

ASSIGNMENT VII

Graphic representation of general solid objects



In this final assignment we shall combine all the experience and the software we have accumulated so far to put together a package of MATHEMATICA functions and procedures that will enable us to depict realistically on the screen any collection of solid objects.

It is necessary to make some general assumptions before we can enter into the details of the package. Firstly, we assume that the solids to be depicted are all polyhedra (not necessarily convex). We also take the view that solids are mutually exclusive. Thus we shall assume that the input consists of a sequence of mutually non intersecting polyhedra. In particular, the faces of these polyhedra will not cut each other. Our depiction thus reduces to the drawing of the visible portions of the faces of these polyhedra. In fact, since we draw faces by drawing their boundary edges, the only things we need to draw are the visible portions of these edges. We can take the view that the *hiding* is actually done by the faces themselves, imagining that they are in fact *opaque*. In this manner we can do our processing, as if we were given a sequence of opaque polygons and we are to construct the visible portions of their boundary edges. As a matter of fact, if we know before hand that some of the objects are convex polyhedra, you can make use of the package you constructed in your fourth assignment to weed out before hand all the faces that you know for sure are going to be entirely hidden from view. As in the previous assignments, visibility of an item here means that the item is visible to an observer placed at a point *far away on the negative z-axis*. Depicting a visible item again means drawing its projection on the x, y -plane.

This given let us assume that our input data is stored in the array “**faces**”, where each component of **faces** consists of a list of points given in the form

$$\{\{\mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1\}, \{\mathbf{x}_2, \mathbf{y}_2, \mathbf{z}_2\}, \{\mathbf{x}_3, \mathbf{y}_3, \mathbf{z}_3\}, \dots, \{\mathbf{x}_{m+1}, \mathbf{y}_{m+1}, \mathbf{z}_{m+1}\}\} \quad (1)$$

(with $\{x_{m+1}, y_{m+1}, z_{m+1}\} = \{x_1, y_1, z_1\}$). This is to give the sequence of vertices of the corresponding face in the order they are encountered as we walk around its boundary *starting* and *ending* with the *same* point!. We do not require here that the walk should be necessarily counterclockwise as seen from the outer normal. Since we need this special arrangement of vertices only for the faces that belong to a convex component of the given composite object. Nevertheless, they must appear in a walk order so that we may be able to construct the edges of the face by reading off consecutive pairs. This requirement will also be crucial in the next stage, as we shall have to use the functions of Assignments V and VI in deciding the visibility of points of our figure

Your first task is to write a MATHEMATICA function with heading

pickedges[**faces**_]

which returns the list of all the edges of all the faces of all the given objects. It will be good that we agree before hand that the output of **pickedges** is stored as the global array **edges**. However, we must assure

that **edges** is *without repetitions*. You can do this if you start **pickedges** by initializing **out** to be the empty sequence, then go through a double loop which picks from each face each of its edges then instead of “**Join**” use the MATHEMATICA command **Union**. This will automatically toss out repetitions, provided that all the edges that represent the same **same geometrical segment** are given by the same **pair of points**. This can be guaranteed, if **pickedges** goes through a face as given in (1), by first picking the the i^{th} pair in the form

$$\mathbf{edge} = \{\{\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i\}, \{\mathbf{x}_{i+1}, \mathbf{y}_{i+1}, \mathbf{z}_{i+1}\}\}$$

then calling the **Union** command not with **edge** as input but rather with **Sort[edge]**. This will assure that all the edges appearing in the returned sequence will be pairs of triplets in lexicographic order.

I must emphasize that it is important that you do this since otherwise each edge will appear at least twice in the output of **pickedges**. This is because it is shared by the two adjacent faces that contain it. In fact, a look at the attached chair should convince you that the same (geometrical) edge may appear even more often than twice in your list of faces. Since your final program will have to successively work on **each** component of **edges**, you can easily see that the processing time will be doubled (at least) if you fail to take this precaution.

After the execution of “**pickedge**”, the really interesting, and most substantial part of the computation begins. Namely, that which constructs the information as to what portion of each edge is visible to the observer.

We do this by breaking up the edges in **edges** into totally visible and totally invisible portions. Under the assumption that our object is composed of a number of non intersecting solids, this *breaking up* can be carried out by a process based on a very simple observation. Namely, note that when we project down onto the x, y -plane all the edges, (by removing the z -coordinates), the segments we obtain will, most likely, cut each other. Now, if we focus on how the projection of a particular edge is cut up in to pieces by the projections of the other segments, we should easily see that

The hidden portion of an edge must project down onto a union of such pieces

The reason for this is that if we let a point P travel along edge **E**, then at the moment that P passes from a visible to a hidden portion of **E** (or viceversa), it must lie directly above an edge **F** of the face that is doing the hiding! Thus the x, y -projection of P must be one of the points in which the projection of **E** intersects the projection **F**.

This given, our next step is to construct and store in **edgepieces** the complete list of all the pieces into which the elements of **edges** break up when we cut them *the same way* as their projections.

I can be more specific. You are to construct a MATHEMATICA function with heading

$$\mathbf{brkup}[\mathbf{edge_}, \mathbf{inedges_}]$$

which returns the list **outpieces** containing all the pieces into which **edge** is to be cut up. You can put together this function by calls of the procedure **cut** you wrote in Assignment seven. You start **brkup** by storing in **Exy** the x, y coordinates of the two vertices of **edge**. Let us say that

$$\mathbf{edge} = \{\{\mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1\}, \{\mathbf{x}_2, \mathbf{y}_2, \mathbf{z}_2\}\} \quad . \quad (2)$$

then

$$\mathbf{Exy} = \{\{\mathbf{x}_1, \mathbf{y}_1\}, \{\mathbf{x}_2, \mathbf{y}_2\}\}$$

Of course the two pairs $\{\mathbf{x}_1, \mathbf{y}_1\}$ and $\{\mathbf{x}_2, \mathbf{y}_2\}$ represent the projections in the x, y -plane of the end points of **edge**. In other words **E_{xy}** is the x, y -projection of **edge**. Next you have **brkup** go through an **i**-loop where at the i^{th} step you read off the x, y coordinates of the two vertices of **inedges[[i]]**, and store them in **F_{xy}**. Let us say that this results in

$$\mathbf{F}_{xy} = \{\{\alpha_1, \beta_1\}, \{\alpha_2, \beta_2\}\} .$$

Then make a call of **cut** by the command

$$\mathbf{cut}[\mathbf{E}_{xy}, \mathbf{F}_{xy}]$$

and store its output precisely as you did in Assignment seven. You will end up with a list of numbers between 0 and 1 intermixed with a few 100's. Sort this list, throw away the 100's and prepend 0 and append 1 (whenever necessary) as you did before. Let us say you store the resulting sequence in **S**. This is the list of successive barycentric coordinates of the points where **edge** has to be cut. Note that from (2) we read that the parametric equation of **edge** are

$$\begin{aligned} x(s) &= x_1 + s(x_2 - x_1) \\ y(s) &= y_1 + s(y_2 - y_1) \quad . \quad (0 \leq s \leq 1) \\ z(s) &= z_1 + s(z_2 - z_1) \end{aligned} \tag{3}$$

This given, the i^{th} component of the returned sequence is simply obtained by setting

$$\mathbf{outpieces}[[i]] = \{\{\mathbf{x}(\mathbf{S}[[i]]), \mathbf{y}(\mathbf{S}[[i]]), \mathbf{z}(\mathbf{S}[[i]])\}, \{\mathbf{x}(\mathbf{S}[[i+1]]), \mathbf{y}(\mathbf{S}[[i+1]]), \mathbf{z}(\mathbf{S}[[i+1]])\}\} .$$

To construct the desired final list **edgepieces** you write a function with heading

$$\mathbf{brkall}[\mathbf{inedges}_-]$$

which starts by initialising **allout** to the empty list. Then goes into an **i**-loop which at the i^{th} stage constructs the pieces into which **inedges[[i]]** has to be cut. This is achieved by the call

$$\mathbf{brkup}[\mathbf{inedges}[[i]], \mathbf{inedges}] .$$

The resulting **outpieces** are then concatenated to a progressively growing list “**allout**” by a call of the MATHEMATICA command **Join**. At the end of this loop **brkall** simply returns **allout**. (Note that this list should now contain all the pieces into which the edges of the given figure have been cut up).

After **brkall** does its job on **edges** and its output is stored in **edgepieces** we are finally ready for the final step: *the selection of the visible edges*. For this we need a MATHEMATICA function with heading

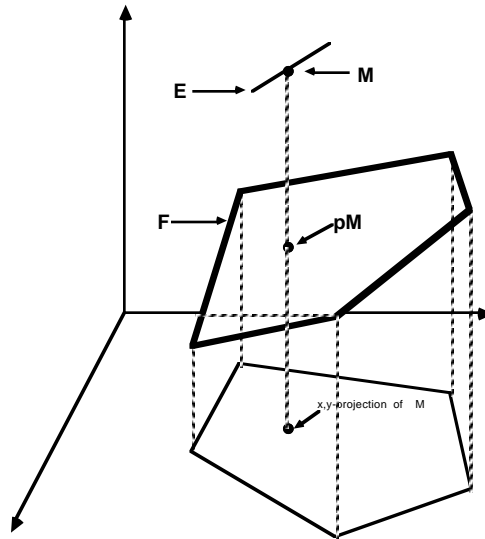
$$\mathbf{hides}[\mathbf{F}_-, \mathbf{E}_-]$$

which returns **True** if the face **F** hides the edge **E** and returns **False** otherwise. The manner in which **hides** produces the answer can again be based on a very simple observation. Note that because of the cutting that has taken place, any of the edges in **edgepieces** is either entirely visible or entirely invisible. Under these circumstances, in a call of **hides** with **F** equal to one of the faces in our original list **faces** and **E** equal to one of the edges in **edgepieces**, you should return **True** if and only if

*The mid point **M** of **E** is hidden by **F***

Now a look at the picture below should make it clear that this will be so if and only if

- 1) The the x, y -projection of \mathbf{M} is surrounded by the x, y -projection of the boundary of F .
- and**
- 2) The point \mathbf{pM} obtained by projecting \mathbf{M} onto the plane of \mathbf{F} has a z -coordinate smaller than that of \mathbf{M} itself.



Clearly, the first condition can be tested by a call of the procedure **inside** of Assignment six. On the other hand, if the mid point of \mathbf{E} is

$$\mathbf{M} = \{\alpha, \beta, \gamma\}$$

and the equation of the plane of \mathbf{F} is

$$ax + by + cz = d \tag{4}$$

then substituting the x and y coordinates of \mathbf{M} for x and y in (4) and solving for z (assuming $c \neq 0$) (*) we get that

$$z = d - \frac{a\alpha + b\beta}{c}$$

is the z -coordinate of the point \mathbf{pM} . Thus the simple inequality

$$\gamma > d - \frac{a\alpha + b\beta}{c}$$

will be the only other thing we need to test to decide on the validity of the second condition.

Clearly you should carry out this test only when the call of **inside** has revealed that condition 1) is satisfied. In that case **hides** must compute the coefficients a, b, c, d . Now recall from analytic geometry that

(*) Note that c turns out to be 0 only when the plane of \mathbf{F} is perpendicular to the x, y plane and the projection of the boundary of \mathbf{F} reduces to a segment. This clearly cannot happen when the first test is satisfied

if a, b, c are the components of a vector that is perpendicular to the plane of \mathbf{F} then the equation of the latter plane may be written as

$$a x + b y + c z = a x_1 + b y_1 + c z_1$$

where x_1, y_1, z_1 is any vertex of \mathbf{F} . Thus once you have constructed a, b, c you may take

$$d = a x_1 + b y_1 + c z_1 . \quad (5)$$

On the other hand we know that a normal to the plane of two vectors

$$U = (u_1, u_2, u_3), V = (v_1, v_2, v_3).$$

has components given by the determinants

$$a = \det \begin{pmatrix} u_2 & u_3 \\ v_2 & v_3 \end{pmatrix}, b = \det \begin{pmatrix} u_3 & u_1 \\ v_3 & v_1 \end{pmatrix}, c = \det \begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix} .$$

you can thus calculate a, b, c from the above formulas using

$$U = Q - P, V = R - Q$$

with P, Q and R taken to be the first three consecutive vertices of \mathbf{F} . This done, the value of d can then be obtained from equation (5) with x_1, y_1, z_1 equal to the coordinates of P .

Once you have finished with **hides** you should construct a function with heading

visible[**E**_, **infaces**_]

which returns **True** if **E** is not hidden by any face in **infaces** and **False** otherwise. You could start **visible** by initializing **out** to be **True**. This done you enter an **i**-loop which at the i^{th} stage makes the call "**hides** [**infaces**[[**i**], **E**]]". If this call returns **True** you change **out** to **False** and exit the loop. The final command in **visible** could be **Return**[**out**], since if you completed the loop and never found a face that hides **E** then **E** must be visible and **out** has remained **True** throughout the calculation.

After you are finished with **visible** you can complete your task by writing a procedure with heading

draw[**infaces**_]

which returns a 2-dimensional array "**visedges**" whose rows contain the graphic commands needed to display the resulting sequence of visible edge-pieces. This final procedure should contain all the procedures we have described. It should start with a call of **pickedges** to produce the array **edges**. Then calls **brkall** to produce the array **edgepieces**. This done it should initialize **visedges** to an empty array then go into an **i** loop which at the i^{th} step tests whether **edgepiece**[[**i**]] is hidden by any face in **infaces**. This can be carried out the call

visible [**edgepiece**[[**i**]] , **infaces**]

If this returns **True**, then you append to **visedges** the graphic commands that draw **edgepiece**[[**i**]] and you do nothing otherwise.

The output of **draw** (that is the array **visedges**) could be either in MATHEMATICA or in POSTSCRIPT graphics commands as you wish.

You should test your package by taking the **chair** as your sample nonconvex object. Then using the rotation program you have used in assignment IV write a procedure that rotates the chair by a user chosen axis and angle of rotation and then feeds the resulting sequence of rotated faces to the procedure **draw** described above.

NOTE: As final requirement for this course post on your website of all the MATHEMATICA functions and procedures that you have written for this assignment together with one of two correct drawings of **chair** as in the samples given below..

