generally perceptible. Any "chattering" induced by sharp surface features may be dealt with by either increasing the sponginess of the surface, or increasing the size of the virtual surface in the user's hand space, thereby effectively increasing the spatial sampling rate for the same hand movement velocity.

The assumptions made about reasonable and expected hand movements could easily be enforced by the addition of velocity dependent forces to restrain motion to a reasonable speed. Users have been surprisingly adept, however, at tuning their hand gestures to give the maximum sensitivity to surface features, and so a need for such restrictions has not yet been seen.

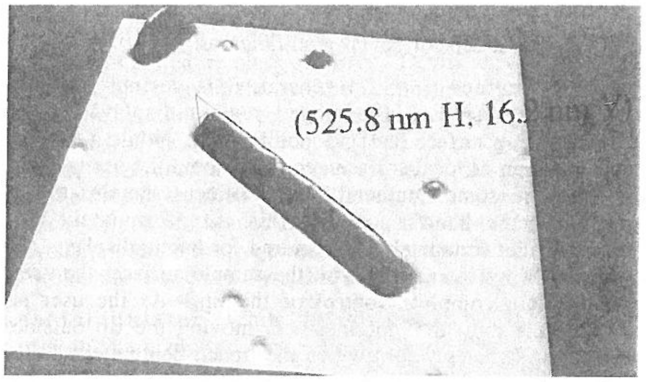


Figure 1 - Measuring a sample of TMV. Height in 3D space is exaggerated by a factor of 5.

4. TOOLS

4.1 DISPLAY TOOLS

The NM has inherited the standard set of virtual-reality (VR) tools from the UNC *vlib*, such as grabbing, scaling, and flying [3]. In addition, tools are added as their desirability becomes apparent during use of the system. When used immersively, fixed lighting sources have proved sufficient, as the user's head position relative to the surface and light determines specular highlighting. By moving about in the scene, the optimal angle for viewing features of interest can be found. While working in groups, however, it frequently proves advantageous to fix a single hypothetical user's position in space, and display to a projection screen a single view which all user's share. Transition from fixed view to head tracked may be performed on the fly for investigation of specific features, the subtleties of which are often more easily discerned in the immersive mode despite the lower resolution display. While in fixed view, it is helpful to adjust the lighting source to bring out specific details. A virtual pointer is supplied to allow the user to point to the directional light source. The lighting of the scene is updated as the user moves the pointer until illumination becomes optimal.

4.2 MEASURING TOOLS

Quantitative tools are essential for full understanding of the data. Often, features are distinguishable only by their absolute size. The user may create a measuring rectangle perpendicular to the horizontal plane by selecting two points, such as at the base and peak of a feature. The rectangle is displayed with height and width in nanometers, as well as a profile of the surface intersecting the rectangle. This display may be independently positioned by the user, and persists until being explicitly dismissed, giving a reference scale for the rest of the image. Since the horizontal shape of features displayed is the convolution of features on the surface with the probe tip, which has a typical radius of curvature of 30 to 50 nm, features tend to appear flattened. This appearance may be corrected by vertically stretching the measurement rectangle until the profile of a reference feature takes the correct shape (e.g. a colloidal ball has same height as width). The height of the rest of the scene is then scaled accordingly.

4.3 VCR TOOLS

While the NM is primarily a real-time data visualization system, it is also valuable for off-line analysis. Snapshots of the scene may be saved to disk at any time. Additionally, the stream of data returning from the microscope is saved and may be replayed interactively, with all tools available except those involving modification, which naturally require an actual surface and microscope. Standard VCR functions are supported, such as control over replay rate, fast forward, rewind, and absolute positioning in the stream. These afford quick review of selected segments within a stream which may be quite large, having been acquired over the span of up to an hour. The viewpoint and vertical scale can be different in the replay than in the original experiment, as they do not depend on the surface data.

The NM is not limited to data collected within the interface. Simple file format conversion routines have been written to allow the importation of data collected elsewhere and by microscopes other than the Digital Instruments Nanoscope III currently used in the system. Data received from the UCLA materials science group is investigated using the interface as a 3-D viewer, and video tapes returned to the group of "walk-throughs" of the surface under study. As a visualization tool alone the interface has proven worthwhile in the understanding of complex surface features.

4.4 MODIFICATION TOOLS

A set of physical knobs on the ARM control the sponginess of the surface and the forces applied by the tip to the sample. That the perceived hardness of the surface determines sensitivity to smaller details is straightforward. The implications of tip force on haptic response is more subtle. If the force applied by the microscope is too great, a feature will be displaced immediately, and will never be felt. If the force is too small, the feature will be felt, but no modification made. The force necessary to modify the surface is determined by factors such as the exact tip shape, the direction of the force, humidity, and surface contaminants, and may vary widely across a given sample and over periods of time as short as tens of minutes. Without knowing a priori the force required, the user must have immediate control over the forces applied. The interface allows the user to control the position of the tip and feel the surface with one hand, while the other hand adjusts physical knobs controlling the force level. The microscope may be toggled between non-damaging and modification modes with a thumb switch, to allow exact positioning of the tip by feel before the application of forces to features.

To supplement the modification mode's immediate haptic display, after a modification event a small area around the event is scanned and updated. This area is generally of greatest interest and most likely to be out of date, and is refreshed in about a hundredth of the time needed to rescan the entire surface. After quickly updating that subset of the grid, imaging of the full selected region resumes.

Area Sweep Since the forces applied by the tip are always under immediate user control, the entire area being scanned may be swept out simply by increasing the force until all materials are removed as desired. This is the interactive method most commonly supported by commercial AFMs. While an efficient way to clear a region, it has several disadvantages. This method is inappropriate for selective removal of material within a rectangle. The force required to move material in one area may be enough to damage the substrate or other desired features nearby. Moreover, the clearing may be incomplete, with ragged edges around the border or debris left in the region which must then be cleaned out.

Line Tool The etching of circuits from a conducting film on non-conducting substrate frequently requires straight lines connecting cleared regions or isolating conducting regions. The user may select any two endpoints of a line segment, and have the tip scratch between the two points at a preset modification force.

Engrave Tool Many commercial microscopes support lithography techniques, allowing the user to preset a path to be traced by the tip at a specific force. These afford efficient etching of an exact known outline into a surface, but leave the same jagged edges and debris as the area sweep. Cleaning these edges is easily performed using the engraving tool, in which the tip tracks the hand exactly over the surface. The effect is like the user having an ice pick with which to feel the surface, scratch it, and push about materials on it. (Depending on the tip radius relative to surface features, it may be a very blunt ice pick.) This gives the finest degree of control available with the microscope.

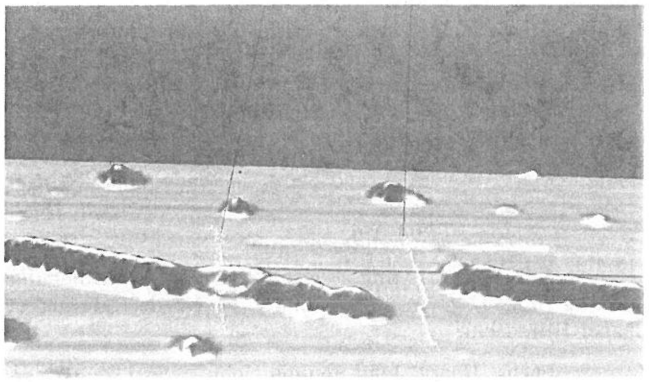


Figure 2 - A segment of TMV is separated using the sweep tool. The two black lines extending upward toward the hand (not shown) define the flat edge of the virtual broom. The two parallel lines of white markers indicate the path having been swept out. The image has not yet been updated to show the removal of the segment.

Sweep Tool As can easily be imagined, pushing materials about with an ice-pick might sometimes be less than convenient. Often, a different instrument is more appropriate. While there is only one physical tip, control of its motion can simulate other, virtual tools. A virtual whisk broom is provided for selective clearing of regions and manipulation of larger objects, or even small objects which are to be swept in a general direction, and then positioned precisely using the engrave tool. In sweep mode, the tip oscillates between the tracked position of the user's hand and a point determined by the orientation of the hand (figure 2). The magnitude and direction of the oscillation is therefore immediately and intuitively determined by the user, giving the illusion of an extended tip, the flat edge of which may be used to scrape out selected areas or push objects. This complements the area sweep mode in that, while it lacks the precise rectangular boundary of area sweep, it is also not limited to any rectangle. The "edge" may become wide or narrow, and change orientation relative to the surface plane as required to navigate though features which must be left undisturbed.

5. RESULTS

5.1 BALL PUSHING

The manipulation of colloidal gold particles has proved an excellent test-bed for the interface, in addition to being a worthwhile pursuit in its own right. Controlled movement of the balls would enable the performance of experiments determining physical properties and materials characteristics which are currently only predicted by theory [1]. Balls are typically deposited randomly about a surface. Isolation and precise positioning of individual balls, either into patterns or within other structures is difficult, if not impossible using means available with commercial microscopes. The interaction between balls and the microscope tip is unpredictable. At the same time, the balls are rigid enough to easily be felt with the NM's haptic display, and image clearly.

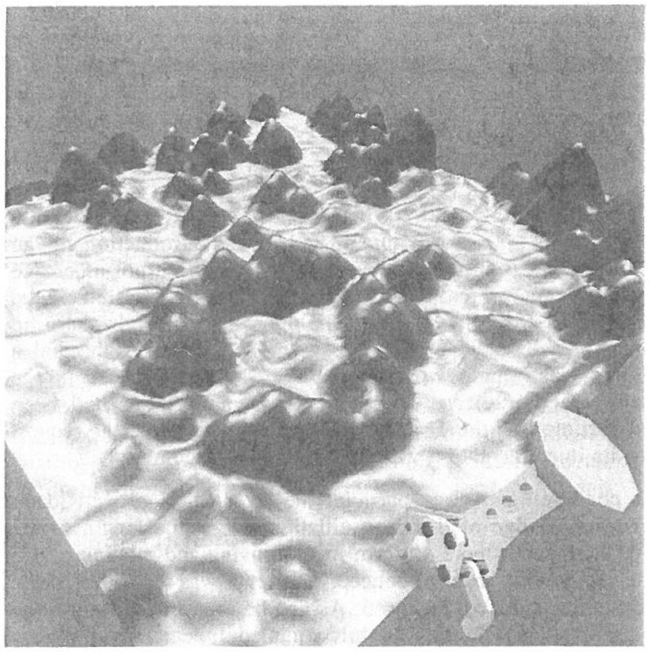


Figure 3 - Colloidal gold balls arranged in a ring. The hand icon is front right.

In one experiment, a thin gold wire (~50 nm wide) was etched into a 15 nm thick gold film on mica substrate using standard AFM lithography techniques. A gap approximately 100 nm wide was then cut into the wire, and colloidal gold balls of 15 nm diameter distributed over the surface. The user was then able to select a ball and maneuver it through the other particles in the area and position it in the gap, without disturbing other material in the region, or damaging the wire. By using a light force in engrave mode, the user could feel the ball on the edge of the tip, and so could follow the ball closely and detect when the ball took an erratic jump, quickly compensating in the direction of pushing, or waiting for the image to be updated to relocate the ball. Fig. 4 shows the trace of the pushing events and the final pushes of the ball into the gap. The entire sequence was performed in a matter of minutes. It is not clear how or even if the same result could be accomplished using methods currently available with commercial AFMs. The experiment provides a convincing proof of concept for the manipulation of a colloid into a gap in a wire which is connected to macroscopic leads for the electrical characterization of the particle. Such a circuit is currently being fabricated at UNC-CH, and characterization experiments are expected in the coming weeks.

Colloidal gold balls have also been arranged into structures such as a ring and a matrix. The ability to arrange the balls into specific patterns is useful both in the fabrication of circuits from the balls, and comparison with theoretic refraction patterns in near field spectroscopy studies. Work is also currently underway to position balls in arrangements for which theoretic predictions of refraction patterns exist.

5.2 VIRUS MANIPULATIONS

The positioning of a virus in a circuit as described above would offer a unique ability to characterize the electrical properties of the virus. Manipulation of the virus is even more challenging than the gold balls, however. Samples of Tobacco Mosaic Virus (TMV) were distributed over a mica substrate. In pushing it with the engrave tool, the TMV was found to be very easy to bend and break. The tip could also be positioned on the TMV and the force turned up slowly until the tip ruptured the virus, an event which could be easily felt by the user. But while the dissection of TMV particles was interesting in its own right, moving a particle as a whole unit was also desirable.

User frustrations with trying to push an extended flexible object with a sharp instrument led to the introduction of the sweep tool. Intuitively, it would have been the tool of choice in an analogous real-world task. Building the illusion of the broad edge from the reality of the microscope's single sharp tip proved easier than coming up with the initial insight that a blunter instrument would sometimes be preferable. In the natural and intuitive environment in which user's had been interacting with the TMV, however, that insight and the request for its implementation were natural and forthcoming.

In positioning a virus particle, the sweep tool proved ideal. The broad edge of the tip oscillations along the length of the TMV applied a more uniform force, moving and rotating it as a unit. Again as proof of concept, a letter T was formed of TMV segments as shown in Fig. 5. As can be seen in the figure, the TMV particles obtain a slightly rumpled appearance after they have been moved. This indicates that, while we are moving the particles as units, we are not doing so without damage. We are investigating possible virtual tools that might be even gentler still, in the hopes of moving the particles while leaving them intact.

6. CONCLUSIONS

The NanoManipulator provides an intuitive interface hiding the details of performing complex tasks using an SPM. Surface features are more easily recognized with the combination of 3-D topography and haptic feedback in real time. Feeding the user's senses more fully allows faster development of manipulation skills. The collaborative nature of the project allows new tools to be developed as the needs of the users become more sophisticated. Many tasks performed using the NM are not well enough understood to be automated, so they require real time feedback to and response from the user. It is hoped that the NM will provide the insight into the manipulation process necessary to automate the fabrication of mesoscopic and nanometer-scale circuits. The NM is valuable now in the building of one-of-a-kind structures which will contribute significantly to the areas of materials science and solid state physics.

ACKNOWLEGEMENTS

We are grateful to the National Institutes of Health (grant #RR02170), the Defense Advanced Research Projects Agency (contract #DABT63-92-C-0048), the National Science Foundation (grant #IRI-9202424), and the Office of Naval Research for funding and support.

REFERENCES

1. Devoret, M. H., D. Esteve and C. Urbina, "Single-electron Transfer in Metallic Nanostructures", Nature **360**, 547 (1992).

2. Kastner, M. A., Reviews of Modern Physics, **64**, 849 (1992).

3. Robinett, Warren, and Richard Holloway, Implementation of Flying, Scaling, and Grabbing in Virtual Worlds. Proceedings of the ACM Symposium on Interactive 3D Graphics (Cambridge, MA, 1992), special issue of Computer Graphics, ACM SIGGRAPH, New York, 1992.

4. Taylor, Russell, Warren Robinett, Vernon L. Chi, Frederick P. Brooks, Jr., William V. Wright, R. Stanley Williams, and Erik J. Snyder, The Nanomanipulator: A Virtual-Reality Interface for a Scanning Tunneling Microscope. Proceedings of SIGGRAPH '93 (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, 1993, pp. 127-134.

5. Mark, Wiliam R. and Scott C. Randolph, "UNC-CH Force Feedback Library", UNC-CH Computer Science Dept. Technical Report #TR94-056, 1994.

6 Ouh-young, Ming. Force Display in Molecular Docking, *Ph. D. Thesis*, University of North Carolina at Chapel Hill, 1990.

7 Fuchs, Henry, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, and Laura Israel. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. Proceedings of SIGGRAPH '89. In Computer Graphics, 19 3 (1989). 79-88.

8 Stroscio, Joseph A. and D. M. Eigler, Atomic and Molecular Manipulation with the Scanning Tunneling Microscope. Science, **254** (1991). 1319-1326.

9 Sarid, Dror, Scanning Force Microscopy,*Oxford Series in Optical and Imaging Sciences*, Oxford University Press, NY 1991.

10 Chen, C. Julian, Introduction to Scanning Tunneling Microscopy,*Oxford Series in Optical and Imaging Sciences*, Oxford University Press, NY 1993.

Combatting Rendering Latency

Marc Olano, Jon Cohen, Mark Mine, Gary Bishop
Department of Computer Science, UNC Chapel Hill
{olano,cohenj,mine,gb}@cs.unc.edu

ABSTRACT

Latency or lag in an interactive graphics system is the delay between user input and displayed output. We have found latency and the apparent bobbing and swimming of objects that it produces to be a serious problem for head-mounted display (HMD) and augmented reality applications. At UNC, we have been investigating a number of ways to reduce latency; we present two of these. Slats is an experimental rendering system for our Pixel-Planes 5 graphics machine guaranteeing a constant single NTSC field of latency. This guaranteed response is especially important for predictive tracking. Just-in-time pixels is an attempt to compensate for rendering latency by rendering the pixels in a scanned display based on their position in the scan.

1 INTRODUCTION

1.1 What is latency?

Performance of current graphics systems is commonly measured in terms of the number of triangles rendered per second or in terms of the number of complete frames rendered per second. While these measures are useful, they don't tell the whole story.

Latency, which measures the start to finish time of an operation such as drawing a single image, is an often neglected measure of graphics performance. For some current modes of interaction, like manipulating a 3D object with a joystick, this measure of responsiveness may not be important. But for emerging modes of "natural" interaction, latency is a critical measure.

1.2 Why is it there?

All graphics systems must have some latency simply because it takes some time to compute an image. In addition, a system that can produce a new image every frame may (and often will) have more than one frame of latency. This is caused by the pipelining used to increase graphics performance. The classic problem with pipelining is that it provides increased throughput at a cost in latency. The computations required for a single frame are divided into stages and their execution is overlapped. This can expand the effective time available to work on that single frame since several frames are being computed at once. However, the latency is as long as the full time spent computing the frame in all of its stages.

1.3 Why is it bad?

Latency is a problem for head-mounted display (HMD) applications. The higher the total latency, the more the world seems to lag behind the user's head motions. The effect of this lag is a high viscosity world.

The effect of latency is even more noticeable with see-through HMDs. Such displays superimpose computer generated objects on the user's view of the physical world. The lag becomes obvious in this situation because the real world moves without lag, while the virtual objects shift in position during the lag time, catching up to their proper positions when the user stops moving. This "swimming" of the virtual objects not only detracts from the desired illusion of the objects' physical presence, but also hinders any effort to use this technology for real applications.

Most see-through HMD applications require a world without these "swimming" effects. If we hope to have applications present 3D instructions to guide the performance of "complex 3D tasks" [9], such as repairs to a photocopy machine or even a jet engine, the instructions must stay fixed to the machine in question. Current research into the use of see-through HMDs by obstetricians to visualize 3D ultrasound data indicates the need for lower latency visualization systems [3]. The use of see-through HMDs for assisting surgical procedures is unthinkable until we make significant advances in the area of low latency graphics systems.

2 COMBATTING LATENCY

2.1 Matching

A possible solution to this lag problem is to use video techniques to cause the user's view of the real world to lag in synchronization with the virtual world. However, this only works while the latency is relatively small.

2.2 Prediction

Another solution to the latency problem is to predict where the user's head will be when the image is finally displayed [10, 1, 2]. This technique, called predictive tracking, involves using both recent tracking data and accurate knowledge of the system's total latency to make a best guess at the position and orientation of the user's head when the image is displayed inside the HMD. Azuma states that for prediction to work effectively, the lag must be small and consistent. In fact he uses the single field-time latency rendering system (Slats), which we will discuss shortly, to achieve accurate prediction.

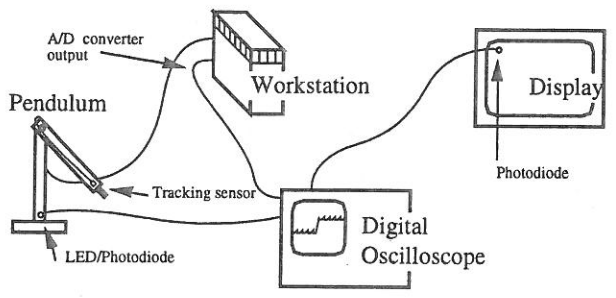


Figure 1: Apparatus for external measurement of tracking and display latency.

2.3 Rendering latency: compensation and reduction

2.3.1 Range of solutions

There are a wide spectrum of approaches that can be used to reduce lag in image generation or compensate for it. One way to compensate for image generation latency is to offset the display of the computed image based upon the latest available tracking data.

This technique is used, for example, by the Visual Display Research Tool (VDRT), a flight simulator developed at the Naval Training Systems Center in Orlando, Florida [5, 6]. VDRT is a helmet-mounted laser projection system which projects images onto a retro-reflective dome (instead of using the conventional mosaic of high resolution displays found in most flight simulators). In the VDRT system, images are first computed based upon the predicted position of the user's head at the time of image display. Immediately prior to image readout, the most recently available tracking data is used to compute the errors in the predicted head position used to generate the image. These errors are then used to offset the raster of the laser projector in pitch and yaw so that the image is projected at the angle for which it was computed. Rate signals are also calculated and are used to develop a time dependent correction signal which helps keep the projected image at the correct spatial orientation as the projector moves during the display field period.

Similarly, Regan and Pose are building the prototype for a hardware architecture called the address recalculation pipeline[15]. This system achieves a very small latency for head rotations by rendering a scene on the six faces of a cube. As a pixel is needed for display, appropriate memory locations from the rendered cube faces are read. A head rotation simply alters which memory is accessed, and thus contributes nothing to the latency. Head translation is handled by object-space subdivision and image composition. Objects are prioritized and re-rendered as necessary to accommodate translations of the user's head. The image may not always be correct if the rendering hardware cannot keep up, but the most important objects, which include the closest ones, should be rendered in time to keep their positions accurate.

Since pipelining can be a huge source of lag, latency can be reduced by reducing pipelining or basing it on smaller units of time like polygons or pixels instead of frames. Most commercial graphics systems are at least polygon pipelined. Whatever level the pipelining, a system that computes images frame by frame is by necessity saddled with at least a frame time of latency. Other methods overcome this by divorcing the image generation from the display update rate.

Frameless rendering[4] can be used to reduce latency in this way. In this technique pixels are updated continuously in a random pattern. This removes the dependence on frames and fields.

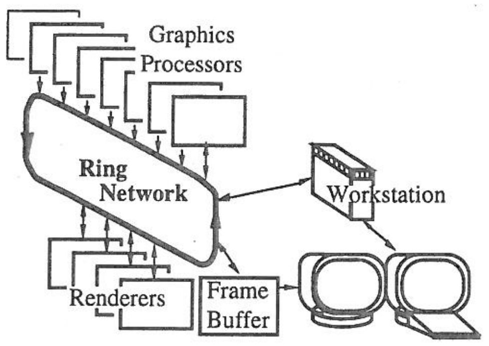


Figure 2: Pixel-Planes 5 system architecture

Pixels may be transformed at whatever rate is most convenient. This reduces latency at the cost of image clarity since only a portion of the pixels are updated. The transform rate can remain locked to the tracker update rate or separated on a pixel-by-pixel basis as with the just-in-time pixels method, discussed next.

2.3.2 Just-in-time pixels (JITP)

We will present a technique called just-in-time pixels, which deals with the placement of pixels on a scan-line display as a problem of temporal aliasing [14]. Although the display may take many milliseconds to refresh, the image we see on the display typically represents only a single instant in time. When we see an object in motion on the display, it appears distorted because we see the higher scan lines before we see the lower ones, making it seem as if the lower part of the object lags behind the upper part. Avoidance of this distortion entails generating every pixel the way it should appear at the exact time of its display. This can lead to a reduction in latency since neither the head position data, nor the output pixels are limited to increments of an entire frame time. This idea is of limited usefulness on current LCD HMDs with their sluggish response. However, it works quite well on the miniature CRT HMDs currently available and is also applicable to non-interactive video applications.

2.3.3 Slats

As a more conventional attack on latency, we have designed a rendering pipeline called Slats as a testbed for exploring fixed and low latency rendering [7]. Unlike just-in-time pixels, Slats still uses the single transform per frame paradigm. The rendering latency of Slats is exactly one field time (16.7 ms). This is perfect for predictive tracking which requires low and *predictable* latency. We measure this rendering latency from the time Slats begins transforming the data set into screen coordinates to the time the display devices begin to scan the pixel colors from the frame buffers onto the screens.

3 MEASURING LATENCY

We have made both external and internal measurements of the latency of the Pixel-Planes 5 PPHIGS graphics library [13, 7]. These have shown the image generation latency to be between 54 and 57 ms for minimal data sets. The internal measurement methods are quite specific to the PPHIGS library. However, the external measurements can be taken for any graphics system.

The external latency measurement apparatus records three timing signals on a digital oscilloscope (see figure 1). A pendulum and led/photodiode pair provide the reference time for a real-world event — the low point of the pendulum's arc. A tracker on the pendulum is fed into the graphics system. The graphics system

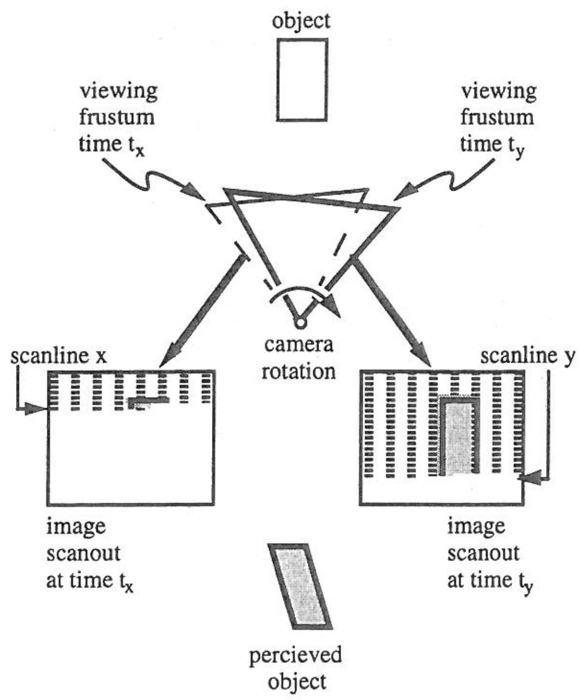

object
viewing frustum time t_x
viewing frustum time t_y
scanline x
camera rotation
scanline y
image scanout at time t_x
image scanout at time t_y
percieved object

Figure 3: Image generation in conventional computer graphics animation. Scanline x is displayed at time t_x, scanline y is displayed at time t_y.

starts a new frame when it detects the pendulum's low point from the tracking data. An D/A converter is used to tell the oscilloscope when the new frame has started. Frames alternate dark and light and a photodiode attached to the screen is used to tell when the image changes. The tracking latency was the time between the signal from the pendulum's photodiode and the rendering start signal out of the D/A converter. The rendering latency was the time between the signal out of the D/A converter and the signal from the photodiode attached to the screen. These time stamps were averaged over a number of frames.

The internal measurements found the same range of rendering latencies. The test was set up to be as fair as possible given the Pixel-Planes 5 architecture (figure 2, explained in more detail later). The test involved one full screen triangle for each graphics processor. This ensured that every graphics processor would have work to do and would have rendering instructions to send to every renderer. The first several frames were discarded to make sure the pipeline was full. Finally, latency determined from time stamps on the graphics processors was averaged over a number of frames.

4 JUST-IN-TIME PIXELS

4.1 The idea

When using a raster display device, the pixels that make up an image are not displayed all at once but are spread out over time. In a conventional graphics system generating NTSC video, for example, the pixels at the bottom of the screen are displayed almost 17 ms after those at the top. Matters are further aggravated when using NTSC video by the fact that not all of the lines of an NTSC image are displayed in one raster scan but are in fact interlaced across two fields. In the first field only the odd lines in an image are displayed, and in the second field only the even.

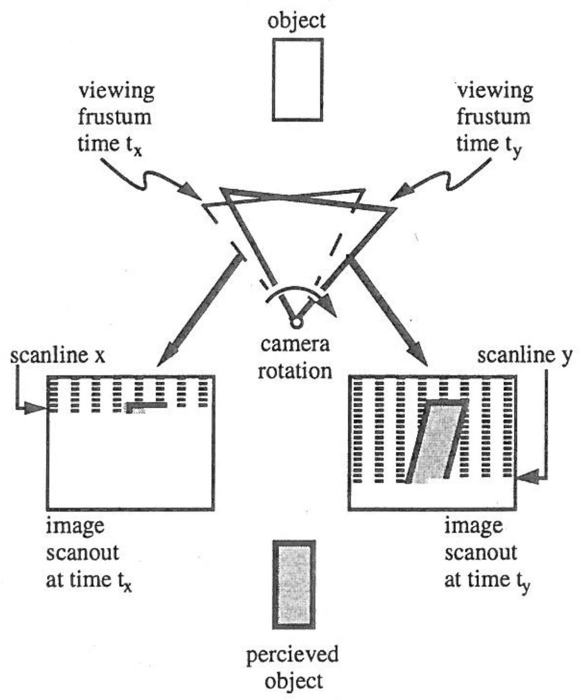

object
viewing frustum time t_x
viewing frustum time t_y
scanline x
camera rotation
scanline y
image scanout at time t_x
image scanout at time t_y
percieved object

Figure 4: Image generation using just-in-time pixels

Thus, unless animation is performed on fields (i.e. generating a separate image for each field), the last pixel in an image is displayed more than 33 ms after the first. The problem with this sequential readout of image data, is that it is not reflected in the manner in which the image is computed.

Typically, in conventional computer graphics animation, only a single viewing transform is used in generating the image data for an entire frame. Each frame represents a point sample in time which is inconsistent with the way in which it is displayed. As a result, as shown in figure 3 and plate 1, the image does not truly reflect the position of objects (relative to the view point of the camera) at the time of display of each pixel.

A quick "back of the envelope" calculation can demonstrate the magnitude of the errors that result if this display system delay is ignored. Assuming, for example, a camera rotation of 200 degrees/second (a reasonable value when compared with peak velocities of 370 degrees/second during typical head motion - see [12]) we find:

Assume:
1) 200 degrees/sec camera rotation
2) camera generating a 60 degree Field of View (FOV) image
3) NTSC video
   60 fields/sec NTSC video
   ~600 pixels/FOV horizontal resolution

We obtain:

$$200\frac{\text{degrees}}{\text{sec}}\times\frac{1}{60}\frac{\text{sec}}{\text{fields}}=3.3\frac{\text{degrees}}{\text{field}}\text{ camera rotation}$$

Thus in a 60 degree FOV image when using NTSC video:

$$3.3\text{ degrees}\times\frac{1}{60}\frac{\text{FOV}}{\text{degrees}}\times 600\frac{\text{pixels}}{\text{FOV}}=33\text{ pixels error}$$

Thus with camera rotation of approximately 200 degrees/second, registration errors of more than 30 pixels (for NTSC video) can occur in one field time. The term registration is being used here to describe the correspondence between the displayed image and the placement of objects in the computer generated world.

Note that even though the above discussion concentrates on camera rotation, the argument is valid for any relative motion between the camera and virtual objects. Thus, even if the camera's view point is unchanging, objects moving relative to the camera will exhibit the same registration errors as above. The amount of error is dependent upon the velocity of the object relative to the camera's view direction. If object motion is combined with rotation the resulting errors are correspondingly worse.

The ideal way to generate an image, therefore, would be to recalculate for each pixel the position and orientation of the camera and the position and orientation of the scene's objects, based upon the time of display of that pixel. The resulting color and intensity generated for the pixel will be consistent with the pixel's time of display. Though objects moving relative to the camera would appear distorted when the frame is shown statically, the distorted JITP objects will actually appear undistorted when viewed on the raster display. As shown in figure 4 and plate 2, each pixel in an ideal just-in-time pixels renderer represents a sample of the virtual world that is consistent with the time of the pixel's display.

Computation of both the viewing matrix and object positions for each pixel is quite expensive. Acceptable approximations to just-in-time pixels can be obtained, however, with considerably less computation. One option is to use a single transformation per scan line. This relies on the changes being small during the short (approximately 65 μs) time for the line. Calculations show this to be a reasonable assumption, allowing on the order of 0.13 pixels error.

Another approximation is to use only two transformations per field, one for the first pixel and one for the last pixel. Object positions are linearly interpolated between these two.

4.3 JITP applied to latency

A partial test implementation has been constructed that renders images using the just-in-time pixels paradigm. This system is intended to be used in a see-through HMD to help reduce image generation latency. In a real-time JITP system, instead of computing pixel values based upon the predicted position and velocity of the virtual camera, each pixel is computed based upon the position and orientation of the user's head at the time of display of that pixel. Generation of a just-in-time pixel in real time, therefore, requires knowledge of when a pixel is going to be displayed and where the user is going to be looking at the time. This implies the continuous and parallel execution of the following two central functions:

1) Synchronization of image generation and image scanout
2) Determination of the position and orientation of the user's head at the time of display of each pixel

By synchronizing image generation and image scanout, the JITP renderer can make use of the details of how the pixels in an image are scanned out to determine when a particular pixel is to be displayed. By knowing what scanline the pixel is on, for example, and how fast the scanlines in an image are displayed, the JITP renderer can easily calculate the time of display of that pixel.

Determination of where the user is looking can be accomplished through use of a conventional head tracking system (magnetic or optical for example). Determination of where the user is looking *at the time of display* of a pixel requires the use of a predictive tracking scheme. This is due to the presence of delays between the sampling of the position and orientation of the user's head and the corresponding display of a pixel. Included in the end-to-end delays is the time to collect tracking data, image generation time and the delays due to image scanout.

In the current implementation, the calculations for each scanline are pushed as late as possible. Ideally data for each scanline is transferred to the frame buffer just before it is read out by the raster scan. This technique, known as beam racing, was first used in early flight simulators. By pushing the graphics calculation as late as possible, beam racing allows image generation delays to be combined with display system delays. The result is lower overall end-to-end delay which simplifies the task of predicting the future position and orientation of the user's head. Prediction also benefits from the fact that the delayed computation makes it possible to use the latest available tracking data in the generation of the predicted user view point.

5 SLATS

5.1 Brief Pixel-Planes 5 description

To understand how Slats works requires some knowledge of Pixel-Planes 5 [11]. Using Pixel-Planes 5 gave us total control over the graphics software, which was all developed in-house. Because our goal was to achieve lower latency by modifying the rendering pipeline, such low-level control was necessary.

Referring to figure 2, Pixel-Planes 5 uses parallelism at both the transformation and rasterization stages of the rendering process. Primitives are typically generated on a host workstation and sent via a ring network to a set of graphics processors (GPs), where they are stored in local display lists. The graphics processors traverse these display lists, transforming the primitives from object coordinates to screen coordinates and generating appropriate rendering commands. The graphics processors then send these commands over the ring to the renderers, which perform rasterization and shading. Each of which handles a 128x128 region of the screen. Finally, the renderers send the resulting pixel values to a frame buffer, which is synchronized with a video display for output.

5.2 PPHIGS pipeline

PPHIGS is the standard rendering library for Pixel-Planes 5. It is controlled by a software layer called Rendering Control [8]. The rendering process is broken into three main stages. In the transform stage, the GPs transform the primitives. In the render stage, the renderers scan convert and shade the primitives. If there are more regions on the screen than there are renderers, the first renderer to finish starts on the next screen region. Finally, in the copy stage, the pixel data is copied into the frame buffer. This is illustrated in figure 5.

In this timing diagram and the ones that follow, each line shows use of an independent hardware resource. So the GPs, renderers, and frame buffer can all be used simultaneously. However one stage on the GPs must be finished before the next can begin. Arrows show, for one frame of interest, the dependencies between the different resources. All other timings can (and probably will) change depending on the contents of the scene.

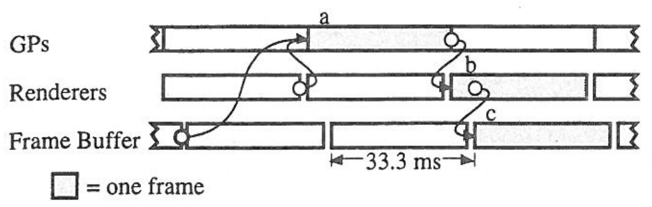


Figure 5: Basic PPHIGS timing for a frame passing through the pipeline. a, b, and c are the transform, render, and copy stages respectively for a single frame. The arrow between the middle of b and the start of c indicates that c can begin as soon as the first region is finished in b.

For stereo operation, PPHIGS handles first the left eye and then the right eye. However, both are considered part of a single unit. When the application software says to draw a frame, images for both eyes are drawn. This is illustrated in figure 6.

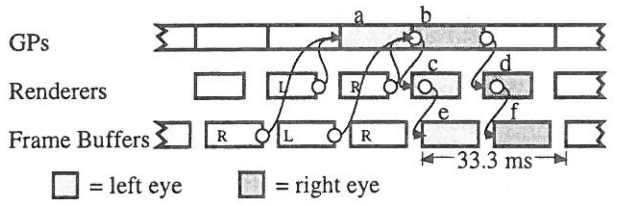


Figure 6: PPHIGS timing for a stereo pair passing through the pipeline. a, c, and e are the transform, render, and copy stages of the left eye. b, d, and f are the right eye.

As was mentioned earlier, the timings, other than those explicitly shown, can vary quite a bit. The lowest latency possible with PPHIGS occurs when the transform and render stages are small and the copy time is the limiting factor. In this case, the synchronization between the stages forces three fields of latency between the time the transformation begins and the time both eyes are complete and the images are displayed. This is illustrated in figure 7.

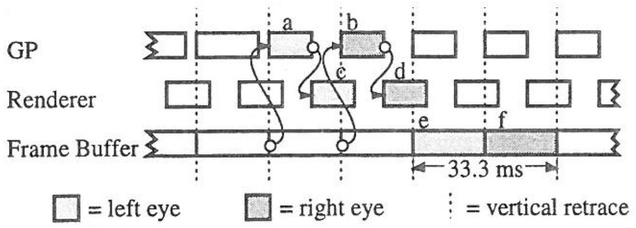


Figure 7: PPHIGS timing for a stereo pair with minimal latency. Render stage to copy stage dependencies are not shown for clarity.

5.3 Slats pipeline

Slats achieves its guaranteed latency by insisting that all the work for one field be finished during the field immediately before it. Since it is built with latency sensitive HMD applications in mind, it always generates stereo images. The pipelining in Slats is at a polygon level. As soon as a set of polygons are transformed (in clumps of 30 for ring network efficiency), they are sent to the renderers. Each renderer handles four screen regions so the entire screen for both eyes can be covered by the available renderers.

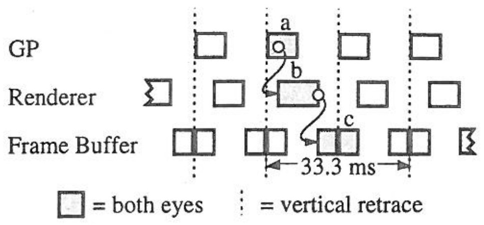


Figure 8: Slats timing for a stereo pair. a, b, and c are the transform, render, and copy stages respectively. Stage b starts after the first batch of triangles are transformed in a. The first half of c must finish before the vertical retrace.

Since a field is two regions high, the copy stage happens in two parts. The copy of the second half of the screen, which only takes 3.9 ms, doesn't occur until after the field is already being displayed. The copying of the first half of the screen must be done before the vertical retrace since those pixels are immediately displayed. This is illustrated in figure 8.

In many ways, Slats falls short of a general graphics library like PPHIGS. For the sake of simplicity, it uses only a single GP instead of the many (up to 50) available to PPHIGS. This severely limits the number of triangles that Slats can handle. The use of four regions per renderer makes polygon level pipelining easier, but also limits the shading model to simple Gouraud color interpolation.

All of the triangles must be transformed and rendered before the first copy begins, a period of about 12.8 ms. If there are too many primitives to make this deadline, Slats fails to generate a correct image. In the current implementation, this translates to about 100 triangles (or 12,000 triangles per second). Even if we optimized the code—and PPHIGS achieved about a factor of three performance increase after the triangle code was optimized to fit in the GP instruction cache—the communication bandwidth out of one GP and the speed of the renderers limits the maximum performance to about 250 triangles. We estimate that using multiple GPs and more renderers we might be able to push this to a few thousand, but currently don't have plans to follow this path.

These limitations are not flaws, Slats excels at what it is built for: experiments requiring low latency, fixed latency, or both. Azuma's work on predictive tracking [1] used Slats for just this reason.

Because it considers both eyes simultaneously, it can share more of the work than PPHIGS, which handles them sequentially but grouped. In fact, both eyes can be copied at the same time. Because it only renders the lines of the image visible in each field — the even lines are rendered while the odd field is visible, and the odd lines are rendered while the even lines are visible — it has half the rendering and half the copying.

As a comparison of the performance of both, figure 9 shows the pixel error for the setup used in our video. There is 33 ms of latency for the optical ceiling tracker[2], making a total of 90 ms for PPHIGS and 50 ms for Slats. Other trackers may have lower latency, but this will only increase the importance of image generation latency since the error is linear with respect to latency[1]. The error was calculated off-line with captured tracker data from a typical demo with a naive user under the optical ceiling tracker. The pixel error shown is computed by taking a point in the center of the field of view for each frame and

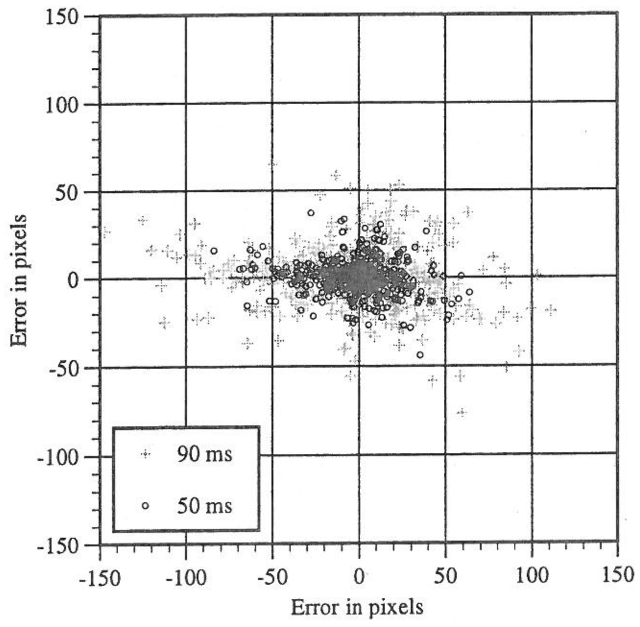


Figure 9: Pixels of error between a pixel at the center of the screen and the location where it should have been displayed by the time the frame was visible. For 90 ms, corresponding to PPHIGS + 33 ms tracker latency, and 50 ms, corresponding to Slats + 33 ms tracker latency.

determining how far from the center it would be when the frame is displayed.

6 CONCLUSION

We have presented two methods for reducing image generation latency. Both, necessarily, at a cost in polygon performance. As HMD applications become more prevalent, people will require minimal latency, much as they do high polygon rendering performance today.

7 ACKNOWLEDGMENTS

We would like to give special thanks to Ron Azuma. Ron is responsible figure 9, and was a huge help in the creation of the Slats video. We would also like to thank Tony Apodaca and Pixar for access to RenderMan, which was used to create the old well JITP simulation.

This project was funded in part by the National Science Foundation, NSF Grant Number MIP-9306208 and NSF Cooperative Agreement Number ASC 8920219, and by the Advanced Research Projects Agency, ARPA ISTO Order Number A410 and ARPA Contract DABT63-93-C-C048.

8 REFERENCES

1. Azuma, Ronald and Gary Bishop. Improving Static and Dynamic Registration in an Optical See-through HMD. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994. ACM SIGGRAPH, New York, 1994, pp. 197–204.

2. Azuma, Ronald. Predictive Tracking for Augmented Reality. UNC Chapel Hill Department of Computer Science PhD Dissertation, 1995.

3. Bajura, Michael, Henry Fuchs and Ryutarou Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In *Computer Graphics*, 26, 2 (July 1992), ACM SIGGRAPH, New York, 1992, pp. 203-210.

4. Bishop, Gary, Henry Fuchs, Leonard McMillan and Ellen Scher Zagier. Frameless Rendering: Double Buffering Considered Harmful. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994. ACM SIGGRAPH, New York, 1994, pp. 175–176.

5. Breglia, Denis, Michael Spooner and Dan Lobb. Helmet Mounted Laser Projector. Proceedings of the Image Generation/Display Conference II (Scottsdale, Arizona June 10–12, 1981). pp. 241–258.

6. Burbidge, Dick, Paul Murray. Hardware Improvements To The Helmet Mounted Projector On the Visual Display Research Tool (VDRT) At The Naval Training Systems Center. Proceedings of the SPIE conference on Head-Mounted Displays, 1989.

7. Cohen, Jon and Marc Olano. Low Latency Rendering on Pixel-Planes 5. UNC Chapel Hill Department of Computer Science technical report TR94-028, 1994.

8. David Ellsworth. Pixel-Planes 5 Rendering Control. UNC Chapel Hill Department of Computer Science Software Documentation, 1989.

9. Feiner, Steven, Blair MacIntyre and Dorée Seligmann. Knowledge-based Augmented Reality. *Communications of the ACM,* 36, 7, July 1993, pp. 52-62.

10. Friedmann, Martin, Thad Starner and Alex Pentland. Device Synchronization Using an Optimal Filter. Proceedings of 1992 Symposium on Interactive 3d Graphics (Cambridge, Massachusetts, March 29–April 1, 1992). Special issue of *Computer Graphics*, ACM SIGGRAPH, New York, 1992 pp. 57-62.

11. Fuchs, Henry, John Poulton, John Eyles, et al. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. Proceedings of SIGGRAPH '89 (Boston, MA, July 31–August 4, 1989). In *Computer Graphics*, 23, 3 (July 1989), ACM SIGGRAPH, New York, 1989, pp. 79–88.

12. List, Uwe Nonlinear Prediction of Head Movements for Helmet-Mounted Displays. Technical Paper AFHRL-TP-83-45, December 1983.

13. Mine, Mark. Characterization of End-to-End Delays in Head-Mounted Display Systems. UNC Chapel Hill Department of Computer Science technical report TR93-001, 1993.

14. Mine, Mark and Gary Bishop. Just-In-Time Pixels. UNC Chapel Hill Department of Computer Science technical report TR93-005, 1993.

15. Regan, Matthew and Ronald Pose. Priority Rendering with a Virtual Reality Address Recalculation Pipeline. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994. ACM SIGGRAPH, New York, 1994, pp. 155–162.

16. Ward, Mark, Ronald Azuma, Robert Bennett, Stefan Gottschalk and Henry Fuchs. A Demonstrated Optical Tracker with Scalable Work Area for Head Mounted Display Systems. Proceedings of 1992 Symposium on Interactive 3d Graphics (Cambridge, Massachusetts, March 29–April 1, 1992). Special issue of *Computer Graphics*, ACM SIGGRAPH, New York, 1992, pp. 43–52.

Underwater Vehicle Control from a Virtual Environment Interface

Stephen D. Fleischer and Stephen M. Rock
Stanford Aerospace Robotics Laboratory

Michael J. Lee
Monterey Bay Aquarium Research Institute

Abstract

This paper describes a collaborative research effort initiated by Monterey Bay Aquarium Research Institute (MBARI), Stanford Aerospace Robotics Laboratory (ARL), and NASA Ames Research Center. The goal of this joint effort was to develop an experimental system which demonstrates real-time supervisory control of an underwater vehicle from an interactive, 3-D graphical interface.

Introduction

OTTER (Oceanographic Technologies Testbed for Experimental Research) is a remotely-operated underwater vehicle jointly constructed by MBARI and ARL. Recently, our research focus has been the creation of a 3-D graphical interface to control OTTER. To accomplish this task, the NASA Ames Virtual Environment Vehicle Interface (VEVI) was chosen as a baseline and extensively modified for use with the underwater vehicle.

This research was divided into two separate objectives. The first objective was to extend the capability of the current X Window-based graphical user interface (GUI) for OTTER, by taking advantage of the current virtual reality (VR) technologies available from NASA Ames. This new GUI is capable of providing the user with better visualization of the underwater environment and improved control of the vehicle.

Our second goal was to demonstrate task-level position control of OTTER from the new virtual environment interface. Specifically, the user should be able to control the vehicle along a desired trajectory by specifying a number of via points along the path. This requires successful integration of the graphical interface into the vehicle control system hierarchy.

Stephen D. Fleischer and Stephen M. Rock
Durand Bldg., Stanford University, Stanford, CA 94305
fleisch, rock@sun-valley.Stanford.EDU

Michael J. Lee
MBARI, 160 Central Avenue, Pacific Grove, CA 93950
lemi@mbari.org

VEVI Structure

In the past, VEVI has been used to control other robotic vehicles of all types with great success. Some examples include the NASA Ames TROV underwater vehicle, which has explored the waters of the Antarctic; the ARL space robots, which simulate the zero-gravity of space in two dimensions; and most recently, the CMU Dante II eight-legged walking vehicle, which explored the Mt. Spurr volcanic crater in Alaska. [1]

Figure 1 shows the current structure of the VEVI software, as implemented in the OTTER control architecture. The core of VEVI is the Renderer, which was written on top of the WorldToolKit (WTK) world simulation library. The WorldToolKit library, developed by the Sense8 Corporation, enables programmers to graphically simulate an environment, including the universe model and any movable objects within that universe. The Renderer has the capability to interact with novel virtual reality devices, such as stereo or head-mounted displays, flying mice, 6-DOF spaceballs, and head-trackers.

In order to communicate with the connected vehicle, VEVI transfers data through shared memory to the VehicleNode, which then communicates to the vehicle over a network. The NDDS network protocol, developed by students in ARL at Stanford University, [2] provided the communications interface between VEVI and OTTER.

The VeviNode provides VEVI with the ability to support multiple users across a network. The Renderer talks to the VeviNode through shared memory, which then broadcasts information to other copies of VEVI running on the network.

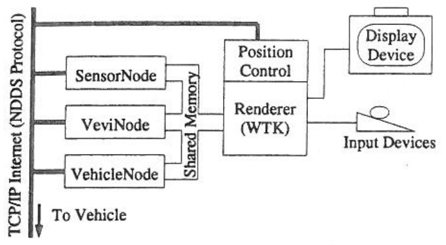


Figure 1: **VEVI Structure**

The SensorNode is only used if there are specialized sensors which cannot be accessed through the VehicleNode. All sensors on OTTER, including the thrusters, cameras, and acoustic positioning system, are accessed directly through the VehicleNode.

Implementation on OTTER

In our attempt to achieve position control from the virtual environment interface, we were able to take advantage of the OTTER Task-Level Control architecture. [3] As seen in Figure 2, this paradigm divides the vehicle control system into three levels, which can each be implemented on separate computers. The lowest (servo) level includes the real-time control loops, which are implemented in the computers on-board the underwater vehicle. Task commands are sent from the organizational level, which are then decomposed into smaller tasks by the middle (task) level and sent to the servo level for execution.

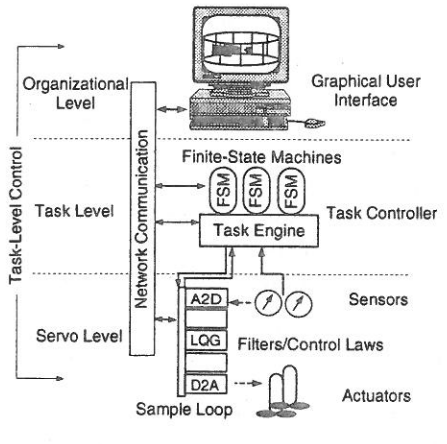


Figure 2: **OTTER Task-Level Control Structure**

This methodology isolates the graphical user interface, which encompasses the entire organizational level, from the lower levels of the control hierarchy. Conceptually, task-level control enables the user to perform complex functions (e.g. driving a vehicle along a desired path) by combining lower-level tasks which can be performed autonomously by the vehicle. After implementing a simple point-to-point transect as a task for the vehicle, we added a position control module to VEVI. With this module, the user is able to graphically specify a series of via points along a path by dragging around a ghost image of OTTER in the virtual environment. These via points are then translated into task commands which are sent to the vehicle.

Conclusions

We have performed several demonstrations of task-level position control from the virtual environment interface in the Naval Postgraduate School (NPS) fresh-water test tank. Currently, VEVI runs in single-user mode, with a standard SGI color monitor for display and a standard 2-DOF mouse for user input.

By developing this operational platform for experimental research, we plan to pursue fundamental research in the development of interactive, 3-D virtual environment interfaces to control complex, real-time robotic systems. In terms of the OTTER project, we believe that continued research will encourage teams of marine research scientists to work with the underwater vehicle remotely, while enabling them to visualize the environment and relevant data in real-time.

In essence, we hope to enable the end-user to become more effective in performing a variety of tasks by maintaining a simple, intuitive interface to an inherently complex system.

References

[1] Fong, T. W. A Computational Architecture for Semi-autonomous Robotic Vehicles. In *Proceedings of AIAA Computing in Aerospace 9 Conference* (San Diego, CA, October 1993), AIAA.

[2] Pardo-Castellote, G., and Schneider, S. A. The Network Data Delivery Service: Real-Time Data Connectivity for Distributed Control Applications. In *Proceedings of the International Conference on Robotics and Automation* (San Diego, CA, May 1994), IEEE, IEEE Computer Society.

[3] Wang, H. H., Marks, R. L., Rock, S. M., and Lee, M. J. Task-Based Control Architecture for an Untethered, Unmanned Submersible. In *Proceedings of the 8th Annual Symposium of Unmanned Untethered Submersible Technology* (September 1993), Marine Systems Engineering Laboratory, Northeastern University, pp. 137–147.

Interactive Design, Analysis, and Illustration of Assemblies

Elena Driskill[†] Elaine Cohen[‡]

Department of Computer Science
University of Utah
Salt Lake City, Utah

Abstract

We present an interactive approach for helping designers describe, revise, analyze, and illustrate assemblies of mechanical parts within the context of a common data structure and set of assembly features. This paper describes an implementation used to test the validity of these ideas, which has been integrated into an existing spline-based geometric modeling system.

Several interactive tools have been implemented. An assembly planner allows the user to design the assembly structure before modeling any geometry by using a combination of top-down and bottom-up design. After the geometry of each part in the assembly, together with its assembly features, has been modeled, the user can interactively put the parts together and perform degree of freedom analysis on them by using another tool. Such an interactive approach can help a designer determine whether the design is sound before the entire assembly is put together. Finally, once all part connections have been established, an exploded view generation tool can help the designer create an informative illustration of the product for the purposes of documentation or further visual analysis.

1. Introduction

Interactive computer-aided design systems were developed to help designers create models of mechanical parts as part of the mechanical engineering process. Instead of making numerous detailed mechanical drawings on paper, a designer can now create a three-dimensional model of a part, experiment with different shapes and proportions, even analyze the part's structural stability, then create rendered images of the part to show others.

A mechanical part is seldom designed to stand alone. More often is created to be used as part of some machine, mechanism, or another kind of assembly, yet few aids for designing assemblies are available. As the engineer designs the part, he already knows what the mechanism is supposed to do and how it will do it, has some ideas of what the other components in the mechanism will look like, and how all of these components will fit and work together. A system can collect such information from the designer and use it to advantage.

This work is a step towards having the design of mechanisms be as straightforward to the user as the design of parts. The methodology described here takes the designer from the early planning stage of the assembly (which takes place before the design of individual assembly components) through assembly analysis and the creation of exploded view illustrations. Each tool described here augments the assembly description and also utilizes information obtained in previous design steps.

2. A Brief Overview of Related Work

Surprisingly little work has been done on assembly planning. Gui and Mantyla [3] have developed a system which supports top-down functional design for creating assemblies based on functional and behavioral, but not necessarily geometric, knowledge about a product.

Once geometry has been designed, it is necessary to specify part positions relative to one another in an assembly. There are several approaches. Sometimes transformation matrices are used to implement the specification of a rigid body transformation necessary to move an object into the assembled position, given relative to the world coordinate system [6] or relative to other components in the assembly [2, 11]. Another style of transformation specification uses mating conditions [4], where particular locations on assembly components are specified to mate, and in our assessment is the most advantageous since the specification of positions by using transformation matrices is error prone. Assembly features may be used for specifying mating conditions. Various issues related to modeling with features in various contexts are discussed in [8]. In [7], mating conditions are derived from the geometric and topological information stored in the model itself. Researchers have also considered the problem of determining translational and rotational degrees of freedom and finding disassembly directions of parts [7, 12].

However, most researchers tend to concentrate on a particular area of assembly design, such as generating an assembly sequence [7, 1], performing kinematic simulation [11], and so on, and few integrate a spectrum of different operations together into a coherent system for assembly design. For example, some researchers rely on asking the user questions about which parts should be assembled before others in order to determine a precedence graph from which they then find assembly sequences [1]. This is an instance of solving a very specialized problem, where geometric information is not even utilized; only precedence data obtained by a rather error-prone process is used. Also, some systems attempt to second-guess the designer and automatically break the assembly into subassemblies [5, 10]. Subassemblies created in this way do not always make

The authors may be contacted via electronic mail at
† elenad@fa.disney.com or elena@cs.utah.edu
‡ cohen@cs.utah.edu

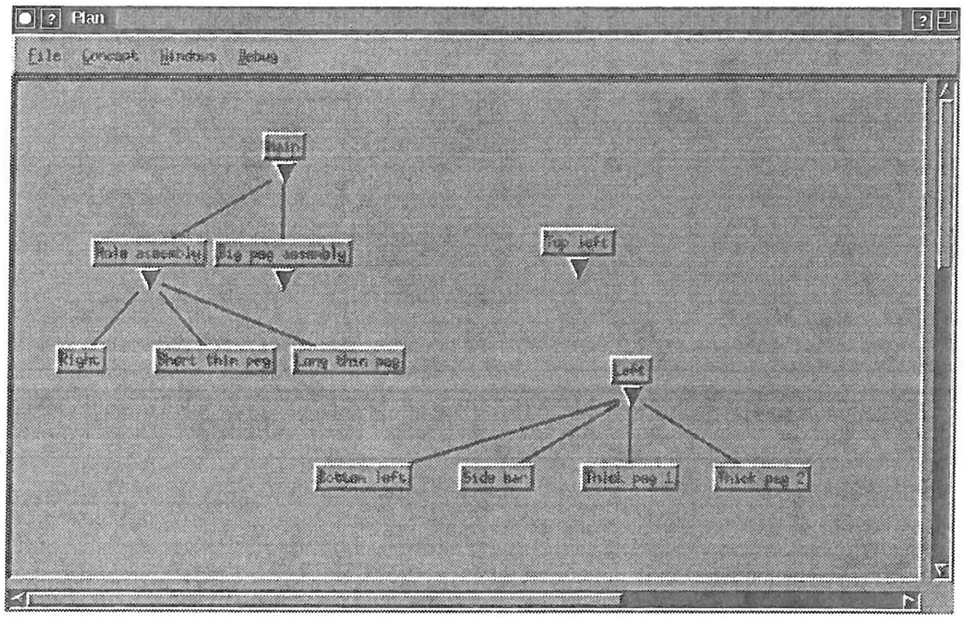


Figure 1. An assembly design in progress.

sense, and a combinatorial explosion of possibilities for part combinations prevents the systems from handling very large assemblies in reasonable time.

As for the automatic generation of exploded views, the authors have been able to find only one reference which even mentions such a capability. Strip and Maciejewski's system Archimedes [9] produces exploded views of assemblies for the purposes of visualization within the robot planning system. However, the geometry of the parts is limited to arbitrarily stepped cylinders and holes whose axes are parallel to each other, a domain in which the generation of exploded views is unidirectional. The production of exploded views of assemblies was only mentioned in passing and does not appear to be interactive or general purpose.

3. Assembly Planning

A straightforward way of visualizing an assembly is as a tree, where the leaves are individual parts and the internal nodes are subassemblies. Simpler subassemblies are connected into more complex ones, and the root of the tree is the final completed assembly.

Different people have different ways of thinking about their designs. Some start with the assembly and break it down into simpler components. Some start with the parts and build the assembly up from them. Typically a combination of these methods is useful. The top-down approach works well for large complex assemblies. The designer knows what the desired final product is and what the major components will be, but at the outset, probably has not thought about the smallest details. However, suppose the designer wants to make a fixture which will hold a part during machining and has a catalog of standard fixturing elements. He may wish to design the fixture from the bottom up, by starting with the fixture components from the catalog and the stock from which the part will be machined, and building the assembly up from there.

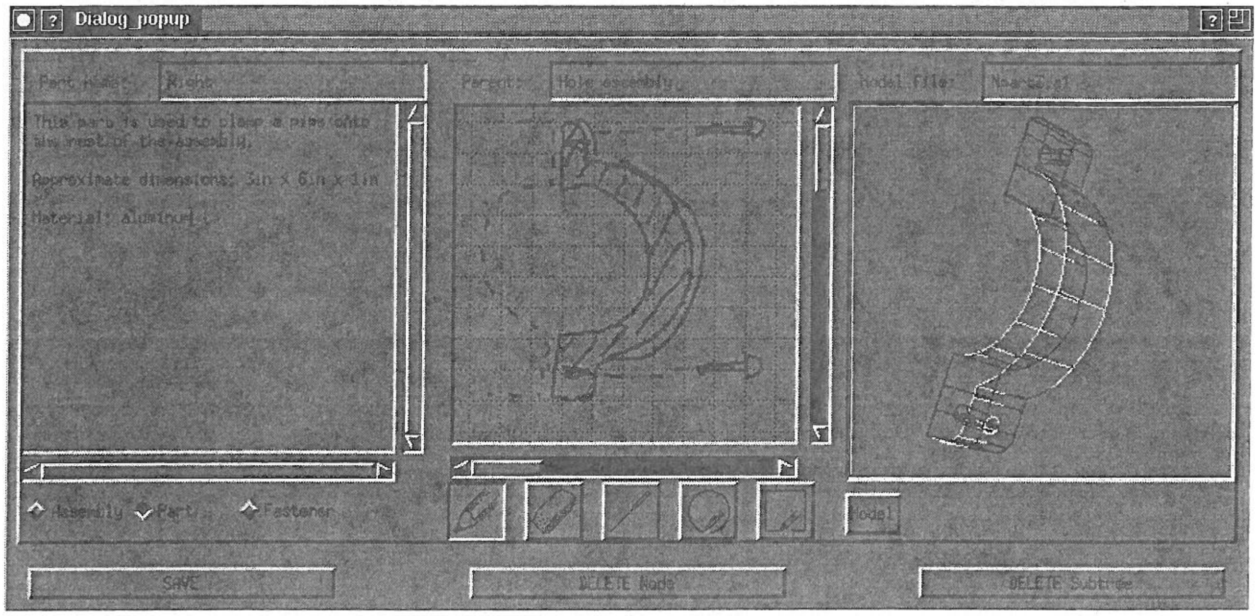
To accommodate these different approaches, the assembly planner allows both methods of design. The user can either create children of any node and design from the top down, or create unconnected nodes and subtrees and then attach them as children of other nodes, designing from the bottom up. In fact, the methods are likely to be used in combination in a repetitive process. Figure 1 shows a screen capture of an assembly in progress. The user has designed three unconnected subtrees, two of which are subassemblies, and the third of which will perhaps be expanded into a subassembly in the near future. Later, the user will connect the components into a single assembly tree.

Simply specifying an assembly structure is not enough during assembly planning. The designer may wish to think about the geometry of the assembly components, make annotations about the components' function, construction, and connections with other components, and later, perhaps associate a three-dimensional geometric model with each part. Our assembly planner includes a simple annotation mechanism, which enables the user to enter textual and graphical documentation for any assembly component. Figure 2 shows a component's information window, which may be opened by clicking on that component's node in the tree with the mouse. It includes an area for entering the component's name (which is the name that appears in the component's pushbutton), an area for textual documentation, and a sketchpad with several drawing tools, which can be used to plan the design. There is also an area for associating a three-dimensional geometry with a component when this geometry becomes available. A three-dimensional description of the part is designed using a geometric modeler, and to associate this data with a component, the user supplies the name of a file containing this description. Each part's geometry may be designed in its own coordinate system.

Although in a finished assembly model it makes sense for only the individual parts to have an associated geometry, a simplified geometry could also be associated with unfinished subassemblies while the design is in progress. These geometries could approximate the sizes and shapes the final subassemblies are expected have, and the designer could use these approximations to determine whether or not the subassembly would fit into the rest of the assembly properly before finishing up the detailed design of the parts.

4. Features and Assembly

Geometry alone does not conveniently provide information about how parts in an assembly might be connected. It is very dif-



Part name: Right

Parent: Hole assembly

Model file: [illegible]

This part is used to clamp a pipe onto the rest of the assembly.

Approximate dimensions: 3in x 6in x 1in

Material: aluminum

Figure 2. The component information window.

ficult, if not impossible, to determine whether a particular indentation in a part geometry is there to make the part lighter by removing nonessential material or whether it is intended to mate with a protrusion on another assembly component.

Features have been successfully used in geometric modelers to supply additional information for tasks such as automated machining. This work uses assembly features to carry information appropriate to aid the process of putting an assembly together. We define a set of assembly features, including several new ones, for this purpose; these features are listed and shown in Figure 3. It is certainly not an exhaustive set of possible assembly features, but it gives a good idea of what a feature-based approach can accomplish. These features are associated with each part's geometry when this geometry is designed in the geometric modeler. Each feature contains a geometric description (e.g. a radius, a length, a description of a surface or of a cross-section curve, etc.) and a center point and orientation, defining the feature's coordinate system with respect to its component's coordinate system.

Assembly-specific information carried or represented by a feature might include the types of other features it can mate with, and the probable removal direction it indicates for the parts it helps connect. For example, a cylindrical peg feature may mate with cylindrical hole features which have the same cross-section radius as the peg. Parts which are mated with the help of the peg and hole may be separated along the axis of the peg. Features may indicate other useful information also. For example, if one component has two peg features and the other has two hole features which are specified to mate, yet the peg features' axes are not parallel, the parts cannot mate because it is physically impossible to insert both pegs into the holes without interference.

Sometimes an assembly feature is distributed over several assembly components and is not complete until all of these components are connected in such a way that the sections of the feature align. To describe this situation, we have defined the concept of partial assembly features. An example of a partial feature is shown in Figure 4 The specification of partial features is accomplished by associating a particular assembly feature with more than one part in the geometric modeler. The feature description also contains the number of parts containing pieces of the same feature. Each time a component containing a section of the feature is attached to a subassembly containing another section of the same feature, the count of feature sections in the subassembly is decremented by the number of feature pieces contained within the newly added component. When the count is 1, the feature is complete.

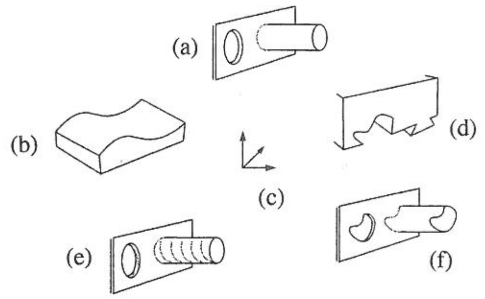


Figure 3. Currently defined assembly features: (a) round peg/hole, (b) surface, (c) location and orientation feature (no geometry), (d) dovetail and dovetail groove, (e) threaded peg/hole, (f) peg/hole of arbitrary cross-section.

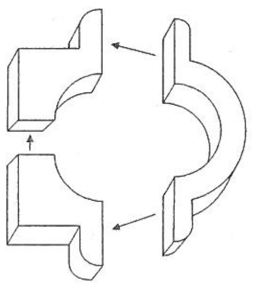


Figure 4. The hole which will be formed when the three components are aligned is an example of a partial feature. Pockets, surfaces, and other features may be similarly distributed over more than one assembly component.

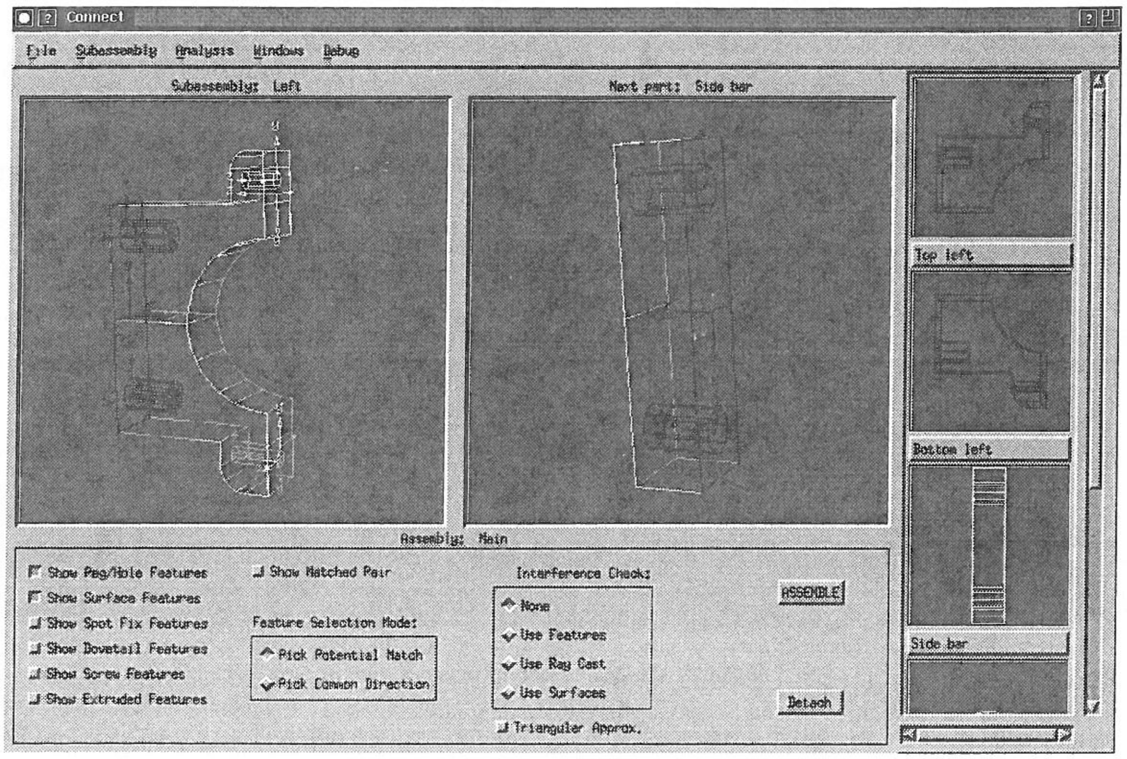


Figure 5. The assembly tool.

We have created an interactive assembly specification capability. It uses data from the assembly planner to determine the order in which to present subassemblies to the user for the interactive specification of connections. A screen capture of the tool is shown in Figure 5. The small display areas on the right show all the component parts or other subassemblies belonging to the current subassembly (as defined in the planner), the larger area in the middle displays the next component the user has selected for attaching to the subassembly, and the display area on the left shows the partially completed subassembly. A subassembly may only be selected for the specification of connections if the geometries of all of its components are defined. In other words, all of its components must be either single parts, whose geometries are known from the start, or subassemblies whose connections, and therefore geometries, have already been specified.

A set of toggle buttons determines which types of features are currently selectable (and highlighted). Only completed features are shown, partial features are not visible until they are completed. At each step, the user selects a feature or a set of features to mate on each of the two components, and the system attempts to attach the part to the subassembly in such a way that all the mating conditions are met. This is accomplished by finding all potentially matching pairs of features (by comparing geometric attributes, such as radius, and location relative to other features on the same component), then finding a set of feature pairs where each feature on the first component matches exactly one feature on the second component. From the mated features, a transformation is computed which aligns the parts in such a way that the selected feature pairs mate. If more than one part mating satisfies the required conditions, all possibilities are found and the user is allowed to toggle through them and select one. However, such an ambiguous situation can potentially indicate to the user that a flaw exists in the design.

As components are added to a subassembly, one of several things can happen to those components' features. First, a copy of any feature not used in the mating is added to the feature list of the subassembly, where it can be used later to help add other components to the subassembly. A feature which participates in a mating may be altered. This currently applies to matings of holes with other holes. When two holes are aligned, it is logical that they coalesce to form a single, longer hole which may be used in a future mating. Such a longer hole is computed and added to the feature list of the subassembly. Finally, a feature may be eliminated in the mating. For example, surface features mate over their entire surface. Because of this, they are no longer useful for future matings, and they are not inherited.

History pointers are stored with each feature. The previous history pointer points at the previous incarnation of the feature, the one on the component the feature was copied from. The next history pointer points in the other direction of increasing complexity. For partial features and coalesced holes, a list of the components where the pieces of the feature originated is stored instead. Because of this, it is possible to implement a multiple undo capability for assembly operations without using special additional storage.

5. Assembly Analysis

Being able to analyze parts for proper fit and removability is an important capability for interactive assembly modeling. The designer receives immediate feedback and can redesign part geometry as necessary without having to put the whole assembly together before finding out that something is wrong.

The assembly tool provides several options for feedback on component interference as parts are added to the current subassembly, with varying degrees of reliability and computational speed. The user can also select a component in the current subassembly and examine its translational and rotational degrees of freedom to determine if the component is constrained or free to move as expected.

Three interference detection methods of progressive accuracy