Digitally Augmented Sketch-planning

By

Kenneth Goulding

Bachelor of Architectural Studies
University of Cape Town
Cape Town, South Africa (1998)

Submitted to the Department of Urban Studies and Planning in partial fulfillment of the requirements for the degree of

Master in City Planning
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2002

Author _____
Department of Urban Studies and Planning
May, 2002

Certified by _____
Professor Joseph Ferreira Jr.
Department of Urban Studies and Planning
Thesis Supervisor

Accepted by _____
Professor Dennis Frenchman
Chair, MCP Committee
Department of Urban Studies and Planning

Abstract

Digitally Augmented Sketch-planning

by

Kenneth R. Goulding

Submitted to the Department of Urban Studies and Planning on May 16, 2002 in Partial Fulfillment of the Requirements for the Degree of Master in City Planning.

ABSTRACT

While many aspects of the planning profession have changed radically in light of recent technological advances, the practice of sketching plans has remained largely unaffected. There may be good reasons for eschewing computers in the design arena such as that their use may detract from the liberty of the design thinking process. This thesis suggests that this reluctance may be overcome by changing the practice from one of emulation with digital tools to one of "augmentation".

In addressing a perceived need to bring computation to the design table a solution called the "digitally augmented sketch planning environment" (DASPE) has been developed. Making use of video projection, DASPE augments the design space with digital visualization and analysis tools and allows planners to sketch using either conventional media or a pen stylus on a digitizing table. Plans can be sketched in the conventional manner, then "hardened" into three dimensional computer models without the need to leave the design space.

Thesis Supervisor: Joseph Ferreira Jr..

Title: Professor of Urban Planning and Operations Research.

Contents

List of Figures

Introduction

Planners need tools to help them think

Modern designers often reflect upon the history of design with a touch of nostalgia and a great deal of respect. We are amazed at how technically advanced and yet ecologically sensitive old solutions to design problems seem to be. However, this should not be surprising when one considers how much time has been put into such solutions. In most cases, many generations were able to give thought to the same design problem, to test it and to modify it. Importantly, the problems and the technologies available to solve them remained the same.

Today we are faced with rapidly evolving technologies that change the way we build and use cities. There is little time for any new technology to become "tried and tested" before we discover another. The design demands of implementing any such new technologies should therefore be immense, and one would expect the task of the city planner to be an impossible one. He or she cannot hope to give the same amount of consideration to a design problem as could be given when cities and technologies evolved more slowly. "This problem can be traced back to the institutional problem of the separation of the planning, architectural, and engineering professions where design at the human scale, no longer having a single, powerful patron, fell between the cracks." (Miller 1988)

As a result, planning efforts will often fail in many respects (even if they succeed in others) simply because not enough thought can be given to all the necessary aspects of the design. In addition, constraints on time and budget compound this already challenging task. One approach to this problem is to abstract the design process (often through use of analogy) so that the task becomes manageable. In addition the planner may employ powerful tools to help with providing quality information, visualizing the implications of decisions, and communicating the logic behind them to other planners, the clients or the public. Such tools have, as yet, done little to enable the planner to provide design at the human scale.

A number of tools exist that are capable of assisting the planning process. However such tools are seldom "pure" planning tools. Planning is a close enough cousin of architecture and engineering for planners to use tools that were designed for those purposes. This tendency is compounded by the need for planners to share drawings with these professions. Furthermore, the planning profession's relatively small user-base does not encourage the creation of specific tools for planners when others will suffice.

Ultimately the use of such tools does little to improve the efficiency of the planning process. In fact it often introduces new inefficiencies associated with producing drawings to be compatible with architects and engineers. The production quality of a planner's output may be much improved (in the sense that the drawings look more professional) but much subtle information that exists in hand-sketches may be lost in the process. As a result many planners still like to produce drawings by hand, and the decision to use computers is often based on competition with other firms in terms of what the clients expect to see.

In this context, the computer is seldom used as a design tool, and is not associated with the design process so much as with a drawing production process. This paper looks at the need to develop computer-based tools that more closely identify with the design needs of the planner. Hopefully, by providing new tools that assist design thinking, visualization and communication, planners may begin to confront the super-human task of designing the city at a human scale.

Introducing DASPE

> *"The planner... sees and understands only those things for which he or she can provide expression."* (Rowe, 1985)

The digitally augmented sketch planning environment (DASPE) developed for this thesis creates a context in which planners may sketch their ideas using both conventional and digital media. Combining the computer's visualization and analysis tools with the freedom to draw with pencil on paper establishes and environment conducive to expressing and understanding design concepts.

Making DASPE

The environment is created using a ceiling mounted video projector that projects onto a large Calcomp (Drawing Board II) digitizing table below. The projector and digitizer are calibrated to ensure that input and display are collocated. This is done by simply projecting the image of the computer screen onto the table, then mapping the whole surface of the digitizing table to the whole screen. Once this is done one only has to zoom the projection and shift the table until the projection fills the active part of the digitizing surface exactly. Input and display will then occur in the same place so that the digitizer may effectively be used as a large touch-screen. A wireless pen stylus (from Calcomp) is used to sketch on the surface of the digitizer. The stylus is pressure-sensitive and senses variable degrees of pressure applied to its tip.

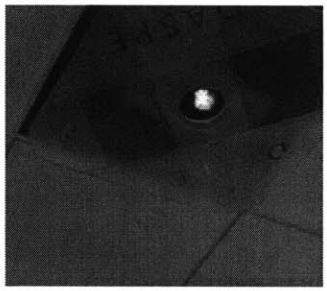


Figure 1: The projector mounted in the ceiling shines through the custom ceiling tile (figure 2B:5).

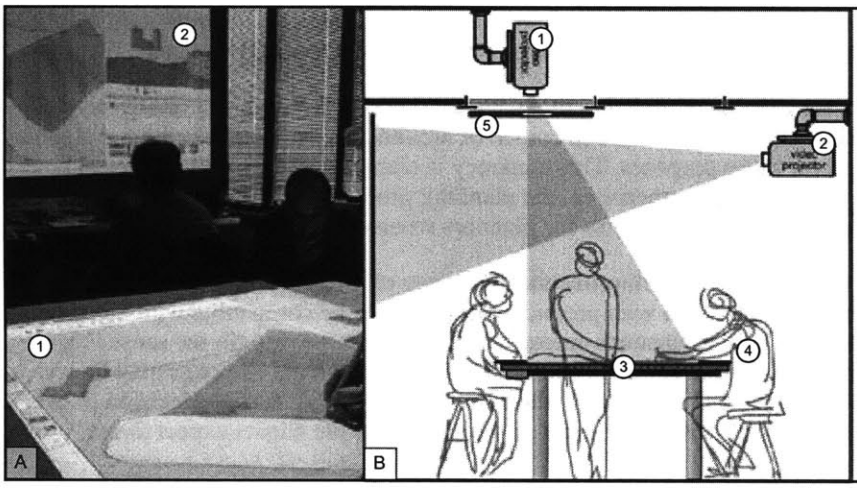


Figure 2: These images show the environment created with two projectors (one in the ceiling (1), the other on the wall (2)). The Calcomp digitizing table has been removed from its original stand and made to lie horizontally as a flat table (3).

Designers and other participants may sketch on the table and view the results of their creations on the wall opposite the main sketcher (4).

Figure 3: Running a MEL script to create a human figure.

The Software

The software component for DASPE has been created in a program called Maya (from Alias Wavefront). Maya is a 3D modelling and animation package used to create animated motion pictures (such as *Shrek* or *Toy Story*). It was chosen for this interface as it comes with a powerful scripting language called MEL (for Maya embedded language). This allows the interface to be customized and new tools to be made so that a working prototype for a 3D software environment can be created in far less time than would otherwise be feasible. The image to the left shows a simple MEL script being used to create a representation of a person by moving the vertices of a cylinder.

Conventional and Digital Media

DASPE encourages conventional drawing media to be used alongside the digital media on the table surface. At any point during the design process sketchers may switch from one medium to another or use a combination of both.

In *Design Thinking*, Rowe (1985) describes the design process for architects and planners as one consisting of many stages and incorporating a variety of design heuristics. This process is not linear and may require many iterations. It is unlikely that any particular tool, medium, or perspective will serve the designer's needs in all iterations and he or she may struggle for inspiration. Providing a number of media along with the ability of computer graphics to easily shift between scales helps to create an environment in which the designer can maintain forward momentum - or step back and reflect as the need may arise.

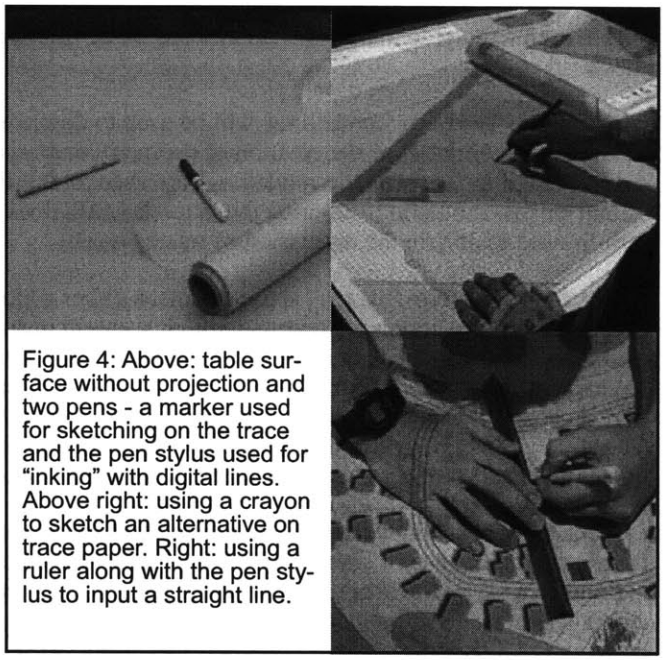


Figure 4: Above: table surface without projection and two pens - a marker used for sketching on the trace and the pen stylus used for "inking" with digital lines. Above right: using a crayon to sketch an alternative on trace paper. Right: using a ruler along with the pen stylus to input a straight line.

Vertical and Horizontal

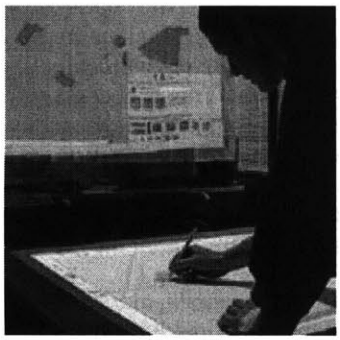


Figure 5: Drawing on the table; viewing on the table and wall.

Projecting onto the wall is desirable for two reasons: A) it is more visible to a larger number of people and therefore necessary for larger presentations and B) it provides the correct orientation of the perspective and elevation views to all present.

The projector in the ceiling is used to project the plan view and tool-bars onto the table (see image B below). Ideally (if a suitable video card capable of splitting the display across two screens were available), a second screen would display the elevation and perspective views on the wall (see A below) freeing up a larger workspace in plan. Currently the same view is projected on the table and on the wall.

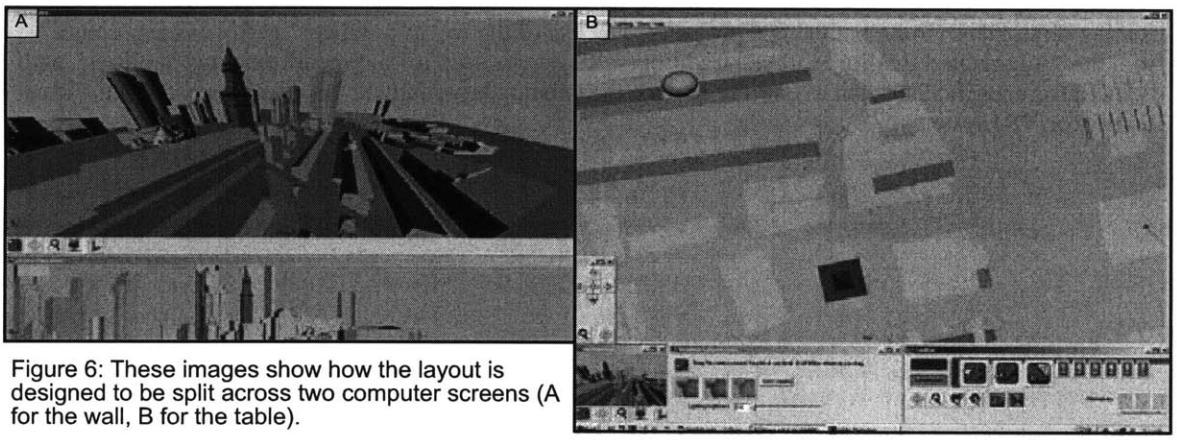


Figure 6: These images show how the layout is designed to be split across two computer screens (A for the wall, B for the table).

Introduction to terms used:

This is an attempt to clarify the distinctions made between certain similar terms as they are used in this thesis.

Modelling: The term "modelling" will be used to describe either the simulation of real-world processes in a computer "model" or the creation of geometry that represents a physical "model". The terms "computer modelling" or "scenario modelling" are used to describe the former process while "geometric modelling" or "3D modelling" describe the latter. DASPE does not make use of "scenario modelling," and the term is used mainly in the discussion of related work.

Sketching: The term "sketch" is used to mean either a loose way of drawing lines with a pen (or pencil) or a more abstract notion of creating a rough scenario in such a manner as to allow decisions to be readily changed. The latter does not necessarily require lines to be drawn (neither with a pen nor any other input device) and may instead make use of a number of points specified by a mouse pointer for input. "Sketching" on computers in this fashion is often frowned upon as being clumsy, restrictive and lacking in expressiveness, but may be desirable for such uses as scenario modelling or automating repetitive tasks. Indeed the term may be broadly used anywhere that the common theme is one of transience or impermanence.

Users, Sketchers and Designers: The term "sketcher," is preferred to "user" in most cases. A "user" is one who makes use of computer software; most software developers will discuss their software as it pertains to a user. However the term "user" also carries connotations of identity for those involved with computers in terms of being considered a computer "user" or a "non-user." In making the distinction between DASPE's approach of allowing "non-users" to sketch with the interface, the term "sketcher" is preferred in referring to all users of the interface. Sketchers may be both designers or non-designers. The term "designer" is often used to distinguish a certain class of sketcher.

3D: The term "3D" is used to describe software and hardware that make use of three dimensional representations of objects and scenes. Characteristic of these 3D applications is the ability to dynamically change one's view so that the three dimensionality of such objects can be appreciated. This is necessary because the view is seldom actually seen with "depth perception" or "in stereo" as both eyes view exactly the same information on a flat plane (3D software most often makes use of two dimensional perspectives when showing its 3D info). The term will be used here to refer to such 3D software and its associated components. The term "three dimensional" on the other hand will be used to refer to truly three dimensional environments such as the physical environment which 3D software is used to emulate.

Conventional Media: This term will be used to refer to media such as paper, pens, pencils, markers, crayons, trace-paper and other paraphernalia commonly used in creating sketches and drawings. Such media may be used alongside "digital augmentation" with digital media in the DASPE system.

Digital Augmentation: This term is used in this thesis to refer to the addition of digital media to the existing physical environment using a projector. The use of the word "augment" in this fashion comes from the "Augmented Reality" (or AR) tradition. AR systems seek to "augment" existing "realities" with digital information rather than to entirely replace the existing environment (as is the practice with Virtual Reality (or VR) systems).

The substance of digital graphics

It would seem to make sense that a sketch planning interface should make use of "sketching" as its chief input, but the nature of computation and digital graphics has not been conducive to allowing this. Indeed there are a number of reasons why the interfaces that planners use are so far removed from what might be seen as ideal from the user's perspective.

1. The amount of *computation* required to implement a natural interface, or any interface that puts user comfort above computer efficiency is invariably greater than an interface that focusses instead on using each cycle of the processor most efficiently to achieve the task at hand. Since any period of unresponsiveness from the computer may be deemed discomforting to the user, it could be argued that efficiency is paramount to user comfort. Only recently have processor speeds increased to the point where we can make use of more complex computation without a noticeable delay. Perhaps the most noteworthy amongst such improvements has been in the speed of 3D graphics rendering. 3D graphics is an inherently processor intensive task and these improvements have brought previously intolerable graphics computations into the realm of feasibility. The demand for better 3D graphics in personal computers has been derived largely from the 3D computer gaming industry, although many innovations have also been made by the US Department of Defense (McCracken,1997).

2. Gestural interfaces are inherently difficult to implement from a *programming* perspective. This is largely due to difference in interpretation between what the computer "sees" and what the user sees. Programming is generally difficult to implement in any situation where there are "grey areas" or ambiguity. Since software development decisions have typically been made by computer programmers, that which is easiest to program often has most often won out over that which makes the most "common sense."

3. The "user base" for planning software is not great and planning is not sufficiently dissimilar to architecture and engineering that it requires its own software.

4. Computer software is more likely to be accepted if it adds to an existing process without upsetting the status quo. A software package may offer the promise of easier production or new computations that can inform the planning process, but developers have wisely avoided the temptation to replace existing processes such as sketch planning.

5. We are good at learning new tools: people can learn by experimentation and, by observing the results of an operation, may become quite comfortable with using nonsensical gestures to communicate with the computer. For example, one thinks nothing of drawing a straight line (a radius, diameter or "bounding box" diagonal) to signal the computer to produce a circle. This is because the feedback provided shows a circle and not a line so this can begin to seem normal. The development of gestural interfaces has therefore not been essential in enabling drawing to be done on a computer.

6. Despite all the advances in technology, human cognition remains far superior to electronic computation at such processes as visual pattern recognition. It is therefore difficult to make the computer behave as a useful and seemingly intelligent entity where pattern recognition is involved. Computers have sensibly been employed in the realm of unambiguous and repetitive tasks that they may do far more accurately and efficiently than humans. However, since computer interfaces must involve communication between humans and computers, and since their use has extended beyond simple, unfuzzy graphics tasks, it will often be frustrating for the user if the computer does not attempt some "common sensical" computations so that it behaves as a seemingly intelligent and trustworthy entity.

There therefore exist rifts between what is computationally efficient, what is easy to implement in code, and what is simplest and most natural for the user. Occasionally, one will encounter a method that just happens to fulfil all three requirements at once. An example of this is the movement of the "Curiosity Camera" in DASPE from one position to another. This is programmed by continually halving the distance from the desired position until it is close enough to stop. This is extremely simple to compute, requires

no extra variables in the programming and has a pleasant motion that starts quickly as though pushed and decelerates towards the point of interest as though it is coming to a halt. However, for the majority of implementations, one has to choose between that which is easy to compute, easy to program, or easy for the user. In order to illustrate this, it may be helpful to look at the requirements for drawing a two dimensional circle in a gestural sketch interface.

By way of background for this discussion, it will be necessary to clarify the distinction between vector and raster graphics. Raster or "bitmap" graphics are comprised of a grid of "pixels." Certain color information is stored for each pixel so that an image may be represented by the grid. Raster graphics make use of the same kind of logic as is used to display graphics to the user on the screen. Drawing in raster-based applications (such as earlier versions of Adobe PhotoShop) tends to allow more free-form drawing, but does not lend itself to accurate drawing or tweaking. Vector graphics (sometimes also known as "object oriented" graphics) represent "graphical objects, such as lines, arcs, circles, and rectangles, with mathematical formulas." (Lycos Tech Glossary) The objects used in vector graphics may contain a variety of attributes that specify how they are to be represented. Other attributes may contain additional hidden information such as GIS data. Vector objects are described in terms of a number of standard "primitives" (such as ellipses, rectangles, lines, curves and points) from which all graphic representations will be made.

A "2D" circle is the simplest vector object that can be represented by computer graphics in terms of the number of variables used to describe it. Any circle may be stored simply as three variables: its x and y positions in the Cartesian plane and its radius (or r). By comparison, a line requires four variables (x1,y1,x2,y2) while a square must require at least a specified rotation. In addition a point is effectively simpler than a circle (it contains only x and y), but is not "renderable" except as a circle or a more complex shape. This concept may seem strange to anyone who has had to draw circles with a pen and paper where it is necessary to use a tool such as circle stencil or a pair of compasses in order to achieve a consistent shape. Indeed, when drawing by hand most other shapes are easier to produce than an accurate circle.

Drawing a circle freehand is an interesting process: it is indeed very difficult to draw a continuous line a certain distance from a center point and ensure that it is smooth and regular. Many sketchers will draw circles iteratively, starting with a rough circle and then refining it with each new stroke drawn. The later strokes are often made firmer than the earlier ones so that a circular shape is most evident to the eye and the rougher lines may be ignored. However, while the human eye may immediately recognize this sketched form as a circle, it is not as easy for a computer to be made to do so. Indeed the implementation of a "common sense" interface must accommodate many such situations in which the "eye" may assume something to be obvious, but the computer may be entirely fooled unless recognition for that particular pattern has been encoded.

For DASPE's circle recognition algorithm, I have employed Lew Hopkins method for determining a circle (Hopkins, 2002) : it is a series of sketched points that, when divided into six equal segments can form a "Jewish" star that has roughly equal sides (see figure 1). This works very well when one considers how simple it is to implement computationally - but only provided that the user finishes the circle near its start point. Many users may however draw the multiple iterative rings described above, or simply end the circle a little further around the circumference than the point where it started. The Hopkins algorithm will try to divide the line into six even parts, but unless the line is a single circular form with a start-point near its end point, it will not be recognized as a circle. In order to accommodate all the human conceptions of something as simple as a circle, one may need to add further logic. Perhaps the circle algorithm should watch to see when the circle can first be considered closed, mark this point and later run the algorithm looking at the sketched curve only up until this point. However, while this may

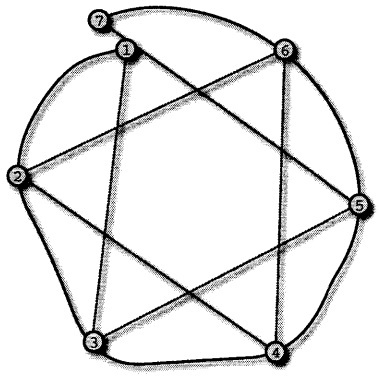


Figure 7: method for circle recognition

ensure that all circles are recognized as circles, there is a danger that other shapes may be "recognized" as circles too. For example a square will most often also meet the requirements of Hopkins' algorithm. An interface that automatically converts a square into a circle will be annoying and is unlikely to earn the user's trust and confidence. In DASPE this problem is overcome (for the most part) by checking two things. Firstly there is an algorithm that looks for corners. Unfortunately in reality any curve drawn with a stylus will have small kinks and since the smoothing algorithm does not successfully straighten these out, the corner-finding algorithm will often find corners in a circle! Therefore it is also desirable to check the average change for all angles that aren't considered part of a corner. After the corners have been removed from a square, most other points will be straight in relation to each other. A circle on the other hand will run up a far higher average change in angle between points, and can thus be recognized once a tolerance is determined (through numerous trials) to separate circles and squares. However, even so, the computer can still get confused between circles and squares or by lines that aren't supposed to represent either primitive exactly. The variability between drawn strokes that to the human eye may appear unambiguous is too large to always ensure correct interpretation.

So, rather than draw a circle in the fashion all people know by common sense it is far simpler computationally and less ambiguous to draw a circle by specifying first its center point and then dragging away from that point to set its radius. Furthermore, since most computer graphics packages rely on mouse input, it would be less accurate and slower to draw circles in the conventional fashion. It is however necessary to provide a specific tool for drawing circles since a circle is really being input by drawing a *line* with the mouse. In addition, it may at times be desirable to draw a circle by dragging from its edge, and many computer programs draw circles or ellipses to fill a "bounding box". In each case, a new tool state must be specified for each input since the mouse drag used to specify the circle looks only at two points - the start and end of the drag operation.

Indeed, for almost all drawing tools used in conventional CAD or graphics programs, input takes the form of a single click, series of clicks, or the start and end of a drag operation. All other intent is specified by tool selection or through keyboard input. Rarely does the software "care" what happens during the drag operation.

A gestural interface on the other hand must rely almost entirely on what happens during that drag - not just on the positions of all the points plotted during the drag, but also on their order and sometimes also on the speed at which they were drawn.

Consider the popular gestural interface used on the Palm handhelds. The letters of the alphabet and a number of other commands and symbols are all represented by the points along a drag operation. In this case an extremely difficult problem of character recognition has been greatly simplified by the requirement of a compromise between what is easy for the computer and what is easy for the user. The computer requires that the user learn a fixed way of drawing each character. Note however that "fixed" here refers to the human perspective and a great deal of computation must still be conducted in order to decipher those lines that are all the same thing to the human eye. This has proved to be a happy and very popular compromise: a compact and natural input device for the user; a reliable and unambiguous set of patterns for the computer.

Components

Lines and Curves

Circles alone would obviously not be the most efficient way to represent any two dimensional drawing. Curves and lines on the other hand may be used to represent almost any shape, especially if they may be drawn as multiple lines and curves used to enclose "fills." Curves and lines are both represented by a number of points along the line or curve. These points are known as control vertices

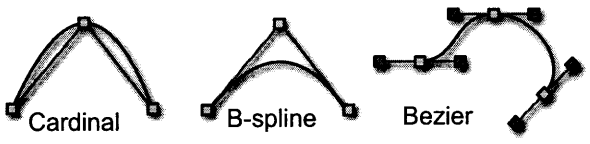


Figure 8: three ways to represent a curve

(or CVs). Curves are stored in the same way as lines but for a few extra control vertices used to determine their curvature. B-spline and cardinal curves make use of one extra CV for each segment. This influences the curvature of the line either strongly (for cardinal curves) or more loosely (for B-splines). Bezier curves use two control nodes or "handles" to describe the curve. These handles and CVs may be used to represent any possible curve - often with fewer points than one would imagine. Many graphics programs make use of sophisticated algorithms to represent a drawn stroke with such vector curves and some even allow the accuracy of the representation to be tightened or loosened to create smoother curves or curves that follow the drawn line very closely. Some programs such as Macromedia Flash will also attempt to interpret whether or not a sketched line is intended to be straight, or whether it should result in a circle, a square, a rectangle or some other regular shape. This may also be turned off in order to draw curves that would otherwise be wrongly interpreted.

Triangles

In the same way that curves may be used to outline any two dimensional form, triangles can be used to represent any 3D geometry. In truth, triangles may not perfectly represent a rounded object such as a sphere, however they may be used to approximate its form to varying degrees of accuracy. The main reason triangles are valued for representing 3D geometry is that a triangle's points will always lie in the same plane No other shape shares this property, and it allows the triangle in question to be rendered with the appropriate amount of light for the plane in which it lies. Seen together, all the triangles will appear to represent a solid, shaded entity such as the shaded spheres on the right in figure 9.

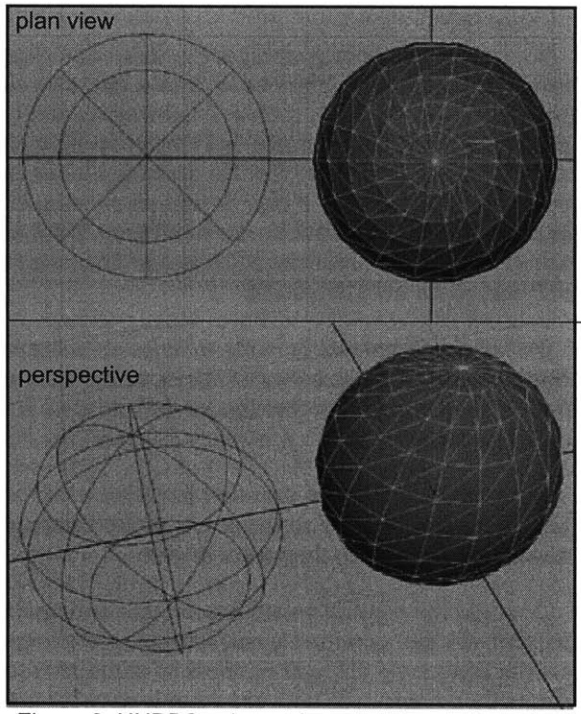


Figure 9: NURBS sphere triangulated and shaded

Polygons

Polygons are similar to triangles, but, as their name would suggest, may consist of more than three sides. Polygons are ultimately "triangulated" in order to form triangles. This ensures that all faces of the object then lie in a single plane so that they may be rendered. Some software packages will take a different approach and will prevent the creation of "non-planar" polygons. In such a case there is no need for triangulation.

NURBS

NURBS stands for Non-Uniform Rational B-Spline. "[NURBS] surfaces are webs of interconnected curves" (from Maya Documentation). In order to be rendered, NURBS surfaces must also be triangulated in a processor intensive process. There is a trade-off between using more triangles for increased accuracy and greater smoothness and the time it takes to create more triangles (through tesselation) and to render them. Figure 9 shows a NURBS sphere that is represented with a web of lines on the left. Before it can be shown as shaded (on the right) it must first be "tessellated" into a number of triangular facets.

Rendering

The term "rendering" refers to the process of generating a "bitmap" or "raster" image from the 3D vector geometry. It is most often used to refer to the final output of the 3D modelling software, but is also used here to refer to any representation of 3D geometry on the screen. Vector geometry cannot after all be seen unless it is "rendered" to form a bitmap image such as can be displayed on a screen. In figure 3, both

spheres would be considered rendered by this definition: the NURBS sphere as a wireframe; the shaded polygonal sphere as both a flat shade and a wireframe together. Note that if one were to "render" this scene as the term is used in Maya, both spheres would be shown identically by default.

Rendering Speed

Rendering is a processor intensive task. Even though processor speeds are becoming increasingly impressive, 3D rendering remains an area in which one may very quickly observe a slow-down in system performance. DASPE relies on real-time rendering in the model viewer within Maya. This model viewer uses many of the "hardware rendering" techniques available on many modern video cards which ensures that an incredibly large number of polygons may be rendered without any apparent slow-down (this means all those polygons are being rendered at least 30 times every second). However, slow-down may become apparent when the polygon count becomes sufficiently high, when lights are to be rendered or especially when "depth map" shadows are projected. The main solution to speed problems is therefore to reduce the number of polygons (or more specifically, triangles) used wherever possible and also to turn off more processor intensive tasks such as lighting and shadows when animating the scene.

Texture maps

The term "texture map" refers to a bitmap or raster "texture" that is "mapped" onto 3D geometry. A texture is generally an image that is repeated a number of times in order to entirely cover the 3D object. For example, an image containing horizontal and vertical lines may form a texture for a building. In DASPE such images repeat once for every storey of the building and a number of times horizontally so that the entire building is covered. Since one is applying a two dimensional texture onto a 3D geometry, a variety of "projections" may be used in order to determine the 3D orientation for the texture maps used. DASPE makes use of "cylindrical projection" for the buildings.

Maya allows texture maps to be rendered using the processing power of the graphics card. "Hardware rendering" in this way is extremely fast and ensures that, in terms of speed, texture mapping is far preferable to representing the same information with 3D geometry.

Tweaking

One of the main advantages of using vector graphics is that it is easy to make changes to a drawing by tweaking a number of variables. For example, all line weights may be increased, or all textures changed for the selected objects. In addition, since all objects are represented with control vertices, their form may be altered at any point. Such small changes are known as "tweaks."

Related Work

"We shape our tools and they in turn shape us" (McLuhan, 1967)

The role of the city planner is ever-changing. This has never been more true than it is now in light of the availability of new tools that both expedite existing processes and enable new ones that could never before have been feasible. It will be curious to watch how the planning profession takes on new roles in light of the new tools available. Here follows a discussion of a number of innovations that have changed the face of planning in recent years and some that are only beginning to make their presence felt. Such a discussion is important to establish an understanding of the technological environment in which DASPE is designed to exist.

CAD

Most planners make use of "industry standard" drawing tools such as MicroStation and AutoCAD. These form a small part of the broader CADD (Computer Aided Drawing and Design) field. It is a highly innovative field, and has revolutionized the design process for many industries. However since most of these innovations assume a product - and most often a physical 3D product, few of them are well suited to planning. Furthermore CAD drawings tend to become too rigid, too quickly. Even though they may be easily modified, it is often easier to start a drawing from scratch. CAD, with its promise of efficiency and ease of modification discourages this practice.

Many newer innovations inspired by the CAD industry may do more to improve the planner's task. Sketch interfaces such as SKETCH and ErgoDesk (discussed below) as well as speech control interfaces that simplify the input of commands are some examples. Some CSCW (Computer Supported Cooperative Work) applications developed for the CAD industry may also be useful to planners although the benefits are unlikely to be as great as for other practices.

GIS

GIS is an innovation so useful to planners, that many people consider it to be a planning application first and foremost. Geographic Information Systems were, however, initially developed for the purpose of cartography and the practice of linking graphic elements to an attribute database is so powerful that GIS is now employed for a broad variety of purposes.

With the right data, GIS can allow planners to make informed decisions from remote locations and with limited time where previously they would have to visit a site, solicit information at great expense or simply guess. It should be possible to have all the information necessary for decision making literally at one's fingertips. However since GIS was not designed for planning, and the information available has been gathered for a variety of purposes, it is often necessary to process the information before it can be used to assist design decisions.

Despite this requirement, it is clear that no planning support tool would be complete without a GIS component to it. However, while I will demonstrate the use of GIS in DASPE, it will not be the focus of this thesis and will often be employed as "vapor-ware." I have made this decision because I am confident that high quality work is being done in the GIS field to equip planners with the tools they need. It is the integration of these tools into a form more useful to planners in the design phase of their work that has been lacking.

Of particular interest to this thesis is the work by Raj Singh in using GIS in the sketch planning process. This work will be considered in more detail later, however it is worth noting here that such uses of GIS may be still more useful when incorporated into an interface such as DASPE. This may be done in an iterative process incorporating traditional sketch planning techniques and the deployment of such GIS based tools as Singh's.

Another initiative of interest is the MIT City Scanning Project. Seth Teller's effort to create realistic 3D models of the city using photography and image-mapping techniques may prove an invaluable resource for planners. These models will provide a previously unavailable opportunity to visualize the city and the effect of one's interventions on the experience "on the ground". Intriguingly the technique of draping bitmaps over buildings to achieve a simplified, but convincing representation of reality is similar to that which I propose in DASPE as a "false-reality" used to "fill in the gaps".

Communication

Portfolio Wall (Alias WaveFront 2001) is a system that, like DASPE, uses Maya and a large screen with collocated input. However, the portfolio wall is not used for design - it is called "digital asset management software." Its purpose is to bring a number of designers or clients up to speed on the progress of a design - essentially replacing the pre-digital bulletin boards that served the same purpose. Users may interact with the touch screen and view any of a selection of models from different perspectives by gesturing with a finger. They may also attach notes to a model by writing on the screen.

As with DASPE, Portfolio Wall serves as a forum for sharing ideas and apparently aims to create shared-understanding. The wall may be used for group discussions about a project, but one of its main objectives is that of remote collaboration. It therefore differs from DASPE which aims primarily to generate discussion between people in the same room.

A CSCW system capitalizes upon the convenience and efficiency of computers in organizing and facilitating the sharing of work documents and drawings. It is becoming increasingly important when using such impersonal systems to encourage individuals to come together and share their ideas in a physical environment in order to ensure that shared-understanding exists. (Schrage 2000) Although DASPE is not demonstrated as part of a greater CSCW system, it could easily be incorporated into one, and would be the natural place for people to come together over a prototype.

3D design software

Most 3D CAD packages are primarily two dimensional drawing tools onto which a third dimension has been added. More recently, some packages have emphasized 3D design as their focus. Perhaps the most popular product in this niche, ArchiCAD (from Graphisoft) has been developed with the intention of streamlining the architectural drawing process through the use of intelligent functions that can draw walls complete with height information and "specs" from a simple line. Doors and windows may be added to the wall with the minimum amount of user input. While demonstrations of this sort of intelligent, dedicated software never fail to impress those who have "wasted" many hours drawing the equivalent manually, many architects complain that its intelligent functions limit the freedom of the architect to design more interesting structures than the functions allow. Overcoming this (if it is possible at all) may often result in frustration and inefficiency such that many of the gains of the intelligent software are negated.

Some architectural firms (particularly those who design more complex structures) therefore opt to use less "intelligent," less dedicated applications that give them more freedom to explore their creativity with a full array of 3D modelling tools. Such packages are far more difficult to learn, and are not closely tied into CAD systems so that few of the efficiencies possible in integrated solutions such as ArchiCAD are appreciated. However since they use software that was primarily designed for 3D animation (such as 3D Studio and Maya), this allows for the creation of photo-realistic 3D fly-throughs and other animations that set new standards for visual communication to clients.

It should also be noted that the freedom to design in 3D without the worries of maintaining a CAD drawing simultaneously allow for far more exploration and experimentation with architectural design. However it has been beyond the scope of most architects to learn these software packages. In addition, it is debatable whether or not the ability to create realistic and complex 3D models with computer tools (even once mastered) is as useful as traditional media in terms of thinking through a design. As a result, there is generally a division of labor between those who do the design and those who create 3D models of

it for visual communication. This amounts to a separation of thinking and production. It is not necessarily the most desirable situation, but has been the most practical given the options available.

In the particular case of Urban Planning there appear to be no dedicated tools that streamline the process of urban design as ArchiCAD has sought to do with architectural design. However planning firms have responded to the growing demand for visual communication by using 3D animation software, and spend needlessly large amounts of time in order to market their designs with these tools. This thesis will seek to create tools that streamline the planning process in the same fashion as ArchiCAD (and others) has done for architecture. However, at the same time it seeks to address the issue of flexibility that is manifest in an over-simplified program such as ArchiCAD by using an application (Maya) that allows those who have mastered 3D modelling tools to use them as their expertise will allow.

3D visioning systems

> *"Urban simulations; that is computer generated simulations of the built environment, are an effective means of improving the public's participation in the planning process".*
> (Bulmer, 2001)

The above is Bulmer's hypothesis in a paper submitted to the online planning journal. He makes a convincing argument for the potential of 3D graphics and visualization for communicating planning concepts to clients and the public. "Simulation" is used to refer both to the creation of 3D models for visualizing urban environments and to the use of computer models for scenario modelling. Both may be used to improve the quality of knowledge transfer. As Schiffer (1996) purports "information technology (IT) can help people to comprehend information, thereby delivering knowledge."

In order to capitalize on this potential, a number of commercial 3D visioning systems for cities have recently been made available to planners. These discussed here were recently presented at the American Planning Association Conference (2002). They are mostly focussed on representing existing conditions from information in a database, but may also be used for design visualization by modifying the data used. For DASPE these packages may provide an invaluable resource in terms of creating 3D *context* from GIS data.

SiteBuilder 3D from MultiGen-Paradigm (www.multigen.com) uses database information to automatically generate photoreal 3D environments. This may take the form of terrain data, point data for objects such as trees and lamps and polygonal data for objects such as buildings. The program is marketed as an extension to ESRI's ArcView. Models of buildings may make use of simple extrusions with generic texture maps, or predefined building shapes (used mainly for houses). The package also includes software for creating more complex 3D objects and applying texture maps created from photographs of the existing buildings.

CommunityViz (www.communityviz.com) markets MultiGen's software along with a "Scenario Constructor" application that allows the user to consider visually the impacts of decisions made. In this case the visualization may extend beyond a representation of the physical to representations of more abstract data such as the results of a suitability analysis.

VantagePoint (www.vantagepointtechnologies.com) take a similar approach although they do not provide scenario analysis tools, and concentrate mainly on visualization. Their package, called Perspective, allows some design to be done in a perspective view: for example, one may place a certain style of tree at a certain point by selecting it from a menu then indicating its desired position in the view-port. Their focus, however, seems to be on collection of data into large, comprehensive databases that include models for every building in the region of interest to as high a degree of detail as is available. The database may also contain temporal data so that various models may be stored to record the changes in a building over time.

All these programs make use of very similar 3D data-representation and visualization techniques. Many are similar to those used in DASPE, but there are also some differences.

Point data.

Most of the objects represented by point data are modelled either as a cross-form or as a head-on texture map that always points toward the user and can never be seen edge-on. The cross method has long been used by the makers of physical models where a simple 2D profile of an object such as a tree is cut out of card or balsa-wood and combined with a second profile placed at 90 degrees. This ensures that the 2D profile can never be seen edge-on without seeing the second profile face-on. The head-on method works by continually rotating an object to face the camera although obviously this happens only in the horizontal plane and it does not also rotate vertically to face a camera above it. This therefore works best for smaller objects seen at eye level.

DASPE does not make use of the cross method, but instead uses Maya's modelling tools to create simple three dimensional models of trees, lamps and other objects. SiteBuilder and Perspective make use of true 3D models only for more complex objects that are externally generated and accessed through libraries. It should be noted that 3D models are not stored only as point data and may therefore have a specified rotation so that, for example, street lamps may lean out over a street.

Polygonal Forest Objects

In SiteBuilder, trees may also be specified as polygons that represent a wooded area. Large clusters of trees are often perceived as an edge to a space and it would be inefficient to represent them as individual trees so large bodies of trees are instead represented as a single forest object mapped with a multiple tree texture. This successfully describes the tree-body as an edge and represents its texture. It should be noted that if the user is allowed to fly over the top of such a forest, it may appear odd.

DASPE deals with trees as either individual objects or as lines of trees. The scale of visualization for which DASPE is designed does not require the sketching of trees in such large bodies as cannot be represented individually, however such techniques would be ideal for showing context around a site. There is no tool for creating such tree bodies in DASPE. It is assumed that packages such as Multigen's would be used to create existing site conditions from GIS data for use as a base model.

Ortho Photos and Terrain

Terrain is represented in both SiteBuilder and Perspective as a triangulated mesh onto which an ortho photo has been draped. This is a common practice and works well for representing landscape. Cars and other three dimensional objects, may however appear squashed onto the pavement as they are not represented by the geometry of the mesh. Furthermore, while buildings may be included in the terrain model and the ortho photo accurately aligned such that each building is draped with the appropriate texture, there are a number of problems associated with representing buildings this way. The most obvious is that the facades are not represented, and appear as vertical streaks of whatever color happens to map to that edge. This is because the ortho photo is "projected" from above. What's more, the perspective on an ortho photo is such that the roofs of tall buildings will have shifted out of alignment with the base. In ortho photos it is the base and not the roof that represents the actual location of a building. Therefore, the ortho-photo will often appear to be out of alignment because the roof will appear to spill off the edges of a building extrusion. As a result this technique is undesirable for models that are viewed from the ground level. However despite these shortcomings, the method is quite remarkable at realistically representing the context when seen more broadly.

More detailed models

Areas that are intended to be viewed from the ground require more detail. This may be achieved in 3 ways: by creating all the building's components from 3D geometry, by applying a generic texture map (as in DASPE), or by using actual photographs of the building in question and mapping these onto a basic 3D

geometry. The latter method is desirable in most cases since fairly realistic context may be quickly generated while keeping the polygon count low. MetaCreations' Canoma (www.canoma.com) was ideal for quickly creating context from oblique aerial photographs. It is a great pity that this product is no longer available at the time of writing, however a more expensive and more complex alternative called PhotoModeller (http://www.photomodeler.com) could probably be used in the same way.

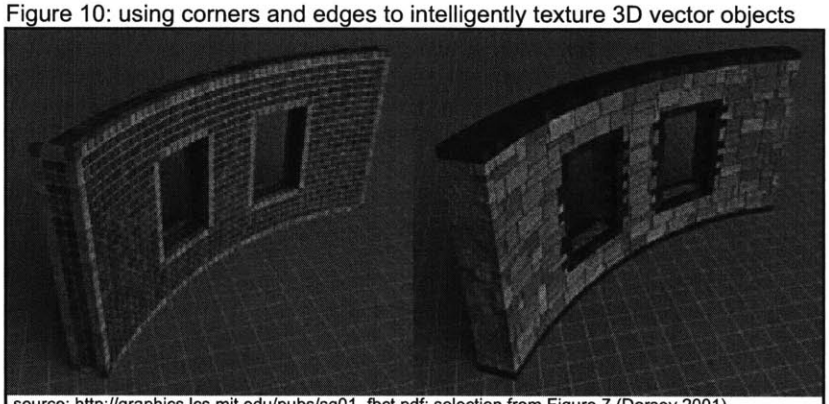
MultiGen provides an application called "Model Builder" that allows textures to be mapped to models such that they may be used in its visualizations. They have some fine demonstrations of its use, however, this is very labor intensive and one would be unlikely to see its use in generating context for an individual planning project. VantagePoint's solution to create comprehensive databases for an entire town may prove a more useful resource. These models may be provided by the architects who designed the building (as required by the town), they may be commissioned by the town, or they may be built for the planner either in-house or as a service rendered by the company. However such an eclectic database may consist of models that are ill-suited for use as context and are unlikely to look coherent. But there may certainly be solutions to the problem (such as providing rough texture-mapped models as well as detailed models) and it is encouraging that there is a perceived market for products that can provide 3D context for use by planners. DASPE is poised to capitalize on such tools.

Texturing and Rendering

DASPE makes use of very basic texturing methods to achieve a sense of scale and offer a suggestion of what may be possible when creating automatic representation of generic building types. The results do not satisfactorily represent buildings realistically, however recent 3D graphics research suggests a number of ways to improve both texturing and rendering. One solution discussed here looks at a method for intelligently rendering a 3D form based on its vector information rather than simply projecting textures at it. An alternative solution may be not so much to automate the construction of a building from a sketch as to present the sketch in a style more appropriate to the level of thought put into it.

Julie Dorsey's work on Cellular Texturing for Architectural Components (Dorsey 2001) may also provide insights into how best to texture map an entire building with a far better degree of realism. While it would be unnecessary (and undesirable) to achieve the level of detail that Dorsey's work allows, the same ordering of texturing operations that starts with corners, then makes edges and finally fills in surfaces would ensure that texture maps do not flow around corners. In addition, edges of buildings could be differentiated with a different material to achieve a more interesting and realistic appearance (figure 10 shows how recognizing corners and edges allows Dorsey's software to intelligently render the stone-work pattern based on a certain logic for stone-work). As well as respecting corners and edges, it may also be desirable to have a center of interest node that could be moved around the perimeter of the building. Early experiments with this technique suggest that it could be implemented fairly easily in Maya by creating a "curve on surface" through "projecting" a curve onto the building facade. When moved, this curve will run around the sides of the building only. The center of interest would specify the location of the building's main entrance and may also invoke a different facade representation at that point. It is beyond the scope of this thesis, but the potential of a cellular based texturing system for creating different building typographies holds a great deal of promise for the success of this first solution.

Figure 10: using corners and edges to intelligently texture 3D vector objects



source: http://graphics.lcs.mit.edu/pubs/sg01_fbct.pdf: selection from Figure 7 (Dorsey 2001)

In the second solution, the goal would be to produce a live 3D rendering that looks and feels like a loose perspective sketch. This solution would rely on the ability of the human eye to interpret loose lines and their intent, and would avoid the fact that it may be bothered by any imperfections exposed in an attempt to render photorealistically.

A number of people have realized that photorealistic rendering of 3D models is not always the most appropriate manner in which to present them. Non-photorealistic Rendering (NPR) is becoming an increasingly popular alternative. Not only does it simplify the rendering process and increase its efficiency, but it also allows the model to presented in a fashion that looks "sketchy." While it would be difficult to discredit the power of photoreal images in communicating a proposal to clients and the public, such images require a great deal of time and attention to produce and such quality is difficult to generate automatically. Automatic renderings that attempt to be photoreal tend to appear "tacky," but perhaps more detrimentally, they appear to be further resolved than they really are. It is difficult to look at a building that is rendered with shading and a texture map and think of it as "just a sketch."

In realizing the importance of representing "sketch" models as sketches (or even rendering more complete models in a fashion that appears more natural and engaging) Richens and Schofield (1995) came up with an architectural rendering system that would employ NPR rendering techniques. This would be done in an interface that allowed the user to provide their own input through painting. The resulting program, called Piranesi, "is a hybrid between a conventional paint program, and a renderer of three-dimensional images. The interface is similar to a paint program, but the pixel planes on which it works are extended to include a z-buffer, which defines the distance to each pixel in the scene, and a material buffer, which defines the layer or material from which that pixel was generated." ([Online] Martin Centre CADLAB, 2002) This product has been very successful and is marketed by manufacturers of 3D software such as Microstation or SketchUp as a solution for presenting the sterile 3D photoreal output of such applications in a more interesting and engaging fashion.

Obviously Piranesi cannot be used to solve DASPE's rendering dilemma as it is designed for painting a single static view, however some of the NPR methods used may be applied in real-time too. Indeed some NPR methods can be extremely simple and require little more than a degeneration of the image from "hard" computer drawn lines to something that appears more spontaneous. Figure 11 shows how simply "wobbling" the lines of a vector drawing can produce a sketch-like effect.

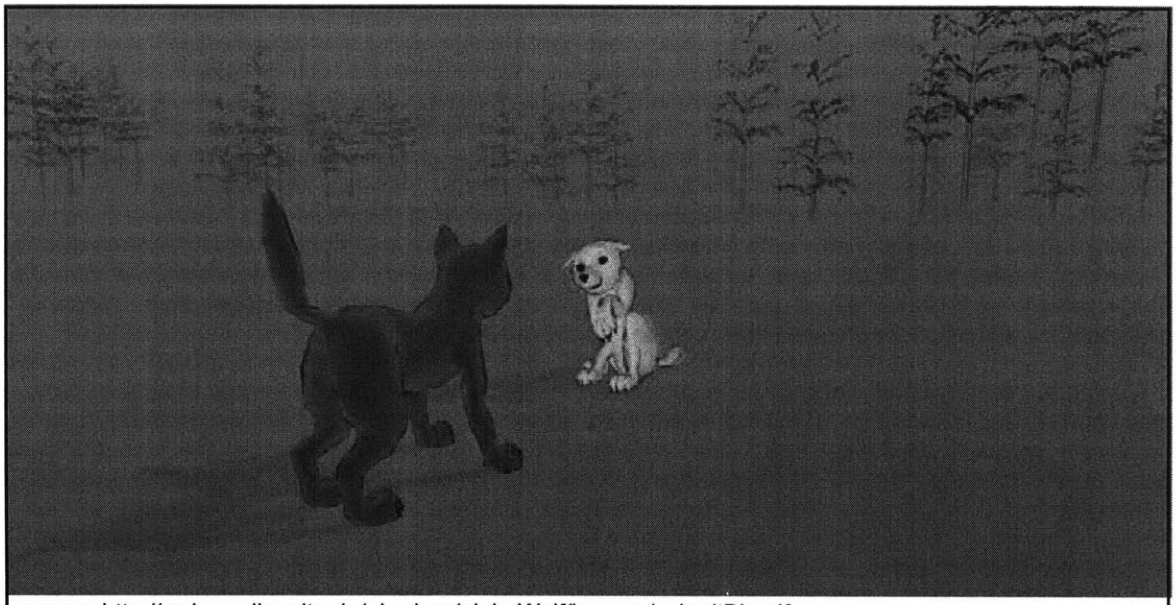
Figure 11: "a hidden-line plot file given the wobbly line treatment using simulated charcoal effect."

source: http://www.arct.cam.ac.uk/research/cadlab/irender/gallery/wigglyline.jpeg

Such degenerations are computationally very easy and this effect could certainly be achieved in a real-time rendering, but requires randomization and may produce undesirable motion in the lines. There are, however, other NPR methods that do work well when animated.

A particularly effective demonstration of this has been employed in the AlphaWolf project at the MIT Media Lab's Synthetic Characters Group (Tomlinson and Downie 2002). Marc Downie has rendered the wolves with loose sketchy lines that easily convey wolves and their body language without trying to represent them in a photorealistic manner. The static screenshots of these sketchy wolves look quite "sketchy" (see figure 12), but one can tell they're not hand drawn and they don't match up to a real hand-drawn sketch by an artist. However when the wolves move about or the camera flies around them in 3D, the effect is quite remarkable. It is hoped that rendering buildings and landscape in a similar fashion for DASPE could be just as effective - particularly for the leashed camera (see page 53) and other moving views.

Figure 12: The white wolf shows submissive body language. This sketchy rendering is used to display 3D geometry and updates as the characters or cameras move.



source: http://web.media.mit.edu/~badger/alphaWolf/images/submitBig.gif

Digitally augmented workspaces

Many people find the "digital workspace" provided by desktop computers to be restrictive due to the size-constraints of the screen and the separation of input and display. The metaphor of the desktop is provided for its familiarity, but the experience of the two is not closely aligned. A digitally augmented workspace goes beyond the metaphorical and takes the physical environment as its starting point. This is then added to by digital inputs and displays. The digital desk that Wellner dreamed up at the Media Lab as long ago as 1992 inspired a number of initiatives to digitally augment our work space. The utopian "vaporware" video (Wellner 1992) produced by Wellner appears comical to today's audience in its quaint technoromanticism. However, like any good vaporware demo, it has broadened the way we think about interacting with computers in our day to day life. DASPE shares a number of technical initiatives with the DigitalDesk. Most notably, it uses projection to illuminate a physical surface, which, but for the illumination and a sensing device, can be used as a normal workspace.

Tangible Interfaces

The MIT Media Lab's Tangible Media Group headed by Hiroshi Ishii has made ground-breaking use of physical, tangible interfaces for urban design. They have made an effort to bridge the gap between the experiences of working with tangible models and working with computer tools. Two projects are of particular relevance to this thesis:

URP -The Urban Planning Workbench:

The "URP" interface (Underkoffler et al, 1999) tracks the position of physical models on a table surface and can be used to project virtual shadows for any time of day and year, conduct basic wind studies, and visualize traffic flows.

The interface has been heralded as a great success and continues to be improved. However a fundamental problem exists in that the physical models are simply "handles" for digital models. If you make a

change to the physical model, you have to make changes separately to the digital model, and vice versa. Efforts to address this issue with "stackables" have been disbanded, and in any case, it is debatable how much design freedom they would allow. As things stand with this interface, one is required to build a computer model to match a physical one or vice versa. The result is that only *placement* of buildings can be looked at using the "tangible" advantages of this interface. Placement is often not the most important solution to either wind or shadow problems.

Shadow-projection is perhaps the most useful analysis function provided by this interface. The wind and traffic simulations are more simplistic and better for visualization and communication rather than actual analysis. Note also that unlike Maya's depth-mapped shadows, all shadows in URP are currently projected as though they fall on the ground - they do not respect the 3D geometry where they fall.

However despite such limitations, URP has proved to be an exceptionally popular means of providing accessible visualization of complex planning information and to facilitate discussion around the table. According to Dennis Frenchman (personal communication, April 26, 2002) , the limitations imposed on the design freedom of buildings may in fact be considered advantages in situations where lay people use URP to express their ideas. The action of grasping and moving buildings is more basic even than sketching and the results of one's simple actions may be immediately appreciated despite the complexity of the calculation involved. As a result participants may become involved in the discussion at a far more advanced level than is usually possible and the quality of dialogue is much improved.

DASPE does not achieve the same level of "tangibility" in its interface, however it deals with objects on a table in a comparable fashion - at least in the sense that it projects a digital representation of 3D objects. In a similar way to moving URP's physical objects, DASPE's objects (be they trees, buildings, lamps or other representations) may be dragged along the table surface with the stylus. It's certainly not the same experience as moving a tangible mass, but has the same advantages in terms of immediately seeing the results of complex computation in an accessible fashion.

Illuminating Clay - A 3D Tangible Interface for Landscape Analysis:

The Illuminating Clay project (Ratti et al, 2002) overcomes the digital / physical disparity inherent in the URP interface by continually scanning the surface of the physical object in three dimensions. This allows the user to modify the physical object, and see how those changes affect an algorithm such as slope, drainage, or contours. The interface can also be used to compare the existing physical model to one saved in memory. This allows the user to track changes made to a landscape, and even calculate the expected costs of such a cut and fill operation.

The scanning method used by this interface lends itself well to landscape analysis because landscapes seldom contain any significant occlusions. If a model of a building were scanned, its roof would be the only part digitized and all details occluded under the roof would go unseen by the scanner and hence unexpressed in the 3D model. For the purposes of planning, this may not be a serious problem (unless the goal is realistic visualization). A larger problem is that of adding vector information to objects in this interface.

The scanner produces purely raster information, so a building and a higher piece of ground are presented and understood in the same way. Without having vector information for the elements in the interface one is unable to associate any information with buildings. It may be conceivable, however to add a vector tracking component in the future so that buildings (and their attributes) could be tracked. Combining this with the ability to scan their size and basic shape in real-time could be extremely powerful.

There is little doubt that such a tool is extremely promising and breaks new ground in adding computer processing to model-making. However the technology for doing this remains prohibitively expensive despite currently being coarse (200 by 200 pixels) and slow (one scan per second and about a 0.3 second delay before the results are displayed). It is thus unlikely to be used by urban planners in the professional arena until the troublesome elements of cost, speed and resolution can be resolved.

Something intangible about tangible interfaces

While the goal in developing these interfaces has been to explore the potential of *tangible* interfaces, the recent disbandment of URP's tracking technology has disabled the tangible component of this interface. However, despite this, people are still drawn to it which shows that it is not "tangibility" alone that is attractive. Furthermore, such table interfaces featuring projection from above appear to be greatly successful at engaging people and generating discussion during meetings - even when they are not actually being used. According to Frenchman the fact that these table interfaces appear to be alive and dynamic gets people to engage with the subject matter with far more intrigue than is usual.

However, other areas of planning require a more liberated type of input than is possible by moving blocks or manipulating clay with one's hands. Indeed, these interfaces do not pretend that they could replace the process of sketching plans for design thinking. Sometimes in using them I have felt the compulsion to just be able to sketch something and get it out onto paper. So, while such interface as URP and the Illuminating Clay will undoubtedly occupy a very important niche, it seems the ideal combination for many designers would be to augment sketching with 3D modelling capabilities.

Sketching in 3D

While most CAD packages make the most of the computer's ability to draw accurately, others have realized that this can be restrictive and is not conducive to designing. One 3D package makes this its focus: SketchUp (from @last Software) is aimed at architects and enables them to sketch shapes using a mouse or stylus and then perform basic 3D operations on them. For example drawing a chimney on top of a roofplane would simply require drawing a circle on the roof, selecting the enclosed "fill" and pulling it up to the desired height using the "push-pull" tool. This is all done in a perspective view. This can often be misleading as there are no depth cues other than perspective cues. Therefore attempting to move something in the y direction by a few inches may actually be interpreted as a movement in the z direction by a few hundred feet. One only discovers what has been done when one rotates the view. This is not a criticism of the software, but a general problem with 3D design from a single viewpoint. This package does a reasonable job of overcoming it and the user can learn how to avoid it. Indeed a number of new skills need to be learned in order to understand how to construct something three-dimensional using sketched lines, but the new skills are easily conceptualized and mastered. Unfortunately this package is let down by its poor compatibility and very basic rendering quality.

Sketch-based interfaces

> *"Perhaps the best testament to the effectiveness of the pencil and paper interface is that very few people even consider it a user interface."* (Zeleznik et al 1996)

In addition, using a pencil in sketching a simple sketch can often convey more three-dimensional information than making a 3D model using a high-powered CAD application. How do we meet the challenge of "[blending] the essence of pencil sketching interfaces with the power of computer model representations." (Zeleznik et al 1996)

D-board:

Another vendor who has recognized the desire of architects and other designers to retain their traditional design interface is Nemetschek. "We're giving you back your pencil" is the claim made to market their D-board product (Nemetschek, 2002). The D-board makes use of a "pressure-sensitive flat monitor" that is similar to Wacom's Cintiq range (and employs the same technology). The user may draw directly onto the screen with a pressure-sensitive stylus that senses 512 levels of pressure. This is a satisfactory way of inputting sketched information into a software package as it is relatively natural, although the slight parallax error that exists and the slippery nature of the glass screen detract from this experience more than the makers would admit. It is nevertheless the best interface currently available for computer-sketching.

Nemetschek sell the D-board as an integrated software and hardware solution. The software aims to provide the user with tools that make the most of the interface - such as natural-media illustration tools. They also claim to have integrated the interface into their line of CAD applications (the "Allplan" range), however I have no way of evaluating the success of this integration. None of the Allplan online communities I approached responded to a request for details of D-board use, so I can only assume that it has not made a great impact. Most likely this is because of its high price-tag.

Smartboard:

The Smartboard was developed by Lew Hopkins of the Department of Urban and Regional Planning, University of Illinois at Urbana-Champaign (Hopkins et al, 2002). It too collocates input and display, but does so on a larger scale by means of a projector and a sensing screen from SMART technologies. The interface is designed to be used from any side and therefore avoids the use of conventional WIMP elements that demand orientation to be fixed. Instead, the system makes use of gestural interpretation. The software will distinguish between the kinds of lines that are drawn and will infer certain properties based on what it finds. The interface is designed for use in planning with the objective of getting people to discuss and communicate ideas around a horizontally oriented display surface. Unfortunately the technology used is not ideal for horizontal orientation since it picks up any pressure applied to its surface as an input. This may be particularly disruptive when using a gestural interface like the one proposed.

Sketch and ErgoDesk:

Robert Zeleznik of Brown University has taken sketch interfaces to a new level with his Sketch and Ergodesk applications (Zeleznik et al, 1996, 1998). Like the Smartboard, and the popular PalmPilot, Sketch makes use of gestures made with a stylus. In Zeleznik's words, "by utilizing gestural interaction, Sketch's interface literally "disappears," enabling users to focus upon their modeling task through an unobscured display of the 3D model." Zelezniks interface requires more sophistication than the Smartboard in that it is used to model in 3D. In this sense it is similar in its objectives to the SketchUp software. To facilitate the extra complication of an added dimension, Zeleznik has had to incorporate many more gestures into his interface. The costs of this addition are that "Sketch users must at least memorize all operations, and to be efficient, must develop sufficient skill so that most gestures are subconsciously encoded in muscle memory." Furthermore, the more gestures that are available to the user, the more complex they must necessarily be. However some innovations such as context-sensitive gestural recognition and the mapping of more complex gestures to more seldom used commands helps simplify the problem of uniquely identifying a large number of gestures.

Unlike Hopkins' system, Sketch does not rely on gestures alone, but also makes use of more conventional, visual GUI elements where appropriate. This is made possible by limiting operation of the interface to one end of the board only. Another interesting innovation in this interface is the use of the non-dominant hand for manipulating objects.

ErgoDesk is a later interface that takes many of the ideas behind Sketch further and employs far more sophisticated virtual-reality innovations. It is designed for "3D modeling with 2D gesture lines (as in Sketch), non-dominant hand camera control, stereoscopic model examination in 3D (using an "object-in-hand" metaphor), stereoscopic model annotation in 3D, toolglasses and magic lens interaction"

While the primary input of this interface remains sketching in the familiar two dimensions, a number of initiatives have been made to create sketch interfaces using 3D input devices as their primary sketch tools (such as 3D mice or the haptic-feedback stylus known as the PHANToM). Such devices offer an entirely different experience to the user and require the development of a number of new skills. They are unlikely to be employed in a largely two dimensional field such as planning owing to the ease-of-use costs implicit in adding another input dimension.

A Question of Progress

"In the name of 'progress', our official culture is striving to force the new media to do the work of the old." (McLuhan 1967)

Many of the examples cited here are of "cutting edge" interface research, but re-considering existing technologies and bringing them together in new ways can often be revealing and may have as its direct goal the resolution of a particular aspect or task. This is possible simply because this is the departure-point for the problem solving process instead of its being in a new research discovery that needs to find a use. Many of the problems faced by planners may have digital solutions, however we should not consider digital solutions superior purely because they are novel and 'progressive.' HCI research has certainly been driven by the desire to improve the lives of people, but there is a disjuncture between what we want in the interest of progress and what really improves our lives.

Furthermore, whenever one is involved in the development of new technology, it is difficult to not get excited by the prospect of adding one component or another and to get caught up in the techno-romance inherent in technical innovation. In the context of DASPE, having the technology take a back seat and letting design sketching and design thinking be the main focus is difficult because it feels like a "cop-out" - as though you're not being loyal to the current technological paradigm. However shifting the focus off technological innovation can allow focus on the real problem to be maintained - if you're not caught at the cutting edge, you don't have to change focus every time a new innovation occurs.

In designing an interface such as DASPE, it has been difficult to remain focussed on the goal of improving the design process of urban planners. It is too easy to become enchanted with a new innovation or clever design solution. I have tried to counteract this tendency by coming up with four design principles that would be used to guide all design decisions. They are presented here in order of priority.

Design Principles

1: Match the level of input to the level of design thinking

"the hand moves, the mind becomes engaged, and vice versa" (Rowe 1985)

Urban planning occupies an interesting niche in the computer-aided design field. Unlike architecture and engineering whose tools it has shared, it is a more general practice in that many aspects of the design remain unspecified. The use of conventional CAD interfaces for planning often involves a great disparity between the amount of thought given to the plan and the amount of work necessary to input it into the computer.

For example, while an architect must give consideration to the facade of a building and its details and must indicate these decisions to the CAD software, the planner will most often give no thought to this. However, when dealing with computer drawing software, it is often necessary for the planner to input such information in order to communicate their ideas with the appropriate degree of realism.

This thesis will explore how the computer can be used to achieve a degree of false-realism. Since so much in a proposal or master plan is out of the planner's aesthetic control, the computer can be used to "fill in the gaps" as architects and developers will ultimately do.

At the other end of the scale, it is possible for the computer to go too far. It may be undesirable for the computer perform too much "intelligent" computation and automate processes that do require design thought and input. For example, one could write an algorithm to calculate the layout of a parking-lot from a roughly sketched form. This would be a useful tool at the point where the planner thinks of that form as just an area of parking. However, it is quite likely that he or she will want to consider that parking lot in more detail later. As soon as the designer is ready to move to the next level of thinking, the interface should allow it to happen.

Sketching as a thinking tool:

"Whence did the wond'rous mystic art arise,
of painting SPEECH and speaking to the eyes.
That we by tracing magic lines are taught,
How to embody and to colour THOUGHT?"
(McLuhan 1967)

I believe McLuhan was reflecting on text as a medium in this short rhyme, but it reads with equal relevance if one considers sketching to be the topic.

Technically a sketch plan should allow for little more to be expressed than basic forms and relationships between elements. Such sketches are highly abstract, simplified and expressly two-dimensional. However with experience one may learn to use this technically limited medium to explore all the complexities implicit in the design decisions being made.

As with written language, there are far greater depths to the lines drawn than may be immediately apparent. And, like any other language, sketch-planning cannot be fully appreciated without experience of its syntactical conventions.

Such conventions may be implied by the basic principle of a scaled down view from above, or they may be derived synthetically. For example, a double-dotted line with long segments between has come to de-

pict a site boundary although this graphic depiction has no relationship to reality. Still other conventions, such as contour lines, may combine arbitrary rules (such as a horizontal cut-line every two feet) with a depiction of reality.

Backed by this broader language of plan-drawing, a sketch-plan may stand alone as a document that describes reality in an abstracted form. However, for a designer, the process of drawing a sketch-plan is an exploration of far more than can ever be described by the plan alone. Indeed, a sketch-plan is essentially little more than a forum used to explore one's design thinking and help it materialize.

Experience with sketch-planning may allow both the sketcher and external observers to understand what is implied by lines drawn on the plan, and since each line is drawn while thinking about a certain design decision, observers may begin to appreciate what the designer was concerned with.

Of course, not all design-sketching is used to materialize a plan in the mind or generate new ideas. Often it is used merely as a tool to communicate it to others. Indeed it is common practice to separate those plans that are used to think and those that are used to communicate.

Computer-aided drawing tools have been used mostly as a tool for the communication process rather than the thinking process. In fact, it may be more accurate to say that they have been used for the purposes of drawing production with the distant goal of communicating design thinking. Computer drawing tools have been designed so as to make the drawing production process more efficient. The down-side to this (even when one improves the interface to be more natural) is that one needs to constantly think about how best to construct a drawing. Less efficient methods of drawing (such as pen on paper) create a mundane task that may allow more freedom to think through a design problem.

After much debate about the topic of whether or not the computer can actually be used for design thinking, computer-design has carved out a very significant niche for itself in the design process. While the computer may not be as useful for thinking about two-dimensional relationships on a plan, the visualization capabilities of 3D software necessitate its use in thinking about complex three-dimensional design problems. However, since such design problems are rarely encountered in the planning field, there is very little use of such tools by planners for design-thinking.

Planners therefore rely on traditional media to sketch out their ideas, and the computer does little to assist their thinking. Freehand sketching provides a great amount of freedom, but there comes a point when one begins to feel restricted in one of two ways: either by lack of accuracy (mostly limited by a fixed scale) or by difficulty in visualization.

It is true that planners can become experienced in using sketch plans to visualize their designs as three-dimensional entities. Perhaps then, there is no need to make something three dimensional (be it a virtual or physical model) in order to think about the experience of a place at the "human scale". But in reality, how much guessing is done about the meaning of the sketch? To what extent is the planner actually composing the plan as an aesthetically pleasing graphic in itself. There is certainly some correlation between plan aesthetics and the quality of the experience provided on the ground, but to what extent is it real and how much are we fooling ourselves? Most importantly, we should consider whether designers would make different design decisions if their tools allowed them to view the implications of their sketches as they would be felt on the ground.

2: Facilitate effective communication and maintain its intent

Sketching as a communications tool:

> *"I came to believe that the comfort and ease with which humans could express their graphical ideas were more important than the machines ability to render them as synthetic photographs."* (Negroponte 1995, p104)

Drawing for communication:

A good planning interface should allow ideas to be communicated as efficiently as possible without compromising the subtleties of what is intended. Communication is the essence of the planning task. Planners don't produce anything tangible. So, in effect their *product* is communication. This product may be tangibly expressed in various media - drawings, models and (more recently) video being the most popular.

One could make that statement about a number of professions - law, architecture, even engineering. However, in these professions the documents produced communicate very specific instructions to those who use them. Planning documents seldom reach the same level of specificity - and even if they do, it is understood that the planner is a "generalist" and that even specific instructions may be interpreted merely as guidelines.

Often the planner will have little control over the end-result and what gets communicated is not a set of instructions for the creation of a physical environment, but a vision of what it should be - most importantly it is about ensuring that a large number of people share this vision.

"Shared understanding" is critical to any task in which a number of entities play a part (Schrage 2000). This becomes increasingly important the further down the line the communication gets. If the builders understand the role a certain tree plays in a scheme, they may be happy to put in the effort on-site to build around it. Similarly, "getting the public on-board" (in other words getting people to share in the vision) is essential to the success of most plans.

> *"Within a generation, the singly largest use of 'new information technology' in planning will no longer be in an exclusively professional context such as in preparing development plans or in development control. It will be in educating the wider community in planning matters and in engaging the community through planning information and participation in the design process."*
> (Smith, 2001, cited Bullmer 2001 p.16)

Unfortunately the current practice of producing extensive hard-copy master-plans has not been conducive to promoting this shared understanding to all interested parties. There is however a growing trend to create marketing documents (and recently videos) that get people to "buy into" a master plan. While working at Sasaki Associates, I looked at how master plans could be made accessible to the public in a manner that made the actual content more engaging instead of simply marketing it. Using interactive media, a master-plan was presented online in an accessible fashion. An earlier project used the internet as a public-feedback forum - also in the interests of promoting shared-understanding.

Sketching for shared-understanding

> *"it takes shared space to create shared understanding."* (Schrage 2000)

Sketch-plans are seldom presented to the public or other non-planners. This may be because they look untidy and lack the aesthetic appeal of cleaner drawings, but it is also because the observers are not familiar with the language used and cannot visualize what is implied. Other forms of sketching - such as perspectives (that effectively communicate the "vision" from an experiential perspective) - are common however.

"In-house" communications exploit the common language of sketch-planning to generate shared-understanding between planners and others who know the language. The advantages of sketching are that design thinking and communication may often occur simultaneously. Indeed an interesting dynamic is created when planners sharing the same space will break communications to think through a plan in a

sketch and then resume discussion in order to present their respective plans. If all goes well, a single plan (often a compromise) will be decided on and accepted as the current "prototype." Once the sketches are all considered, one sketch can be taken further in 3D to make use of the visualization and analysis tools. This will allow the shared plan to be considered from a different perspective (quite literally) and will most likely generate new discussions. Indeed the verbal dialogue that surrounds the process of sketching is essential to the communication of ideas and the generation of shared understanding. The sketch may be more useful as a forum for discussion than as a product in itself. (Schrage 2000)

Communicating vision

It is often not specific ideas for a scheme that the designer needs to communicate, but the vision that determines those ideas. The importance of communicating a vision for a scheme does not go unappreciated and firms will spend much of their presentation effort in communicating their vision. It is generally accepted that the vision should tend towards the utopian and matters little that the project will never live up to that vision. Utopian romanticism is, after all, one of the forces at work in making improvements to our living environments. In a sense, planners are simply partaking in this tradition. It should also be noted that once "sold" on a vision, the public will be more partial to accepting the "logic" presented to them.

This is important to this thesis since much of the effort made is to *visualize* ideas. This is quite different from presenting a vision. When one presents a vision, one has the vision (often literally a visual image) in one's head and sketches (or paints in watercolor as the tradition would have it) in a manner befitting the utopian tradition. While this is a successful communications tool, it is often divorced from the real ideas that are explored in the design. This interface is more impartial in that it does not communicate a vision so much as it enables the user to compare schemes in a visually accessible fashion without the pretence and bias associated with most 3D visualizations.

3: Add a dimension without adding complexity

> *"The ability to visualize potential modifications to the urban fabric and experience these changes in their actual context allows planners and designers to evaluate alternatives rapidly, in more detail, and for lower cost than through more traditional analysis. It also makes the results of planning process visible, allowing the public to view the proposed changes to their environment in a realistic fashion."*
> (Jepson 2001, cited Bullmer 2001 p.16)

The addition of a third dimension to this drawing interface gives it great potential as a visualization tool. What's more, it can encourage designers to consider their design implications at different scales. While a number of "heuristics" will always be available to the designer (Rowe 1987, p85), anthropometric design (that is design for the human scale) can be given a fighting chance by this interface. 3D software allows the designer to make the bridge between one scale of design and another. Therefore one can draw a city block at a broad scale in plan and immediately see the impact of its massing from the level of the street. Indeed Rowe suggests that any single design heuristic is unlikely to provide all the necessary information for the design process and frequent switching between heuristics is common practice (1987, p107).

The temptation to further allow drawing and not only visualization to be conducted in the 3D view is great. Indeed drawing 3D objects in perspective view in a computer interface can be extremely impressive. Maya allows objects to be drawn in the 3D window by choosing either a "live" object upon which to draw or placing a drawing plane object in the scene. Moving and scaling objects may be done with 3D handles that ensure that one always knows in what direction one is moving an object.

For those who have not tried drawing in the perspective view of a 3D graphics program, it might be worth describing the chief problem that the interface must overcome. This is that the sketcher looks at a

two dimensional representation of a 3D scene and has no real depth cues other than those indicated by perpective lines and the diminishing of objects of known scale. Often an attempt to move or draw something in one direction in 3D space will instead move it in another direction. Maya provides separate "handles" for each dimension in order to ensure that it is always obvious what the user is intending. Unfortunately this requires a number of new concepts and skills to be developed. SketchUp takes an approach that is easier to learn but more susceptible to error: it tries to guess the plane for the stroke according to where some axes have been set. It will also provide feedback to the user about the direction of the stroke according to the plane. This too is easier for the user only after coming to terms with some new concepts.

For DASPE's tools the 3D view can be used to draw on the horizontal plane only. This is essentially the same as drawing in plan, and avoids the confusion mentioned above. But this approach is discouraged for a few reasons. One is that drawing in perspective (with or without stereo vision) may be confusing in terms of the relationships between objects in all directions. Objects going into the perspective will look closer together than objects with the same spacing across the perspective. So, for example, it is very difficult to draw a circle on the floor when looking in perspective. Whats more, this effect changes depending on the focal length of the camera used. I also felt that if users were encouraged to draw in 3D they might expect to be able to sketch in perspective in the same direct way as sketching on a piece of paper (after all they are encouraged to do so in the plan view). Research into "projective drawing" (Tolba et al, 1999) suggests that users could draw into the perspective in such a fashion, but it would be difficult to implement in software such as Maya and also requires some new skills to be learned.

By focussing on the plan alone, one can greatly simplify the task and reduce the need to learn confusing concepts about 3D computer graphics. This is important if one wants to let people feel comfortable using the software in a public arena. In addition by drawing in plan but seeing the effects in 3D, one becomes very aware of the relationship between the two and the implications of decisions made in plan on the 3D world.

4: The computer as butler - pervasive computing

In a world of increasing complexity, it is necessary to filter the inputs one encounters on a day to day basis. (Milgram, 1967) Digital enthusiasts such as Negroponte insist that "digital butlers" would be able to perform this task with the necessary intelligence (Negroponte, 1995). I find this vision steeped in the sentiments of the technoromanticist tradition (Coyne,1999), but the concept is intriguing: the computer becomes an entity that takes a background, but ever-present informative role. Interruptions are timeous, courteous and select.

This thesis embodies the spirit of the "pervasive computing" revolution. The idea that rather than work with computers, we do the work we'd like and the computers are simply present, helping as need be. However, the full expectations of this revolution are not fulfilled here. In pervasive computing, the computer would be aware of all the pencil-sketching going on in the room and would be able to assist that process intelligently. One would not need to work with an interface as such at all. However the separation of interface and the pencil-sketching as input (even if display is concurrent) acts as the first filter to tell the butler whether or not it's assistance is wanted.

This design principle is the most ambitious given the constraints imposed by the current software and the operating system - particularly with regards to interruptions. However, I have included it as a principle that will be beneficial to strive towards even if the ideal is not met. In DASPE the sense that the computer can be present when needed, but is otherwise standing by, waiting to be called could be described as "butleresque." Having the interface serve the user in this way can do much to improve the "relationship" between the designer and the computer.

What it means to "Digitally Augment"

Augmenting instead of emulating

This computer interface takes an unusual stance in presenting itself as something you can have at hand and choose to use or not use. Most often when a new tool is invented for a physical process, it complements many of the existing tools instead of replacing all of them. With the computer it has been difficult to incorporate existing drawing tools into the new medium for drawing. Instead these are emulated.

To use the computer to complement sketch-planning rather than replace and emulate it is sadly a novel approach although it's precisely what would have happened when French-curves were invented - or any other physical drawing tool. In this interface, if one chooses to make use of the digital augmentation, one does so because of a desire for it to be there and not because one has to choose between using it completely or not at all. One is not forced to define oneself either as a "user" or to resign oneself to being considered a technophobic "non-user."

Using tools digitally

What does it mean to use tools digitally? While augmentation of sketching tools with computer tools in this interface may be preferable to replacing and emulating physical tools, the computer tools used will be subject to the usual constraints and conditions of computer tools. Perhaps the main problem with computer tools is that they are abstract entities with no physical suggestion of their use. The popular solution has been to tie them into a physical analogy. This is used to determine both the name of the tool and the icon used to depict its function.

Such analogies are very useful in the initial stages of getting to know the software. Experimenting with such tools (or following hints and instructions) may then give a clearer conception of how they really function. As with learning physical tools it is sensible to learn to use it as it was intended to be used. Once mastered it may be applied in unintended creative ways. With physical tools, however, creative uses may be imagined simply by looking at the tool. One could pick up a hammer (never having seen one before) and use the back end as a chisel. In the digital realm, there is little about a tool to suggest its real nature or the creative ways it may be used in conjunction with other tools. The icon and associated analogy represents what the tool is intended to be used for, but not all it can be used for.

There is often a right way to use a tool that must be followed in order to work best within the software, but this does not necessarily restrict freedom of design any more than the requirement that one use the front end of a pencil for drawing and the back end for erasing. However allowing creative use of tools in conjunction with each other presents a serious challenge for software developers since debugging and testing requires that any possible combinations be tested against each other in order to eliminate bugs. It is therefore much easier to build software if you limit the number of ways in which a tool can be used, even if this detracts from the potential of the application. Hence computer interfaces usually present a myriad of tools that are unrelated to each other and which each perform a simple, solitary function. It is intriguing, however that some tools which perform complex computation may be framed as something that is easy to grasp conceptually. In addition, such tools may allow many variations to be explored and provide the user with the ability to design creatively in ways not necessarily paralleled in the physical world. Many commonly used tools in 3D graphics packages such as "boolean operations", "extrusions" and "lofting" fall into this category.

Gestures

The ability to interpret gestures drawn with a pen stylus can enable the creation of a far more natural and "casual" interface than is otherwise possible. A small survey was conducted in order try to establish whether there were any consistent patterns in gestural sketching. Such patterns would make it possible to take some short-cuts in the implementation of gestural sketching. In conducting this small survey (of 12 participants) it was not always easy to elicit sketched information from participants without giving too

much away about what might be expected. In some cases this resulted in confusion and the observations were not always useful. For example exercises 7 and 8 on the survey sheet (figure 13) were not useful in communicating that the gesture was to "screw" in or out, but it was felt that using the word "screw" would make too obvious a connection to the convention of using clockwise and counter-clockwise rotation to drive screws in or out. In hindsight it may have been better to test this assumption instead of trying to see whether sketchers would also follow the convention when drawing spirals.

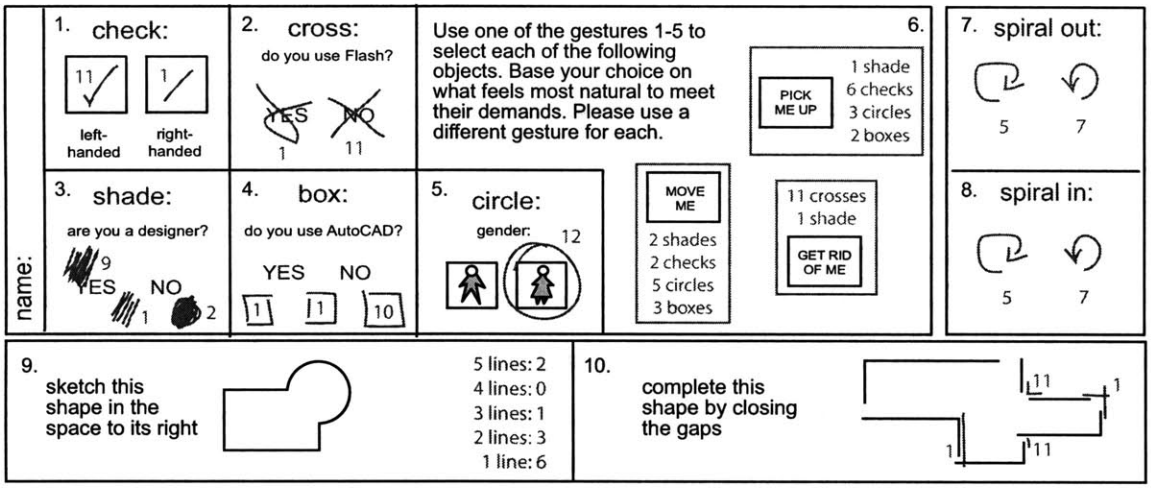


Figure 13: the survey with a summary of the results: note that the answers to the questions themselves were not important (and are not reflected here). What was important was the manner in which each sketched gesture was made. The numbers reflect the number of sketchers who made that gesture out of a total of 12 surveyed.

The most significant discovery from this survey was the manner in which the users drew boxes (5) and the building footprint (9). Initially it was hoped the Building Sketch Tool would be used by drawing each line for the footprint individually, but after the significant preference was discovered for drawing such shapes as single lines without lifting the pen or drawing them with two such lines, it was decided that it would be necessary to write a script that would recognize corners in a line and divide it into straight and curved segments. This script is included in Appendix A.

Other decisions based on this survey were the decision to recognize only those crosses (2) made as two separate strokes and to ignore those made as a single line. In addition shades (3) are recognized as a single stroke who's drag-points are densely packed together. Separate strokes (as used by only one user) for a shade will not be registered. All users drew circles counter-clockwise and most ended their circle a little way down the line from the start point (as opposed to before or at the start point). The circle recognition algorithm was therefore altered so that a circle would still be recognized if the end-point was near the beginning of the line and not just if the sketcher finished it near the start point as was formerly the case.

While exercise 10 showed that most sketchers like to extend lines at corners with a single bent line, it was decided that the current method of extending lines individually and as straight lines would be sufficient. A better algorithm may be developed for completing corners in the future. Exercise 6 was difficult to explain (even though much verbal direction was given in addition to the written instructions). Most users showed no particular preference and were not convinced about their choices. The only exception to this was in using the cross gesture to delete objects.

Scale

One of the most obvious complications of digitally-augmenting sketch plans is that sketches on paper have a fixed scale while drawings on the computer have a variable scale. While it is no problem to fix the scale of a computer drawing to match a hand-drawing, the experience of drawing with the computer at

a fixed scale will be inherently different from the common CAD experience in which scale is constantly adjusted in order to draw accurately.

The ability to zoom in or zoom out easily is seen as an enormous advantage of computer drawing software. Historically, low quality displays of limited size and processors incapable of "anti-aliasing on the fly" have made the ability to zoom an absolute necessity in order to see what one is drawing. However advances in display technology, processor speed and anti-aliasing techniques have brought us to the point where we realistically work with computers at a single scale. This is important to enabling sketching with conventional media at fixed scales.

Allowing this to happen in an interface that uses a computer alongside conventional sketching has an interesting spin-off in that moving between scales becomes a simple matter. This allows one to combine the advantages of working at a fixed scale with the ability to quickly move to a new scale once one is ready to give greater thought and work in greater detail.

Inherent Problems with Digital Media

One of the reasons for choosing to augment sketching rather than replace, emulate and improve upon it using a digital medium is that there are a number of problems associated with being digital for which we may never find solutions in furthering technological progress.

Using digital media encourages certain behaviors that are not conducive to design freedom and thinking:

1. A sense of final product creation (a computer drawing or model) rather than design as the focus.

2. A sense of continuity and impermanence - computer drawings can be endlessly modified and one may feel a compulsion to perfect them.

3. The perceived need to refine one's methods to the most efficient possible and the sense that because you can be using better tools (lines, rectangles, extrudes) you should. Thinking about this during the design process rather than just thinking about the design may be detrimental to its quality.

Digital displays are designed to be dynamic and we expect to see their image change. It may be desirable at times to have a single consistent image on the screen for the duration of a meeting. Computer media may be ill-suited for such a use owing to an associated sense of impermanence. Sometimes the transient nature of digital displays can be undesirable.

Current limitations that may be solved technologically:

1. The decreasing, but ever present disjunction between the actions of the pen tip and the resultant lines.

2. The physical constraints of the interface in terms of size and fixed orientation.

Iterative methods with various media

When combining various media into a single interface, it makes sense to use each for its own strengths. It may be necessary to do this at different stages in an iterative process in which the media are used alternately and build upon each other. In my undergraduate architectural studies, I developed a method in which I would first sketch a plan, then scan this and use a 3D package to explore the relationships of basic three dimensional forms. In order to give further design consideration to those elements, I would print out a number of perspective views and sketch into them. Exploring those same sketch concepts in the 3D software would have been prohibitively complicated and frustrating. Even though the method was cumbersome in comparison to that proposed by this interface where input and output are collocated, it used the strengths of both sketching and 3D modelling software where they were most appropriate in the design process.

Integration with Maya

Flexibility versus Simplicity:

When designing tools for an interface there is a trade-off between their flexibility and their ease of learning. This is best demonstrated by "dedicated" software such as ArchiCAD where more sophisticated modelling capabilities are intentionally sacrificed in the interests of serving the majority of uses with the most simplicity and greatest efficiency. There should really be no limit to what one can design in a 3D software application. However imposing such limits on freedom makes such packages more manageable for the user - and, importantly, far less intimidating.

"Intuitive design" has come to mean design for skill-sets already developed. This means that people should need to do the minimal amount of learning possible in order to master the interface. This is admittedly a sensible approach, but one that has become cliched and is practiced too religiously. Mastery of more difficult tools can be a rewarding learning experience, and enable far more complex and powerful interactions with the computer. Much mastery may be self-taught - but this takes confidence and interfaces that encourage it are usually not "user-friendly." An interface such as Maya's is far too intimidating for most computer users to learn without assistance.

Taming Maya:

Ideally an interface should achieve a balance between flexibility and simplicity. It should offer all the depth and complexity you could hope to master, but also allow a layer of simple "top-level" use. This is especially useful if a number of individuals are involved with the same design process (each may work to his or her strengths at different levels in the software).

Maya (from Alias Wavefront) is a very powerful program in terms of its broad array of 3D modelling tools that can be used to create just about any 3D geometry one can imagine. The interface provided, however, reflects this breadth and is so complex as to be unusable before learning a number of new concepts and gaining a lot of experience with the tools provided. Maya is acknowledged to be one of the most difficult software packages to learn, however it is also designed with the intention of being customized and simplified for a specific "work-flow." A C-based scripting language called MEL (for Maya Embedded Language) is provided so that programmers can create custom tools for designers to use at a higher level without the need to learn the program in its entirety.

Generally, the down-side to this method is that users may become too comfortable in the top-level with which they are familiar, will not explore the software at a deeper level and may work in a needlessly inefficient manner. However, from my perspective, the use of those deeper layers is not so much that of efficiency as providing the tools to complete the design as needed when the top-level fails to provide all the necessary tools. At the top-level are dedicated tools to be used for those specific functions for which they are designed. If there are gaps left by those tools, they may be adequately filled by more sophisticated tools at a deeper level or new top-level tools may be written.

DASPE

Description

DASPE stands for the Digitally Augmented Sketch Planning Environment. The acronym is used to refer to both the software and hardware used to create an environment in which planners may sketch in the conventional fashion, while having this process "augmented" with digital information and digital tools. The software component of DASPE is created by adapting Maya using MEL. The hardware consists of a digitizing table and roof-mounted video projector. By projecting the image of the screen onto the digitizer one may create the illusion of a large touch-screen. The experience is similar to using a very large version of Wacom's LCD-backed graphics tablets (such as the Cintiq) although the quality of the DASPE display is not as high (individual pixels can be seen) and the collocation calibration of the stylus and the display is more prone to "getting out of sync".

It is quite feasible that future renditions of this environment will make use of a large LCD-backed graphics tablet, however using projection has its own advantages. With information projected from above, layers of paper, trace-paper and other paraphernalia used in producing sketch-plans may be placed on the table without obscuring the digital display. This allows the sketcher to draw on the table in the conventional fashion without worrying about the table surface and indeed giving little thought to the presence of the computer until it is needed. Information can then be projected onto the surface in order to inform the sketch process. Layers may be turned off or on in considering different types of data such as land-use zoning plans, topography or vegetation coverage. (Note that producing such layers from GIS data is not facilitated within Maya, although it may be possible to do so by creating a plug-in that allows shape-files to be displayed).

Sketching on paper and later "digitizing" sketched information into a software package is a common practice. There are a number of reasons why it is desirable to move the plan from "hard-copy" to "soft-copy." In my own experience, this has been done to improve upon the drawing in terms of accuracy and final production quality, although increasingly plans are being traced into software packages to make use of their computer modelling and visualization capabilities.

In the same physical space, DASPE both provides information to inform the sketch-planning process and allows the plan to be sketched directly into the computer for further visualization and analysis. Both processes happen on the same table without moving the drawing. Instead one simply changes one's drawing device from a pen that draws with ink to one that draws with digital lines. Therefore, once the planner has sketched out a scenario, the pen stylus may be used to "solidify" it and input it into the computer. It is hoped that this will not be a "dumb" process and that the sketcher will be making decisions important to refining their sketch as this next stage is explored. For example a decision may be made on how far apart the trees should be that were originally sketched as a rough squiggle. Furthermore, some objects such as buildings may be solidified in plan to have straight lines or made to conform to an orthogonal grid (with "ortho snap"). In addition, thought must be given to the vertical component of the buildings being drawn. Other processes such as drawing roads, concrete and hedges make use of automation from a center-line, and still others such as drawing water or grass may be thought of simply as a quick way to render an area by tracing its outline. The point here is that the process of moving the drawing from a rough sketch to a 3D computer model is no longer a disparate one, but rather one comparable to tracing over one's rough pencil sketch using a pen and ruler and coloring the refined drawing with markers. The goal is to provide tools that are really no more complicated to use than picking up a new marker color or learning to use compasses or French curves. Ideally one will be almost inadvertently creating a three dimensional model in hardening the sketch plan, and thought need only be given to the added dimension when it is appropriate to do so. Visualization tools may then be used to capitalize on the third dimension as one feels the curiosity or the need to use them.

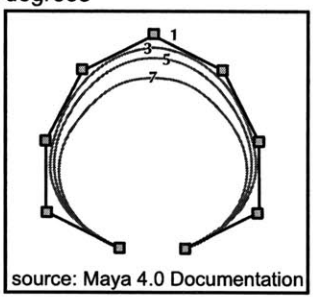
Maya Commands Used

Before describing the tools created for DASPE, it will be useful to describe all the native Maya tools that are used and how they may be accessed from an unmodified Maya interface (referred to here as the Maya GUI for graphic user interface). This should help to clarify for each of the tools what is being done by Maya and what has been added for DASPE. The tools are grouped according to their use in the DASPE interface. For most tools there is both a GUI command available from a menu in Maya and a MEL command that can be either used in a script or entered in the "command line." In describing the tools, I will list the MEL command first, followed by the GUI location in parentheses, for example: planarSrf (Modelling | Surfaces > Planar). Here "Modelling" refers to the toolbar, "Surfaces" to a menu on that toolbar, and "Planar" to the menu item in that menu.

General use - lines and dragging:

Curve: specify a curve spanning a number of points. Curves must be of a specified "degree": degree 1 is "linear" and creates the curve as a series of straight lines between points. The higher the degree, the smoother the curve. Degree 3 is also known as a "b-spline" (see discussion on "Lines" under components). Maya provides two different tools for placing curves by picking a series of points. These nay be either "edit points" (Modelling | Curves > EP Curve Tool), or "control vertices" (Modelling | Curves > CV Curve Tool). Curves may also be drawn freehand with a pencil tool (Modelling | Curves > Pencil Curve Tool). These tools all make use of the same MEL command ("curve") followed by specifications of the desired degree and a list of points in the Cartesian plane.

Figure 14: curves of varying degrees



source: Maya 4.0 Documentation

The Pencil Curve Tool would seem ideal for a sketch interface such as this. Unfortunately its usefulness is limited by its inability to be modified or communicate with MEL scripts. There is no way to execute commands to work while the curve is being drawn - so for example it could not be used to make the tree-placing script that places trees along the curve while the sketcher is drawing a stroke. In order to execute scripts on completion of a line, one has to make use of a "scriptJob" that checks continually for a selection change to indicate the completion of the line. In other words one must employ a number of work-arounds in order to make use of the tool in customized contexts. The methods are messy and may interfere with other scripts if not terminated properly.

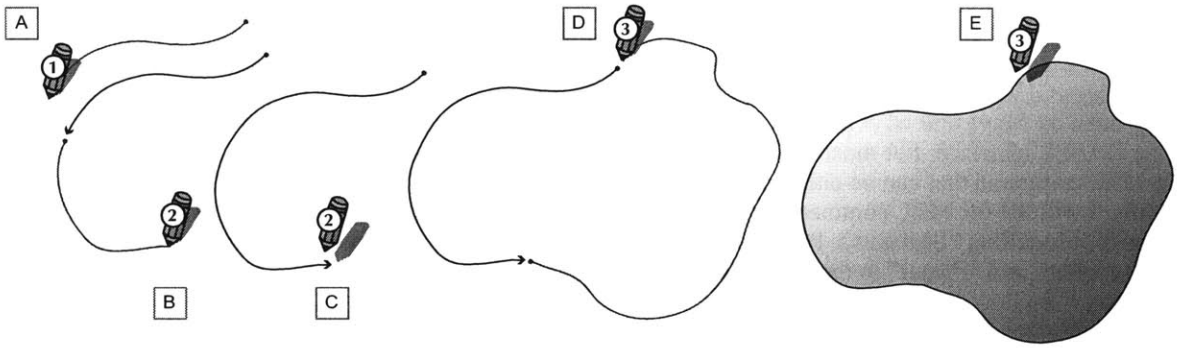
For these reasons, most of the tools written for DASPE look directly at all drag points and make a curve from those points if the tool requires it. The curve is linear (or degree 1) and its quality is not as good because there is currently no interpretation of the points and only a basic averaging is done for smoothing purposes. This could be improved in the future by better emulating the Pencil Curve Tool or similar tools found in other software products.

The command used by the scripts to look at drag points is called "draggerContext." This command allows MEL scripts to be executed at the start of the drag, at each new drag-point, at the end of the drag operation, or any time the cursor halts during a drag. It allows a great deal of control to be achieved for any MEL script. Especially important for a gestural interface such as DASPE is the ability to run scripts while the mouse is being dragged. This is critical for more advanced "interpretation" as used in tools such as the building sketch tool. Note that draggerContext is available for use as a MEL command only. One uses it to create new "tool contexts." It is therefore not directly available through the Maya GUI although it may be present in GUI tools.

Zoning tool, grass and water.

These tools are the only tools that make use of the Pencil Curve Tool. Shapes for land-use zones, grass and water are characteristically drawn large and curvilinear. They may therefore take advantage of the

Figure 15: automation for drawing large curvilinear shapes (land-use, grass and water)



Pencil Curve Tool's smooth lines. Furthermore, it is common practice to draw larger curvilinear shapes by leaving a small gap when one lifts one's hand. Each curve sketched is therefore automatically joined to the previous one (see figure 15:C) using "attachCurve" (Modelling | Edit Curve > Attach Curves.) This happens whenever a scriptJob determines that the line has been completed. In the event that the end-point and start-point of this connected curve are within a given tolerance (D), the shape is closed using the "close-curve" command (E).

The curve is then filled (E) by creating a planar NURBS fill with "planarSrf" (Modelling | Surfaces > Planar). This command can be used on any number of lines (not just closed curves) provided that the lines do touch and form a closed plane.

It was initially hoped that this method could be used for all shapes, but it proved too basic and unreliable for sketching buildings. It does not handle intersecting lines well, corners are always slightly rounded and "loaded" with CVs and there is no way to ensure that the building will be right-angular. The rounding and CV clustering is most noticeable if the shape is extruded. It is therefore best used for rough, mostly curvilinear shapes that aren't extruded and is hence well suited to the task of drawing areas for grass, water and land-use.

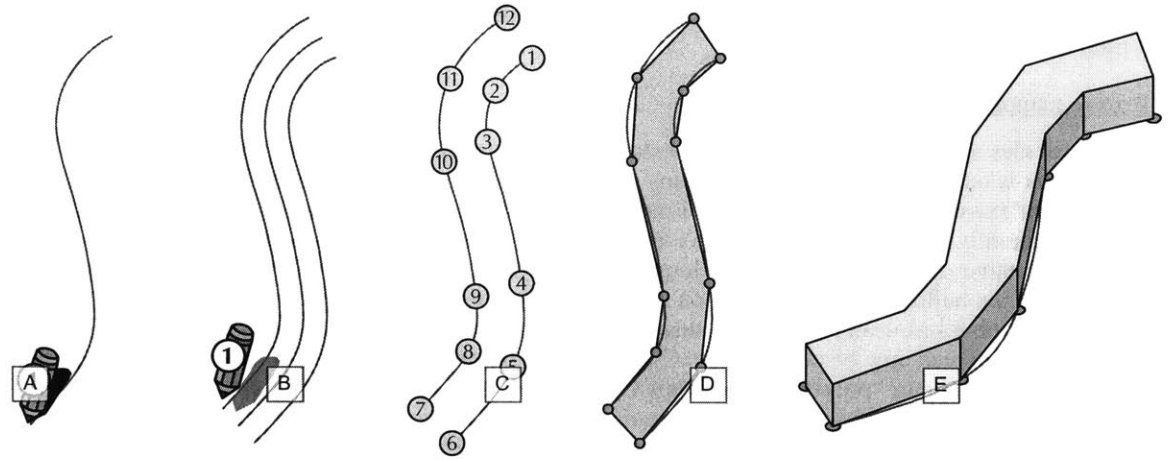
Commands used in making buildings, concrete, roads and hedges:

These tools share the characteristics of using "offsets" and "extrusions" based on the lines drawn to create shapes that have volume and height.

Hedges, concrete and roads make use of the same method for constructing a raised, thickened stroke along a line. A number of methods were considered in order to construct such an extruded shape. The most obvious way to do this when making models manually in Maya would be to extrude a cross section of the road, concrete object or hedge along the line. However, automating this to work in all situations becomes a little more complicated because the cross-section must first be rotated to match the direction of the line and placed at the beginning of the line. It is also necessary to close off the ends of the shape using a planar surface otherwise a hollow tube is left. Furthermore, since roads are to be used in a boolean operation and Maya Complete does not allow boolean operations to be performed on NURBS objects, it became apparent that this method would not be adequate.

Instead, it seemed desirable to write a script that would convert the center-line into a polygon facet with the appropriate width before extrusion. Figure 16 shows this process. The curve (A) is offset (B) to the desired width on either side. The script then loops through all CVs for each line and stores this information as a string of points (C). These points are then fed to the "polyCreateFacet" command (see below) in order to create a single polygon facet (D). This polygon facet is then extruded (E) to form hedges, concrete and the road object to be used in the boolean operation (see DASPE Tools: Road Tool). Hedges and concrete objects are completed through application of a texture to this extruded shape. Hedges make

Figure 16: automation for extrusions along lines (used in hedges, concrete and roads)



use of a bitmap texture and use "polyAutoProjection" (Modelling | Edit Polygons > Texture > Automatic Mapping) to achieve the correct mapping to all the various angles of the extruded form.

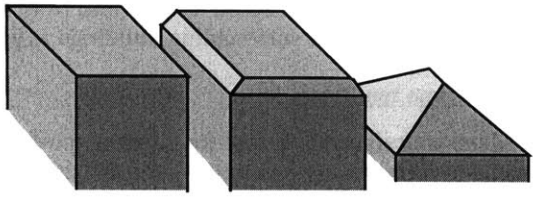
"offsetCurve" (Modelling | Edit Curves > Offset > Offset Curve): the process of "offsetting" is a common CAD function in which a new curve is created such that it lies a set distance from the first curve at all points along the line. This command is used for roads, concrete and hedges to offset the centre line on both sides (figure 16: B). For buildings it is employed to create a slightly larger footprint used to make the lower story and roof-line. This gives a slight overhang for the eaves and allows the lower storey to wrap around the rest of the building with a different texture map. If there was no offset for the lower storey the renderer would attempt to render both texture maps at once resulting in an undesirable shimmering surface.

"polyCreateFacet" (Modelling | Polygons > Create Polygon Tool): this command creates a single polygon facet made up from a series of specified points. The order of points specified for this tool is very important as edge-lines joining the points can cross each other.

"polyExtrudeFacet" (Modelling | Edit Polygons > Extrude Face): the command used to "extrude" a polygon so that it becomes a solid shape with a specified depth. In the case of buildings, the extrusion height is set such that it is linked to the roof object. When the roof-object is moved, the height of the extrusion changes to match it. One cannot currently change the height of hedges and concrete objects easily once drawn, however this function could be added simply by allowing the user to change the extrusion height for the object (it has not yet been added as a function because there is no easy way to change the width and one would expect to be able to change both.)

polyBevel (Modelling | Edit Polygons > Bevel): The bevel command is mostly used to "bevel" the edges of a 3D solid such as a box (see figure 17). The physical equivalent would be to run a chisel or plane down the edge of a block to give a more rounded edge - or a flat one at 45 degrees. For making roofs, the bevel is set so that its height is equivalent to the height of the roof object. It therefore starts at the bottom of the object (at the edges of the roof) and cuts it back at 45 degrees in all directions. A hipped roof may be formed if the bevels meet at the top. For smaller buildings, the rise of the roof is set such that the bevel will form a hip but for larger buildings the bevel is stopped at a certain height or it would result in an unrealistically high roof. It is easy to created unwanted shapes with the bevel

Figure 17: bevelling a block to create different roof styles



- for example if two bevels collide they create a strange, inverted form at the pitch. In order to guard against this, every edge of a building is checked before bevelling and the height is set such that it creates the right bevel for the smallest span.

Texture mapping:

Maya provides a number of ways to apply texture maps to objects. A texture map is a bitmap image or "texture" that is mapped onto the surface of any 3D geometry. There are a number of mapping options available - for example one may apply the bitmap so that it is "draped" onto the object from a particular direction (generally from above). This works well with flat objects, or objects viewed mainly from one side, but with other objects it will result in long streaks on the faces that aren't in the specified plane (such as the faces of the buildings). Another option may be to create a separate map for each face of the object (automatic mapping - as used for hedges), this ensures that each face is mapped proportionally, but connections between faces may be problematic. Cylindrical mapping (Modelling | Edit Polygons > Texture > Cylindrical Mapping or "polyProjection -type Cylindrical") will project the map radially from a centerpoint. It is best used for cylinders, or similar regular prisms, but it works well for buildings too, since it allows windows to run in a consistent band around the building. Problems exist mainly with indented corners that run towards the center of the "cylinder" as these will appear with a significantly stretched map. Large buildings will also exhibit stretching, but this looks more coherent since all windows are stretched together.

Texture maps are an excellent way of increasing the apparent complexity of a model without slowing down the rendering speed. Maya also allows an "alpha" transparency mask to be used with a texture. An example of this can be seen in the trees where the leaves are rendered as opaque and the spaces between as transparent. This also helps break up the rough geometry of the trees without adding any more polygons.

The texture mapping aims to provide a sense of scale for the surrounding spaces. In this first rendition of DASPE, the buildings do not aim to be realistic, however, as Steven Wilson from the planning firm Sasaki Associates has asserted, people will be "hung up on those ugly buildings" if they do not appear well designed. It seems that most clients or members of the public will have difficulty seeing past the poor rendering quality. There are two ways to address this situation. The first is to write better, more intelligent algorithms for creating buildings from the footprint. The second would be to present the 3D view rendered as a sketch with loose lines and little detail, but with enough information to indicate the feeling of the space. These options are discussed in more detail in the Related Work section under Texturing and Rendering Methods.

Trees:

Trees make use of spheres ("polySphere") and cylinders ("polyCylinder") whose vertices are then selected either individually or in groups and moved and scaled using the "move" and "scale" commands.

Cameras:

Maya allows camera objects to be placed at any specified x,y and z position. Cameras may either be made to point towards a node placed anywhere in 3D space, or have their rotation in all axes set independently. Cameras may be assigned to a certain model panel (or viewport). When one uses the dolly or track tools to change a 3D view one is actually changing the position of a camera. One may set the "focal length" of a camera in order to make it show a wider angle or appear more zoomed in. For example the sun tool uses a 100mm zoom lens to look from the sun, but a 15mm wide angle to look at the sun so as to show as much of the surrounding buildings as possible.

Real-time rendering

Maya is designed as a tool for creating movies with animated characters. The final "rendering" of a scene involves many processor-intensive tasks and is time-consuming. In order that the animator may be

confident that processor time is not spent on renderings that are not what he or she was hoping for, Maya provides a real-time view of one's model that can be made to match the rendering as closely as possible. The real-time model viewer includes options for flat or smooth shading, mapping textures to surfaces, basic representation of lighting, depth map shadows and a few others not used in DASPE. Most of these features have become possible only recently owing to improvements in graphics cards. The advantage of these "hardware renderers" is that the speed of refresh is high enough to be able to experience the model in real-time. However the user changes the view, the representation of the model, complete with textures, lighting and shadows can be refreshed as quickly as necessary for a smooth viewing experience.

This is a mind-boggling achievement of computer power, and it is not surprising that performance drops off visibly when the number of polygons is substantially increased. Performance is likely to be improved further in the future and when this happens it may be conceivable to use realistic models for every tree, building, car and person in the city. As the time of writing, however, while performance is very impressive for a large number of polygons, it is good practice to do all one can to reduce the number of polygons particularly for repetitive objects such as trees and cars that are not the focus of attention.

DASPE Tools

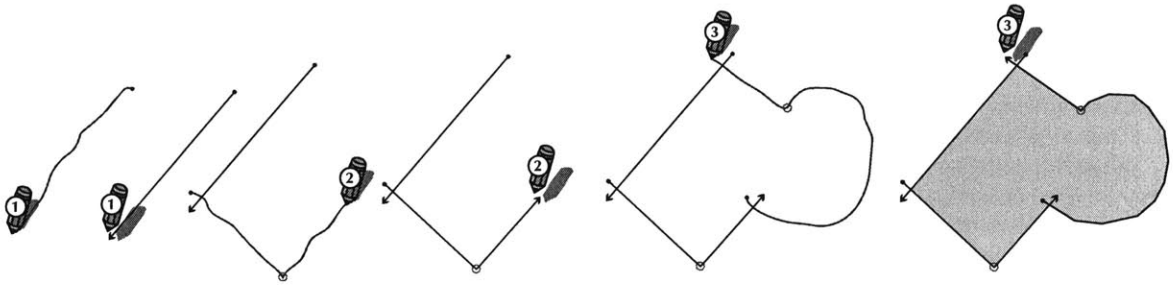
Construction Tools:

1.Building Sketch Tool

This is the main building sketch tool (also referred to as the "pletch tool" as derived from "poly sketch"). It embodies the main principles of this thesis and is the tool on which the most programming effort was spent.

My ambition for the pletch tool was to create a tool that would allow buildings to be sketched, moved, deleted and have their height changed all with little thought to the tool needed to do this. I have mentioned already my idea that efficiency is not always the most important consideration for computerized drawing tools. Efficient methods require thought and effort to be put into deciding on the best method rather than into design thinking. The goal of the pletch tool is to provide a single tool that can be used to create and modify buildings with little thought to the tools used. Gestural interpretation causes less disruption than choosing a tool from a toolbar or palette and its use can become second nature: one merely has to think about moving a building and the "select building for move" gesture is made by one's hand.

Figure 18:The process for drawing building footprints. Note how corners are found and the lines between them are straightened if that segment is found to be mostly straight.



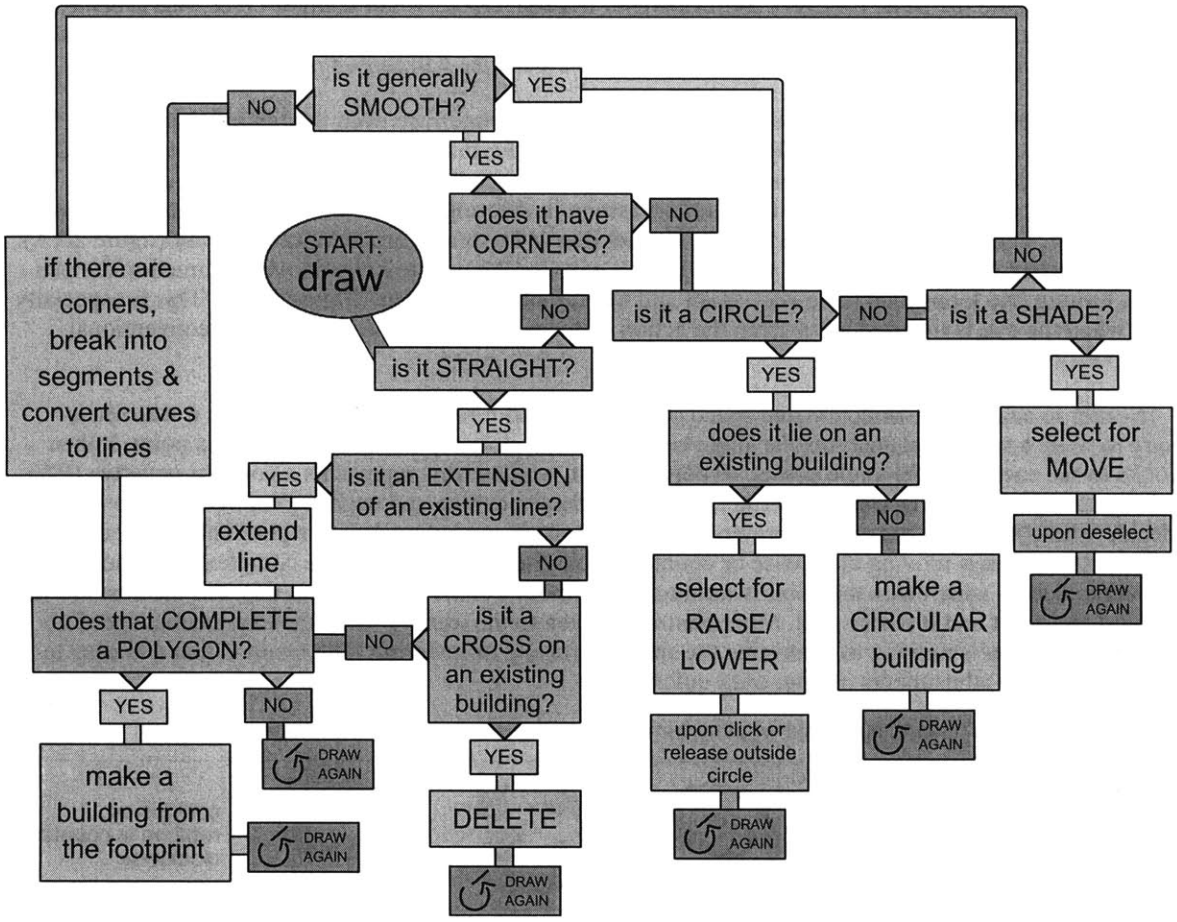
To create a building, the sketcher draws lines for each edge. Once complete, a building is automatically popped up to the height specified and given an appearance appropriate to the selected land-use.

Choices for land-use and height are made in the tool pallette where a slider can be used to enter height numerically without the use of a keyboard.

Figure 18 shows the steps for drawing a series of intersecting lines and curves until an enclosed shape is completed. Completion is determined by looking at the area of a polygon made from a surface trimmed by the lines being drawn. Maya allows a number of lines to be used to trim a planar surface that is created to fit the "bounding box" surrounding all the lines. If the lines do not form an enclosed shape, the trimmed surface will still exist but will be invisible until an enclosed shape is formed. Once the lines are found to enclose a shape and the trimmed surface becomes visible, it will have an associated area value. By watching this value, the script can decide when the footprint has been completed.

If automatic land-use is selected, the outline of the footprint is checked against the land-use zones and the appropriate land-use is applied to the building. The polygon is then extruded to the appropriate height and the line of the footprint is offset to create a slightly larger footprint that is used to create the roof and ground-floor. It is necessary to distinguish the ground floor from the others since this allows a separate texture map containing doors. Furthermore land-use often differs from the lower floor to those above. This is currently implemented only for the "residential above retail" land-use.

Figure 19: Flow diagram for the Building Sketch Tool that allows a number of processes to be selected using gestures.



Select building for move:

A shade action (see figure 13 on page 31 for examples of shades) is made with the pen on the building to be moved. The building is selected and the context set to Maya's move tool. The box used to drag the building is at the point where the shade was made so that shading to select for move and moving the object to its new position can be done with brisk, easy movements of the pen. Clicking outside the box will return to the normal state of the building sketch tool.

Select for raise/lower.

Circling the center of the building will select it for "raise/lower". Note that a circle may also be used to draw a circular building. The distinction is made based on whether or not the circle surrounds a building center (refer to figure 18).

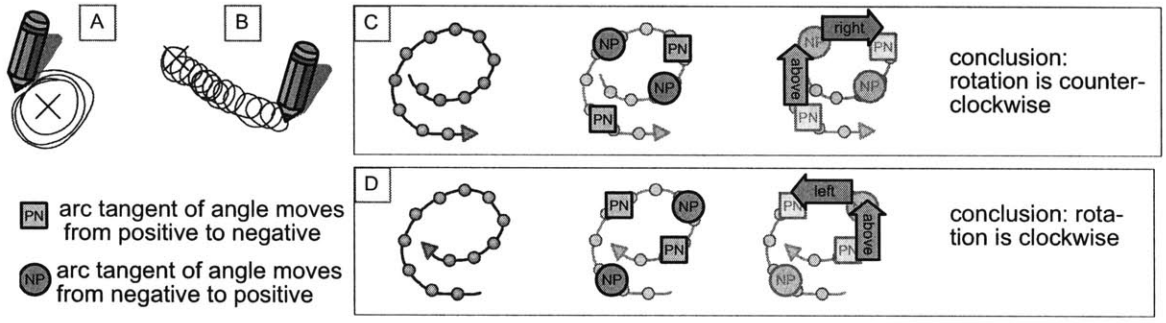
Maya provides no tools for raising and lowering objects in the plan view. However since the interface here is designed such that a 3D model may be created entirely in the plan view, it was necessary to come up with a method for indicating one's intention to move up or down. On a computer screen, drawing vertically can consistently mean "up" or "down", but on a flat table surface, one cannot assume that a line

drawn in a certain direction will always mean up or always mean down. This is especially true when the surface is being used from many sides. For a multisided drawing surface such as DASPE's, there are no directions that are the same for every user. However rotation clockwise or counter-clockwise is always the same for sketchers from all sides. Furthermore there is a universal standard established that turning clockwise will tighten or move in, and turning counter-clockwise will loosen or move out.

I have not seen this interpretation used before in a computer interface even though using the mouse or pen to specify *rotation* is common. Rotation is most often determined by comparing the point at which the pen is placed to a center point and rotating the object to face that point. It would have been fairly simple to write a script that worked in a similar fashion in determining the direction in which the rotation is changing. This would require that the pen should always be rotated around a center point (figure 20:A). However when drawing with a pen, while it is very easy for the user to know in which direction the pen tip is turning, the loops tend to be very small and wander across the page (Figure 20:B). This is especially true when the eye is diverted away from the action of the pen. It was undesirable to use conventional methods of analyzing rotation.

In order to analyze the direction and speed of turning without relying on a centerpoint, it was necessary to write a script that analyses the angle between every plotted point and the previous point. It then looks to see each time the angle passes 90 degrees from whether this represents a positive-negative (PN) or negative-positive (NP) change. Further analysis of these positive and negative changes allows one to decide where it is in relation to the previous change - is it above, below, left or right? This will indicate whether the point is moving clockwise or counter-clockwise. This sounds more complex than one would expect to decide which way the point is turning, but without a static center point, this was the best method I could construct and it works well. More control is given to the user by using the exponential speed at which the pen tip is moving to determine the speed of raising or lowering. The result is that it is easy to make both large adjustments or finer ones quickly.

Figure 20: the logic used to figure out whether turning is clockwise or counter-clockwise.



Delete:

To delete a building one simply draws a cross through it. A cross shape will often be formed when drawing a building footprint. As with the circle used to select for raise-lower, this operation will only be performed if the cross is found to lie on an existing building (see figure 19).

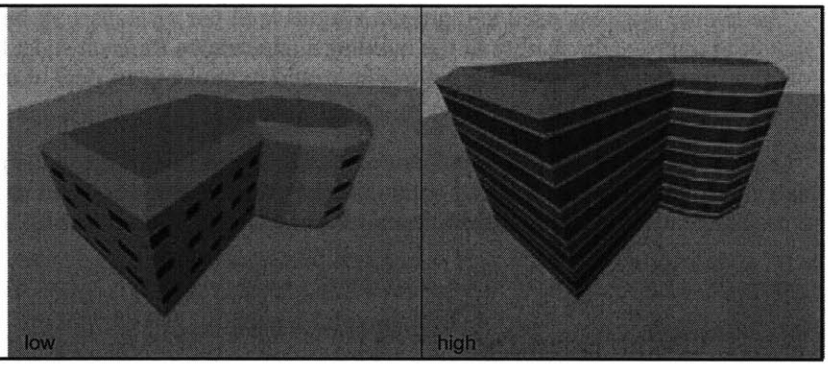
Making the building:

Buildings are made from the footprint in two or three extrusions: one for the roof, one for the body, and, if needed, one for the lower portion at the street level. The roof is bevelled to a certain height depending on the land-use style for the building and texture maps are applied to the other portions. DASPE does not currently allow the user to make any choices about the aesthetic of a building beyond setting its height and choosing its land-use.

Figure 21: texture mapping for different land-use typologies

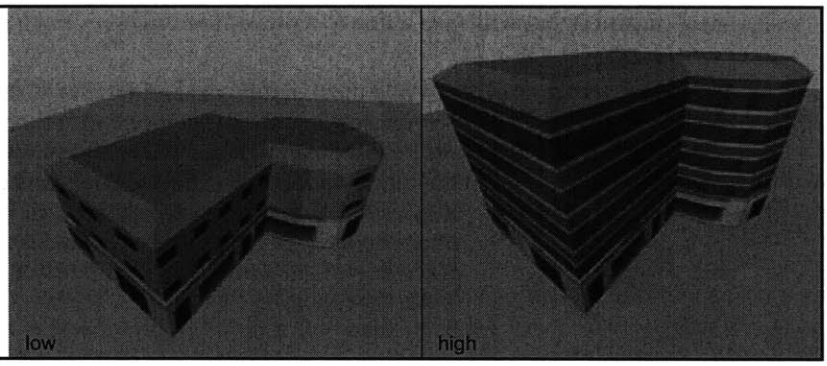
residential

Low residential buildings show a brick facade regularly punctuated by medium-sized windows. High residential makes use of a banded facade loosely characteristic of taller residences.



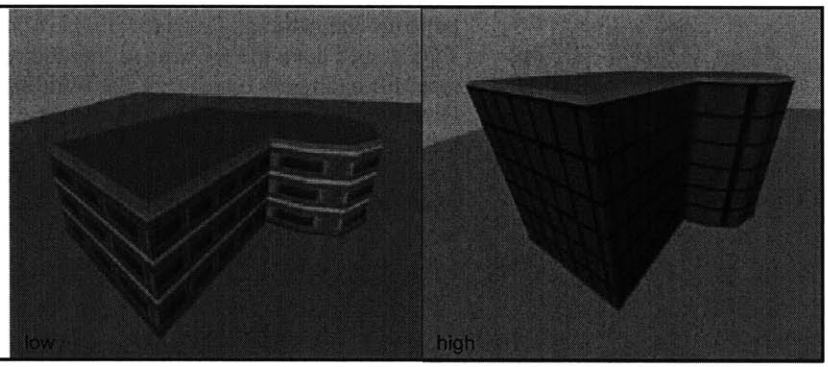
residential above retail

"Residential above retail" simply adds a "retail strip" to the base of the building. Roofs for residential buildings are hipped for smaller footprints or bevelled to a set height for larger ones.



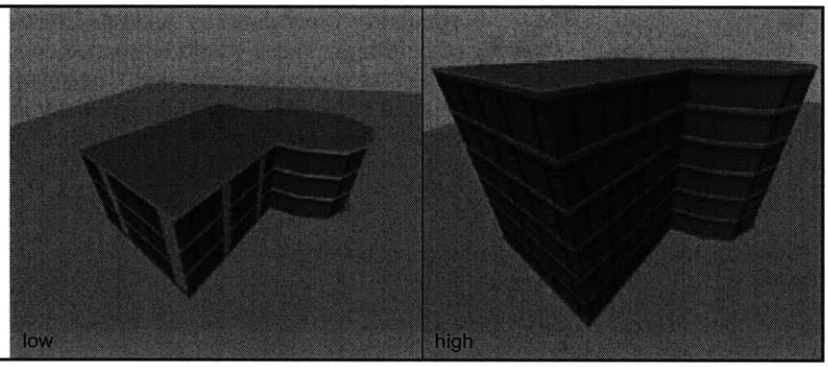
office

Low office buildings make use of larger windows and banding at each floor slab. Higher office buildings are rendered as glass boxes.



industrial

Industrial buildings represent the large floor to ceiling windows that characterize this use. Lower industrial buildings have vertical sections (their structure is outside) while higher buildings show more continuous banding (their structural elements are internalized although floor slabs are visible.)



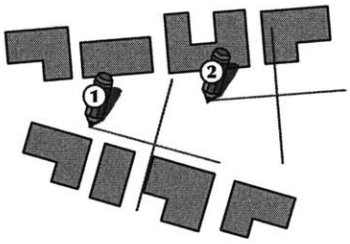
Land-use styles

The texture mapping used to represent various land-use styles is very basic. It would be a fairly simple matter to improve the quality of the building's appearance for each style, but to really capture the essence of land-use and create anything believable would require a great deal of investigation and is beyond the scope of this thesis.

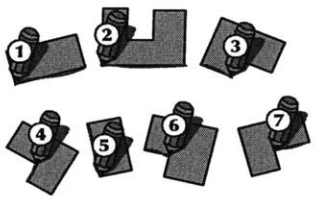
One basic trend in building construction that *is* represented is the tendency for low buildings to externalize their structure and employ solid materials such as brick or concrete in their facades whereas higher buildings tend to use internalized structural elements and "curtain walls" for their facades.

2.Ortho Snap Tool

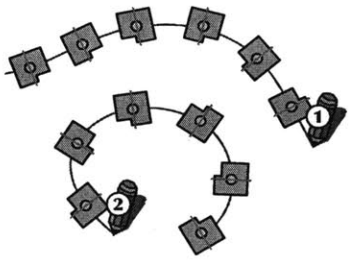
Most buildings drawn by planners are made from rectilinear shapes. It is therefore desirable to have an "ortho-snap" function that will snap a line to a predefined angle if drawn close enough.



While most buildings are drawn with right-angles at their corners, they are very seldom drawn pointing exactly north, east, south or west. In order to set the angle to which the line should snap, the user must specify the direction for one of the sides. This is done by dragging the stylus in the desired direction. A second line will be drawn perpendicular to the drag line through its midpoint for better visualization of the right angles. Once drawn, the ortho angle will remain set until it is reset or turned off. This is the method chosen for DASPE.



Another alternative that would also ensure rectilinear buildings would be to set the ortho angle based on the first line drawn for the building. This would have the advantage that the ortho angle would not have to be reset for each new building if the buildings were at a variety of angles. However, since most planning scenarios involve buildings that are regularly arranged, the first method is preferred as it allows buildings to be drawn not only with right angles at their corners, but also aligned to each other. Ideally one would allow the user to choose which of these methods to use, but this second method has not yet been implemented.



A third method, also not implemented, would be ideally suited to laying out suburban developments where buildings are often arranged in circular or wavy patterns. A single line would be used to place a number of buildings. These would be spaced evenly along the line and rotated to match the direction of the line. The spacing between buildings could be set, and preferably altered dynamically after the fact. This would function very similarly to Adobe Illustrator's "scatter brush" tool or Maya's "paint effects" tool.

Figure 22: methods for setting orthogonal building orientation

3.Land-use Zoning Tool

The land-use zoning tool is used to draw the land-use objects. These are used to decide the land-use for the building sketch tool when it is set to have automatic land-use. If one is drawing a large number of buildings over different regions of a plan, it is useful to declare land-use zones so that one does not have to keep switching between land-uses in the palette. The tool could also be used to draw zones that have more information than land-use alone, for example building set-backs, maximum height or FAR (see section on Future Work). In the case of FAR, one would use this tool to draw a specific lot rather than a broader land-use region.

One uses the land-use zoning tool by drawing either a single line whose end-point is near its start, or a series of lines end to end. These are automatically connected as the user draws the shape. Refer to the sub-section on "zoning tool, grass and water" under Maya Tools.

4.Concrete Pen Tool

The Concrete Pen tool can be used to draw paths, walls or simple representations of buildings. Such basic solids are ideal for creating context for a model by very quickly sketching up simple massing studies. One sets the height and width for the Concrete Pen, then sketches the center-line for it to follow.

The images A through C in Figure 22 show the tool being used to sketch a rough organic form in plan (A) while the Curiosity Camera looks at the results of each stroke drawn (B). The sketcher then switches to the Leashed Camera view (C) in order to move around the space created.

When drawing on varied topography, the walls will be placed on the land so that they start out at a chosen height above the surface. The user may choose whether the walls should then

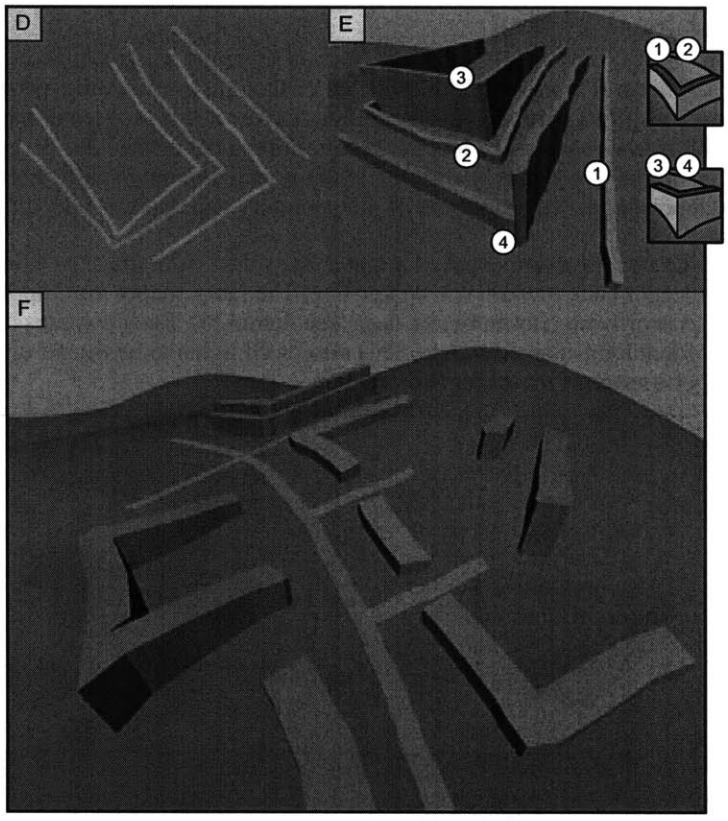
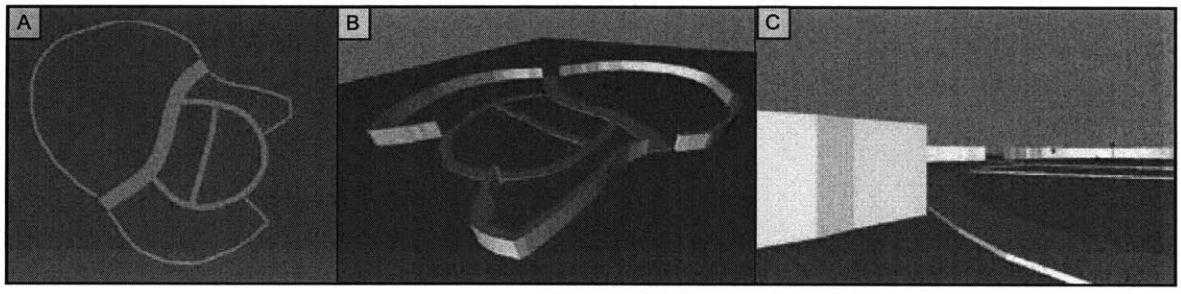


Figure 23: Using the Concrete Pen for paths, walls and a massing study



remain level (horizontal) or should follow the land. The walls drawn in images D (plan) and E (perspective) are all drawn with the same width and height, however walls 1 and 2 are made with the "follow land" option turned on. Level walls (as drawn in 3 and 4 with "follow land" off) are useful for creating foundations, dam walls or buildings set into a hill. Wall 4 is drawn with 2 strokes so that the height is reset for the second stroke whereas 3 is drawn with a single stroke and is therefore level along its entire length. The sketch in F uses a combination of different options and size settings and shows how a rough massing study may be quickly sketched and viewed.

5.Roads

Roads are automated from a single line. The user may set the desired width (in lanes), choose a style for the lamps to be placed on its edges and decide which sides of the roads should feature parking lanes if any.

This tool works well for the level of thought at which the designer thinks of the road in fairly basic terms. In its current implementation, however this tool would be frustrating if the designer wanted to show the roads at the level of resolution typical of the "design guidelines" documents planners are often required to produce. However, at the sketch planning stage, this tool provides a quick and easy way to represent the large amount of 3D geometry necessary for depicting roads..

Lamps are evenly spaced along a line offset from the center line. They are rotated so that they face straight back towards the center-line. The lamp shapes are imported from a certain file depending on the style of lamp chosen by the user (see figure 26). Each contains a light object that is linked to a central, global light-intensity node. This allows all lights to be turned on as the sun sets. It is therefore possible to get a sense of the street at night.

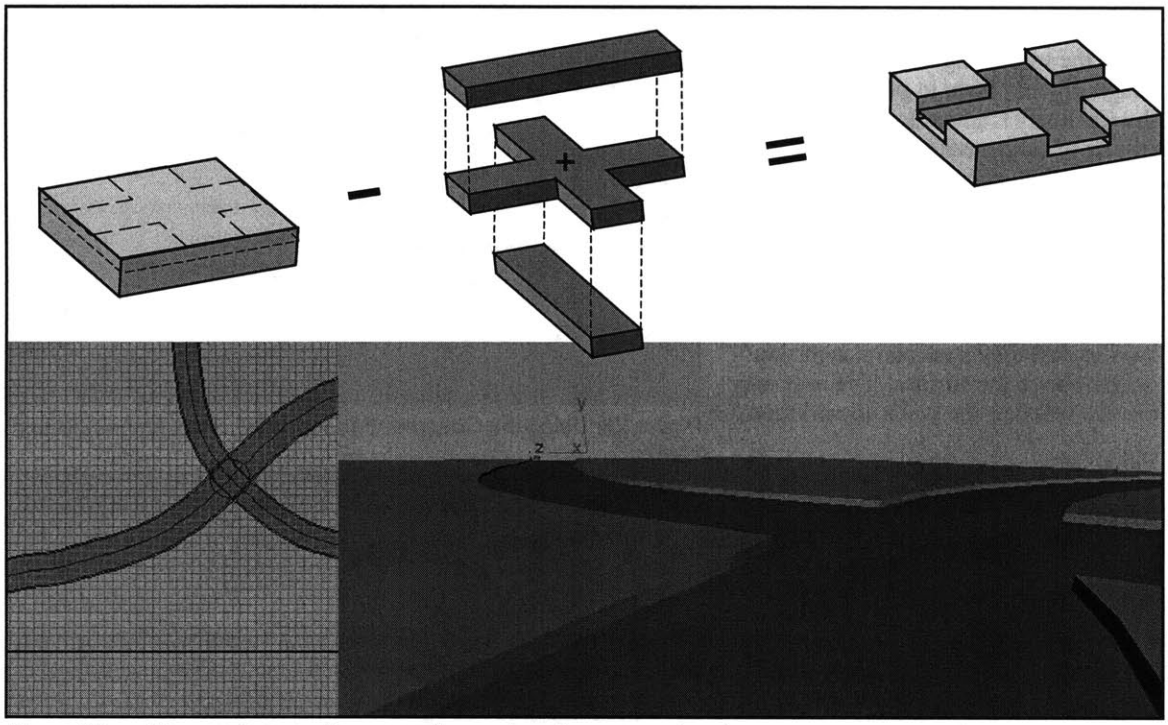


Figure 24: cutting roads away from a paving surface to represent curbs

Parking meters (see figure 25) are placed along the edge of the road in a similar fashion. The meters are randomly loaded from a selection of files each containing a meter as well as a variety of parked cars. This is a simple way of representing most elements in a streetscape. It was tempting to use some of the realistic 3D car shapes available online, but in the interests of minimizing the polygon count it was necessary to create new car shapes with very few polygons. As processor speeds continue to improve, it is hoped that more realistic cars could be used instead without compromising the refresh speed.

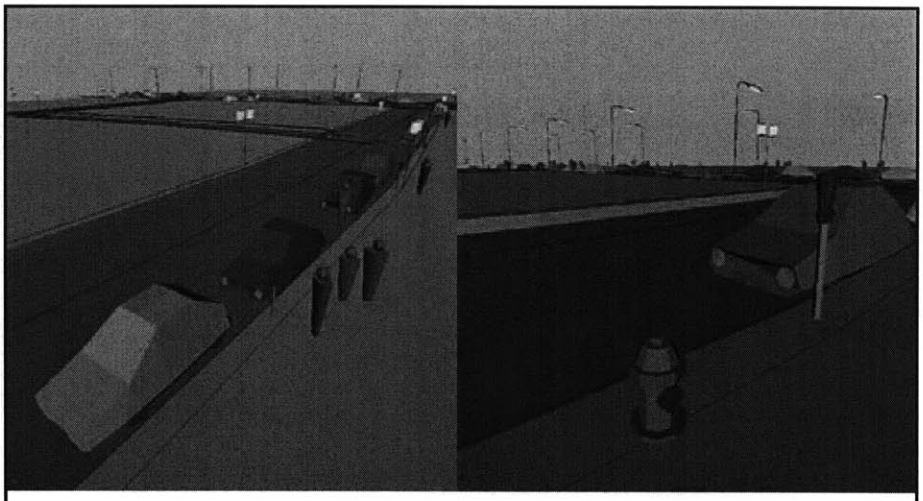


Figure 25: Lamps and parking meters are placed down the side of a road at a set spacing. The meters are randomly loaded from a series of files. Parked cars, hydrants and people are also contained in these files and are thus randomly placed on the edge of the road.

Where roads intersect, it is necessary to remove the features at the intersection so that these do not appear to float in the middle of the intersection. This is done by finding the point of intersection and removing all features within a radius that relates to the width of the road just drawn. These objects are placed on a hidden trash layer so that they may be brought back if desired.

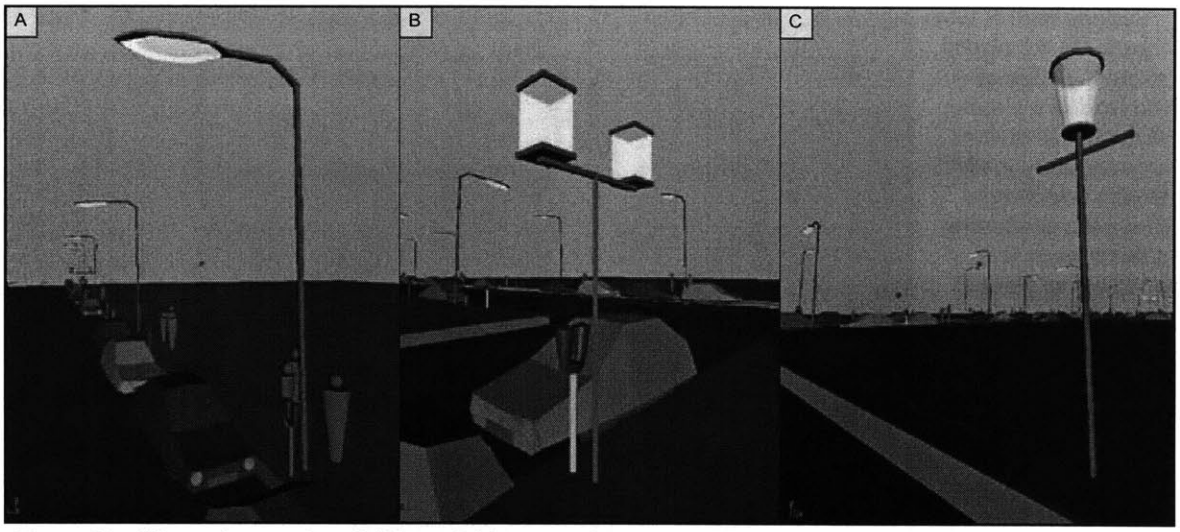


Figure 26: different lamp styles, A-Urban, B-Nouveau, C-Victorian

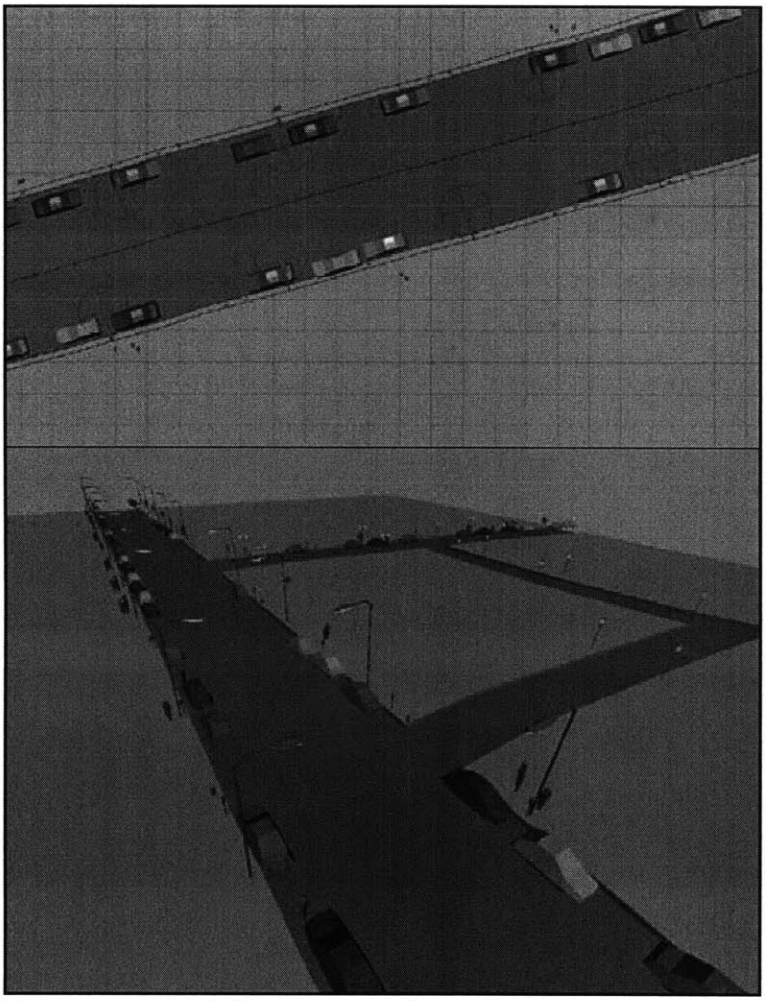


Figure 27: the resulting roads in plan and perspective

The roads themselves are cut out of the paving surface. A grey "plane" is revealed by the cutting operation. This is the surface of the road and can be seen in both plan and perspective (see figure 27).The perception is therefore that one is drawing with grey road lines, although in reality a "boolean" subtraction is used to cut a positive road extrusion from the larger paving object (figure 24).

Many other features could also be associated with roads. For example, in future version of DASPE, building setbacks could be used to force buildings to be placed a certain distance from the road or lotting diagrams could be generated based on the road center line. Indeed a number of automation features could be added to this interface. Such automated tools are common in CAD packages and are not the focus of this thesis. If any thought is to be given to a process, automating it may be as limiting as it is liberating. However it is ideal to be able to automate a process that is mindless and repetitive.

Landscape tools:

1.Water

The water tool is used in exactly the same way as the land-use zoning tool. Once the shape is complete, the surface makes use of a "phong" shader that gives it reflectivity in the modelling view. When drawn on existing topography, the water object will be placed such that its center lies on the land. Some parts will be obscured beneath the surface of the land while others will hover above it. The 3D Grass tool may be used to sculpt the land to accomodate the water and the Concrete Pen tool may be used to create a dam wall if desired.

2.3D Grass

3D grass makes use of the same scripts as the water and zoning tools in order to create a colored surface, but then takes an additional step in that the NURBS surface is converted into a polygonal mesh. The grass may then be sculpted with Maya's "Sculpt Surfaces" tool that allows the user to "sculpt" a surface by choosing either to push or to pull vertices when they "rub" the surface with the mouse pointer or pen. This is useful for creating rough 3D topography such as berms or small hills.

Ideally this tool would be adapted to conform to the concept of using clockwise rotation to push down and counter-clockwise rotation to pull out, but this would need to be implemented in Maya at the "plug-in" level. As it stands the user must choose whether they would like to push, pull, smooth or erase from a tool palette. The tool is automatically selected and set to "pull" whenever the user creates a new grass object. Note that it is sometimes difficult to tell how one is affecting the topography when drawing in plan view. Unfortunately the perspective view is only updated after the drag operation is complete and the sculpt surfaces tool therefore works best if used directly in the perspective view. In this particular case, it is desirable to break the rule of drawing in the plan view only, but it should be noted that this would not be necessary if the aforementioned plug-in could be developed.

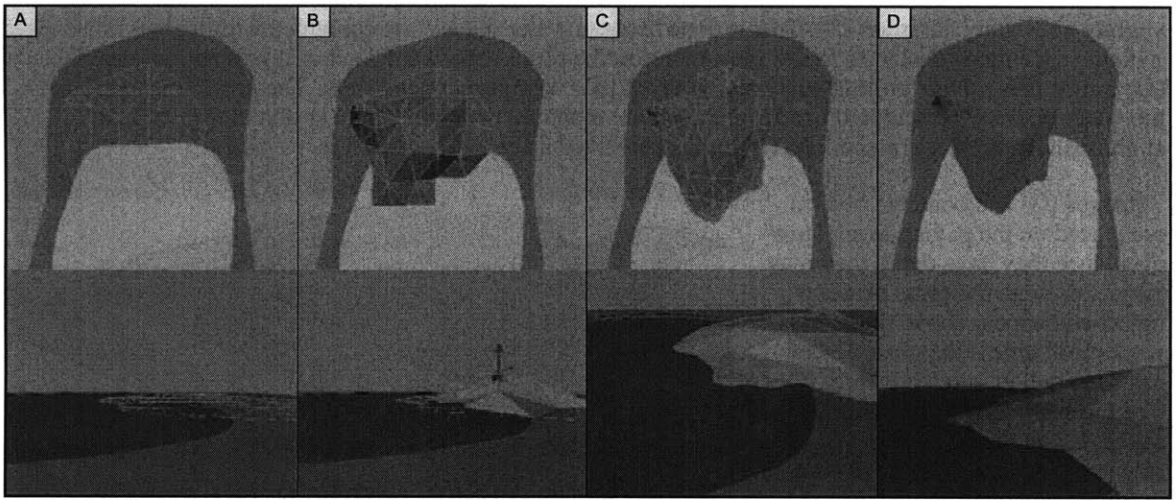
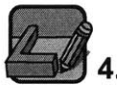


Figure 28: making a hill with Maya's "sculpt polygons" tool



4.Hedges

The sketcher chooses the height and width for the hedge then draws a line for it to follow. Hedges are an important screening element in city design. DASPE allows the user to choose between solid hedges (as shown in figure 29) or "see-through" ones (not shown). From the leashed camera view (B) one may appreciate the difference in spatial quality provided by hedges that are above one's head or below it. Hedges will always follow the topography on which they are drawn (they are very similar to objects drawn with the concrete pen when set to "follow land").

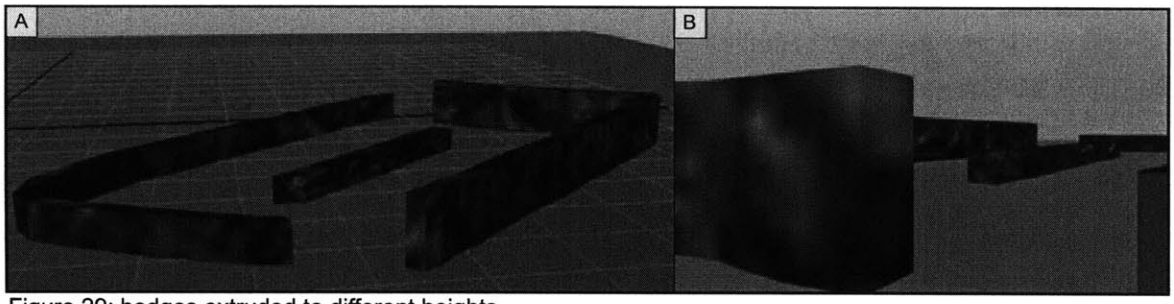


Figure 29: hedges extruded to different heights.

Layering

As shown in figure 30, roads (F) are cut out of the base (A) so that the curb-cuts may be represented on the edges of the roads. This could also have been done using a raised sidewalk object on either side of the road, but the latter method would not work at intersections. It may not be exactly true to life that the sidewalk level of pavement (A) should be a base object covering the entire site, but it works well since adding grass (B, C, D) seems more natural if done as a "positive" object rather than a negative one.

Similarly, although water (E) always occurs below the land which surrounds it, it is actually drawn slightly above any flat grass (B). This is done because water objects are contained within grass in all cases with the exception of islands. Water must therefore be placed above the land or it will not be seen. Islands (D) can be created by "pulling" the grass with Maya's "sculpt polygons" tool. The grassed land around the water (C) may be sculpted in a similar fashion. It should be noted however that the water is only slightly above the flat grassed land (B) and is perceived as being flush with it.

Hedges (G) and concrete objects are placed on the paving level. However since they have depth they may project through the grass or water object and appear above them. This works well unless the grass object has been pulled into a hill form (C) (see the section on Future Work called "Capitalizing on 3D" which discusses how this may be overcome in future versions.)

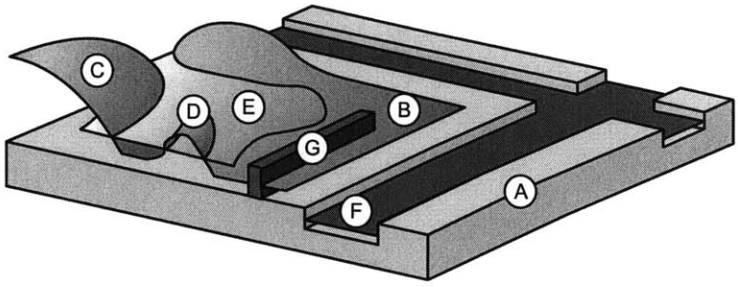


Figure 30: layering solution for representing roads, sidewalks, grass and water

5.Trees

Trees are comprised of basic cylinders and spheres that are warped to form branches and leaves. The sphere contains only eight divisions vertically and eight horizontally (twenty is the default for a reasonably smooth sphere). It is therefore extremely rough and this is only heightened by randomization of its facets, however Maya allows texture maps to be rendered with transparency in the modelling view. Applying a texture of leaves with a transparent background to the sphere breaks up the appearance of the geometry so that the reduction of facets is not bothersome. While texture-maps may appear more complex to the eye, they are very efficient to render when compared to creating the same complexity with a large number of polygons. The advantage of this is that the sketcher can place fairly realistic trees quite liberally without worrying about reducing system performance (see figure 31).

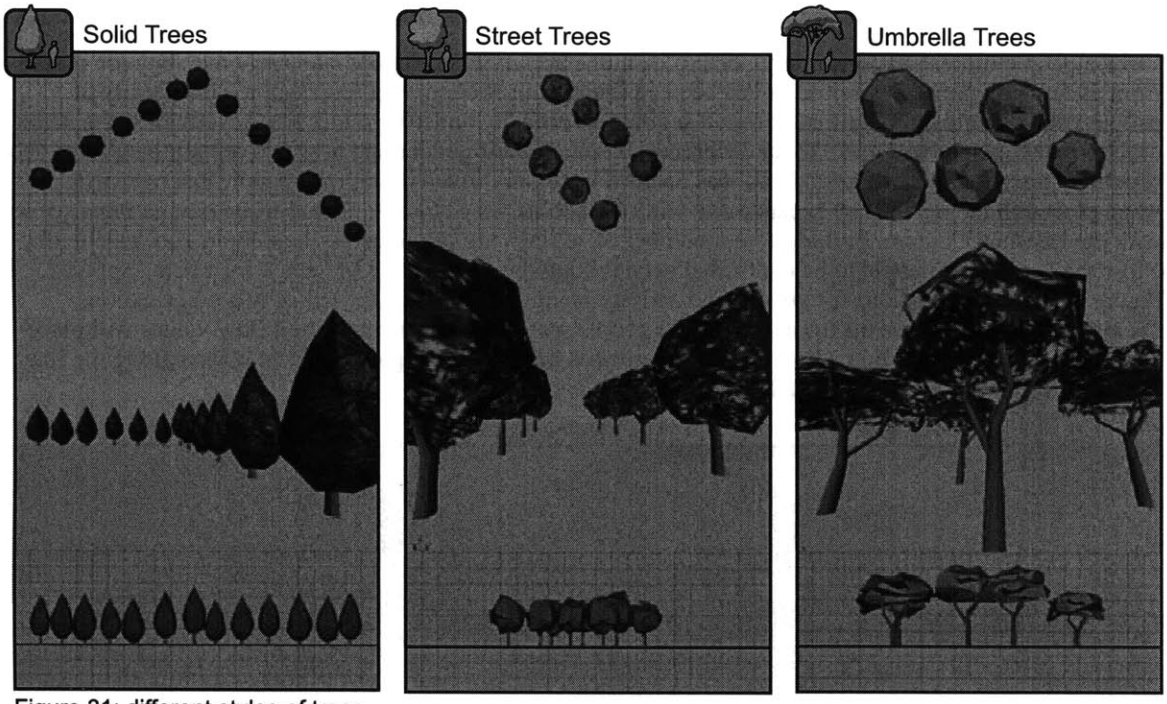


Figure 31: different styles of trees

It should be noted that Maya does contain a sophisticated tool for easily creating complex organic forms such as trees and grass from simple brush-strokes. The rendering quality for these objects in the final render is excellent. However, since these make use of a number of "tubes" each of which must be rendered individually, Maya displays only a simple representation of them as a series of lines in the modelling view. Since DASPE relies on this view for viewing the scene in realtime, this is unsuitable for achieving the appropriate sense of what those trees contribute.

6.People Spray

The people spray tool allows the user to put representations of people into the scene by drawing a line along which they should be placed. People are extremely small in the context of most urban design projects and can barely be seen at most scales used offered by DASPE (and commonly used by planners). However when moving around the scene and appreciating the spaces created, the inclusion of people is important to creating a sense of place. It has therefore been desirable to offer the sketcher a simple means with which to populate spaces with rough representations of people. These also serve to add a sense of scale to the model.

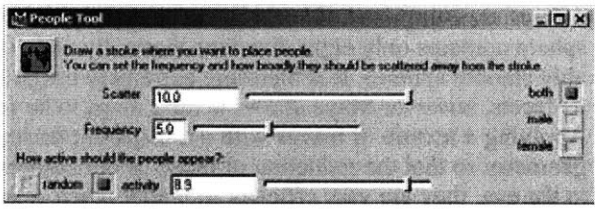


Figure 30: tool options for the People Tool

Since people representations might need to be numerous in order to populate a scheme, they are made from as few polygons as possible. It has been an interesting challenge to represent a wide variety of people with a single cylinder made from 8 segments vertically and 10 around. But nevertheless the script used to make people will make them differently based on their gender and how active their hands and feet should appear. Such attention to detail was deemed necessary (even though the people can barely be seen when placed in the plan view) because one will often come very close to people when navigating a scene with the Leashed Camera. Furthermore a number of settings are provided on the People Tool palette that allow the "spray" of people to be controlled even though they can barely be seen whilst being sprayed. Figure 32 shows the options for gender and activity mentioned already as well as two other options. "Scatter" determines how far from the line the people may be randomly scattered (low values will place all people exactly on the line. "Frequency" determines how often people should be placed along the line (a higher frequency will result in a higher density of people.)

Figure 32: Images A and B contrast the different spatial quality when people do or don't populate the same space. Image C shows how people's heights and genders may vary. Person 1 is female, while person 2 is male. Note also that the people walking on the path (in C) have been made with a higher activity setting than the people in standing by the path (in A and B).

Visualization:

One of the main advantages of building 3D models on a computer is the ability to place cameras in places that are not easily accessible in physical models. For example one can place a camera at eye level and move it around to any point on the plan. This allows an on the-ground experience to be represented.

In Maya's GUI, one can position the camera using the move tool and rotate it to give the desired view. This is sufficient to achieve any static viewpoint, but in order to create a more dynamic visualization experience, it has been desirable to create new tools that are more appropriate to a sketched based interface.

1.Sketched View

The "sketched view" tools make the placing of a camera and its view point a matter of drawing strokes with the pen. This is a natural way to gesture the desire to either "look this way" (by drawing a straight line where one wants to look from and to) or "consider this from here" (by drawing a node followed by a line). In the latter case, the camera aim object will be placed at the node of interest while the camera itself will be placed at the end of the line and selected for use with the move tool. This allows one to move around a point of interest and consider it from all angles. For an example of its use see image 8 on page 59 in the Use Case.

2.Leashed Camera

The leashed camera attempts to overcome the problem of specifying the camera's position and rotation simultaneously. Usually these two actions are separated, and it is not easy to do both at once in a natural fashion. The leashed camera takes its inspiration from the practice of using animation path curves to create a "fly-through." In these a camera is made to move from the start to the end of the path and is most often rotated to match the path's direction. The leashed camera basically takes this idea and attempts to implement it in real time. As the line is being drawn, so the camera is moved along it a short distance behind. The camera actually rotates to face the point being drawn, not the direction of the line as such, but the experience would be very similar in both cases. This is achieved by placing a camera aim object at the position of the stylus. Cameras will always point at their aim object so this can be used to set the camera's rotation. All of the points drawn are stored and the camera object itself will lag a number of points behind the aim object.

The number of lag points used depends on the velocity at which the stylus tip is determined to be moving. This is important because a large number of lag points will delay the camera's catching-up too long when the pointer is moving fast. It therefore becomes difficult to understand the cause and effect when looking at the perspective view. A large number of lag points are necessary however for smooth motion when moving the stylus slowly.

The leashed camera can be used to simulate the experience of moving through any of the spaces created and can be used for almost all on-the ground visualization requirements.

3.Analyze Sun

This tool allows one to consider the affect of the sun on a site. The URP interface (see page 20) demonstrated how shadows could be dynamically projected by the computer for models on the table for any specified time of day and year. DASPE provides similar shadow analysis capabilities and adds some features.

1. Single slider object for whole year.

A trapezoidal shape (see figure 33, images A and D) is used to represent the daylight hours for the entire year. Horizontal space represents time of day. Thus the narrow end represents winter and the broad end the longer hours of summer. Each point in the shape actually represents the two times of the year when the sun is at the same position. The horizontal line through the center represents the equinoxes in spring and autumn. The left and right edges represent sunrise and sunset respectively. Using this shape it is really easy to consider the path of a day at any time of year, or just as simple to consider the difference from summer to winter for a set time of day.

2. View point of interest from sun.

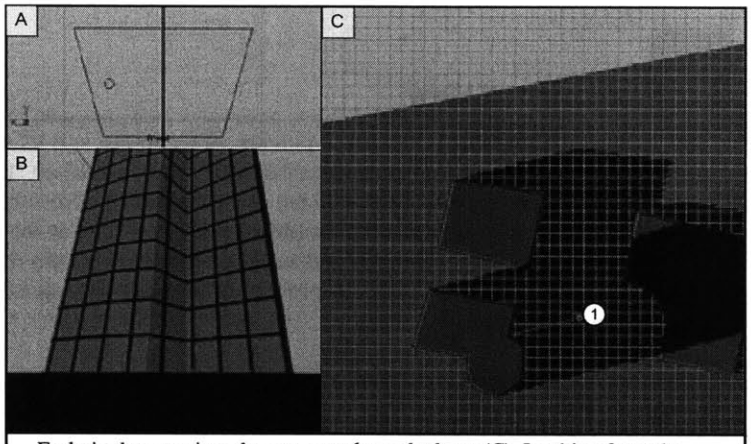
Placing a camera at the position of the sun object allows one to view a point of interest as the sun would see it. Any object that can be seen from this view would be in sunlight, anything hidden would be shaded at the specified time.

3. View sun from point of interest.

Switching the positions of the camera and view object allows one to view the sun from the point of interest. This allows one to quickly assess the sun's impact on a particular spot at different times of day and year.

A separate window contains the trapezoidal shape. Dragging in this window will place a circle at the sun's location in the diagram and will rotate and move the solar system object accordingly. The point of interest for both centricities is set by clicking or dragging in the plan view. Maya does not directly allow for different operations to be performed in different views, but checking the window's identity as it is clicked allows one to switch the procedure called by the drag command of the tool.

A "directional light" is placed directly behind the camera and represents the light of the sun. The light may be used to cast depth-map shadows that give an accurate, but pixelated representation of the shadows cast by the sun (see figure 33 C and H). These shadows may be seen in any view. Further improvements in video card technology will hopefully improve the resolution of the depth-map and reduce the pixelation. Note that the current version of Maya (4.0)



Early in the morning, the sun casts long shadows (C). Looking from the point of interest (C1) toward the sun, one looks straight at the tall building (B). The sun first shines on this point during the equinoxes at around 10:20am (see below). This information is displayed in all the GUI windows and shadows are turned off during the drag operation (see D, E and F). Upon completion of the drag, the shadows are displayed (H). These are true depth map shadows. They are pixelated, but have the advantage of projecting onto buildings or topography correctly (notice that the shadow edge at H1 is closer to the building than the shadow edge at H2 even though it represents the same edge of the roofline).

Once the point of interest becomes exposed to the sun it can both be seen from the sun camera (G) and the sun can be seen from the ground (E). Learning to understand the implications of the human-centric and solar-centric views can allow for highly sophisticated understanding of the behavior of the sun on one's site in relation to one's design decisions. It also allows for well informed placing of objects that depend on sunlight - for example trees or play-grounds.

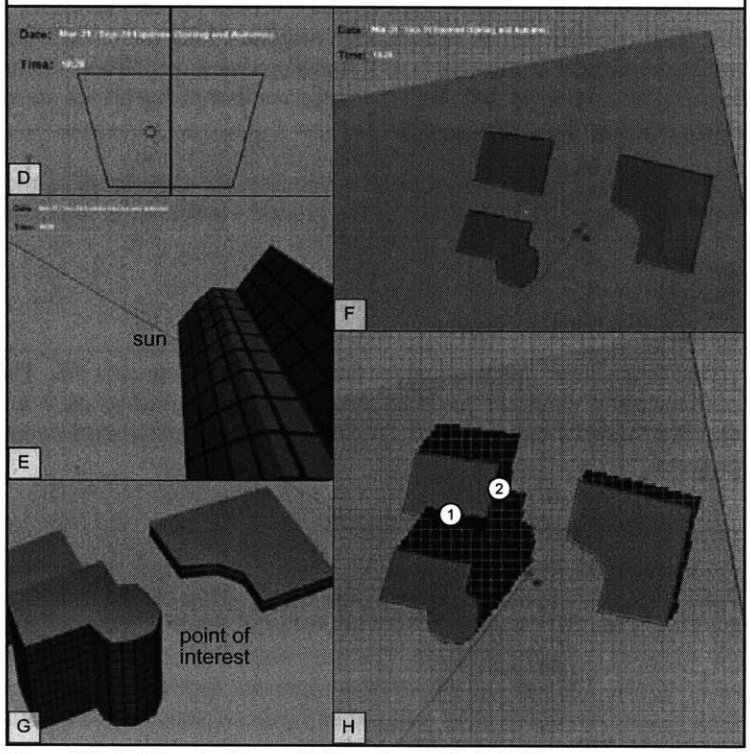
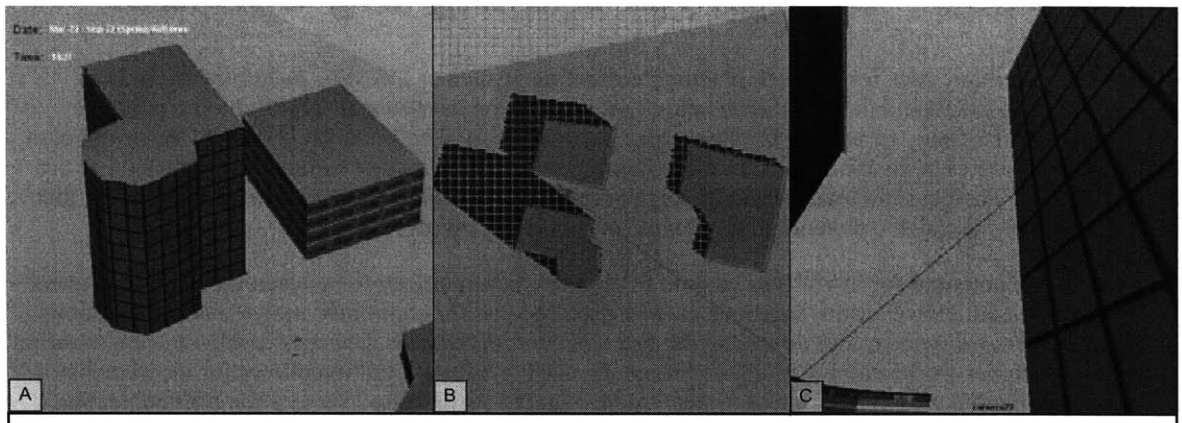


Figure 33: studying shadows with the solar analysis tools

Figure 34: the sun first becomes visible at a new point of interest



One may change one's point of interest simply by dragging in the plan view. The perspective view will change during this drag so one can quickly see when the view becomes obscured even though the shadows are turned off during a drag. In this diagram the sketcher has decided to analyze the conditions in the space between these buildings. The sun first shines here at 2:27pm around the equinoxes.

contains a bug that crashes the program on Microsoft Windows 2000 if depth map shadows are turned on while dragging. Shadows are therefore turned off during a drag operation and are turned back on upon its completion.

4.Curiosity Camera

Plans are inherently two dimensional so when one draws 3D objects in plan it is difficult to get a sense of their height. One method to show height in plan is to represent it with shadows that lengthen as the object gets taller. This is more useful for a *relative* sense of height as shadow length may vary according to the direction of the light. Ideally one would like to see a perspective or isometric view of the object being drawn, however placing the view so that one can always see what one is drawing can be tedious. The Curiosity Camera attempts to provide such a view automatically by flying over to the position of the newly drawn object.

This view becomes essential when using the raise/lower tool. Recall that this tool is used to change the height of objects in the plan view using the convention of clockwise to mean down and counter-clockwise to mean up. While one has a sense of how quickly one is changing the height of the object and of the direction of this change, one needs to rely on the perspective view provided by the Curiosity Camera to get feedback for one's actions.


One may opt to manually choose an object in the scene in which the Curiosity Camera should take interest. This provides a quick way of navigating the plan from one object to another and helps one keep a sense of which objects in plan correspond to which objects in the perspective view.


The Curiosity Camera may be disabled if one would rather use a set view (such as one specified with the Sketched View Tool) to assess the visual impact of creating a new intervention or changing its height. The software will still keep track of the "object of interest" and turning the Curiosity Camera back on will immediately cause the camera to fly over to that object.

Use Case - Design for the Big Dig Corridor

There may be many uses for a sketch planning environment such as DASPE. As argued by Bulmer (2001) one such use would be to facilitate better communication between designers and the public or client. The interface may be used to bridge the gap between those who understand designs shown in plan and those who prefer to visualize them in more familiar terms. This is done through the provision of perspective views. The possible uses for facilitating communication and encouraging conversation that this provides are very broad and depend greatly on the personalities involved.

Similarly, the potential for DASPE to be used to facilitate communication between groups of designers has been discussed. However this interface may be used not just for communicating ideas and decisions, but also for *generating* ideas and *informing* the decision making process, This use case looks at a situation in which a designer works alone with the interface. Focussing on this situation allows for the demonstration of DASPE in its role as a design assistant, independent of its promise as a tool for communication.

This use case does not attempt to be a comprehensive demonstration of all the tools available in DASPE, but rather tries to illustrate the use of this software from the perspective of the designer. The designer's use of DASPE tools and traditional media are shown step by step as he/she goes about making design decisions.

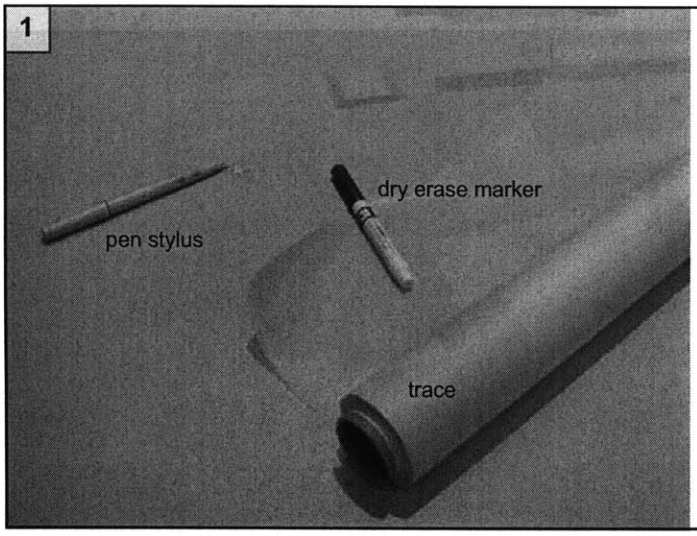
The Big Dig Corridor deals with the land that will be recovered once the Central Artery highway in Boston has been successfully buried. This site has been chosen for this use case because it is relevant and interesting to planners and also because the data for the base was readily available. The base model comes from a larger and very comprehensive model of the city produced by Urban Data Systems. Figure 6 (in the introduction on page 7) shows the base for this model in plan, perspective and elevation with the Customs House highlighted in red. This kind of model is perhaps not ideal for representing *context* as it contains many polygons and is not texture mapped with photographic information (refer to discussion on 3D Visioning Systems under Related Work, page 16). The base is however sufficient for this use case and Maya handles this scale of model quite capably with only a barely perceptible reduction in general performance.

The designer will be considering a key point along the Big Dig Corridor. This is the point where the Central Artery broke State Street's historic connection to Long Wharf. Faneuil Hall is nearby and the site lies on the route between this popular tourist attraction and attractions on the waterfront which include the aquarium and ferry terminal.

Christopher Columbus park also adjoins the site. This park currently features a modernist landscape by Hideo Sasaki, but is in the process of being redesigned to be more "neighborhood friendly" (Flint 2002). Our designer has decided that his/her design approach will look to celebrate the submersion of the Central Artery highway and capitalize on the new views back towards the iconic Customs House. This example demonstrates how design decisions may be made based upon analysis conducted with DASPE's visualization tools.

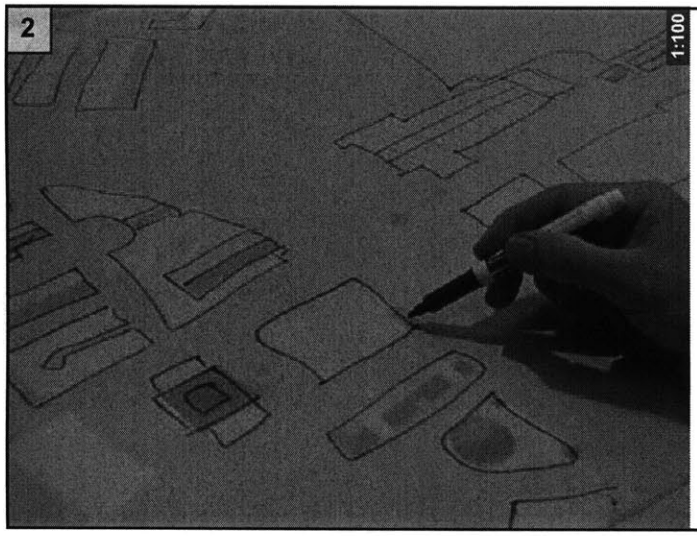
Another concern that our designer will address is the separation of the waterfront from the rest of the city. More specifically he/she is concerned with reconnecting State Street to the historic Long Wharf. These were separated to make way for the construction of the Central Artery overpass in the early 1950s. The designer has decided that creating a consistent street edge along State Street is the best way to celebrate this renewed connection. This may be prove to be more difficult than it appears in plan since the design of the Marriot Long Wharf hotel on the wharf side of the overpass made no attempt to maintain alignment with State Street. As a result, some skillful manipulation of building placement and massing may be necessary to achieve the desired affect.

Figure 35: Big Dig Corridor 'Use Case.' This figure contains 29 images with textual annotation.



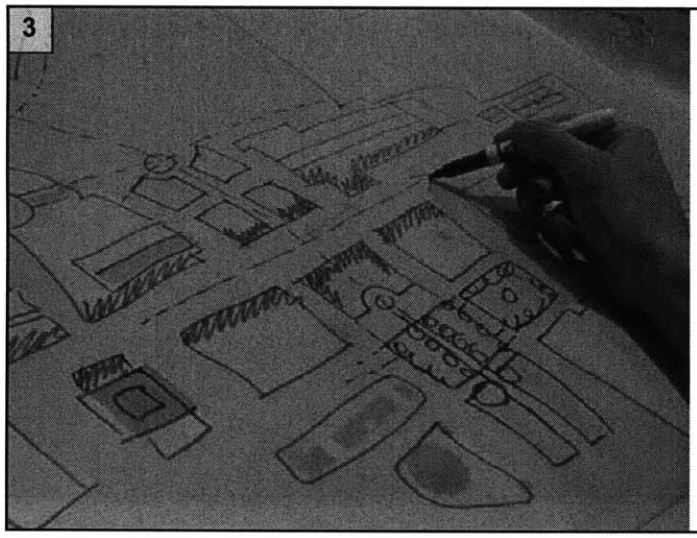
Projection and Trace

"Trace" paper can be used on the table in the conventional fashion. The projected information simply becomes a layer of information such as a base map. A dry-erase marker is used as it is easy to forget about the edges of the trace when looking at projected information.



Context Tracing

It is common practice when using "trace paper" to roughly trace context lines. This serves to both highlight important contextual information and ensure that the page may be easily realigned if shifted

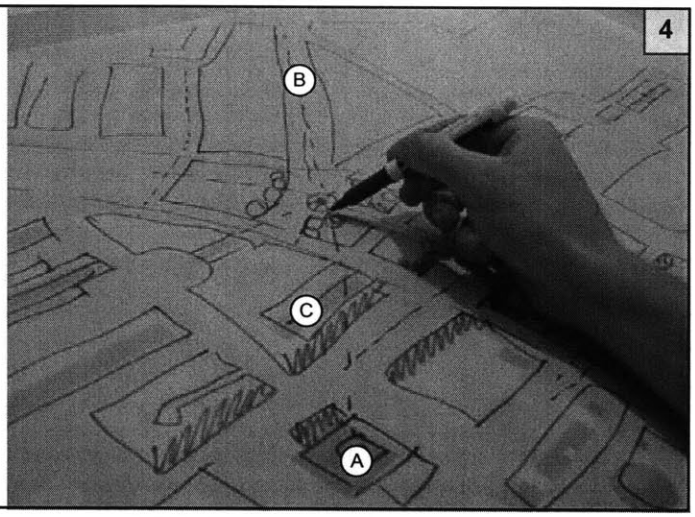


Design Sketching.

Designs may be sketched using a regular marker on trace. This process is independent of the software so any medium may be used.

Desirable View Line.

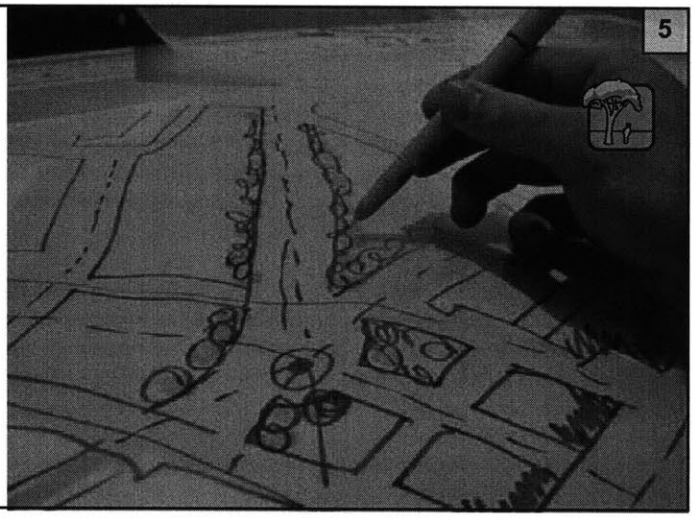
Here the designer has decided that the view of the Customs House building (A) should be celebrated. The central path through the new park (B) is oriented such that it forms a view corridor that terminates in the Customs House. The designer does not know at this point to what extent the tall building (C) will obscure the view or if the Customs House is visible from the path at all.



Drawing Trees.

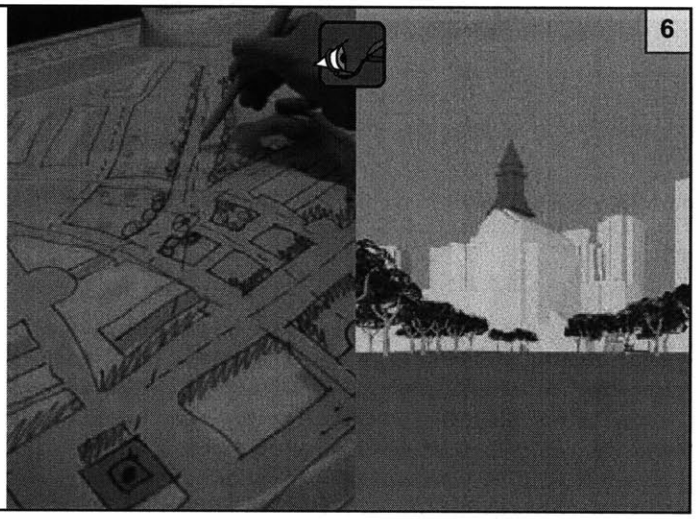
In order to draw trees with DASPE, the user simply traces his/her tree-scribbles with the pen stylus. The tree tool will place trees along the drawn line at a specified spacing chosen by the designer.

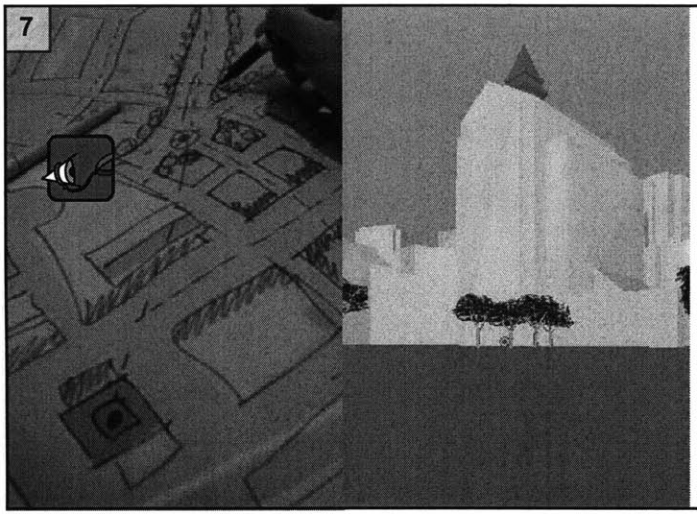
Note that the trees appear faint here because the projector is "competing" with bright neon lights. Dimming the lighting in the room would make the projected information more dominant.



View from ground.

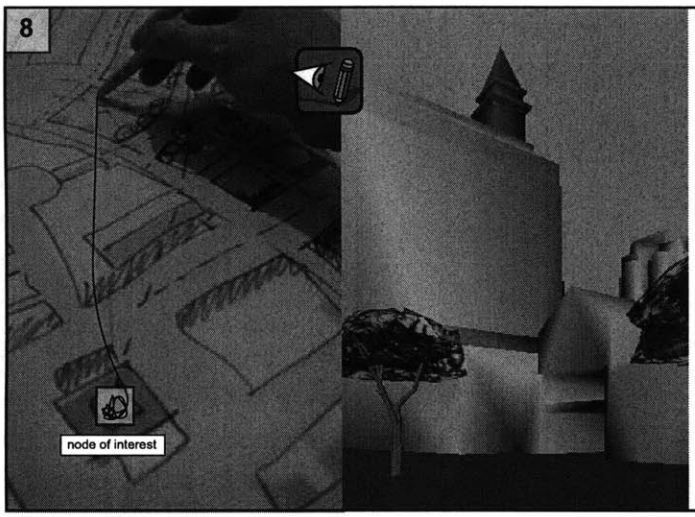
Dragging the "leashed camera" down the path will show the on-the-ground perspective moving down the view corridor. Highlighted in red is the Customs House tower in which the designer is interested. At this point on the plan, the tower is still sufficiently visible as to provide a desirable view.





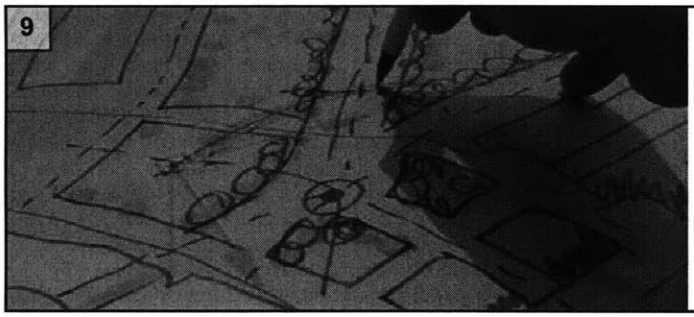
View becomes obscured.

The designer moves the "leashed camera" along the path towards the tower until he/she decides that the view is no longer interesting. This point is marked with a line across the path (although in reality the designer knows only about the one point where the camera is, not about all the points for that line.)



Point of Interest

In order to better determine all the points at which the Customs House becomes sufficiently obscured as to be uninteresting, the sketched view tool is used to draw a node of interest on the Customs House. The viewer's position may be then dragged with the stylus to any point in order to look back at the tower.



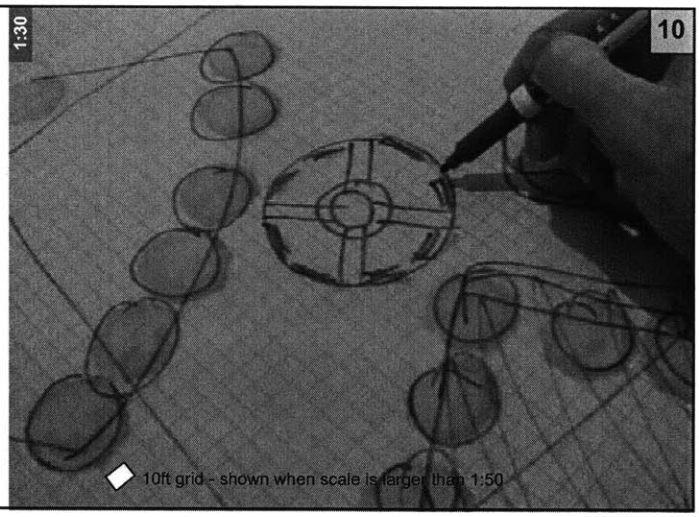
Line of obstruction

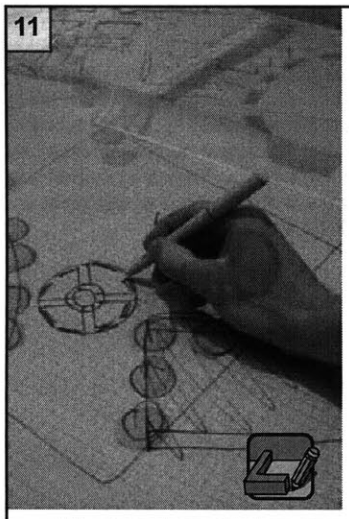
It soon becomes obvious where the line of obstruction lies. This line may either be remembered and sketched in (as shown) or it may be desirable to run the ballpoint pen alongside the stylus in order to draw sketched lines while changing the view.

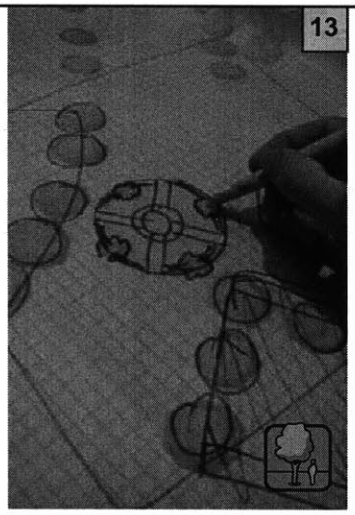
The designer will also notice that the customs house will seldom be obscured by trees before the building in front obscures it. This may not have been obvious in the plan view. Furthermore since the building in front of the customs house has been designed to face onto a busy highway and presents an unattractive facade to this site, it may be desirable to use methods such as these described here to place trees in such a way as to best soften or obscure the facade of the building.

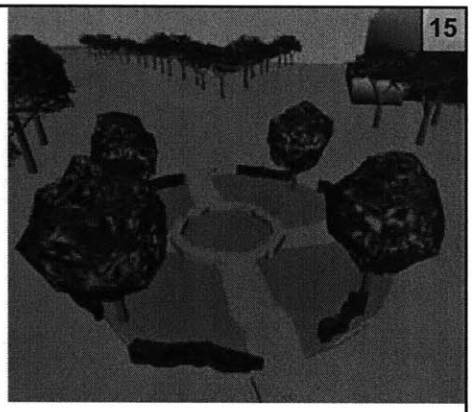
Zooming in

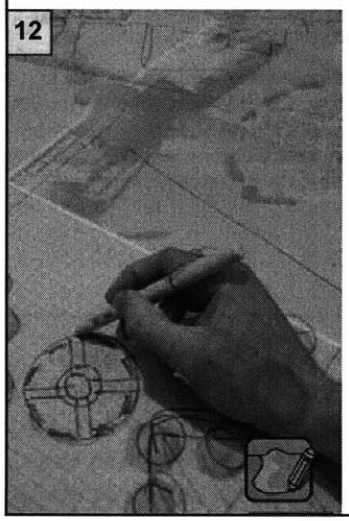
The designer has decided that the experience of the view corridor cannot be appreciated without first articulating a fountain as a node of interest. He/she zooms in on the space to 1:30 scale (that is 1 inch on the table represents 30 feet on the ground) and sketches the fountain plan. A grid becomes visible at this scale. Each block represents 10 feet by 10 feet.

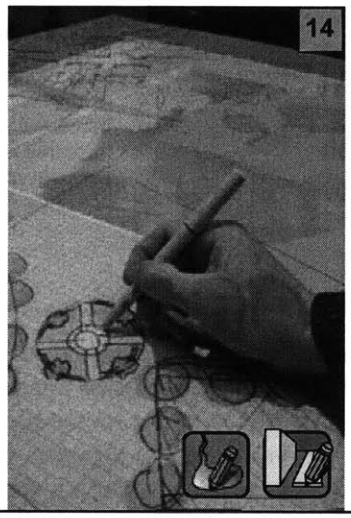






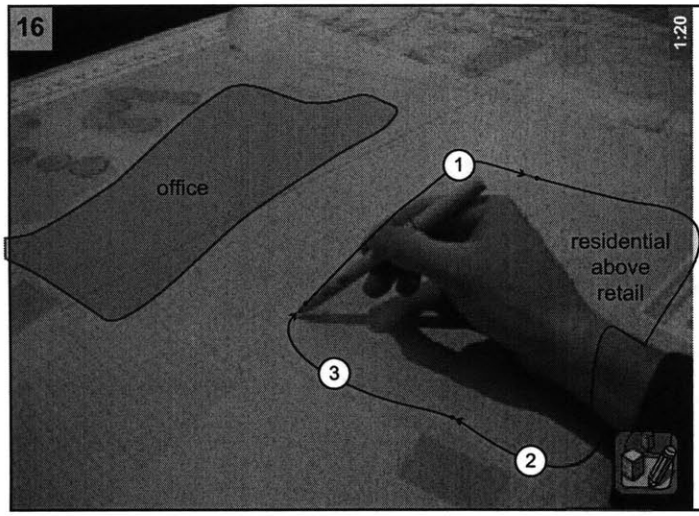






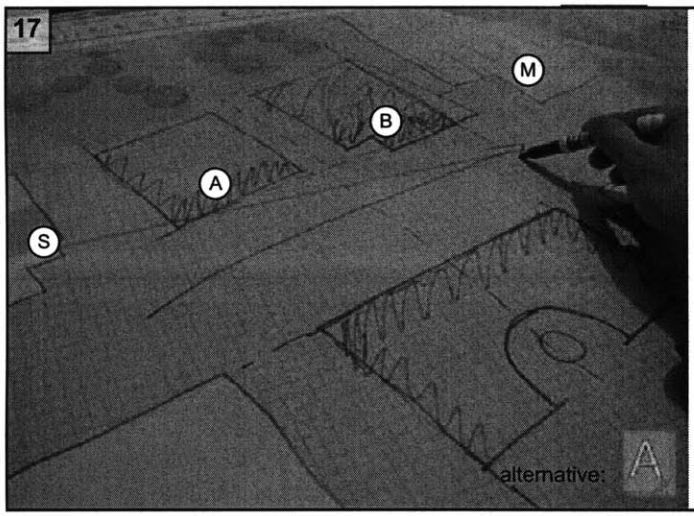
Landscaping tools

Hedges and paths (11 & 14) are constructed along the lines drawn. Note that the Concrete Pen tool may be used to draw both paths and walls (see 15). Trees (13) are placed with a single click at their center in this case. Water and grass (12 & 14) are drawn as a single line or a series of lines that complete a closed shape. The Curiosity Camera will re-center itself on each new object drawn. Notice that the view from this camera (shown above, 15) is also visible to the right of the designer on the table. DASPE is designed to be used with this view projected much larger on a wall to one side. Not only would this avoid obstruction by trace and other objects on the table, but it gives all participants a correctly oriented view regardless of where they stand around the table.



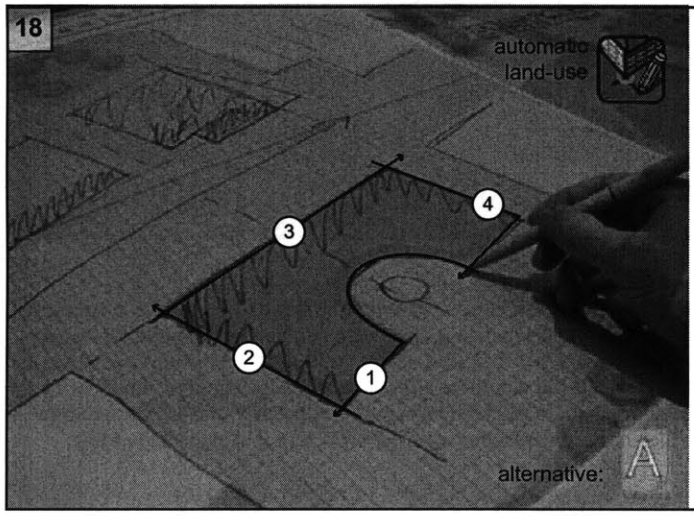
Rough Land-use

The sketcher chooses a land-use option from the palette then draws a series of lines head to tail finishing close to the start of the first line. The tolerance to register completion depends on the scale - here 1:20. After quickly sketching these rough land-use objects, the user chooses to hide them in order to concentrate on the building footprints. The land-use objects will still be recognized later and will determine the use of the building if land-use is set to automatic for the building sketch tool.



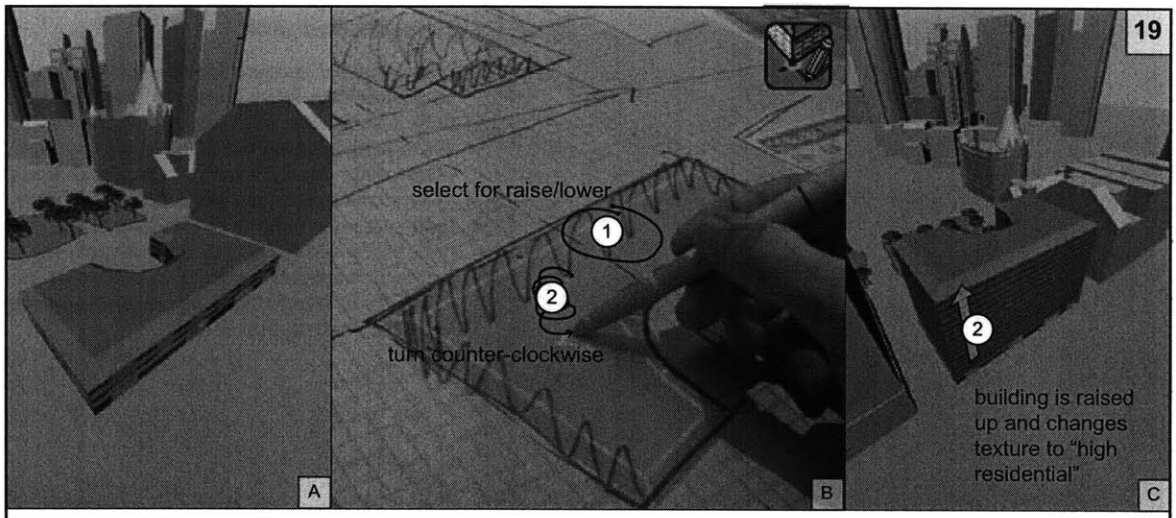
Buildings on State St.

The designer decided to experiment with a number of options for State Street. This first example (alternative A) places the buildings so as to create a consistent street edge on both sides. Building B is stepped so as to take up the difference between the Marriot Long Wharf Hotel (M) and building A which is aligned with the existing building (S) on State St. The designer does not know how this stepped back solution will "feel" at this point but is curious to try it.



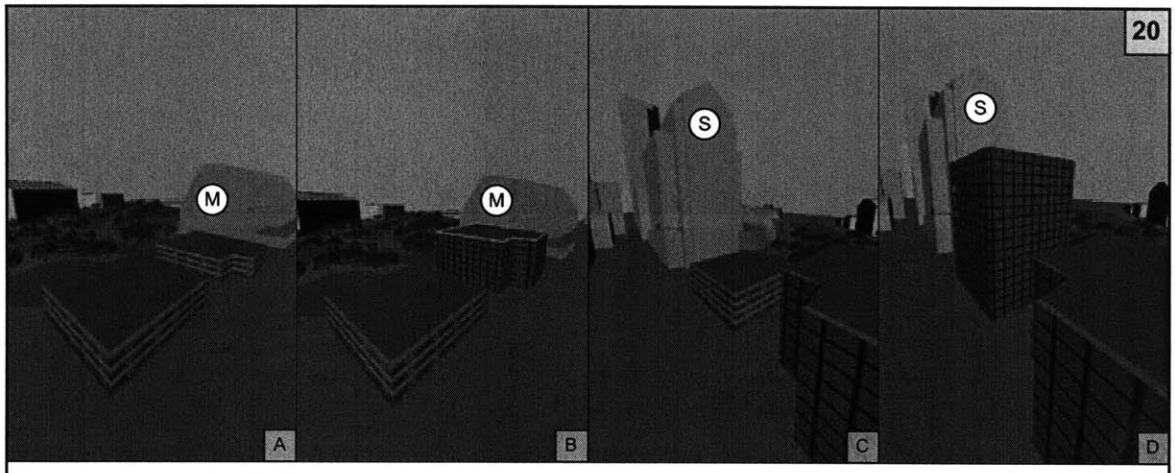
Building Sketch tool

The designer draws a number of lines with the stylus tracing the outline drawn with the marker. Any line deemed straight is straightened (2 & 3). A line found to contain corners will have its segments straightened if they are straight (4) or any curved segments will be smoothed and broken into a number of smaller straight segments (1). Once the shape is found to be complete, a building is automatically popped up from the footprint. Its texture map is determined by the land-use diagram since the "automatic" option is selected.



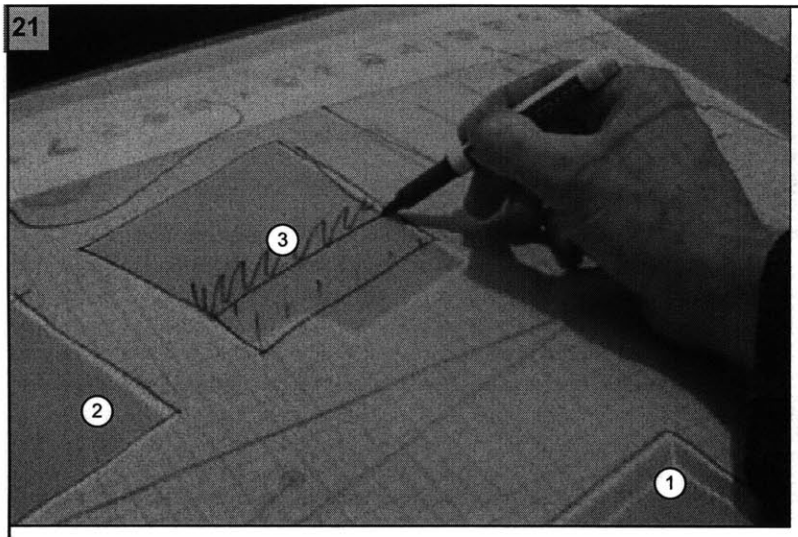
Raise Building

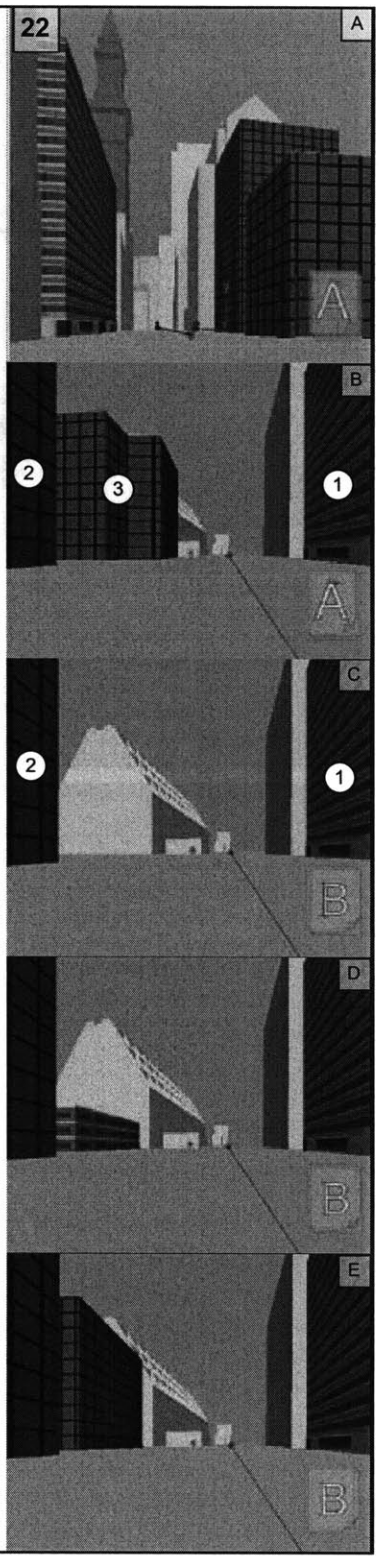
The sketcher draws a circle around the center point of the building to select it to be raised or lowered (note that the circle shown (B1) is purely illustrative as it does not enclose the building center which is obscured by the pen). He/she then turns the pen-tip counter-clockwise to raise the building "out" of the page (C2). Its texture is automatically changed to one more appropriate for a taller building. If the user goes too far he/she may lower the roof again simply by changing to a clockwise motion. The quicker or larger the turns, the faster the raising or lowering. Therefore smaller turns can be used to fine-tune the height.



Context sensitive heights

The designer next draws in the buildings on the other side of State St. (see A & B from image 17). He/she is concerned with creating a consistent street edge along both sides. One of the challenges provided here is that the buildings toward the city (S) are far higher than the Marriot Hotel (M). The designer therefore decides to step down the new intervention to ensure a smooth transition. This is not done mathematically (it would be difficult with such irregular forms) but by constructing a perspective view and raising and lowering them until they feel right. The sketch-view tool is used to set the perspective first towards the Marriot (A & B), then back towards the city (C & D) before raising each building.





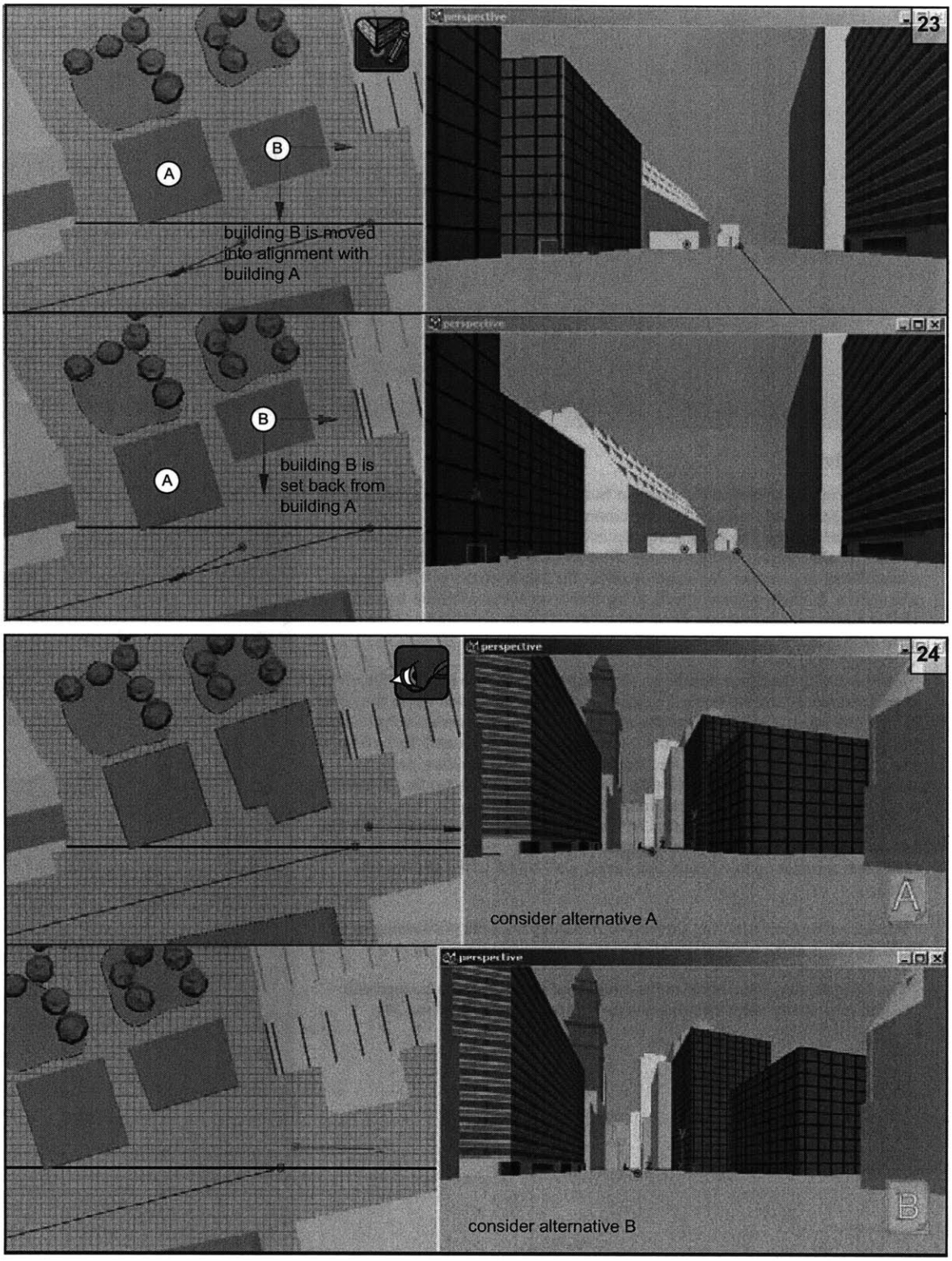
Alternatives

Alternatives are useful to the designer both for keeping his/her options open while designing and also as a tool for communicating decisions. In order to illustrate the reasons for making any decision it is useful to be able to show a number of alternatives to the decision. Furthermore presenting alternatives is a useful way to generate conversation about the implications of each alternative shown. DASPE allows objects to be drawn on either a default layer that is visible across all alternatives, or on any of three alternative layers (A, B or C). Here only A and B are used.

After drawing the office buildings, the designer decides to see how they feel in perspective. Using the leashed camera he/she moves around down the street in both directions. Going toward the city (see view 22A) presents no problems from the designer's perspective: the street edge is consistent and the buildings step up comfortably. However, the designer decides that walking down towards the sea (22B) feels awkward as the building steps out too far. In order to try different options, the designer first sketches some ideas (21) then chooses the current building 3 in alternative A. Once alternative B is selected, building 3 is no longer visible (22C). The designer sketches a new building in its place (22D) and raises it up to a height that seems well suited to the streetscape (22E).

Further experimentation for alternative B is done with the positioning of the building (23 A & B).

The designer may once more move around the street with the leashed camera, and may at any point switch between alternatives A and B (see 24).

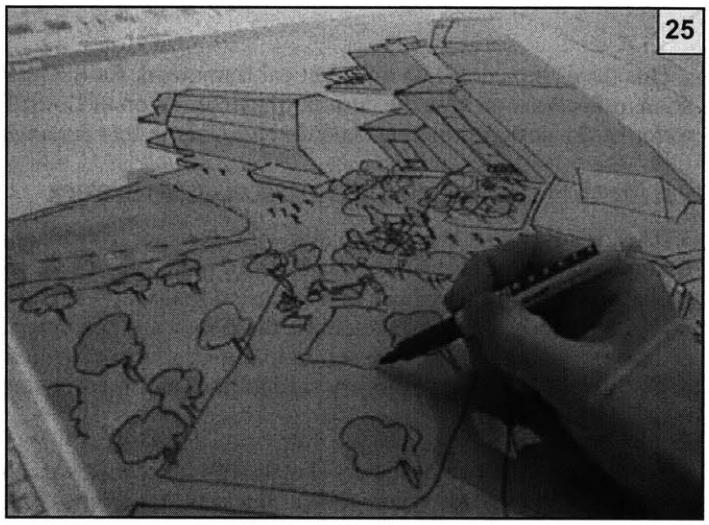

perspective
23
A
B
building B is moved into alignment with building A
perspective
B
A
building B is set back from building A
perspective
24
consider alternative A
A
perspective
consider alternative B
B

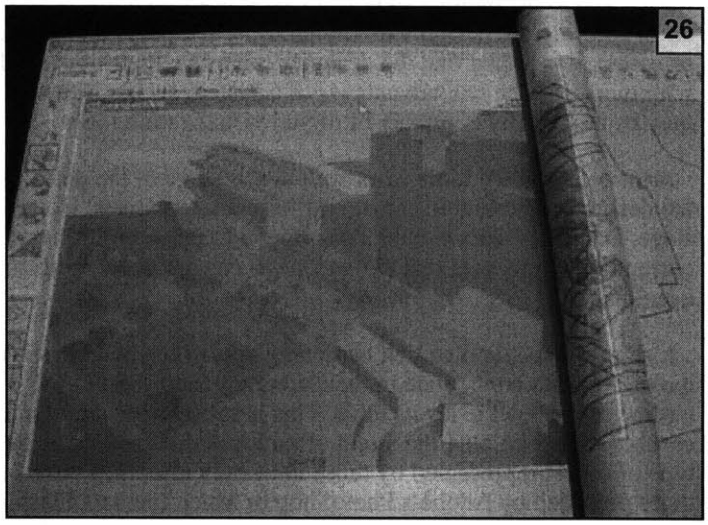
The Next Iteration

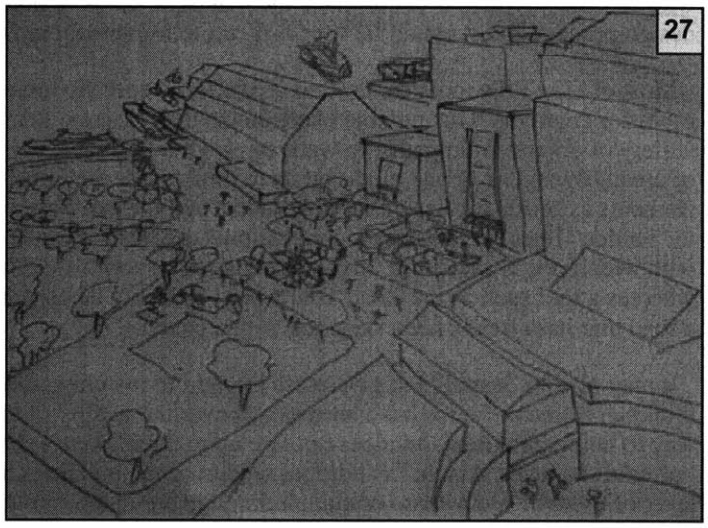
Once the designer has reached a certain level of completion with the computer model made in DASPE, it may be desirable to revert to conventional media in order to keep moving the design forward. While DASPE makes every effort to facilitate natural sketching in the plan view and to expedite the process of model creation, it cannot compete with the freedom of a pen or pencil when it comes to sketching into a drawing. The solution in DASPE is not to emulate the sketching experience with the computer, but to augment it with digital media.

Here the perspective view is projected onto the table and becomes a layer of information upon which to trace one's lines. In a sense it is no different to any other layer of information placed on the table. However the ease with which the designer can generate new perspective views allow him or her to make the most of the digital nature of this layer. But most importantly, this can happen alongside the conventional media with which he or she feels comfortable exploring design ideas.

Computer graphics often fall short of hand-drawn sketches in terms of offering visual intrigue and a sense of vibrancy. Note that even though the sketch in image 27 is little more than a rough trace of the computer generated perspective seen in image 26, it suggests a far more interesting, lively and dynamic space than the view of the computer model does. These qualities may be important in getting one's audience to "buy into" one's design, and sketching over computer perspectives in this way can be easily learned without the need for training in the art of perspective drawing.







Reflections

This thesis includes both a theoretical framework for the interface and a working prototype. Here follow some reflections on how the current implementation of DASPE lives up to its theoretical aspirations in terms of the design principles laid down earlier in this document.

1: Match the level of input to the level of design thinking

DASPE's key goal has been to allow the sketcher to work at a level of detail that is most useful to the designer when thinking through a sketched plan. When sketching with conventional media the designer will at times be drawing lines that represent roads, at others they will be drawing lines for trees or buildings. In the early stages of sketching, these may all look very similar and a great deal of imagination is necessary on the part of the designer to differentiate between lines on the sketch. DASPE does not attempt to distinguish between these very similar lines at this early stage of the design since no computer input takes place.

However, even when designers sketch only for themselves and even at the roughest stages, many designers will look to add color, cleaner lines and sometimes quick shadows in order to better visualize their design and assist their thought process. It is at this stage of design that DASPE looks to assist the designer. In coloring the design with digital tools and providing greater clarity for each element (such as roads or paths), it is hoped that the designer can move their thinking forward in the a similar fashion to coloring with markers. The real advantages of a system such as DASPE is that this stage of "thinking through" may be assisted by a number of digital visualization tools.

Most of DASPE's tools work well at this stage of the design and provide adequate visualization to move the design further and to consider different alternatives. However since the design does not end at this stage, DASPE's tools should also allow it to be taken further at later stages. The building sketch tool is currently the only tool that allows one to change one's mind and alter the tool options after the sketch has been made.

It should also be noted that hardening one's sketched lines is most often done by continually draw them darker as one's confidence in their accuracy increases. DASPE requires that the user instead draw a single outline or center-line for its tools. This is satisfactory, but it may be more natural to develop an algorithm capable of recognizing the intent of multiple sketched lines. In addition it may be desirable to choose the type of object represented by those lines only after they have been drawn. A "paint bucket" tool similar to those provided by Adobe's PhotoShop or Macromedia's Flash applications could be used to "fill" the lines with an object type in order to make them hedges, paths or buildings. This could provide a more natural design process, but it would be very difficult to implement in Maya.

Most of DASPE's tools are used to create fairly specific representations of objects and have limited room for creativity. One notable exception is the Concrete Pen tool which may be used to draw a large variety of objects from paths, to walls to rough buildings. Such a tool allows a the designer a great deal of creativity and he or she might begin to think of its use in new terms - for example as drawing concrete elements as brush-strokes of varying widths and thicknesses. As such it may provide not so much a way to "harden" lines sketched with conventional media as a whole new way of thinking about the medium with which one is sketching. My guess that such a tool would be used more as a sketching tool in itself whereas a tool such as the Road Tool would probably be used more as a tool for representing or hardening a road that has already been sketched with a pencil.

It may also be desirable to give more thought to matching the level of output to the level of design thinking. Some concerns have been raised over the quality of the rendering in DASPE: it goes part of the way to being photoreal, but does not live up to the expected standards for the genre of 3D flythroughs - after all its just a sketch. In addition, rendering the buildings as solid objects with hard lines suggests a level of thought and a sense of completion that are inappropriate for the level of design thinking. An NPR

rendering solution may be the best way to resolve this problem (see Related Work, Texturing and Rendering) however it may prove difficult to implement in Maya.

There is a long way to go before DASPE can really match the level of input to the level of design thinking, but the tools provided thus far show enough to suggest that such a goal may be worth pursuing. However it should also be noted the "level" and manner of design thinking at each stage of the design is also likely to change in response to the new sketch medium. But in all cases we should respect the continuity between mind and hand because in design sketching when "the hand moves, the mind becomes engaged, and vice versa" (Rowe, 1995)

2: Facilitate effective communication and maintain its intent

DASPE has not yet been developed to a level of sophistication that would allow a realistic assessment of the quality of communication it facilitates. However the ability to quickly communicate plan sketches in perspective holds great promise for bridging the gap between those who know the language and syntax of sketch planning and those who do not. The ability to move around spaces with the leashed camera or to consider a node of interest from a number of perspectives with the sketched view provides a useful forum for talking about the experience of the spaces created. Furthermor, the ability to flip between alternatives and make comparisons at any point while using these view tools enables to sketcher to communicate a number of ideas and demonstrate their relationships.

3: Add a dimension without adding complexity

Two main innovations have enabled 3D modelling while maintaining the requirement of plan-only input. These are the raise/lower tool and the Curiosity Camera.

The Curiosity Camera allows the user to have a 3D perspective view with which to watch the results of their sketched strokes made in the plan view. In flying over to the new object of interest, the camera is currently susceptible to becoming obscured or not capturing the whole object of interest in the viewplane. This is overcome by providing tools with which to easily make manual adjustments. It may be preferable however to create a more intelligent script that would always ensure a clear view of the whole object. But the concept seems to be extremely promising and it helps make the link between what one sees in plan and the implications that has for a perspective experience.

The raise/lower tool provides a means of moving objects up or down. It requires input only from above and is consistent from all directions. Using this tool, it is a simple matter to change the height or vertical placing of objects. Feedback is provided by the Curiosity Camera. The tool itself does well to provide a simple way of moving objects up and down and it feels natural in that making bigger, faster turns will move the object more quickly and smaller turns allow for fine-tuning. The raise/lower tool is very effective at overcoming the limitation of 2D input for 3D manipulation and has been essential to the goal of creating an effective plan-view interface.

4: The computer as butler - pervasive computing

In order to determine the effectiveness of DASPE as a "pervasive" augmentation of design space, one would need to test it on a real project in which designers would be asked to sketch plans using both conventional media and the tools provided by DASPE. It is unclear whether users would find it obvious at what stage to make use of digital media or whether it would require a set routine for using each type of media. However such a design environment seems to have great promise in providing the ability to sketch using conventional media then quickly transfer that sketch to the digital realm while at the same time engaging with the necessary practice hardening one's design decisions.

Conclusions

The current implementation of DASPE shows promise as a sketching tool for the early stages of design. However if it is to be used at the heart of the urban planning process, it will need to evolve to the point

where it allows refinement of its sketch objects in such a way as to match the level of the thinking of the user at each point of the process. As it stands DASPE's vizualization tools allow one to see how ones rough sketches have implications at the ground level, but while this will almost certainly influence the decisions made by the designer there is currently no easy way to refine ones model and keep moving it forward.

These are interesting times for the planning profession and it stands to be influenced by a variety of new technologies. DASPE should not be thought of in isolation but as part of a larger framework of computer support systems for planners. However while the computer may continue to support designers and influence their decisions, computer support remains largely separated from design sketching. A system such as DASPE can serve to bring a wide spectrum of digital support literally "to the table" of the designer.

Future Work

Parking Tool

A parking tool is one obvious omission from the DASPE tool set. This tool would most likely be used to draw a parking lot shape in the same manner as grass or water. This would then be cut away from the paving surface using a boolean operation in the same way as roads. Parking spaces with cars could then be drawn as lines within this shape. This would be a fairly simple addition that has not yet been added owing to time constraints.

Spreadsheet Tools

All buildings are made as a group of objects, and the group is given a number of fields that store information on the building. One such piece of information is the area for the building's footprint. This is multiplied by its height in stories to achieve a total square footage (called "sfootage" in figure 36). The user may then input square footage for each land-use and compare this to an "unused" field at the bottom. (Note that the units used here are incorrect as they have not been scaled.)

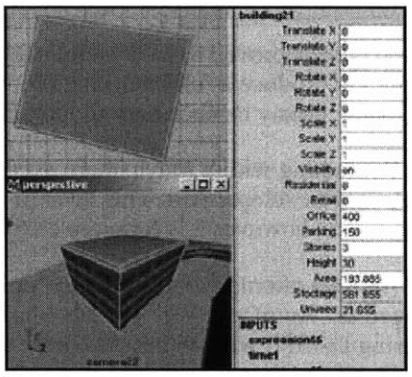


Figure 36: allocating square footage to buildings

As it stands, this provides a useful way of seeing how much "program" one may fit into a certain building for each land-use type, however this would be far more useful if one could view a spreadsheet for all buildings in the scheme and see how each contributes to a *total* for each land-use type. This could be compared to a desired target or "program" for each. Changing figures for square footage in the spreadsheet would automatically change the height of the building to accommodate the new program. This tool has been partially developed, but problems were encountered in getting all the information for more than one building to display in the spreadsheet. Presumably this problem can be overcome, but this feature is not available at the time of writing.

Floor-Area Ratio (FAR) calculations

The user may use the FAR of a land-use object to determine the maximum allowable height of a building. This is determined from the footprint area of the building, the floor area ratio (FAR) and the area of the land-use object. The user would choose the FAR option only if the land use object represented a lot. If not, a preferred default height could be set for all buildings.

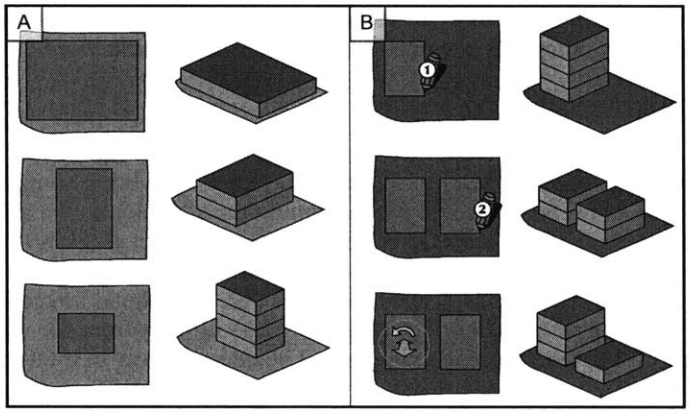


Figure 37: a proposed tool to automate FAR calculations

The examples in figure 37 show how the FAR option would work for a lot drawn with a FAR of 1.0. If a rectangular building is drawn to cover the whole site, the building will have its height set at 1 storey. This is the maximum allowable height for a building covering the whole lot when the FAR is 1.0, however for smaller footprints, the maximum height will increase proportionally (A).

If a second building is added to the land-use object, the first must be lowered to accommodate the new conditions (B). Both will initially be set at the same height, but if one is raised, the other must be lowered

in a seesaw fashion.

Modelling and Simulations

URP has demonstrated that very simple models may be useful in generating design discussions about phenomena that would most likely be otherwise avoided owing to their complexity (Frenchman, personal communication, April 26, 2002). It would be desirable to include modelling capabilities in DASPE that would play the same role. For example simulating the behavior of cars as they would behave under different "levels of service" at intersections allows a better understanding of what the numbers mean. For DASPE it would be hoped that such models would not need to be entirely rewritten but instead just adapted so that their output might be viewed in 3D.

Looking at alternatives: Plug-ins, API and 3D Studio

Maya has proved to be a useful package through the ease of customization it provides and its ability to quickly produce a fully working prototype such as DASPE. However, in considering the future of such a system, it may be necessary to look at other alternatives to MEL.

Remaining within Maya, there exists the option to write some of the more sophisticated tools using Maya's Developer API. This level of programming allows the developer to have far more control over the software, however it is far more complex and difficult to learn.

Another alternative might be to "port" all the code to another application such as 3D Studio (now packaged as "AutoDesk Viz" or "3ds Max") in the form of plug-ins. A feasibility study would be in order, but using DASPE with an initially more user-friendly application such as 3D Studio would have numerous advantages. It would also reduce software costs for implemented DASPE and allow users to use DASPE alongside a number of other plug-ins designed to enhance the interface. Although Maya also uses plug-ins, the user base is not as large as Max's and there are not as many plug-ins available.

Further into the future:

Voice and hand-writing recognition are two technologies that can complement a gestural sketch interface and improve the range of commands that can be used without the need to keep changing the state of the tool. Ultimately the entire DASPE interface could be achieved without a traditional GUI but simply by using voice commands and informative feedback loops for tools. Handwriting is slow and not useful for choosing functions, however it may be useful for annotation or in assisting a voice driven interface that may otherwise become confused between similar words.

Augmented Reality

Virtual reality systems are not conducive to encouraging face to face interaction around a table, however some "augmented reality" systems may be able to bring some of the advantages of true stereo representation to users of DASPE while keeping the advantages of sharing the same physical space. In the augmented reality systems described by Billinghurst et al (2000) "users can see and interact with physical and virtual objects, and also see the other participants." Users wear 3D goggles with built in video cameras to represent the existing scene (see figure 38). The digital image displayed is also augmented with computer-generated information. The software picks up cues from the video captured by the camera in order to know where to place the augmented objects and how to rotate them.

In DASPE this could mean that the 3D model would be added to the participant's 3D view so that it would appear to pop out of the table. Using the table digitizer as input, this could be a quite incredible way of creating a 3D model that appears to have a physical presence, but is scalable, editable and whose construction is easily automated. This may sound like a dream for the distant future but it is actually a highly feasible existing technology. The only current barrier to its implementation is the high cost of 3D goggles that can match the 3D view captured by the video camera to the real view.

But while this technology might not be a dream, its ability to be integrated into Maya is uncertain. I will however take the liberty here of briefly dreaming about some of the possibilities that this new level of visualization would allow. For example the 3D goggles may be used to either augment the environment or to "go fully immersive." One of the chief design challenges is in switching between immersion levels.

Billinghurst et al (2000) demonstrate the ability to fly into a book and walk around in its spaces in 3D. In DASPE one could design it so that when one started dragging the leashed camera one's view would fly down onto the table and follow it through the buildings. Upon release, one could fly back out to ones real position in the room and see the bigger picture and one's human companions once more.

Figure 38: An Augmented Reality Interface for Collaborative Computing

source: www.hitl.washington.edu/magicbook/papers/icme2000.pdf

Conclusion

According to Frenchman (personal communication, April 26, 2002) we have only just begun to scratch the surface of the potential for digital tools to radically change the face of planning as we know it. While many digital innovations such as GIS, CAD and, more recently, 3D visualization systems have already made their presence felt, the area of sketch planning has remained largely aloof.

With DASPE, an attempt has been made to bring many of the advantages of computation to planners while they sketch without requiring them to conform to a predetermined method or medium in order to make use of the software. By augmenting the process of sketch planning rather than replacing it, DASPE allows designers to continue in their own chosen fashion, drawing on the set of computer tools provided only as the need arises. This is done without leaving the sketching table and without changing the input paradigm from pen to mouse. A gestural interface allows the sketcher to capitalize on the greater freedom of input that the pen gives and can facilitate a number of different processes without the need to change tools.

The process of moving from conventional media to a digital medium then becomes an important and integral part of the design methodology. Rather than being a mindless task, hardening one's lines from pencil or marker to digital "ink" corresponds to a stage at which one is looking to harden one's design thinking. Furthermore, the visualization that the digital model allows can be used to liberate one's thinking from the confines of a plan and help generate new ideas.

In addition to its potential as a tool for design, DASPE looks to bridge the ever-present gap between those who understand the implications and "language" of design sketches and those who prefer to understand space in terms of a more familiar representation such as a perspective view. It is hoped that such a "translation" will allow for a better quality of discussion between all participants.

Moving forward, it will be interesting to observe how designers will respond to the new possibilities DASPE provides as well as how it may be used to inform the decision making process when more voices are brought to the table.

References

ALIAS WAVEFRONT, 2001, *Portfolio Wall: the digital chalkboard* [online], Available from: - http: //www.aliaswavefront.com/en/WhatWeDo/studio/see/see_portfoliowall.shtml [Accessed 04/26/2002]

BILLINGHURST, M., POUPYREV, I., KATO, H., AND MAY, R., 2000, *Mixing Realities in Shared Space: An Augmented Reality Interface for Collaborative Computing*, ICME 2000, pp1641-1644, also vailable from [online]: - http://www.hitl.washington.edu/magicbook/papers/icme2000.pdf [Accessed 04/26/2002]

BULMER, D., 2001, *How can computer simulated visualizations of the built environment facilitate better public participation in the planning process?*, Online Planning Journal [online], Available from: - http://www.casa.ucl.ac.uk/planning/articles61/complanning.pdf [Accessed 04/26/2002]

COYNE, R. D., 1999, *Technoromanticism: Digital Narrative, Holism, and the Romance of the Real*, MIT Press, Cambridge, Massachusetts, November 1999

DOWNIE, M AND TOMLINSON, B., 2002, *AlphaWolf* [online] Available from: - http:// badger.www.media.mit.edu/people/badger/alphaWolf/resources.html [Accessed 04/26/2002]

FLINT, A., 2002: *A Sense Of Place: Modernist landscapes just don't get any respect - or protection,* Boston Globe (newspaper) 2/24/2002

FORSBERG, A., LAVIOLA, J. AND ZELEZNIK, R..1998 Ergodesk: *A framework for two and three dimensional interaction at the activedesk*, In Proceedings of the 2nd International Immersive Projection Technology Workshop, Ames, Iowa 1998.

HOPKINS, L. RAMANATHAN, R., AND VARKKI, G., 2002 *Interface for a Planning Workbench* [online], Department of Urban and Regional Planning, University of Illinois at Urbana-Champaign. Available from: - http://www.rehearsal.uiuc.edu/projects/DesignWorkspace [Accessed 04/26/2002]

LEGAKIS, J., DORSEY, J. AND GORTLER, S., 2001, *Feature-Based Cellular Texturing for Architectural Models*, SIGGRAPH'01, pp 309-316, also vailable from [online]: - http://graphics.lcs.mit.edu/pubs/ sg01_fbct.pdf [Accessed 04/26/2002]

MCCRACKEN, E. R., 1997, *Computer Graphics: Ideas and People from America's Universities Fuel a Multi-Billion-Dollar Industry* [online], Computing Research Association. Available from: - http: //www.cs.washington.edu/homes/lazowska/cra/graphics.html [Accessed 04/26/2002]

MCLUHAN, M., 1967, *The Medium is the Massage*, New York: Bantam.

MILGRAM, S., 1967. *The Small-World Problem*, Psychology Today. Volume 1(1);

MILLER, C. G., 1988. *Carscape: a Parking Handbook.* Washington Street Press, Columbus, IN.

NEGROPONTE, N., 1995. *Being Digital.* New York: Vintage Books.

NEMETSCHECK, 2002, *D-Board* [online]. Available from: - http://www.nemetschek.co.uk/products/ dboard.html [Accessed 04/26/2002]

RATTI, C., PIPER, B., AND ISHII, H., 2002, *Illuminating Clay: A 3-D Tangible Interface for Landscape Analysis.* Available at: http://web.mit.edu/ebj/www/lpt/clay.pdf

RICHENS, R. & SCHOFIELD, S, 1995: Interactive Computer Rendering, in *Architectural Research Quarterly*, Volume 1(1) Autumn 1995.

ROWE, P. G., *Design Thinking*. Cambridge, MA: MIT Press, 1987

SCHRAGE, M., 2000, *Serious Play*. Harvard Business School Press.

SHIFFER, M.J., 1996, *Community-Building communications Technologies and Decision Support Systems*. Colloquium on advanced Information Technology, Massachusetts Institute of Technology, Cambridge, MA.

TOLBA, O., DORSEY, J., AND MCMILLAN, L, 1999, *Sketching with Projective 2D Strokes*. In UIST 99: Proceedings of the 12th Annual ACM Symposium on User Interface Software & Technology, CHI Letters, 1(1): 149-157, November 1999.

UNDERKOFFLER, J. AND ISHII, H., 1999 *Urp: A Luminous-Tangible Workbench for Urban Planning and Design*, in Proceedings of CHI '99, May 1999

WELLNER, P., 1992, *Tactile Manipulation on the DigitalDesk*. (video) in CHI '92 Special Video Program, ACM SIG-GRAPH Video Review issue no. 79.

ZELEZNIK, R., HERNDON, K. AND HUGHES, J., 1996, *SKETCH: An Interface for Sketching 3D Scenes*, Proc. SIGGRAPH'96, pp. 163-170

Appendix A: MEL Scripts

The scripts in this thesis are written in the Maya Embedded Language (MEL). The language is basically the same as "C" except that it features a number of extra commands used to communicate with Maya.

The scripts are grouped here according to the different kinds of tools they support. This grouping does not correspond to the grouping of the tools under "Construction," "Landscape" and "Visualize" but rather to the internal patterns shared by a number of these tools.

While it is customary to include all scripts in such an appendix, many scripts are either uninteresting or repetitive and this would be extremely wasteful. There are over 100 scripts for DASPE with a total of just under 12,000 lines (11,932) which would require over 150 pages to print (with an 8 point font as used in this appendix). All these scripts are instead available as files with a ".mel" extension at the following location (the link is to an index page that includes the same categories as below): http://gis.mit.edu/projects/daspe/scripts.

The main reason for this bulk is that MEL commands for making and modifying windows can be very lengthy. Only those scripts specifically discussed in the text or otherwise thought to be unusual or interesting are included here. All scripts are however listed and a brief descriptions of each is given. If interested, the reader is asked to access them on the internet at the aforementioned location.

Contents

Buildings/Pletching Tool

The Building Sketch Tool

pletching_tool.mel

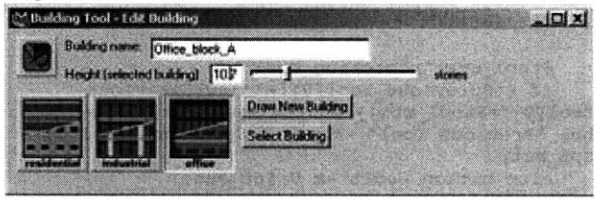
This script is used to create the windows used for the pletching tool (also known as the Building Sketch tool) and to create the tool "contexts" for this tool. A "context" is just a new tool state in Maya. The main "context" in this case is the "pletchContext" create as a draggerContext at the end of this script. The Building Sketch tool may have both a creation or "sketch" state and an edit state so different tool options will be created by this tool depending on the $bt_state variable.

One may notice that the tool window is not being created in this script, but only edited (using "window -edit"). This is because the same window is used for all the DASPE tool options. Each time a new tool is selected, the window is cleared (using the custom written "clear_content") and new options are specified.

```
global proc pletching_tool()
{
 // general variable declarations
 global int $line_count;
 global int $cept_count;
 global int $bt_state;
 global string $LU_choice; if ($LU_choice=="") $LU_choice="residential";
 global float $cbldg_height;if (($cbldg_height<1)||($cbldg_height>100)) $cbldg_height=3;

 // If the "building chooser" tool doesn't exist, make it now. This tool lets one pick buildings
for modification and is available from the "select building" button.

 if (! `scriptCtx -exists BldgChooser`)
 {
  scriptCtx
   -title "Edit Building"
   -totalSelectionSets 1
   -fcs "select -r $Selection1;$selected_bldg=`getAttr ($Selection1[0]+\".number\")`;nameField
-edit -object $bldg_group[$selected_bldg] $field1;$bt_state=2;pletching_tool;"
   -cumulativeLists false -expandSelectionList false
   -setNoSelectionPrompt "Choose Building to Edit"
   -setAutoToggleSelection true -setSelectionCount 1
   -setAutoComplete true
   -exitUponCompletion true
  BldgChooser;
 }

 // Make the tool menu depending on the state of the tool
 tool_window("none");

 modelEditor -e -wos 0 modelPanel1;

 global string $tool_image;
 global string $toolwindow;
 global string $tool_tips;
 global string $toolform;
 global string $ctrl_slider1;
 global string $slider1;
 global string $pt_but;
 global string $check_pic1;
 global string $check_pic2;
 global string $check_pic3;
 global string $check_pic4;
 global string $check_pic5;
 global string $check_pic6;
 global string $t1;
 global string $b1;
 global string $b2;
 global string $field1;
 global string $previous_tool[];
```

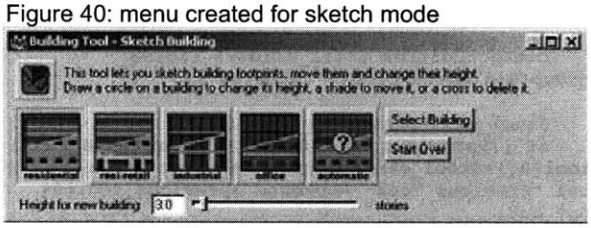
```
  string $tool_ctx;

  clear_content;

  symbolButton -edit -image "D:/DASPE/icons/small/pletch-s.bmp" -c "$bt_state=1;pletching_tool"
$tool_image;

  if ($bt_state>=2) //the menu used when a building is made and/or selected
  {
   global string $bldg_group[];global int $selected_bldg;
   global float $building_x[];global float $building_z[];
   construction_cam($building_x[$selected_bldg],0,$building_z[$selected_bldg]);
   select -r $bldg_group[$selected_bldg];
   window -edit -title "Building Tool - Edit Building" $toolwindow;
   text -edit -m 1 -label "Building name: " $t1;
   nameField -edit -m 1 -w 200 -object $bldg_group[$selected_bldg] -cc "$bldg_group[$selected_
bldg]=`nameField -query -object $field1`" $field1;
   symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/resi_on.bmp" -ofi "D:/DASPE/icons/resi.bmp"
-onc "gc_switch(1,3,1);setAttr ($body_of[$selected_bldg]+\".landUse\") -type \"string\" \
"residential\";change_ht($selected_bldg,1);" $check_pic1;
   symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/indust_on.bmp" -ofi "D:/DASPE/icons/indust.bmp"
-onc "gc_switch(1,3,2);setAttr ($body_of[$selected_bldg]+\".landUse\") -type \"string\" \
"industrial\";change_ht($selected_bldg,1);" $check_pic2;
   symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/office_on.bmp" -ofi "D:/DASPE/icons/office.bmp"
-onc "gc_switch(1,3,3);setAttr ($body_of[$selected_bldg]+\".landUse\") -type \"string\" \"office\
";change_ht($selected_bldg,1);" $check_pic3;

   button -edit -m 1 -label "Draw New Building" -c "$bt_state=1;pletching_tool" $b1;
   button -edit -m 1 -label "Select Building" -c "changeSelectMode -hierarchical;select -
cl;setTool(1,\"BldgChooser\");" $b2;
   floatSliderGrp -edit -m 1
    -cc "setAttr ($bldg_group[$selected_bldg]+\".stories\") `floatSliderGrp -query -value $ctrl_
slider1`"
    -cw 1 133 -cw 2 33
    -label "Height (selected building)"
    -extraLabel "stories"
    -minValue 1 -maxValue 50
    -fieldMinValue 1 -fieldMaxValue 100
    -value $cbldg_height
   $ctrl_slider1;

   connectControl $ctrl_slider1 ($bldg_
group[$selected_bldg]+".stories");

   $tool_ctx="pletching_tool";
   if (($previous_tool[0]!="")&&($previous_
tool[0]!=$tool_ctx)) button -edit -m 1 -la-
bel "Previous Tool" -command $previous_tool[0]
$pt_but;
   else button -edit -m 0 $pt_but;

   formLayout -edit

   -attachForm $t1 "top" 7
   -attachControl $t1 "left" 7 $tool_image
   -attachForm $field1 "top" 7
   -attachControl $field1 "left" 2 $t1

   -attachControl $ctrl_slider1 "top" 7 $t1
   -attachControl $ctrl_slider1 "left" 3 $tool_image

   -attachForm $check_pic1 "left" 7
   -attachControl $check_pic1 "top" 7 $ctrl_slider1
   -attachControl $check_pic2 "left" 3 $check_pic1
   -attachControl $check_pic2 "top" 7 $ctrl_slider1
   -attachControl $check_pic3 "left" 3 $check_pic2
   -attachControl $check_pic3 "top" 7 $ctrl_slider1

   -attachControl $b1 "left" 7 $check_pic3
   -attachControl $b1 "top" 7 $ctrl_slider1
   -attachControl $b2 "left" 7 $check_pic3
   -attachControl $b2 "top" 5 $b1
```

Figure 39: menu created for edit mode



```
    $toolform;
    global string $body_of[];
    string $current_use;
    $current_use=`getAttr ($body_of[$selected_bldg]+".lu")`;
    if ($current_use=="residential") {symbolCheckBox -edit -value 1 -en 0 $check_pic1; gc_
switch(1,3,1);} else symbolCheckBox -edit -value 0 -en 1 $check_pic1;
    if ($current_use=="industrial")  {symbolCheckBox -edit -value 1 -en 0 $check_pic2; gc_
switch(1,3,2);} else symbolCheckBox -edit -value 0 -en 1 $check_pic2;
    if ($current_use=="office")      {symbolCheckBox -edit -value 1 -en 0 $check_pic3; gc_
switch(1,3,3);} else symbolCheckBox -edit -value 0 -en 1 $check_pic3;

   }
   else //the menu used when in sketching mode
   {
    window -edit -title "Building Tool - Sketch Building" $toolwindow;
    text -edit -m 1 -label "This tool lets you sketch building footprints, move them and change
their height.\nDraw a circle on a building to change its height, a shade to move it, or a cross
to delete it." $tool_tips;
    symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/resi_on.bmp" -ofi "D:/DASPE/icons/resi.bmp"
-onc "gc_switch(1,5,1);$LU_choice=\"residential\";" $check_pic1;
    symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/resi-ret_on.bmp" -ofi "D:/DASPE/icons/resi-
ret.bmp" -onc "gc_switch(1,5,2);$LU_choice=\"resi_retail\";" $check_pic2;
    symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/indust_on.bmp" -ofi "D:/DASPE/icons/indust.bmp"
-onc "gc_switch(1,5,3);$LU_choice=\"industrial\";" $check_pic3;
    symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/office_on.bmp" -ofi "D:/DASPE/icons/office.bmp"
-onc "gc_switch(1,5,4);$LU_choice=\"office\";" $check_pic4;
    symbolCheckBox -edit -m 1 -dni "D:/DASPE/icons/auto_on.bmp" -ofi "D:/DASPE/icons/auto.bmp"
-onc "gc_switch(1,5,5);$LU_choice=\"automatic\";" $check_pic5;

    button -edit -m 1 -label "Select Building" -c "changeSelectMode -hierarchical;select -
cl;setTool(1,\"BldgChooser\");" $b1;
    button -edit -m 1 -label "Start Over" -c "delete_lines;$line_count=0" $b2;
    floatSliderGrp -edit -m 1
     -cc "$cbldg_height=`floatSliderGrp -query -value $slider1`"
     -label "Height for new building"
     -cw 1 173 -cw 2 33
     -extraLabel "stories"
     -minValue 1 -maxValue 50
     -fieldMinValue 1 -fieldMaxValue 100
     -value $cbldg_height
    $slider1;

    $tool_ctx="pletching_tool";
    if (($previous_tool[0]!="")&&($previous_
tool[0]!=$tool_ctx)) button -edit -m 1 -la-
bel "Previous Tool" -command $previous_tool[0]
$pt_but;
    else button -edit -m 0 $pt_but;

    formLayout -edit

    -attachForm $check_pic1 "left" 7
    -attachControl $check_pic1 "top" 7 $tool_image
    -attachControl $check_pic2 "left" 3 $check_pic1
    -attachControl $check_pic2 "top" 7 $tool_image
    -attachControl $check_pic3 "left" 3 $check_pic2
    -attachControl $check_pic3 "top" 7 $tool_image
    -attachControl $check_pic4 "left" 3 $check_pic3
    -attachControl $check_pic4 "top" 7 $tool_image
    -attachControl $check_pic5 "left" 3 $check_pic4
    -attachControl $check_pic5 "top" 7 $tool_image

    -attachControl $b1 "left" 7 $check_pic5
    -attachControl $b1 "top" 7 $tool_image
    -attachControl $b2 "left" 7 $check_pic5
    -attachControl $b2 "top" 5 $b1
    -attachControl $slider1 "top" 7 $check_pic3
    -attachForm $slider1 "left" -45

    $toolform;
```

Figure 40: menu created for sketch mode



```
   if ($LU_choice=="residential") {symbolCheckBox -edit -value 1 -en 0 $check_pic1; gc_
switch(1,5,1);} else symbolCheckBox -edit -value 0 -en 1 $check_pic1;
   if ($LU_choice=="resi-retail") {symbolCheckBox -edit -value 1 -en 0 $check_pic2; gc_
switch(1,5,2);} else symbolCheckBox -edit -value 0 -en 1 $check_pic2;
   if ($LU_choice=="industrial")  {symbolCheckBox -edit -value 1 -en 0 $check_pic3; gc_
switch(1,5,3);} else symbolCheckBox -edit -value 0 -en 1 $check_pic3;
   if ($LU_choice=="office")      {symbolCheckBox -edit -value 1 -en 0 $check_pic4; gc_
switch(1,5,4);} else symbolCheckBox -edit -value 0 -en 1 $check_pic4;
   if ($LU_choice=="automatic")   {symbolCheckBox -edit -value 1 -en 0 $check_pic5; gc_
switch(1,5,5);} else symbolCheckBox -edit -value 0 -en 1 $check_pic5;
  }

 // create the pletchContext tool that is used to sketch, move, delete and raise/lower buildings
 if (`draggerContext -exists pletchContext`==0)
 {
   draggerContext
   -il "D:/DASPE/icons/small/pletch.bmp"
   -pressCommand "pletch_down"
   -dragCommand "plot_points"
   -releaseCommand "check_pletch"
   -cursor "crossHair"
   -sp "world"
   pletchContext;
 }
 select -cl;
 setTool(1,"pletchContext");

 }//endp//
```

You will notice how long this script is even though it is just used to create a couple of window layouts and tool contexts. Most other scripts (apart from similar tool declarations) are shorter, use far fewer MEL-specific commands, and are sometimes indistinguishable from "C".

pletch_down.mel

The "pressCommand" specified above for pletchContext is "pletch_down." The pressCommand is called whenever the mouse button is clicked (or stylus tip is depressed) while the current tool "context" is set to pletchContext.

plot_points.mel

The "dragCommand" for the Pletch tool. This script is called each time the pointer is determined to have moved during a mouse drag. Plot_points is used to store information about the line while it is being drawn. It also plots a line that follows the cursor so the user can see what they're drawing.

check_pletch.mel

The "realeaseCommand" for pletchContext is "check_pletch." This procedure (shown below) is called whenever the mouse-button or stylus is released. The check_pletch procedure is at the heart of the pletching_tool's gestural functionality and corresponds to the flow-diagram on page 41. Scripts specifically associated with this procedure are shown in bold type and are listed after it. Comments showing the logic of the flow-diagram are also shown in bold type.

```
global proc check_pletch()
 {
 //declare general variables
 global float $xpos[]; global float $zpos[];
 global int $count_g;
 global float $cum_dist[]; //cumulative distance//
 global string $name_of_dline[];
 global int $line_count;
 string $temp_cons;
 int $count;
 string $ext_num;
 global float $ortho;
 float $line_dist;
 float $temp_z;float $temp_x;
 float $increment=90;
```

```
   global int $ext_number;
   global int $has_corner;

   if ($count_g>0) //ie has a line actually been drawn...
   {
   //STRAIGHT OR NOT STRAIGHT??//
   if ($cum_dist[$count_g-1]>(distcalc($xpos[$count_g-1],$zpos[$count_g-1],$xpos[0],$zpos[0]))/
0.85)
    {

     //NOT STRAIGHT//
     delete_plotted;
     //DOES IT HAVE CORNERS?//
     string $temp_curve=find_corners();int $use_temp=1;
     if ($has_corner!=1)
     {
      if(check_circ(0.7,1.2))//IS IT A CIRCLE?//
      {
      //YES - CIRCLE//
       global float $circ_center_x;
       global float $circ_center_z;
       global float $circle_rad;
       //IS THE CIRCLE ON A BUILDING//
       if(on_building($circ_center_x,$circ_center_z,$circle_rad))
       {
          //YES - LIES ON BUILDING//
          $use_temp=0;delete $temp_curve;
          global string $roof_of[];
          int $numma=on_building($circ_center_x,$circ_center_z,$circle_rad);
          global int $bt_state;
          $bt_state=3; pletching_tool;
          raise_lower($roof_of[$numma]);
          //select the roof of the building the circle is on for raise/lower//
       }else{
          //NO - NOT ON BUILDING
          //make circular building//
          if (check_circ(0.3,0.7)) //more strict if not on building...
          {
           circular_building($circ_center_x,$circ_center_z,$circle_rad);
           $use_temp=0;
          }
       }//endif on building
      }else{
      //NO - NOT CIRCLE//
      //IS IT A SHADE?//
       if (check_shade()) {$use_temp=0;delete $temp_curve;}
       else $line_count=0;
       //NO - NOT A SHADE//
      }//end if it is a circle//
     }//end if it has no corners
     if ($use_temp)
     {
       $line_count++;
       $name_of_dline[$line_count]=$temp_curve;
       check_for_completion($name_of_dline[$line_count],$line_count,"none");
     }
    }else{

    //IF LINE IS STRAIGHT//
     delete_plotted;

     //first check to see if its an extension of another line.
     string $extension=is_it_an_extension();//returns name of line just extended...//

     if ($extension=="none")
     {
      $line_count=$line_count+1;

      int $cross=0;

      //IS IT A CROSS//
      if ($line_count==2) $cross=check_cross();
```

```
     //NOT A CROSS (If it is a cross, a building will be deleted in the check_cross procedure.)
     if (!$cross)
     {
      //ORTHO SNAP//
      float $lortho;$lortho=$ortho;
      if (($ortho>90)&&($ortho<=180)) $lortho=$ortho-90;
      if (($ortho>180)&&($ortho<=270)) $lortho=$ortho-180;
      if (($ortho>270)&&($ortho<=360)) $lortho=$ortho-270;
      if ($ortho!=-100)//ie if ortho is on//
      {
      for($count=-90;$count<=90;$count=$count+$increment)
       {
       float $abs_helper=($count+$lortho-(angle($xpos[$count_g-1],$zpos[$count_g-
1],$xpos[0],$zpos[0])));
       if (`abs $abs_helper`<18)
        {
         $line_dist=distcalc($xpos[0],$zpos[0],$xpos[$count_g-1],$zpos[$count_g-1]);//
         $temp_z=($zpos[0]+$line_dist*(sin(deg_to_rad($count+$lortho))));
         $temp_x=($xpos[0]+$line_dist*(cos(deg_to_rad($count+$lortho))));

        if (distcalc($xpos[$count_g-1],$zpos[$count_g-1],$temp_x,$temp_z)<$line_dist)//ie the
distance from the old end point to the new one is not bigger than the whole line...
         {
          $zpos[$count_g-1]=$temp_z;
          $xpos[$count_g-1]=$temp_x;
         }else{
          $zpos[$count_g-1]=($zpos[0]-$line_dist*(sin(deg_to_rad($count+$lortho))));
          $xpos[$count_g-1]=($xpos[0]-$line_dist*(cos(deg_to_rad($count+$lortho))));
         }
        }
       }//endfor
      }//endif ortho is on//

      $name_of_dline[$line_count]=`curve -d 1 -p $xpos[$count_g-1] 0 $zpos[$count_g-1] -p
$xpos[0] 0 $zpos[0]`;
      namespace -set ":";
      select -cl;
      //CHECK TO SEE IF THAT COMPLETES A BUILDING FOOTPRINT//
      check_for_completion($name_of_dline[$line_count],$line_count,"none");

     }//endif cross
    }//endif ext

    if ($extension!="none")
    {
     $temp_cons=`curve -d 1 -p $xpos[$count_g-1] 0 $zpos[$count_g-1] -p $xpos[0] 0 $zpos[0]`;
     check_for_completion($temp_cons,$ext_number,$extension);
     delete $temp_cons;
    }//endif extension=none//

   }//endif not straight//
  }
  global string $top_tool;if ($top_tool=="pletchContext") {select -cl;}
 }//endp//
```

The following scripts are listed in the order in which they appear in the above procedure:

find_corners.mel

This script looks at the angles between lines drawn from one drag-point to the next. It is used to find corners in a line drawn with the pletch tool. It will also straighten out lines between corners if deemed straight or will otherwise create these segments as curves. Some corners will not occur neatly around a single drag point and the drag points comprising the corner may instead lie in a number of configurations. A, B and C are used to look at each possibility.

```
global proc string find_corners()

{
 global float $xpos[]; global float $zpos[];
```

```
   global int $count_g;
   global float $cum_dist[];
   global int $line_count;
   float $A_angle1;
   float $A_angle2;
   float $A_angle_dif;
   float $B_angle1;
   float $B_angle2;
   float $B_angle_dif;
   float $C_angle1;
   float $C_angle2;
   float $C_angle_dif;
   string $completed;
   global int $has_corner;$has_corner=0;
   float $acute_tol;$acute_tol=15;
   float $obtuse_tol;$obtuse_tol=35;
   float $corner_x[];
   float $corner_z[];
   int $num_corners;$num_corners=0;
   int $corner_pos[];
   float $total_angle;$total_angle=0;
   float $x;float $z;
   float $last_corn[];//0=x 2=z;
   float $zoom=(`camera -q -ow top`)/46.25;//5=1:50ft 2=1:20ft etc

   //Find the corners
   /*
   move down the line getting the angle between every 2 points
   if the difference in angle between any 2 angles is big (say 75-105)
   then mark it as a corner.
   */

   //make the namespace so that all temporary construction may be deleted at the end.
   namespace -add "FIND_CORNRS";
   namespace -set "FIND_CORNRS";

   //A, B and C all look at different possible corner configurations. If no corner is found for A,
it will check configuration B and if there is still no corner it will use C.
   for ($loop=5;$loop<$count_g;$loop++)
   {
    $A_angle1=CASTangle($xpos[$loop-5],$zpos[$loop-5],$xpos[$loop-3],$zpos[$loop-3]);
    $A_angle2=CASTangle($xpos[$loop-2],$zpos[$loop-2],$xpos[$loop],$zpos[$loop]);
    $A_angle_dif=$A_angle1-$A_angle2;
    $total_angle=$total_angle+abs($A_angle_dif);

    $B_angle1=CASTangle($xpos[$loop-4],$zpos[$loop-4],$xpos[$loop-2],$zpos[$loop-2]);
    $B_angle2=CASTangle($xpos[$loop-2],$zpos[$loop-2],$xpos[$loop],$zpos[$loop]);
    $B_angle_dif=$B_angle1-$B_angle2;
    $total_angle=$total_angle+abs($B_angle_dif);
    if ($loop>=7)
    {
     $C_angle1=CASTangle($xpos[$loop-7],$zpos[$loop-7],$xpos[$loop-4],$zpos[$loop-4]);
     $C_angle2=CASTangle($xpos[$loop-3],$zpos[$loop-3],$xpos[$loop],$zpos[$loop]);
     $C_angle_dif=$C_angle1-$C_angle2;
    }
    if (((abs($A_angle_dif)>(90-$acute_tol))&&(abs($A_angle_dif)<(90+$obtuse_tol)))||((abs($A_an-
gle_dif)>(270-$obtuse_tol))&&(abs($A_angle_dif)<(270+$acute_tol))))
    {
     $x=($xpos[$loop-2]+$xpos[$loop-3])/2;
     $z=($zpos[$loop-2]+$zpos[$loop-3])/2;
     if (distcalc($x,$z,$corner_x[$num_corners],$corner_z[$num_corners])>0.2*$zoom)/*ARBITRARY*/
     {
      circle -nr 0 1 0 -c $x 0 $z -r 0.2;
      $has_corner=1;
      $num_corners++;
      $corner_x[$num_corners]=$x;
      $corner_z[$num_corners]=$z;
      $corner_pos[$num_corners]=$loop-2;
      $total_angle=$total_angle-abs($A_angle_dif);//eliminate outliers...
     }
    }
    else
```

```
   {
    if (((abs($B_angle_dif)>(90-$acute_tol))&&(abs($B_angle_dif)<(90+$obtuse_tol)))||((abs($B_an-
gle_dif)>(270-$obtuse_tol))&&(abs($B_angle_dif)<(270+$acute_tol))))
    {
     $x=$xpos[$loop-2];
     $z=$zpos[$loop-2];
     if (distcalc($x,$z,$corner_x[$num_corners],$corner_z[$num_corners])>0.5*$zoom)/*ARBITRARY*/
     {
      circle -nr 0 1 0 -c $x 0 $z -r 0.2;
      $has_corner=1;
      $num_corners++;
      $corner_x[$num_corners]=$x;
      $corner_z[$num_corners]=$z;
      $corner_pos[$num_corners]=$loop-2;
      $total_angle=$total_angle-abs($B_angle_dif);//eliminate outliers...
     }
    }
    else if (((abs($C_angle_dif)>(90-$acute_tol))&&(abs($C_angle_dif)<(90+$obtuse_
tol)))||((abs($C_angle_dif)>(270-$obtuse_tol))&&(abs($C_angle_dif)<(270+$acute_tol))))
    {
     $x=($xpos[$loop-4]+$xpos[$loop-3])/2;
     $z=($zpos[$loop-4]+$zpos[$loop-3])/2;
     if (distcalc($x,$z,$corner_x[$num_corners],$corner_z[$num_corners])>0.5*$zoom)/*ARBITRARY*/
     {
      circle -nr 0 1 0 -c $x 0 $z -r 0.2;
      $has_corner=1;
      $num_corners++;
      $corner_x[$num_corners]=$x;
      $corner_z[$num_corners]=$z;
      $corner_pos[$num_corners]=$loop-3;
     }
    }
   }
  }//endfor
  string $curve_string;

  //Make the straight lines or curves between the corners.

  float $cumulative;string $curvy;float $straight;string $straight_line;
  $curve_string="curve -d 1 ";//start

  if ($num_corners>0)
  {
   $corner_pos[0]=0;$corner_x[0]=$xpos[0];$corner_z[0]=$zpos[0];//start
   $corner_pos[$num_corners+1]=($count_g-2);//must be -2 because there is an n+1...
   $corner_x[$num_corners+1]=$xpos[$count_g-2];$corner_z[$num_corners+1]=$zpos[$count_g-2];//end

   //make lines for corners and end
   for ($corna=0;$corna<=$num_corners+1;$corna++)
   {
    $cumulative=0;$curvy="";
    if ($corna<=$num_corners)
    {
     for ($n=$corner_pos[$corna];$n<=$corner_pos[$corna+1];$n++)
     {
      $cumulative=$cumulative+distcalc($xpos[$n],$zpos[$n],$xpos[$n+1],$zpos[$n+1]);
      $curvy=$curvy+" -p "+$xpos[$n]+" 0 "+$zpos[$n];
     }//end for
     $straight=distcalc($corner_x[$corna],$corner_z[$corna],$corner_x[$corna+1],$corner_
z[$corna+1]);
     $straight_line=(" -p "+$corner_x[$corna]+" 0 "+$corner_z[$corna]);
    }//endif

    if ($corna>$num_corners)
    {
     $straight=$cumulative;//this just tricks the next check into using the string below...
     $straight_line=(" -p "+$xpos[$count_g-1]+" 0 "+$zpos[$count_g-1]);//end
    }

    if ($cumulative>1.1*$straight)//arbitrary
    {
     $curvy=smooth_curve($curvy);
```

```
     $curve_string=$curve_string+$curvy;
    }else{
     $curve_string=$curve_string+$straight_line;
    }
   }//endfor corna
  }else //if there are no corners - note there is no need to determine whether the whole thing is
straight - this has been done in check_pletch...
  {
   for ($n=0;$n<$count_g;$n++)
    {
     $curvy=$curvy+" -p "+$xpos[$n]+" 0 "+$zpos[$n];
    }//end for
    $curvy=smooth_curve($curvy);
    $curve_string=$curve_string+$curvy;
  }//end if corners

  //delete the temporary lines
  catch(`delete "FIND_CORNRS:*"`);
  namespace -set ":";
  catch(`namespace -rm "FIND_CORNRS"`);
  string $completed=`eval($curve_string)`;

  //the average of the angle between all lines that don't make a corner allows the script to de-
cide whether a line may be a circle even if it appears to have some corners.

  if ($avg>50) $has_corner=0;//this allows circle to be created...
  return $completed;
 }//endproc
```

check_circ.mel

In order to determine whether or not a given line is a circle, this script divides the line into 7 points then looks at the sections between 1-3, 2-4, 3-5, 4-6, and 5-7. (See figure 1).

The following scripts are used only in check_circ and were created for convenience only:

averageof.mel -used to get the average length of a number of lines.

longestof.mel - used to get the longest length for a number of lines.

shortestof.mel - used to get the shortest length of a number of lines.

on_building.mel

This script scans through all building groups and looks to see if their center position is near the position specified, then returns the number of the building. It is used to determine if a circle, shade or cross lies on a building.

check_shade.mel

Checks to see if the user has "shaded" a building - basically just looks to see whether the points are drawn in a dense area or not.

is_it_an_extension.mel

Checks a line to see if it is an extension of another line.

check_cross.mel

Checks for a cross shape as determined by where two straight intersecting lines are found to cross each other. Deletes a building found to be inside the cross shape.

check_for_completion.mel

This script looks at all lines drawn and uses a planar NURBS surface to determine whether a building footprint has been completed. The script then converts the NURBS surface to a single polygonal facet that gets extruded to form the building.

delete_lines.mel - deletes temporary lines used to make buildings and in other "pletch" commands.

Scripts used to make buildings

popupBldg.mel
Gets land-use properties from land-use objects if they exist and uses "straightBldg" to pop up a building based on what it finds.

straightBldg.mel
Makes buildings in three parts based on a given footprint. The middle section is made from the footprint (as is) while the roof and lower section (if used) are made from a slightly large offset of the building footprint. A number of attributes also get connected to each other in this script so that the building height will change the roof height and vice versa.

makeRoof.mel
Used by "straightBldg" to make the roof by bevelling an extruded polygon.

change_ht.mel
Used by a scriptJob to continually check the height for a building - calls for a texture change (using change_texture) according to the height it finds.

change_texture.mel
change_ht and straightBldg call this to change the texture of a building according to its land-use.

circular_building.mel
For now this script makes a hut shaped building - it may be adapted in future to make a fountain, statue or other central, circular "node" of interest instead.

Raise/Lower Tool

raise_lower.mel
Creates the tool used to screw (or raise/lower) objects up and down.

screw_down.mel - the pressCommand for the raise/lower tool.

screw_points.mel
The dragCommand for the raise/lower tool. This script looks at each point plotted and considers the angle between it and the previous point. It watches the arc tangent of these angles for positive and negative "switches" which it analyzes in order to determine whether rotation is clockwise or counter-clockwise. The script will then change the height (or y position) of the specified object according to the size of the turns being made and their speed.

```
global proc screw_points()
{
   global float $xpos[]; global float $zpos[];
   global int $count_g;
   global float $prev_angle;
   float $angle;
   global float $running_total;
   float $dragPosition[] = `draggerContext -query -dragPoint screwContext`;
   global float $neg;global float $pos;
   global string $isneg;
   global float $last_posx;global float $last_posz;
   global string $wise;
   global string $time_before;
   float $numdivs;float $factor;
   global float $target;
   global int $drag_counter;
   global int $num_divs;
   global int $surity;
   global string $object;

   float $xdif;float $zdif;
   string $last_pos_is;
```

```
    $xpos[$count_g]=$dragPosition[0]; $zpos[$count_g]=$dragPosition[2];

    if ($count_g>5)
    {

     if ($xpos[$count_g-5]!=$xpos[$count_g]) $angle=angle($xpos[$count_g],$zpos[$count_
g],$xpos[$count_g-5],$zpos[$count_g-5]);//avoid division by 0 and also only consider changes...//

      if (($angle>0 && $isneg=="yes")||($angle<0 && $isneg=="no")) //if there is a switch//
      {

       //figure out the direction of the last position based on whether the change is horizontal
or vertical and the position of the previous switch.
        if (abs($xpos[$count_g]-$last_posx)>abs($zpos[$count_g]-$last_posz)) //ie if change is
horizontal
        {
          if (($xpos[$count_g]-$last_posx)>0) $last_pos_is="left"; else $last_pos_is="right";
        }
        if (abs($xpos[$count_g]-$last_posx)<abs($zpos[$count_g]-$last_posz)) //ie if change is
vertical
        {
          if (($zpos[$count_g]-$last_posz)<0) $last_pos_is="below"; else $last_pos_is="above";
        }

        //upon negative positive swich or positive negative switch determine the direction of
turning.
        if (($angle>0 && $isneg=="yes")||($angle<0 && $isneg=="no"))
        {

          if ($time_before=="above")
          {
            if ($last_pos_is=="right") $wise="clock";
            if ($last_pos_is=="left") $wise="anti";
          }

          if ($time_before=="below")
          {
            if ($last_pos_is=="right") $wise="anti";
            if ($last_pos_is=="left") $wise="clock";
          }

          if ($time_before=="left")
          {
            if ($last_pos_is=="below") $wise="anti";
            if ($last_pos_is=="above") $wise="clock";
          }

          if ($time_before=="right")
          {
            if ($last_pos_is=="below") $wise="clock";
            if ($last_pos_is=="above") $wise="anti";
          }

        }//end if switch

        //base the factor by which to move the object on the size of the turns being made.
        $factor=`pow (distcalc($xpos[$count_g],$zpos[$count_g],$last_posx,$last_posz)) 2`/2;
        if ($wise=="clock") $target=-$factor;
        if ($wise=="anti")  $target=$factor;

        $num_divs=3+`pow $drag_counter 2`;
        $drag_counter=1;
        $surity=$surity+1;

        $time_before=$last_pos_is;$last_posx=$xpos[$count_g];$last_posz=$zpos[$count_g];

      }//end: if there is a switch//

      //this is used to determine whether there is a switch.
      if ($angle>=0) $isneg="no";
      if ($angle<0) $isneg="yes";
```

```
      //use the loops between switches to move smoothly to the set target - ensures a smoother
motion.

      //once the sure about the direction (it has seen enough changes to be confident) it will
move the object by changing its running_total.
      if (($num_divs>0)&&($surity>1)) $running_total=$running_total+$target/$num_divs;
      $drag_counter=$drag_counter+1;

      //set maximum and minimum positions
      if ($running_total>45) $running_total=45;

      global string $tool_choice;
      if ($tool_choice=="pletch")
      {
        if ($running_total<1) $running_total=1;
        if ($running_total>25) $running_total=25;
      }
      else
      {
        if ($running_total<-27) $running_total=-27;
        if ($running_total>45) $running_total=45;
      };

      float $obj_y=$running_total;
      setAttr ($object+".ty") $obj_y;refresh;

      $prev_angle=$angle;

    }//end: if (count_g>5)

    $count_g=$count_g+1;
  }//end proc//
```

check_screw.mel - the releaseCommand for the raise/lower tool.

Set Ortho Angle Tool

ortho_set.mel
creates a draggerContext tool used to set the ortho-snap angle. It makes use of the following scripts:

ortho_complete.mel - releaseCommand for ortho_set.

orthoset_down.mel - pressCommand for ortho_set.

orthoset_drag.mel - dragCommand for ortho_set.

ortho_angle.mel
A very simple script just used to display the ortho angle in the "heads up display".

Zoning Tool, 3D Grass and Water

sketch_land.mel
This creates window options and sets up a scriptJob so that the pencil curve tool may be used to draw enclosed shapes for water, 3D grass and land-use shapes.

joiner.mel
This script will attempt to join lines drawn with the pencil curve tool together and complete a shape with any of a number of different surfaces once the end of the line is found to be near its start. This script must be called from a "scriptJob" because of limitations with the pencil curve tool.

putty_opts.mel
Used by the 3D grass tool to set a number of options for use with Maya's sculpt surfaces tool.

Concrete Pen, Roads and Hedges

General scripts for these tools

cvline_tool.mel

This is a generic tool used to draw a line then run another procedure (for roads, concrete, hedges, and Sketched Camera). Attempts to overcome the limitation of Maya's Pencil Curve tool.

plot_pointsCV.mel

The dragCommand for the cvline_tool. It is basically the same as plot_points (used by the pletching tool) except that it uses a different context for the points (cvlineContext).

line_then.mel

Used by the "cvline_tool" to make a line and smooth it, then call on a specified procedure with which to use the line.

Hedges and Concrete Pen

hedge-etc.mel

This script is used to make a polygon of a certain width either side of a center-line and then extrude it to a certain height to form a solid object along the line. Refer to figure 9 to understand what this script is doing.

```
global proc hedge_etc(string $center_line, string $type)
{
 global float $hedge_width;
 global float $hedge_height;
 global float $path_width;
 global float $path_depth;
 float $prev_x;float $prev_z;
 float $cv_xyz[];
 string $polystring="";
 float $width;
 global int $fol_top;float $low;float $high;

 if ($type=="hedge")
 {
  $width=$hedge_width/20;//unit conversion and half...
 }
 else
 {
  $width=$path_width/20;//unit conversion and half...
 }

 //offset the curves on either side of the center-line

 string $curve1er[]=`offsetCurve  -ch off -rn false -cb 2 -st true -cl true -cr 0 -d $width -tol
0.01 -sd 5 -nr 0 1 0 $center_line`;
 $curve1=$curve1er[0];
 string $curve2er[]=`offsetCurve  -ch off -rn false -cb 2 -st true -cl true -cr 0 -d (-1*$width)
-tol 0.01 -sd 5 -nr 0 1 0 $center_line`;
 $curve2=$curve2er[0];

 int $num_cvs=1+(getAttr ($curve1 + ".spans"));

 //set variables for the lowest and highest points along the lines - this is used to place level
concrete objects at the right height.

 $low=100; $high=-100;

 //move through all CVs of the offset curves and add them to a "string" of points that will be
used to make the polygon. Also use the y value for each of the points if the object is set to
follow the land.
```

```
  $prev_x=0;$prev_z=0;
  for ($loop=0;$loop<=$num_cvs;$loop++)
  {
  float $cv_xyz[]=`getAttr ($curve1 + ".cv["+$loop+"]")`;
   if (distcalc($prev_x,$prev_z,$cv_xyz[0],$cv_xyz[2])>0.2)//avoid non_manifold geometry...
   {
    if ($low>y_of($cv_xyz[0],$cv_xyz[2])) $low=y_of($cv_xyz[0],$cv_xyz[2]);
    if ($high<y_of($cv_xyz[0],$cv_xyz[2])) $high=y_of($cv_xyz[0],$cv_xyz[2]);
    if ($fol_top)  $polystring=$polystring+" -p "+$cv_xyz[0]+" "+(y_of($cv_xyz[0],$cv_xyz[2]))+"
"+$cv_xyz[2];
    if (!$fol_top) $polystring=$polystring+" -p "+$cv_xyz[0]+" 0 "+$cv_xyz[2];
    $prev_x=$cv_xyz[0];
    $prev_z=$cv_xyz[2];
   }
  }

  $prev_x=0;$prev_z=0;
  for ($loop=$num_cvs;$loop>=0;$loop--)
  {
   float $cv_xyz2[]=`getAttr ($curve2 + ".cv["+$loop+"]")`;
   if (distcalc($prev_x,$prev_z,$cv_xyz2[0],$cv_xyz2[2])>0.2)//avoid non_manifold geometry...
   {
    if ($low>y_of($cv_xyz2[0],$cv_xyz2[2])) $low=y_of($cv_xyz2[0],$cv_xyz2[2]);
    if ($high<y_of($cv_xyz2[0],$cv_xyz2[2])) $high=y_of($cv_xyz2[0],$cv_xyz2[2]);
    if ($fol_top)  $polystring=$polystring+" -p "+$cv_xyz2[0]+" "+(y_of($cv_xyz2[0],$cv_
xyz2[2]))+" "+$cv_xyz2[2];
    if (!$fol_top) $polystring=$polystring+" -p "+$cv_xyz2[0]+" 0 "+$cv_xyz2[2];
    $prev_x=$cv_xyz2[0];
    $prev_z=$cv_xyz2[2];
   }
  }

  //now use that string to make the polygon
  string $polynamer[]=`eval("polyCreateFacet -ch on -tx 1 -s 1"+$polystring)`;
  string $polyname=$polynamer[0];

  //depending on the type of object (ie whether its a hedge or concrete) extrude it to a certain
height and set its texture.
  if ($type=="hedge")
  {
   string $poly_extruder[]=`polyExtrudeFacet -ch 0 -ty ($hedge_height/10) ($polyname+".f[0]")`;
   polyAutoProjection -ibd 1 -cm 0 -l 2 -sc 1 -o 1 -p 6 -ps 0.2 -ch 1 ($polyname+".f[0:"+($num_
cvs*2)+"]");
   global int $hollow_hedges;
   if ($hollow_hedges) catch(`sets -e -forceElement "hollow_treeSG"`);
   else catch(`sets -e -forceElement "solid_treeSG"`);
  }
  else
  {
   string $poly_extruder[];
   if ($fol_top) $poly_extruder=`polyExtrudeFacet -ch 0 -ty ($path_depth/10)
($polyname+".f[0]")`;
    else $poly_extruder=`polyExtrudeFacet -ch 0 -ty (($high-$low)+$path_depth/10)
($polyname+".f[0]")`;
   polyAutoProjection -ibd 1 -cm 0 -l 2 -sc 1 -o 1 -p 6 -ps 0.2 -ch 1 ($polyname+".f[0:"+($num_
cvs*2)+"]");
   catch(`sets -e -forceElement "pavementSG"`);
   if (!$fol_top) move -a 0 $low 0 $polyname;
  }

  delete $center_line $curve1 $curve2;

  select -cl;
 }//endp
```

Roads

road.mel

Makes a road given a center-line and setting for width, parking lanes and lamp style.

make_features.mel
Runs down lines along the sides of the road and places a specified "feature" (i.e. lamps or parking meters and cars) along its edge. Rotates the features so the they face down the road by comparing a point on the feature line to the corresponding point on the center-line.

make_lamp.mel
Called by "make_features" if a lamp is to be made. Will import a lamp from a file and place it at a specific rotation.

make_pcar.mel
Called by "make_features" if a parking meter (and its associated cars) is to be made. The file is randomly chosen from 6 possible files to ensure variation.

hide-features.mel
If a road is found to intersect another, this script looks for any "features" along the roads that lie near the intersection and deletes them so that they do not clutter the intersection.

Trees and People

Trees

tree_tool.mel
creates the tool window and the draggerContext for the Tree Tool.

placeTree.mel - pressCommand for the Tree tool - places a single tree.

spaceTrees.mel - the dragCommand for the tree_tool. This places trees based on options for spacing, tree type and size during a mouse drag.

upTree.mel - releaseCommand for the Tree tool

street_tree.mel
Takes a sphere and some cylinders and warps them with a certain amount of randomization to form a rough tree with a small number of polygons. Each tree will be different owing to the randomization. The sphere is then texture mapped with a partly transparent leaf texture.

```
global proc street_tree(float $x,float $z)
{
  global int $tree_count;
  global string $tree_name[];
  global float $tree_scaling;
  float $scale;
  $scale=(0.15+(`rand 1`/20))*$tree_scaling;

  //Make the tree body as a sphere and set its height with a small amount of randomization.
  string $treebodyer[]=`polySphere -r ($scale*4) -sx 8 -sy 8 -ch 0`;
  string $treebody=$treebodyer[0];

  setAttr ($treebody+".ty") ($scale*(7.5+(`rand 0.5`)));
  setAttr ($treebody+".tx") $x; setAttr ($treebody+".tz") $z;
  string $tree_trunk[];

  //Move the vertices of the tree body randomly by a small amount to roughen the sphere shape.
  for ($loop=0;$loop<58;$loop=$loop+1)
  {
    float $x;float $y;float $z;
    $x=$scale*(0.5-(`rand 1`));$y=$scale*(0.8-(`rand 2`));$z=$scale*(0.5-(`rand 1`));
    move -r -os $x $y $z ($treebody+".vtx["+$loop+"]");
  }

  //Give the tree body a partly-transparent leaf texture.
  select -r $treebody;
  sets -e -forceElement "hollow_treeSG";
```

```
   //Make the trunk of the tree from 2 cylinder and move the vertices mid-way and higher. This
forms a forked trunk.
    for ($loop=1;$loop<=2;$loop=$loop+1)
    {
      $tree_trunker=`polyCylinder -r ($scale*0.3) -h ($scale*5) -sx 10 -sy 4 -sz 1 -ax 0 1 0 -tx
1 -ch 0`;
      $tree_trunk[$loop]=$tree_trunker[0];

      // bottom 0:9 then 10:19 then 20:29 then 30:39 top 40:49
      setAttr ($tree_trunk[$loop]+".ty") ($scale*2.5);
      setAttr ($tree_trunk[$loop]+".tx") $x; setAttr ($tree_trunk[$loop]+".tz") $z;

      $move_branch=($scale*(`rand 1`));
      select -r ($tree_trunk[$loop]+".vtx[40:49]");select -add ($tree_trunk[$loop]+".vtx[51]");//
top
      scale .4 1 .4; move -r $move_branch 0 $move_branch;

      select -r ($tree_trunk[$loop]+".vtx[30:39]");//1down...
      scale .5 1 .5; move -r ($move_branch/2) 0 ($move_branch/2);

      select -r ($tree_trunk[$loop]+".vtx[0:9]");//bottom
      scale 1.2 1 1.2;

      select -r $tree_trunk[$loop];
      sets -e -forceElement "tree_trunkSG";
      float $rotate=30-(`rand 60`);
      if ($loop==2) rotate 0 (180+$rotate) 0;
    }

   //Group the parts of the tree and place it so that it sits on the landscape - y_of will return
the height value for the landscape at the given point.
    $tree_count=$tree_count+1;
    $tree_name[$tree_count]=`group $treebody $tree_trunk[1] $tree_trunk[2]`;
    setAttr ($tree_name[$tree_count]+".ty") (y_of($x,$z));

  }//endp
```

umbrella_tree.mel

Similar to the above procedure except in this case the sphere is made so that the bottom is pushed back up underneath the top to form and "umbrella" shape. This shape is duplicated and shifted is then texture mapped with a mostly transparent texture. These trees are made larger by default and may be used to form a canopy.

solid_tree.mel

The sphere for the solid tree is elongated then texture mapped with a solid pine-needle texture. These trees are smaller and can be used as ornamental trees.

People

peopleTool.mel

Creates the tool window and the draggerContext for the People Tool.

placePeep.mel - pressCommand for the People Spray tool - places a single person.

spacePeep.mel - the dragCommand for the peopleTool. This places (or sprays) people based on options for scatter, frequency, gender and activity during a mouse drag.

upPeep.mel - releaseCommand for the People Spray tool

maketh_man.mel

This is a similar tool to trees in that it also automated the moving and scaling of vertices in order to make an object. It also makes used of randomization to ensure that the people don't look repetitive. The script is used to make a person of a specified gender and will move that person's arms and legs according to an "activity" setting.

```
global proc maketh_man(float $x,float $z, float $rot, string $sex,float $activity)
```

```
{
 string $man_cylinderer[]=`polyCylinder -r 0.1 -h 0.6 -sx 10 -sy 8 -sz 1 -ax 0 1 0 -tx 1 -ch 0`;
 $man_cylinder=$man_cylinderer[0];
 move -r $x 0.3 $z;

 //head
 select -r ($man_cylinder+".vtx[70:89]");
 scale -r 0.45 0.7 0.45;
 move -r  0 0.073 0;
 select -r ($man_cylinder+".vtx[80:89]");
 scale -r 0.9 1 0.9;

 //neck
 select -r ($man_cylinder+".vtx[60:69]") ;
 scale -r 0.3 0.3 0.3;
 move -r 0 0.166 0;
 select -r ($man_cylinder+".vtx[50:59]");
 scale -r 0.4 0.4 0.4;
 move -r 0 0.162 0;

 if ($sex=="female")
 {//make hairdo
  move -r 0 0.3 0 ($man_cylinder+".scalePivot");
  select -r ($man_cylinder+".vtx[70:71]") ($man_cylinder+".vtx[77:81]") ($man_cylinder+".vtx[87:
89]") ;
  scale -r 1 (1+`rand 0.6`) 1;
 }

 //shoulders
 select -r ($man_cylinder+".vtx[40:49]") ;
 if ($sex=="female") scale -r .7 .7 (`rand 0.1`+0.8);
 else scale -r (`rand 0.1`+0.8) (`rand 0.1`+0.8) (`rand 0.2`+1.1);
 move -a -y 0.46;

 //waist
 select -r ($man_cylinder+".vtx[30:39]") ;
 scale -a .8 .8 1;
 move -a -y 0.268;
 select -r ($man_cylinder+".vtx[20:29]") ;
 scale -r .6 .6 .8;
 move -a -y 0.25;

 //feet
 select -r ($man_cylinder+".vtx[10:19]") ;
 scale -r .3 1 .6;
 move -a -y 0.025;
 select -r ($man_cylinder+".vtx[0:9]") ;
 scale -r .4 1 .8;

 //set the size of the person
 if ($sex=="female")
 {
  move -r 0 -0.6 0 ($man_cylinder+".scalePivot");
  scale -r 0.85 (0.8+`rand 0.1`)  0.85 $man_cylinder;
 }
 else
 {
   move -r 0 -0.3 0 ($man_cylinder+".scalePivot");
   scale -r 0.9 (0.85+`rand 0.2`) 1 $man_cylinder;
 }

 //now give them movement according to activity variable passed...
 int $negpos=-1+(`rand 3`);if ($negpos<=0) $negpos=-1;
 select -r ($man_cylinder+".vtx[0:4]") ($man_cylinder+".vtx[9:14]") ($man_cylinder+".vtx[19]");
 move -r -x ($negpos*$activity*0.05);

 select -r ($man_cylinder+".vtx[5:8]") ($man_cylinder+".vtx[15:18]") ;
 move -r -x (-1*$negpos*$activity*0.05);

 select -r ($man_cylinder+".vtx[36:37]");
 move -r ($negpos*$activity*0.05) 0 0.02;
```

```
  select -r ($man_cylinder+".vtx[31:32]");
  move -r (-1*$negpos*$activity*0.05) 0 -0.02;

  select -r ($man_cylinder+".vtx[60:89]") ($man_cylinder+".vtx[91]") ;//head
  move -r -x (-0.015*$activity);

  select -r ($man_cylinder+".vtx[40:89]") ($man_cylinder+".vtx[91]") ;
  move -r (-0.025*$activity) ((-0.01+`rand 0.02`)*$activity) 0;

  select -r ($man_cylinder+".vtx[48:49]") ($man_cylinder+".vtx[40]");//flatten chest
  scale -r 0.6 1 1;
  move -r -x (-0.008*$activity);

  select -r $man_cylinder;
  rotate -r 0 $rot 0;
  move -r -y (y_of($x,$z));
  select -cl;

}
```

Vizualization

Sketched Camera

cam_sketch.mel

Works out whether a line is straight or a node then a line and sets the Sketched Camera accordingly. The actual tool context for the sketched camera is made by the cvline_tool which is listed under "Concrete Pen, Roads and Hedges."

Leashed Camera (or CamFollow)

camfollow_tool.mel

Create the tool window and tool contexts for the Leashed Camera tool.

placeCamFollow.mel - the pressCommand for the Leashed Camera tool - can also be used to set the camera at eye level or set it to be level with the ground.

CamFollow.mel

The dragCommand for the Leashed Camera tool. This script sets the position of the camera and the camera view node. Cameras will always rotate to look at their view nodes. This script sets the view node at the position of the mouse pointer while the camera is made to "lag" a few steps behind. The result is similar to a real-time motion path because the camera will fly along the line drawn by the user as it is being drawn.

```
global proc CamFollow()
{
   float $dragPosition[] = `draggerContext -query -dragPoint camfol2Context`;
   global float $posx[];
   global float $posz[];
   global int $since_start;
   global int $lag=35;
   float $velocity;
   float $lag_val;
   int $counter;
   global int $lag_vel;
   global string $cam_fol_name;
   global string $cam_fol_light;
   float $view_x;float $view_z;
   $velocity=velcalc();

   //use the velocity of the camera to determine how far behind the drag point the camera should
lag (ie the length of the leash).
   $lag_val=($lag-($velocity-0.2)*38); //values based on how it feels - this could be changed to
accomodate performance drop-off when dealing with more polygons - not implemented yet//
   if ($lag_val<12) $lag_val=12;
```

```
    $lag_vel=$lag_val;
    if ($lag_vel>$lag) $lag_vel=$lag;

    if ($since_start<$lag)
    {
      $lag_vel=$since_start/($lag_val/$lag);
      if ($lag_vel>$lag_val) $lag_vel=$lag_val-1;
      $since_start=$since_start+1;
        //increases every time a drag move is registered since the start of the drag.
    }

    $counter=$lag;

    //average the view points for a smoother ride - ie to prevent the camera from moving abruptly
as it looks at each new point
    $view_x=($dragPosition[0]+$posx[0])/2;
    $view_z=($dragPosition[2]+$posz[0])/2;


    //set the view_point at the current cursor position and place it on the landscape.
    global float $view_yoff;

    setAttr ($cam_fol_name+"_view.tx") $view_x;
    setAttr ($cam_fol_name+"_view.ty") (y_of($view_x,$view_z)+$view_yoff);
    setAttr ($cam_fol_name+"_view.tz") $view_z;

    while ($counter>0)
    {$counter=$counter-1; //makes counter 15 initially...//
    $posx[$counter+1]=$posx[$counter]; //compare [16] and [15] first... down to [1] and [0]//
    $posz[$counter+1]=$posz[$counter];
    }

    float $avg_x;float $avg_z;

    //average the camera points for a smoother ride - using 3 points on a short leash (20-35 lag
points) or 6 points when the leash is long (35 lag points).
    //there will be no averaging if the leash is less than 20 lag points.
    if ($lag_vel>20)
    {
    $avg_x=($posx[$lag_vel-2]+$posx[$lag_vel-1]+$posx[$lag_vel])/3;
    $avg_z=($posz[$lag_vel-2]+$posz[$lag_vel-1]+$posz[$lag_vel])/3;
    }

    if ($lag_vel==35)
    {
    $avg_x=($posx[$lag_vel-5]+$posx[$lag_vel-4]+$posx[$lag_vel-3]+$posx[$lag_vel-2]+$posx[$lag_
vel-1]+$posx[$lag_vel])/6;
    $avg_z=($posz[$lag_vel-5]+$posz[$lag_vel-4]+$posz[$lag_vel-3]+$posz[$lag_vel-2]+$posz[$lag_
vel-1]+$posz[$lag_vel])/6;
    }

    if ($lag_vel<=20)
    {
     if ($lag_vel>1)
     {
      $avg_x=($posx[$lag_vel-1]+$posx[$lag_vel])/2;
      $avg_z=($posz[$lag_vel-1]+$posz[$lag_vel])/2;
     }
     if ($lag_vel<=1)
     {
      $avg_x=$posx[$lag_vel];
      $avg_z=$posz[$lag_vel];
     }
    }

    //set the camera at the position determined by the length of the leash and the averaging pro-
cess and place it on the landscape at that point.
    global float $cam_yoff;

    setAttr ($cam_fol_name+".tx") $avg_x;
    setAttr ($cam_fol_name+".ty") (y_of($avg_x,$avg_z)+$cam_yoff);
    setAttr ($cam_fol_name+".tz") $avg_z;
```

```
    $posx[0]=$dragPosition[0];
    $posz[0]=$dragPosition[2];

    refresh;
}//end proc//
```

velcalc.mel
Calculates the velocity of the pointer while dragging the Leashed camera. This is used to lengthen or tighten the "leash" depending on how fast one moves the pointer.

Solar Analysis Tools

sunset_tool.mel
Sets the tool window (by calling sunset_window) and creates the tool context for the Solar Analysis tool.

sunset_down.mel - the pressCommand for the sunset_tool. This will set the dragCommand to sunset_drag or sunset_interest based on the window in which the mouse is pressed. It will also turn shadows off in the interests of optimization.

sunset_drag.mel - the dragCommand for the sunset_tool called when dragging in the tool window (in the trepezoid). This calls on position_sun to set the position of the solar system object for that day and time.

sunset_interest.mel - the dragCommand for the sunset_tool called when dragging in the plan view. This moves the entire solar system object so that the center of interest is placed at the position of the drag.

sunset_up.mel - the releaseCommand for the sunset_tool.

position_sun.mel
Used to position the sun in the solar system based on where the sun object lies in the trapezoid.

sunset_window.mel
Creates the tool options for the Solar Analysis tool. This window includes model view from the side of the trapezoidal object used to place the sun.

display_date.mel - display the date in an intelligible form - used in the "heads-up display" to display the date.

display_date.mel - display the time in an intelligible form - used in the "heads-up display" to display the time.

make_solar_system.mel
This script makes a solar system object that is moved and rotated to represent changes in season and time respectively. The solar system contains an "ecliptic" - a circle for the sun to follow - a sun, and a node of interest. The sun object contains a directional light that is set to cast "depth-map" shadows. Both the sun and the node of interest contain cameras and view points so that the user can look either from the sun at the point of interest or from the point of interest towards the sun.

Curiosity Camera

construction_cam.mel
The curiosity camera was formerly known as the "construction camera" as it looks at objects while they are being constructed. This script is used to create the curiosity camera if it doesn't exist and sets up a scriptJob that uses background_process to move the camera to the specified position.

kitty_interest.mel
This script creates a new tool context that allows the user to select an object of interest for the Curiosity Camera.

background_process.mel
Used to move the Curiosity Camera when all else is idle.

Windows, View and Toolbars

shelf_window.mel
This creates the main toolbox window for DASPE. Three shelf states are possible: Construction, Landscape and Visualize.

alt_switch.mel - called when the user clicks an alternative - switches the alternative buttons and turns layers on and off.

tool_window.mel
This script creates the window that is used to display options for each different tool. The window is first created with a number of generic elements. These are all turned off initially and are turned back on by specific tools as needed.

toggle_tools.mel - ensures the tool buttons work as radio-buttons and disables the current tool from being deselected.

spective_window.mel
Creates a window with a perspective view and buttons used to control it.

spect_switch.mel - used to ensure that check-boxes on the perspective window act as radio-buttons rather than check-boxes.

elevation_window.mel
Create the elevation/section window and its associated buttons.

RL_selector.mel
Creates a tool used to pick an object for use with the raise/lower tool.

focuscondi.mel
This script is used in creating a "condition" that is used to determine when the window under the mouse pointer gets changed.

switch_tools.mel
Called whenever the window under the pointer changes, it will switch the tool state to suit the specified window if necessary.

gc_switch.mel
Used to switch check-boxes so that they act like radio buttons.

lamp_switch.mel - similar to gc_switch, but specific to lamp selection.

set_section.mel
-

set_sect_pos.mel
-

setTool.mel
DASPE uses two tool states (one for the perspective window, the other for the plan). This script allows either or both to be set to a specified tool.

Library scripts

(These scripts are used by many other procedures and fall under no particular category):

angle.mel
(convenience) just a more convenient way to get an angle between two point in plan - used by a number of scripts including check_screw.

averagepoint.mel

(convenience) used to get the average of a number of points - useful for smoothing.

CASTangle.mel
(convenience) the same as angle except that the angle returned is out of 360 degrees, not just 90 degrees.

cvdist.mel
(convenience) returns the distance between two specified CVs for a given line.

delete_plotted.mel
Most sketched tools plot a line as the user draws. This script deletes it.

distcalc.mel
(convenience) return the linear distance between 2 points.

osTest.mel
Checks the operating system so that some commands may be disabled for older versions.

point_is_on_line.mel
Checks to see whether a given point lies on a certain line.

smooth_curve.mel
Basic averaging used to smooth the curve made from mouse or particularly stylus input.

y_of.mel
For any given x and z position on the plan, this script will return an associated y position based on the surface of the land.

Acknowledgements

This thesis was made possible by the assistance and support of a large number of people.

First I would like to thank Joe Ferreira, Eran Ben Joseph and Dennis Frenchman of the Department of Urban Studies and Planning, as well as Hiroshi Ishii of the MIT Media Lab's Tangible Media Group for their support and guidance throughout the development of this thesis.

Thanks should also go to those who took the time to partake in my gestural sketching survey: Antonio, Tim, Martha, Sushila, Apiwat, Shahid, Marisa, Rossana, Sunil, Kris, Roohi and Michelle. While the survey itself provided only limited substance for this thesis, the discussions it generated were greatly inspiring.

Phil Thompson provided advice, encouragement and a great deal of assistance with the physical setup of DASPE in the Computer Resource Lab. Duncan Kincaid also gave inspiration and support and supplied the most thorough user-testing I could have hoped for by bringing his gifted nephew to try out the system. That testing provided numerous insights about the usability of the system and informed a number of changes that were subsequently made.

The students in Eran Ben Joseph's Site Planning class gave comments and support at a number of presentations and I'd like to thank Jordan Karp in particular for his patience in putting up with all the hassles involved in being the first user to put DASPE through its paces on a real project.

Brygg Ullmer (of Ishii's Tangible Media Group) was a valuable resource for pertinent examples of related work. Ben Piper and Yao Wang provided much inspiration through their work on the Illuminating Clay project and were also forthcoming with advice for DASPE. Andy Bennett and Bryan Ewert provided valuable e-mail support for Maya and MEL problems and I thank them for their time.

I would like to thank MassGIS and the Town of Brookline for help in locating a digitizing table, and in particular Robert Kefalas for taking the trouble to fetch and deliver this hefty piece of equipment right to our door. Steven Wilson of Sasaki made a special visit to see DASPE and gave a valuable critique on its potential use a design tool in an urban planning firm.

Lastly I'd like to thank my father for his pedantic proof-reading without which the text of thesis may have been an embarrassment. Of course I take full responsibility for any latent typographic, structural or grammatical errors he and my readers may have missed.