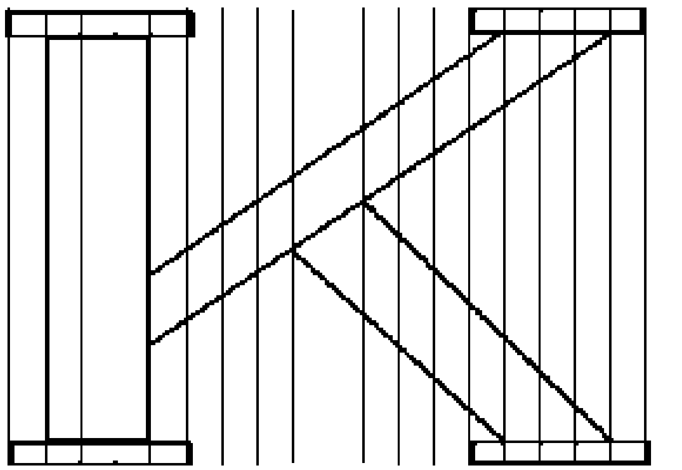


Its a simple matter to add a red "K" to the other side of this block.

```
A1:=[[1/10,1.01,1/10],[1/10,1.01,9/10],[3/10,1.01,9/10],[3/10,1.01,1/1
0]];
B1:=[[3/10,1.01,4/10],[3/10,1.01,6/10],[6/10,1.01,9/10],[9/10,1.01,9/1
0]];
C1:=[[4/10,1.01,5/10],[6/10,1.01,6/10],[9/10,1.01,1/10],[7/10,1.01,1/1
0]];
 K1:=plots[polygonplot3d]([A1,B1,C1],color=red,style=patch):
 plots[display]([K,whitecube,K1]);{
}
```

3.5.1 EXERCISE: Improve the "Kcube" by creating one with a better "K" by more careful graphing.

One way to do this is to draw a "K" on graph paper and take the coordinates from there. The command plots[coordplot](cartesian, view=[-10..10,-10..10], grid=[20,20]); will create a "graph paper" grid. Copy and paste it into "Paint" or a similar program and draw a better "K" from which coordinates of component polygons can be read.



Assume the lower left corner is the origin [0,0]. Then the "K" is made of seven polygons. For instance the three pieces comprising the left "post" are

```
P1:=[[0,0],[5,0], [5,1],[0,1]];
P2:=[[1,1],[4,1],[4,18],[1,18]];
P3:=[0,18], [5,18],[5,19],[0,19]];
```

Determine the other four polygons, make plots of each of these with commands of the form k2:=plots[polygonplot](p2, style=LINE, color=blue):

(note the seimcolon terminator). Then creats an image of the compete "K with newK:=plots[display]([k1,k2,k3,k4,k5,k6.k6]):

This has created a two dimensional K. Now follow the steps in creating the original "K" to produce a more carefully drawn "K-block".

3.5.2 *EXERCISE: Change the "Kcube" so that there is a "K" on each face.*

3.5.3 *EXERCISE: Make a yellow block with one the letters A,B,C,D,E,F on each face.*

3.5.4 *EXERCISE: Give each of the letters in the previous problem a different color.*

3.5.5 *EXERCISE: Make the block in the previous problem appear to be made of a transparent material with the colored letters painted on its faces. HINT: Wireframe*

3.5.6 *EXERCISE: Make a row of alphabet blocks that display your initials.*

3.5.7 *EXERCISE: Create a "solid" "K"*

3.5.8 *Exercise: The following are pictures of arrangements of children's blocks. Create a Maple image that "model's" each picture.*

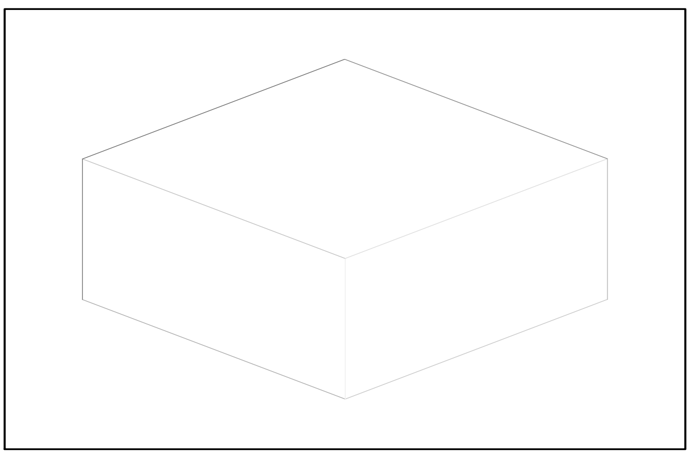
4 Building a Block Factory: Creating a Maple Procedure

In this section we want to begin to learn how to define new **Maple words.** After a little bit of effort, you will find that it is as natural as driving a car (and not nearly as dangerous, because there is no mistake in Maple that can't be fixed.) The subject is blocks, also known as cubes.

4.1 Start with a single cube.

If we are to construct more complex figures out of cubes and manipulate them in more interesting ways we need a more efficient means of producing the rectangular boxes with which to build them. It would be nice if there were a Maple word which instructed the program to make, say, a cube of side 1 and place it at a location of our choice. In the previous chapter we did all of the work that such a word would do. We constructed the basic cube and placed it in the desired location by specifying a 'shift'. The steps to create a simple, colorless cube might be:

```
front:=[v1,v2,v3,v4]:  back:=[v5,v6,v7,v8]:
left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:  bottom:=[v3,v4,v8,v7]:  top:=[v1,v2,v6,v5]:
box:=[front,back,left,right,top,bottom]:
shift:=[0,0,0]:
v1:=[0,0,0]+shift:  v2:=[1,0,0]+shift:  v3:= [1,1,0]+shift:
v4:=[0,1,0]+shift:
v5:=[0,0,1]+shift:  v6:=[1,0,1]+shift:  v7:=[1,1,1]+shift:
v8:=[0,1,1]+shift:
plots[polygonplot3d](box);
```



If we wanted the cube placed at any particular place in space all we have to do is change the value of "shift". If we wanted to retain the previous cube then instead of simply having **plots[polygonplot3d]** immediately draw the picture we assigned the plot structucture generated to a word, say "cube1". Then we could change the shift and the name, say to "cube2" and have plots[display] show us our cubes separately or individually as we wish. What we would like to do is have a Maple word, say "cube" which would take in the shift and produce the plot structure for the corresponding cube. We want the command "cube([0,0,0])" to produce the results of the previous set of commands and would want "cube([0,0,1])"

to produce the results of simply changing "shift" to "[0,0,1]" in the same list of commands. That is, all we want is a way to vary

the value of "shift " in this set of commands. Maple does this with **procedures.** The form of a Maple procedure is simple:

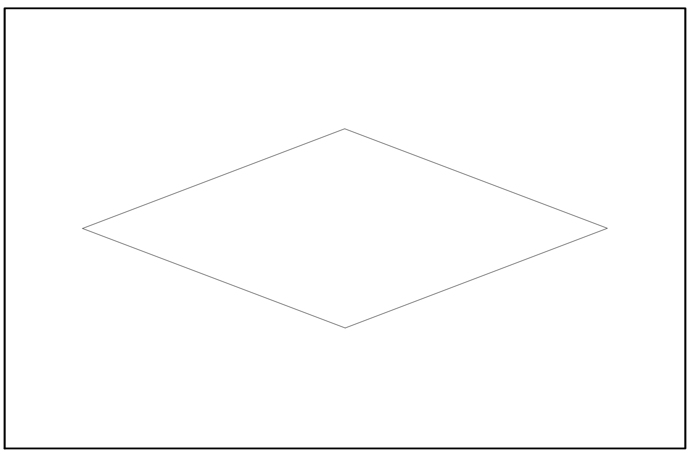
ProcedureName:= proc(input_variable)

Maple commands in which "input_variable" is treated as input.

end

For instance suppose we want a Maple word which always returns a picture of a unit square of whichever color we specify. We need a choice for this word, the ProcedureName. Any choice will work provided its not "reserved" by Maple. For instance we might choose "colorsquare". We need a name for the procedure variable. Since we are going to vary color we choose something like "clr" since we have seen that "color" is a word already defined in Maple. Finally we need to know the Maple word which will produce a square of a given color. We know that the following will work:

```
plots[polygonplot3d]([[0,0,0],[1,0,0],[1,1,0]
,[0,1,0]], color=red);
```



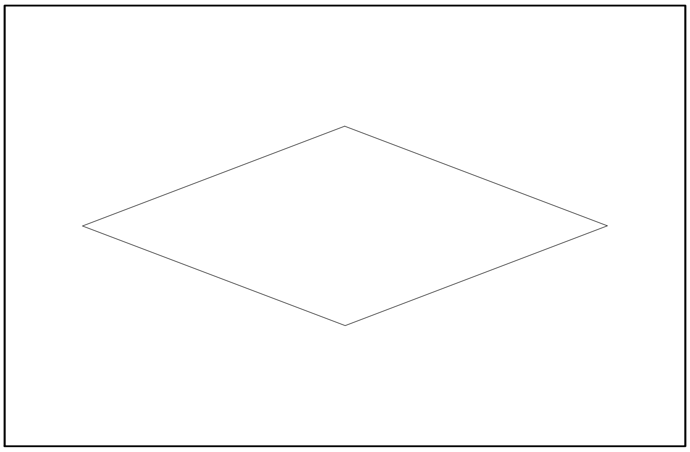
Thus following the above format we create a new word:

```
colorsquare:=proc(clr)
plots[polygonplot3d]([[0,0,0],[1,0,0],[1,1,0],[0,1,0]], color=clr)
end;
```

$$colorsquare := \mathrm{proc}(clr)\ plots_{polygonplot3d}([[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0]], color = clr)\ \mathrm{end}$$

We can test the word out right away.

```
colorsquare(blue);
```



This word only colors the boundary of the square. To find out how to color the interior of the square, read on.

Creating our 'cube' word is no more difficult and is done essentially by cutting and pasting the code that does one case, changing the specific values in the code to the variable quantities for the procedure and 'wrapping' this in the 'proc()...end;' format.

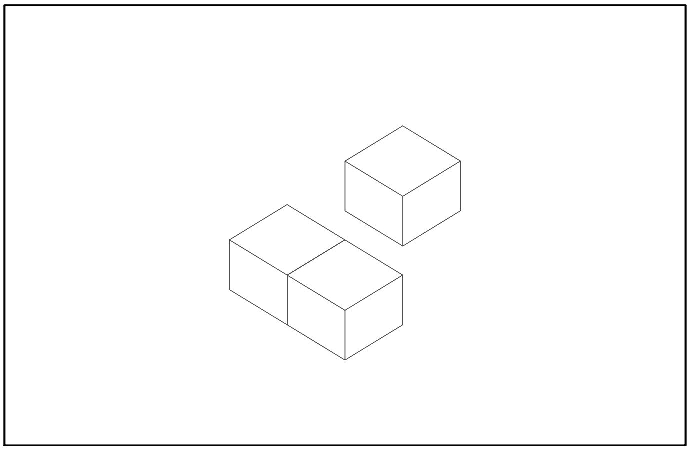
```
cube:=proc(shift)
v1:=[0,0,0]+shift:  v2:=[1,0,0]+shift:  v3:= [1,1,0]+shift:
v4:=[0,1,0]+shift:
v5:=[0,0,1]+shift:  v6:=[1,0,1]+shift:  v7:=[1,1,1]+shift:
v8:=[0,1,1]+shift:
front:=[v1,v2,v3,v4]:  back:=[v5,v6,v7,v8]:  left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:  bottom:=[v3,v4,v8,v7]:  top:=[v1,v2,v6,v5]:
box:=[front,back,left,right,top,bottom]:
plots[polygonplot3d](box);
end:
```

For the moment pay no attention to the "warning" messages, we will clean that up shortly. Right not what we need to see is that we do indeed have a "cube" word. The idea is always to get something working and improve it. We can give names to some particular cubes and then display them all together.

```
joe:=cube([1,2,4]);
sam:=cube([1,0,1]);
sally:=cube([1,1,1]):
plots[display]([joe,sam,sally],color=red,scaling=constrained);
```

$joe := \mathbf{PLOT3D(POLYGONS(}[[1., 2., 4.], [2., 2., 4.], [2., 3., 4.], [1., 3., 4.]],$
$[[1., 2., 5.], [2., 2., 5.], [2., 3., 5.], [1., 3., 5.]],$
$[[1., 3., 4.], [1., 3., 5.], [1., 2., 5.], [1., 2., 4.]],$
$[[2., 3., 4.], [2., 2., 4.], [2., 2., 5.], [2., 3., 5.]],$
$[[1., 2., 4.], [2., 2., 4.], [2., 2., 5.], [1., 2., 5.]],$
$[[2., 3., 4.], [1., 3., 4.], [1., 3., 5.], [2., 3., 5.]]))$

sam := **PLOT3D(POLYGONS(**[[1., 0, 1.], [2., 0, 1.], [2., 1., 1.], [1., 1., 1.]],
[[1., 0, 2.], [2., 0, 2.], [2., 1., 2.], [1., 1., 2.]], [[1., 1., 1.], [1., 1., 2.], [1., 0, 2.], [1., 0, 1.]],
[[2., 1., 1.], [2., 0, 1.], [2., 0, 2.], [2., 1., 2.]], [[1., 0, 1.], [2., 0, 1.], [2., 0, 2.], [1., 0, 2.]],
[[2., 1., 1.], [1., 1., 1.], [1., 1., 2.], [2., 1., 2.]]))



When a word which is not a Maple or user-defined word is used within a Maple procedure it is allowed to have meaning only within the procedure. It is used only locally, hence the "local variable" designation. This allows words to be used within procedures that actually have an external meaning. This is very useful in the development of procedures. For instance, our objective in developing "cube" to no longer have to think about its details when we need a cube in the future. Suppose we were to use it in the future as part of a project in which we have also used the word "top" to mean the amount of fuel in a tank when it is full. Now having forgotten all about the internal workings of "cube" we issue the command "cube([0,0,0])". Note that within "cube" there is a command "top:=[v1,v2,v6,v5]". If "top" in "cube" were not restricted to have meaning only within that procedure its invovation would have changed the "top" we were thinking of from a number to a four element list. This could cause all kinds of problems.

The default is that Maple makes all such variables local variables and tells us about it with that message. If we want to tell the program in advance that these are to be local variables we provide it the a sequence of those variables in the code defining the procedure.

There is a corresponding idea of **global variable,** which is what we would use if we really did want "cube" to modify "top". Here is the way we would have entered "cube" including the local variables.

```
cube := proc (shift)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
```

```
plots[polygonplot3d](box)
end:
```

We will use this **code** again in next worksheet. The word code is used throughout this text to refer to sequences of Maple statements which may be part of a procedure definition.

Now that we have a basic version of "cube" we can add features. Suppose we want to specify the style and color of the cube as well as the shift. This is easy as Maple procedures are not restricted to a single variable. It is a good idea to make a single improvement at a time, though. This makes finding errors much easier. Lets specify the style first. All we need to do is copy the code over. We don't go back and change our only copy of a piece of working code. We copy the working version, paste it in and (ususally) change the name at least temporarily

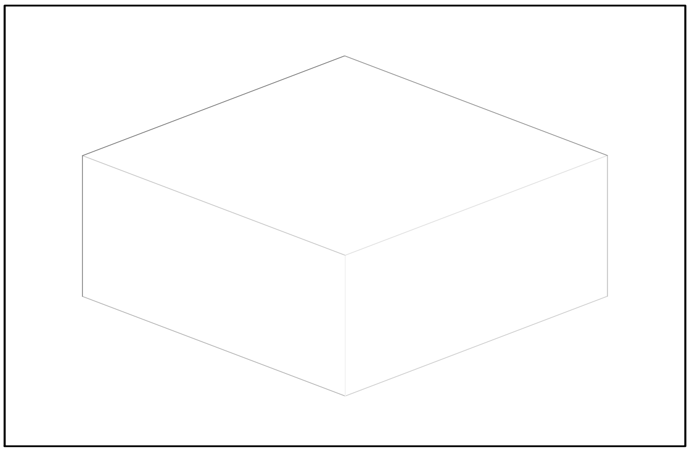
```
tempcube := proc (shift)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box)
end:
```

This is simply the original code with a different name. Now modify it to take both a shift and a color. We have indicated the changes in upper case in the following copy. In this case we simply add a variable "clr" and the option "color=clr" in the plots[polygonplot3d] command.

```
tmpcube := proc (shift,CLR)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=CLR)
end:
```

Now test this change right away.

```
tempcube ([0,0,0],red);
```

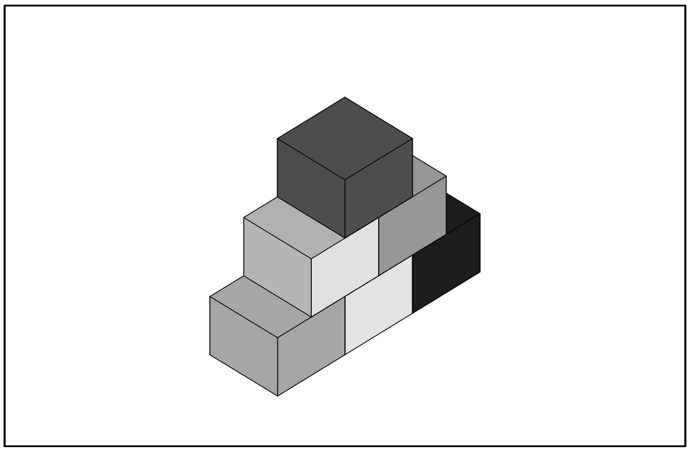


Now we can go another step and put the style choice in the inputs.

```
tempcube2 := proc (shift,CLR,STY)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=CLR,style=STY)
end:
```

Now we can play with colored cubes:

```
cube1:=tempcube2([0,0,0],blue,patch):
cube2:=tempcube2([1,0,0],yellow ,patch):
cube3:=tempcube2([2,0,0],tan ,patch):
cube4:=tempcube2([1/2,0,1],green ,patch):
cube5:=tempcube2([3/2,0,1],purple ,patch):
cube6:=tempcube2([1,0,2],red,patch):
plots[display]([ cube1, cube2, cube3, cube4, cube5, cube6 ],
scaling = constrained);
```



4.2 Another Example of how to 'procedurize' a sequence of commands

It is very important to learn the process of organizing sequences of Maple commands which solve a specific problem into a procedure for doing a general problem of that form. For example. The following Maple code calculates the equation of the line though the two points [1,3] and [-5,12].

```
eqn:= y=slope*x+yintercept;
```

$$eqn := y = slope\, x + yintercept$$

```
slope:= (12 - 3)/(-5 - 1);
```

$$slope := \frac{-3}{2}$$

```
yintercept:=3-slope*1;
```

$$yintercept := \frac{9}{2}$$

```
eqn;
```

$$y = -\frac{3}{2}\,x + \frac{9}{2}$$

We want to modify this to take in the x and y coordinates of any two points [x1,y1] and [x2,y2] and to return the equation of the line through them. Once we see that we should think of [1,3] as the particular [x1,y1] and [-5,12] as the particular [x2,y2] we can readily wrap this into a procedure with a name such as "lineequ". Note that we are ignoring for the moment the possibility that x1=x2 in which case Maple will complain about division by zero. We'll learn to take care of such things later.

```
lineequ:=proc(x1,y1,x2,y2)
local slope, intercept, eqn;
eqn:= y=slope*x+intercept;
slope:= (y2 - y1)/(x2 - x1);
intercept:= y1 - slope*x1;
y=slope*x+intercept;
end:
```

Then to calculate the equation of the line passing through [1,3] and [-5,12] one can execute the command

```
lineequ( 1,3, -5,12);
```

$$y = -\frac{3}{2}x + \frac{9}{2}$$

4.3 EXERCISES

4.3.1 Exercise: *Modifty the procedure lineqn above so its inputs are a two points in the plane rather than a four numbers.*

4.3.2 Exercise : Modify the tempcube2 procedure to make a word which makes cubes which have white tops and bottoms and whose sides are any color we specify.

4.3.3 Exercise: Modify the following Maple commands find the point of intersection of two specific lines in the plane from their equations, each of the form $y = mx + b$. Make this code into a procedure intersect(m1,b1,m2,b2) which finds the point of intersection of the lines with equations $y = m1x + b1$ and $y = m2x + b2$. (remark: We are not worrying at this point about such niceties as whether the two lines are parallel or not. We will learn to check for this sort of thing later). Test your word on several lines.

```
m1:=10; b1:=13; m2:=7; b2:=25;
solve({y=m1*x+b1,y=m2*x+b2},{x,y});
```

$$m1 := 10$$

$$b1 := 13$$

$$m2 := 7$$

$$b2 := 25$$

$$\{y = 53,\ x = 4\}$$

4.3.4 Exercise : The Maple command sum(i,i=1..4781); adds up the first 4781 positive integers. Adapt it to make a Maple word addpowers(n,m) which returns the sum of the nth powers of the integers from 1 to m. Use your word to answer the following question: Which is larger, the sum of the 4th powers of the first 100 positive integers, or the sum of the third powers of the first 300 integers?

4.3.5 Exercise: *What does the following Maple procedure do? If a word is unfamiliar, use the Maple Help to find out about it.*

```
ndigits:=proc(n) local tmp:
Digits:=n; tmp:=evalf(Pi); Digits:=10:  tmp; end;
```

$$ndigits := \mathrm{proc}(n)\,\mathrm{local}\,tmp;\ Digits := n\,;\ tmp := \mathrm{evalf}(\pi)\,;\ Digits := 10\,;\ tmp\,\mathrm{end}$$

4.3.6 Exercise *What does the following Maple procedure do? If a word is unfamiliar, use the Maple Help to find out about it.*

```
dig:=proc(n) floor(10*frac(10^(n-1)*Pi))
end;
```

$$dig := \mathrm{proc}(n)\ \mathrm{floor}(10 \times \mathrm{frac}(10^{(n-1)} \times \pi))\ \mathrm{end}$$

4.3.7 Exercise: The following is a Maple procedure which almost works. It should input a list of four numbers and make a bar graph representation of that data with the bar representing the first number having as it base the line segment [0,1] on the x-axis, the bar representing the second number should have as its base the segment [1,2], etc. The color of the first bar should be red, the second green, the third yellow, and the fourth blue.

```
barit:=proc(l) local bar1,bar2,bar3, bar4;
bar1:=plots[polygonplot]( [[2,0],[4,0],
[4,l[3]],[2,l[3]]],color=green):
bar2:=plots[polygonplot]( [[0,0],[1/2,0],
[1/2,l[4]],[0,l[4]]],style=patch,color=red):
bar3:=plots[polygonplot]( [[3,1],[5,1], [5,l[1]],[3,l[1]]],
style=LINE,color=blue,thickness=3):
bar4:=plots[polygonplot]( [[5,0],[6,0],
[6,-l[2]],[5,-l[2]]],style=patch,color=yellow):
plots[display]([bar1,bar2,bar3,bar4]); end;
```

Here is barit([5,2,7,-4])

```
\textbf{Exercise}: The following Maple procedure is supposed to input
three points in the plane and check to see if they are collinear. Read
the instructions carefully and explain exactly what the procedure is
doing.
```

```
checkit:=proc(P,Q,R) local slope, intercept,
eqn;
eqn:= y=slope*x+intercept;
slope:= (Q[2] - P[2])/(Q[1] - P[1]);
intercept:= Q[2] - slope*Q[1];
R[2]=slope*R[1]+intercept
end;
```

4.3.8 Exercise: *The following procedure inputs a list of four points in the plane and does a calculation which hopes to tell whether the four points are the vertices of a square. Draw a diagram of a quadralateral, label the vertices in accordance with the notation used by the procedure and explain with reference to the geometry of the figure exactly what the procedure is doing. Explain why its output can or can not tell one whether the figure is in fact a square*

Recall that if P is a list of things that are themselves lists then P[3,7] denotes the seventh element of the third member of the list P.

```
isit:=proc( P) local tmp1,tmp2:
```

```
tmp1:= ((P[1,1]-P[3,1])^2 + ( P[1,2]-P[3,2])^2) - (
(P[2,1]-P[4,1])^2 + (
P[2,2]-P[4,2])^2 );
tmp2:= ((P[1,1]-P[2,1])^2 + ( P[1,2]-P[2,2])^2) - ( (P[2,1]-P[3,1])^2
+ (
P[2,2]-P[3,2])^2 );
tmp1^2+tmp2^2;
end:
```

5 Laying Bricks:Getting the Computer to do Repetitive Tasks

5.1 Working with expression sequences and lists

The process of analyzing or solving a problem can involve a lot of repetition. Anyone who has graphed many different functions knows the tedious repetition of that process. Maple's built-in "plot" commands allow the computer do the mindless repetition, freeing time for people to look at the relationships implied by the graph - something the machine can't do. Graphing, however is only one example of this of this capability which is absolutely critical to effective use of the computer for solving general problems. Lets look at another - building lists.

We have already made and used a lot of Maple lists. A more primitive notion than that of a list is that of an expression sequence

1 *Definition: An expression sequence is a comma-separated expression of the form "a,b,c,...,z" where the elements "a", "b" , etc. can be any symbols recognizable by Maple.*

The order of elements in an expression sequence is important: "a,b" is a different expression sequence from "b,a"

For our immediate purposes the most important thing about expression sequences is how easy it is to extend them. If joe is an expression sequence and we want to add "Hortense" to the end of joe we can simply enter joe:=joe, Hortense; . If we wanted to place Hortense at the beginning we would enter joe:= Hortense,joe; This idea also allows us to "build" sequences.

If we initially have a sequence named "joe" and need, in the course of a calculation to extend "joe" then this can be done through steps like

```
joe:=I;
```

$$joe := I$$

```
joe:=joe,like;
```

$$joe := I, like$$

```
joe:=joe,math;
```

Sometimes it is useful as a starting point to have an empty expression. That is something which is an expression but has no items in it. This is evidently analogous to the empty set in set theory. This idea is so useful that Maple has a special name for the emptyexpression sequence. It is called NULL....

$$joe := I, like, math$$

```
joe:=NULL;
```

$$joe :=$$

```
joe:=joe,I;
```

$$joe := I$$

```
joe:=joe,like;
```

$$joe := I, like$$

```
joe:=joe,math;
```

$$joe := I,\ like,\ math$$

With the notion of expression sequence we can observe that a list is simply an expression sequence delimited by brackets "[ ]". Most important, to convert an expression sequence, joe to a list sam we simply enclose it in brackets

```
sam:=[joe];
```

$$sam := [I,\ like,\ math]$$

Since a list is simply an expression sequence enclosed in brackets there is a simple way to extract the undellying expression sequence from a list with a command called **op.** We will encounter op quite often. This is but one of its many uses. (op **stands for operands)**

5.2 How to Build a list by hand

Maple has a word **whattype** which we can use to ask what type it has assigned to various objects (there are many kinds of types). Now let's illustrate.

First let's make a simple sequence. We will start with the NULL sequence to indicate how it is used as a starting point for building sequences

```
joe:=NULL;
```

$$joe :=$$

```
harry:=joe,sally,1;
```

$$harry := sally,\ 1$$

```
whattype(joe);
```

$$exprseq$$

```
whattype(harry);
```

$$exprseq$$

```
joe:= harry,dog, Pi;
```

$$joe := sally,\ 1,\ dog,\ \pi$$

```
whattype(joe);
```

$$exprseq$$

```
sam:=[joe];
```

$$sam := [sally,\ 1,\ dog,\ \pi]$$

```
whattype(sam);
```

$$list$$

```
tim:=op(sam);
```

$$tim := sally,\ 1,\ dog,\ \pi$$

```
whattype(tim);
```

$$exprseq$$

Now suppose we want to make the list stuff:= [sally,1,dog, π, Lance]; . We observe that we should be able to do this simply by adding "Lance" to the end of the list "sam". We would like to do something like stuff:=sam,Lance; **or stuff:=[sam,Lance]; but neither of these will work. This is not to say that Maple won't accept the commands**

```
stuff:=sam,Lance;
```

$$stuff := [sally,\ 1,\ dog,\ \pi],\ Lance$$

```
whattype(stuff);
```

$$exprseq$$

```
stuff:=[sam,Lance];
```

$$stuff := [[sally,\ 1,\ dog,\ \pi],\ Lance]$$

Neither of these works. The first produces an expression sequence, not a list. Moreover it has only two elements, the first being the list "sam" and and the second the expression "Lance". The second does produce a list but it is a list with two elements: the first element of this is the list " [sally,1,dog, π]" and the second is "Lance".

In order to produce the list we want we have to extract the expression sequence from "sam", append "Lance " to it and then convert that sequence to a list.

```
tmp:=op(sam);
```

$$tmp := sally,\ 1,\ dog,\ \pi$$

```
tmp:=tmp,Lance;
```

$$tmp := sally,\ 1,\ dog,\ \pi,\ Lance$$

```
stuff:=[tmp];
```

$$stuff := [sally,\ 1,\ dog,\ \pi,\ Lance]$$

```
whattype(stuff);
```

$$list$$

We can be more economical by combining steps by composing commands:

```
stuff:=[op(sam),Lance];
```

$$stuff := [sally,\ 1,\ dog,\ \pi,\ Lance]$$

Incidentally, to go with "op" there is a very useful commnad **nops (number of operands)** which we can use to tell us how many items there are in a list

```
junk:=[a,b,cow];
```

$$junk := [a, b, cow]$$

```
nops(junk);
```

$$3$$

Note that nops doesn't work on an expression sequence.

```
nops(NULL);
```

```
Error, wrong number (or type) of parameters in function nops
```

```
sally:=1,2,3;
```

$$sally := 1, 2, 3$$

```
nops(sally);
```

```
Error, wrong number (or type) of parameters in function nops
```

Thus in order to count the number of items in an expression sequence we must ask nops to act on a **list** of which the expression sequence is the operand.

```
sally:=1,2,3;
```

$$sally := 1, 2, 3$$

```
nops([sally]);
```

$$3$$

```
whattype(sally);
```

$$exprseq$$

It is important to note that writing "[sally]" above does not change "sally" into a list. The **assignment** command sally:=[sally]; would do that but above we have not assigned a new value to "sally". All we have done is ask nops to act on an unnamed list constructed from "sally". This leads to a comment we must make regarding what happens when we make an expression sequence some of whose elements are objects that have assigned values. For instance suppose we have executed bird:=turkey; and subsequently make an expressions sequence animals:=cat,dog,bird; The sequence animals will then be cat,dog,turkey. That is, the value that has been assigned to any of the terms in the sequence we write down will be substituted immediately. If we then redefine "bird", say bird:=pigeon; the sequence animals will still be cat,dog,turkey unless we do something which re-defines animals (e.g. we might re-execute the command animals:=cat,dog,bird; at which time animals will become cat,dog,pigeon **.**

```
bird:=turkey;
```

$$bird := turkey$$

```
animals:=cat,dog,bird;
```

$$animals := cat,\ dog,\ turkey$$

```
print([animals]);
```

$$[cat,\ dog,\ turkey]$$

```
bird:=pigeon;
```

$$bird := pigeon$$

```
print([animals]);
```

$$[cat,\ dog,\ turkey]$$

```
animals:=cat,dog,bird;
```

$$animals := cat,\ dog,\ pigeon$$

```
print([animals]);
```

$$[cat,\ dog,\ pigeon]$$

This property of sequences is extremely important for it allows us to build a sequence in steps re-using a temporary name for the item we are adding.

```
dogs:=NULL;
```

$$dogs :=$$

```
tmpdog:=Beagle;
```

$$tmpdog := Beagle$$

```
dogs:=dogs,tmpdog;
```

$$dogs := Beagle$$

```
tmpdog:=Airdale;
```

$$tmpdog := Airdale$$

```
dogs:=dogs,tmpdog;
```

$$dogs := Beagle,\ Airdale$$

5.3 Automatically Generating Sequences and Lists

Now suppose we wanted to make a list of the first 25 positive intgers. One way to do this is simply to enter it:

```
numlist:=[1,2,3,4,5,6,7,8,9,10,11,
12,13,14,15,17,17,18,19,20,21,22,23,24,24,25];
```

$$numlist := \\ [1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15,\ 17,\ 17,\ 18,\ 19,\ 20,\ 21,\ 22,\ 23,\ 24,\ 24,\ 25]$$

This is tedious and fraught with opportunity to make mistakes. Indeed the careful reader will have noted that this isn't the correct list, "17" was entered twice. Even that careful soul might not take the time to check whether this sequence which clams to list the numbers from 672345 to 677345 is actually correct.

5.3.1 Boring List of Numbers from 672345 To 677345

boringlist := [672345, 672346, 672347, 672348, 672349, 672350, 672351, 672352, 672353, 672354, 672355, 672356, 672357, 672358, 672359, 672360, 672361, 672362, 672363, 672364, 672365, 672366, 672367, 672368, 672369, 672370, 672371, 672372, 672373, 672374, 672375, 672376, 672377, 672378, 672379, 672380, 672381, 672382, 672383, 672384, 672385, 672386, 672387, 672388, 672389, 672390, 672391, 672392, 672393, 672394, 672395, 672396, 672397, 672398, 672399, 672400, 672401, 672402, 672403, 672404, 672405, 672406, 672407, 672408, 672409, 672410, 672411, 672412, 672413, 672414, 672415, 672416, 672417, 672418, 672419, 672420, 672421, 672422, 672423, 672424, 672425, 672426, 672427, 672428, 672429, 672430, 672431, 672432, 672433, 672434, 672435, 672436, 672437, 672438, 672439, 672440, 672441, 672442, 672443, 672444, 672445, 672446, 672447, 672448, 672449, 672450, 672451, 672452, 672453, 672454, 672455, 672456, 672457, 672458, 672459, 672460, 672461, 672462, 672463, 672464, 672465, 672466, 672467, 672468, 672469, 672470, 672471, 672472, 672473, 672474, 672475, 672476, 672477, 672478, 672479, 672480, 672481, 672482, 672483, 672484, 672485, 672486, 672487, 672488, 672489, 672490, 672491, 672492, 672493, 672494, 672495, 672496, 672497, 672498, 672499, 672500, 672501, 672502, 672503, 672504, 672505, 672506, 672507, 672508, 672509, 672510, 672511, 672512, 672513, 672514, 672515, 672516, 672517, 672518, 672519, 672520, 672521, 672522, 672523, 672524, 672525, 672526, 672527, 672528, 672529, 672530, 672531, 672532, 672533, 672534, 672535, 672536, 672537, 672538, 672539, 672540, 672541, 672542, 672543, 672544, 672545, 672546, 672547, 672548, 672549, 672550, 672551, 672552, 672553, 672554, 672555, 672556, 672557, 672558, 672559, 672560, 672561, 672562, 672563, 672564, 672565, 672566, 672567, 672568, 672569, 672570, 672571, 672572, 672573, 672574, 672575, 672576, 672577, 672578, 672579, 672580, 672581, 672582, 672583, 672584, 672585, 672586, 672587, 672588, 672589, 672590, 672591, 672592, 672593, 672594, 672595, 672596, 672597, 672598, 672599, 672600, 672601, 672602, 672603, 672604, 672605, 672606, 672607, 672608, 672609, 672610, 672611, 672612, 672613, 672614, 672615, 672616, 672617, 672618, 672619, 672620, 672621, 672622, 672623, 672624, 672625, 672626, 672627, 672628, 672629, 672630, 672631, 672632, 672633, 672634, 672635, 672636, 672637, 672638, 672639, 672640, 672641, 672642, 672643, 672644, 672645, 672646, 672647, 672648, 672649, 672650, 672651, 672652, 672653, 672654, 672655, 672656, 672657, 672658, 672659, 672660, 672661, 672662, 672663, 672664, 672665, 672666, 672667, 672668, 672669, 672670, 672671, 672672, 672673, 672674, 672675, 672676, 672677, 672678, 672679, 672680, 672681, 672682, 672683, 672684, 672685, 672686, 672687, 672688, 672689, 672690, 672691, 672692, 672693, 672694, 672695, 672696, 672697, 672698, 672699, 672700, 672701, 672702, 672703, 672704, 672705, 672706, 672707, 672708, 672709, 672710, 672711, 672712, 672713, 672714]

"boringlist" is actually correct because it was not typed in by hand but rather done by a Maple command **seq** which generates expression sequences according to a "formula" we prescribe which tells Maple how to find the "nth" element. A few examples suffice to illustrate this very useful command. First, here is the command that generated "boringlist"

```
boringlist:=[seq(i,i=672345..672714)];
```

Note that seq returns an expression sequence. The brackets in turn make that a list. An expression sequence of the cubes of the integers from -3 to 13 is simply

```
cubesequence:=seq(i^3,i=-3..13);
```

$$cubesequence := \\ -27, -8, -1, 0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197$$

We can convert it into a list

```
cubelist:=[cubesequence];
```

$$cubelist := [-27, -8, -1, 0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197]$$

if the list was all we wanted we could have skipped naming the sequence and simply entered

```
cubelist:=[seq(i^3,i=-3..13)];
```

$$cubelist := [-27, -8, -1, 0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197]$$

Some further examples of lists:

Make a list if the odd integers from 1 to 99

```
oddlist:=[seq( 2*j+1,j=0..49)];
```

$$oddlist := [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, \\ 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, \\ 95, 97, 99]$$

Make an expression sequence of all the natural numbers less than 137 which are 2 plus a multiple of 7.

```
twomodsevenseq:=seq(2+7*q,q=0..137/7);
```

$$twomodsevenseq := \\ 2, 9, 16, 23, 30, 37, 44, 51, 58, 65, 72, 79, 86, 93, 100, 107, 114, 121, 128, 135$$

Make a list of lists of numbers so that the nth element in your list is the list $[n, \frac{1}{n^2}]$, starting with n=10 and ending with n=30.

```
listoflists:=[seq([n,1/n^2],n=10..30)];
```

$$listoflists := [[10, \frac{1}{100}], [11, \frac{1}{121}], [12, \frac{1}{144}], [13, \frac{1}{169}], [14, \frac{1}{196}], [15, \frac{1}{225}], [16, \frac{1}{256}], \\ [17, \frac{1}{289}], [18, \frac{1}{324}], [19, \frac{1}{361}], [20, \frac{1}{400}], [21, \frac{1}{441}], [22, \frac{1}{484}], [23, \frac{1}{529}], [24, \frac{1}{576}], \\ [25, \frac{1}{625}], [26, \frac{1}{676}], [27, \frac{1}{729}], [28, \frac{1}{784}], [29, \frac{1}{841}], [30, \frac{1}{900}]]$$

5.4 Exercises: Sequences and Lists

5.4.1 Exercise: Make a list of all the natural numbers which are less than 1000 and are the square of another natural number.

5.4.2 Exercise: *Make a sequence of length 100 such that the jth element of list whose first element is the integer j and who second element is the factorization of j into primes.*

5.4.3 Exercise: *Make a list of length 5 whose jth element is a plot of the function sin(j*x) on the interval [0, $2\,\pi$]. Display all of the graphs at once with plots[display].*

5.4.4 Exercise: *Make a sequence of 100 random integrers between 0 and 10. (If necessary, use "help" to look up the topic "random").*

5.4.5 Exercise: If "n" is a natural number then n factorial, denoted "n!" is the product n(n-1)(n-2)... (3)(2)(1). For instance 3!=(3)(2)(1) = 6, 1! = 1, 4!=(4)(3)(2)(1) = 24. Maple recognizes the notation n! and will compute the factorial of a natural number. Make a list of the factorials of the numbers from 1 to 20.

5.4.6

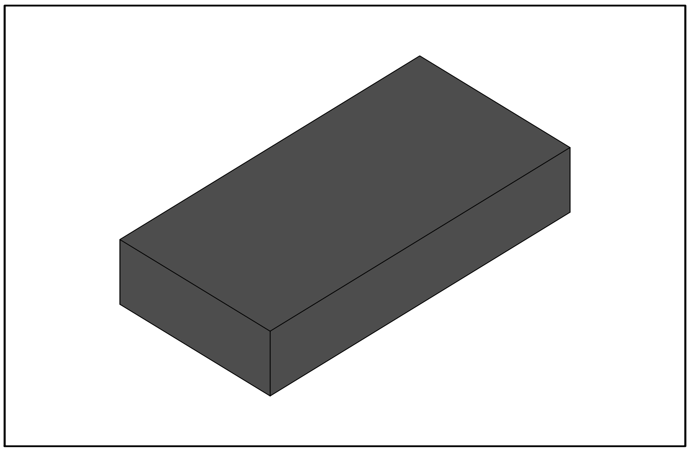
5.4.7 Exercise: The Maple command nextprime(n) will return the next prime after the integer "n". Make a list of pairs of numbers so that the nth element of your list is the pair whose first element is an integer and whose second element is the next prime after "n!". Let the first elements of your lists be the even integers from 8 to 20.

5.5 Laying bricks. Applications of sequences and lists

We can now generate sequences and lists with any command which takes in a single integer and produces an output. It even works with pictures. For instance suppose we want to lay bricks to make (a small part) of a sidewalk,a wall, or the like. First we modify our cube maker from the last worksheet to make only red bricks that a eight units long, four units wide, and two units high. Lets call it "BL" (for bricklayer).

```
BL := proc (shift)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [8, 0, 0]+shift; v3 := [8, 4, 0]+shift;
v4 := [0, 4, 0]+shift; v5 := [0, 0, 2]+shift; v6 := [8, 0, 2]+shift;
v7 := [8, 4, 2]+shift; v8 := [0, 4, 2]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=red,style=patch, scaling=constrained )
end:
```

```
BL([0,0,0]);
```



Suppose we want to make a stack of 10 bricks. Of course we could "personally" make each brick:

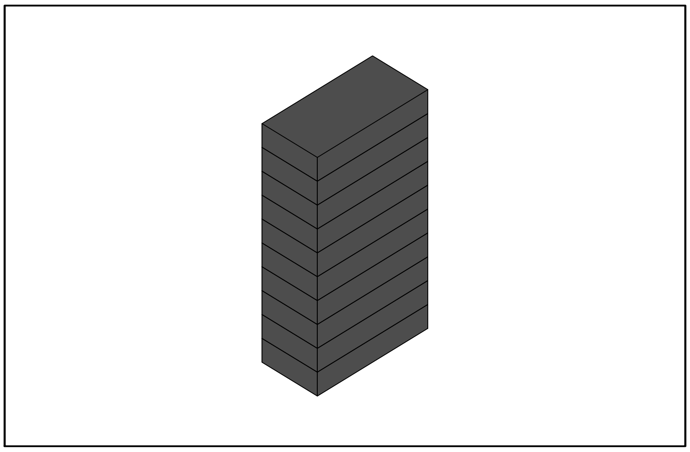
```
b1:=BL([0,0,0]):
b2:=BL([0,0,2]):
b3:=BL([0,0,4]):
b4:=BL([0,0,6]):
b5:=BL([0,0,8]):
b6:=BL([0,0,10]):
b7:=BL([0,0,12]):
b8:=BL([0,0,14]):
b9:=BL([0,0,16]):
b10:=BL([0,0,18]):
```

Then we produce a list to feed to **plots[display]** . Note that we terminate with a semicolon. The reader might want to change it to a colon and execute the command to remind him/herself why - then change it back to a colon and execute again.

```
tenbricks:=[b1,b2,b3,b4,b5,b6,b7,b8,b9,b10]:
```

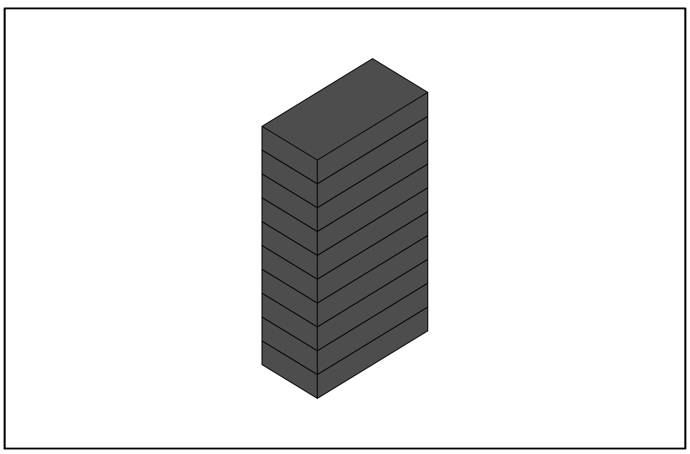
Now we can look at the picture:

```
plots[display](tenbricks);
```



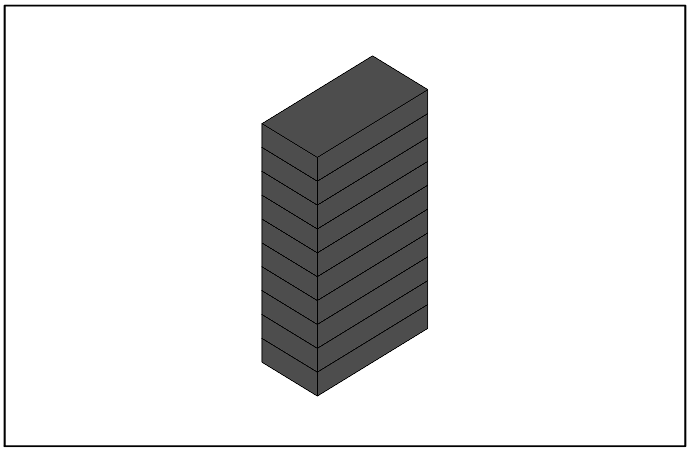
We observe that what we are doing is in making the nth brick is just BL([0,0,2*n]); and this depends only on the integer "n" (we are starting with n=0). So to get our list all we need do is the following. Again we terminate with a colon ":" as we don't want to see the list in this form - we only want the picture.

```
bricklist:=[seq(BL([0,0,2*n]),n=0..9)]:plots[
display](bricklist);
```



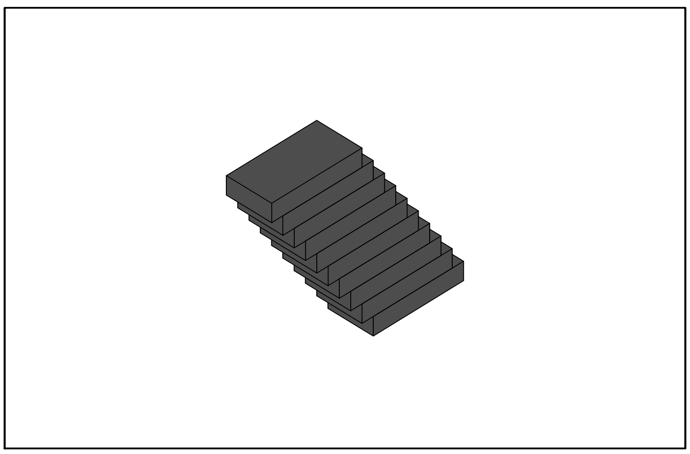
We can improve even further with a compound command which simply combines the previous steps

```
plots[ display ]([ seq(BL([0,0,2*n]), n=0..9)
]);
```



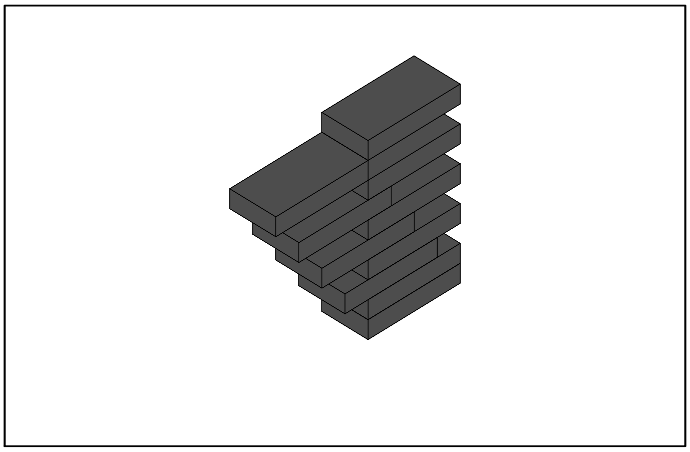
Suppose we want our bricks in a "staircase" pattern, say each brick is to shift over by one unit in the positive "X" direction. Since the description of the nth brick depends solely on "n" we can do it.

```
plots[display] ([seq(BL([n,0,2*n]),
n=0..9)]);
```



Suppose we want only the even numbered bricks to shift

```
plots[display] ([ seq(
BL([n*(1+(-1)^n)/2,0,2*n] ), n=0..9) ]);
```



5.6 EXERCISES: Making Brick Patterns With seq

5.6.1 Exercise: Stack the ten bricks in so that they shift both in the "X" and "Y" directions, making each brick after the first shift 1/3 of a unit in the positive "X" direction and 1/4 of a unit in the negative "Y" direction.

5.6.2 Exercise: Stack the ten bricks so that they don't shift in the "Y" direction but we want each odd numbered brick to be shifted 1 units more than the previous odd numbered brick in the positive "X" direction and the ith even numbered brick to be shifted 1 unit more than the previous even-numbered brick in the negative "X" direction.

5.6.3 Exercise: Using the "seq" command place the ten bricks so that are end to end.

5.6.4 Exercise: Using the "seq" command place the ten bricks so that they are in a rectangular pattern of five rows and two columns.

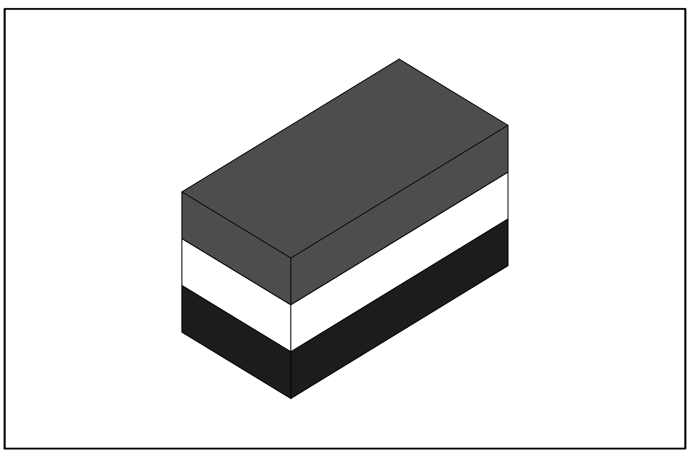
6 Stacking Colored Bricks:

An Introduction to for .. do ... od and if ... then ...fi Commands.

6.1 A Chromatic Brick Placer

Suppose we want to stack ten bricks but we want to alternate the colors. First we need to restore the color option to our "BL" command. Lets modify it to a "CBP" (chromatic brick placer) which allows us to specify the color.

```
CBP:= proc (shift,clr )
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [8, 0, 0]+shift; v3 := [8, 4, 0]+shift;
v4 := [0, 4, 0]+shift; v5 := [0, 0, 2]+shift; v6 := [8, 0, 2]+shift;
v7 := [8, 4, 2]+shift; v8 := [0, 4, 2]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=clr,style=patch, scaling=constrained )
end:

plots[display]([CBP([0,0,0],blue),CBP([0,0,2]
,white),CBP([0,0,4],red)]);
```



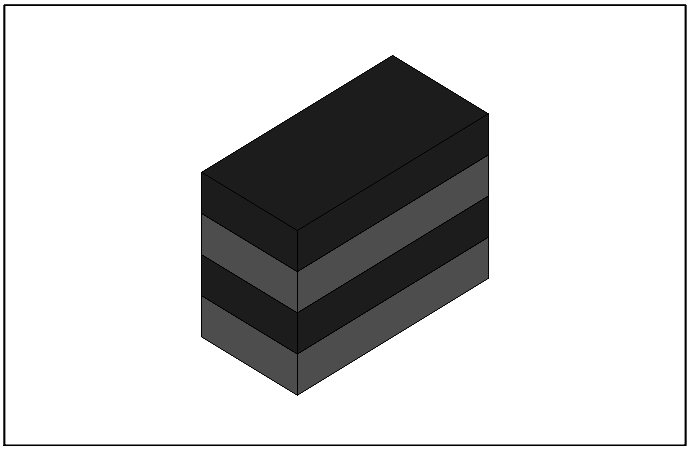
Suppose we want to stack alternating blue and red bricks. The obvious way to do it "by hand" would be to manufacture the bricks one at a time assemble them into a list and use plots[display] to view the result. Indeed lets do a few steps in the process to see what we want to automate. This time we want to assemble the list as we go rather than waiting until the end. This should keep us from having to give a permanent name to each one. We simply give each new brick a temporary name and append it to the list. Lets let "CBS" stand for colored brick stack.

```
CBS:=NULL;tmpbrick:=CBP([0,0,0],red):
CBS:=CBS,tmpbrick:
```

```
tmpbrick:=CBP([0,0,2],blue):
CBS:=CBS,tmpbrick:
tmpbrick:=CBP([0,0,4],red):
CBS:=CBS,tmpbrick:
tmpbrick:=CBP([0,0,6],blue):
CBS:=CBS,tmpbrick:
```

$$CBS :=$$

```
plots[display]([CBS]);
```



Like the other processes we have been working with recently we can see that this is terribly repetetive. All we are doing is repeating the two commands

tmpbrick =CBP([0,0,x], somecolor); CBS:=CBS,tmpbrick;

and we are doing it ten times.

What we need is the ability to tell the computer to do something a fixed number of times - in this case 10 times. The basic format for the command that does this is

for VARIABLE from STARTINGNUMBER to STOPPING NUMBER
do
"sequence of Maple Commands"
od;

You can always spot the end of a do statement with do spelled backwards. od

To have Maple print "n!" for n from 1 to 10 one could do the following

```
for n from 1 to 10 do n!  od;
```

To have Maple print "n" followed by "n!" for n from 1 to 10 one could do any of the following

$$1$$

$$2$$

$$6$$

$$24$$

$$120$$
$$720$$
$$5040$$
$$40320$$
$$362880$$
$$3628800$$

```
for n from 1 to 10 do [n,n!]  od;
```

$$[1, 1]$$
$$[2, 2]$$
$$[3, 6]$$
$$[4, 24]$$
$$[5, 120]$$
$$[6, 720]$$
$$[7, 5040]$$
$$[8, 40320]$$
$$[9, 362880]$$
$$[10, 3628800]$$

or

```
for n from 1 to 10 do print(n,n!)  od;
```

$$1, 1$$
$$2, 2$$
$$3, 6$$
$$4, 24$$
$$5, 120$$
$$6, 720$$
$$7, 5040$$
$$8, 40320$$
$$9, 362880$$
$$10, 3628800$$

or

```
for n from 1 to 10 do cat(`The factorial of
`,n,` is ` ,n!,`.`) od;
```

The factorial of 1 is 1.

The factorial of 2 is 2.

The factorial of 3 is 6.

The factorial of 4 is 24.

The factorial of 5 is 120.

The factorial of 6 is 720.

The factorial of 7 is 5040.

The factorial of 8 is 40320.

The factorial of 9 is 362880.

The factorial of 10 is 3628800.

To build an expression sequence called SS containing the squares of the integers from 62 to 73 we could do the following

```
SS:=NULL;
for w from 62 to 73 do SS:= SS,w^2 od;
```

$$SS :=$$

$$SS := 3844$$

$$SS := 3844,\ 3969$$

$$SS := 3844,\ 3969,\ 4096$$

$$SS := 3844,\ 3969,\ 4096,\ 4225$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184,\ 5329$$

If as is the case here we really don't care to see all of the intermediate steps we can suppress the intermediate display by terminating with a colon ":". Change the semicolons to colons. After, type an "SS;" to see the final sequence.

```
SS:=NULL:
for w from 62 to 73 do SS:= SS,w^2 od;
```

$$SS := 3844$$

$$SS := 3844,\ 3969$$

$$SS := 3844,\ 3969,\ 4096$$

$$SS := 3844,\ 3969,\ 4096,\ 4225$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900$$

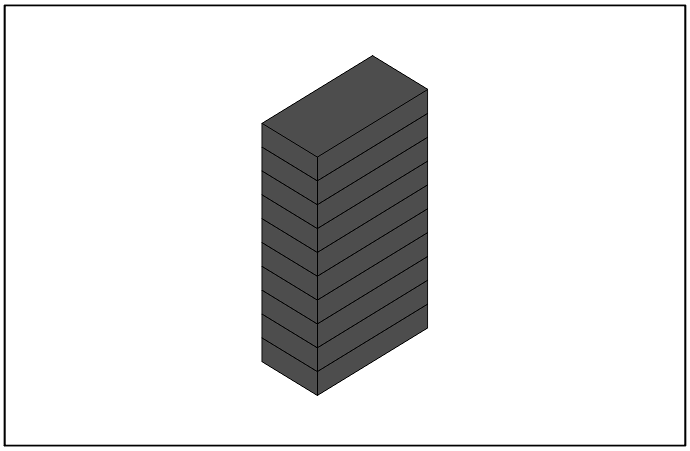
$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184,\ 5329$$

So, for instance to build a sequence of ten red blocks like the one we did with "seq" we could simply

```
bricks:=NULL: for t from 0 to 9 do
bricks:=bricks,BL([0,0,2*t]) od:plots[display]([bricks]);
```



6.2 Exercises: DO .. OD

6.2.1 *Exercise: Modify the following to produce a command sequence which prints out the first 100 powers of 2.*

```
for s from 0 to 5 do print(x,2*x) od;
```

6.2.2 *Exercise: Write a command sequence which prints out* $[n,\ (1+\frac{1}{n})^n]$ *for the first 100 values of n.*

6.3 Conditional statements

Now we would like to make a stack of differently colored bricks. We will use our "CBP" command in a do .. od constructionl but we need to add commands within the "loop" to tell Maple how to color the tiles.

Suppose we want to start with a red brick on the bottom and then alternate with blue blocks from then. We need a way to issue the following instructions to the computer.

"if t is even then make the brick red otherwise make it blue".

We do this with the if ... then ... else .. fi command. Just as we end the do statement with do spelled backwards an if..then .. statement is ended with if spelled backwards.

Like the do..od command the if..fi command is best explained with an example. Lets do the alternating red-blue bricks. There are lots of possible ways to do this. We will be very pedantic at first.

We need one additional tool. First recall that an integer is even if it yields a remainder of zero when divided by 2; it is odd if it leaves a remainder of 1 under the same circumstances. If we enter the command "a mod b;" Maple returns the remainder upon division of "a" by "b".

```
6 mod 2;
```

$$0$$

```
17 mod 2;
```

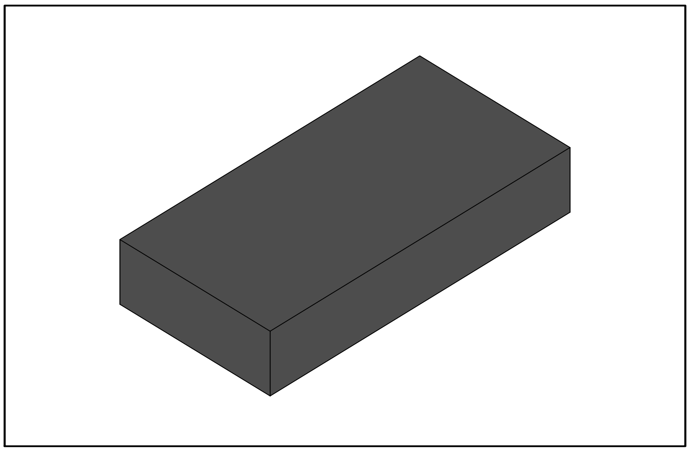
$$1$$

Now let "T" have some integral value and lets test whether its even or odd and make a red or blue brick accordingly

```
T:=17;
```

$$T := 17$$

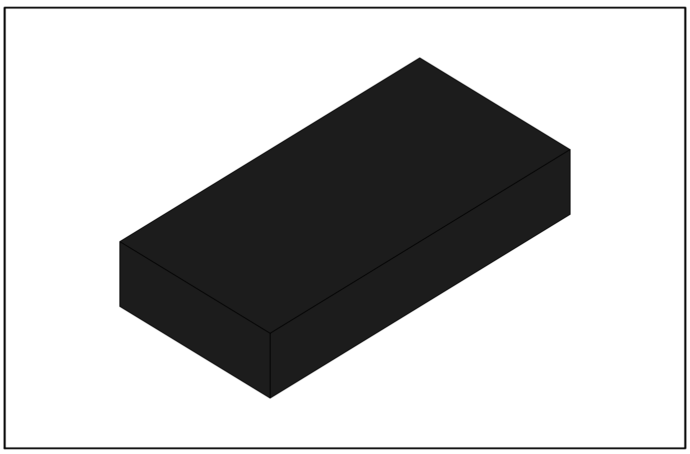
```
if T mod 2 = 1 then CBP([0,0,0],red) else
CBP([0,0,0],blue) fi;
```



```
T:=18;
```

$$T := 18$$

```
if T mod 2 = 1 then CBP([0,0,0],red) else
CBP([0,0,0],blue) fi ;
```

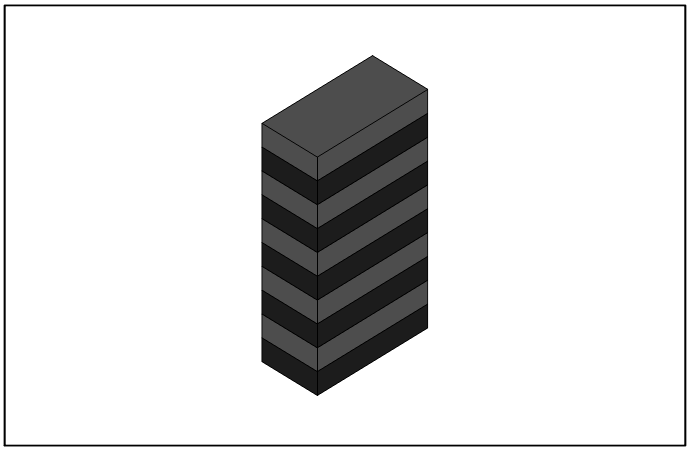


Note that it is often convenient to enclose expressions of the form **a mod b** in parenthesis. This is optional but it tends to make Maple code easier to follow.

Be especially careful to distinguish the **equality** = symbol in the above as opposed to the assignment symbol := . Although these look similar they are miles apart. The statement T := 17; assigns the value 17 to the name T. The equation T mod 2 = 1 is tested in the if .. fi statements above. Now all we have to do is modify the "body" of our "do..od" command which stacked ten red bricks

```
bricks:=NULL:
for t from 0 to 9 do
if (t mod 2) = 1 then
tmpbrick:=CBP([0,0,2*t],red) else
```

```
tmpbrick:=CBP([0,0,2*t],blue) fi:
bricks:=bricks,tmpbrick; od:
plots[display]([bricks]);
```

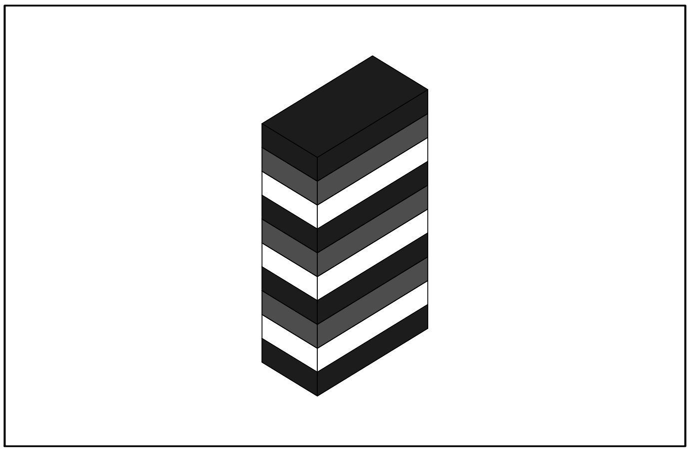


It is no harder to use alternating red, white, and blue bricks. We just need to know how to have Maple choose among three alternatives. The format is

if ...then...elif.. then ...elif ...then...fi;

Here is an example. Recall that the possible remainders upon division by 3 are 0,1, and 2.

```
bricks:=NULL:
for t from 0 to 9 do
if (t mod 3) = 2 then
tmpbrick:=CBP([0,0,2*t],red) elif (t mod 3) = 1 then
tmpbrick:=CBP([0,0,2*t],white) elif (t mod 3) = 0 then
tmpbrick:=CBP([0,0,2*t],blue) fi:
bricks:=bricks,tmpbrick; od:
plots[display]([bricks]);
```



The Boolean operators 'and' and 'or' can be used the with the if then elif then else ficonstruction in expressions like the following defintion.

```
compare:=proc(a,b)
if (a<0) or (b>1) then print( a , ‘is less than 0 or ‘,b,‘ is greater
than 1‘)
elif (a=0) and (b=0) then print(‘both are zero‘)
elif (a>0) then print(‘the first is positive‘)
else print(‘the other tests fail‘) fi:
end:
```

```
compare(-1,0);
```

$$-1, \textit{ is less than 0 or }, 0, \textit{ is greater than 1}$$

6.3.1 *Exercise: Is it possible to find numbers a and b so that 'the other tests fail' is the output?*

6.4 Exercises: If .. elif .. fi; and if .. else.. fi

6.4.1 *Exercise: The following procedure "tosses" a coin and reports "heads" or "tails". Modify it to have the computer toss two coins and report (heads,heads), (heads, tails), etc, Note that the procedure has no inputs - this is perfectly ok. Note also that the "rand" function used actually returns a procedure (called tmp) in below which itself has no variables.*

```
toss:=proc(x)
local coin:  coin:=rand(2):
if coin()=0 then print(‘heads‘) else print(‘tails‘) fi:
end:
```

6.4.2 *Exercise: Make a procedure which throws a six-sided die - that is one which returns a random digit from 1 to 6 (hint: die:= rand(1..6);)*

6.4.3 *Exercise: Adapt the following procedure to draw a red square if its input is between 0 and 1, a blue rectangle if its input is between 3 and 4, and a green triangle otherwise.*

```
drawit:=proc(n) local sq, tri, ans;
sq:=plots[polygonplot]([[0,0],[1,0],[1,1],[0,1]],
color=red,thickness=4);
tri:=plots[polygonplot]([[0,0],[1,0],[1,1]], color=blue,thickness=3);
if (n>=0) and (n<=2) then ans:=sq:
elif (n>=3) and (n<=5) then ans:=tri;
else ans := plots[polygonplot]([[0,0],[2,0],[2,1],[0,1]],
thickness=3);
fi:
ans;
end;
```

6.5 A vertical chromatic brick placer

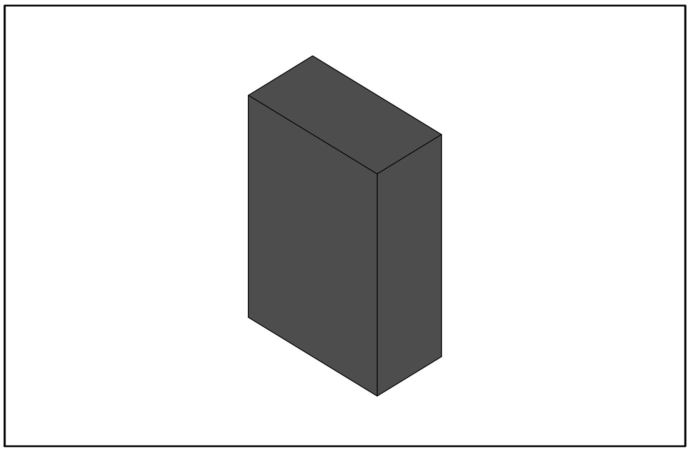
If we had vertical bricks we could make more elaborate patterns. Lets cook up a quick vertical chromatic brick placer called 'VCBP'.

```
VCBP := proc (shift,clr )
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0] + shift; v2 := [0, 0, 8] + shift; v3 := [0, 4, 8] +
shift;
v4 := [0, 4, 0] + shift; v5 := [2, 0, 0] + shift; v6 := [2, 0,
8]+shift;
v7 := [2, 4, 8]+shift; v8 := [2, 4, 0]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color = clr, style = patch, scaling =
constrained )
end:
```

We can test it with

```
VCBP([0,0,0],red);
```

Now we can begin to make more elaborate placement. Note that we are using both CPB and VCPB below - a plotting error will reult if either is not compiled



```
bricks:=NULL:
END:=0:
for t from 0 to 19 do
if (t mod 3) = 2 then tmpbrick:=VCBP([END,0,0],red);
END:=END+2;
elif (t mod 3) = 1 then tmpbrick:=VCBP([END ,0,0],white);
END:=END +2;
elif (t mod 3) = 0 then tmpbrick:=CBP([END,0,0],blue);
END:=END+8;
fi:
bricks:=bricks,tmpbrick; od:

plots[display]([bricks]);
```

```
Error, (in plots/object2plot) bad object to display, string
```

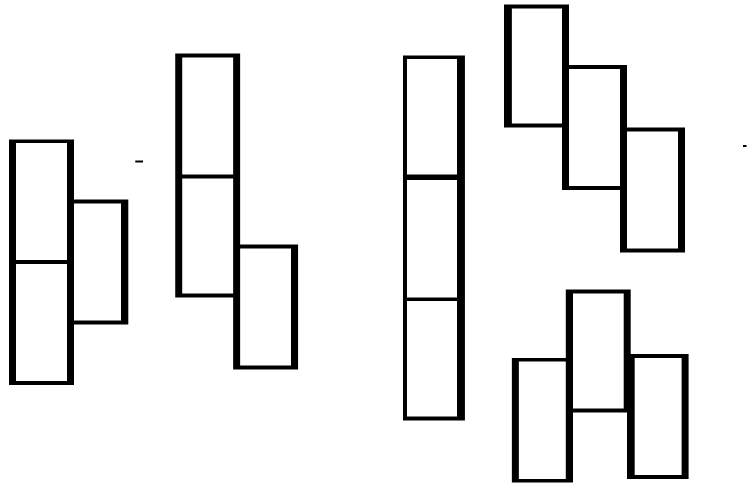
6.6 Exercises: Colored Brick Stacking

6.6.1 *Exercise: Make a stack of 15 bricks that alternate in color: white, red, blue, yellow - with a white brick on the bottom.*

6.6.2 *EXERCISE: Do the previous problem, making the stack end with a white brick on the top.*

6.6.3 *Exercise: Use a "do..od" command to stack sixteen bricks in eight layers such that each layer has two side-by-side bricks with each brick on every layer but the first lying across both bricks on the level below it.*

6.6.4 *EXERCISE: Place nine bricks in a line on the base level such that except for the ends each horizontal brick is flanked by two vertical bricks and each vertical brick is flanked by two horizontal bricks.*



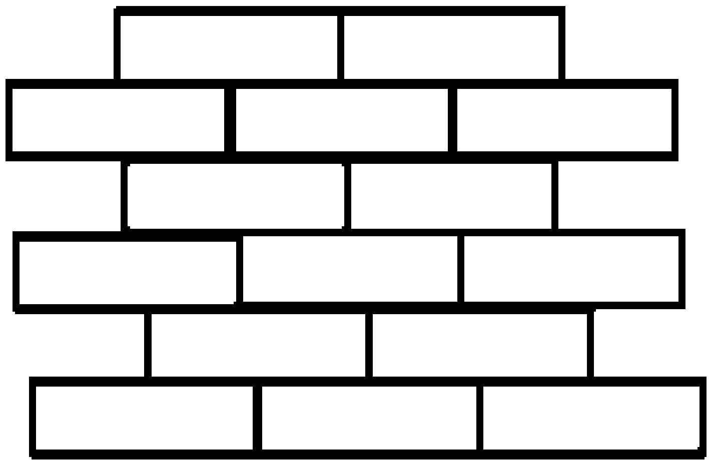
6.6.5 *EXERCISE: A puzzle called Brick by Brick which is marketed by the Binary Arts Corporation of Alexandria, Virginia consists of five pieces, each made of three bricks The left diagram shows a two dimensional sketch. The object of the puzzle is to assemble the five pieces in to various shapes. Pieces may be inverted or rotated 180 degrees. Thus for instance the upper left piece could be used in either of the orientations indicated at the right. One of the pieces has four orientations.*

6.6.5.1 *a. Make a three dimensional representation of each of the possible orientations of each piece.*

6.6.5.2 *b. Make a set of procedures, one for each possible orientation of one of the pieces, which will place any piece wherever you want it.*

6.6.5.3 *c. Merge your procedures into a single procedure which inputs the name of the piece, the orientation, where you want it, and a color.*

6.6.5.4 *d. Assemble the following pattern. Make one solution with the bricks all the same color and a second with each of the five pieces a different color. .*



6.6.5.5 *e. Is this arrangement possible?*

6.6.5.6

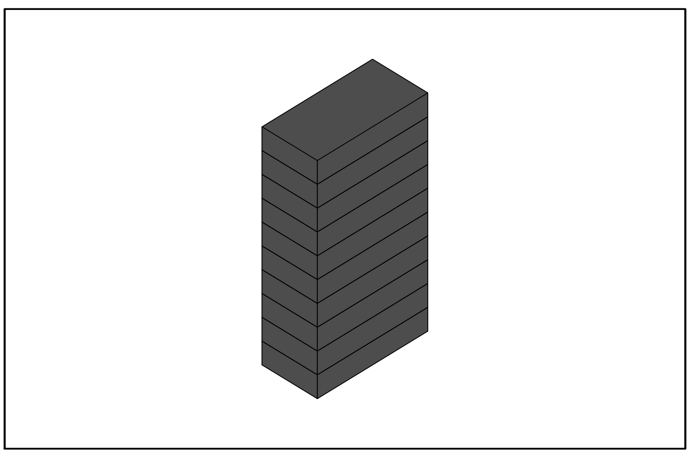
7 Making it Move - an Introduction to Animations

In previous chapters we have learned how to make lists of Maple plot structures and then to display them all together in the same frame as a single picture. There is a simple option to "plots[display]" called "insequence" which if set to "true" displays the pictures not all together but one at a time. This is all it takes to make movies. Lets start simply, by animating the stacking of a small pile of bricks. In Chapter 4 we made stacks of bricks using a word BL defined as follows:

```
BL := proc (shift)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1:=[0, 0, 0]+shift; v2 := [8, 0, 0]+shift; v3 := [8, 4, 0]+shift;
v4 := [0, 4, 0]+shift; v5 := [0, 0, 2]+shift; v6 := [8, 0, 2]+shift;
v7 := [8, 4, 2]+shift; v8 := [0, 4, 2]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=red,style=patch, scaling=constrained )
end:
```

```
bricklist:=[seq(BL([0,0,2*n]),n=0..9)]:
```

```
plots[display](bricklist);
```



Now lets try that last command with "insequence" set equal to "true". After executing the command "click" anywhere on the display to open a new "toolbar" and click on the large solid triangular icon or pull-down the "animation" menu which appears and select "play".

```
plots[display](bricklist,insequence=true);
```

Suppose we **want a brick to "slide"** along the x-axis. We simply generate a sequence of bricks occupying intermediate positions and display it "insequence =true". For example we could define

```
slider:= seq( BL([t/5,0,0]),t=0..20):
```

```
plots[display]([slider],insequence=true,
axes=boxed);
```

The brick would look better if it had a "road" to slide along. that just means we need a colored polygon "under" it.

```
road:=plots[polygonplot3d]([[-1,-1,0],[20,-1,
0],[20,5,0],[-1,5,0]],
color=blue,scaling=constrained,style=patch):
```

Now we just add the road to each frame:

```
slider:= seq(
plots[display]([road,BL([t/5,0,0])]),t=0..20):
plots[display]([slider],insequence=true);
```

Maybe we would like to see a brick "hop up and down". All we need to do is:

```
hopper:=seq( BL([0,0, 3*(t mod 2)]),t=0..10):
```

```
plots[display]([hopper],insequence=true);
```

Maybe we would like to have it simply sit there and alternate its colors. Recall that we have a word to make colored bricks

```
CBP := proc (shift,clr )
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [8, 0, 0]+shift; v3 := [8, 4, 0]+shift;
v4 := [0, 4, 0]+shift; v5 := [0, 0, 2]+shift; v6 := [8, 0, 2]+shift;
v7 := [8, 4, 2]+shift; v8 := [0, 4, 2]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8]; left := [v4, v8,
v5, v1];
right := [v3, v2, v6, v7]; bottom := [v3, v4, v8, v7]; top := [v1, v2,
v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=clr,style=patch, scaling=constrained )
end:
```

This time we **make the list to be animated** the "movie", as we go. Recall that it is usually best to build lists by buiding sequences and then converting them to lists with brackets "[]". We start with Maple's empty list "NULL"

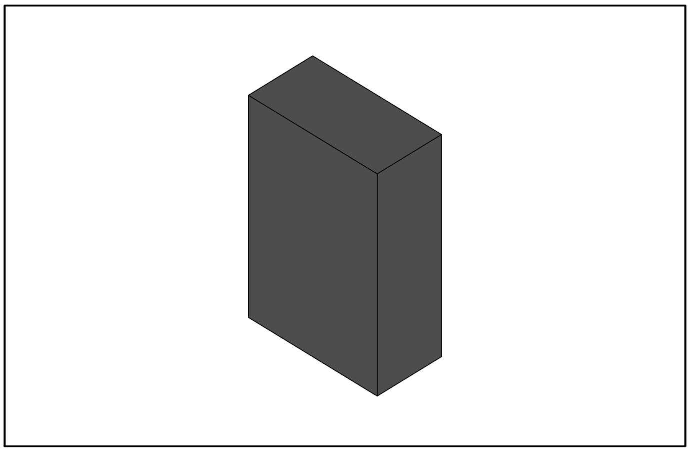
```
movie:=NULL:
for i from 1 to 10 do
if (i mod 2) = 0 then frame:=CBP([0,0,0],red)
else frame:=CBP([0,0,0],blue) fi;
movie:=movie,frame:
od:

plots[display](movie,insequence=true);
```

7.0.6 *The above illustrates a useful model for making animations We use an empty list called something like "MOVIE" and a word like "FRAME" to represent a typical frame of the movie. We have a sequence of Maple commands whose output is a plot structure and use a "do-loop" to construct the animation sequence one frame at a time:*

MOVIE:=NULL;
for INDEX from START to STOP do
{sequence of Maple commands which produces a plot structure "FRAME"}
MOVIE:=MOVIE,FRAME;
od:
plots[display]([MOVIE], insequence=true);

We can further illustrate this by making a movie of a brick moving "end-over-end". What we need to do is alternate horizontal and vertical bricks and account for the different translation distances. We need a vertical brick layer which we might as well call "VBL"

```
VBL := proc (shift,clr )
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [0, 0, 8]+shift; v3 := [0, 4, 8]+shift;
v4 := [0, 4, 0]+shift; v5 := [2, 0, 0]+shift; v6 := [2, 0, 8]+shift;
v7 := [2, 4, 8]+shift; v8 := [2, 4, 0]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=clr,style=patch, scaling=constrained )
end:

VBL([0,0,0],red);
```



Lets put a vertical brick down first and alternate horizontal and vertical after that. This means that, starting with 0 the even bricks will be vertical and the odd ones horizontal. We need to move over 2 inches after a vertical brick and eight after a horizontal

```
movie:=NULL: frame:=NULL: tmp:=[0,0,0]:
for i from 0 to 9 do
if (i mod 2) = 0 then frame:=VBL(tmp,red):  tmp:=tmp+[2,0,0];
else frame:=CBP(tmp,red):  tmp:=tmp+[8,0,0]:  fi;
movie:=movie,frame:
od:

plots[display]([movie],insequence=true);print
level:=0;
```

Suppose we would like to see a stack of bricks growing as each successive brick is added. What we need then is a new list whose first element is the picture of the first brick, whose second is a picture of the first and second bricks, whose third is the first three bricks, ... etc. Then, when we display this as an animation we will see the stack of bricks appear to grow. There are several ways to do this. First, if the list already exists it is easy to do it by hand. Lets make a list of the first three frames of the movie of this stack of bricks. An extremely useful bit of Maple notation first:

NOTATION: If "L" is a Maple list then the "nth" element of "L" is "L[n]"—

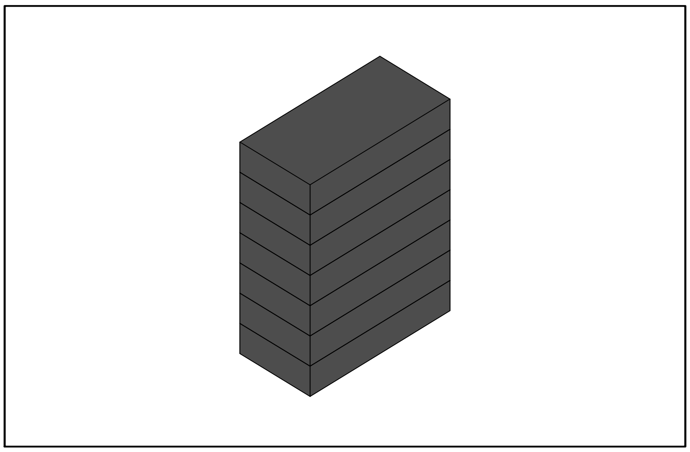
For instance "L[1]" is the first element, "L[2]" is the second, etc.

Recall that "plots[display]" wants a list of displays as input.

```
movielist:=[bricklist[1],
plots[display]([bricklist[1], bricklist[2]]),
plots[display]([bricklist[1],bricklist[2], bricklist[3]]) ]:

plots[display](movielist,insequence=true);
```

Obviously construction of this list would be tedious to do by hand if it were of any length. However, we know how to make the computer do that sort of thing. However before we are able to make the computer do anything we have to know how to do it at least in principle by hand. We first ask ourselves: "How do I build the seventh "frame" of the movie?" The answer is "Make a list in the same order of the first seven pictures in "bricklist" and feed it to "plots[display]". Thats easy - we do it the same way as we made the original stack of ten.

```
stack7:=plots[display]([seq(
bricklist[j],j=1..7)]):

stack7;
```

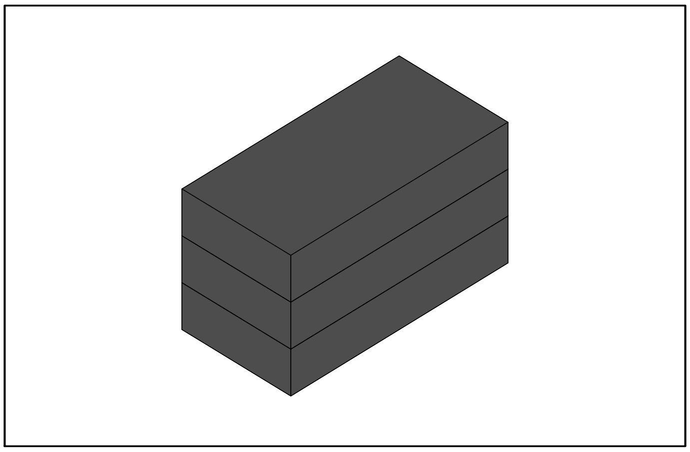


We can make a Maple word which makes the nth stack

```
stackn:=proc(n)
plots[display]([seq( bricklist[j],j=1..n)])
end;
```

$$stackn := \text{proc}(n)\ plots_{display}([\text{seq}(bricklist_j,\ j = 1..n)])\ \text{end}$$

```
stackn(3);
```



Now we can easily make our movie

```
movie:=[seq(stackn(t),t=1..10)]:

movie:=[op(movie),seq(movie[11-i],i=1..10)]:

plots[display](movie,insequence=true);
```

In the previous example we were dealing with an extant list of known length. What if we didn't know the length of "bricklist". We might ask Maple to display it by entering its name and try to count the entries. However it might be very long with each entry very complicated. There is a simple solution If "L" is a list then **nops (L);** returns the length of "L".

```
nops(bricklist);

nops(movie);
```

Thus this sequence works and would work even if we didn't know the length of "L"

```
movie:=[seq(stackn(t),t=1..nops(bricklist))]:

plots[display](movie,insequence=true);
```

Usually we don't encounter a list we want to animate but rather we have a process we want to illustrate and we make a "movie" by assembling the final list - the one which will produce the animation we want from the outset. We don't usually make the analog of "bricklist" but go straight to "movie". Suppose we want to make a 7-frame brick stacking movie. Recall that "NULL" is the expression sequence with no elements.

```
movie:= NULL ;
frame:=NULL;
```

```
for p from 1 to 7 do
tmp:= BL([0,0,2*p]);
frame:=plots[display]([frame,tmp]);
movie:=movie,frame;
od:

plots[display]([movie],insequence=true);
```

This illustrates a basic pattern one can use to animate a "growing" structure. We start with an empty sequence "MOVIE" and an empty sequence "FRAME".

```
MOVIE:=NULL;
FRAME:= NULL;
for INDEX from START to STOP do
`[sequence of Maple commands which generates a plot structure TMP]`;
FRAME:=plots[display]([FRAME,TMP]);
MOVIE:=MOVIE, FRAME ;
od:

plots[display]([MOVIE],insequence=true);
```

We can illustrate by making a movie which stacks alternate colored red and blue bricks, using the "CBP" word from above.

```
movie:=NULL: frame:=NULL:
for q from 1 to 10 do
if (q mod 2 = 0) then tmp:=CBP([0,0,2*q], red);
else tmp:= CBP([0,0,2*q], blue) fi;
frame:=plots[display]([frame,tmp]);
movie:=movie,frame;
od:

plots[display]([movie],insequence=true);
```

7.1 Exercises on Movie Making

7.1.1 Exercise: *The following makes a movie that counts from 0 to 10.*

```
MOV:=NULL;
for k from 0 to 10 do
txt:=cat( k,` `,`and`,` `,`counting`):
MOV:=MOV, plots[textplot]([0,0,txt], axes=none, font=[TIMES,
BOLD,28]): od:

plots[display]([MOV],insequence=true);
```

7.1.2 *a. Modify the above to count to 50.*

7.1.3 *b. The following will make a sequence which reverses MOV*

```
VOM:=seq(MOV[11-i],i=0..10):
```

7.1.4 *Put the sequences MOV and VOM together into a movie (by combining the two sequences into one) which counts up to 10 then back down to 0.*

7.1.5 *c, Make a movie that counts from 3 to 9 then down to -5.*

7.1.6 Exercise (Movie Splicing):

7.1.7 *a. Modify the following movie from the chapter text to make 10 frame movie of bricks stacking. Call it "tenbricks".*

```
sevenbricks:= NULL ;
frame:=NULL;
for p from 1 to 7 do
tmp:= BL([0,0,2*p]);
frame:=plots[display]([frame,tmp]);
sevenbricks:=sevenbricks,frame;
od:

plots[display]([movie],insequence=true);
```

7.1.8 *b. Make a movie called "OnetoTen" which counts from 1 to to by making a sequence of all but the first frame of the movie "MOV" in the previous problem.*

7.1.9 *c. Splice the two movies together by making a 10-frame sequence "MergedMovie" in which the ith element is plots[display[(MOV[i], OnetoTen[i]);*

7.1.10 *d. When "MergedMovie" is played the text meets the bricks. Look at the code for "MOV" and determine the computer is being told to place the text. Modify the location of the text so that it sits below the brick stack with the stack approximately over the center of the text.*

7.1.11 Exercise: *As in the chapter text the following makes a movie of alternate red, white, and blue bricks stacking.*

```
movie:=NULL: frame:=NULL:
for q from 1 to 10 do
if (q mod 2 = 0) then tmp:=CBP([0,0,2*q], red);
else tmp:= CBP([0,0,2*q], blue) fi;
frame:=plots[display]([frame,tmp]);
movie:=movie,frame;
od:
```

7.1.12 *If one selects the 'continuous' option in the animation menu the movie will cycle when you play it. The motion is not smooth however, because the movie is played from beginning to end over and over. One would like to have the bricks stack up and then "unstack". Do this by making a movie which is twice as long as this one, having the bricks unstack in the last half of the movie so that when you play it continuously, the motion is smooth. this can be done without making any more images.*

7.1.13 Exercise: *Make the hopping brick hop along a yellow road. Let each hop cover one brick length with three frames of the brick "in the air" beween frames in which it is "on the road".*

7.1.14 *Exercise: Make the brick move end over end along a yellow road.*

7.1.15 Exercise: *Make a pair of bricks, a blue one and a red one move in a straight line with each one "leap frogging" the other.*

8 New Problems from Old-A Case Study

One of the best ways to develop new problem contexts is to take old, familiar problems and look at them in new ways. Developing alternate perspectives is as much art as science. However there are some general techniques. One of the simplest is taking an old war horse of a problem and producing a visual representation it and/or its solution. One almost always finds that this leads to new, interesting problems. Here is an example:

8.1 The lawn mowing problem

8.1.1 *Problem: Sam can mow the lawn in 2 hours and Bill can mow it in 3. How fast can they mow it together?*

Solution: The problem implicitly assumes that Sam and Bill can work independently. When this holds their rates add. Sam can mow one lawn in 2 hours so his rate is $\frac{1\,lawn}{2\,hours} = \frac{1}{2}$ lawn per hour. Similarly Bill has a rate of $\frac{1}{3}$ lawn per hour. Together their rate is $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$ lawn per hour so together they can mow one lawn in $\frac{5}{6}$ hour. Similar problems with essentially the same solution can be stated for such things as two pipes filling a tank, two boys eating a pizza, etc. and of course the extension to more than two is no problem, with the assumption of independence of action any number of rates add. Traditional extensions don't go much further than considering more than two mowers, pipes, etc.

8.2 Visual problems arising from the lawn mowing problem.

There any number of ways in which we might create a visual representation of this problem. We could do something as simple as making three bricks whose lengths represent the time Sam, Bill, and the two together could mow the lawn. This would be a straightforward application of what we have already done but it wouldn't be very interesting. For instance, it would not reflect the visual image engendered by the original question. It would apply equally well to two pipes that could individually drain a tank in 2 and 3 hours. Moreover a static image would not convey the sense of time or the feeling of motion conveyed by the mowing of a lawn.

8.2.1 *Let us start by trying to convey the notion of one individual mowing one lawn*

Although our work has been in three dimensions up to now the two dimensional case of this problem is simpler and less consumptive of computing resources so we start there.

. First we need a lawn:

```
v1:=[0,0]:  v2:=[1,0]:
v3:=[1,1]:  v4:=[0,1]:
lawn:=[v1,v2,v3,v4]:

plots[polygonplot ](lawn,color=green,
style=patch);
```

A better lawn might have border or a more interesting shape but a simple, square lawn is a good place to start.

We can vary the meaning of v1,v2,v3,v4 and have different lawns. We will want to use different colors to represent "unmown" and "mown" grass. Following what now a standard procedure we produce a word which produces a quadrilateral with specified vertices and color.

```
lawn := proc (v1,v2,v3,v4, clr )
plots[polygonplot]([v1,v2,v3,v4],color=clr,style=patch,scaling=constra
ined )
end:
```

Now lets make a lawn that is 10 by 10 with the lower left corner at the origin and with "grass" 1 unit thick.

```
L1:=lawn([0,0],[10,0],[10,10],[0,10],
green):
```

```
L1;
```

If we want to shift this lawn so that its lower left corner is at [11,5] we can

```
L2:=lawn([0,0]+[11,5],[10,0]+[11,5],[10,10]+[
11,5],[0,10]+[11,5], green):
```

Now display both lawns together

```
plots[display]([L1,L2]);
```

If we decide that "unmown" grass is aquamarine and "mown" grass is green then we can indicate a partially mown lawn by laying a "mown" green plot on top of an "unmown" aquamarine one. We could just as easily have adjoining plots of different colors but our there will be less calculating further on if we do it this way.

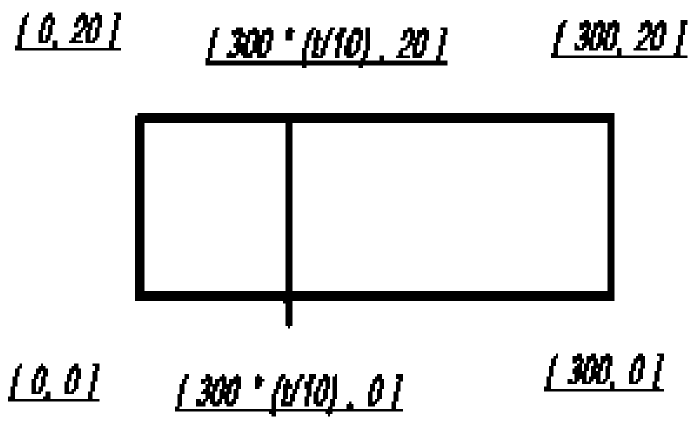
```
L1:=lawn([0,0],[10,0],[10,10],[0,10],
aquamarine):
L2:=lawn([0,0] ,[4,0] ,[4,4] ,[0,4] , green):
```

```
plots[display]([L2,L1]);
```

Note in the plots[display]([L2,L1]) the "L2,L1" order. This tells the computer to lay L1 down first and L2 on top of it. If we do things in the opposite order there will be a difference. Change the order and note the difference.

Now we have all we need to animate mowing a lawn - all we need to do is successively enlarge the "mown" portion until it covers the lawn. We "mow" it a strip at a time, combining the strips we have done into a single rectangle. First lets mow a strip.

Suppose, we have a strip which is 2 units wide and 30 long and wish to animate mowing it with a lawnmower 2 units wide over the course of 10 units of time. We can draw this sketch. Letting "t" vary from 0 to 10 we note that the middle line in our sketch varies from the left edge to the right of the larger rectangle. Thinking of this as the boundary of our mown and unmown grass we can, for any value of t between 0 and 10 delineate the two.



For instance if "t" were equal to "7" then we should be able to represent 70% of the strip as having been mowed. While we are at it lets use different colors for "mown" and "unmown" grass.

```
t:=7;
L1:=lawn([0,0],[30 ,0],[30 ,2 ],[0,2 ], aquamarine):
L2:=lawn( [0,0], [30 *(t/10),0],[30 *(t/10),2 ],[0,2], green ):
plots[display]([L2,L1]);
```

To animate mowing this strip all we need to do is run t from 0 to 10.

```
movie:=NULL:
L1:=lawn([0,0],[30 ,0],[30 ,2 ],[0,2 ], aquamarine):
for t from 0 to 10 do
movie:=movie,plots[display]([lawn( [0,0], [30 *(t/10),0],[30
*(t/10),2 ],[0,2], green ),L1]):
od:

plots[display]([movie],insequence=true);
```

Not overly spectacular but it does work.

This strip is being mown from left to right but we will need also to mow from right to left. This simply means that we have to shift the mown portion to the right side of the rectangle rather than the left.

```
movie:=NULL:
L1:=lawn([0,0],[30 ,0],[30 ,2 ],[0,2 ], aquamarine):
for t from 0 to 10 do
movie:=movie,plots[display]([lawn( [30 *(1-(t/10)),0], [30,0],
[30,2], [30 *(1-(t/10)),2 ] , green ),L1]):
od:
plots[display]([movie],insequence=true);
```

The reader should note the simple changes which reversed the direction and make sure that he/she understands them.

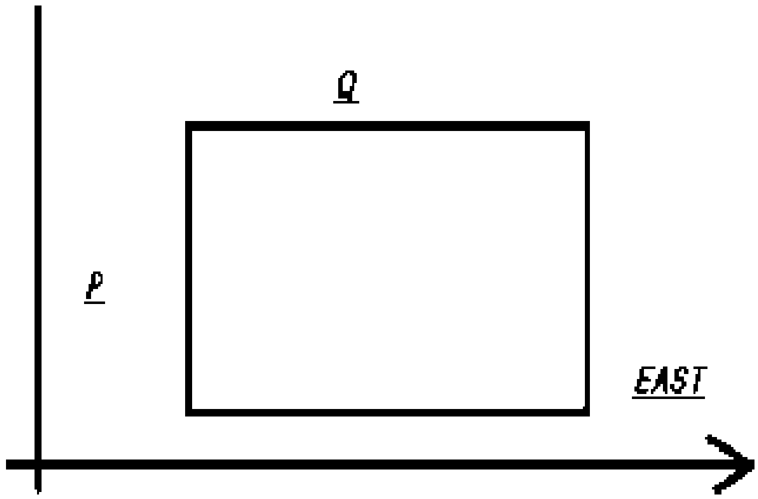
8.3 A specific problem.

Now we have all of the tools we need to animate the mowing of a simple lawn.

8.3.1 *Lets consider the problem of a 10 foot by 30 foot lawn which is to be mown in 20 minutes with a mower that cuts a path two feet wide.*

We have in mind a person mowing the lawn who starts in the lower, left corner and proceeds in the normal manner, reversersing direction each time he or she reaches the opposite edge. We want to make a picture of the lawn at each integer value of t from 0 to 20, a total of 21 frames, and to display them as a movie. We have already seen that there is no problem in keeping track of the mown and unmown portion of the strip the mower happens to be on. The only real problem is in how to know which direction the mower is moving at any particular time and how to represent that idea geometrically. Here is a statement of what we need to know as an elementary arithmetic problem. It is typical of the types of elementary problems we must solve in order to represent processes of this type.

8.3.2 Arithmetic Problem: *A person is mowing a lawn which is Wdth units wide by Lngth units long . Starting at a corner and mowing along an edge of length Lngth she heads due east. Upon reaching the opposite side she reverses direction and mows another strip of length Lngth which is contiguous to the previous one. It takes TOTALTIME minutes to mow the lawn in " n " strips. At each time t between 0 and TOTALTIME, on which strip is she mowing, what direction is she headed, and what fraction of the strip she is on remains to be done?*



Solution: For a time t between 0 and TOTALTIME write $\frac{t\,n}{TOTALTIME} = a.b$, a decimal number. Then "a" is an integer between 0 and n-1. At time "t" the mower is on the (a+1)st strip and the portion remaining to be done is "1 - 0.b". Since she is going east on the first strip (when a=0) and alternates directions after that she is going east when "a" is even and west when it is odd.

The Maple command for the integer part of a number, the greatest integer function, is called **floor.**

The remainder of an integer "m" upon division by two is **m mod 2**

```
floor(3.2);
floor(3.2) mod 2;
```

$$3$$

$$1$$

With the notation of the Arithmetic Problem our original problem has Wdth=10, Lngth=30, n=5, TOTALTIME=20. Ultimately we will make a 21 frame animation but the cardinal rule is always develop an individual frame first. We start with, say, t=9. then for this frame

```
t:=9; n:=5; TOTALTIME:=20;Wdth:=10;
Lngth:=30;
a:= floor(t*n/TOTALTIME);
b:=t*n/TOTALTIME - a;
a;
```

$$t := 9$$

$$n := 5$$

$$TOTALTIME := 20$$

$$Wdth := 10$$

$$Lngth := 30$$

$$a := 2$$

$$b := \frac{1}{4}$$

2

This tells us that the third strip is being mown, moving east (since "a" is even). One forth of the strip has been mown and 3/4 remains to be mown.

Now we can animate the lawn being mown. Each frame correesponds to a value of t from 0 to TOTALTIME. For the frame at time t we need to:

- 1. Calculate which strip we are mowing, the direction and what portion of the strip has been mown.
- 2. Make a rectangle green MOWN consisting of all complete strips mown
- 3. Make a green rectangle STRIPMOWN consisting of the portion of the current strip that has been mown then the frame is [STRIPMOWN, MOWN, UUNMOWN] where UNMOWN is the complete lawn colored aquamarine.

Here is a first try at the code

```
TIME:=9; NUMSTRIPS:=5;
TOTALTIME:=20;Wdth:=10; Lngth:=30;movie:=NULL:

YARD:=lawn([0,0],[30 , 0],[30 ,10 ],[ 0,10
],aquamarine):
for TIME from 0 to TOTALTIME do
WHICHSTRIP:= floor(NUMSTRIPS*TIME/TOTALTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/TOTALTIME) - WHICHSTRIP;
MOWN:=lawn( [0,0],[30,0],[30,2*WHICHSTRIP],
[0,2*WHICHSTRIP] , green):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawn( [0,2*(WHICHSTRIP )] ,
[30 *FRACSTRIP,2*(WHICHSTRIP )], [30*FRACSTRIP,
2*(WHICHSTRIP+1) ], [0,2*(WHICHSTRIP+1)], green ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawn( [30 *(1-FRACSTRIP),
2*(WHICHSTRIP )], [30,2*(WHICHSTRIP )],
[30,2*(WHICHSTRIP+1)], [30 *( 1- FRACSTRIP),
2*(WHICHSTRIP+1) ] , green ) ],YARD):
fi:
movie:=movie,FRAME:
od:

plots[display]([movie],insequence=true);
```

This works pretty well but but we need to test it by varying the values of the parameters. Lets check it with some new ones. As is our procedure we make another copy of the working code rather than tampering with it.

```
NUMSTRIPS:=7; TOTALTIME:=25;Wdth:=37;
Lngth:=30;


movie:=NULL:
YARD:=lawn([0,0],[30 , 0],[30 ,10 ],[ 0,10 ], aquamarine):
for TIME from 0 to TOTALTIME do
WHICHSTRIP:= floor(NUMSTRIPS*TIME/TOTALTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/TOTALTIME) - WHICHSTRIP;
MOWN:=lawn( [0,0],[30,0],[30,2*WHICHSTRIP],
[0,2*WHICHSTRIP] ,green):  if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawn( [0,2*(WHICHSTRIP )] ,
[30 *FRACSTRIP,2*(WHICHSTRIP ) ],
[30*FRACSTRIP,2*(WHICHSTRIP+1) ],
[0,2*(WHICHSTRIP+1)], green ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawn(
[30 *( 1-FRACSTRIP),2*(WHICHSTRIP )],
[30,2*(WHICHSTRIP )], [30,2*(WHICHSTRIP+1)],
[30 *( 1- FRACSTRIP),2*(WHICHSTRIP+1) ] , green ) ],YARD):
fi:
movie:=movie,FRAME:
od:

plots[display]([movie],insequence=true);
```

Hmmm.... little problem here. What happened caused it to work for one set of data and not for another is that we had "hard coded" some "magic numbers". One of them was "2" the width of the strip. We told the computer to mow a 37 foot wide lawn in 7 strips two foot wide strips. It got confused. We correct that by letting ir calculate the width of its mower as the ratio of the width of the lawn to the number of strips.

```
movie:=NULL:
YARD:=lawn([0,0],[30 , 0],[30 ,10 ],[ 0,10 ], aquamarine):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to TOTALTIME do
WHICHSTRIP:= floor(NUMSTRIPS*TIME/TOTALTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/TOTALTIME) - WHICHSTRIP;
MOWN:=lawn( [0,0],[30,0],[30,MRWDTH*WHICHSTRIP],
[0,MRWDTH*WHICHSTRIP] ,green):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawn( [0,MRWDTH*(WHICHSTRIP )] ,
[30 *FRACSTRIP,MRWDTH*(WHICHSTRIP )],
[30*FRACSTRIP,MRWDTH*(WHICHSTRIP+1)],
[0,MRWDTH*(WHICHSTRIP+1)], green ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawn(
[30 *( 1-FRACSTRIP),MRWDTH*(WHICHSTRIP )],
[30,MRWDTH*(WHICHSTRIP )], [30,MRWDTH*(WHICHSTRIP+1)],
[30 *( 1- FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] , green ) ], YARD):
fi:
movie:=movie,FRAME:
```

```
od:

plots[display]([movie],insequence=true);
```

Pretty close now - we forgot the other "magic numbers" "30" and "10" in the code. They were the length and width of our original lawn. All we have to do is substitute the variables "Wdth" and "Lngth" for them:

```
movie:=NULL:
YARD:=lawn([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to TOTALTIME do
WHICHSTRIP := floor(NUMSTRIPS*TIME/TOTALTIME);
FRACSTRIP := TIME*(NUMSTRIPS/TOTALTIME) - WHICHSTRIP;
MOWN := lawn( [0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],
[0,MRWDTH*WHICHSTRIP] ,green):
if WHICHSTRIP mod 2 = 0 then
FRAME := plots[display]([MOWN, lawn(
[0,MRWDTH*(WHICHSTRIP )] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )],
[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1)],
[0,MRWDTH*(WHICHSTRIP+1)], green ) ],YARD):
else
FRAME := plots[display]([MOWN, lawn(
[Lngth *( 1-FRACSTRIP),MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)],
[Lngth *( 1- FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] ,
green ) ],YARD):
fi:
movie:=movie,FRAME:
od:

plots[display]([movie],insequence=true);
```

This works pretty well. The reader will have noticed the small "antenna" that sprouts at the end of the mowing. In some sense its a "ghost" mower waiting to start a strip thats not there. To look for it change the loop in the code to run from TIME=0 to TIME = TOTALTIME -1 (one less frame) and see what happens. This makes it apparent that the artifact is generated in the last frame. We want the final view to be a completely mown lawn anyhow so we just modify the code accordingly.

```
movie:=NULL:
YARD:=lawn([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine):
MOWEDYARD:=lawn([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
green):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to TOTALTIME-1 do
WHICHSTRIP:= floor(NUMSTRIPS*TIME/TOTALTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/TOTALTIME) - WHICHSTRIP;
```

```
MOWN:=lawn( [0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],
[0,MRWDTH*WHICHSTRIP] ,green):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawn( [0,MRWDTH*(WHICHSTRIP )] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )],
[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1)],
[0,MRWDTH*(WHICHSTRIP+1)], green ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawn(
[Lngth *( 1-FRACSTRIP),MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)],
[Lngth *( 1- FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] ,
green ) ],YARD):
fi:
movie:=movie,FRAME:
od:
LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:

plots[display]([movie],insequence=true);
```

8.3.1 A lawn mowing word

Finally, we make this into a procedure which accepts the dimensions of a lawn, the number of strips, and the total time it takes to mow it (i.e. the number of frames in the movie) and returns a movie of that lawm being mowed in that time.

```
lawnmower:=proc(Lngth, Wdth, TOTALTIME,
NUMSTRIPS)
local movie, YARD, MOWEDYARD, MRWDTH, TIME, WHICHSTRIP, FRACSTRIP,
MOWN, FRAME, LASTFRAME;
Digits :=4:
movie:=NULL:
YARD:=lawn([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine):
MOWEDYARD:=lawn([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
green):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to TOTALTIME-1 do
WHICHSTRIP:= floor(NUMSTRIPS*TIME/TOTALTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/TOTALTIME) - WHICHSTRIP;
MOWN:=lawn( [0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],
[0,MRWDTH*WHICHSTRIP] ,green):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawn( [0,MRWDTH*(WHICHSTRIP )] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )],
[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1)],
[0,MRWDTH*(WHICHSTRIP+1)], green ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawn(
[Lngth *( 1-FRACSTRIP),MRWDTH*(WHICHSTRIP )],
```

```
[Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)],
[Lngth *( 1- FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] ,
green ) ],YARD):
fi:
movie:=movie,FRAME:
od:
LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:
plots[display]([movie],insequence=true):
end:

lawnmower(38,21,20,10); :
```

8.3.2 Two Mowers mowing side by side.

At this time we can momentarily revisit the original problem, and seek an animation of both mowers mowing.

8.3.3 Problem: *Make a movie of Sam and Bill mowing their respective lawns which shows two lawns side by side. Both Sam and Bill should start mowing at the beginning of the movie. The movie should end when the slowest mower completes his lawn.*

Let us assume that Sam and Bill who stared this are both around and we want to stage a "race" with them mowing side by side lawns. Our " lawnmower" does the two separately. The interesting problem will be to make them do it side by side. First we want to modify our "lawnmower " to take separately the number of time units the mower needs to mow the lawn, MOWINGTIME and the number of time units the movie is to represent MOVIELENGTH. If the number of movie units is greater than the number of units it takes to mow the lawn then the movie should show the lawn being mowed in the correct number of units and should simply show a mown lawn for the remaining frames. If there are not enough movie time units then the movie should end with the lawn incompletely mown. Of course we want to do this by adapting our "lawnmower code".

What was originally our TOTALTIME we are now calling MOVIELENGTH and the MOWINGTIME is the number of frames actually needed to complete the lawn. If MOWINGTIME is less than the MOVIELENGTH then the mower finishes early in the movie and the last few frames show a mown lawn. If the MOWINGTIME is greater than the MOVIELENGTH then the movie runs out of frames before the lawn is completely mown. What we want to do is use MOVIELENGTH in some places where we use TOTALTIME in the original code and MOWINGTIME in others. We will certainly need to change our LASTFRAME because it now ends the movie with a completely mown lawn. Now if the movie runs out we want to see the unmown remainder. The reader should note the very small number of changes we have made to the working code and should be sure to understand what each is doing.

```
lawnmower2:=proc(Lngth, Wdth, MOWINGTIME,
MOVIELENGTH, NUMSTRIPS)
local movie, YARD, MOWEDYARD, MRWDTH, TIME, WHICHSTRIP, FRACSTRIP,
MOWN, FRAME, LASTFRAME;
movie:=NULL:
YARD:=lawn([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
```

```
aquamarine):
MOWEDYARD:=lawn([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
green):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to MOVIELENGTH-1 do
if TIME <=MOWINGTIME then
WHICHSTRIP:= floor(NUMSTRIPS*TIME/MOWINGTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/MOWINGTIME) - WHICHSTRIP;
MOWN:=lawn([ 0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],
[0,MRWDTH*WHICHSTRIP] ,green):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawn( [0,MRWDTH*(WHICHSTRIP )] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )],
[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1) ], [0,MRWDTH*(WHICHSTRIP+1)],
green ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawn(
[Lngth *( 1-FRACSTRIP),MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)],
[Lngth *( 1- FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] ,
green ) ],YARD):
fi:
movie:=movie,FRAME:
else
movie:=movie,MOWEDYARD:
fi:
od:
if MOVIELENGTH >= MOWINGTIME then LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:
plots[display]([movie],insequence=true):
fi:
end;

lawnmower2(30,20,7,20,10);
```

Now if we want side-by-side mowers we need different lawns. This we need to introduce a shift which moves the original lawn to a different location. BEFORE we start directly into this however we examine what we have done and notice that the only way that coordinates are used is in the word "lawn". Thus if we simply adapt the word "lawn" to accept a shift (it probably should have had one anyhow). We are also changing our porgram so that it returns the movie rather than plays it. This is because we are going to want to run two movies at the same time so we will need to "splice" them together.

```
lawnwithshift := proc (v1,v2,v3,v4, clr,
shift)
plots[polygonplot]([v1+shift,v2+shift,v3+shift,v4+shift],
color=clr,style=patch,scaling=constrained ) end:

lawnmower3:=proc(Lngth, Wdth, MOWINGTIME,
MOVIELENGTH,
```

```
NUMSTRIPS,SHIFT)
local movie, YARD, MOWEDYARD, MRWDTH, TIME, WHICHSTRIP,
FRACSTRIP, MOWN, FRAME, LASTFRAME;
movie:=NULL:
YARD:=lawnwithshift([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine,SHIFT):
MOWEDYARD:=lawnwithshift([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth
],
green,SHIFT):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to MOVIELENGTH-1 do
if TIME <=MOWINGTIME then
WHICHSTRIP:= floor(NUMSTRIPS*TIME/MOWINGTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/MOWINGTIME) - WHICHSTRIP;
MOWN:=lawnwithshift(
[0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],
[0,MRWDTH*WHICHSTRIP] ,green,SHIFT):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawnwithshift( [0,MRWDTH*(WHICHSTRIP
)] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )],
[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1) ],
[0,MRWDTH*(WHICHSTRIP+1)], green,SHIFT ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawnwithshift(
[Lngth *( 1-FRACSTRIP),MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)],
[Lngth *( 1- FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] ,
green,SHIFT ) ],YARD):
fi:
movie:=movie,FRAME:
else
movie:=movie,MOWEDYARD:
fi:
od:
if MOVIELENGTH >= MOWINGTIME then
LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:plots[display]([movie],insequence=true):
fi:
[movie];
end;

MOVIE:=lawnmower3(30,20,13,20,10,[32,0]):
plots[display]( MOVIE,insequence=true);
```

Now lets assume that Sam and Bill are next door neighbors with idential 20 by 30 lawns. They each have a lawnmower which will mow their lawn in 7 strips. Sam can mow his lawn in 15 time units and Bill can mow his in 30. We can view each of them mowing his lawn with what we have:

```
SAMMOWING:=lawnmower3(30,20,15,15,7,[0,0]):
plots[display](SAMMOWING,insequence=true):
```

```
BILLMOWING:=lawnmower3(30,20,30,30,7,[32,0]):
plots[display](BILLMOWING,insequence=true);
```

If we are going to watch both of them we want to run the movie for the length of time it takes the slowest person to do the job. Thus we need to make Sam's movie run 30 units.

```
SAMMOWING:=lawnmower3(30,20,15,30,7,[0,0]):
plots[display](SAMMOWING,insequence=true);
```

Now we have two movies of the same length. Check that with nops.

```
nops(SAMMOWING); nops(BILLMOWING);
```

So now we can view Sam and Bill mowing their lawns side by side if we can solve the problem of viewing two movies at the same time.

8.3.4 Problem: *(Movie splicing problem). Given two movies (as lists of plots) how can we view them simultaneously?*

Solution: We use seq. Each of the two movies is simply a list of plot structures. We simply want to make a new list in which the ith element of the new list is the display of both the ith elements of each list. We may as well illustrate on the ones we have.

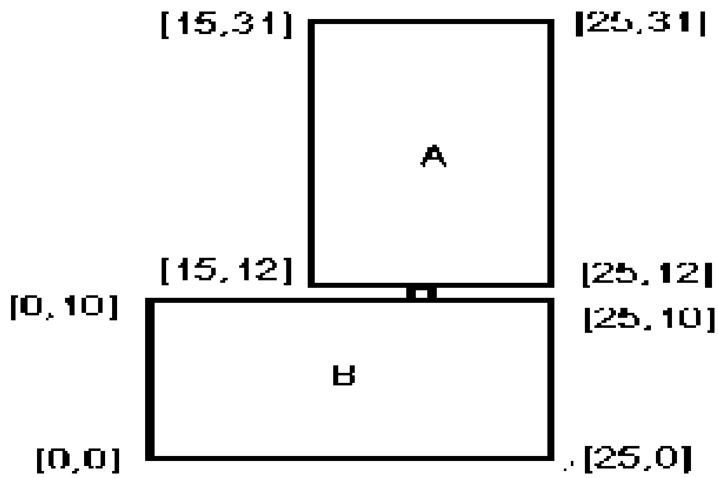
```
MOWINGSIDEBYSIDE:=seq(plots[display]([SAMMOWI
NG[i],
BILLMOWING[i]]), i=1..nops(SAMMOWING)):

plots[display]([MOWINGSIDEBYSIDE],insequence=true);
```

8.3.3 Exercises

8.3.5 Exercise: *Suppose A and B are rectangular tanks with identical dimensions, only B is below A and rotated 90 degrees. At a certain time A is full and B is empty, then the valve between them is opened and all the fluid flows into B at a constant rate determined by the valve. Make a 20 frame movie showing the fluid level in each tank with A full at the beginning and B full at the end.*

Beginning of Solution:

We aren't given the dimensions of the tanks so we will assume some. The spacing isn't given either, nor the orientations. In this case its our job to make some reasonable assumptions. This is basically a two dimensional problem so we will let the tanks be one unit thick. The side view would look the same regardless of how thick they were. Here is a diagram:



There isn't any mathenatical difference between the description of a "lawn" and the lateral view of a tank so we use our lawn procedure.

```
lawn := proc (v1,v2,v3,v4, clr )
plots[polygonplot]([v1,v2,v3,v4],color=clr,style=patch,scaling=constra
ined ) end:
```

Our time is to run from t=0 to t=19 (20 frames) so at time t in this range tank B is t/19 full while tank A is (1-t/19) full. This means that at time t between 0 and 19 the fluid level of tank B is at 10*t/19 and that at tank A is at 37-25*t/19. Thus the fluid in tank A is described by

lawn([0,0],[25,0], [25,10*t/19],[0,10*t/19], red);

and that in tank B is described by

lawn([15,12],[25,12], [25,37-25*t/19],[15,37-25*t/19], red);

A first approximation to our basic movie is:

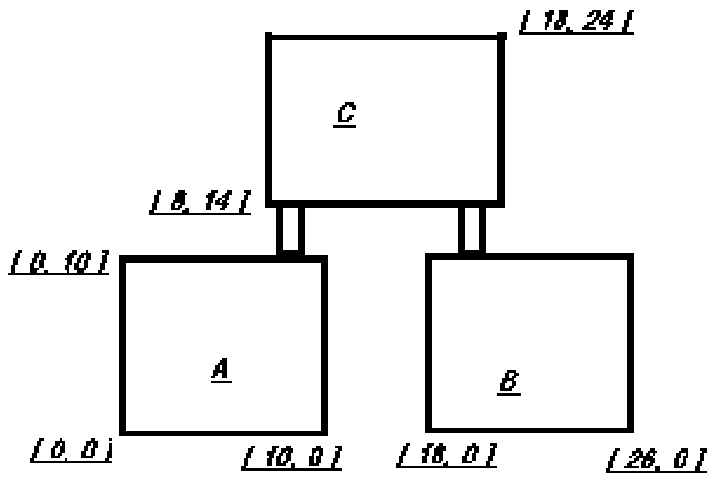
```
movie:=NULL: for t from 0 to 19 do
frame:= plots[display]([ lawn([0,0],[25,0], [25,10*t/19],[0,10*t/19],
red), lawn([15,12],[25,12], [25,37-25*t/19],[15,37-25*t/19], red)]
):
movie:=movie,frame:  od:

plots[display]([movie],insequence=true);
```

a. Modify the basic movie so that the complete tanks are there - not just the fluid.

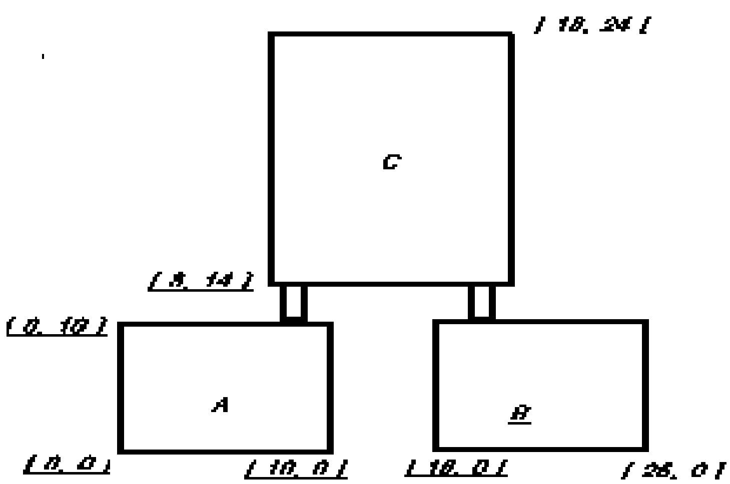
b. Modify the movie so that the color of the fluid changes as if moves from tank A to tank B.

c. Add the pipe connecting the two tanks to the movie. Have the pipe full thorugh the last (t=19) frame of the current movie then add a new last frame (t=20) in which the pipe is empty.

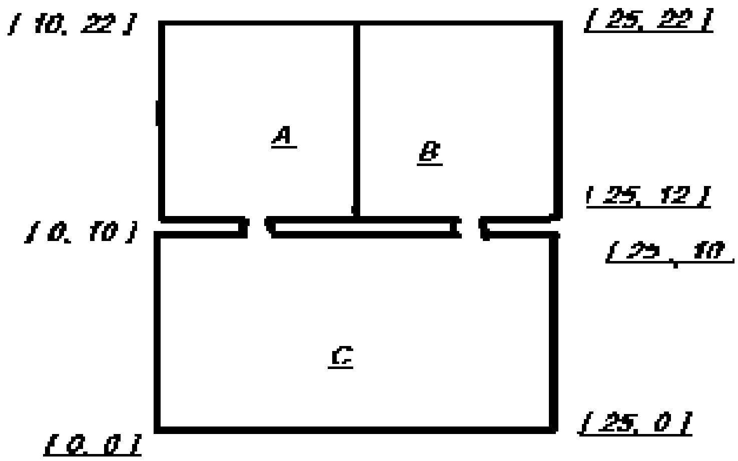
8.3.6 Exercise: *Suppose there are three tanks A,B,C arranged as in the picture. The volumes of tanks A and B are equal to that of tank C. When tank A is empty and C is full and the valve between A and C is open then tank A will fill in 20 minutes. When tank B is empty and C is full and the valve between B and C is open then tank B will fill in 35 minutes. If both tanks A and B are empty and tank full is* *empty, how long will it take for tank C to empty and how much will be in each of tanks A and B at that time?*



8.3.7 Exercise: *Suppose there are three tanks A,B,C arranged as in the picture. The volumes of tanks A and B are equal to half that of tank C. When tank A is empty and C is full and the valve between A and C is open then tank A will fill in 20 minutes. When tank B is empty and C is full and the valve between B and C is open then tank B will fill in 35 minutes. If both tanks A and B are empty and tank full is empty, how long will it take for both tanks A and B to fill up? Animate your answer.*



8.3.8 Exercise: *Suppose there are three tanks A,B,C arranged as in the picture. The volumes of tanks A and B are equal to half that of tank C. When tank A is full and tanks A and C are empty all of the fluid in tank A will flow into tank C in 20 minutes. When tank B is full and tanks A and C are empty all of the fluid in tank B will flow into tank C in 35 minutes. If both tanks A and B are full and tank C is empty, how long will it take for tanks A and B to fill tank C? Animate your answer.*



8.3.9 Exercise: *Modify the lawnmower code to accept the dimensions of the lawn and the width if the mower rather than the number of strips. Have it then calculate the number of strips and have it deal rationally with what happens if the number of strips is not a whole number. This should be a reasonable thing for a real person to do when mowing such a lawn.*

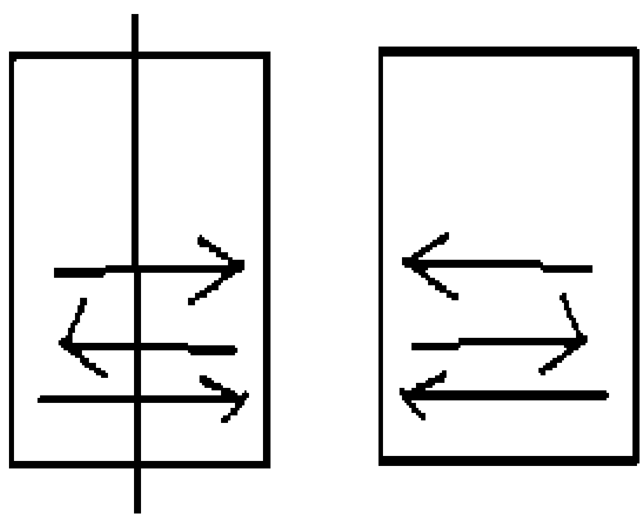
8.3.10 Exercise: *Suppose Larry, Curley, and Moe are neighbors with identical 10 by 10 square lawns. It takes Larry 20 minutes to mow his lawn, Curley takes 15, and Moe 10. Make a movie of all three mowing their lawns, starting at the same time.*

9 Mowing Yards (part 2)

9.1 Starting at Different Corners: Going in Different Directions

Up until now all of our mowing starts in the lower left corner of a square or rectangular lawn. Maybe we would like to start in the upper left and move in either direction, or even in the upper right. One way to do this would be to re-do all we have done before, only making everything start at some other point. That would involve repeating a considerable amount of effort when we know intuitively that these problems and their solutions are all essentially equivalent. "Lower left corner" obviously has no intrinsic meaning to a square. By rotating it any point could be "lower" and by turning it over or looking at its reflection in a mirror the notions of "left" and "right" are reversed. By geometrically (visually) applying these "symmetries" of the square we realize that the analysis of starting at any particular corner and heading in either possible direction can be described in terms of what happens in the case we have already done. What we need to do is understand how to quantify the symmetries of the problem mathematically and apply the corresponding mathematical symmetries to the work we have already done.

For instance the diagram below shows what we are currently doing: starting at the lower left abd proceeding to the right. We could start at the lower right and proceed to the left simply by reflecting everything through a line perpendicular to the x-axis which bisects the yard. Suppose that we have arranged things so that the bisecting lineis in fact the y-axis



Then the analytic (algebraic) realization of this reflection is simply reversing the sign of the x-coordinates. That is if P is the point [x,y] then the point which is the reflection of P through the y-axis is simply [-x,y]. What this means is that we can convert the algebra of mowing from left to right to that of mowing from right to left simply by changing some signs. This promises to be far more simple that re-doing all of our previous work.

Of course in general the y-axis isn't a line of symmetry of a problem but our powerful idea of symmetry tells us how to handle this: if there is a line of symmetry then since we can intuitively see that that all lines in the plane are "equivalent" we can take the alebra which expersses the equivalence of the given line and the y-axis and use it to convert the analysis of the simple case into that of the one which interests us. For example, if the line of symmetry isn't the y-axis but is parallel to it we can simply apply a shift which moves the line of symmetry to the y-axis, then refelect through the y-axis, then shift back. If the shift which moves our line to the y-axis is [SHIFTX, SHIFTY] then the recipe would simply be:

[X, Y] -> [X + SHIFTX, Y + SHIFTY] -> [-X - SHIFTX, Y + SHIFTY] -> [-X -2* SHIFTX, Y]

Applying this transformations to the vertices of a polygon (lawn) would then reflect that polygon through the given line. For instance if Larry's lawn is

Larryslawn:=[[0,0],[3,0],[3,1], [0,1]];

and we want to refelect Larry's lawn through its centerline, x=1.5 we would use a shift of [-1.5,0]. Then

```
SHIFTX:=-1.5; SHIFTY:=0;
reflectedLarryslawn:=[[-0-2*(SHIFTX),0],[-3-2*SHIFTX,0],[-3-2*SHIFTX,
1], [-0-2*SHIFTX,1]];
```

Now if we view both Larryslawn and reflected Larryslawn, one red and one blue we see:

```
LL:=plots[polygonplot](Larryslawn,
color=blue);
RLL:=plots[polygonplot](reflectedLarryslawn, color=red);
plots[display]([LL,RLL],scaling=constrained);
```

We are missing red one. Actually not, lets reverse the order

```
plots[display]([
RLL,LL],scaling=constrained);
```

The point is that the reflection transformed Larry's lawn **onto itself**, reversing left and right. The red and blue lawns occupy the same space.

Now we can use this idea to "reflect" our previous mowing. The primary idea, which we have actually used before, is that our lawnmower code employs the coordinates **only through the** `lawn function. Thus all we will` need to do is modify lawn to accomplish reflection. Lets first write a version of lawn that reflects everything through the line x=3 to check that we know how to do it.

```
lawnreflxeq3:= proc (v1,v2,v3,v4, clr )
local w1,w2,w3,w4,shift;
shift:=[-3,0]:
w1:=[-v1[1]-2*shift[1],v1[2]]:
w2:=[-v2[1]-2*shift[1],v2[2]]:
w3:=[-v3[1]-2*shift[1],v3[2]]:
w4:=[-v4[1]-2*shift[1],v4[2]]:
plots[polygonplot]([w1,w2,w3,w4],color=clr,style=patch,scaling=constra
ined ):
end:

lawnreflxeq3([3,0],[7,0],[7,2],[3,2],
```

```
red);
```

We modify lawnmower3 simply by changing lawnmowerwithshift to lawnreflxeq3.

```
lawnmower4:=proc(Lngth, Wdth, MOWINGTIME,
MOVIELENGTH, NUMSTRIPS, SHIFT )
local movie, YARD, MOWEDYARD, MRWDTH, TIME, WHICHSTRIP, FRACSTRIP,
MOWN, FRAME, LASTFRAME ;
movie:=NULL:
YARD:=lawnreflxeq3([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine,SHIFT):
MOWEDYARD:=lawnreflxeq3([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
green,SHIFT):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to MOVIELENGTH-1 do
if TIME <=MOWINGTIME then
WHICHSTRIP:= floor(NUMSTRIPS*TIME/MOWINGTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/MOWINGTIME) - WHICHSTRIP;
MOWN:=lawnreflxeq3(
[0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],[0,MRWDTH*WHICHSTRIP]
,green,SHIFT):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawnreflxeq3( [0,MRWDTH*(WHICHSTRIP )]
, [Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )
],[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1) ],[0,MRWDTH*(WHICHSTRIP+1)],
green,SHIFT ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawnreflxeq3([Lngth *(
1-FRACSTRIP),MRWDTH*(WHICHSTRIP )], [Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)], [Lngth *( 1-
FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] , green,SHIFT ) ],YARD):
fi:
movie:=movie,FRAME:
else
movie:=movie,MOWEDYARD:
fi:
od:
if MOVIELENGTH >= MOWINGTIME then LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:plots[display]([movie],insequence=true):
fi:
[movie];
end;

MOVIE:=lawnmower4(30,20,25,20,10,[0,0] ):

plots[display]( MOVIE,insequence=true);
```

Thus we see that by altering the "lawn" component of our lawnmower program we can produce lawnmower starting at other corners and moving in other directions. Now to make things a little simpler we will concentrate on a single square, 10 by 10 lawn which is symmetric about the origin and return to the original problem. We want to make a movie of Sam and Bill mowing the **same** lawn.

9.2 Sam and Bill Mowing the Same Lawn

Our lawn will be:

```
SQUAREYARD:=[[-5,-5],[5,-5],[5,5],[-5,5]]:
```

We make versions of lawn which will reflects through the x-axis or the y-axis

```
SQUAREYARD:=[[-5,-5],[5,-5],[5,5],[-5,5]]:
lawnx:= proc (v1,v2,v3,v4, clr,shift ) local w1,w2,w3,w4 ;
w1:=v1+shift:
w2:=v2+shift:
w3:=v3+shift:
w4:=v4+shift:
w1:=[-w1[1] ,w1[2]]:
w2:=[-w2[1],w2[2]]:
w3:=[-w3[1] ,w3[2]]:
w4:=[-w4[1] ,w4[2]]:
plots[polygonplot]([w1,w2,w3,w4],color=clr,style=patch,
scaling=constrained ):
end:

lawny:= proc (v1,v2,v3,v4, clr,shift ) local
w1,w2,w3,w4 ;
w1:=v1+shift:
w2:=v2+shift:
w3:=v3+shift:
w4:=v4+shift:
w1:=[ w1[1] ,-w1[2]]:
w2:=[ w2[1] ,-w2[2]]:
w3:=[ w3[1] ,-w3[2]]:
w4:=[ w4[1] ,-w4[2]]:
plots[polygonplot]([w1,w2,w3,w4],color=clr,style=patch,
scaling=constrained ):
end:
```

Now we can make mower who starts in different corners and proceeds in different directions simply by changing the version of "lawn" as we did in lawnmower4. Lets make a mower who starts at the lower left and mows SQUAREYARD. All we do is grab a copy of lawnmower4 and change the "lawn" function to "lawnx"

```
lawnmower4x:=proc(Lngth, Wdth, MOWINGTIME,
MOVIELENGTH, NUMSTRIPS, SHIFT )
local movie, YARD, MOWEDYARD, MRWDTH, TIME, WHICHSTRIP, FRACSTRIP,
MOWN, FRAME, LASTFRAME ;
movie:=NULL:
YARD:=lawnx([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine,SHIFT):
MOWEDYARD:=lawnx([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
green,SHIFT):
```

```
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to MOVIELENGTH-1 do
if TIME <=MOWINGTIME then
WHICHSTRIP:= floor(NUMSTRIPS*TIME/MOWINGTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/MOWINGTIME) - WHICHSTRIP;
MOWN:=lawnx(
[0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],[0,MRWDTH*WHICHSTRIP]
,green,SHIFT):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawnx( [0,MRWDTH*(WHICHSTRIP )] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )
],[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1) ],[0,MRWDTH*(WHICHSTRIP+1)],
green,SHIFT ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawnx([Lngth *(
1-FRACSTRIP),MRWDTH*(WHICHSTRIP )], [Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)], [Lngth *( 1-
FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] , green,SHIFT ) ],YARD):
fi:
movie:=movie,FRAME:
else
movie:=movie,MOWEDYARD:
fi:
od:
if MOVIELENGTH >= MOWINGTIME then LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:plots[display]([movie],insequence=true):fi:
[movie];
end;
```

Now **lawn mower lower right,** `LMLR` is

```
LMLR:=proc(MOWINGTIME, MOVIELENGTH,
NUMSTRIPS)
lawnmower4x (10, 10, MOWINGTIME, MOVIELENGTH, NUMSTRIPS, [-5,-5]
)end;
joe:=LMLR(20,20,9):

plots[display](joe,insequence=true);
```

A left upper mower, `LMUL would use` lawny . Just make a lawnmower4y

```
lawnmower4y:=proc(Lngth, Wdth, MOWINGTIME,
MOVIELENGTH, NUMSTRIPS, SHIFT )
local movie, YARD, MOWEDYARD, MRWDTH, TIME, WHICHSTRIP, FRACSTRIP,
MOWN, FRAME, LASTFRAME ;
movie:=NULL:
YARD:=lawny([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine,SHIFT):
MOWEDYARD:=lawny([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
green,SHIFT):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to MOVIELENGTH-1 do
```

```
if TIME <=MOWINGTIME then
WHICHSTRIP:= floor(NUMSTRIPS*TIME/MOWINGTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/MOWINGTIME) - WHICHSTRIP;
MOWN:=lawny(
[0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],[0,MRWDTH*WHICHSTRIP]
,green,SHIFT):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawny( [0,MRWDTH*(WHICHSTRIP )] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )
],[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1) ],[0,MRWDTH*(WHICHSTRIP+1)],
green,SHIFT ) ],YARD):
else
FRAME:=plots[display]([MOWN, lawny([Lngth *(
1-FRACSTRIP),MRWDTH*(WHICHSTRIP )], [Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)], [Lngth *( 1-
FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] , green,SHIFT ) ],YARD):
fi:
movie:=movie,FRAME:
else
movie:=movie,MOWEDYARD:
fi:
od:
if MOVIELENGTH >= MOWINGTIME then LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:plots[display]([movie],insequence=true):
fi:
[movie];
end;
```

Now use it to define **lawn mower upper left**, `LMUL` . `Note the similarity to` LMLR

```
LMUL:=proc(MOWINGTIME, MOVIELENGTH,
NUMSTRIPS)
lawnmower4y (10, 10, MOWINGTIME, MOVIELENGTH, NUMSTRIPS, [-5,-5]
)end;
```

```
joe:=LMUL(20,20,9)
```

```
plots[display](joe,insequence=true);
```

Our original `lawnwithshift` gives us **lawn mower lower left** LMLL but thats just lawnmower3

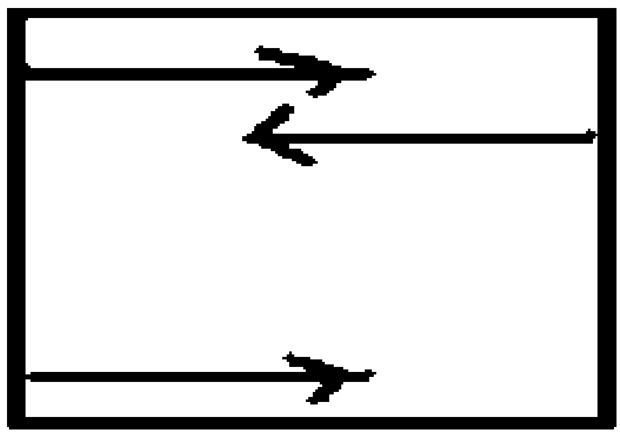
```
LMLL:=proc(MOWINGTIME, MOVIELENGTH,
NUMSTRIPS)
lawnmower3 (10, 10, MOWINGTIME, MOVIELENGTH, NUMSTRIPS, [-5,-5]
)end;
```

```
joe:=LMLL(20,20,9):
```

```
plots[display](joe,insequence=true);
```

Now we are getting close to having Sam and Joe cooperate on mowing the lawn. Knowing how long each takes to mow the lawn

separately we can calculate how long they should mow together. We can have one of them start at, say, the lower left and the other at the upper left and have them each mow for the computed time.



For instance suppose Sam mows the lawn in 30 minutes and Bill mows it in 40. Then if together they mow in TOGETHERTIME we have the equation eq:

```
eq:=1/TOGETHERTIME = 1/30 + 1/40;
ans:=solve(eq,TOGETHERTIME);
ans:=evalf(ans)
```

This says that if we, for instance make a 17 frame movie of Bill mowing from the top and Sam mowing from the bottom then we should be able to put them together and make a movie of both mowing most of the lawn together together. Lets make the two separate ones first.

```
BILLMOWING:=LMLL(40,17,7):

SAMMOWING:=LMUL(30,17,7):

plots[display](BILLMOWING,insequence=true);

plots[display](SAMMOWING,insequence=true);
```

Recall that we know how to run two movies at simultaneously. This should also allow us to overlay one upon another. (Note this isn't going to work exactly).

```
movie:=seq(plots[display]([SAMMOWING[i],BILLM
OWING[i]]),
i=1..nops(SAMMOWING))

plots[display]([movie],insequence=true)
```

There are two problems. One is that the movie only has our mowers mowing 17 time units while it takes a little more than that to finish the job. The other is that we have one movie playing on top of the other SAMMOWING is on top of BILLMOWING. This would be ok except that each frame of SAMMOWING also puts down the unnown lawn as a base. Thus we need a slightly modified version of SAMMOWING in which the base isn't put down. We just need to go back and modify lawnmower4y to lawnmower4y without background. The reader should compare the following to lawnmower4y. All we have done is delete the background (YARD) from each frame to get "lawnmower4 without background". Note that the deletions are accomplished by "commenting" out the definition of "YARD" and references to "YARD" in "lawnmower4y".

```
lm4ywobg:=proc(Lngth, Wdth, MOWINGTIME,
MOVIELENGTH, NUMSTRIPS, SHIFT )
local movie, YARD, MOWEDYARD, MRWDTH, TIME, WHICHSTRIP, FRACSTRIP,
MOWN, FRAME, LASTFRAME ;
movie:=NULL:
#YARD:=lawnreflxeq3([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
aquamarine,SHIFT):
MOWEDYARD:=lawny([0,0],[Lngth , 0],[Lngth ,Wdth ],[ 0,Wdth ],
green,SHIFT):
MRWDTH:=Wdth/NUMSTRIPS:
for TIME from 0 to MOVIELENGTH-1 do
if TIME <=MOWINGTIME then
WHICHSTRIP:= floor(NUMSTRIPS*TIME/MOWINGTIME);
FRACSTRIP:=TIME*(NUMSTRIPS/MOWINGTIME) - WHICHSTRIP;
MOWN:=lawny(
[0,0],[Lngth,0],[Lngth,MRWDTH*WHICHSTRIP],[0,MRWDTH*WHICHSTRIP]
,green,SHIFT):
if WHICHSTRIP mod 2 = 0 then
FRAME:=plots[display]([MOWN, lawny( [0,MRWDTH*(WHICHSTRIP )] ,
[Lngth *FRACSTRIP,MRWDTH*(WHICHSTRIP )
],[Lngth*FRACSTRIP,MRWDTH*(WHICHSTRIP+1) ],[0,MRWDTH*(WHICHSTRIP+1)],
green,SHIFT ) ] # ,YARD
):
else
FRAME:=plots[display]([MOWN, lawny([Lngth *(
1-FRACSTRIP),MRWDTH*(WHICHSTRIP )], [Lngth,MRWDTH*(WHICHSTRIP )],
[Lngth,MRWDTH*(WHICHSTRIP+1)], [Lngth *( 1-
FRACSTRIP),MRWDTH*(WHICHSTRIP+1) ] , green,SHIFT ) ] #,YARD
):
fi:
movie:=movie,FRAME:
else
movie:=movie,MOWEDYARD:
fi:
od:
if MOVIELENGTH >= MOWINGTIME then LASTFRAME:=MOWEDYARD:
movie:=movie,LASTFRAME:plots[display]([movie],insequence=true):
fi:
[movie];
end;
```

Now we can make the associated "lawn mower upper left without background"

```
LMULwobg:= proc(MOWINGTIME, MOVIELENGTH,
NUMSTRIPS)
lm4ywobg (10, 10, MOWINGTIME, MOVIELENGTH, NUMSTRIPS, [-5,-5]
)end;

SAMMOWINGwobg:=LMULwobg(30,17,7):

movie:=seq(plots[display]([SAMMOWINGwobg[i],
BILLMOWING[i]]),i=1..nops(SAMMOWING)):
plots[display]([movie],insequence=true);
```

Now the obvious way to finish is to have the last frame be the completely mown yard. Thats of course easy to lay in. We called it SQUAREYARD.

```
MOWNSQUAREYARD:=plots[polygonplot](SQUAREYARD
,
color=green, scaling=constrained):
```

Now using the previously defined "movie"

```
movie:=movie,MOWNSQUAREYARD:
plots[display]([movie],insequence=true);
```

This isn't completely satisfactory, however because we don't get to see the two lawnmowers come together at the point where each has exactly done his part. There are other problems, too. Depending on the number of strips it might be necessary for Sam and Bill to

start at diagonally opposite corners. For instance, suppose our current problem used 3 strips rather than 7.

```
BILLMOWING:=LMLL(40,17,3):
SAMMOWINGwobg:=LMULwobg(30,17,3):
movie:=seq(plots[display]([SAMMOWINGwobg[i],
BILLMOWING[i]]),i=1..nops(SAMMOWINGwobg)):
plots[display]([movie],insequence=true);
```

What has happened here is that the faster mower got to the middle strip and started mowing it ahead of the slower fellow. In order to keep this from happening they should have started at antipotal corners. In other word we need a version of "lawn" which starts at the upper right corner. We need an "LMURwobg" word to handle this situation. Thats easy enough, if we reflect through both the x and y axes with a "lawnxy" we can easily make one.

9.2.1 Exercise: *Make a "lawnxy" word and use it to create a mower who starts in the upper right corner and mows toward the y-axis. Then use this to make a movie of Sam and Bill mowing a three-strip lawn in which they don't try to mow the same grass.*

9.2.2 Mathematics Problem: *Sam and Bill have a square lawn which is 100 by 100 meters. They have two lawn mowers each of which cuts a path 1/2 meter wide. Using one of the mowers Sam can mow the lawn in 2 hours and Bill can mow it in 3. Using the algebra they are studying in school they conclude that using both mowers they should be able to mow the lawn in an hour and 12 minutes.*

Where does each start and how does he proceed so that neither ever passes his mower over grass that the other has already mown?

9.2.3 Problem Variation:(*Emptying The Swimming Poo***l)** *The "mowing problem" is really about adding rates of things that happen. A simpler problem to visualize might be that of emptying a swimming pool. Suppose there are two drains. One can empty the pool in 24 hours, the other in 36. Make a two dimensional animation of the pool emptying through each drain and the two drains together. Make a movie which shows the three cases side by side.*

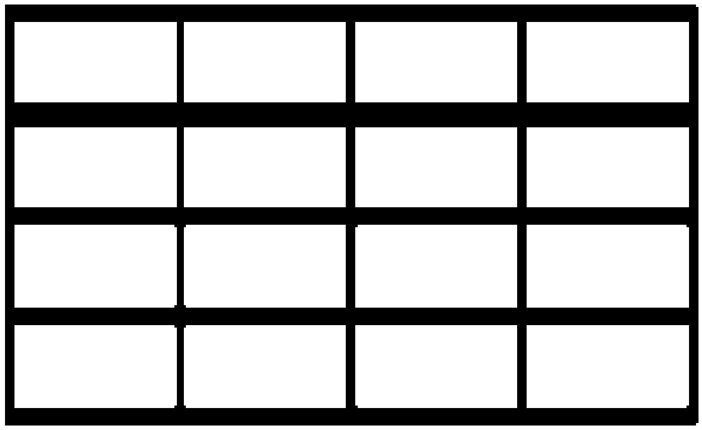
9.2.4 *Start: For the moment assume the pool is constant depth. There isn't much difference between a lawn and a (vertical section of a pool). The "empty" pool could be the "unmown" yard and the "mown" grass the water in the pool (or visa versa). Changing colors to grey and blue would promote the image. The draining pool would not involve any partial strips so most of the difficulty of the mowing problem would not be present.*

9.3 EXERCISES

The following exercises emphasize the idea of (when possible) isolating the coordinate-dependent components in a small number of Maple words and then modifying only those to take advantage of symmetry in a problem. In this case we will use a simplified version of CPB (chromatic brick placer) which we previously developed. This is a two dimensional version which places a horzontal brick of a specified color with its lower left corner at a specified point.

```
CBP2d:= proc ( pt ,clr )
local v1, v2, v3, v4, brick;
v1 := [pt[1],pt[2]];
v2 := [pt[1]+8, pt[2] ]; v3 := [pt[1]+8, pt[2]+4 ];
v4 := [pt[1], pt[2] +4]; brick := [ v1, v2, v3, v4];
plots[polygonplot](brick,color=clr,style=patch,
thickness=3,scaling=constrained ) end;

CBP2d([0,3],red);
```

Now we develop a word which lays down a small brick walls (say four bricks wide by five bricks high) which have the following form:



We make a "small brick wall" procedure called SBW

```
SBW:=proc(wide,high) local movie,frame,i,j;
movie:=NULL; frame:=NULL;
for i from 0 to wide-1 do
for j from 0 to high-1 do
frame:=plots[display](frame,CBP2d([8*i,4*j],red)):
movie:=movie,frame:
od:
od:
plots[display]([movie],insequence=true);
end;
```

```
SBW(4,5);
```

Note that the only part of `SBW` which uses the coordinates is PBC2d. Thus by modifiying the way PBC2d uses coordinates we can change the manner in which the bricks are set down.

9.3.1 Exercise: *Modify CPB2d so that SBW lays its bricks starting at the left.*

HINT: The wall is symmetric about the line x=(8*wide)/2 (at least the set of lower left vertices is). As noted at the beginning of the chapter the algebraic transformation which reflects the plane about the line x=wide/2 is

`[ X, Y ] -> [ X -8*wide/2, Y ]` -> [-X +8*wide/2, Y] -> [-X +2* 8*wide/2, Y]

Thus if PBC2d originally places a brick at [x,y] we now want it to place it at [-x+8*wide,y].

This means that if we change only one line in SBW we can produce a newSBW which does the exercise. One possibility for the changed line is:

```
frame:=plots[display](frame,CBP2d([-8*i+8*wide,4*j],red)):
```

```
newSBW:=proc(wide,high) local
movie,frame,i,j;
movie:=NULL; frame:=NULL;
for i from 0 to wide-1 do
for j from 0 to high-1 do
frame:=plots[display](frame,CBP2d([-8*i+8*wide,4*j],red)):
movie:=movie,frame:
od:
od:
plots[display]([movie],insequence=true);
end;

newSBW(5,4);
```

9.3.2 *EXERCISE:* `SBW(a,b)` *builds starts at the lower left, builds a stack "b" bricks high, then moves a step to the right and builds the next stack of "b" bricks, doing this "a" times. Modify SBW to produce SBWh (h for horizontal) which first lays a layer of "b" bricks, then starts over at the left and lays a second layer on top of this one, etc.*

HINT: This can be done by permuting two lines in SBW

9.3.3 *EXERCISE: Modify* `SBW to get SBWul ( "ul"` *for " upper left") so that SBWul(a,b) starts on the left and builds a stack of bricks of "b" bricks from the top down .*

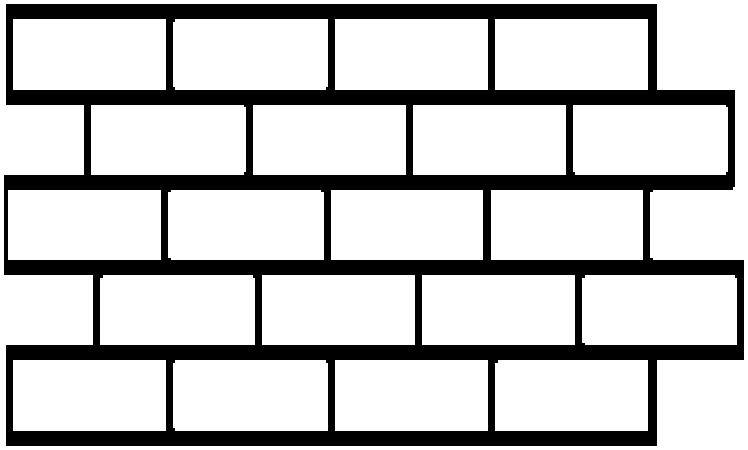
HINT: This "stacking" (it might be a brick sidewalk) is the reflection of the action of SBL through the line y=4*b/2. The algebraic realization of this symmetry is [x,y]->[x,-y+2*(4*b/2)] = [x, -y+4*b].

9.3.4 *EXERCISE: Permute two lines in SBW to get SBWulh ("ulh" for " upper left, horizontal ") so that SBWul(a,b) starts on the upper left and builds "a" descending rows of "b" bricks.*

9.3.5 *EXERCISE: Modify* `SBW to get SBWbf` *(for "back and forth") so that SBWbf(a,b) starts laying bricks at the lower left, lays horizontal rows of "b" bricks, each time reversing direction to lay the next row.*

HINT: This switches between the (horizontal versions of) the actions of SBW and newSBW, depending on the parity of the row to be completed.

9.3.6 *EXERCISE: Make SBWs, a version of SBW which lays bricks in the familiar "shifted" pattern, one horizontal row at a time.*



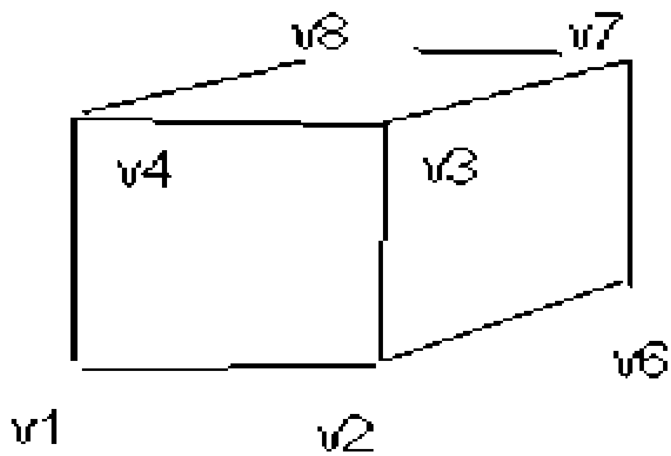
10 Shadows: Problems from Nature

How do you draw the shadows that you see on a sunny day? What happens to a shadow as the day wears on? What happens to a shadow as the object it is a shadow of moves?

You could answer the second and third questions without having an answer to the first, of course. You would just play a movie in your mind of the sun moving in an arc and the shadows lengthening. Or you could imagine the shadow of a person walking under a streetlight at night. But after getting an answer to the first question in a Maple worksheet, we can model the two motions of a shadow (one when the light source is moving, the other when the object is moving) pretty easily, as we shall see.

Analysis.

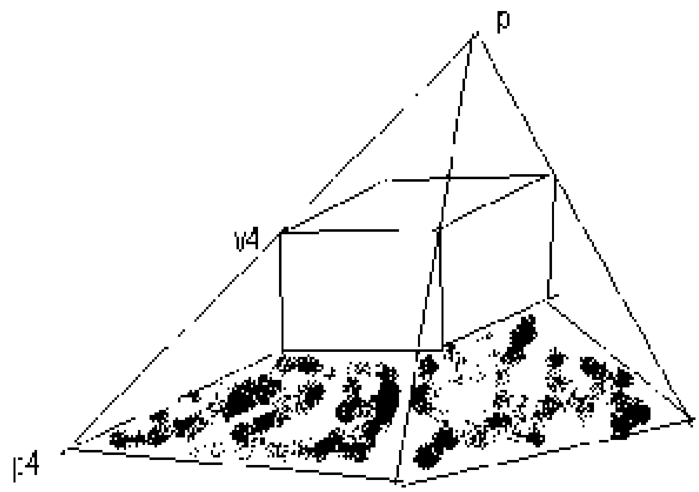
Shadows are very complex things in nature. You can see this by watching them. They don't always behave as you would predict. For the sake of simplicity, assume that we are drawing the shadow cast by a box sitting on the ground.



So, our box will be the unit cube. We can give names to the vertices and faces, as before.

```
v1:=[1,0,0]:  v2:=[1,1,0]:  v3:=[1,1,1]:
v4:=[1,0,1]:  v5:=[0,0,0]:  v6:=[0,1,0]:  v7:=[0,1,1]:  v8:=[0,0,1]:

front := [v1,v2,v3,v4]:  back :=
[v5,v6,v7,v8]:
left := [v1,v5,v8,v4]:  right := [v2,v6,v7,v3]:
top := [v4,v3,v7,v8]:  bottom := [v1,v2,v6,v5]:

box :=
plots[polygonplot3d]([front,back,left,right,top,bottom],style=patch,co
lor=tan):
```

Also, we will assume that the light is coming from a single point and the shadow is being cast onto the xy-plane $z = 0$.



Clearly, we need to find the shadows p1, p2, of the corners v1, v2, ... of the box on the xy plane. From the picture, we can see that $p4 = p + s\,(v4 - p)$ for some number s. In the z-coordinate (that is the third coordinate), we know the left hand side of the equation is 0, and the right hand side is $p_3 + s\,(v4_3 - p_3)$, so we can solve for t to get $s = \frac{p_3}{p_3 - v4_3}$. This works for any point *vi*, as long as p is above the top of the box. Let's try it on one vertex first. Take some arbitrary point p with p_3 greater than 1 (why this restriction?).

```
p :=[10,7,3];
```

$$p := [10, 7, 3]$$

```
s := p[3]/(p[3]-v4[3]);
```

$$s := \frac{3}{2}$$

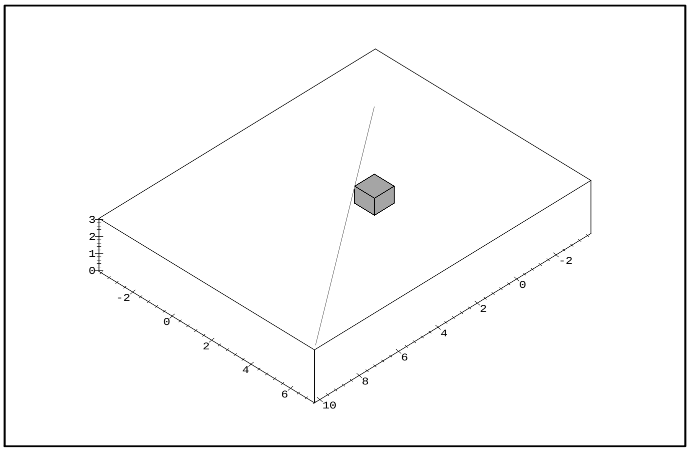
```
p4 := p+s*(v4-p);
```

$$p4 := [\frac{-7}{2}, \frac{-7}{2}, 0]$$

So the corner $v4 = [1, 0, 1]$ projects onto the point $p4 = [-3.5, -3.5, 0]$. Draw the line segment connecting the light source to the shadow point. It will pass through the vertex v4.

```
ray := plots[polygonplot3d]([p,p4] ):
```

```
plots[display]([ray,box],scaling=constrained,
axes=boxed);
```



Actually what we want to do is draw the projection of each face of the box onto the xy-plane. So let's define a word to do just that. Let's start with a particular face, say the front face.

```
face := front;
```

$$face := [[1, 0, 0], [1, 1, 0], [1, 1, 1], [1, 0, 1]]$$

```
shad := NULL;
```

```
for pt in face do
t := p[3]/(p[3]-pt[3]);
newpt := p + t*(pt-p);
shad := shad,newpt;
od:
```

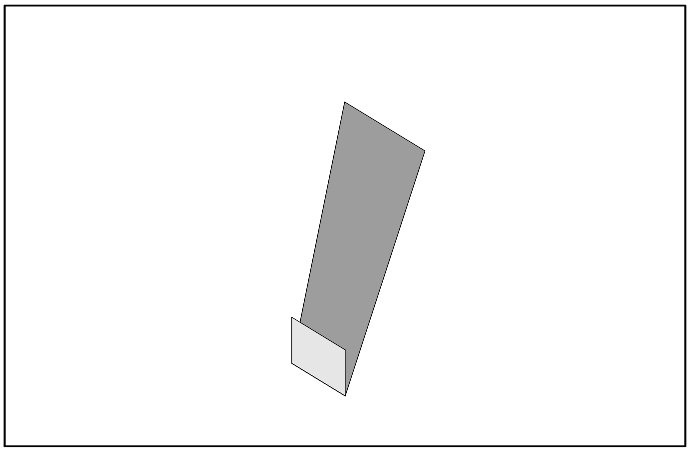
$$shad :=$$

```
shad;
```

$$[1, 0, 0], [1, 1, 0], [\frac{-7}{2}, -2, 0], [\frac{-7}{2}, \frac{-7}{2}, 0]$$

Now to see the shadow, use plots[polygonplot3d].

```
plots[polygonplot3d]([[shad],front],style=pat
ch,scaling=constrained);
```



Notice the bottom two corners of the front project onto themselves, since they are on the xy-plane already. The word shadow defined below takes two inputs: a light source p and a polygon face=[v1,v2,v3, ..., vn] and outputs the polygon shad=[p1, p2,...,pn] which is the shadow of face on the xy-plane cast by the light source at p.

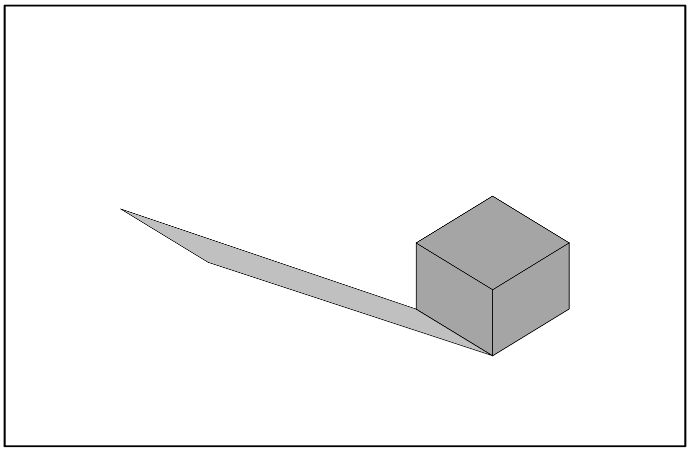
```
shadow := proc(p,face)
local shad, pt, t, newpt;
shad := NULL;
for pt in face do
t := p[3]/(p[3]-pt[3]);
newpt := p + t*(pt-p);
shad := shad,newpt;
od;
plots[polygonplot3d]([shad],style=patch, scaling
=constrained,color=grey);
end:
```

Test this word out by looking at the shadow of the front of the box

```
q := [-5,21,8];
```

$$q := [-5, 21, 8]$$

```
plots[display]([box,shadow(q,front)]);
```

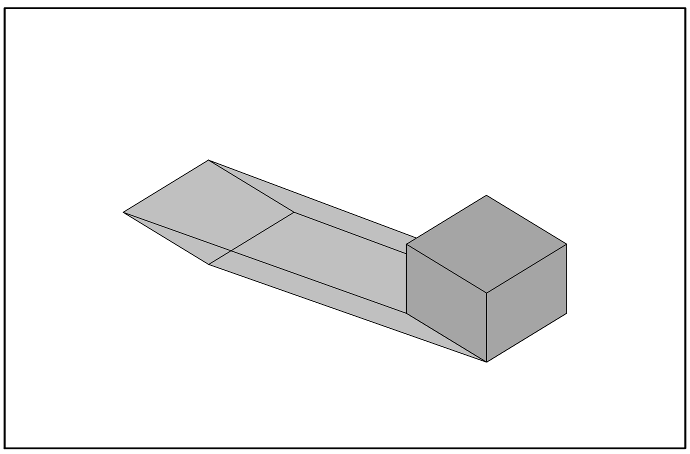


It seems to do what we want. Now put together the shadow of each face of the box (except the bottom).

```
shadowbox := p ->
plots[display]([box,shadow(p,front),
shadow(p,top),shadow(p,left),shadow(p,right), shadow(p,back)] );
```

$$shadowbox := p \to plots_{display}([\\ box, \mathrm{shadow}(p, front), \mathrm{shadow}(p, top), \mathrm{shadow}(p, left), \mathrm{shadow}(p, right), \mathrm{shadow}(p, back)\\ ])$$

```
shadowbox(q+q);
```



10.1 Problems:

10.1.1 *1. Find the area of the shadow shown above, not counting the base of the box. Hint: Redraw with axes = boxed and notice the shadow 'sits' in a rectangle.*

10.1.2 *2. Modify the definition for shadowbox so that box could be any collection of faces, not just the faces of the box. Use the newly defined word to draw the shadow of a box that is raised off of the xy-plane.*

10.1.3 *3. Shadows cast by the light of the sun or some far off light source are traced out by parallel light rays, rather than rays coming from a point p. Develop a word pshadow which draws such a shadow and use it to draw a shadowbox.*

Solution to 3.

```
pshadow := proc(dir,face)
local shad, pt, t, newpt;
shad := NULL;
for pt in face do
t := pt[3]/(-dir[3]);
newpt := pt + t*dir;
shad := shad,newpt;
od;
plots[polygonplot3d]([shad],style=patch, scaling
=constrained,color=grey);
end:
```

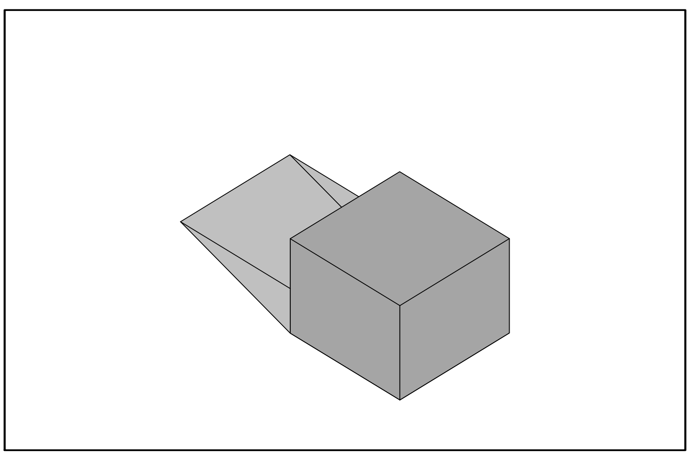
```
pshadowbox := p ->
plots[display]([box,pshadow(p,front),
pshadow(p,top),pshadow(p,left),pshadow(p,right),
pshadow(p,back)] );
```

$$pshadowbox := p \to plots_{display}([box, \mathrm{pshadow}(p, front), \mathrm{pshadow}(p, top), \mathrm{pshadow}(p, left), \mathrm{pshadow}(p, right), \mathrm{pshadow}(p, back)])$$

```
dir := [1,1,3];
```

$$dir := [1, 1, 3]$$

```
pshadowbox([1,4,3]);
```



10.2 Moving shadows

Now to make the shadows move, all we need to do is move the light source, taking pictures as it moves. While we're at it, we can provide a tabletop for the shadow to move on.

```
tabletop :=
plots[polygonplot3d]([[-2,-4,-.01],[2,-4,-.01],
[2,2,-.01],[-2,2,-.01]],color=magenta,style=patch);

movie := [seq(plots[display]([tabletop,
shadowbox(q+[ i,i/2,0])]),i=0..20)]:

plots[display](movie,insequence=true);
```

10.2.1 Moving Shadow Problem *Make a movie of the shadow of the box as it moves from its position at the origin to the point [0,0,3] above the xy-plane. Keep the light source fixed at q=[-5,23,8]. Use your definition of shadowbox from the last problem.*

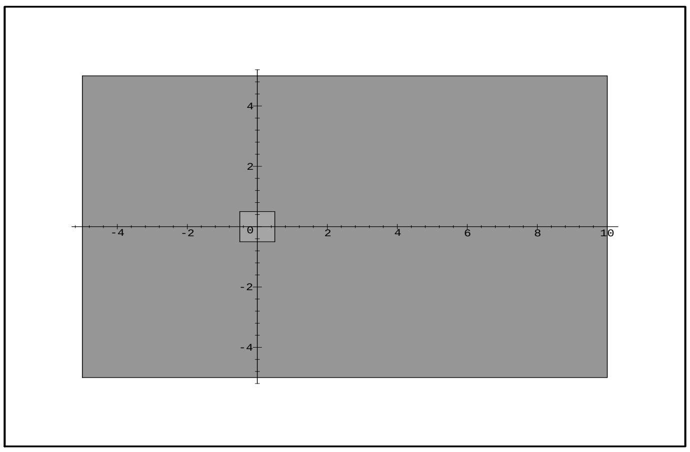
11 More Exercises

11.0.2 *Exercise: Modify the vertices of "box" below to make a "lightpole" with the unit square as base and 3 units high. This can be done by modifying the following instructions.*

```
v1:=[.5,0,0]:  v2:=[.5,.5,0]:  v3:=[.5,.5,3]:
v4:=[.5,0,3]:
v5:=[0,0,0]:  v6:=[0,.5,0]:  v7:=[0,.5,3]:  v8:=[0,0,3]:
front := [v1,v2,v3,v4]:  back := [v5,v6,v7,v8]:
left := [v1,v5,v8,v4]:  right := [v2,v6,v7,v3]:
top := [v4,v3,v7,v8]:  bottom := [v1,v2,v6,v5]:
pole :=
plots[polygonplot3d]([front,back,left,right,top,bottom],style=patch,co
lor=tan,scaling=constrained):
```

11.0.3 *EXERCISE: Use* `plottool[sphere]` *to place a light on top of the pole in the form of a sphere one unit in diameter.*

11.0.4 *EXERCISE: Place the lightpole on a rectangular area according to the diagram.*



11.1 DIAGRAM CODE

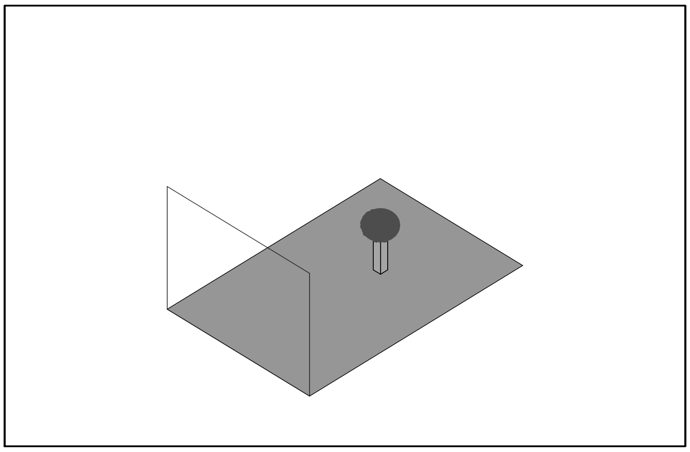
```
base:=plots[polygonplot]([ [-.5,-.5], [
.5,-.5], [ .5,.5], [-.5,.5]], style=patch,color=tan,
scaling=constrained):  ground:=plots[polygonplot]([[ -5,-5 ],[ 10,-5
],[ 10, 5 ],[ -5, 5 ]],color=green, style=patch ):
plots[display]([base, ground],scaling=constrained);
```

11.2 GROUND PLACEMENT

```
light:=plottools[sphere]( [.5,.5,4], 1,
color=red,style=patchnogrid):
ground:=plots[polygonplot3d]([[ -5,-5,0],[ 10,-5,0],[ 10, 5,0],[ -5,
```

```
5,0]],color=green, style=patch ):
```

11.2.1 *Exercise: Errect a wall 10 units from the light pole which is ten units high, as indicated in the diagram*



11.3 Wall Placement

```
wall:=plots[polygonplot3d]([[ 10, -5, 0],[
10,-5,10],[ 10, 5,10],[ 10, 5,0]],color=blue,
style=wireframe,thickness=2):


plots[display]([light,ground,pole,wall ]);



plots[display]([light,ground,pole,wall,stickp
erson]);
```

11.3.1 *Exercise: Place a "stickperson" 5.5 units high midway between the light and the wall. The stickperson is assumed to be two dimensional and to lie entirely in a plane parallel to the wall and perpendicular to the ground. The diagram below provides a sketch of "stickperson". From it one can derive coordinates for the face, body, legs, hands, and arms.*

11.4 PARTIAL STICKPERSON

Make individual polygonplot3d plots for stickperson's various components and display stickperson

```
leftleg:=[[ 5,-1,0],[ 5,0,2]]:
Leftleg:=plots[polygonplot3d]( leftleg,style=wireframe, color=orange,
thickness=3):

body:=[[ 5,0,2],[ 5,0,3.5]]:
```

```
Body:=plots[polygonplot3d](body,style=wireframe, color=red,
thickness=3):

leftarm:=[[ 5,0,3],[ 5,-2,3]]:
Leftarm:=plots[polygonplot3d](leftarm,style=wireframe, color=purple,
thickness=3):

partialstickperson:=plots[display]([Leftarm,
Body,Leftleg]):
```

11.4.1 *Exercise: Assume that all of the light from the globe emits from the center of the light source and determine stickperson's shadow and add it to the picture. Note that the shadow will consist of two parts - the part on the ground and the part on the wall.*

11.4.2 *Exercise: Make an animation in which the light moves toward stickman. Make the shadow move appropriately.*

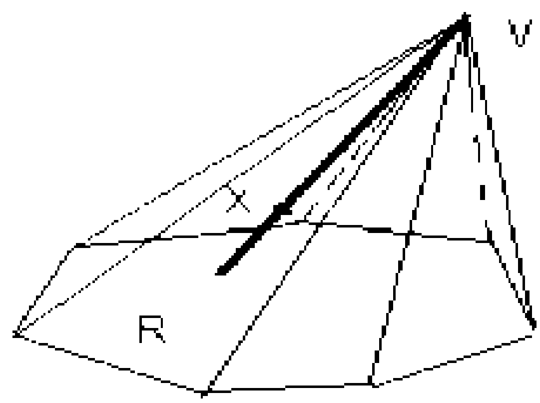
11.4.3 *Exercise: Put a flag in stickman's hand and extend the shadow to include the flag.*

11.4.4 *Exercise: Animate stickman to the extent that he/she waves the flag, extend the animation to include the flag shadow.*

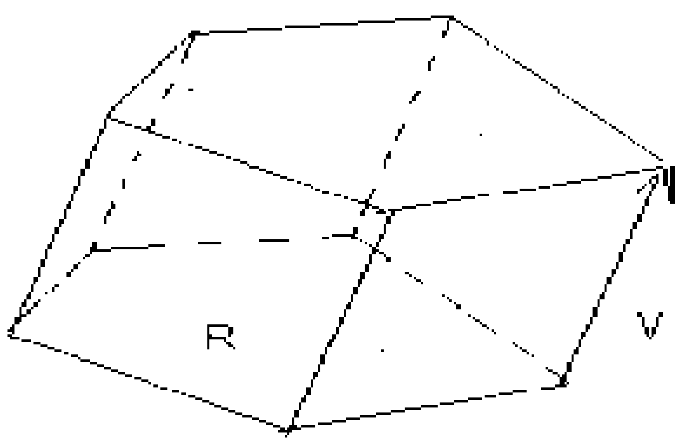
11.4.5 *Exercise: Make stickperson's head a (three dimensional) cube and extend the shadow.*

12 Pyramids and Prisms

2 Definition of pyramid *Take a polygonal region R and a point V not in the plane of R. The set of points X which are on the line segment from V to some point in R is called the pyramid with vertex V and base R.*



3 Definition of prism *Take a polygonal region R and a vector V. The set of all points X such that* $X = P + tV$*, where P is in the region and* $t \varepsilon [0, 1]$ *is called the prism with base R and height vector V. If the vector V is perpendicular to the base then the prism is called a right prism.*



4 *The top of the prism with base R and height vector V is defined to be the set of all points X such that* $X = P + V$*, where P is in R. Note that the top of the prism is the translate of the base by adding V. So it is congruent to the base.*

12.1 Exercise.

12.1.1 *Use plots[polygonplot3d] to draw the base and sides of the pyramid with base R= [[0, 0, 0], [1, 0, 0], [2, 1, 0], [1, 2.5, 0], [0, 2, 0], [-1, 1, 0]] and vertex V=[1,2,4]. Then use plots[display] to draw the pyramid.*

```
base :=
[[0,0,0],[1,0,0],[2,1,0],[1,2.5,0],[0,2,0],[-1,1,0]];
```

$$base := [[0, 0, 0], [1, 0, 0], [2, 1, 0], [1, 2.5, 0], [0, 2, 0], [-1, 1, 0]]$$

```
V := [1,2,4];
```

$$V := [1, 2, 4]$$

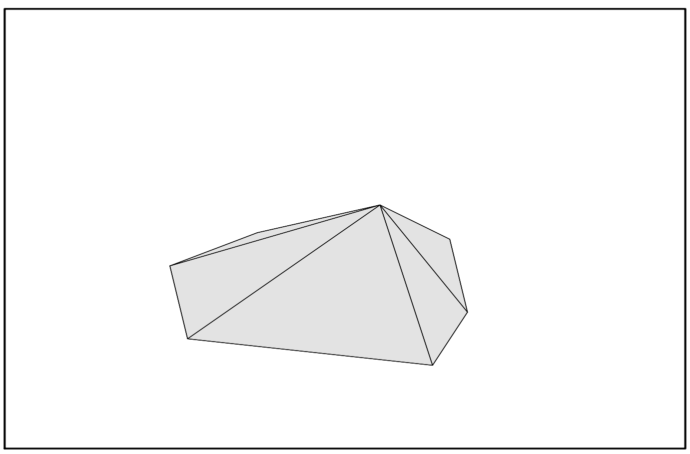
```
pl1 :=
plots[polygonplot3d](base,style=patch,color=magenta):
```

```
sides := NULL:
for i from 1 to nops(base)-1 do
sides := sides,plots[polygonplot3d]([V,base[i],base[i+1]],
style=patch,color=yellow) od:
```

Color the remaining side.

```
sides :=
sides,plots[polygonplot3d]([V,base[1],base[-1]],
style=patch,color=red):
```

```
plots[display]([pl1,sides]);
```



12.2 More drawing exercises on pyramids.

1. Define a word in Maple, pyramid(R,V), which draws the pyramid with base R and vertex V.

2. Modify your definition of pyramid to include a color option.

3. Another way to draw a pyramid, is to draw the base and then draw the segments from the vertex of the pyramid to each vertex of the base. You can draw a segment in Maple with plots[polygonplot3d]. Modify your definition of pyramid to do this.

12.3 Exercises

12.3.1 *Exercise. Draw the cylinder with base base R= [[0, 0, 0], [1, 0, 0], [2, 1, 0], [1, 2.5, 0], [0, 2, 0], [-1, 1, 0]] and height vector V=[1,2,3]. Hint: All you need to do is draw the base, the top, and the sides of the prism.*

12.3.1 A solution

: We can start with the drawing we made of a pyramid, and make a few modifications.

```
base :=
[[0,0,0],[1,0,0],[2,1,0],[1,2.5,0],[0,2,0],[-1,1,0]];
```

$$base := [[0, 0, 0], [1, 0, 0], [2, 1, 0], [1, 2.5, 0], [0, 2, 0], [-1, 1, 0]]$$

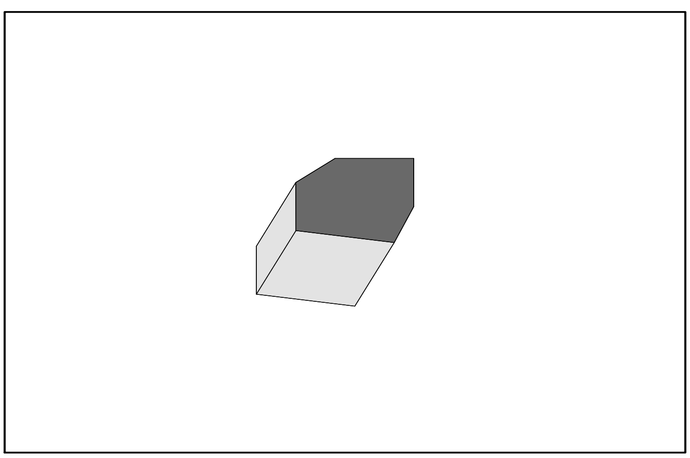
```
V := [1,2,4];
```

$$V := [1, 2, 4]$$

```
top := map(`+`,base,V) ;
```

$$top := [[1, 2, 4], [2, 2, 4], [3, 3, 4], [2, 4.5, 4], [1, 4, 4], [0, 3, 4]]$$

```
pl1 :=
plots[polygonplot3d]([top,base],style=patch,color=magenta):

sides := NULL:
for i from 1 to nops(base)-1 do
sides := sides,plots[polygonplot3d]([
base[i],base[i+1],V+base[i+1],V+base[i]],style=patch,color=yellow)
od:

sides :=
sides,plots[polygonplot3d]([base[1],base[-1],V+base[-1],V+base[1]],sty
le=patch,color=red):

plots[display]([pl1,sides],scaling=constraine
d);
```



Exercises on prisms.

12.3.2 *Define a Maple word prism(R,V) to draw the prism with base R and height vector V.*

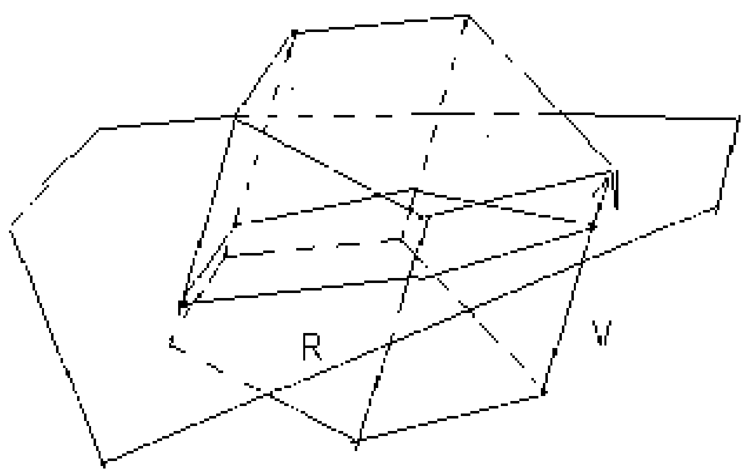
12.3.3 *Is a cube a prism? What about a tetrahedron?*

12.3.4 *What is another name for this prism?*

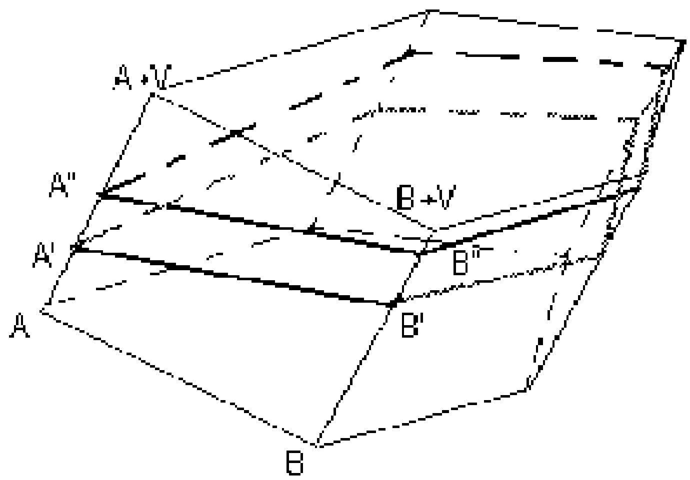
prism([[0,0,0],[2,0,0],[2,0,1],[1,0,2],[0,0,1]],[0,2,0])

12.4 Cross Sections of pyramids and prisms

5 *Take a pyramid and a plane which cuts the vertex away from the base. The intersection of the plane with the pyramid is called a cross section of the pyramid. Similarly, a cross section of a prism is the intersection of the prism with a plane which cuts the top from the base of the prism. In each case, the plane which is used to form the cross section is called the cross sectioning plane.*

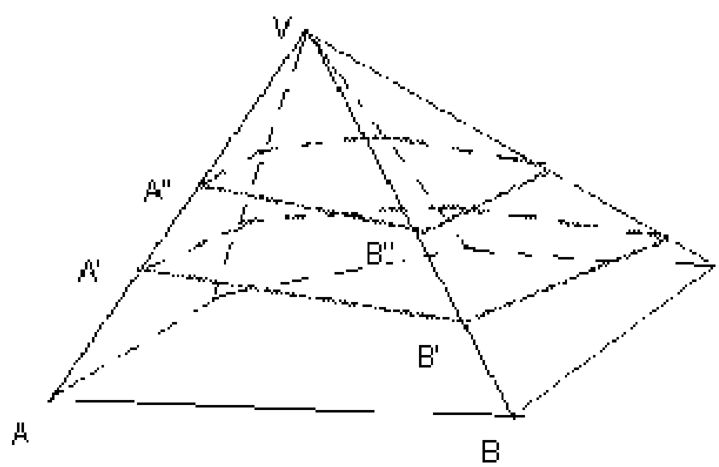


1 Theorem about cross sections of a prism. Parallel cross sections of a prism are congruent. *(Note: two cross sections are parallel if their cross sectioning planes are parallel.)*



Proof: Take any side AB(B+v)(A+V) of the prism. The two parallel planes intersect this side in two parallel segments A'B' and A"B", where A' and A" are on the segment A(A+V), labelled with A' between A and A" and B' and B" are on B(B+V), labelled with B' between B and B". The segments A'A" and B'B" are parallel, since they are both parallel to the vector V. So the quadrilateral A'B'B"A" is a parallelogram. Let U be the vector from A' to A". Then since A'B'B"A" is a parallogram, B" = B' + U. Continue all the way around the cross section containing A', and use the same argument to show that the vertices on the cross section containing A" are all of the from X" = X'+U, where X' is a vertex on the cross section containing A' . This means that the cross section containing A" is the translate of cross section containing A' by adding U. So these cross sections are congruent by translation. *qed*

2 Theorem about cross sections of a pyramid. Parallel cross sections of a pyramid are similar.



Proof. We can assume (by translating if need be) that the vertex of the pyramid is at the origin. Take any one of the sides ABV of the pyramid. The two parallel planes intersect this side in two parallel

segments A'B' and A1B1, where A' and A1=A" are on the segment AV, labelled with A' between A and A1 and B' and B1=B" are on BV, labelled with B' between B and B". So the angles VA'B' and VA"B" are congruent, since they are corresponding angles of the transversal VA of the parallel lines A'B' and A"B". Then the triangles VA'B' and VA1B1 are similar since corresponding angles are congruent. Hence corresponding sides are proportional. In particular,
$\frac{VA'}{VA1} = \frac{VB'}{VB1}$. Call this constant of proportionality k. Then for each vertex X of the cross section containing A', the point $X1 = kX'$ is a vertex of the cross section containing A1. Hence the cross section containing A' is carried onto the cross section containing A1 by the similiarity transformation $X \to kX$. *qed*

Exercises on cross sections of a prism.

12.4.1 Exercise: *Show that the converse of the theorem about prisms is false, by finding two congruent cross sections of a cube which are not parallel.*

12.4.2 Exercise: *Define a maple word crossect(base,vertex,plane) which takes a polygonal base, a height vector, and an equation for a cross sectioning plane and returns the polygonal cross section of the prism.*

13 Cross Sections of triangular prisms

13.1 The Theorem and its proof.

3 *Theorem: If P is a triangular prism and T is any triangle then there is a plane whose intersection with P is similar to T.*

We will prove the following restatement of the theorem:

Let ABC and PQR be any triangles in the x-y plane, K the vector [0,0,1] and s,t any numbers. Then s and t may be chosen so that the triangle C(A+tK)(B+sK) is similar to PQR..

Proof: Let K=[0,0,1]. Then the theorem asserts that given any other triangle PQR we can choose t and s so that the triangle (A+tK)C(B+sK) is similar to PQR. The two triangles are similar if two legs are proportional and the included angles are equal ("side angle side").

Referring to the diagram place the triangle ABC (or one similar to it) so that C=[0,0,0], B=[1,0,0], and A=[a,b,0]] with a,b positive. Place the triangle PQR (or one similar to it) so that P=A, Q=C, and R=[p,q,r[with p,r positive. Then we want to choose s and t so that the $\angle((A + t\,K)\,C\,(B + s\,K))$ has the same cosine as QPR and the lengths of the vectors A+tK and B+tK are proportional to QP and RP, the legs of QPR by some factor k. We have arranged for C and P to be the origin so that we can use that the cosine of the angle between two vectors is their dot product divided by the products of their lengths. We then have the three equations:

$$\frac{\mathrm{dotprod}(A{+}tK,\,B{+}tK)}{\mathrm{length}(A{+}t\,K)\,\mathrm{length}(B{+}s\,K)} = \frac{\mathrm{dotprod}(Q,R)}{\mathrm{length}(Q)\,\mathrm{length}(R)}$$

and the other two are

$$\text{length(A+tK)} = \text{k length(Q)}$$

$$\text{length(B+sK)} = \text{k length(R)}$$

Using the second two equations in the denominators of the first and knowing that the dot product of Q and R is p we have, on substitution the three equations:

$$\mathbf{dotprod}(A + tK,\; B + sK) = k^2\,p\ ,$$

$$\mathbf{length}(A + t\,K) = k\ ,$$

$$\mathbf{length}(B + s\,K) = k^2\,(p^2 + q^2)$$

Putting these into Maplese,

```
A:=[a,b,0]; B:=[1,0,0];
C:=[0,0,0];K:=[0,0,1];
P:=[0,0,0];Q:=[1,0,0];R:=[p,q,0];
eq1 := linalg[dotprod](A + t*K,B + s*K) = k^2*p;
eq2 := linalg[dotprod](A + t*K,A + t*K) = k^2;
eq3 := linalg[dotprod](B + s*K,B + s*K) = k^2*(p^2 + q^2);
```

$$A := [a,\ b,\ 0]$$

$$B := [1, 0, 0]$$

$$C := [0, 0, 0]$$

$$K := [0, 0, 1]$$

$$P := [0, 0, 0]$$

$$Q := [1, 0, 0]$$

$$R := [p, q, 0]$$

$$eq1 := a + t\,s = k^2\,p$$

$$eq2 := a^2 + b^2 + t^2 = k^2$$

$$eq3 := 1 + s^2 = k^2\,(p^2 + q^2)$$

Use eq2 to eliminate k^2 from the first and third euqations.

```
EQ1:= lhs(eq1)=lhs(eq2)*p;
EQ2:=lhs(eq3)=lhs(eq2)*(p^2+q^2);
```

$$EQ1 := a + t\,s = (a^2 + b^2 + t^2)\,p$$

$$EQ2 := 1 + s^2 = (a^2 + b^2 + t^2)\,(p^2 + q^2)$$

We can find solve for values of s and t which satisfy these equations by:

```
solve({EQ1,EQ2},{s,t});
```

$$\{t = \%1,\ s = \frac{-a + p\,a^2 + p\,b^2 + p\,\%1^2}{\%1}\}$$
$$\%1 := \mathrm{RootOf}(\_Z^4\,q^2 + (-1 - a^2\,p^2 + 2\,a\,p - b^2\,p^2 + a^2\,q^2 + b^2\,q^2)\,\_Z^2 - p^2\,b^4 - a^2 + 2\,p\,a^3 + 2\,a\,p\,b^2 - p^2\,a^4 - 2\,p^2\,a^2\,b^2)$$

This tells us that if at least one of the roots of "%1" non-zero and real then there is a solution. We need to analyze "%1" to determine whether it has a non-zero root. The following extracts the actual polynomial

```
p1:=op(1,%1);
```

$$p1 := \_Z^4\,q^2 + (-1 - a^2\,p^2 + 2\,a\,p - b^2\,p^2 + a^2\,q^2 + b^2\,q^2)\,\_Z^2 - p^2\,b^4 - a^2 + 2\,p\,a^3 + 2\,a\,p\,b^2 - p^2\,a^4 - 2\,p^2\,a^2\,b^2$$

By inspection we see that although p1 is a quartic polynomial it is actually a quadratic in $\_Z^2$. We might work with the quadratic by substituting $\sqrt{X}$ for _Z

```
p2:=subs(_Z=sqrt(X),p1);
```

$$p2 := X^2\,q^2 + (-1 - a^2\,p^2 + 2\,a\,p - b^2\,p^2 + a^2\,q^2 + b^2\,q^2)\,X - p^2\,b^4 - a^2 + 2\,p\,a^3 + 2\,a\,p\,b^2 - p^2\,a^4 - 2\,p^2\,a^2\,b^2$$

Thus in order for for there to be a solution we must find a positive root to p2.

Since the lead coefficient of p2 is the positive number q^2 its graph is a upward-opening parabola which crosses the x-axis twice if p2(z) is negative for some value of z. (The "twice" is important since we need a non-zero root): if there are two roots one of them must be non-zero. In fact "const", the constant term of p2 is the negative of a square.

```
const:=subs(_Z=0,p1);
const:=factor(const);
```

$$const := -p^2\,b^4 - a^2 + 2\,p\,a^3 + 2\,a\,p\,b^2 - p^2\,a^4 - 2\,p^2\,a^2\,b^2$$

$$const := -(-a + p\,a^2 + p\,b^2)^2$$

If conts is nonzero, the polynomial will have two real roots. We can easily find when the constant is zero by solving

```
solve(-a+p*a^2+p*b^2=0,p);
```

$$\frac{a}{a^2 + b^2}$$

Thus const is zero if and only if $p = \frac{a}{a^2+b^2}$. and we must deal with this possibility. If it happens then we can incorporate this into the original equations and solve again.

```
EQ1a:=subs(p=a/(a^2+b^2),EQ1);
EQ2a:=subs(p=a/(a^2+b^2),EQ2);
SLNS:=solve({EQ1a,EQ2a},{s,t});
```

$$EQ1a := a + t\,s = \frac{(a^2 + b^2 + t^2)\,a}{a^2 + b^2}$$

$$EQ2a := 1 + s^2 = (a^2 + b^2 + t^2)\,(\frac{a^2}{(a^2 + b^2)^2} + q^2)$$

$$SLNS := \{t = 0,\ s = \mathrm{RootOf}((a^2 + b^2)\,\_Z^2 - q^2\,a^4 + b^2 - 2\,q^2\,a^2\,b^2 - q^2\,b^4)\}, \{$$
$$s = \frac{\mathrm{RootOf}((a^2 + b^2)\,\_Z^2 + q^2\,a^4 + 2\,q^2\,a^2\,b^2 + q^2\,b^4 - b^2)\,a}{(a^2 + b^2)\,q},$$
$$t = \frac{\mathrm{RootOf}((a^2 + b^2)\,\_Z^2 + q^2\,a^4 + 2\,q^2\,a^2\,b^2 + q^2\,b^4 - b^2)}{q}\}$$

The problem is to see that one of these posibilities is a pair of real numbers. We will have a solution with t=0 if there is a real root to the quadratic

```
q1:=
(a^2+b^2)*_Z^2-q^2*a^4-2*q^2*b^2*a^2-q^2*b^4+b^2;
```

$$q1 := (a^2 + b^2)\,\_Z^2 - q^2\,a^4 + b^2 - 2\,q^2\,a^2\,b^2 - q^2\,b^4$$

Or there will be a solution if the other quadratic has real roots.

```
q2:=(a^2+b^2)*_Z^2+q^2*a^4+2*q^2*b^2*a^2+q^2*
b^4-b^2;
```

$$q2 := (a^2 + b^2)\,\_Z^2 + q^2\,a^4 + 2\,q^2\,a^2\,b^2 + q^2\,b^4 - b^2$$

A quadratic has real roots if and only if its discriminant is not-negative so we must see that the discriminnant of one of q1, q2 is non-negative. In this

```
discrim(q1,_Z);
discrim(q2,_Z);
```

$$4\,(a^2+b^2)\,(q^2\,a^4+2\,q^2\,a^2\,b^2+q^2\,b^4-b^2)$$

$$-4\,(a^2+b^2)\,(q^2\,a^4+2\,q^2\,a^2\,b^2+q^2\,b^4-b^2)$$

Since each of these discriminants is the negative of the other, one of these quadratics is guaranteed to have real roots. Hence the problem can be solved.

One value of having an argument like this worked out in Maple is that we can use the argument to solve specific problems. For instance:

13.2 A prism with an equilateral base and a 3-4-5 right cross-section

13.2.1 Exercise: *Cut a triangular prism with an equilateral base so that the cross-section of the cut is a 3-4-5 right triangle?*

Refering to the original diagram we want ABC to be the equilateral triangle with C=[0,0,0], B=[1,0,0], and A= [$\frac{1}{2}$, $\frac{\sqrt{3}}{2}$,0]. We want PQR to be a 3-4-5 triangle. One choice is P=[0,0,0], Q=[1,0,0], R=[0, $\frac{3}{4}$,0]. Once these are determined the general calculations become specific and we simply specify:

a:=1/2;b:=sqrt(3)/2; p:=0;q:=3/4;

Now we simply follow the calculations in the proof

```
A:=[a,b,0]; B:=[1,0,0]; C:=[0,0,0];K:=[0,0,1];
P:=[0,0,0];Q:=[1,0,0];R:=[p,q,0];
eq1:=linalg[dotprod](A+t*K,B+s*K) = k^2*p;
eq2:=linalg[dotprod](A+t*K,A+t*K) = k^2;
eq3:=linalg[dotprod](B+s*K,B+s*K) =
k^2*(p^2+q^2);
```

Use eq2 to eliminate k^2 from the first and third equations

```
EQ1:= lhs(eq1)=lhs(eq2)*p;
EQ2:=lhs(eq3)=lhs(eq2)*(p^2+q^2);
```

we can find solve for values of s and t which satisfy these equations by:

```
ans:=solve({EQ1,EQ2},{s,t});
```

At this point we are happy with floating point values for s and t.

```
ans:=solve({EQ1,EQ2},{s,t});
ans:=evalf(ans):
```

To get the actual triangles we substitute for s and t in the vertices A+tK, B+tK, and the origin.

```
ANS:=subs(ans ,[A+s*K,C,B+t*K]);
ANS:=expand(ANS);
```

To check that these are actually the vertices of a 3-4-5 triangle we can calculate the lengths of the sides of the triangle. We need a distance function

```
dist:=proc(X,Y)
sqrt((X[1] - Y[1])^2 + (X[2] - Y[2])^2 + (X[3] -
Y[3])^2)
end;
```

```
legs:=[dist(ANS[1],ANS[2]),dist(ANS[2],ANS[3]), dist(ANS[3],ANS[1])];
```

If this is proportional to a 3-4-5 triangle then the first leg must correspond to one of length 3 thus we scale it.

```
check:=3/legs[1] *legs;
check:=expand(check);
```

13.3 EXERCISES

13.3.1 Exercise: *Make a three dimensional Maple plot of the of the figure sketched below. Its base is an equilateral triangle the sides are perpendicular to the base, and the top is a 3-4-5 triangle.*

13.3.2 Exercise: *Do the previous exercise with the top an isosceles right triangle.*

13.3.3 Exercise: *Suppose* $\mathrm{f}(x) = a\,x^4 + b\,x^3 + c\,x^2 + d\,x + e$ *with a positive. Suppose there is a z such that* $\mathrm{f}(z) < 0$. *Prove that f has at least two different roots.*

13.3.4 Exercise: *Suppose* $\mathrm{f}(x) = a\,x^2 + b\,x + c$ *with a positive. If* $c < 0$ *prove that* $\mathrm{g}(x) = a\,x^4 + b\,x^2 + c$ *has a real root.*

14 Exploring Desargues Theorem.

14.1 Points and Lines at Infinity

The principal "synthetic" tool for determining when one plane figure in space is the shadow of another is Desargue's Theorem which is most properly stated as a theorem in Projective Geometry rather than the cartesian or affine geometry in which we are working. This would be a good place to begin an enjoyable study of projective geometry, however one of the prinicpal ideas in problem solving is to learn how to collect, understand, and employ mathematical tools to solve the problem at hand, usually postponing their full development. This often entails extracting a tool, theorem, or technique from its "natural" milieu and phrasing it and its proof or explanation in the language of the problem at hand. This often results in much faster progress on the study of the current problem purchased at the cost of more lengthy and less elegant statements and arguments. With regard to the Desargues Theorem we need the notion of "point at infinity" of a line.

6 Definition: *If L is a line in any number of dimensions) and L is written in parametric form $P + t\,D$ then we call "D" the direction of the parameterization. A line has many parameterizations and different parameterizations may have different directions. , e.g. the parameterization $P + t\,2\,D$ has the 2D as direction. However the directions of any two parameterizations of the same line are proportional (i.e. one is a numerical multiple of the other).*

We define the direction of a line L as the totality of all of the directions of its parameterizations. For any line this is an infinite number of things but any two of them are proportional so if we know one of them we know them all. We use any of its members as a name for the direction of the line. Thus for exampe we say that " [1,0,0] is the direction of the x-axis". We also say "[27,0,0] is the direction of the x-axis". What we we would say to be perfectly precise is "[1.0,0 is a name for the direction of the x-axis". However that is too wordy and experience shows that there is no confusion in the "shorthand" version provided everyone understands the convention. Lest one feel that this is something new we note that this is exactly the same idea as saying " $\frac{3}{2}$ is the average of 1 and 2" while " $\frac{6}{4}$ is the average of 1 and two" is mathematically just as correct. The symbols $\frac{1}{2}$ and $\frac{6}{4}$ are but two of the infinitely many names (numerals) for a particular number. With this introduction we can now give the following:

7 Definition: *If L is a line then "the point at infinity on L" is the direction of L.*

Example: The point at infinity on the x-axis is the direction [1,0,0]. Note that the "finite" point [1,0,0] is also on the x-axis but this is different from the direction [1,0,0] which happens to be repersented by the same name. Note that the point at infinity on the x-axis is also the direction [-193,0,0].

Example:What is the point at infinity on the line K which contains the points [1,2,3] and [8,-3,7].

Solution: A parametric equation for ths line is [1,2,3] + t([8,-3,7] - [1,2,3]). Thus the direction of this parameterization of the line is [7,-5,4] so this is one of the names for the direction of the line, i.e. one of the names for the point at infinity on the line K. $[\,1,\ -\frac{5}{7},\ \frac{4}{7}]$ is another name for this direction. Note that the "finite point" [7,-5,4] does not lie on the line K, In the exercises we find that the "finite point" with coordinates given by one of the names for the point at infinity on a line will also be on the line if and only if the line goes through the origin.

Example: Determine if the line L which contains [1,2,3] and [5,1,2]

and the line M which contains [6,0,1] and [3,2,7]] have a point in common (including points at infinity)

Solution: We can use the Maple solve function to find such points from the parametric forms
.

```
L:=expand([1,2,3]+t*([5,1,2]-[1,2,3]));
M:=expand([6,0,1]+s*([3,2,7]-[6,0,1]));
eq1:= L[1]-M[1]=0;
eq2:=L[2]-M[2]=0;
eq3:=L[3]-M[3]=0;
solve({eq1,eq2,eq3},{s,t});
```

The absence of a solution to these equations tells us that there is no point of intersection "at finite distance". To determine if there is a common point at infinity we calculate the point of infinity of each and see if they are the same. The point of infinity of L is its direction $[5, 1, 2] - [1, 2, 3] = [4, -1, -1]$ and that of M is $[3, 2, 7] - [6, 0, 1] = [-3, 2, 6]$. These are not proportional since there is no number k such that $k\,[4, -1, -1] = [-3, 2, 6]$. Thus these lines in 3-space do not meet, even at infinity.

It is true, however that any two lines in space which lie in a plane will either meet at finite distance or will have a common point at infinity. Indeed if the two lines do not meet they are parallel in that plane and thus have the same direction. This fact is checked in the exercises.

Now we can state a the following "obvious" fact:

4 *Theorem: Two lines are parallel if and only if they have the same point at infinity.*

The proof is worked out in the exercises.

Now we have some new ways to express familiar ideas

8 *Definition: The line L which contains the point A and the point at infinity P is simply the line through A with direction P. For instance one parameterization of L would be $A + t\,P$. The line through two points at infinity is the collection of all directions of lines in a plane which contains two lines having those points at infinity The projection of a point W onto a plane L from a point at infinity, P, is the point of intersection with L of the line through W having direction P. The projection of an object H onto a plane L from a point at infinity, P, is the set of projections of all points of H onto L from P.*

14.2 Exercises On Points and Lines at Infinity

14.2.1 *EXERCISE: If two lines lie in a plane and do not have a point at finite distance in common then they have the same direction and thus have a common point at infinity.*

Hint: Suppose L and M are lines in a plane P which do not meet at finite distance. This means that if A and B are on L and C and D are on M then the equation A+t*(B-A) = C+s*(D-C) has no solution for any s an t. That is there are no s,t such that A-C= t*(B-A) + s*(D-C). However if aX+bY+cZ+d=0 is the equation for P then you can check that A-C, B-A, and D-C are all in the plane Q whose equation is aX+bY+cZ=0. If, say, c is not zero then this means that the Z-coordinate of any element of Q is completely determined by its X and Y coordinates. This means that one can't find s and t so that the first two coordinates of t*(B-A) + s*(D-C) are euqal to the first two coordinates of

A-C because if you could then the third coordinates would automatically agree. However solving for the first two coordinates amounts to solving two linear equations in two unknowns which can always be done unless the rows of coefficients are proportional. This means that the first two coordinates of B-A and D-C are proportional but the third coordinates will then also be proportional by the fact that we can solve for the third coordinate (This ability to solve for one of the coordinates in terms of the other two is how the fact that the lines are in a plane comes into play).

14.2.2 *EXERCISE: Show that if L is a line in space then the directions of any two parameterizations of L are proportional*

14.2.3 *EXERCISE: If K is a line and W=[a,b,c] is the point at infinity on K then the "finite point [a,b,c]" is on K if and only if K goes through the origin.*

14.2.4 *EXERCISE: Prove that two lines in the plane are parallel if and only of they have the same point at infinity.*

Hint: First suppose the lines have the same point at infinity. If they are different lines we need to show that they do not meet. One way to do this is to show that if you could find a point of intersection then the lines are identical. Let Dir be the common point at infinity then parameterizations for K and L are of the form $\boldsymbol{A + t\,Dir}$ and $\boldsymbol{B + s\,Dir}$. If A and B are the same then the lines K and L are the same. If there were a common point we could find s_0 and t_0 such that $\boldsymbol{A} + t_0\,\boldsymbol{Dir} = \boldsymbol{B} + s_0\,\boldsymbol{Dir}$, i.e. $\boldsymbol{A} - \boldsymbol{B} = (s_0 - t_0)\,\boldsymbol{Dir}$. Show that if there are such numbers then the points A and B would be on both lines.

For the converse one must show that if the lines are parallel then any two paramaterizations of these lines have proportional directions. We may assume that the lines are distinct. Suppose E+tF is a parameterization of L and $\boldsymbol{G + s\,H}$ is a parameterization for K. We can attempt to find values of s and t which give common points on K and L. Since the lines are distinct and parallel the following must fail to find such and s and t. Find the condition on E and H which this implies and explain why this means that the two directions are proportional.

```
E:=[e1,e2]; F:=[f1,f2]; G:=[g1,g2]; H:=[h1,h2];
DIFF:=expand(E+t*F - (G+s*H));
solve({DIFF[1]=0,DIFF[2]=0},{s,t});
```

14.2.5 *EXERCISE: Show that if C is not equal to C' then the points from which the triangles ABC and ABC' are in perspective are an entire line (including its point at infinity).*

14.3 Desargues's Theroem

We want to explore a very useful result, discovered in the mid 1600's by Girard Desargues, a french geometer who pioneered the development of projective geometry. First, we will just provide a couple of words to draw some pictures and then use them to draw a few pictures.

The first word projxy takes two points X and P and returns the point Q on the line thru X and P which is on the xy plane. The assumption is the line XP strikes the xy plane in only one point. The point Q is called the projection of X from P onto the xy plane.

```
projxy:=proc(X,P)
```

```
local Line,t,T,Q:
Line:=expand(t*P+(1-t)*X):
T:=solve(Line[3]=0,t):
Q := subs(t=T,Line):
end:
```

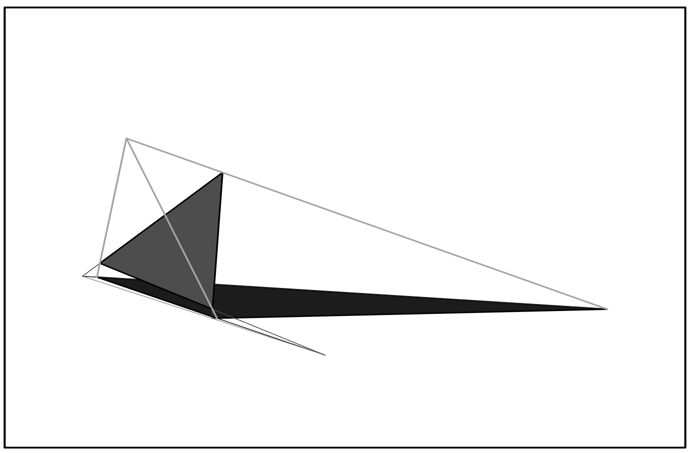
The second word takes a triangle ABC (think of it as lying above the xy plane) and a point P (think of it as lying above the triangle), computes the triangle obtained by the projection of ABC from P , call it triangle A1B1C1. Then line sements and triangles are plotted and displayed in one big picture.

```
desarguexy:= proc (A, B, C,P)
local triABC, triABCprime,A1,B1,C1, AA1, BB1, CC1, AB, abpoint, BC,
bcpoint, CA, capoint, abline, abprimeline, bcline, bcprimeline,
caprimeline, caline, axis, DEAS;
A1:=projxy(A,P);
B1:=projxy(B,P);
C1:=projxy(C,P);
triABC := plots[polygonplot3d]([A, B, C],style = patch,thickness =
3,color = red);
triABCprime := plots[polygonplot3d]([A1, B1, C1],style =
patch,thickness = 3,color = blue);
AA1 := plots[polygonplot3d]([P, A1],style = wireframe,color = tan,
thickness = 3);
BB1 := plots[polygonplot3d]([P, B1],style = wireframe,color = tan,
thickness = 3);
CC1 := plots[polygonplot3d]([P, C1],style = wireframe,color = tan,
thickness = 3);
AB := expand(s*A+(1-s)*B-t*A1-(1-t)*B1);
abpoint := expand(subs(solve({AB[1] = 0, AB[2] = 0, AB[3] = 0},
{s, t}),s*A+(1-s)*B));
BC := expand(s*C+(1-s)*B-t*C1-(1-t)*B1);
bcpoint := expand(subs(solve({BC[3] = 0, BC[2] = 0,BC[1]=0},
{s, t}),s*C+(1-s)*B));
CA := expand(s*A+(1-s)*C-t*A1-(1-t)*C1);
capoint := expand(subs(solve({CA[1] = 0, CA[3] = 0, CA[2] = 0},{s,
t}),s*A+(1-s)*C));
abline := plots[polygonplot3d]([B, abpoint],style = wireframe,color =
red,
thickness = 2);
abprimeline := plots[polygonplot3d]([B1, abpoint],style =
wireframe,color = blue,thickness = 2);
bcline := plots[polygonplot3d]([C, bcpoint],style = wireframe,color =
red,thickness = 2);
bcprimeline := plots[polygonplot3d]([C1, bcpoint],style =
wireframe,color = blue,thickness = 2);
caprimeline := plots[polygonplot3d]([A1, capoint],style =
wireframe,color = blue,thickness = 2);
caline := plots[polygonplot3d]([A, capoint],style = wireframe,color =
red,thickness = 2);
axis := plots[polygonplot3d]([abpoint, bcpoint, capoint],style =
wireframe,
color = green,thickness = 2);
DEAS := [triABC, triABCprime, AA1, BB1, CC1, abline, abprimeline,
```

```
bcline,
bcprimeline, caline, caprimeline, axis];
plots[display](DEAS)
end;
```

Here is a short movie made with desarguexy.

```
movie:=seq(
desarguexy([1,0,1],[2,3,4],[3,1,2],[1,-2+i/2,5]),i=0..10):

plots[display]([movie],orientation=[-36,41],i
nsequence=true,axes=normal);


plots[display](desarguexy([1,0,1/2],[2,3,4],[
3,1,1/4],[1,1,5]), orientation=[-36,41]);
```



9 *Definition: Two triangles ABC and A'B'C' are said to be in perspective if there is a point O such that OAA', OBB' and OCC' are all straight lines. In this case they are said to be in perspective from 0.*

Note that the point O can be a point at infinity (which says that the lines OAA', OBB' and OCC' are parallel. Further, O need not be unique, for instance if A=A',B=B',C=C' then 0 can be any point.

Desargue's Theorem gives necessary and sufficient conditions for two triangles to be in perspective. It can be easily stated in that form. However for convenience in understanding why it is true we break it into two theorems. Our treatment follows that of Hodge and Pedoe in *Methods of Algebraic Geometry*, Cambridge University Press, Cambridge University Press 1968

5 *Theorem 1: If two triangles ABC and A'B'C' are in perspective then there exists a line l passing through every point of intersection of (extensions of) corresponding pairs of edges of the two triangles. That is there is a line which contains: the intersection of AA' and BB', the intersection of AA' and CC', and the intersection of BB' and CC'.*

6 *Theorem 2: If ABC and A'B'C' are two triangles such that there is a line l passing through a point common to BC and B'C', a point common to AC and A'C', and a point common to AB and A'B' then the triangles are in perspective.*

14.4 Proof of Desargue's Theorems

The theorems include the cases where the points of intersection and the line which contain them may lie at infinity. The case where the line is at infinity is simply that where the two planes defined by the triangles are parallel.

CASE 1: The two triangles do not lie in a plane.

Theorem 1: Suppose, as in the diagram below that the two triangles are in perspective and that their planes are distinct. Then the line of intersection of the two planes must contain all points of intersection of corresponding sides since each such point is the intersection of a line in one of the planes with a line in the other. Thus if triangles in different planes are in perspective then the points of intersection of corresponding sides lie on a line.

Theorem 2: Suppose the two triangles ABC and A'B'C' are arranged so that the corresponding extensions of edges meet along a line - as shown in the diagram.

Plot: desar1.eps

We claim that there is a point of perspective which is the same as saying that the three lines AA' BB', and CC' are concurrent, i.e have a point in common. Note that since BC and B'C' meet, the points B,B', C. C' are all in a plane, hence the lines BB' and CC' must meet or be parallel (i.e. meet at infinity). If these lines have two points in common or are parallel and have a point in common then they are the same line and since each lies in either the plane of ABC or that of A'B'C' they would have to lie in both and hence be the line of intersection (the green line). In such a case we would have C=C' and B=B' in which case projection from any point on the line AA' would project ABC onto A'B'C'.

If the points A and A' coincide then projection from the point of intersection of BB' and CC' (or from infinity if they are parallel) places the triangles in perspective. Thus the only remaining case is when none of the corresponding vertices are identical.

Let O be the point of intersection of BB' and CC' (or their common direction if they are parallel). If O does not lie on AA' (or,in the parallel case, if AA' is not parallel to the other two lines) then the points (A,A', and O) determine a plane L. In the parallel case we take L to be the plane containing line AA' and a line through A parallel to the lines BB' and CC'.

Since AB and A'B' meet, the lines AA' and BB' are in a common plane. Since the point 0 is on BB' that plane is L. Similarly L contains CC' and consequently A,B,C and A',B',C' are all co-planar. In the parallel case, since A,B,A', and B' are co-planar a line through A parallel to BB' would line in the plane of ABA'B' and hence L would be that plane. However the same holds for AA' and CC' so in this case also, L is the plane containing AA' and CC'. Thus in this case also we have that A,A',B,B', C,C' are all co-planar. This contradicts the assumption that ther planes of the two triangles are different and we have proved Case 1 of Desargue's Theorem.

Case2: The triangles lie in the same plane We will do this case in the form of an exercise set.

Theorem 1: (The Planar Case)

14.4.1 *EXERCISE: If A=A', B=B', C=C' then the theorem is trivial. (You should check through this to be sure you understand the statement(s) in the theorem.*

14.4.2 *EXERCISE: If B=B', C=C' then the line BC satisfies the theorem.*

14.4.3 *EXERCISE: If A=A' the line connecting A to any point common to BC and B'C' satisfies the theorem.*

We now show that the if there is a line which contains the edges BC and B'C' then we are done.

14.4.4 *EXERCISE: If no two corresponding vertices are equal and there is a line contining both the edges BC and B'C' then no other line contains an edge of each triangle.*

14.4.5 *EXERCISE: If BC and B'C' lie on a line they are the the only corresponding sides which extend to the same line. We then have that AB meets A'B' in one point and AC meets A'C' in just one point. The line connecting these two points satisfies the theorem.*

14.4.6 *EXERCISE: We now have that the coresponding vertices are all distinct and that no pairs of them lie on the same line. If the point O is one of the vertices, say A then B'C' satisfies all of the requirements of the theorem.*

14.4.7 *EXERCISE: If two non-corresponding vertices, say B and C' are equal but not they are not the point O then the lines BC and B'C' must coincide. Thus this case is contained in one covered in a previous exercise.*

We are thus in the planar case of Theorem 1 where all of the six vertices and the point of projection are distinct and thepairs of lines extending corresponding sides are also distinct.

14.4.8 *EXERCISE: Let L be a line through the point O which is not contained in the plane of the two triangles. Let V and V' be two points other than O on the line L and take the lines VV', and AA' which meet at 0. These lines span a plane PI and in PI the lines VA and V'A' have a unique point, A*, in common. SImilarly VB and V'B' have B* and VC and V'V' have C*.*

14.4.9 *EXERCISE: With the notation of the preceeding exercise, none of A*,B*,C* is in the plane of the two triangles, nor is any of them on the line L.*

14.4.10 *EXERCISE: The points A*, B*,C* do not lie on a line*

hint: Show that if they did then these points and V would lie in a plane which would be different from the plane of the triangles and would contain the points A,B, and C. Show that this would make A.B, and C lie on a line.

14.4.11 *EXERCISE: The triangles ABC and A*B*C* are two triangles in perspective from V and the triangles A'B'C' and A*B*C* are in perspective from V'*

Now that we know that these triangles are in perspective, by Case 1 we know that the lines AB and A*B* meet along the line of interesection of the planes of ABC and the point of intersection must be the point of intersection of the line A*B* with the plane of the triangles. Similarly, the lines AB and A'B' meet along the same line of interesection and the point of intersection must also be the point of intersection of the line A*B* with the plane of ABC. Thus AB and A'B' meet on the line of interesection of the two planes. The same argument holds for the intersections of the lines AC and A'C' and BC and B'C'. Thus line intersection of the two planes is the line promised by the theorem. This completes the proof of Theorem 1.

Theorem 2 (The Planar Case)

Recall that for Theorem 2 we must show that if the extended sides meet along a line then the triangles are in perspective. We proceed as with Theorem 1, reducing to the case where all points and lines are distinct. Let L be the line along which corresponding sides meet.

14.4.12 *EXERCISE: If A=A', B=B',C=C' then the theorem is trivial.(One should go thorugh this to firm up the statement of the theorem).*

14.4.13 *EXERCISE: If A=A' and B=B' but C and C' are distinct then the two triangles are in perspective from any point on CC'*

14.4.14 *EXERCISE: if A=A' but B and B' and C and C' are distinct then the triangles are in perspective from any point common to BB' and CC'.*

14.4.15 *EXERCISE: If the line BB' is equal to the line CC' then the triangles are in perspective from any point common to this line and AA'*

14.4.16 *EXERCISE: If L coincides with the line BC then C'A' must pass through C and B'A' must pass through B. The triangles are therfore in perspective from A'.*

14.4.17 *EXERCISE: If (the extensions of) two non-corresponding sides are equal, but not equal to L, say BC = C'A' then C=C' and this case reduces to a previous one.*

Hint: Argue that C' must be the point of intersection of BC and L and that C must be the point of intersection of A'C' and L

The previous cases place us in the situation where all of the corresponding vertices are distinct and all of the lines extending edges of the triangles are distinct.

Choose a plane PI which meets the common plane of ABC and A'B'C' at the line L and let P, Q, R be the intersections of be the intersections of BC, CA, and AB respectively with L.

14.4.18 *EXERCISE: Show that the points P, Q, R are distinct.*

hint: Show that if Q=R then A=A'

Draw through P, Q, R three lines in the plane PL which meet to form a triangle which we denote by A*B*C*.

14.4.19 *EXERCISE: Show that the triangles A*B*C* and ABC are in perspective from some point V and similarly A*B*C* and A'B'C' are in perspective from some point V'*

hint: Use the non-planar case of the theorem

14.4.20 *EXERCISE: Show that V and V' are distinct and neither lies in the plane of ABC*

hint: Show that if V=V' then A=A'. Show that if V were in the plane of ABC then A*B* and C* would be in that plane too.

14.4.21 *EXERCISE: Let O be the point where VV' meets the plane of ABC. Show that the lines AA', BB', and CC' all pass through O and hence the triangles ABC and A'B'C' are in perspective from O.*

hint: VA and V'A' meet in A*. Show that this implies that VV' and AA' meet in O.

14.5 Remarks on Using Desargues Theorem To Represent one Object as a Shadow of Another

Remark: To use Desargue's theorem to decide if two figures are in perspective we note that if they are in perspective then any pair of correpsponding triangles must also be in perspective. Assuming they are in different planes this means that all lines through corresponding pairs of lines in the figures must meet at the line of intersection of the two planes. Moreover since the point of perspective (or common direction) is determined by any pair of lines through corresponding points (or their direction if they are parallel), any two pairs of triangles which share an edge and are in perspective with their corresponding triangles, must correspond under projection from the same point (namely the point of intersection or common direction) of the two corresponding lines.

Thus if we want to determine if two figures which are not co-planar are in perspective we check that:

(1) The figures are composed of corresponding triangles such that each pair of points in a figure is contained in two members a chain of triangles with successive members of the chain sharing an edge

(2) Each pair of lines through corresponding vertices either meet along or are both parallel to the line of intersection of the planes of the two figures.

If both of the above conditions are true then the two figrues are in perspective.

Remark 2: If two non-coplanar, planar figures are in perspective then they remain in perspective through any rotation of one of them about the line of intersection of their planes.

The previous remarks provide us with a way to construct plane figures which are shadows of another plane figures which we state for quadrilaterals.

1. Draw the first quadrilateral on a plane surface and draw one of its diagonals

3. Draw a line (the axis) transversal to all of the extended edges and the diagonal but not intersecting the original figure.

4. From the point of intersection of the diagonal with the axis draw line.

5. From the points of intersection of each pair of edges which do not meet in one of the chosen diagonal points, draw a pair of lines which meet along the line in the previous step. These lines form the extended edges of a quadrilateral which is in perspective to the original.

6. "Fold" the plane at the axis so that the two quadrilaterals now lie in different planes which meet along the axis. They remain in perspective from a point which can be found as the intersection of the lines through any pair of corresponding points.

15 The Harmonic rays theorem.

7 Theorem (harmonic rays): *Let ABCD be a quadrilateral. Let F be the point of intersection of* $\overleftrightarrow{\mathrm{AD}}$ *with* $\overleftrightarrow{\mathrm{BC}}$ *and let G be the point of intersection of* $\overleftrightarrow{\mathrm{AB}}$ *with* $\overleftrightarrow{\mathrm{CD}}$ *. One of both of the points F and G may be on the line at infinity. Let* l *be the line through A parallel to* $\overleftrightarrow{\mathrm{FG}}$ *and let M be the intersection of the diagonals of ABCD. If* $\overleftrightarrow{\mathrm{BF}}$ *meets* l *at P and* $\overleftrightarrow{\mathrm{MF}}$ *meets* l *at Q then the line segments PQ and QA are congruent.*

15.1 A diagram

(execute these to set up diagram unless the geometry package setup for the theorem on shadows of squares has already been executed)

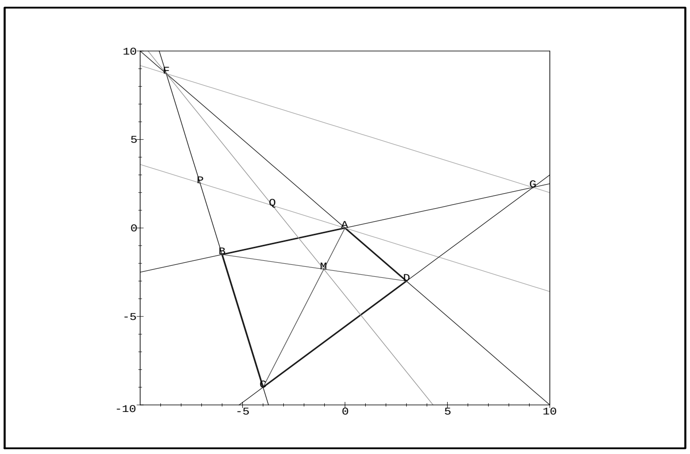
```
a1 =0:  a2 =0:

b1:=-6:  b2:=-3/2:

c1:=-4:  c2:=-9:

d1:=3:  d2:=-3:

geometry[point](A,[a1,a2]):
geometry[point](B,[b1,b2]):
geometry[point](C,[c1,c2]):
geometry[point](D,[d1,d2]):

geometry[segment](AB,[A,B]):
geometry[segment](AC,[A,C]):
geometry[segment](BC,[B,C]):
geometry[segment](CD,[C,D]):
geometry[segment](AD,[A,D]):
geometry[line](lineAD,[A,D]):
geometry[segment](BD,[B,D]):

geometry[line](lineCD,[C,D]):

geometry[line](lineBC,[B,C]):

geometry[line](lineAD,[A,D]):

geometry[line](lineBD,[B,D]):

geometry[line](lineAC,[A,C]):
geometry[line](lineAB,[A,B]):

geometry[intersection](F, lineBC,lineAD):

geometry[intersection](M, lineBD,lineAC):

geometry[intersection](G, lineAB,lineCD):

geometry[line](lineFM,[F,M]):

geometry[line](lineGM,[G,M]):
geometry[line](vanishingline,[F,G]):

axis:='axis':geometry[ParallelLine](axis,A,va
```

```
nishingline):

T:='T':geometry[intersection](T,
lineBD,axis):

Q:='Q': geometry[intersection](Q,
lineFM,axis):

R:='R':geometry[intersection](R, lineGM,
axis):

geometry[intersection](P, axis,lineBC):

harmonicdiagram:={axis(color=tan,
thickness=2), vanishingline(color=tan, thickness=2), P,Q,A,
lineAD(color=blue), AC(color=red),
BD(color=red),F,G,lineAB(color=blue),lineCD(color=blue),BC(color=blue,
thickness=3),B,C,D,M,lineBC(color=blue),AD(color=blue,thickness=3),CD(
color=blue,thickness=3),AB(color=blue,thickness=3),
lineFM(color=green)}:
```

15.2 PROOF

First we draw the diagram. When F or G is at infinity then ABCD has at least one pair of parallel sides. We leave it as exercises to handle theses very intersting cases.

```
geometry[draw](harmonicdiagram,scaling=constr
ained);
```



There is no loss in assuming that the point A is at the origin [0,0] and that B is [1,0] as this is a matter of choice of coordinates and scale, neither of which affects this problem. We use the notation Apt, Bpt, .. etc. to indicate points and to avoid conflict some Maple reserved symbols (e.g. "D" and "O").

```
restart;
```

```
Apt:=[0,0]:Bpt:=[1,0]:
Cpt:=[c1,c2]:Dpt:=[d1,d2];
```

$$Dpt := [d1,\ d2]$$

The theorem is all about points of intersection of lines so we provide a word which calculates points of intersection of pairs of lines described by points on them. "intsect(Apt,Bpt,Cpt,Dpt)" is the point of intersection of the line through Apt and Bpt with the one through Cpt and Dpt

```
intsect:=proc(Apt,Bpt,Cpt,Dpt) local s,t
,L1,L2,ANS,EQ:
L1:=Apt+t*(Bpt-Apt):
L2:=Cpt+s*(Dpt-Cpt):
EQ:=expand(L1-L2):
ANS:=solve({EQ[1]=0,EQ[2]=0},{s,t}):
simplify(expand(subs(ANS,L1))) end:
```

```
Mpt:=intsect(Apt,Cpt,Bpt,Dpt);
```

$$Mpt := [\frac{c1\ d2}{c2 - c2\ d1 + c1\ d2},\ \frac{c2\ d2}{c2 - c2\ d1 + c1\ d2}]$$

Now calculate F and G, the points of intersection of opposite sides

```
Fpt:=intsect(Bpt,Cpt,Dpt,Apt);
```

$$Fpt := [-\frac{c2\ d1}{-c2\ d1 - d2 + c1\ d2},\ -\frac{c2\ d2}{-c2\ d1 - d2 + c1\ d2}]$$

```
Gpt:=intsect(Apt,Bpt,Cpt,Dpt);
```

$$Gpt := [-\frac{-c2\ d1 + c1\ d2}{c2 - d2},\ 0]$$

The line parallel to the line through F and G which passes through A has parametric form A+t(F -G). With t=1 we have a point other than A on this line

```
Tpt:=Apt+(Fpt-Gpt);
```

$$Tpt := [\frac{-c2\ d1 + c1\ d2}{c2 - d2} - \frac{c2\ d1}{-c2\ d1 - d2 + c1\ d2},\ -\frac{c2\ d2}{-c2\ d1 - d2 + c1\ d2}]$$

Now we calculate P and Q, the points of intersection of this parallel with the lines BC and MF

```
Ppt:=intsect(Bpt,Cpt,Apt,Tpt);
```

$$\begin{aligned} Ppt := [&(d1^2\ c2^2 - c1\ d2^2 + c1^2\ d2^2 - 2\ d1\ c2\ c1\ d2 + 2\ d2\ c2\ d1 - c2^2\ d1)/(c1^2\ d2^2 \\ &- 2\ c1\ d2^2 + c2\ c1\ d2 - 2\ d1\ c2\ c1\ d2 + d2^2 - c2\ d2 + 2\ d2\ c2\ d1 + d1^2\ c2^2 - c2^2\ d1), -c2 \\ &d2\ (c2 - d2)/(c1^2\ d2^2 - 2\ c1\ d2^2 + c2\ c1\ d2 - 2\ d1\ c2\ c1\ d2 + d2^2 - c2\ d2 + 2\ d2\ c2\ d1 \\ &+ d1^2\ c2^2 - c2^2\ d1)] \end{aligned}$$

```
Qpt:=intsect(Mpt,Fpt,Apt,Tpt);
```

$$\begin{aligned} Qpt := [&\frac{1}{2}\ \frac{d1^2\ c2^2 - c1\ d2^2 + c1^2\ d2^2 - 2\ d1\ c2\ c1\ d2 + 2\ d2\ c2\ d1 - c2^2\ d1}{(-c2\ d1 - d2 + c1\ d2)\ (c1\ d2 - d2 - c2\ d1 + c2)}, \\ &-\frac{1}{2}\ \frac{c2\ d2\ (c2 - d2)}{(-c2\ d1 - d2 + c1\ d2)\ (c1\ d2 - d2 - c2\ d1 + c2)}] \end{aligned}$$

The theorem says that the distance from P to Q is equal to the distance from Q to A. Since we have set A to be the origin this is equivalent to saying that the distance from P to the origin is twice the distance from Q to the origin which is equivalent to "ratsq" the ratio of the squares being 4.

```
ratsq:=(Ppt[1]^2+Ppt[2]^2)/(Qpt[1]^2+Qpt[2]^2
);
```

$$ratsq := ((d1^2\, c2^2 - c1\, d2^2 + c1^2\, d2^2 - 2\, d1\, c2\, c1\, d2 + 2\, d2\, c2\, d1 - c2^2\, d1)^2 \Big/ (c1^2\, d2^2 - 2\, c1\, d2^2 + c2\, c1\, d2 - 2\, d1\, c2\, c1\, d2 + d2^2 - c2\, d2 + 2\, d2\, c2\, d1 + d1^2\, c2^2 - c2^2\, d1)^2 + c2^2\, d2^2\, (c2 - d2)^2 \Big/ (c1^2\, d2^2 - 2\, c1\, d2^2 + c2\, c1\, d2 - 2\, d1\, c2\, c1\, d2 + d2^2 - c2\, d2 + 2\, d2\, c2\, d1 + d1^2\, c2^2 - c2^2\, d1)^2) \Big/ (\frac{1}{4} \frac{(d1^2\, c2^2 - c1\, d2^2 + c1^2\, d2^2 - 2\, d1\, c2\, c1\, d2 + 2\, d2\, c2\, d1 - c2^2\, d1)^2}{(-c2\, d1 - d2 + c1\, d2)^2\, (c1\, d2 - d2 - c2\, d1 + c2)^2} + \frac{1}{4} \frac{c2^2\, d2^2\, (c2 - d2)^2}{(-c2\, d1 - d2 + c1\, d2)^2\, (c1\, d2 - d2 - c2\, d1 + c2)^2})$$

```
simplify(expand(ratsq));
```

$$4$$

This completes the proof.

15.3 EXERCISES

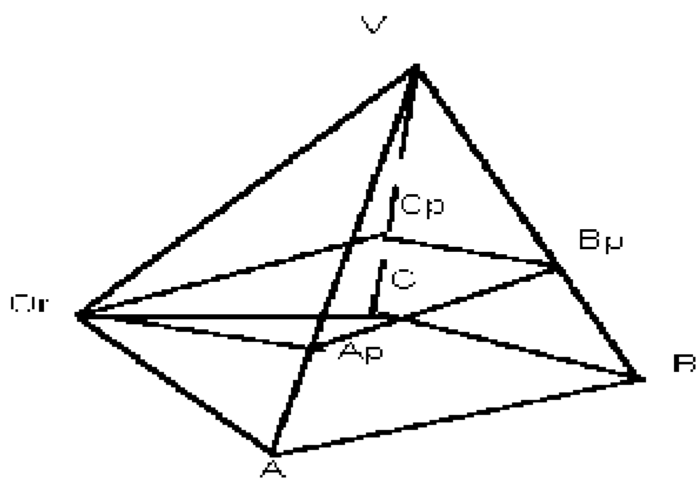
15.3.1 *EXERCISE: Explain why statement of the harmonic ray theorem if F and G are points at infinity the theorem should be interpreted as assuming that the quadrilateral is a rectangle. Prove the theoem in this case (Maple is not required)*

15.3.2 *EXERCISE: Interpret the harmonic ray theorem when G is a point at infinity and F is not. Prove the theorem in this case (Maple is not required)*

15.3.3 *EXERCISE: Check the necessity of the hypothesis that L be parallel to the line FG by doing the calculations for the harmonic ray theorem in a case when L is not parallel to the line FG.*

16 Cross sections of quadrilateral pryamids

16.0.4 Question: *Given two convex quadrilaterals, is there a pyramid having each of these as a cross section?*



Analysis: Let VOrABC be an arbitrary quadrilateral pyramid, with base OrABC in the xy-plane and vertex V. By translation and scaling we can assume that Or is the origin and $A = [1, 0, 0]$. So the three points B, C, and V determine the pyramid.

```
Or := [0,0,0]:  A := [1,0,0]:  B := [b1,b2,0]:
C:= [c1,c2,0]:  V := [v1,v2,v3]:
```

Now any cross-section of the pyramid is similar, by parallel translation, to one containing the origin Or. So we can fix one point of the square to be Or. Call the other vertices of the cross-section Ap, Bp, and Cp as shown in the diagram above. Since Ap is on the line through A and V we can write Ap = (1-r)A + rV, for some number r. Similarly, Bp = (1-s)B + sV and Cp = (1-t)C + tV for some numbers s and t.

```
Ap := expand((1-r)*A + r*V):
Bp := expand((1-s)*B + s*V ):
Cp := expand((1-t)*C + t*V):
```

Since Or, Ap, Bp, and Cp all lie in the cross sectioning plane, and Ap and Cp are independent, we also know that Bp is a linear combination of Ap and Cp, say $Bp = s1\ Ap + s2\ Cp$ for some numbers s1 and s2.

```
eqn := expand(Bp-(s1*Ap+s2*Cp));
```

$$\begin{aligned} eqn := [&-v1\ t\ s2 - c1\ s2 + c1\ t\ s2 - v1\ r\ s1 - s1 + r\ s1 + v1\ s + [\\ &[8.888888889 - 8.888888889\ s,\ 8.888888889 - 8.888888889\ s],\\ &[-.0999999994 + .0999999994\ s,\ -1.000000001 + 1.000000001\ s]],\ -v2\ t\ s2 - c2\ s2\\ &+ c2\ t\ s2 - v2\ r\ s1 + v2\ s + [[8.000000006 - 8.000000006\ s,\ 0],\\ &[-.0999999994 + .0999999994\ s,\ 1.000000001 - 1.000000001\ s]],\\ &-v3\ t\ s2 - v3\ r\ s1 + v3\ s] \end{aligned}$$

There are three (linear) equations in s, s1, and s2. These can be solved directly with 'solve'.

```
sol:=solve({op(eqn)},{s1,s2,s});
```

$$sol := \{s2 = \frac{b2\,(-1+r)}{\%1},\ s1 = \frac{b2\,c1 - c1\,t\,b2 - b1\,c2 + b1\,c2\,t}{\%1},$$
$$s = \frac{-b2\,t + r\,b2\,t + c1\,b2\,r - c1\,t\,b2\,r - b1\,r\,c2 + b1\,r\,c2\,t}{\%1}\}$$
$$\%1 := c1\,b2\,r - c1\,t\,b2\,r - c2 + c2\,t - b2\,t + r\,c2 - r\,c2\,t + r\,b2\,t - b1\,r\,c2 + b1\,r\,c2\,t$$

This shows that s, s1, and s2 are functions of r and t, given that b1, b2, c1, and c2 are constants. Note that v1 and v2 do not appear in the expressions for s, s1, and s2 given in sol.

```
indets(eqn),indets(sol);
```

$$\{b1,\ b2,\ c1,\ c2,\ v1,\ v2,\ r,\ s,\ t,\ s1,\ s2,\ v3\},\ \{b1,\ b2,\ c1,\ c2,\ r,\ s,\ t,\ s1,\ s2\}$$

Also, notice that there is a denominator %1 in sol. When $\%1 = 0$, the system has no solution.

We are left with 4 variables, v1, v2, r, and t. The idea is we want to choose values for these so that Or Ap Bp Cp is a square. What are the equations which force Or Ap Bp Cp to be a square? First it must be the case that $\boldsymbol{Bp} = \boldsymbol{Ap} + \boldsymbol{Cp}$, that is $s1 = 1$ and $s2 = 1$.

```
eq1 := s1 =1;
eq2 := s2 = 1;
```

$$eq1 := s1 = 1$$

$$eq2 := s2 = 1$$

These equations taken together say that Or Ap Bp Cp is a parallelogram. To get a square, we need two other conditions: Or Ap and Or Ap have the same length (equation 3) and are perpendicular (equation 4).

```
eq3 := Ap[1]^2+Ap[2]^2=Bp[1]^2+Bp[2]^2;
```

$$eq3 := (v1\,r + 1 - r)^2 + v2^2\,r^2 = (v1\,s + b1 - b1\,s)^2 + (v2\,s + b2 - b2\,s)^2$$

```
eq4 := Ap[1]*Bp[1]+Ap[2]*Bp[2] ;
```

$$eq4 := (v1\,r + 1 - r)\,(v1\,s + b1 - b1\,s) + v2\,r\,(v2\,s + b2 - b2\,s)$$

Now solve these 4 equations for the 4 variables r, t, v1 and v2 in terms of b1,b2, c1, and c2.

```
sln
:=solve(subs(sol,{eq1,eq2,eq3,eq4}),{v1,v2,r,t});
```

$$sln := \{r = \frac{-c2 + b2}{\%1},\ v1 = -($$
$$b1^2\,c2^2 - 3\,b1\,c2^2 - c2\,c1\,b2\,b1 + 2\,b1\,c2\,b2 - c2^2\,b2 + c2\,c1\,b2 + c2\,b2^2 - b2^2\,c1)/(\%2)$$
$$,\ t = \frac{-c2 - b2\,c1 + b1\,c2}{\%1},\ v2 = -(2\,c2\,c1\,b2 + c1^2\,b2^2 - b2^2\,c1 - b1\,c2^2 + b1^2\,c2^2$$
$$+ c2\,b2^2 - 2\,c2^2\,b2 + b1\,c2^2\,b2 - b2^2\,c1\,c2 - 2\,c2\,c1\,b2\,b1)/(\%2)\},\{v2 = (b2^2\,c1\,c2$$
$$- c2\,b2^2 - b2^2\,c1 + c1^2\,b2^2 - b1\,c2^2\,b2 + 2\,c2^2\,b2 - 2\,c2\,c1\,b2\,b1 + 2\,c2\,c1\,b2 - b1\,c2^2$$
$$+ b1^2\,c2^2)/(\%2),\ r = \frac{-c2 + b2}{\%1},\ v1 = -($$

$$b1^2\,c2^2 - 3\,b1\,c2^2 + 2\,b1\,c2\,b2 - c2\,c1\,b2\,b1 + c2^2\,b2 + c2\,c1\,b2 - c2\,b2^2 - b2^2\,c1)/(\%2)$$
$$,\ t = \frac{-c2 - b2\,c1 + b1\,c2}{\%1}\}$$
$$\%1 := -b2\,c1 - c2 + b2 + b1\,c2$$
$$\%2 := c1^2\,b2^2 + 4\,c2\,c1\,b2 - 2\,c2\,c1\,b2\,b1 - 2\,b2^2\,c1 - 6\,c2\,b2 + 2\,b1\,c2\,b2 + 5\,c2^2$$
$$- 4\,b1\,c2^2 + b1^2\,c2^2 + 2\,b2^2$$

```
indets([sln]);
```

$$\{b1,\ b2,\ c1,\ c2,\ v1,\ v2,\ r,\ t\}$$

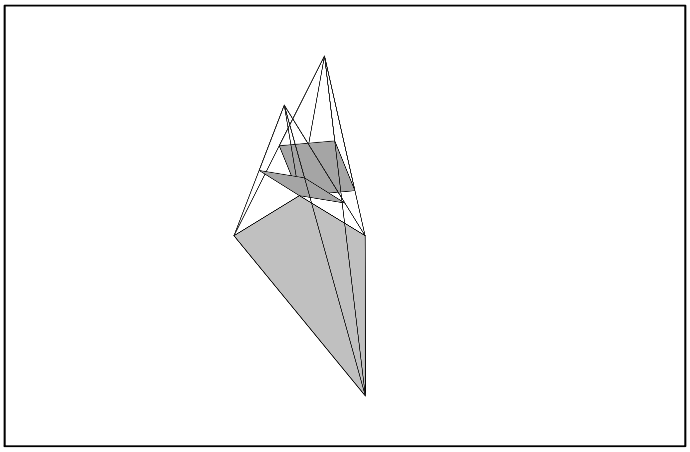
There are two solutions to this system. Each solution contains two denominators, %1 and %2. When either of these is 0, the system of equations does not have a solution.

Here is a word which draws a picture of the square cross section of the pyramid with base Or A B C.

```
cross_section := proc(h,x,y,z,w,i)
local vars, vars2, vert, quad2, quad1, rays;
vars :=subs({b1=x,b2=y,c1=z,c2=w},sln[i]);
vars2 := subs({b1=x,b2=y,c1=z,c2=w,op(vars)},sol);
vert :=
subs([v3=h,b1=x,b2=y,c1=z,c2=w,op(vars2),op(vars)],[v1,v2,v3]);
quad2 :=plots[polygonplot3d](
subs([v3=h,b1=x,b2=y,c1=z,c2=w,op(vars2),op(vars)],
[Or,Ap, s1*Ap+s2*Cp,Cp]),style=patch,color=tan,scaling=constrained);
quad1 :=plots[polygonplot3d](
subs([v3=h,b1=x,b2=y,c1=z,c2=w,op(vars2),op(vars)],
[Or ,A,B,C]),style=patch,color=grey,scaling= constrained);
rays := plots[polygonplot3d](
subs([v3=h,b1=x,b2=y,c1=z,c2=w,op(vars2),op(vars)],
{[[0,0,0],vert],[A,vert],[Ap,vert],[B,vert],[Bp,vert],[C,vert],[Cp,ve
rt]}),
thickness=2,color=blue);
plots[display]([rays,quad1,quad2]);
end:
```

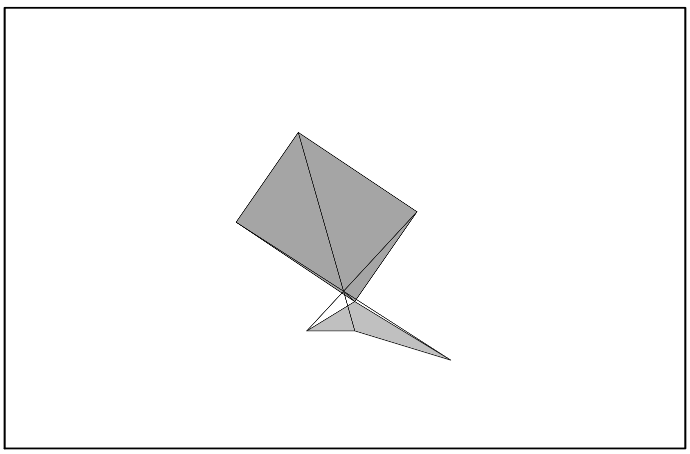
Here is a display showing that for a given height and base, there are two distinct square cross sections.

```
plots[display]([cross_section(1,2,3,0,1,1),
cross_section(1,2,3,0,1,2)]);
```



This picture shows that when the quadrilateral is not convex, the square has three vertices on the reflection of the pyramid through its vertex.

```
cross_section(1,1/2,1/2,0,2,1);
```



16.1 Exercises

16.1.1 *Exercise: Solve these equations for s, s1, and s2 by elimination. That is, solve eqn[1] for s and substitute that into eqn[2]. Then solve eqn[2] for s1 and substitute that and the value for s into eqn[3]. Then solve eqn[3] for s2, and substitute that into the values for s and s2. Check your result by substituting that back into the original equations.*

16.1.1 Solution.

First solve eqn[1] for s.

```
ssln := solve(eqn[1],s);
```

$$ssln := \frac{-v1\,t\,s2 - c1\,s2 + c1\,t\,s2 - v1\,r\,s1 - s1 + r\,s1 + b1}{-v1 + b1}$$

Then substitute that value for s into eqn[2] and solve the result for s1.

```
s1sln := solve(subs( s=ssln,eqn[2]),s1);
```

$$s1sln := (-v2\,t\,s2\,b1 + c2\,s2\,v1 - c2\,s2\,b1 - c2\,t\,s2\,v1 + c2\,t\,s2\,b1 + b2\,v1\,t\,s2 + b2\,c1\,s2 - b2\,c1\,t\,s2 - v2\,c1\,s2 + v2\,c1\,t\,s2 + v2\,b1 - b2\,v1)/(v2 - v2\,r + v2\,r\,b1 - b2\,v1\,r - b2 + b2\,r)$$

Then substitute that value for s1 into

```
ssln := simplify(subs(s1=s1sln,ssln));
```

$$ssln := (v2\,r\,b1 - c1\,s2\,v2\,r - b2\,v1\,r + r\,c2\,s2\,v1 + v2\,t\,s2 - c2\,t\,s2 + c2\,s2 + r\,c2\,t\,s2 + c1\,t\,s2\,v2\,r - r\,v2\,t\,s2 - r\,c2\,t\,s2\,v1 - r\,c2\,s2 + b2\,r - b2)/(v2 - v2\,r + v2\,r\,b1 - b2\,v1\,r - b2 + b2\,r)$$

```
s2sln :=
solve(subs({s=ssln,s1=s1sln},eqn[3]),s2);
```

$$s2sln := \frac{b2\,(-1 + r)}{c1\,b2\,r - c1\,t\,b2\,r - c2 + c2\,t - b2\,t + r\,c2 - r\,c2\,t + r\,b2\,t - b1\,r\,c2 + b1\,r\,c2\,t}$$

```
ssln := simplify(subs(s2=s2sln ,ssln));
```

$$ssln := \frac{-b2\,t + r\,b2\,t + c1\,b2\,r - c1\,t\,b2\,r - b1\,r\,c2 + b1\,r\,c2\,t}{c1\,b2\,r - c1\,t\,b2\,r - c2 + c2\,t - b2\,t + r\,c2 - r\,c2\,t + r\,b2\,t - b1\,r\,c2 + b1\,r\,c2\,t}$$

```
s1sln := simplify(subs(s2=s2sln,s1sln));
```

$$s1sln := \frac{b2\,c1 - c1\,t\,b2 - b1\,c2 + b1\,c2\,t}{c1\,b2\,r - c1\,t\,b2\,r - c2 + c2\,t - b2\,t + r\,c2 - r\,c2\,t + r\,b2\,t - b1\,r\,c2 + b1\,r\,c2\,t}$$

Checking, we see the equations are satisfied.

```
simplify(subs({s=ssln,s1=s1sln,s2=s2sln},eq
n));
```

$$[0, 0, 0]$$

16.1.2 *Question: What geometrical conditions would result in %1 being zero?*

16.1.2 Partial Solution

This question always comes up when we solve a set of equations which arise from a geometrical problem. It is not always easy to 'see' the geometry in the algebra. What one can do, quite independently of the solution to the equations, is go back to the original problem and envisage the conditions under which the problem becomes degenerate or nonsensical.

First, turn the denominator into a function.

```
f := unapply(%1,r,t,b1,b2,c1,c2);
```

$$f := (r, t, b1, b2, c1, c2) \to c1\,b2\,r - c1\,t\,b2\,r - c2 + c2\,t - b2\,t + r\,c2 - r\,c2\,t + r\,b2\,t - b1\,r\,c2 + b1\,r\,c2\,t$$

If Ap = V = Cp then r = t = 1 and

```
f(1,1,b1,b2,c1,c2);
```

$$0$$

Now, examine the case where B = C.

```
f(r,t,b1,b2,b1,b2);
```

$$-b2 + b2\ r$$

In this case, %1=0 if r=1 or b2 = 0. That is, if Ap = V or B is on the line through A and Or. If B and C are collinear with A and Or, then c2=b2=0 and

```
f(r,t,b1,0,c1,0);
```

$$0$$

If B is on the line through A and C, then B = wA+(1-w)C.

```
Bs := expand(w*A+(1-w)*C ) ;
```

$$Bs := [c1 - c1\ w + w,\ c2 - c2\ w,\ 0]$$

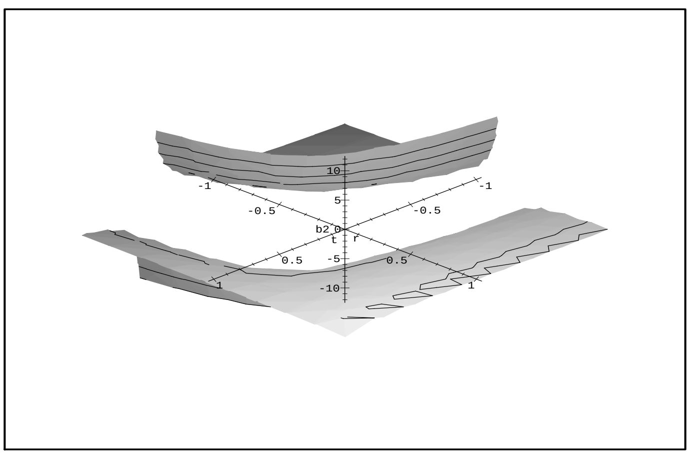
```
expand(f(r,t,Bs[1],Bs[2],c1,c2));
```

$$-c2 + t\ c2\ w + r\ c2 - r\ c2\ w$$

Here %1 is 0 if c2 =0 or if t=0 and r=w=1.

We can check graphically to see that %1 is not 0 over normal ranges for r and t (both less than 1, provided C and B are chosen so that Or A B C is a quadrilateral.

```
plots[implicitplot3d](f(r,t,.5,b2,-.5,1),r=-1
..1,t=-1..1,b2=-12..12,style=patchcontour,axes=normal);
```



16.1.3 *Question. What geometrical conditions would result in %1 or %2 being zero?*

16.1.3 Work on solution

```
%1;%2;
```

$$-b2\ c1 - c2 + b2 + b1\ c2$$

$$b1^2\,c2^2 - 4\,b1\,c2^2 - 2\,c2\,c1\,b2\,b1 + 2\,b1\,c2\,b2 + 5\,c2^2 - 6\,c2\,b2 + 4\,c2\,c1\,b2 + c1^2\,b2^2 - 2\,b2^2\,c1 + 2\,b2^2$$

If B is on the line through A and C, %1 is zero.

```
Bs :=
solve({op(expand(B-w*A-(1-w)*C))},{b1,b2});
```

$$Bs := \{b1 = c1 - c1\,w + w,\ b2 = c2 - c2\,w\}$$

```
simplify(subs(Bs,%1));
```

$$0$$

```
Bs :=
solve({op(expand(B-w*A-(1-w)*C))},{b1,b2});
```

If

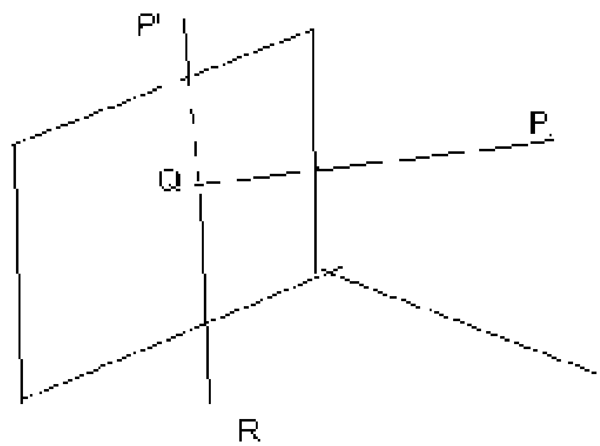
17 Mirrors: More Problems from Nature

17.1 Discussion.

What do you see when you look at something in a mirror? You see the mirror image of it! Mathematically, the object is reflected in the plane of the mirror. It is an interesting problem to draw the reflection of an object in a Maple worksheet. Just how would you do that? For example, in the diagram below, P is a point on the object and R is your eye. How do you determine the point Q that your eye sees when you look at the reflection of P in the mirror? The answer is that Q is the point on the mirror which also lies on the line from R to P', the reflection of P through the mirror. P' is defined mathematically as the point on the opposite side of the mirror such that the segment PP' is perpedicular to the mirror and is bisected by the mirror. So the point Q is a convex combination of P' and R. The equation is $\boldsymbol{Q} = s\,\boldsymbol{R} + (1-s)\,\boldsymbol{P}'$. If we take the plane of the mirror to be the xz-plane (y=0), then we can determine s by solving the equation in the second coordinate (indicated by a "2" subscript).

$$0 = \boldsymbol{Q}_2 = s\,\boldsymbol{R}_2 + (1-s)\,\boldsymbol{P}_2{}' \text{ or } s = \frac{-\boldsymbol{P}_2{}'}{\boldsymbol{R}_2 - \boldsymbol{P}_2{}'}$$

Since $\boldsymbol{P}_2{}' = -\boldsymbol{P}_2$, the reflection of $\boldsymbol{P}_2$ in the yz plane, we have that $s = \frac{\boldsymbol{P}_2}{\boldsymbol{R}_2 + \boldsymbol{P}_2}$.

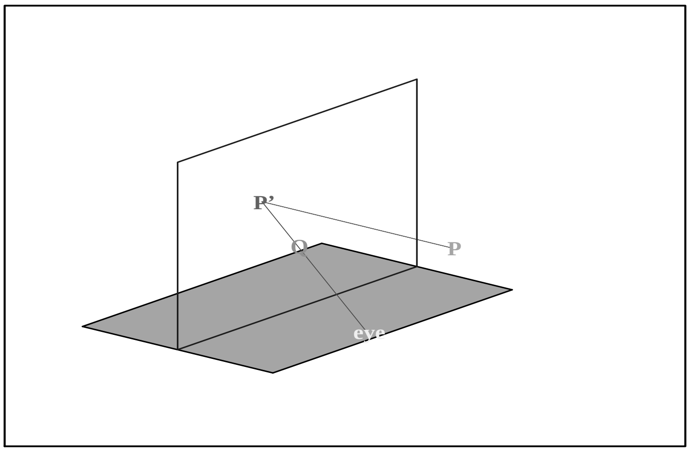


17.1.1 Diagram "light1"

```
miror:=plots[polygonplot3d]([[-2,0,0],[2,0,0]
,[2,0,3],[-2,0,3]],
style=wireframe,thickness=3,color=blue):
flor:=plots[polygonplot3d]([[-2,-1,0],[2,-1,0],[2,1,0],[-2,1,0]],
style=patch,thickness=3,color=tan):
pointP:=plots[textplot3d]([-1,1,1,` P`], font=[TIMES,BOLD,24]):
refP:=plots[textplot3d]([-1,-1,1,` P'`],
font=[TIMES,BOLD,24]):plots[display]([miror,flor,pointP,refP]):
eye:=plots[textplot3d]([2,2,1,` eye`], font=[TIMES,BOLD,24]):
P_refP:=plots[polygonplot3d]([[-1,1,1],[-1,-1,1]],
style=wireframe,color=red):
```

```
eye_refP:=plots[polygonplot3d]([[2,2,1],[-1,-1,1]],
style=wireframe,color=red):
pointQ:=plots[textplot3d]([0,0,1,` Q`], font=[TIMES,BOLD,24]):
light1:=plots[display]([miror,flor,pointP,refP,eye,P_refP,
eye_refP,pointQ],orientation=[40,60]):
```

```
light1;
```



We can define a word 'image' to return this point Q given the points P and R.

```
image := proc(p,r)
local s,q,refp;
s := p[2]/(r[2]+p[2]);
refp :=[p[1],-p[2],p[3]];
q := s*r+(1-s)*refp;
end:
```

Test this word out by taking p to be at [0,2,1] and r at [1,1,1]. In this case, the reflection of p through the xz plane is refp = [0,-2,1], the scalar s is 2/(1+2), and so the image of p on the mirror y=0 when the observer r is at [1,1,1] is 2/3*[1,1,1]+ 1/3*[0,-2,1] = [2/3, 0, 1].

```
p := [0,2,1]; r:=[1,1,1];
q := image(p,r);
```

$$p := [0,\ 2,\ 1]$$

$$r := [1,\ 1,\ 1]$$

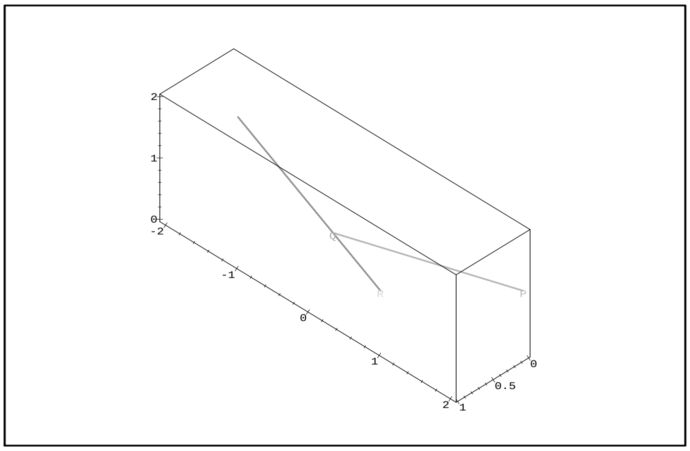
$$q := [\frac{2}{3},\ 0,\ 1]$$

That's right. Now we can draw the picture.

```
pl1 :=
plots[polygonplot3d]({[p,q],[r,[p[1],-p[2],p[3]]]},
thickness=3,scaling=constrained,axes=boxed):
```

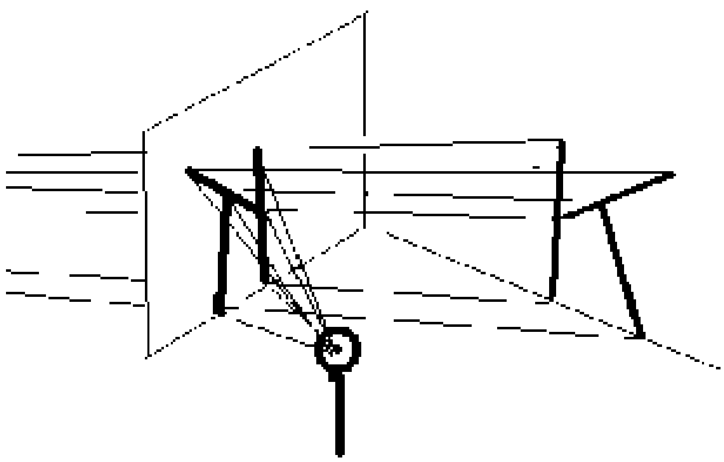
```
pl2 :=plots[textplot3d]({[op(q),`Q`],
[op(r),`R`],[op(p),`P`]},
align=BELOW):
```

```
plots[display]([pl1,pl2]);
```



17.1.1 *Problem: Draw the letter K and its reflection in a mirror.*

For the sake of simplicity, let us assume that the letter K is standing in the yz plane on the y-axis, with one base at say [0,1,0] and the other at [0,2,0]. Our mirror will be a square in the xz plane with base [[0,0,0],[2,0,0]]. We, the observer will be standing at a point (a,b,c) to be determined later. The picture looks like this:



First define the mirror and the letter k.

```
mir :=
[[0,0,0],[2,0,0],[2,0,2],[0,0,2]];
```

$$mir := [[0, 0, 0], [2, 0, 0], [2, 0, 2], [0, 0, 2]]$$

```
mirror :=
plots[polygonplot3d](mir,style=patch,color=gray):

kay :=
[[0,1,0],[0,1,1]],[[0,1,.5],[0,1.5,1]],
[[0,1,.5],[0,1.5,0]];
```

$$kay := [[0, 1, 0], [0, 1, 1]], [[0, 1, .5], [0, 1.5, 1]], [[0, 1, .5], [0, 1.5, 0]]$$

```
ka
:=plots[polygonplot3d]({kay},thickness=3,color=red):
```

Now the image of the letter k in the mirror is a function of the position (a,b,c) of the observer.

```
refkay := proc(a,b,c)
local i;
seq(map(image,kay[i],[a,b,c]),i=1..nops([kay]));
end:
```

So let's pick an observation point ob

```
ob := [2,3,1];
```

$$ob := [2, 3, 1]$$

obs is a blue stick located at the observation point.

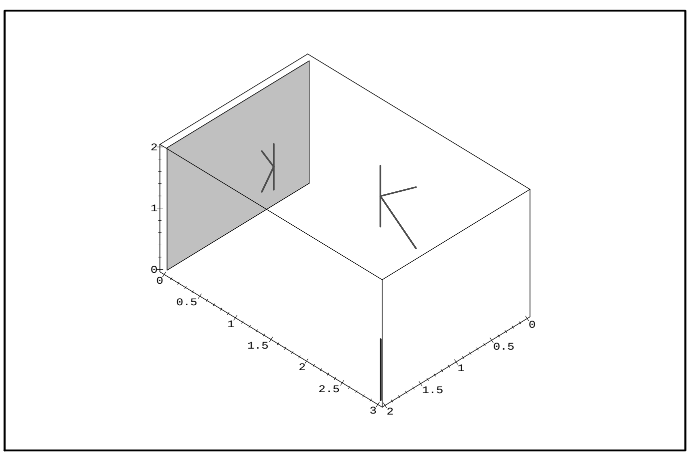
```
obs :=
plots[polygonplot3d]([ob,[ob[1],ob[2],0]],color=blue,thickness=3):
```

reka is the image of the letter k in the mirror from the observation point ob.

```
reka
:=plots[polygonplot3d]({refkay(op(ob))},thickness=3,
color=red):
```

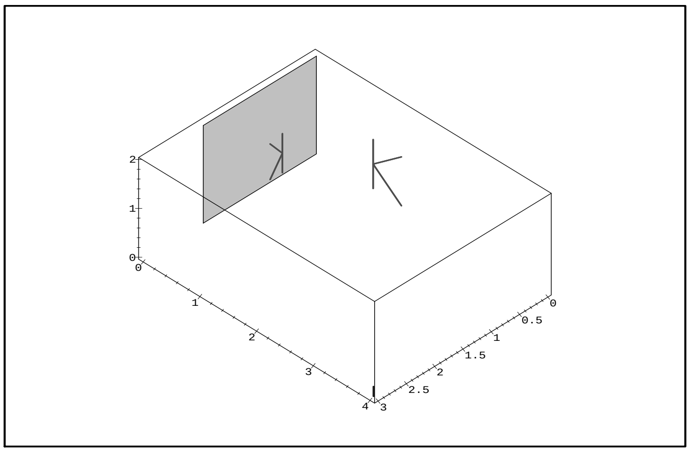
We can see what this looks like with display.

```
plots[display]([obs,mirror,ka,reka],scaling=c
onstrained,axes=boxed);
```



Looks fine. Copy down the input cells from above to make a word to draw the mirror, the letter, and its image in the mirror as seen from the observer.

```
mirimage := proc(ob)
local obs, reka;
obs :=
plots[polygonplot3d]([ob,[ob[1],ob[2],0]],color=blue,thickness=3):
reka :=plots[polygonplot3d]({refkay(op(ob))},thickness=3,color=red):
plots[display]([obs,mirror,ka,reka],scaling=constrained,axes=boxed);
end:

mirimage([3, 4,.2]);
```



17.1.2 *Exercise: Make a movie showing what happens to the reflection as we move the observer around in a circle .*

```
movie :=
[seq(mirimage([2+cos(j*Pi/5),2+sin(Pi*j/5),1]),j=1..10)]:

display(movie,insequence=true);
```

17.2 Questions:

17.2.1 *Question: Suppose we had another mirror in a vertical plane perpendicular with the first mirror with its base on the segment [[3,1,0],[3,3,0]]. If an observer at [2,2,1] looked in the second mirror, what would he see (besides himself)? Model this.*

17.2.2 *Question If you have two perfect mirrors, and hold them just right, you can see down an infinite corridor. Even with two good mirrors, you can see a large number of reflections of reflections. Observe this and describe what you see exactly. As an experiment, hold up a transparency with a letter k written on it. How hard would this be to model?*

17.2.3 *Question: If you have three mirrors which form a corner of a box and shine a beam of light almost into the corner, what happens to the beam?*

17.2.4 *Problem: Model a kaleidoscope.*

17.2.5 *Problem: Here's a two dimensional problem. Suppose you shine a light down a mirrored angle. How many times will it bounce before it comes out? (Or will it come out?).*

17.3 Some solutions

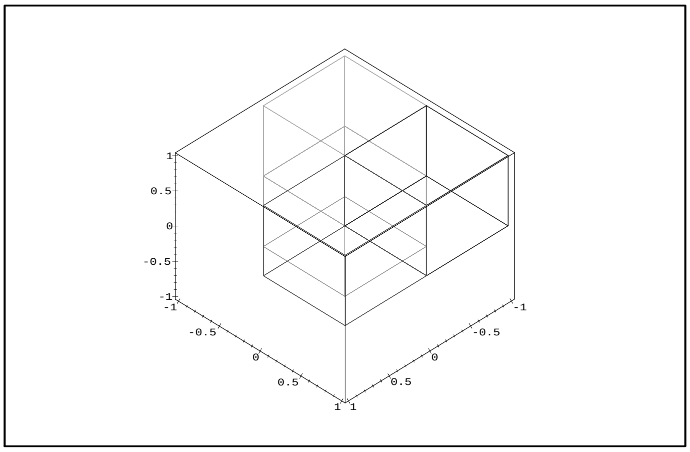
A solution to question 3.

If we shine the light from say [1,1,1] almost into the corner [0,0,0], then it will be reflected in some succession off of each coordinate plane, effecting a sign change in the corresponding component of its velocity vector with each bounce.. Upon the third bounce, its velocity will be the negative of its original velocity and so it will be moving parallel to its original direction but in the opposite direction. We can model this. First we need our cube word.

```
cube := proc (shift,clr)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift; v3 := [1, 1, 0]+shift;
v4 := [0, 1, 0]+shift; v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=clr,scaling=constrained,
style = wireframe,axes=boxed)
end:
```

We propose to construct the path of the light ray in the cube by first constructing the 'straightened out' path we would get if the ray continued on through the walls of the cube. This straight path, which could be called the 'virtual path' of the light ray, can then be fairly easily 'wound back up' by reflecting the points where the straight path crosses the coordinate planes into the original cube.

```
plots[display]([cube([0,0,0],red),cube([-1,0,
0],blue),
cube([-1,-1,0],tan),cube([-1,-1,-1],green)]);
```



So assume our light ray starts at $\boldsymbol{p} = [.9,\, 1,\, .9]$ (in the red cube) and heads toward $\boldsymbol{q} = [-1,\, -.5,\, -.25]$ (in the green cube). Along the way it will successively cross the xy, yz, and zx planes in some order. We can easily determine the crossing points and order of crossing by parameterizing the segment from p to q and solving three equations.

```
p := [9/10,1,9/10]:  q := [-1,-1/2,-1/4]:
```

Using t as the parameter, a general point r on the segment pq looks like

```
r := expand(t*q+(1-t)*p);
```

$$r := [\frac{9}{10} - \frac{19}{10}\,t,\, 1 - \frac{3}{2}\,t,\, \frac{9}{10} - \frac{23}{20}\,t]$$

where $t \,\varepsilon\, [0,\, 1]$. Now to get the crossing points of the yz, xz, and xy planes solve the equations.

```
tyz := solve(r[1]=0,t);
txz := solve(r[2]=0,t);
txy := solve(r[3]=0,t);
```

$$tyz := \frac{9}{19}$$

$$txz := \frac{2}{3}$$

$$txy := \frac{18}{23}$$

By inspection, we have tyz < txy < txz, so the yx, xy, and xz planes are crossed in that order. The crossing points are obtained by substitution.

```
c1 := subs(t=tyz,r);
c2 := subs(t=txz,r);
c3 := subs(t=txy,r);
```

$$c1 := [0, \frac{11}{38}, \frac{27}{76}]$$

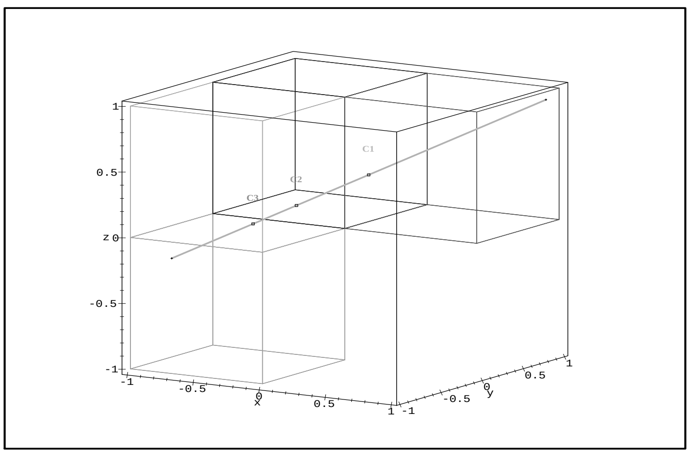
$$c2 := [\frac{-11}{30}, 0, \frac{2}{15}]$$

$$c3 := [\frac{-27}{46}, \frac{-4}{23}, 0]$$

In terms of the colored cubes, the virtual light ray traverses the red cube, then crosses the yz plane at c1 into the blue cube, traverses the blue cube, then crosses the xz plane at c2 into the tan cube, traverses that, and finally crosses the xy plane at c3 and goes into the green cube.

Here is a plot showing the segment pq and the three crossing points c1,c2, c3.

```
ptplt :=
plots[pointplot3d]({p,q},color=black,thickness=3):
seg := plots[polygonplot3d]([p,q],thickness=3):
pts:=plots[textplot3d]({[op(c1 +[0,0,.2]),`C1`],[op(c2
+[0,0,.2]),`C2`],[op(c3 +[0,0,.2]),`C3`]},font=[TIMES,BOLD,10]):
ptplt2 := plots[pointplot3d]([c1,c2,c3],color=blue,symbol=box):
plots[display]([pts,
ptplt2,ptplt,seg,cube([0,0,0],red),cube([-1,0,0],blue),
cube([-1,-1,0],tan),cube([-1,-1,-1],green)],labels=[x,y,z],orientation
=[-58,78]);
```



The actual light ray traverses the red cube, bounces off of the yz plane at c1, traverses the red cube again, then bounces off the xz plane at the reflection rc2 of c2 through the yz plane, traverses the red cube again, then bounces off the xy plane at the second reflection rrc3 of c3 first through the yz plane and then through the xz plane.

Finally, it strikes the wall x=1 of the red (original) cube at -q = [1,.5,.25] at t=1.

```
rc2 :=[-c2[1],c2[2],c2[3]];
rrc3 := [- c3[1],-c3[2],c3[3]];
mq := -q;
```

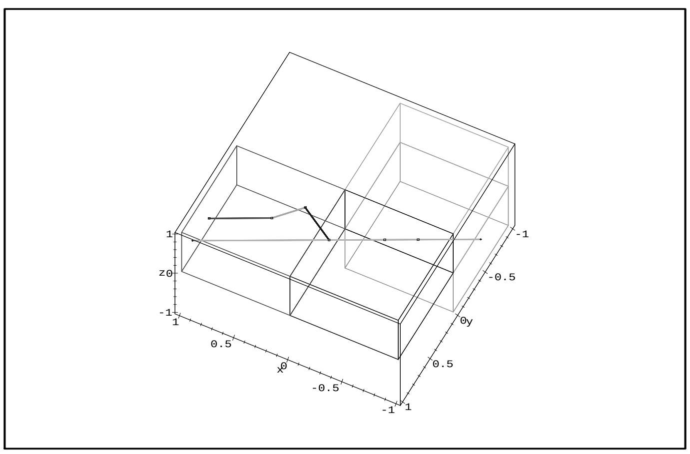
$$rc2 := [\frac{11}{30},\ 0,\ \frac{2}{15}]$$

$$rrc3 := [\frac{27}{46},\ \frac{4}{23},\ 0]$$

$$mq := [1,\ \frac{1}{2},\ \frac{1}{4}]$$

Here is a plot showing the actual path of the light ray in the cube and the virtual path..

```
seg2 :=
plots[polygonplot3d]([c1,rc2],thickness=3,color=blue):
seg3 := plots[polygonplot3d]([ rc2,rrc3], thickness=3, color=tan):
seg4 := plots[polygonplot3d]([rrc3,mq], thickness=3,color=red):
ptplt3 := plots[pointplot3d]([rc2,rrc3,mq],color=blue,symbol=box):
plots[display]([ptplt3,ptplt2,ptplt,seg,seg2,seg3,seg4,
cube([0,0,0],red),cube([-1,0,0],blue),cube([-1,-1,0],tan),
cube([-1,-1,-1],green)],labels=[x,y,z],orientation=[117,22]);
```



From within the red box the the ray bounces off the "far wall" to the left following the thick blue path, then to the "floor" along the tan path, then back along a path parallel to its original path of entry.

17.3.1 *Problem: Define a word cornerbounce(p,q) where p is a point on the red cube with exactly one coordinate 1 and no coordinate 0, and q is a point on the green cube with exactly one coordinate -1 and no coordinate 0, which returns a plot of both the virtual and actual paths of a beam of light from p to q.*

17.3.2 *Problem: Model the path of a light beam bouncing around in an enclosed cubical room.*

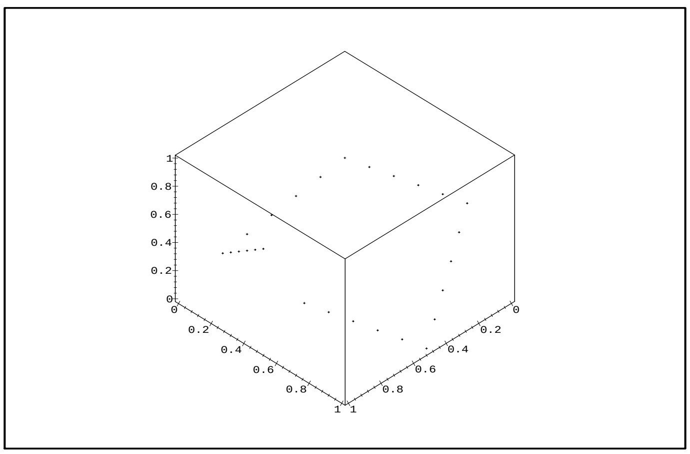
Analysis: We can continue with the 'virtual ray' idea that we were using before, but with a slightly different twist. Think of the virtual light beam as parameterized by the point $p + tv$ where p is the starting point, t is the (time) parameter, and v is the velocity vector. The room is the set of all points [x,y,z] such that each coordinate is between 0 and 1. A point q on the on the actual light beam is obtained from a point $p + t\,v$ on the virtual light beam as follows: for each coordinate i, if the integer part of $p_i + t\,v_i$ is even, then q_i is the fractional part of $p_i + t\,v_i$ and if the integer part is odd, then q_i is 1 - fractional part.

The word pathpt computes q from p, t, and v.

```
pathpt := proc(p,v,t)
local q,i,coord;
for i from 1 to 3 do
coord := p[i] + t*v[i];
if floor(coord) mod 2 = 0 then q[i] := frac(coord)
else q[i] := 1-frac(coord) fi od;
[q[1],q[2],q[3]]; end:


path := proc(p,v,s,n) local i;
[seq(pathpt( p,v,s*i),i=0..n)] end:
```

The word path constructs a list of points on the actual path, and the word drawpath uses spacecurve from the plots package to draw a picture of the path of the light beam in the cube.

```
drawpath := pth ->
plots[spacecurve](pth, color=black, style=point, scaling=constrained,
axes=boxed):


drawpath(
path([1/2,0,0],[.1,.05,.1],1,25));
```



```
movie := pth -> plots[display](
[seq(drawpath(pth[1..j]),
j=1..nops(pth))],insequence=true):
```

Here is a movie of the beam bouncing.

```
movie(path([1/2,0,0],[.1,.05,.1],1,50));
```

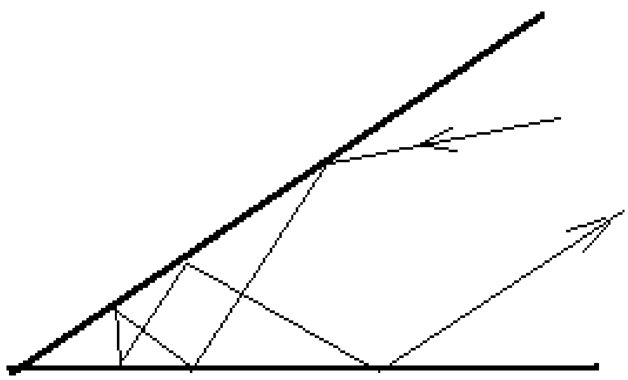
17.3.3 *Exercise: Draw a few paths in the cube. Look at them from the top of the cube by grabbing it, pulling it so that you only see the top of the cube, and choosing Redraw from the menu. Describe what you see. How does it relate to the problem of modeling the path of a light beam in a square?*

17.3.4 *Problem. A path in the cube is called a closed reflective path if after a finite number of bounces, the light beam begins to repeat its path in the cube. Find some closed reflective paths in the cube. Are there any which bounce off of each face of the cube at exactly one place?*

A discussion and solution(s) to question 5.

Recall the question: Suppose you shine a light down a mirrored angle. How many times will it bounce before it comes out? Or will it come out?

First we create a word "geombouncer" is written with the geometry package which will allow us to experiment with the question. "geombouncer(A,B,dir,maxbounces)" assumes that one is looking at the angle formed by two mirrors. Looking along their edges one sees the line segment OA where O is the origin and A is a specified point in the first quadrant, and a segment along the x-axis. The point P is the starting point of a ray that is directed in to the angle with direction "dir". "geombouncer" then follows the path of the ray as it reflects from the two mirrors as long as it remains to the left of the point A up to "maxbounces" reflections.



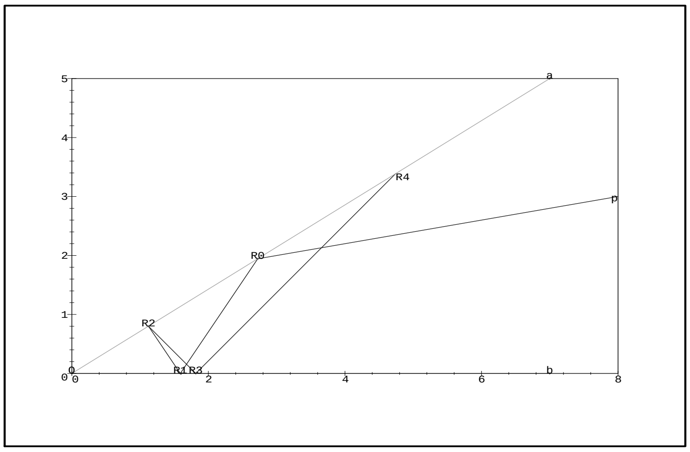
17.4 geombouncer instructions

```
geombouncer:=proc(A,P, dir,maxbounces)
local objlist,j:
geometry[point](O,[0,0]);
geometry[point](a,A):
geometry[point](c,P+dir):
geometry[point](b,[A[1],0]):
geometry[point](p,P):
geometry[line](topline,[O,a]):
geometry[segment](top,[O,a]):
geometry[line](baseline,[O,b]):
geometry[segment](base,[O,b]):
geometry[line](lnofpth0,[p,c]):
geometry[intersection](R0,lnofpth0,topline):
geometry[reflection](lnofpth1,lnofpth0,topline):
geometry[intersection](R1,lnofpth1,baseline):
geometry[dsegment](path0, [R0,R1]):
geometry[dsegment](initialpath, [p,R0]):
objlist:=NULL:
objlist:=R0,a,b,O,top(color=tan,thickness=2),base(color=tan,thickness=
2),initialpath(color=blue,thickness=2),path0(color=blue,thickness=2):
for j from 1 to maxbounces while (
geometry[coordinates](cat(R,j))[1]< A[1] and
geometry[coordinates](cat(R,j))[1]>0 ) do
geometry[draw]({objlist});
if (geometry[coordinates](cat(R,j))[2]<
geometry[coordinates](cat(R,j-1))[2]) then
geometry[reflection](cat(lnofpth,j+1),cat(lnofpth,j),baseline):
geometry[intersection](cat(R,j+1),cat(lnofpth,j+1),topline):
else
geometry[reflection](cat(lnofpth,j+1),cat(lnofpth,j),topline):
geometry[intersection](cat(R,j+1),cat(lnofpth,j+1),baseline):
fi:
geometry[dsegment](cat(path,j),[cat(R,j),cat(R,j+1)]):
```

```
if geometry[coordinates](cat(R,j+1))[1] >0 then
objlist:=objlist,cat(R,j),cat(path,j)(color=blue,thickness=2): fi:
od:
geometry[draw]({objlist},scaling=constrained);
end:
```

```
restart;
```

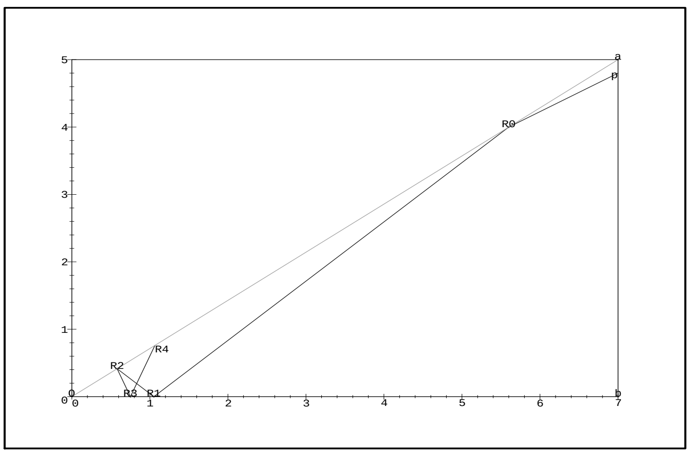
```
geombouncer([7,5],[8,3],[-1,-.2],10);
```



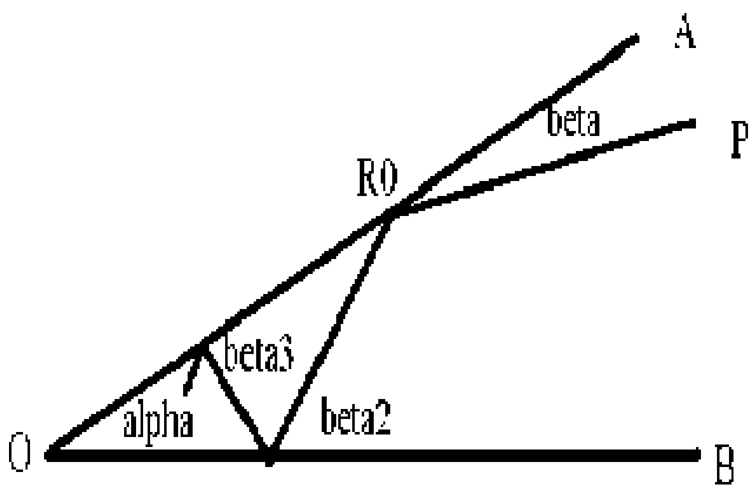
In the above example the ray emits from [8,3] in the direction [-1,-.2] and bounces five times before it leaves the mirrors.

17.4.1 *Exercise: Experiment with geombouncer to see if you can make the ray bounce a large number of times. Note that simply chosing a shallow angle won't necessarily produce a small number.*

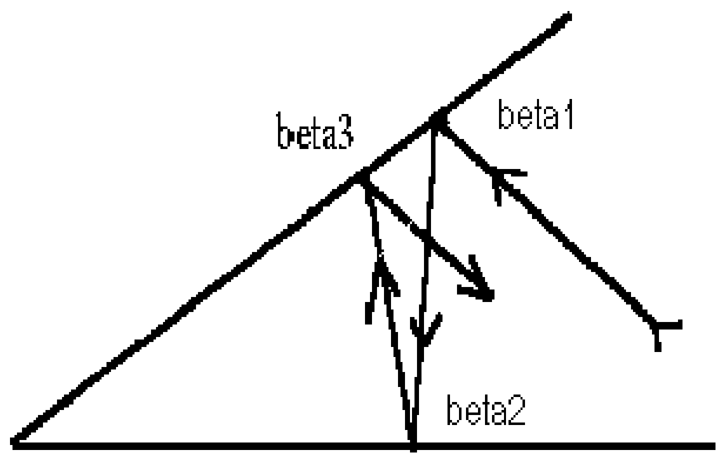
```
geombouncer([7,5],[7,4.8],[-7,-4],10);
```



To analyze this problem we will find it convenient to use the two angles α and β as indicated in the following diagram.

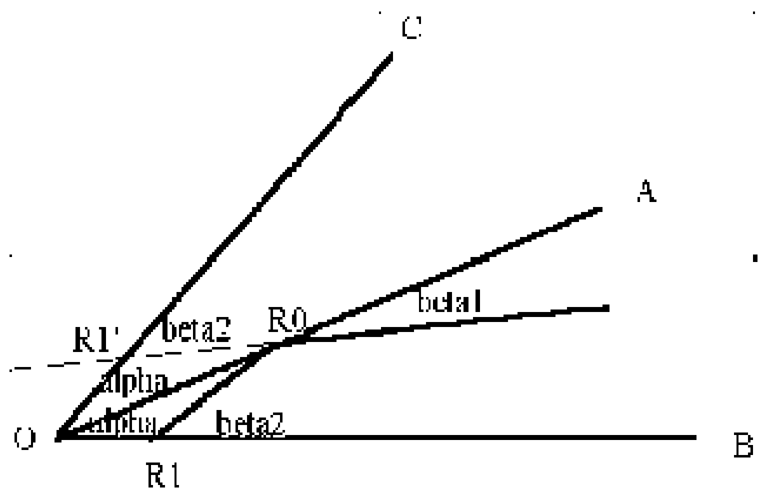


That is, α is the angle between the two mirrors (which we take to be acute) and
β is the angle of incidence of the ray upon the upper mirror. Actually, what we want to look at is the sequence of incidence angles
$\beta = \beta_1, \beta_2, \ldots$ The basic idea is that the angles β_i are increasing in size and as soon as β_j becomes greater than or equal to $\frac{\pi}{2}$ the ray will be on its way out of the region between the mirrors



Thus we need to see how to calculate β_i for each i. This is not hard to do if we determine the reflection of the angle rather than the angle itself. Consider the following diagram in which the angle AOB is reflected through the line AO to produce the angle AOC. The initial ray strikes AO at the point R0 and angle β_1 . It is reflected, hitting the segment AO at the point R1 at angle
β_2. If the original path were continued it would meet OC at a point R1'.

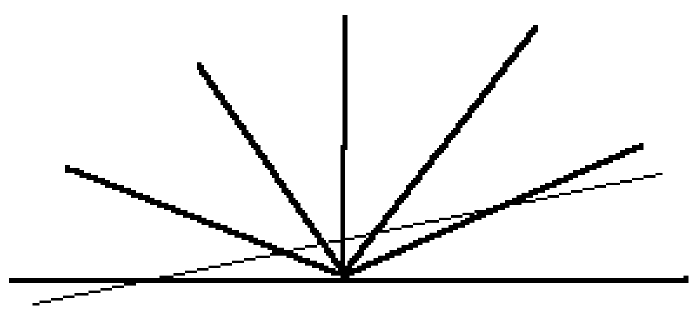
17.4.2 *Exercise: Show that, as indicated in the diagram, the angle R0R0'C is equal to* β_2 *and that* $\beta_2 = \beta_1 + \alpha$



17.4.3 *Exercise: Extend the previous argument by reflecting AOB first through AO and then through OC and use the same idea to show that* $\beta_3 = \beta_2 + \alpha$. *This* $\beta_3 = \beta_1 + 2\,\alpha$ *and in general* $\beta_n = \beta_1 + (n-1)\,\alpha$.

17.4.4 *Exercise: Argue that there are exactly as many bounces of the light beam between the two mirrors as there are points of intersection of the extended original path of the fan of rays with rays spaced exatly* α *apart opening counterclockwise from the x-axis.*

Show that the ray will start moving away from the center on that pth bounce where p is the smallest integer such that $\frac{\pi}{2} \leq \beta + (p-1)\,\alpha$



17.4.5 *Exercise: Show that there will be q bounces where q is the largest integer such that* $q\,\alpha < \pi + \beta$

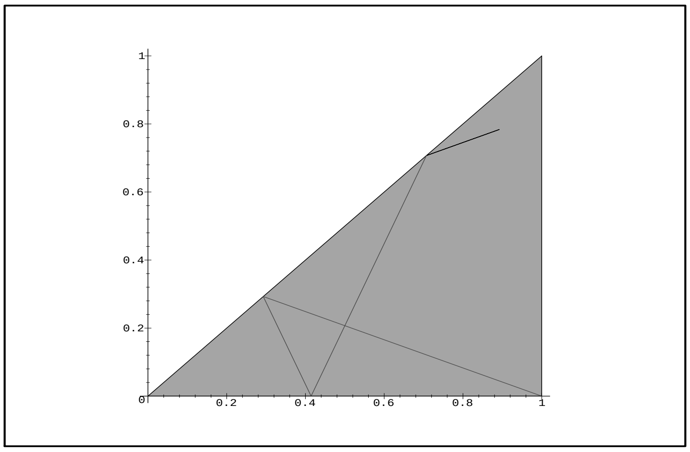
17.4.6 *Exercise: Show that the light will exactly retrace its enrty route to leave if and only if* $\frac{\frac{\pi}{2}-\beta}{\alpha}$ *is a non-negative integer.*

17.4.7 *Exercise: If two mirrors meet at a 30 degree angle what is the maximum possible number of reflections off of the mirrors of a light ray which is incident upon one of them?*

17.4.8 *Exercise: Adapt the instructions in the "geombouncer" word to produce an animation of the ray following its path rather than a static image of the final path.*

We used the geometry package for our diagrams earlier but there is no reason for our not writing a "bouncer" word ourselves. The following word "bouncer" takes in angles alpha and beta where theta is the angle between two lines and alpha is an angle of at most alpha which an incident ray makes with one of the legs of the angle. The procedure returns a plot of the path of the ray as it enters and bounces out of the angle between the two lines. It is followed by an an alternate discussion of the calculation of the numnber of intersections, paths that retrace themselves, etc.

```
bouncer:=proc(alpha, beta)
local
STARTPT,n,w,line,nalphaline,eqn,eqns,PTS,leg1,leg2,refpts,i,N,slns;
STARTPT:=evalf([cos(alpha),sin(alpha)]);
line:=evalf(expand(STARTPT+t*[ cos(alpha-beta), sin(alpha-beta)]));
nalphaline:=s*[cos(n*alpha) ,sin(n*alpha)];
eqn:=expand(line-nalphaline);
eqns:={eqn[1]=0,eqn[2]=0};
PTS:=NULL;
N:= floor((Pi-beta)/alpha):
if frac((Pi-beta)/(alpha))=0 then N:=N-1 fi;
for w from 1 to N+1 do
slns:=fsolve(subs(n=w,eqns),{s,t}):
PTS:=PTS,subs(slns, line):
od:  PTS:=[PTS];
leg1:=evalf([cos(alpha),sin(alpha)]);leg2:=[1,0];
refpts:=NULL:
for i from 1 to nops(PTS) do
if i mod 2 = 1 then refpts:=refpts,linalg[norm](PTS[i],2)*leg1; else
refpts:=refpts, linalg[norm](PTS[i],2)*leg2 fi; od:
plots[display](
[plots[polygonplot]([[0,0],[1,0],[1,tan(alpha)]],style=patch,color=tan
), plot([refpts]),
plots[polygonplot]([STARTPT,subs(t=.2,line),STARTPT],scaling=constrain
ed )]):
end:


bouncer(Pi/4,Pi/8);
```



The unterminated line is the incident ray and the returning path ends at its last bounce point. That is the returning ray will reflect at an angle smaller than theta, the lower left angle of the colored triangle and will therefore not hit either leg of the triangle again.

If α is the incident angle which is less than or equal to theta then the angle at which the ray rebounds at the ith bounce is $\alpha + i\,\theta$ (where the initial bounce is the "zeroth"). The condition for the ray to return exactly along its original path is that it hit one of the legs at a right angle, that is that $\alpha + n\,\theta$ is a right angle

```
eqn1:=beta+n*alpha=Pi/2;
```

```
critbeta:=solve(eqn1,beta);
```

Since $\beta \leq \alpha$ we see that n is the integer part of $\frac{\pi}{2\,\alpha}$ unless this is actually an integer in which case this value of n is one less than $\frac{\pi}{2\,\alpha}$. The latter situation only occurs when theta has the form $\frac{\pi}{2\,k}$ where k is an integer. For the moment, we ignore this possibility and have the formula

```
critbeta:=proc(alpha)
Pi/2-floor(Pi/(2*alpha))*alpha end;
```

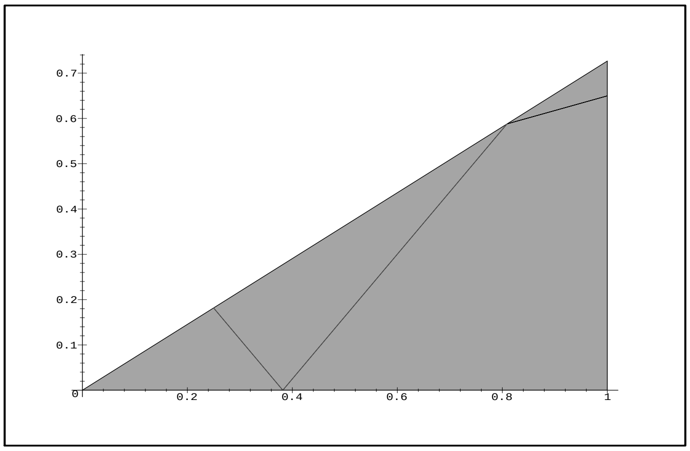
$$\mathit{critbeta} := \mathrm{proc}(\alpha)\, 1/2 \times \pi - \mathrm{floor}(1/2 \times \pi/\alpha) \times \alpha\, \mathrm{end}$$

Now we can draw most critical paths:

```
backbouncer:= proc(alpha) bouncer(alpha,
critbeta(alpha))end;
```

$$\mathit{backbouncer} := \mathrm{proc}(\alpha)\, \mathrm{bouncer}(\alpha,\, \mathrm{critbeta}(\alpha))\, \mathrm{end}$$

```
backbouncer(Pi/5);
```



Note that backbouncer does fine with angles other than those like $\frac{\pi}{6}$, $\frac{\pi}{4}$ etc.

17.4.9 *Exercise: Complete backbouncer so that it works for all positive angles less that* $\frac{\pi}{2}$.

17.4.10 *Exercise: Do an animation of backbouncer as theta changes between* $\frac{\pi}{5} - \frac{1}{4}$ *and* $\frac{\pi}{5} + \frac{1}{4.}$

17.5 Solution

```
movie:=NULL: for i from 1 to 20 do
movie:=movie,backbouncer(Pi/5-1/4+i/40):
od:plots[display]([movie],insequence=true);
```

Exercise: In the animation for the previous exercise the leg at which the ray meets perpendicularly switches several times. Where (i.e. at which angles theta) do such switches happen?

18

19 Closed reflective paths in a polygon.

10 *Definition: A closed polygonal path P is a finite sequence of line segments, F_1, F_2 ... F_n , such that for each i from 1 to n-1, F_i and F_{i+1} have a common vertex which is their only point of intersection and F_n and F_0 have a common vertex which is their only point of intersection. If each F_i is in the interior of a region F then P is said to be "in F". If whenever F_i and F_{i+1} share the vertex q the line extending F_i is the reflection of the line extending F_{i+1} through the tangent to F at q, then P is said to be a* **closed reflective path** *in F.*

It is permissible for two nonadjacent segments in a closed reflective path to cross.

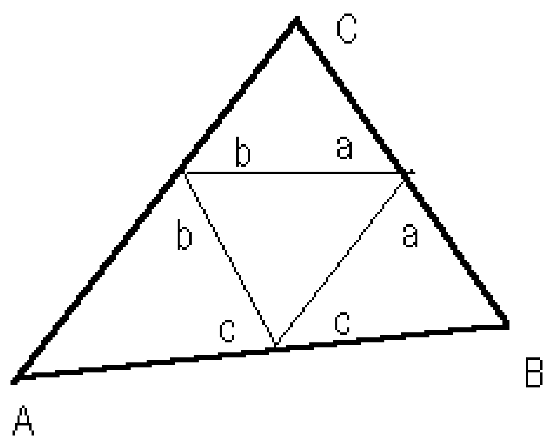
The perimeter of a polygon is a closed path. A square inscribed in a circle is a closed reflective path in the circle. We are interested in which figures are sure to have closed reflective paths and (when possible) describing them. For the general case this is a difficult problem. However there are cases which are readlily, completely, understood and others in which interesting, partial results can be obtained.

Here we investigate the closed reflective paths in (convex) polygons. We will concern ourselves primarily with triangles, describing a quite general technique for deducing interesting results from simple euclidean geometry and Maple's ability to solve systems of linear equations. We will illustrate the general method by applying it to the following question:

19.0.1 *Question: Which triangles have closed reflective paths?*

19.0.2 *Easier Question: Which triangles have triangular closed, reflective paths?*

The easier question is about a general triangle and whether it has closed trangular reflective path. We draw a general triangle and a prospective closed triangular reflective path.



We have six unknown quantities: A,B,C,a,b,c and five equations implied by the five labeled triangles. There are also three inequalities imposed by the situation.

```
restart;
eq0:=A+B+C=Pi;
```

```
eq1:=A+b+c=Pi;
eq2:=a+B+c=Pi;
eq3:=a+b+C=Pi;
eq4:=Pi-2*b + Pi-2*c + Pi-2*a = Pi;
ineq5:=a<Pi/2;
ineq6:=b<Pi/2;
ineq7:=c<Pi/2;
```

$$eq0 := A + B + C = \pi$$

$$eq1 := A + b + c = \pi$$

$$eq2 := a + B + c = \pi$$

$$eq3 := a + b + C = \pi$$

$$eq4 := 3\pi - 2b - 2c - 2a = \pi$$

$$ineq5 := a < \frac{1}{2}\pi$$

$$ineq6 := b < \frac{1}{2}\pi$$

$$ineq7 := c < \frac{1}{2}\pi$$

```
sln1:=solve({eq0,eq1,eq2,eq3,eq4,ineq5,ineq6
,ineq7},{a,b,c,A,B,C});
```

$$sln1 := \{ -b - c + \frac{1}{2}\pi < 0,\ 0 < c,\ b < \frac{1}{2}\pi,\ 0 < b,\ c < \frac{1}{2}\pi,\ A = -b - c + \pi,\ C = c,\ a = -b - c + \pi,\ B = b \}$$

What do we conclude from this computation? Notice that $B = b$ and $C = c$. Hence it follows that $A = a$. and we conclude that the three corner triangles are all similar to the original triangle. Also, we see that all three angles of the orginal triangle must be acute. This proves the following theorem.

8 *Theorem: Suppose a $\triangle ABC$ has a triangular closed reflective path. Then each angle of $\triangle ABC$ is acute and each angle of the path is the supplement of the angle of the triangle which is opposite it. Furthermore, each of the three corner triangles cut off by the path is similar to $\triangle ABC$*

Two natural questions occur here.

19.0.3 *Question: Does every triangle with all angles acute have a closed triangular path?*

19.0.4 *Question: Can a triangle have more than one closed triangular path?*

19.0.5 *Exercise: Show that every non-obtuse, isosceles triangle has a closed reflective path and describe a simple way to construct it (even with straight-edge and compass).*

19.0.6 *Exercise: Create a Maple word which inputs the three vertices of a triangle and draws the triangle with its unique closed reflective path if it has one.*

Look in the next worksheet if you want to see the answers to some of these questions and exercises.

19.0.7 *QUESTION: Can triangles have a closed quadrilateral reflective paths?*

One investigates this question just as we did for triangular paths. There are more sub-figures, hence more equations.

Plot: rpathq2.eps

19.0.8 *Exercise: Does the equilateral triangle have a closed quadrilateral reflective path?*

19.0.9 *Exercise: Does the 3-4-5 right triangle have a closed quadrilateral reflective path?*

19.0.10 *Exercise: Is there a triangle with a pentagonal closed reflective path of length 5?*

11 *DEFINITION: A figure F has a self-tracing reflective path if it contains a polygonal arc made of segments F_1, F_2 ... F_n such that (1) each F_i has its end points on the boundary of F, (2) for each i from 1 to n-1, F_i and F_{i+1} have a common vertex which is their only point of intersection and (3) for each i from 1 to n-1 the line extending F_i is the reflection of the line extending F_{i+1} through the tangent to F at their common vertex, and (4) F_0 and F_n are normal to the boundary of F.*

The idea of a self-tracing reflective path is one that is reflected back upon itself. Note that a self-tracing reflective path in a triangle

Exercise: Find all triangles that have self-tracing reflective paths of length 2. Within those triangles find all self-tracing reflective paths.

Plot: rpathq3.eps

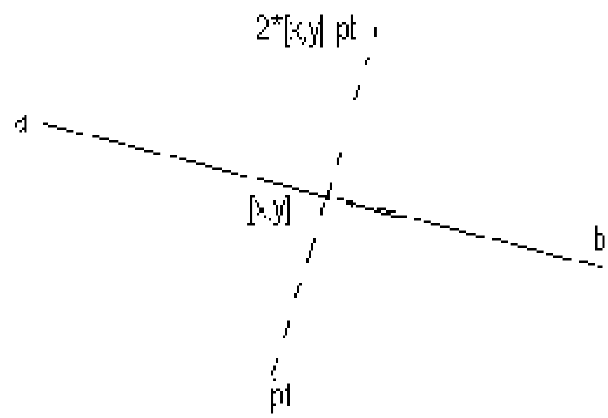
Hint: The rectangle gives the equation $\pi - 2\,b + a + c + B = 2\,\pi$

19.0.11 *Exercise: Show that a triangle cannot have a triangular self-tracing reflective path.*

20 Closed reflective paths in a triangle.

We will investigate here some of the questions raised in the previous worksheet..

Investigation: In order to draw some pictures, we need a word that reflects a point pt about the line through two other points a and b. This word is defined in the most natural way: First write an equation for the line through a and b, and an equation for the line through pt which is perpendicular to the line through a and b. These equations are called lin and linperp in the definition below. Then solve the system of equations $\{lin, linperp\}$ simultaneously for x and y, and substitute that into the expression for the reflected point, $pt + 2([x, y] - pt) = 2[x, y] - pt$.




```
reflect := proc(pt,a,b)
local linperp,cen,x,y,lin;
lin := (b[2]-a[2])*(x-a[1])-
(b[1]-a[1])*(y - a[2]);
linperp := (b[1]-a[1])*(x-pt[1])+
(b[2]-a[2])*(y - pt[2]);
cen := solve({linperp,lin},{y,x});
subs(cen,pt+2*([x,y]-pt))
end:
```

```
reflect([0,0],[1,0],[0,1]);
```

$$[1, 1]$$

Now, let's draw a closed path in a triangle. Take a sample triangle.

```
a := [0,0]:  b :=[3,4]:  c:=[-2,7]:
```

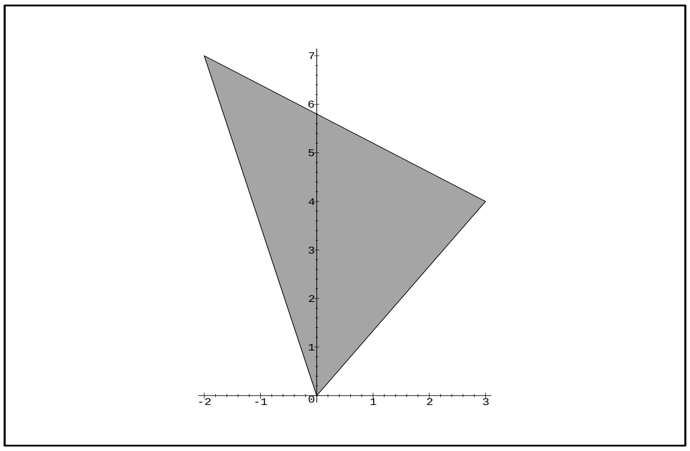
```
pla := plots[textplot]([[op(a),`A`
],[op(b),`B`],
[op(c),`C`]],align={ABOVE,RIGHT},font=[TIMES,BOLD,18]):;
```

```
with(plots):
```

tr1 is a tan copy of the triangle abc

```
tr1:=plots[polygonplot]([a,b,c],style=patch,c
olor=tan,scaling=constrained) :
```

```
tr1;
```



drawtr is a word which makes it easy to draw colored triangles. We will use it to draw the reflection of abc about bc and the reflection of that about the segment c ra.

```
drawtr := (a,b,c,cl) -> plots[polygonplot]([a,b,c],style=patch,color=cl,scaling=constrained):
```

```
ra := reflect(a,c,b);
```

$$ra := [\frac{87}{17}, \frac{145}{17}]$$

```
tr2 :=drawtr(ra,b,c,grey):
```

```
rb := reflect(b,c,ra);
```

$$rb := [\frac{1195}{901}, \frac{10622}{901}]$$

```
plra := plots[textplot]([[op(ra),‘rA‘
],[op(rb),‘rB‘]
],align={ABOVE,RIGHT} );
```

$plra :=$ **PLOT(TEXT(**[5.117647059, 8.529411765], rA, ***ALIGNABOVE***, ***ALIGNRIGHT***),
TEXT([1.326304107, 11.78912320], rB, ***ALIGNABOVE***, ***ALIGNRIGHT***))

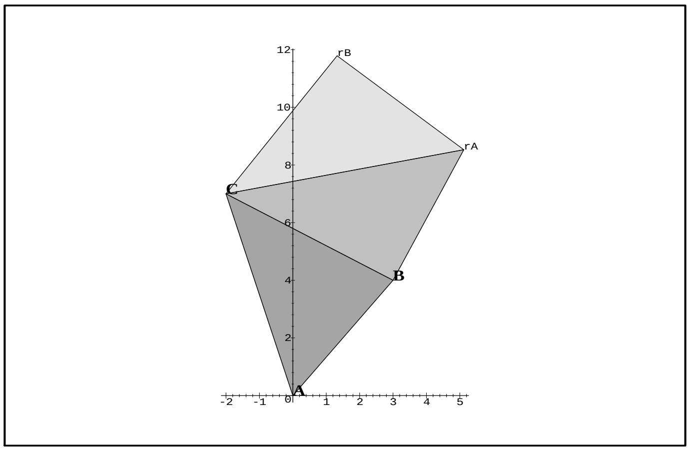
```
tr3 := drawtr(rb,ra,c,yellow):
```

Now look at all three triangles together.

```
printlevel:=1;
```

$$printlevel := 1$$

```
plots[display]([pla,plra,tr1,tr2,tr3]);
```



Take a point p on ab and its image q on ra rb. These can be obtained by specifying a number r between 0 and 1 and forming convex combinations. We will take r to be 1/3 for starters.

```
r := 1/3;
```

$$r := \frac{1}{3}$$

```
p := r*a+(1-r)*b; q := r*ra+(1-r)*rb;
```

$$p := [2, \frac{8}{3}]$$

$$q := [\frac{7001}{2703}, \frac{9643}{901}]$$

```
plqp := plots[textplot]([[op(q),`Q`
],[op(p),`P`]
],align={below,RIGHT},font=[TIMES,BOLD,18]);
```

$$\mathit{plqp} := \mathbf{PLOT}(\mathbf{TEXT}([2.590085091,\ 10.70255272],\ Q,\ \mathit{ALIGNRIGHT}),\ \mathbf{TEXT}([2.,\ 2.666666667],\ P,\ \mathit{ALIGNRIGHT}),\ \mathrm{FONT}(\mathit{TIMES},\ \mathit{BOLD},\ 18))$$

If we look at the intersection p1 of the segment pq with the segment bc, p1 must be of the form t*p + (1-t)*q for some t in [0,1] and also of the form s*b + (1-s) for some s in [0,1]. Thus the difference must be zero.

```
z1 := evalm(t*p+(1-t)*q-( s*b+(1-s)*c));
```

$$z1 := \left[-\frac{1595}{2703}\,t + \frac{12407}{2703} - 5\,s,\ -\frac{21721}{2703}\,t + \frac{3336}{901} + 3\,s\right]$$

```
sol1 :=solve({z1[1]=0,z1[2]=0},{s,t});
```

$$\mathit{sol1} := \{s = \frac{9703}{11730},\ t = \frac{177}{230}\}$$

So p1 is determined.

```
p1 := subs(sol1,t*p+(1-t)*q);
```

$$p1 := [\frac{5011}{2346}, \frac{17667}{3910}]$$

In the same way, the intersection p2 of the segments pq and c ra can be determined.

```
z2 := evalm(t*p+(1-t)*q-(s*c+(1-s)*ra));
```

$$z2 := \left[-\frac{1595}{2703}\,t - \frac{6832}{2703} + \frac{121}{17}\,s,\ -\frac{21721}{2703}\,t + \frac{1958}{901} + \frac{26}{17}\,s\right]$$

```
sol2 :=solve({z2[1],z2[2]},{s,t});
```

$$sol2 := \{t = \frac{34}{99},\ s = \frac{6038}{15741}\}$$

```
p2 := subs(sol2,expand(t*p+(1-t)*q));
```

$$p2 := [\frac{58079}{24327}, \frac{2125457}{267597}]$$

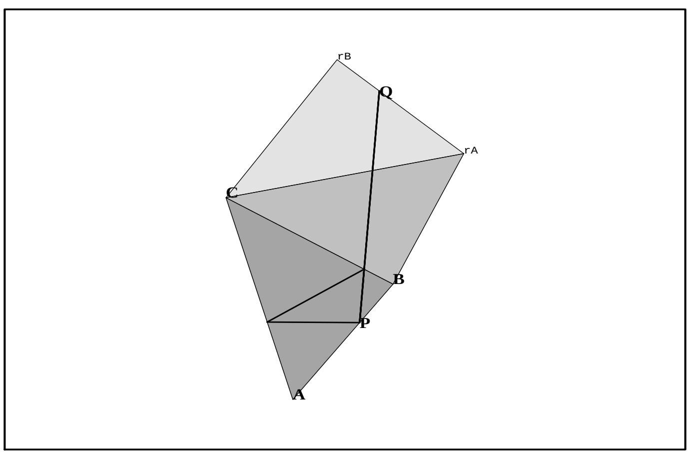
Now reflect p2 about bc to rp2. So the triangular path p p1 rp2 lying in triangle abc represents the path of a ball starting at p on ab, 'bouncing off' side bc at p1, then bouncing off side ac at rp2, and returning to side ab at the point where it started, p.

```
rp2 := reflect(p2,b,c);
```

$$rp2 := [\frac{-12076}{15741}, \frac{42266}{15741}]$$

```
path :=polygonplot({[p,q],[p,p1,rp2]},
style=patch,thickness=3,color=red):
```

```
plots[display]([tr1,tr2,tr3,path,pla,plra,plq
p],axes=none);
```



Whoa! We can see clearly that the triangular path in the tan (original triangle abc) is not a reflective path because the angles formed at the beginning and ending point are not the same.

We can back up and make use of a standard method to decide when there is a reflective path.

Make a sketch of the problem and give names to the variable angles and segments in the sketch. Then use basic geometric theorems such as the sum of the angles in a triangle add up to Pi, vertical angles are equal, etc to write down equations about these angles and segments. Then solve the equations. Use solve in Maple. This enables you work on larger problems than you would do by hand.

The picture for our triangle problem is something like this.

Plot: rpaths2.eps

```
a := 'a':  b := 'b':  c := 'c':
eq1 := A+B+C=Pi;
```

$$eq1 := A + B + C = \pi$$

```
eq2 := C+b+a=Pi;
```

$$eq2 := C + b + a = \pi$$

```
eq3 := A+b+c=Pi;
```

$$eq3 := A + b + c = \pi$$

```
eq4 := (Pi-2*b)+ (Pi-2*c)+ (Pi-2*a)=Pi;
```

$$eq4 := 3\,\pi - 2\,b - 2\,c - 2\,a = \pi$$

```
eq5 := a + c + B = Pi;
```

$$eq5 := a + c + B = \pi$$

```
sol :=solve({eq1,eq5,eq4,eq2,eq3
},{A,B,C,a,b,c});
```

$$sol := \{c = c,\ b = b,\ C = c,\ B = b,\ a = -b - c + \pi,\ A = -b - c + \pi\}$$

```
subs(sol,a+b+c);
```

$$\pi$$

Note it follows from the solution here that the angles A,B, and C must all be acute angles in order that there be a closed reflective triangular path. It also says that the three triangles cut off by the path are similar to the original triangle. The path is unique in this case, as we can see below.

```
drawtr := (a,b,c,cl) ->
polygonplot([a,b,c],style=patch,color=cl,scaling=constrained);
```

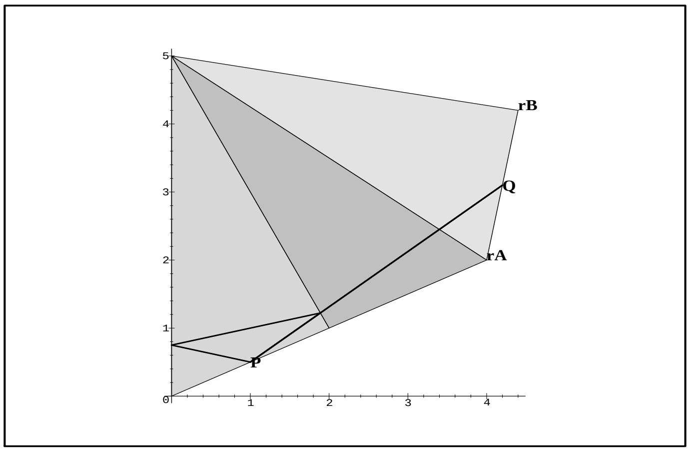
$$drawtr := (a,\ b,\ c,\ cl) \to \mathrm{polygonplot}([a,\ b,\ c],\ style = patch,\ color = cl,\ scaling = constrained)$$

```
drawit := proc(a,b,c,r )
local tr1, ra, tr2, rb, tr3, p, q, eq1, sol1, p1, eq2, sol2, p2, rp2,
rq1, rq, path;
tr1 := drawtr(a,b,c,turquoise);
ra := reflect(a,c,b);
tr2 :=drawtr(ra,b,c,gray);
rb := reflect(b,c,ra);
tr3 := drawtr(rb,ra,c,yellow):
p := r*a+(1-r)*b; q := r*ra+(1-r)*rb;
eq1 := evalm(t*p+(1-t)*q-( s*b+(1-s)*c));
sol1 :=solve({eq1[1],eq1[2]},{s,t});
p1 := subs(sol1,t*p+(1-t)*q);
eq2 := evalm(t*p+(1-t)*q-(s*c+(1-s)*ra));
sol2 :=solve({eq2[1],eq2[2]},{s,t});
p2 := subs(sol2,t*p+(1-t)*q);
rp2 := reflect(p2,b,c);
rq1 := reflect(q,c,ra);
rq := reflect(rq1,b,c);
plra := plots[textplot]([[op(ra),`rA`
],[op(rb),`rB`]],align={ABOVE,RIGHT},
font=[TIMES,BOLD,18]);
plqp := plots[textplot]([[op(q),`Q` ],[op(p),`P`]
],align={below,RIGHT},font=[TIMES,BOLD,18]);
path :=polygonplot({[p,q],[p,p1,rp2]},style=patch, thickness =3,
color=magenta):
plots[display]([tr1,tr2,tr3,path,plra,plqp]);
end:
```

```
Warning, `plra` is implicitly declared local
```

```
Warning, `plqp` is implicitly declared local
```

```
drawit( [0,0],[2,1],[0,5],1/2);
```



We can animate this to show better what we mean.

```
plots[display]([seq(drawit([0,0],[2,1],[0,5],
i/14),i=1..13)],insequence=true);
```

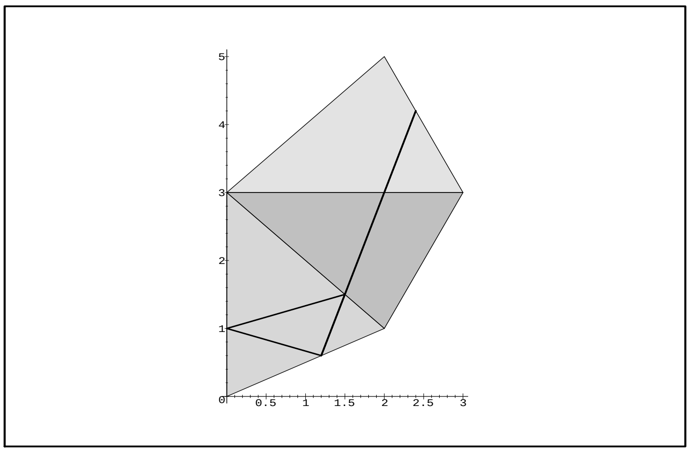
This word draws the unique reflective triangular path if you input a triangle a,b,c with all angles acute.

```
drawit := proc(a,b,c )
local tr1, ra, tr2, rb,r, tr3, p, q, eq1, sol1, p1, eq2, sol2, p2,
rp2, rq1, rq, path,eq;
tr1 := drawtr(a,b,c,turquoise);

ra := reflect(a,c,b);

tr2 :=drawtr(ra,b,c,grey);

rb := reflect(b,c,ra);

tr3 := drawtr(rb,ra,c,yellow):

p := r*a+(1-r)*b ; q := r*ra+(1-r)*rb
;

eq := linalg[dotprod](q-p,b-a) =
linalg[dotprod](p-q,ra-rb) ;

r:= solve(eq, r );
p := r*a+(1-r)*b ; q := r*ra+(1-r)*rb ;

eq1 := evalm(t*p+(1-t)*q-( s*b+(1-s)*c));

sol1 :=solve({eq1[1],eq1[2]},{s,t});

p1 := subs(sol1,convert(evalm(t*p+(1-t)*q)
,list));
```

```
eq2 := evalm(t*p+(1-t)*q-(s*c+(1-s)*ra));

sol2 :=solve({eq2[1],eq2[2]},{s,t});

p2 :=
subs(sol2,convert(evalm(t*p+(1-t)*q),list));

rp2 := reflect(p2,b,c);

rq1 := reflect(q,c,ra);

rq := reflect(rq1,b,c);

path :=polygonplot({[p,q],[p,p1,rp2]},
style=patch, thickness=3,color=red):

plots[display]([tr1,tr2,tr3,path ]); end:

printlevel:=1;
```

$$printlevel := 1$$

```
drawit([0,0],[2,1],[0,3]);
```



20.1 Problems

We used an algebraic argment to show that there is only one closed reflective path in an acute angled triangle. Show that the following theorem can be used to give a synthetic argument for this (and then prove the theorem). Theorem: Suppose that triangles ABC and A'B'C' are inscribed in triangle T so that AB is parallel to A'B', BC is parallel to B'C' and CA is parallel to C'A'. Then the triangles are equal.

12 *Definition. A linear reflective path is one which is piecewise linear (with two endpoints). So at the endpoints it bounces off at a right angle.*

20.1.1 *Show that if a triangle has a linear reflective path consisting of two line segments end to end then it is an isosceles triangle.*

20.1.2 *Find a triangle which has a linear reflective path with four line segments.*

20.1.3 *The bowtie problem. Does a triangle have closed reflective bowtie path?*

20.1.1 Hints on a solution

```
rhs(eq1);

eq1 := A+B+C=Pi;

eq2 := a+B+c =Pi;

eq3 := d + b+C =Pi;

eq4 := 2*Pi-(A+b+c)=Pi-(a+d);

eq5 := Pi-2*c+ (Pi-2*a) +a+d = Pi;

eq6 := a+d + (Pi-2*d) + (Pi-2*b)=Pi;

sol :=
solve({c<Pi/2,b<Pi/2,eq1,eq2,eq3,eq4,eq5,eq6},
{A,B,C,a,b,c,d});
```

20.1.4 *Does there exist a closed reflective path in a triangle with six bounces?*

20.1.2 Hints on a solution.

```
restart;

eq0 := A+B+C=Pi;

eq1 := A+a+b=Pi;

eq2 := 0=0;

eq3 := C +c +d = Pi;

eq4 := B + f +e = Pi;

eq5 := Pi-2*f + a+f + Pi-2*e = Pi;

eq6 := 0=0;

eq7 := Pi-2*a + Pi-2*b + b +c + a +f =
2*Pi;

eq8 := Pi - (a+f) + Pi - (b+c) + (e+d) =
2*Pi;

eq9 := Pi-2*d + b+c + Pi-2*c = Pi;

sol
:=solve({f<Pi/2,e<Pi/2,b<Pi/2,c<Pi/2,d<Pi/2,eq0,eq1,eq2,eq3,eq4,eq5,e
q6,eq7,eq8,eq9},{A,B,C,a,b,c,d,e,f});
```