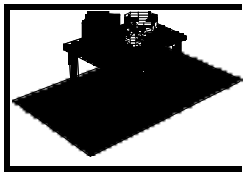
A R C H 3 7 0

3D Modeling & Rendering

3D Concepts:
Data types, Graphics Pipeline, Construction Planes
Form•Z:
Basic operations



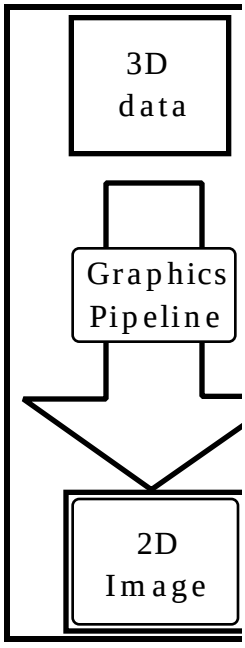
Two-dimensional graphics are traditionally used for capturing the precise kinds of data which are needed to construct a new building, landscape, etc. This is why drafting systems have such complicated snaps and grids and sometimes require (not simply *permit*) the numeric entry of data.

The world of three dimensions, on the other hand, is most often used during the design process to assess the visual character of different design ideas and to answer questions such as "Can we see the garbage can storage area from the deck?".

Thus, 3D systems are often simpler than their 2D cousins, at least in terms of the primitives which they support and the precision of the data created. Many do not even include text primitives! Instead, they focus on making it easy to create and view a 3D representation of the design.

While they may use simpler data, three-dimensional systems are very different from their two-dimensional counterparts in that the 2D image you see on the screen changes dramatically as you manipulate the viewpoint, which adds a layer of complexity to the program as well. Where the display list data in a 2D system pretty much "is" the drawing, the model data of a 3D system is only the raw material from which the screen image is made. In fact, you never "see" the 3D data, only 2D representations of it. The data is delivered to the screen through what is called the "Graphics Pipeline".

The Graphics Pipeline



This term refers to a series of transformations that are applied to the data in order to convert it from its 3D form into the 2D screen image. The "complete" pipeline, like an assembly line, is composed of several distinct data-processing stages. Also like an assembly line, data generally passes along the pipeline unaware of data ahead or behind. Further, depending on the desired "product", we might leave out steps or add steps to the process. Lastly, regardless of the quality of the assembly process, if some of the raw materials (data or information) are missing, the product cannot be completed. Different manufacturers (programs) may utilize different procedures, or choose to skip steps–differences which make the products different (in short, not all 3D programs are alike).

The Raw Material: 3D data

There are three main types of 3D data. These can be thought of as occupying the three geometric domains of 1D (lines), 2D (surfaces), and 3D (solids).

1D Wireframe
If the data consists simply of lines in space it is called *wire frame* (a term which is also used to refer to a “see-through” drawing). Wire frame data is quite easy to create and maintain, and can be converted easily from 2D data such as that of a 2D drafting system by just adding an extra coordinate. The graphics pipeline for wire frame data is also simple. However, as the name implies, it is not possible to view the model "as if it were solid" since opacity data simply isn't present. Nor is it possible to prepare shaded views of wire frame data.

2D Boundary Representation
Data consisting of two-dimensional elements, or polygons, defined in a 3D coordinate system, is often called *boundary representation (b-rep)* data. This is because, like a cardboard or paper model of a shape, it achieves its shape by describing the surface boundaries, or facets of the 3D shape. Because b-rep models contain explicit surface information, they may be used with a graphics pipeline which computes "hidden line" images and surface-shaded images. More complex renderings displaying surface reflectivity and texture can also be computed from b-rep models. For architects and other form-oriented designers, this is often as complex a system as is needed. The problem with this kind of data is that their geometries need not be "real". For instance, it is perfectly legal to create a single polygon floating in space. Since polygons are (by definition) flat, it has no thickness, and thus, no volume. It might *look* like a piece of paper, but a stack of them does not fill up a box! They work in the visualization system, but you couldn't manufacture such a shape; it can't exist in the *real world.*

by Brian R. Johnson

3D Solid Models

When the basic shapes which the system works with are 3D primitives, the system is said to be a "solid modeling system". Such systems are commonly used in industries which actually manufacture objects from CAD data, such as the aerospace, automotive, and consumer-products industries, because the system can guarantee that the shapes it produces can exist in the real world. One strong feature of solid models is that they can contain "negative" data. That is, you can take any two shapes and "subtract" one shape from the other. This is something which b-rep systems generally cannot do. On the other hand, solid modeling systems generally require substantially more compute power to operate than b-rep systems.

The Pipeline: Step 1: *Projection*, or squashing 3D info into 2D

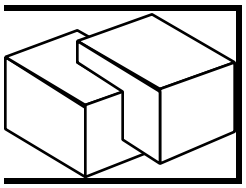
The program needs to convert the (3D) model data into a viewable (2D) form. It turns out that certain matrix formulations can be used for this (a process called *projection*). While many such projections exist, as do manual construction techniques (think of plan-oblique, or axonometric, in addition to perspective), most programs offer only a very limited set, usually perspective and orthographic projections (which are good for elevations, plans, etc.). You can think of the projection like the lens on a camera–when you point a camera at a scene the type of lens (zoom, wide angle, fish-eye) you are using changes the character of the image on the film, but it certainly doesn't change the geometry!

In addition to providing a mechanism for "changing lenses", the program must provide a mechanism (called a *viewing model*[1]) by which the user can control the view. There are two basic ways to approach this: what we call the *environmental* model and the *object-oriented* model. While they are (nominally) equivalent, in that each may be used to specify any view possible in the other, the relative ease with which they may be done is very different.

environmental In environmental viewing models the user positions the "camera" or "eye point" in the same geometrical construct (i.e., in the same coordinate system) as the model itself. Thus, very precise questions may be asked, like "How much of that building will I see if I stand in the mayor's office and look out the window?"

object-oriented Object-oriented viewing models[2] behave as if the camera is fixed (perhaps on a tripod?) and the object (the model) is being manipulated–almost as if you held it in your hands. You bring it closer, turn it this way, or that, and so on. You always look toward the center of the model (which is a good thing!), but you never know exactly where you are relative to the geometry of the model.

The Pipeline: Step 2: *Hidden-line/surface Removal*



Projection is just the beginning of the process too. If the program simply projects all the vertexes of all the polygons onto the picture plane, then all lines will be visible, creating a "wire frame" graphic† . Most programs include algorithms for *hidden line removal* or *hidden surface removal*, processes which make the polygons look solid by obscuring or not drawing the lines "behind" them. Each of these algorithms makes trade-offs between computation time and image quality (i.e., they make errors, but asking for perfection may cost you a long time). Since different data files (i.e., different geometrical configurations) are subject to different types of errors, a program which gives you choices between different algorithms is desirable. You can use the "cheap" (fast) algorithm where possible, but resort to the expensive one(s) as needed.

While a general discussion of hidden-surface removal is beyond the scope of this course, you should be aware that the most common cause of problems is intersecting geometry (like two interlocked cubes). The lines of intersection are *implicit* in the construction, they are not explicitly defined. They occur because of the interaction of two separate polygons. Remembering that the graphics pipeline used in many simple modeling programs, or in quick "preview" displays in more capable programs, processes each polygon separately, you can begin to understand how it can make a mistake trying to process this geometry.

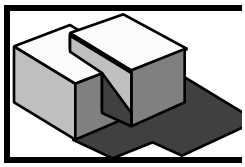
---

[1] *Model* as it is used in this context means "way of doing or describing something". This is more like the usage of "model" in a phrase such as "the parliamentary model of government" than like its usage in "he built a balsa-wood model".

[2] Don't confuse this usage of "object-oriented" with "object-oriented programming" or "object-oriented graphics"–they all refer to different concepts–too many new concepts, not enough words (sigh).

† **Note**: As we have seen elsewhere, it is easy to convert data from a "higher order data type" such as boundary-representation into a "lower order data type" such as wire frame data. Similarly, if the program "knows" what a cube is, as a solid, it's easy to find the six polygons which describe the faces. As before, converting "downward" is easy, while converting upward (wire-frame to polygon, or polygon to solid) is quite difficult or impossible without a source of "superior knowledge" (info that isn't in the lower order model).

The Pipeline: Step 3: *Rendering* or Hand me my Prismacolors!



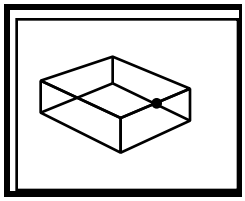
Once projection and hidden-line-removal have occurred, the surfaces are *rendered* onto the screen. This step depends in part on the hardware you are using (you can't display color on a black and white screen, even if you have calculated it!) In the simplest approach each polygon is simply colored white. More complex algorithms employ surface color and algorithmic *shading models* to describe how light interacts with the surfaces of the model. A notch above simple shading is the casting of *shadows*. Such algorithms are gerally called "flat" or "simple cosine" shading algorithms.

> ☛ *Even though we know both shade and shadow are caused by light interacting with the surface, rendering shaded images is algorithmically very different from casting shadows, and many programs (Super3D and AutoShade for example) do shading, but not shadows. For a sense of how much more work it takes to cast shadows, turn on shadows in your program and note how long it takes to re render the image.*

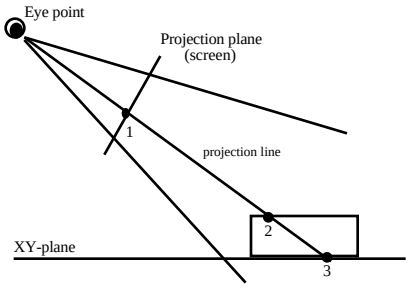
Beyond shading and shadows, some rendering algorithms are able to make a coarsely faceted model (like a hexagonal tube) look smoothly curved (like a true cylinder). This characteristic, called "smoothing", is exhibited by "Gouraud" and "Phong" shading algorithms. Yet more complicated rendering algorithms with names like "ray tracing", and "Radiosity" take into account how light bounces between surfaces within the model. A variety of surface characteristics may be applied like "wallpaper" using a technique generally called *texture mapping.* With moderately complex data they can take hours or even days to render a single image, even on a fast computer. They also make very attractive images, which looks so real that they are called "photorealistic".

In an interesting convergence, rendering algorithms such as radiosity now make more realistic looking images by employing more and more of what we know about the physics of light, so that their results come to more and more resemble, not so much a rendering, as a simulation of the scene. That is, the results are calculated to *be* right, not just *look* right.

Construction Planes: 3D data, 2D screen

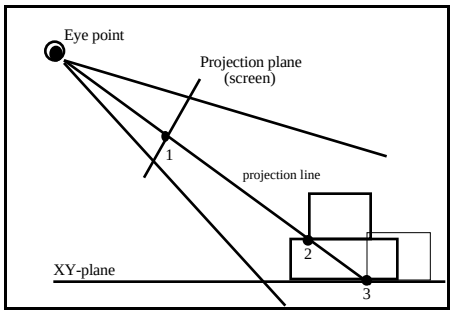


Consider the figure at right, showing a Wireframe view of a simple box. Notice the point on the image indicated by the dot. That single 2D image location represents two distinctly different points in space, one on the front top edge of the box, and one on the back bottom edge of the box. This is the nature of "projection"–it takes 3D data and compresses or discards some of it in such a way that we get a 2D image. The reason that multiple 3D points "map" onto the same 2D point is illustrated in the following figure, which shows a vertical section through the 3D environment, along the view vector (line of sight). This slice also cuts through the projection plane, or screen image, indicated by the line crossing through the cone of view. As with the physical world, we look along straight projection lines, seeing objects at various distances into the scene. Sometimes those objects coincide or overlap. We know they aren't in the same place in space because they move differently as we move around, but just for the one moment they sure seem to be!



Now, consider the problem of pointing in 3D. The *mouse* is restricted to identifying points on the screen, so if we click at point #1 in the figure, there is some ambiguity as to where we are pointing in 3D. We could be pointing at point #2, point #3, or anywhere along the projection line. To remove the ambiguity the 3D program needs another restriction on your pointing operation. If we assume you are indicating locations in the XY plane, the click at point #1 is immediately identifiable as indicating point #3 in space. The XY plane functions as a construction, or *reference* plane. Form•Z permits the definition of any number of reference planes, parallel to the ground plane or one of the elevation planes, or arbitrarily oriented in 3D space.

One of the common mistakes that beginners make when working with 3D programs is not knowing what or where the construction plane is. If our diagram is a cut through the same box shown in the initial figure, and we wish to draw another box "on top" of the first box, as shown in the figure below, we might very well indicate point #1 during the process of drawing the box. If we have not taken care to set the construction plane correctly, the program will think we mean point #3 and we will produce the second box, on the ground, rather than the desired box stacked on top of the original at point #2. The insidious thing is that both interpretations yield the same wireframe axonometric view!



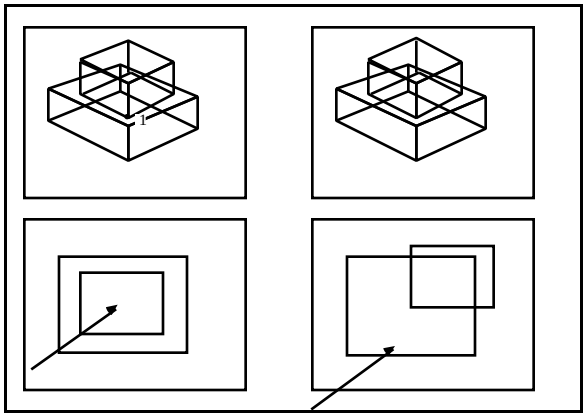
If you Click the mouse at point #1 on the screen, in the Axonometric view...

...and the program thinks the click is at point #2...

...and the program thinks the click is at point #3...

The <u>Axonometric</u> view of the results looks like this, but ...

the <u>Plan</u> view of the results looks like this.



To avoid this mistake, try to think about where you are pointing as a 3D, not a 2D issue; pay attention to the active construction plane setting, and if you aren't sure if you "did it right", switch to another view (plan or elevation are good) to check the results of the operation. As you work with 3D you will find that this becomes easier.

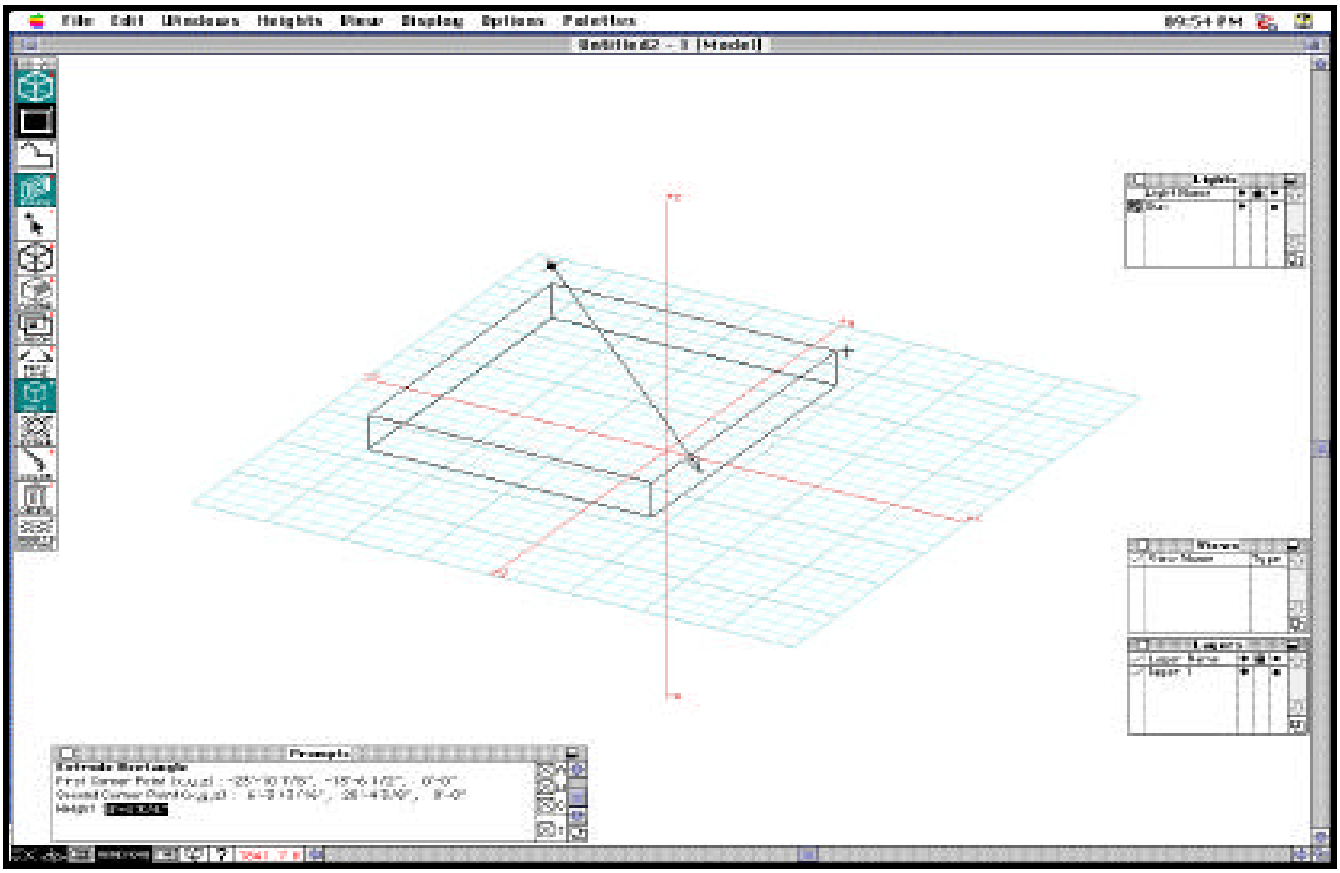
> ☛ *Normally we think of altering the view point and altering the construction plane as quite separate activities, but the Form•Z "preset" view commands (Plan, Elevation, and Axonometric) change both the view point and the current construction plane, so use caution!*

Working Smart in 3D

As with our previous experiences, the primary element in Working Smart is telling the computer what to do, not doing it ourselves. The catch, of course, is in knowing how to tell it to do something. For that you will have to explore the program, finding out what features it provides which allow you to instruct it. In addition, here are some general guidelines which well may make your lives easier.

- Use drawing aids where possible. As in 2D drawing programs, most 3D programs offer some sort of snaps: grid, object, etc. If you use these wherever possible your data will be crisper and more accurate. Also, object snaps in 3D often violate the "drawing happens in the reference plane" idea since the snap points offer real positions in 3D to which you may refer.
- Apply the KISS principle (Keep it <u>Simple</u> Stupid). Most 3D programs offer commands which potentially generate lots of data (the *revolve,* or *lathe* tool is one example). Use these with caution as it is very easy (and tempting) to generate too much data. As a result operations on the data take longer and longer, and the time required to edit or view the model becomes unacceptable.

  If you are interested in smooth curves, look for programs with options for smooth rendering faceted shapes rather than trying to make smooth shapes with vast quantities of small polygons. Use the rendering options, when available, and live with roughly faceted shapes when they aren't.
- Structure your work into hierarchical shapes or layers. You will probably be editing the 3D data using a 2D screen. As the model becomes more and more complex, it is easy for the screen to become cluttered, making it hard to point at or select the things you want to manipulate. Most programs offer commands which help create clump "objects". Since the objects can generally be edited independently of the composite model, they function as a simple way of segregating the data. Further, many programs utilize a system of "layers" by which data may be isolated or segregated as well.



Step 1: Getting Acquainted

Before you start to work on the project, you should take Form•Z for a "test drive" and see how it works.

Launching the Program

Form•Z is a complex program. In order to keep your first encounter with it reasonably uncomplicated, I have used a configuration option available in the program to create a simplified set of menus. In order to use theese, you need to copy the two files **form•Z Menus.370** and **form•Z Preferences.370** to your floppy disk or working directory. You can find copies in the "Resources" folder on the course file server. Once copied, you may launch form•Z by selecting both files in the Finder and then selecting the FILE/OPEN command (or typing -O). This will launch form•Z and set up your environment according to the settings in these two files. Once you launch it you will see a screen display similar to that shown above. The following paragraphs discuss elements of this display.

The Reference Plane

Form•Z displays a light blue grid on a portion of the reference plane (the plane extends to infinity). This is visible in the middle of the screen above. You do not need to constrain your drawing work to the gridded part of the plane.

The Prompts Window

Form•Z displays kind or running dialog with the user in the "Prompts" window here shown located in the lower-left corner of the screen. You will often find it useful to refer to this area to see what the program is expecting of you next.

"Pallette" Windows

Along the right edge of the screen shot you can see three of the various "palette" windows available in form•Z. These display, and allow you to manipulate various aspects of the program, such as the defined layer names, light sources, or saved viewing conditions. These can be moved around, or simply closed when not needed (the screen sometimes get's a bit crowded).

The Menu

(Check out the Form•Z manual for a complete overview and a more complete tutorial)

If you click on one of the icons in the menu along the left edge you will see a menu of options open up to the right. The highlighted choice represents the default choice, but this may be easily changed.

Rows 1, 2 & 3

The Extrusion-mode indicator

During data input you will select primitives from the **second** and **third** rows of the menu to input geometry. Depending on the setting of the options in the **first** row, form•Z will create different kinds of 2D or 3D data. Withe the setting shown in the screen shot, and at left, a rectangle drawn in the construction plane would be automatically extruded into a 3D volume, or brick (you can see one of these shapes, in wire frame, being drawn right now).

Rows 4 & 5

Topological Level

The setting shown in the fourth row of the menu indicates the current "topological level". This setting relates to what selection actions mean and the fact that form•Z's data is that of a solid modeler. The setting shown here (Face) indicates that if we use the pointer tool located in row 5 to select a portion of the data, we will be selecting faces. This makes a significant difference in terms of the editing actions we might make on the shape.

Row 6

Extrusion operation

The icon shown in the next row is almost identical to that of the first row. The only difference is that the setting in the first row relates to actions creating NEW data, while the selection from the sixth row is used to extrude EXISTING data. In other respects they act much the same.

Rows 7, 8 & 9

Editing commands

Some for form•Z's editing options are illustrated in the seventh row of icons, with the "rounding" tool shown here. This tool may be used to round-off corners of square box shapes and so on. Depending on the setting for topological level, this tool may be used to round off one single edge, or segment, all the edges of a face, or all the edges of the object. Rows 8 and 9 provide additional editing operations. The "union" operation shown in row 8 is an example of a Boolean operation, a type of data manipulation unique to true solid modeling programs. The Boolean "difference" operation may be used to literally subtract one shape from another, makeing it easy to "punch holes" in walls, and so on.

Rows 8 & 9

Editing & replicating data

This setting identified in row 8 changes the results of many of the editing operations available in row 9. The simplest example is the difference between "self" and "copy" when performing a "Move" operation. If the operative mode is "self", the selected data is simply relocated. If the operative mode is "copy", then the selected data is used to create new data at the destination, but is left unaltered. This makes it easy to duplicate a shape at some specific offset from the parent shape.

Note: You cannot "move" (translate in space) a shape by simply dragging it as you might elements of a 2D graphic. You must select it, then select the Move tool and specify the starting and ending points of the move.

Rows 10

Properties

Row 10 contains tools for setting and querying data attributes, such as object color, layer, transparency, and such. The "query" tool, which has a question mark as an icon, is quite useful as well, because it can be used to query the volume of a complex shape or to change the layer it is placed on.

Row 11

The purpose of this tool is probably self-evident. It is also possible to delete data by selecting it and the pressing the "delete" key.

Row 12

The last row is used to select or define different construction plane or local coordinate systems. Your most likely choices here are simply the XY, YZ, or XZ planes.

Refernce Plane manipulation

ALONG THE LOWER LEFT EDGE OF THE WINDOW

Across the lower left edge of the window are a number of small controls that you may want to know about. Some of them are a bit beyond the scope of this course, the the ones you might care about are shown here.

"**Perpendicular**" toggle. When darkened, as shown here, linear distances are "snapped" perpendicular to the current construction plane.

Zoom options. Click here to see a variety of zoom options.

Snaps. They're currently set to "none" in this case, but these two icons give access to object and grid snaps that may prove useful.

"Vertex Editing"

One of the interesting capabilities which Form•Z provides is sometimes called "vertex editing" or "tweaking". Tweaking means distorting objects by moving individual points. By setting the topological level to "Point", you indicate that any "Move" operations operate on only single vertecies. Now, click and drag any vertex which you wish to shift. (It moves parallel to the working plane, unless the perpendicular toggle is on.)

Step 2: Build your model

The Base Model

Each of you will be doing a separate project involving the design of a building just north of Gould. To simplify things, I have created a "base model" of the surrounding buildings. The standard base model is located on the Mac file server in the **Resources folder**. Copy it to your disk. Now open it using Form•Z and try out different viewing options to become familiar with them, BEFORE you try editing (that way, if you accidentally trash the data, you can just get a clean copy and start over). Try looking at the data using the various preset options, using different rendering algorithms, etc.

There are several techniques you can use to create your 3D model. It is <u>not</u> possible to simply "draw in plan and elevation" and convert the 2D drawings into 3D data (plans and elevations don't resolve certain ambiguities). You will, however, use 2D primitives as the basis for your 3D data. For example, you might be able to draw a 2D plan of your design and then take the 2D outlines for your walls and *extrude* them into the vertical dimension.

Some shapes won't easily extrude. For some of these you might find that the *revolve* operation, which creates axially symmetrical objects from their profile lines, can be used.

Finally, you might have to simply draw new data into the model, using the drawing tools found in the second and third rows of the menu. Depending on whether you're in 2D mode or 3D mode (selected by choosing one of the options from the first row of the menu) these actions will make simple 2D shapes or extruded 3D shapes.

Task

Using the available tools, build up your 3D model. How complicated does it need to be? Well, work around the problem, starting with the basic massing and roof forms, then adding detail (windows, trees, people, etc.) as time and interest allow. You need to do enough work to explore the program, and to demonstrate that you can make it do what you want it to. As with most design projects, you could spend many hours working on this, without doing an appreciably better job than someone who spends less time. Try to enjoy yourself, but also practice some discipline, so that the time you do spend is effective learning time, not just “futzing” time.

You will probably find tools and editing actions in addition to those described here. These are generally described in the <u>Form•Z User's Guide</u>, available in the CAUP library. There is also an extensive online help facility within form•Z. If you find a command you particularly like, make a note of how it was useful (see questions below).

If you do opt to create a more detailed model (by revolving your own trees, etc.), think about how you will handle all that data before execution of editing commands slows to a crawl. You might look into Layers, for some help. Simply adding data-intensive things last can also help a lot!

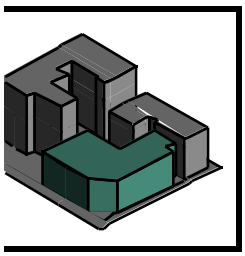
Step 3: View it!

Making the data is only part of the puzzle. After you have a model (and while you are building it) you will want to generate views of it. These are controlled through the **VIEW** menu. Remember that 3D programs specify the view in terms of the eye position (also called the "station point" in some systems), and another point at (or through) which you are looking (called the "focal point" in some programs, or the "center of view" [COV]). The **EDIT CONE OF VISION** option might interest you--try dragging the station point or center of view to new locations. The **DISPLAY** menu will probably be of interest here too. (Please note: you can't cast a shadow without a light-source! and in form•Z you have to enable shadow casting for the light **AND** the rendering algorithm--hold down the option key when you pick "shaded render" to make that setting).

Task

Explore your model using the different viewing tools. Do you feel that you really have control of what you are going to see? Keep notes about what's missing or hard to do so you can answer question #4. Print at least three of these views to turn in.

Form•Z-to-Word (or...) Image Transfers



For additional discussion of this topic, refer to CompuFacts #4.

This discussion is not just pertinent to getting from Form•Z to Word. It also relates to getting any color data, especially variable color information, into black and white or gray-scale documents, and it addresses what a 3D program generates as a 2D image, especially as information for subsequent editing.

Since the Form•Z drawing, like a spreadsheet display, is only one representation of the model (like a photograph is one of many possible photographs of the same reality) "data transfer" probably means "image transfer" rather than referring to transference of the actual 3D data. [Transfer or exchange of 3D data follows rules very similar to those for exchange of 2D CAD data.]

What Form•Z does do is "**SAVE AS**" (in the **File** menu) the 2D screen images to PICT files (the most common Mac graphics-file format). While this is good, we need to be aware that the rendering algorithm chosen determines what *kind* of PICT file we will generate.

> ☛ *It is unfortunate that the SAVE-AS "FORMAT" pop-up-menu, which offers several choices in addition to PICT, makes no distinction between which of those options pertain to saving a 2D* image *and which are saving 3D* data, *which can be quite confusing! DO remember to save your data as normal form•Z data too!*

PICTs: Raster or Vector format?

PICT files

> ☛ *PICT stands for "picture". The PICT file format, which is defined by Apple Computer, is the format used to store graphics on the Macintosh clipboard. This is why it is so commonly accepted by Mac programs.*

Macintosh PICT files can contain either purely raster or object data, or a mixture of the two. The contents are determined by the program which generates the PICT.

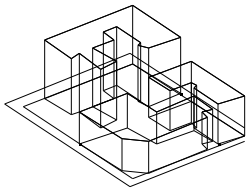
Form•Z, in turn, has several rendering algorithms which can be used to generate the image, some using raster logic, and the others using object or vector logic. The **DISPLAY** options for renderer select among the choices, but activating shadows (for instance) might cause a switch from an object to a raster rendering algorithm. Based on what kind of data the rendering algorithm generates, the **SAVE AS** formatting option PICT will create **DRAW** or **PAINT** data. The only way you can know for sure what you will get is to experiment a bit.

Picking your PICT type

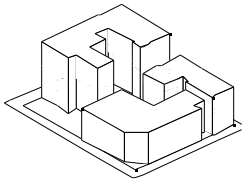
The type of PICT image that you will need for a particular task will depend on that task as well as the complexity of the 3D model, the kind of final output or display and so on. .

Selecting color depth in form•Z

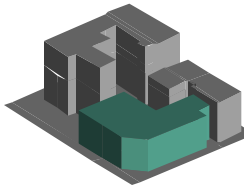
The "Display" setting interacts with the current "display depth" (set using the **IMAGE OPTIONS** menu item at the bottom of the display menu) to determine what color information is included in the PICT. This option may be set to 8, 16, or 24 bits per pixel.



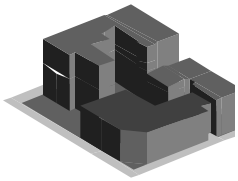
Wire frame



Hidden line



Shaded



Shaded & Edited in Word

Study Questions

1. 3D *data* generally occurs in one of three forms, which can be thought of as representing 1D, 2D, and 3D geometries: What is the name for each? 1D: _____, 2D: _____, 3D: _____.
2. Which of the above types of data is required for "Boolean Operations"? _____.
3. The process of converting (squashing) 3D coordinates into 2D coordinates is called what?
4. Identify at least two ways of “working smart” in 3d programs such as form•Z.
5. What is the most common cause of problems for programs when doing hidden line/surface removal?
6. The person next to you is using form•Z. They are looking at a plan view of their model and swearing. It seems they were working in axonometric view, where they drew one building block, then drew another one on top of it, but when they changed to plan view it moved. What haven't they figured out? Explain.
7. Most 3D programs utilize the idea of a "view vector" in some form or another. This view vector is the line from the viewers eye (or camera) to the center of the view, or focal point. In a perspective view, the view vector determines whether the perspective is a 1-point, 2-point, or 3-point perspective. What orientations of the view vector give each type of perspective? (give examples).
8. Most polygon-based modeling programs create models by addition, that is, each primitive or tool creates a visible element of data which is added to the existing data. While you can select and delete entire polygons, few of these programs support the idea of "negative data", tools with which you can remove part of the model. Does form•Z?
9. What is the name for the modeling process which creates 3D data by extending a 2D view (such as a plan) into space in a specific direction?
10. If you are trying to model a smooth cylinder, you may need to choose between more data and a more complex rendering algorithm. Which would you choose, and why?
11. True or False? Programs capable of “shading” the surfaces of a building must also be capable of “casting shadows” on those surfaces?
12. Let’s say you wish to include a Form•Z "rendering" in a presentation which will involve layout of several separate images in one document. Name two Form•Z options which you should think about and state why.
13. What were the three generic reasons to select a raster PICT file as a transfer medium for a graphic?
14. What were the three generic reasons to select an object PICT file as a transfer medium for a graphic?

The Exercise

Follow the instructions in the previous sections to create a "proposal" for an addition to the College Inn. Print appropriate hardcopy images and answer the questions below.

Your electronic submission should contain ...

1. A copy of your finished Form•Z document.

Your hardcopy submission should contain ...

2. At least three black & white or color laser prints showing the model from different view points. One of these should present a “street level” view of the building.
3. Answers to the questions below (done in Word).

Questions

1. What kind of 3D data does Form•Z utilize: wireframe, b-rep, or solid?
2. What does the term "topological level" refer to?
3. In a perspective view, the view vector determines whether the perspective is a 1-point, 2-point, or 3-point perspective. What orientations of the view vector give each type of perspective? (give examples).
4. What purpose do Form•Z's "Reference planes" serve?
5. If you use the SAVE-AS option PICT to create a file, can you use it in another session of form•Z later on if you decide you want another view point?

For Extra Credit

Use Form•Z's SAVE AS operation to export at least three color images of the model. Open these using Canvas, Word, or Page Maker and arrange them on a single 11"x17" page, making a one-page presentation of the design. Add lines and/or text as needed to enhance the drawing. Turn in a color print-out of the final layout, and include the Canvas file in your submission folder.

Canvas Ojbect Type Icons

Picture Object

Painting

Canvas is a 2D dual-mode program. It is one of the options here because it "does color". One particular thing to be aware of is that Canvas treats raster data in two very different ways, as a "painting" or as a "picture object". The difference is that when you scale a "painting" any data which is not actually visible on the screen is discarded, so if you scale it way down and then back up again, you'll find much of your image missing. A "picture object" (or Pixmap) on the other hand, preserves all it's image data when scaled up or down, so it will print better (why?). When you first open or import ("Place") a PICT file saved from form•Z, it will be a "painting". Use tje Object | Object specs ... dialog box to change the type of the object (click on the paintbrush and a palette of options will appear).