Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling

David Anderson[1], James L. Frankel[1,4], Joe Marks[1],
Aseem Agarwala[1], Paul Beardsley[1], Jessica Hodgins[2], Darren Leigh[1],
Kathy Ryall[3], Eddie Sullivan[1], Jonathan S. Yedidia[1]

[1]MERL—Mitsubishi Electric Research Laboratory [2]Georgia Institute of Technology
[3]University of Virginia [4]Frankel and Associates, Inc.

Contact: marks@merl.com

Abstract

Construction toys are a superb medium for creating geometric models. We argue that such toys, suitably instrumented or sensed, could be the inspiration for a new generation of easy-to-use, tangible modeling systems—especially if the tangible modeling is combined with graphical-interpretation techniques for enhancing nascent models automatically. The three key technologies needed to realize this idea are embedded computation, vision-based acquisition, and graphical interpretation. We sample these technologies in the context of two novel modeling systems: physical building blocks that self-describe, interpret, and decorate the structures into which they are assembled; and a system for scanning, interpreting, and animating clay figures.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—interaction techniques.

Keywords: Applications, Geometric Modeling, Graphics Systems, HCI (Human-Computer Interface), Shape Recognition, User Interface Hardware.
Additional Keywords: Embedded Computation, Tangible User Interfaces, Perceptual User Interfaces, Transmedia.

1 Introduction

Artists using standard 3D modeling packages must specify precisely the geometric and material properties of the models they create, and therein lies much of the complexity and tedium of using those tools. By contrast, children playing with construction toys like Lego™ and K'nex™ make simple models easily, and use their imaginations to fill in the details. We would like to transform computer-based geometric modeling into that same kind of playful, tactile experience but without sacrificing the ability to create the interesting geometric detail and movement that make 3D graphics and animation compelling. To retain the tactile experience of model manipulation, we look to tangible-interface technology; and to create detailed, fully realized models, we use new methods for graphically interpreting a nascent model by recognizing and augmenting its salient features.

SIGGRAPH 2000, New Orleans, LA USA

This combination of *tangible interaction* and *graphical interpretation* is investigated in a pair of case studies. Tangible modeling can be approached in two ways: either by directly instrumenting the modeling medium with embedded computation or by using external sensors to capture the geometry. Our first system consists of computational building blocks assembled into physical structures that in their aggregate determine and communicate their own geometric arrangement. A rule-based system interprets these structures as buildings, parses their architectural features, then adds geometric detail and decorative enhancements (Figures 1 and 4). Our second system uses simple and robust computer vision to capture volumetric scans of clay models of such common toy-like objects as people, animals, trees, houses, cars, and boats. A volumetric matching algorithm allows us to recognize, interpret, and animate the clay models (Figure 2).

2 Computational Building Blocks

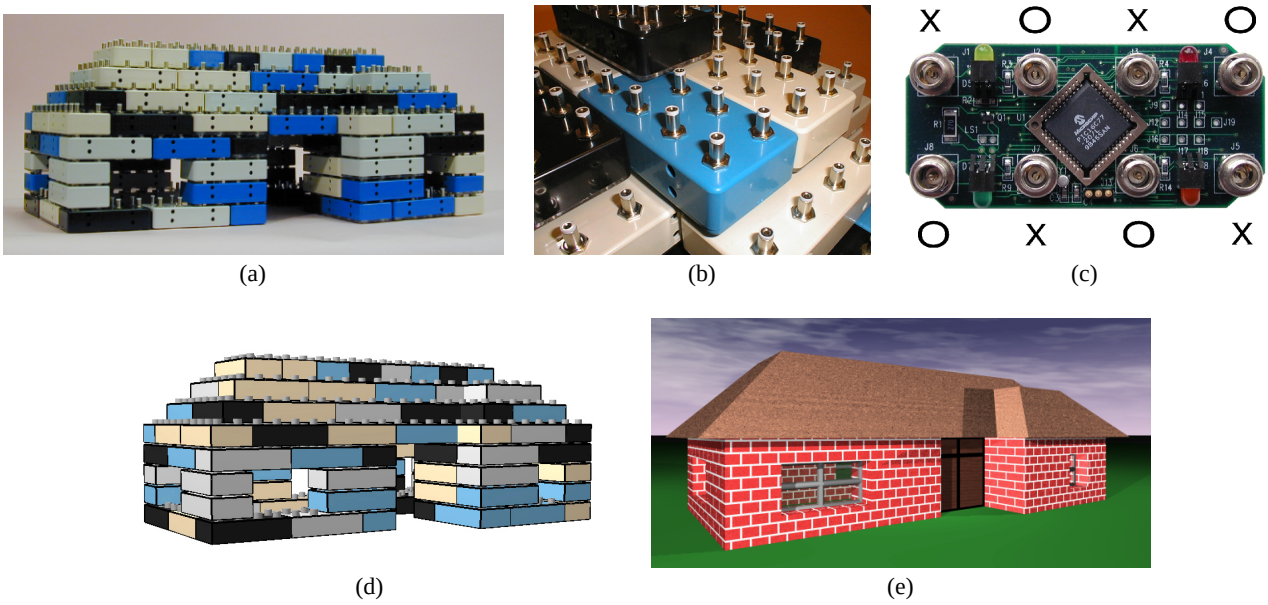
The vision of a tangible 3D geometric modeling system that uses building blocks with embedded computation has been pursued by several groups over the past 20 years. Research on this topic began with the pioneering projects of Aish [1, 2] and of Frazer [15, 16, 12, 14, 13], and was renewed more recently by Dewey and Patera [7, 3].[1]

All of these systems take advantage of the idea that completely and robustly determining the geometry of a tangible model follows naturally if the model is built from rigidly connected building blocks of known size and shape. Recovering 3D geometry is then reduced to the problem of determining the identity and connectivity of the blocks and communicating that information to a host computer. However, these systems differ significantly in the details of their design and implementation. A broad range of solutions have been tried for these fundamental engineering problems:

- *How do blocks connect?* Blocks that can stack only vertically have a low "constructive versatility" relative to, say, Lego™ blocks (a pair of standard 2×4 Lego™ blocks can connect in 184 different configurations). Simple, symmetrical connectors are the key to achieving high constructive versatility.
- *How are blocks powered?* Self-powered blocks allow use of simpler connectors but increase the cost, maintenance,

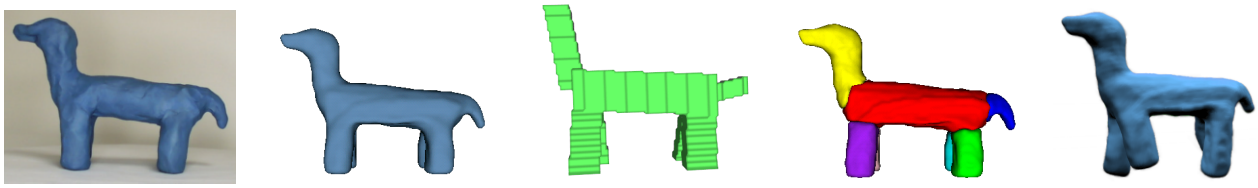
[1]The AlgoBlock [31] and Triangles [18, 19] systems are similar in architecture to the tangible modelers cited above. However, their target application is visual/tangible programming, not geometric modeling; and both systems enable the general description of 2D structure only, not 3D.

Two other tangible modeling systems deserve mention. The "Active Lego™ Baseplate Project" at MIT [23] addressed the issue of 3D geometric modeling, but it was only a paper design and was never implemented. The Monkey™ is a posable articulated linkage that is used for keyframing and performance capture [10]; it is a successful product.



(a) (b) (c) (d) (e)

Figure 1: (a) a physical block structure comprising 98 blocks; (b) a close-up of the blocks; (c) a bottom view of the circuit board inside each block; and renderings of the virtual model recovered from the structure, one literal (d) and one interpreted (e). The literal rendering uses associated shapes and colors to render the blocks. The virtual model is augmented automatically for the interpreted rendering.



(a) (b) (c) (d) (e)

Figure 2: (a) a clay model; (b) its volumetric scan, computed from silhouette information; (c) the best match for it from a small library of object templates; (d) the constituent parts of the interpreted virtual model; and (e) a frame from an automatically generated animation of the virtual model running.

and complexity of the individual blocks. Externally powered blocks require connectors that handle the distribution of power.

- *How do blocks communicate?* The earliest systems used simple electronics to create a circuit-switched network in a block structure. Recent systems have used a microcontroller in each block, and a variety of message-passing architectures for communication.
- *How is geometry computed?* There are two general strategies for computing connectivity, and thereby geometry. At one extreme the connectivity computation can be controlled centrally by the host computer; at the other extreme, it can be organized as a distributed computation among the computing elements in the blocks.

The system we have developed is just one point in a large space spanned by the dimensions of connection, communication, and computation, but it illustrates well the various design and engineering issues involved. Its distinctive characteristics include the following:

- *Very simple physical/electrical connection:* We have based the physical design of our blocks on the popular Lego™ block. Although this choice achieves much greater constructive versatility than any previous system, it comes at the price of extremely simple connectors. Our standard block has eight plugs on the top, and eight jacks on the bottom. The plugs and jacks have only two conductors each, one for power distribution and one for bidirectional signals.
- *Asynchronous, distributed communication:* These simple connectors make it impossible to have a common bus linking all our blocks (in addition to point-to-point connections). The software of many previous systems was simplified by using such a bus for global synchronization and communication. All communication in our block structures is based on asynchronous message passing between physically connected blocks.
- *Parallel, distributed computation of structure:* Our design goal was to build self-describing structures of up to 500 blocks. To complete the distributed computation of structure for a 500-block model in a reasonable time we had to exploit parallelism, which further complicated an already complicated distributed computation.
- *Automatic detailing:* A modeling system that makes it easy to

create coarse models will be of limited use if the refinement of the models requires learning a complex user interface such as that found in today's animation and CAD systems. Automatic detailing based on graphical interpretation can make block structures look less blocky and more interesting. As an illustration of what is possible, our system interprets a block structure as a building; it identifies walls, roofs, windows, doors, and other features, and augments them with additional geometric and surface detail in various styles to produce more interesting models.

2.1 System description

For economy and ease of development, we used mostly off-the-shelf components in the construction of our building-block prototypes (Figure 1(b)). A block consists of a 100mm (L) $\times$ 50mm (W) $\times$ 25mm (H) plastic box that is drilled to accommodate slightly modified DC power connectors. Eight plugs are on top of the block, and eight jacks are on the bottom. The dimensions of the box and the locations of the plugs and jacks are such that our building blocks fit together like Lego™ blocks.

Each connector has just two conductors. However, instead of using one for power and one for ground, we use the inner pin as a signal line for bidirectional communication, and the outer sleeve for power distribution. Each block is wired internally so that connectors with power and ground on their outer sleeves, respectively, are arranged in an alternating pattern, as shown in Figure 1(c). The polarity of the connector sleeves marked 'X' is different from that of the sleeves marked 'O.' Thus each block has at least one connection to power and one to ground in any typical Lego™-block structure, i.e., one in which no block is connected at only one corner or at only diagonally opposite corners. There is no way to tell a priori which connector sleeves have power or ground, but this problem is solved by the use of a full-wave bridge rectifier.

We chose the PIC16F877 microcontroller as the best single-chip solution to the various design problems posed by our application. Its features include: a relatively large number of I/O pins (16 are required for communication, and it has 33); compact size; low-power, high-speed (20 Mhz) CMOS technology; an 8-bit RISC CPU; $8\text{K} \times 14$-bit words of FLASH program memory; 368×8-bit bytes of data memory (RAM); 256×8-bit bytes of EEPROM data memory; a hardware Universal Synchronous Asynchronous Receiver Transmitter (USART) that we use for debugging; and interrupt handling. The program and data in each block's microcontroller are identical except for a unique ID number.

We left several pads in the periphery of our custom circuit board to accommodate such additional transducers and sensors inside a block as speakers, and proximity and touch detectors. Alternatively, the board can be trimmed to fit inside a 2×2 building block without affecting its basic functionality.

2.2 Geometry determination

A fully assembled block structure computes its own geometry in three phases. When a block is powered on, it immediately enters Phase 1 of the geometry-determination algorithm. Lacking a global communication bus, the switching on of electrical power is the only source of synchronization, which is necessarily approximate because of small delays in power propagation throughout a block structure.

All 16 signal lines in a block are normally held high by pull-up resistors. Phase 1 begins with each block pulling its top signal lines (those in the plugs) low. Each block then tests its bottom signal lines (those in the jacks) to determine and record which of them have been pulled low by some other block. After a short delay to ensure that the approximately synchronized blocks do not try to drive shared signal lines simultaneously in both directions, this test is then repeated with the roles of top and bottom lines reversed. Thus when Phase 1 is complete, each block has identified in parallel which of its lines are *connected*, i.e., are attached to other blocks, but it does not know the identity of these neighboring blocks.

After another short delay, each block enters Phase 2 of the algorithm during which blocks communicate with their neighbors over the connected lines found in Phase 1. At the start of Phase 2, each block listens on its connected bottom lines for transmitted packets that contain the ID of the transmitting block and the number of the connector over which it is transmitting.[2] The receiving block records this information with its own ID number and the number of the connector over which it received the transmission. The combined data form a complete record of a single connection between two blocks. When a block has successfully received a transmission on all of its connected bottom lines, it begins transmitting on its connected top lines, iterating through them in order. Connectivity information, therefore, flows initially through the block structure from bottom to top, with the potential for significant parallel communication.

After a block has completed the first half of Phase 2, it knows to which connector of which block each of its own bottom connectors is attached. During the second half of Phase 2, the procedure is repeated with blocks listening on their top connected lines and transmitting on their bottom connected lines. Thus, at the end of Phase 2, each block has acquired and recorded in its database complete knowledge about all of its connected lines: the IDs of the connected pair of blocks and the connector numbers by which they are attached. Each connected line that is processed successfully in Phase 2 is termed *valid*.

In Phase 3, the connectivity information determined in Phase 2 is communicated to the host computer through the *drain*, a special block that runs slightly different software and has a serial connection to the host computer. In addition to mediating communication between a block structure and the host, the drain also supplies power to the blocks and may be attached to any part of the structure. During Phase 3 all blocks listen for messages on all of their valid lines. When a *request-to-drain* message is received, a block transmits packets containing all of its connectivity information on its *drain connector*, the one from which it received the request-to-drain message. When the block has successfully completed these transmissions, it forwards the request-to-drain message on the first of its valid lines, and enters a message-forwarding mode. If it receives a packet containing connectivity information, it stores and forwards it on its drain connector; if it receives subsequent request-to-drain messages, it responds with an *already-drained* message; when it receives an already-drained or *done* message, it forwards the request-to-drain message on its next valid line or sends a done message on its drain connector when all its valid lines have been processed.

The first request-to-drain message is injected into the structure by the drain, and permission to drain then percolates through the block structure in a preorder traversal of the blocks. Although this traversal is performed sequentially—only one block has permission to drain at any point in time—the forwarding of messages towards the drain is pipelined, thereby achieving some parallelism in this phase as well.

At the end of Phase 3 the host computer should have complete connectivity information for the block structure. (In fact, it should have redundant information because each connectivity datum is reported twice, once by each of the two blocks involved. This re-

[2] Transmitted packets may be missed if the receiving microcontroller is busy when transmission commences. Noise may also corrupt a message. Therefore all packets transmitted in Phases 2 and 3 have checksums and are acknowledged, and faulty transmissions are retried after an appropriate timeout.

dundancy contributes to the robustness of the system, but it can be eliminated for greater efficiency.) The host also has shape data for each block, indexed by ID. These data are recorded when a block is programmed. A straightforward recursive procedure ought now to give the 3D structure of the block structure, which can then be used to produce a geometric scene description suitable as input to a variety of common 3D graphics applications. However, occasionally the host does not obtain complete information: due to mechanical stresses in the structure, some connections fail (about 0.5% of them on average) such that their connectivity data are not acquired. Our geometry-recovery procedure therefore determines the most likely block structure given potentially incomplete connectivity data.

The block structure in Figure 1(a) contains 98 blocks. The time required for this structure to compute its own geometry is 35 seconds. The structure in Figure 4(c) contains 560 blocks and requires 53 minutes for geometry determination. Almost all of the time is spent in Phase 3 of the algorithm.

2.3 Graphical interpretation

For a *literal rendering* of the block structure, the host uses associated values for the rendering attributes of each block, such as shape, color, and texture. Figure 1(d) shows a literal rendering of the block structure in Figure 1(a); color and shape values have been chosen to mimic the physical blocks.

Examples of *graphical interpretations* are shown in Figures 1(e) and 4(b) and (d). To produce these enhanced renderings, our system generates a description of a block structure as a set of logical axioms, one to assert the existence and location of each block. These axioms serve as input to a logic program written in Prolog that can identify architectural features of a block structure. For example, the rules in Figure 3 compute which blocks constitute the walls and roof of a structure, interpreted as a building.[3] Recognized structural elements can be decorated with additional geometry and surface detail to enhance the visual appearance of the rendered model.

To validate the ability of the computer to generate interpreted renderings, we handcrafted a few distinct styles that can be applied to block structures automatically. For a fully realized application, we would develop more interactive user interfaces for customizing and applying these interpretive styles. We return to this point in the concluding section of the paper.

3 Clay

Using external sensors to capture geometry is the alternative technology for supporting tangible modeling. For our second case study, we were inspired by the ancient myth of Pygmalion, whose sculpture of a woman was brought to life by Venus [24]. We set ourselves the goal of bringing clay models to life automatically. Although any practical application would divide this task more evenly between user and computer, we tried to fully automate the system in order to explore the limits of the technology, just as we did in our previous case study. In the following subsections we present the details of the hardware and software used to scan, recognize, interpret, and animate 3D clay models.

[3] In the early 70's Winston developed a landmark program that could learn about simple architectural structures from positive and negative examples of those structures [33]. However, the robust recognition of the important structural elements in our block models requires hand-crafted rules of much greater complexity than those that could be learned by Winston's approach.

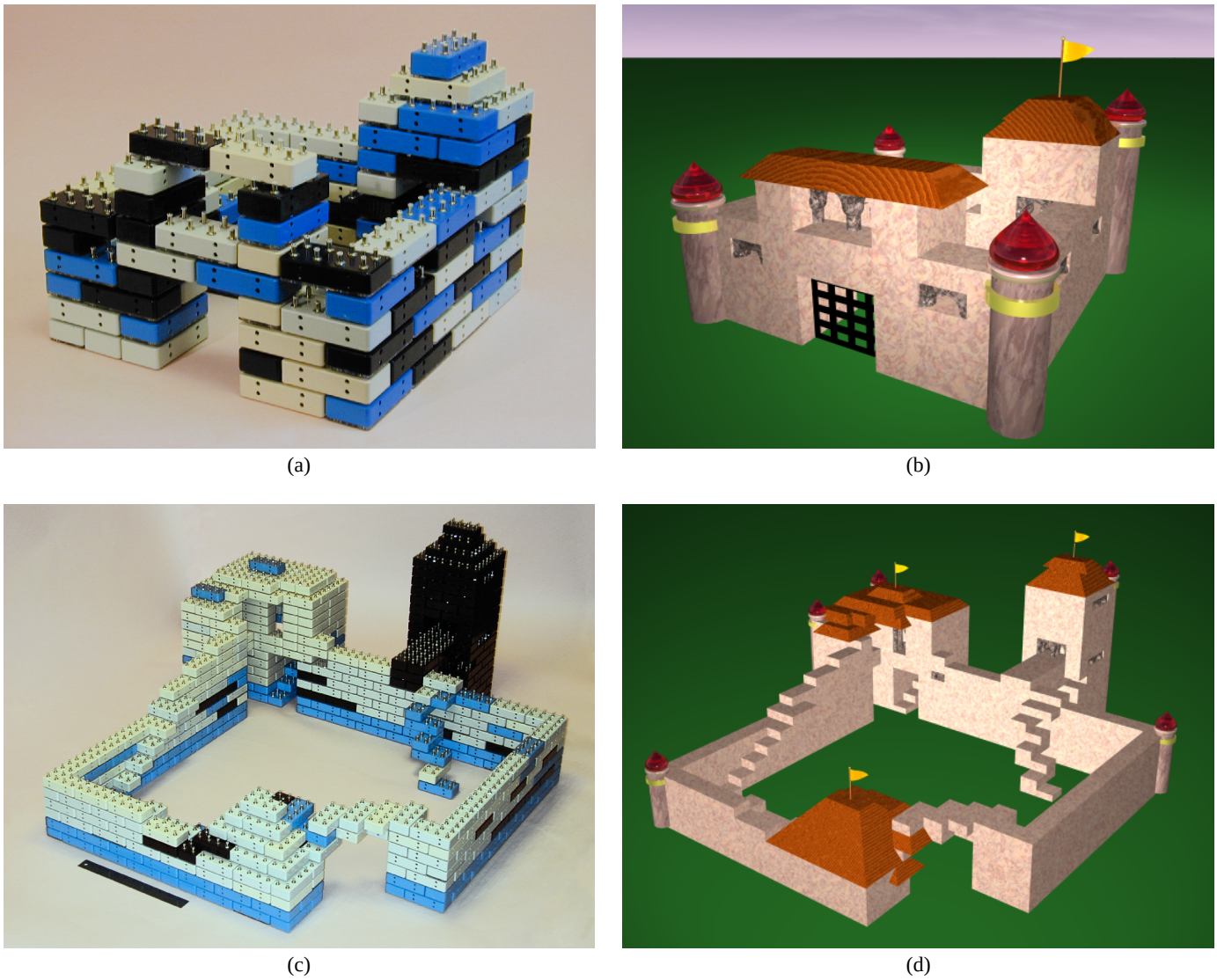
```
% wall/1 finds sets of blocks that form walls. A wall is defined to be
% a contiguous set of blocks that lie flush against some vertical plane,
% and that constitute a given fraction of the structure.
wall(WALL_BLOCKS) :-
  structure_bbox(X_MIN, X_MAX, _, _, Z_MIN, Z_MAX),
  candidate_planes(X_MIN, X_MAX, Z_MIN, Z_MAX, U, V, W, R),
  lies_flush_against(U, V, W, R, PLANE_BLOCKS),
  contiguous_subsets(PLANE_BLOCKS, PLANE_BLOCKS_SUBSETS),
  member(WALL_BLOCKS, PLANE_BLOCKS_SUBSETS),
  big_enough(WALL_BLOCKS).

% wall_tops/1 finds the blocks that are the tops of walls.
wall_tops(WALL_TOPS) :-
  setof(BLOCK,
        WALL_BLOCKS ^
          (wall(WALL_BLOCKS),
           member(BLOCK, WALL_BLOCKS),
           not_overhung(BLOCK, WALL_BLOCKS)),
        WALL_TOPS).

% roof_blocks/1 computes the set of blocks make up the roof, which is
% defined to be those blocks that rest directly or indirectly on the tops of
% walls. The indirectly resting blocks are computed by grow_roof/2.
roof_blocks(ROOF_BLOCKS) :-
  findall(BLOCK1,
          (wall_tops(WT_BLOCKS),
           member(WT_BLOCK, WT_BLOCKS),
           on_top_of(BLOCK1, WT_BLOCK)),
          BASE_BLOCKS_BAG),
  setof(BLOCK2,
        member(BLOCK2, BASE_BLOCKS_BAG),
        BASE_BLOCKS),
  grow_roof(BASE_BLOCKS, ROOF_BLOCKS).

grow_roof(NASCENT_ROOF_BLOCKS, FINAL_ROOF_BLOCKS) :-
  member(BLOCK1, NASCENT_ROOF_BLOCKS),
  on_top_of(BLOCK2, BLOCK1),
  not member(BLOCK2, NASCENT_ROOF_BLOCKS),
  grow_roof([BLOCK2 | NASCENT_ROOF_BLOCKS], FINAL_ROOF_BLOCKS),
  !.

grow_roof(ROOF_BLOCKS, ROOF_BLOCKS).
```

Figure 3: Recognizing the structural elements of a block structure by logic programming.

3.1 System description and geometry determination

Our scanning system consists of a motorized rotary table, a consumer-quality digital camera, a laser striper (optional), and a host computer (Figure 5). The camera is calibrated from an image of an object of known dimensions. The clay model to be scanned is placed upright and face forward on the rotary table. (It is convenient for the matching process to have models placed in a known orientation. Inferring orientation automatically is certainly feasible but seems unnecessary for modeling applications that involve cooperative users.) The camera captures an image sequence of the model as it rotates, and a volumetric scan is generated from silhouettes [4]. This approach worked well on the majority of models we scanned, but when significant concavities were present (e.g., the door and windows of the house in Figure 8), the laser striper could be used to refine the shape of the model.[4] The use of silhouettes and laser striping is well suited to our smooth-surfaced, single-color clay models; however, systems that rely on surface-color variation [29] or uncalibrated systems that require a significant number

[4] Affordable 3D scanning systems that operate on the same principles as our laboratory system are now available commercially from Geometrix, Sanyo, and others.



(a) (b)

(c) (d)

Figure 4: (a) a model of a castle comprising 118 blocks, and (b) an interpreted rendering of it. The automatic enhancements in this graphical interpretation include the addition of turrets, roofs, windows, archways, a portcullis, and a flagpole in appropriate locations, as well as the selection of suitable surface properties and features for all the geometry. The 560-block model in (c)—a 12-inch ruler is included to show scale—was built as a challenging virtual environment for Quake II, the data format for which is another output option in our system. Applying the same interpretive style to this larger model to get the rendering in (d) requires changing only one numerical parameter indicative of building scale: it specifies the smallest number of blocks in the structure that can constitute a distinct architectural feature.

of point matches between surface features to be visible in the image sequence [11, 22] would likely encounter difficulty.

3.2 Graphical interpretation

The technical novelty in our system lies in our approach to model recognition and interpretation, both of which are accomplished by comparing a set of parameterized object templates to a scanned clay model. The templates are deformed to match the model, and the matching score determines how the model is classified.[5]

The templates are articulated linkages in which the links are truncated rectangular pyramids, or *beams*. A beam is completely defined by 10 numbers that specify the positions and dimensions of two parallel *base* rectangles. By requiring that the bases always be orthogonal to some major axis, a beam can be efficiently rasterized in three dimensions, which is important for the efficiency of the matching algorithm described below. A linkage is formed by connecting beams at the corners or centers of any of their six faces. Figure 6 shows representative templates for the 13 object categories currently recognized by our system. For each category there are 10 actual templates, created by hand, that differ only in the relative

[5] Brooks' ACRONYM system [5] is an early example of the use of parameterized models (generalized cylinders) for object recognition. Solina et al. [30] describe how to recover parametric models (superquadrics with global deformations) from range data by minimizing surface distances, which is similar to our maximization of volumetric overlap. Surveys of other related work on shape recovery and object recognition from image and range data can be found in more recent papers, e.g., [8, 9]. Finally, an application closely related to ours, shape-based indexing and retrieval of mechanical objects, is discussed in [6].

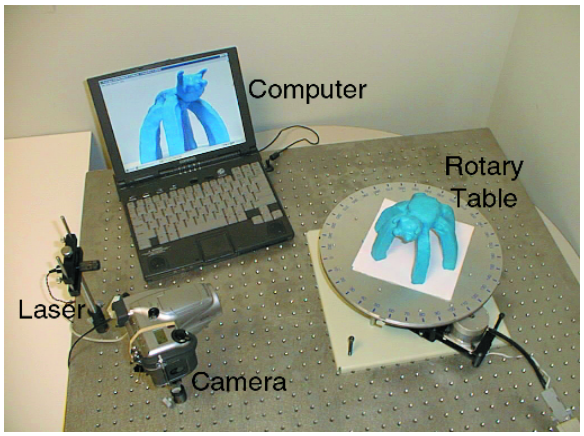


Figure 5: A low-cost scanning system.

proportions and orientations of their constituent beams: they provide multiple starting points for the matching algorithm, thereby reducing its chance of being confounded by matches that are locally optimal but globally inferior. A complete set of templates for a single category is shown in Figure 7.

Each biped template comprises six beams. Hence there would seem to be $6 \times 10 = 60$ parameters available for deforming a template. However, many of the parameters are constrained relative to each other. Some of these constraints derive implicitly from the enforced connectedness of beams in an articulated linkage. Others are the result of explicit programmer-specified constraints that apply to beam-size parameters, e.g., a constraint that the dimensions of the base rectangles for both arm beams be the same. When these constraints are applied, the number of free parameters for a biped template reduces to a more manageable 25. By modifying these parameters the computer can attempt to deform a template to best match a given clay model.

Quantifying the notion of a best match in an objective function is the essential problem in this optimization-based approach. Our objective function has terms for the following characteristics of a voxel-based match:

- *Superposition:* Each voxel occupied by both the rasterized object template and the scanned clay model contributes $+1$ to the objective-function score.

- *Excess:* This term penalizes voxels occupied by the rasterized template but not by the clay model. A simple approach would be to add a score of -1 for each such voxel. A much better idea is to add a score of $-r$, where r is the distance to the nearest occupied voxel in the clay model. This value can be computed for each voxel by an efficient two-pass algorithm [28]. The advantage of this distance-based penalty is that its gradient still gives useful information even when there is minimal overlap between the template and clay model.

- *Deformation:* Without some penalty for excessive deformation, templates can sometimes achieve excellent superposition and excess scores through absurd contortions. Deformation beyond a certain threshold is therefore penalized by an exponential function of the distance between the original and current parameter vectors.

Dividing the superposition and excess terms by the number of occupied voxels in the scanned volume normalizes for the volume of the clay model; dividing the deformation term by the number of beam vertices normalizes for the complexity of the object template.

Given this objective function, the matching algorithm is straightforward. First, the object template is adjusted for fit: the template is scaled in all three dimensions so that the beam endpoints lie just within the bounding box of the scanned model. (There is no need to normalize for orientation because we assume that scanned models have been placed upright and facing forward.) Matching is then just a matter of gradient descent using the negative of the objective function above. We use the conjugate-gradient method of gradient descent [26], approximating partial derivatives of the objective function by central differences. For best results we run the gradient-descent algorithm to quiescence three times in succession: first, we vary only the location parameters of the object template, then only the size parameters, and finally the location parameters again. We also schedule the relative weights of the objective-function terms over the three runs; the superposition and excess terms decrease in significance, and the deformation term increases. Good values for the weights were determined empirically for a small subset of the clay models, and then applied uniformly to all the models in our experiments. Matching is performed against a total of 130 object templates, 10 from each of the 13 categories shown in Figure 6.

Figure 8 shows the 16 clay models on which we tested our system. Each model is from one of the 13 object categories listed in Figure 6, with some duplication. These categories were based on the objects that figured most often in an informal survey of children's drawings. The artists who created the models worked independently of the programmer who fashioned the object templates, so that sometimes there are significant differences in the artists' and programmer's conceptions of a modeled object, e.g., compare the clay model of the Insect in Figure 8 with the corresponding object template in Figure 6. Each volumetric scan of a clay model was computed from 180 images, taken a uniform 2° apart. An additional 180 images were taken with the laser stripe on, though this additional data improved the scan significantly for only one of the reported models (the indented windows and door of the house were found). The scanned volumes were subsampled to a resolution of $128 \times 128 \times 128$ voxels for the purposes of matching.

Table 1 lists the top two matches for each clay model; Figure 9 illustrates the best matches graphically. Matching a single clay model against all 130 object templates took an average of 85 minutes on a 200 MHz Pentium Pro PC, and required rasterizing about 100,000 object templates; the bulk of the time was spent in the rasterization step. The top match was correct for 14 of the 16 clay models.

An examination of the two matching errors was instructive. Although the Insect template deformed to cover the Insect model almost perfectly, the degree of deformation was sufficient to result in the Quadruped and Chair templates receiving better matching scores. Reducing the deformation penalty would cause the Insect template to match best, but would also cause many incorrect best matches for the other clay models. The failure of the Car template to be the best match for the Car #1 model is due to a limitation of our modeling language for articulated linkages: one beam can attach to another only at the four corners or center of one of its faces. The offset of Car #1's wheels are such that the Car template cannot deform to cover them very well with its wheel beams constrained to attach as they do.

When the best-matching template has been found for a given clay model, an interpretation step parses the model into its constituent parts. For example, if a model is recognized as a biped, the match between the clay model and the deformed biped template is used to identify the model voxels that constitute the head, arms, legs, and torso. This voxel classification is based on the shortest distance from each voxel to each beam through clay-occupied space. Voxels are then assigned to their closest beams, with ties broken by distance to the beams' center axes. Once the best match is known, parsing takes about a minute. The 14 correctly matched models were all parsed acceptably well. Sample parses are shown

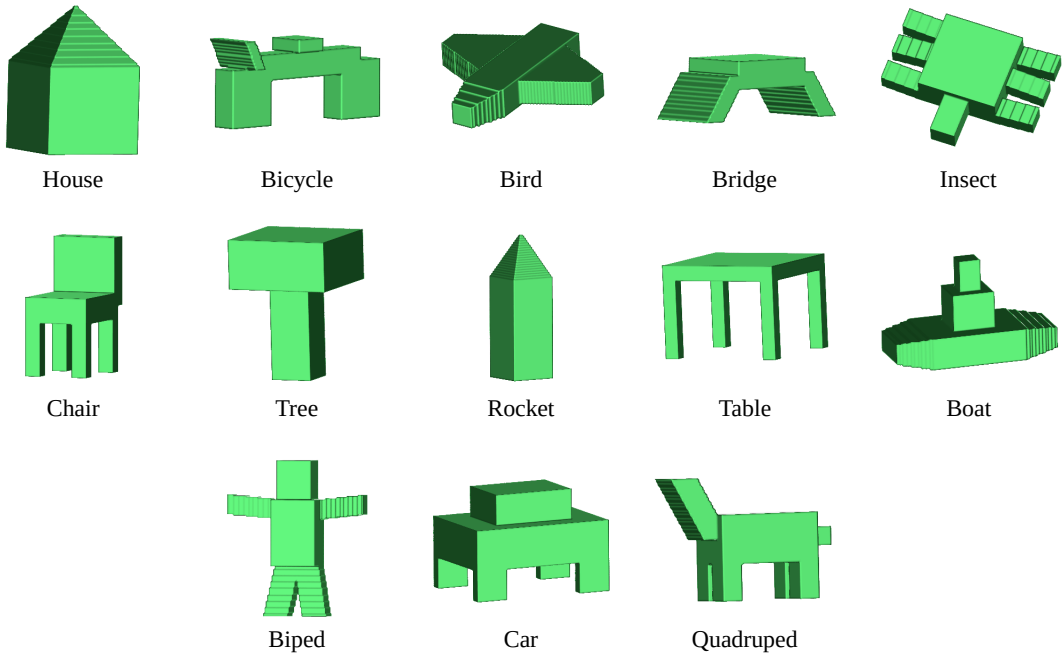


Figure 6: Representative templates from 13 categories of toy-like objects.

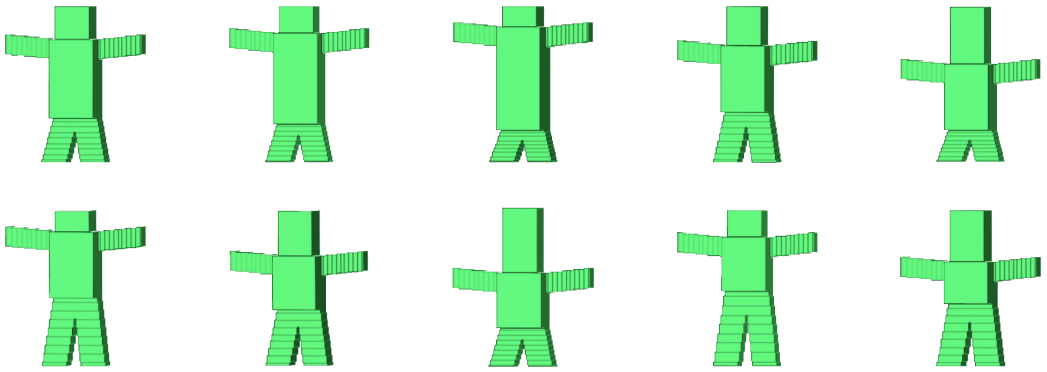


Figure 7: The 10 templates for the Biped category.

in Figure 2(d) and Figure 10.

With this information we can bring a clay model to (virtual) life. We did this automatically for the quadruped shown in Figure 2. The body measurements, masses, and moments of inertia were computed from the parse of the clay model. These values were passed as input to a control and simulation system for a four-legged robot, which adapted an existing control system to the dynamics of this particular clay model [20, 27]. The motion data computed by the simulation were then used to animate the object template, which in turn was used to animate the scanned volume by moving its voxels in rough concert with the beams of the template to which they were assigned in the parsing phase. However, care must be taken to avoid the introduction of tears and cracks in the model as it deforms. Such unwanted artifacts will appear if each voxel maintains position relative to just its associated beam. It is better to have all beams influence the movement of a voxel in inverse proportion to the square of their distance from the voxel; this reduces tears at the junctions of different model regions. (See Figure 2 and the companion videotape.) Related work on animating volumetric objects is described in [17].

An alternative and more general way to bring these models to life is with keyframes specified by the user. Commercial animation packages are notoriously complex because of the large number of features that must be provided. However, these clay models have been parsed by the computer and, therefore, the internal skeletal structure and kinematics are already known. The skeleton should allow the construction of an intuitive animation interface for this specific character. The user need only specify the motion of the skeleton because the internal details of the motion of the clay can be computed automatically using heuristic algorithms as was done for the quadruped or a more general physical simulation of clay [32]. Examples of this approach are shown on the companion video.

4 Conclusions and future work

In our case studies we have investigated the combination of tangible modeling and graphical interpretation. Tangible modeling

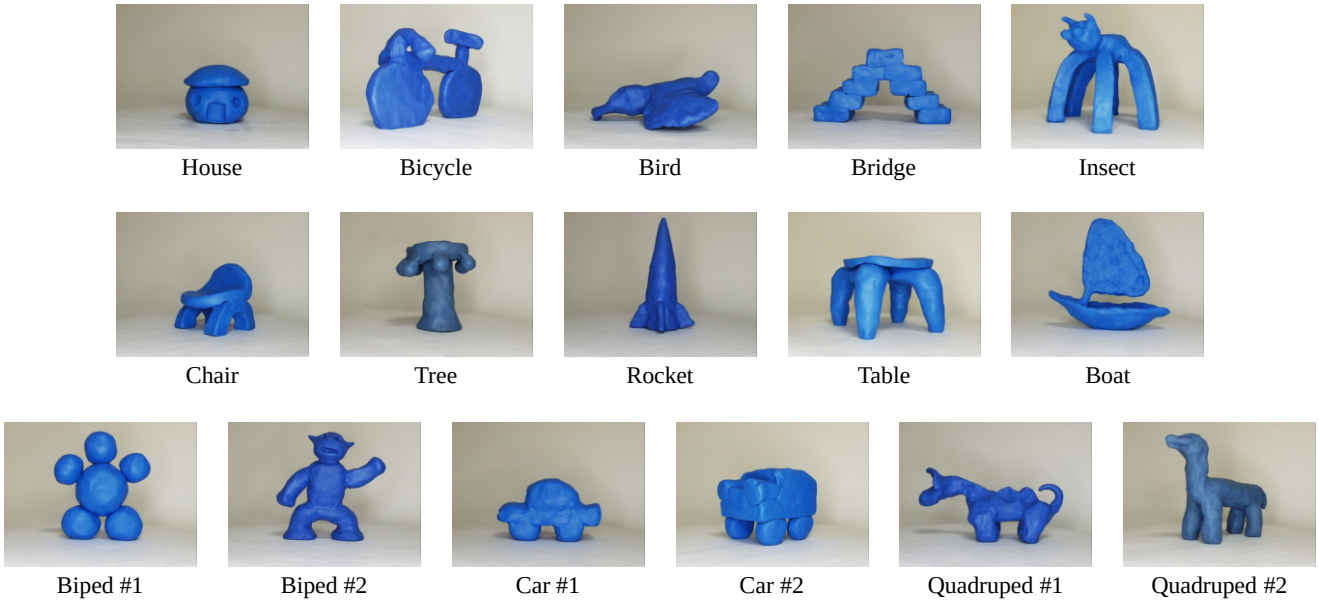


Figure 8: Examples from the image sequences for the 16 clay models captured by the camera illustrated in Figure 5.

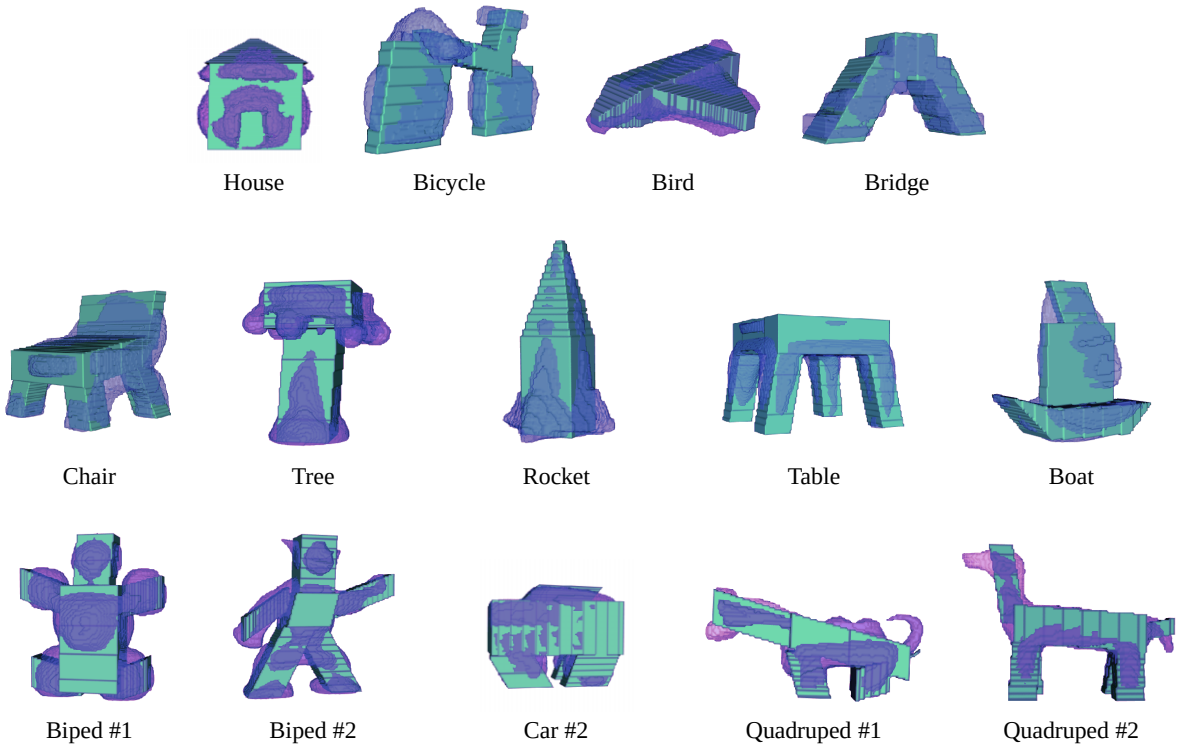


Figure 9: Best matches illustrated: deformed object templates superimposed on the scanned volumetric models.

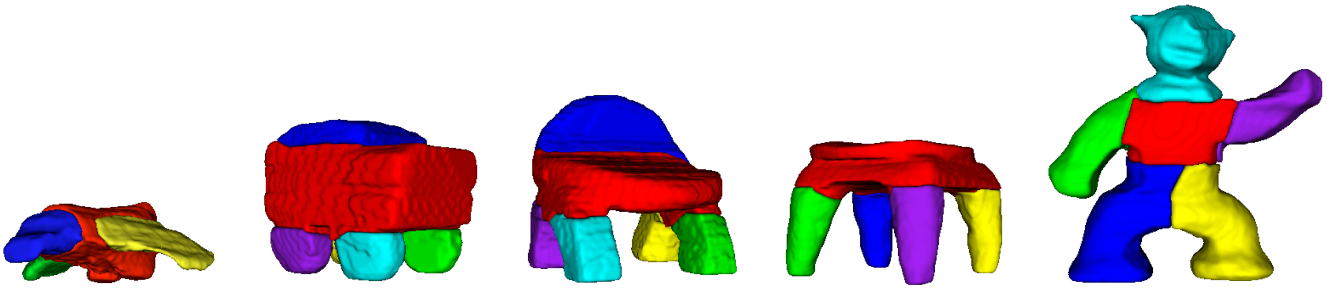


Figure 10: Sample parses of five clay models. The colored regions correspond to specific beams in the corresponding best-matching object template.

Clay Models	***Object Templates Ranked by Score***	
	First	Second
House	House(105)	Car(77)
Bicycle	Bicycle(87)	Boat(47)
Bird	Bird(79)	Insect(76)
Bridge	Bridge(115)	Insect(104)
Insect	*Q'ped(54)*	*Chair(50)*
Chair	Chair(78)	Bird(56)
Tree	Tree(82)	Biped(71)
Rocket	Rocket(89)	Q'ped(87)
Table	Table(107)	Car(102)
Boat	Boat(92)	Bicycle(31)
Biped #1	Biped(82)	Q'ped(67)
Biped #2	Biped(98)	Q'ped(76)
Car #1	*Boat(105)*	*Car(80)*
Car #2	Car(115)	Chair(92)
Q'ped #1	Q'ped(86)	Insect(55)
Q'ped #2	Q'ped(113)	Bicycle(71)

Table 1: The best-matching object templates for all the clay models, along with the matching scores. Of the 16 models, 14 were matched correctly; the entries for the two erroneous matches are shown in italics.

makes it easy for the user to create simple geometry quickly; the computer performs the tedious detailing tasks via graphical interpretation. Combined, these two ideas make for a new, more accessible approach to 3D modeling. As with recent sketch-based systems [34, 21], some generality is sacrificed in return for dramatically simpler interfaces that are accessible to unskilled and untrained users.

However, the modeling systems we have developed are only research prototypes. To make a truly useful and affordable system, more investigation is required. Our current work and future plans include:

- *Alternative architectures for embedded computation:* The most significant practical problem with our computational building blocks is that of power supply: the 560-block model in Figure 4(c) required a peak current of 8 amps at 13.8 volts to determine its geometry and to illustrate algorithmic progress via the blocks' LEDs. To reduce the power requirements (and thereby the cost) of the blocks, we are looking at ways of capturing a block structure using only a bare minimum of active components in each block. We are also considering the design of embedded computation architectures that make use of a broadcast medium. And to realize a more interactive experience, we are exploring hardware and software modifications that allow for interactive, incremental adjustment of the physical models.

- *Applications ancillary to geometric modeling:* We have built blocks with LEDs, speakers, switches, and motion sensors that support "world-in-miniature" interaction metaphors for virtual environments in which our miniature worlds are physical rather than virtual [25]. We are also exploring game applications that make use of these same sensors and actuators.

- *Interactive embodiments of graphical interpretation:* In our case studies, we focused on fully automatic graphical interpretation with the goal of understanding the limits of such an approach. We plan to study semi-automatic systems in which the computer plays a more assistive role. For example, it would be straightforward for our interpretation systems to prompt the user for help with ambiguous interpretations (e.g., "Is this part of the wall or roof?" or "Is this a quadruped or a chair?") or with a choice of enhancement (e.g., "Do you want a portcullis or a drawbridge here?" or "Do you want the quadruped to run or gallop?"). This kind of user input would do much to address problems of brittleness and speed in our prototypes.

  More interestingly, we could develop mixed-initiative systems that make more use of the fully automatic algorithms we have developed. For example, a mixed-initiative system might begin by prompting the user for all stylistic and aesthetic choices, but then begin making suggestions that are consistent with the user's previous selections. Our case studies show that these kinds of systems can be built, and hold promise to enable a new paradigm of computer-assisted graphical-interpretation applications.

5 Acknowledgments

Special thanks to Christopher Marks, whose playful blending of the real and virtual inspired this work. We also want to thank Emily Anderson, Lynn Anderson, Bob Bell, Dirk Brinkman, Audrey Hodgins, Clifton Leigh, Neil McKenzie, Egon Pasztor, Hanspeter Pfister, Rusty Ventura, Dick Waters, Jonathan Wolff, and Bill Yerazunis for their last-minute help, encouragement, and advice, and Altec Plastics, Flextronics, and Switchcraft for satisfying our unusual requirements. And thanks to the anonymous SIGGRAPH reviewers for helpful suggestions and comments.

References

[1] R. Aish. 3D input for CAAD systems. *Computer-Aided Design*, 11(2):66–70, Mar. 1979.

[2] R. Aish and P. Noakes. Architecture without numbers – CAAD based on a 3D modelling system. *Computer-Aided Design*, 16(6):321–328, Nov. 1984.

[3] G. Anagnostou, D. Dewey, and A. Patera. Geometry-defining processors for engineering design and analysis. *The Visual Computer*, 5:304–315, 1989.

[4] B. G. Baumgart. Geometric modeling for computer vision. Technical Report AIM-249, AI Laboratory, Stanford Univ., Oct. 1974.

[5] R. Brooks. Model-based 3D interpretations of 2D images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 5(2):140–150, 1983.

[6] G. Cybenko, A. Bhasin, and K. D. Cohen. Pattern recognition of 3D CAD objects: Towards an electronic Yellow Pages of mechanical parts. *Smart Engineering Systems Design*, 1:1–13, 1997.

[7] D. Dewey and A. Patera. Geometry-defining processors for partial differential equations. In B. Alder, editor, *Special Purpose Computers*, pages 67–96. Academic Press, 1988.

[8] S. Dickinson, A. Pentland, and A. Rosenfeld. From volumes to views: An approach to 3D object recognition. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 55(2):130–154, 1992.

[9] S. J. Dickinson and D. Metaxas. Integrating qualitative and quantitative shape recovery. *Intl. Journal of Computer Vision*, 13(3):311–330, 1994.

[10] C. Esposito, W. B. Paley, and J. Ong. Of mice and monkeys: A specialized input device for virtual body animation. In *Proc. of Symposium on Interactive 3D Graphics*, pages 109–114, 213, Monterey, California, Apr. 1995.

[11] A. W. Fitzgibbon, G. Cross, and A. Zisserman. Automatic 3D model construction for turn-table sequences. In *Proc. of SMILE Workshop*, Freiburg, Germany, June 1998.

[12] J. Frazer. Use of simplified three-dimensional computer input devices to encourage public participation in design. In *Proc. of Computer Aided Design 82*, pages 143–151. Butterworth Scientific, 1982.

[13] J. Frazer. *An Evolutionary Architecture*. Architectural Association, London, 1994. Describes several tangible modelers developed by Frazer's group from 1979 onwards.

[14] J. Frazer, P. Coates, and J. Frazer. Software and hardware approaches to improving the man-machine interface. In *Proc. of the First International IFIP Conf. on Computer Applications in Production and Engineering*, pages 1083–94, Amsterdam, Holland, 1983. North Holland.

[15] J. Frazer, J. Frazer, and P. Frazer. Intelligent physical three-dimensional modelling system. In *Proc. of Computer Graphics 80*, pages 359–370. Online Publications, 1980.

[16] J. Frazer, J. Frazer, and P. Frazer. New developments in intelligent modelling. In *Proc. of Computer Graphics 81*, pages 139–154. Online Publications, 1981.

[17] N. Gagvani, D. Kenchammana-HoseKote, and D. Silver. Volume animation using the skeleton tree. In *Proc. of the IEEE Symposium on Volume Visualization*, pages 47–53, Research Triangle Park, NC, Oct. 1998.

[18] M. G. Gorbet and M. Orth. Triangles: Design of a physical/digital construction kit. In *Proc. of DIS 97*, pages 125–128, Amsterdam, Holland, Mar. 1997. ACM.

[19] M. G. Gorbet, M. Orth, and H. Ishii. Triangles: Tangible interface for manipulation and exploration of digital information topography. In *Proc. of CHI 98*, pages 49–56, Los Angeles, California, Apr. 1998. ACM.

[20] J. K. Hodgins and N. S. Pollard. Adapting simulated behaviors for new characters. In *Proc. of SIGGRAPH 97*, pages 153–162, Los Angeles, California, Aug. 1997.

[21] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. In *Proc. of SIGGRAPH 99*, pages 409–416, Los Angeles, California, Aug. 1999.

[22] S. B. Kang. Quasi-euclidean recovery from unknown but complete orbital motion. Technical Report TR 97-10, Compaq CRL, 1997.

[23] F. Martin and R. Borovoy. The active LEGO baseplate project. `http://fredm.www.media.mit.edu/people/fredm/projects/ab/`, 1994.

[24] Ovid. Metamorphoses:X. Rome, 1 AD.

[25] R. Pausch, T. Burnette, D. Brockway, and M. E. Weiblen. Navigation and locomotion in virtual worlds via flight into hand-held miniatures. In *Proc. of SIGGRAPH 95*, pages 399–400, Los Angeles, California, Aug. 1995.

[26] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[27] M. H. Raibert. *Legged Robots That Balance*. MIT Press, Cambridge, 1986.

[28] J. C. Russ. *The Image Processing Handbook*. CRC Press, 1998.

[29] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173, 1999.

[30] F. Solina and R. Bajcsy. Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(2):131–146, 1990.

[31] H. Suzuki and H. Kato. AlgoBlock: A tangible programming language — a tool for collaborative learning. In *Proc. of the 4th European Logo Conference*, pages 297–303, 1993.

[32] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, December 1988.

[33] P. H. Winston. Learning structural descriptions from examples. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[34] R. C. Zeleznik, K. Herndon, and J. F. Hughes. SKETCH: An interface for sketching 3D scenes. In *Proc. of SIGGRAPH 96*, pages 163–170, New Orleans, Louisiana, Aug. 1996.