



AFRL-HE-WP-TR-2007-0038

Task Adaptable Display of Information for Training, Maintenance, and Emergency Response

**David S. Ebert
Jingshu Huang**

**Purdue University
Sponsored Program Services
302 Wood Street (Young Hall)
West Lafayette IN 47907-2108**

December 2006

Final Report for June 2005 to December 2006

**Approved for public release;
distribution is unlimited.**

**Air Force Research Laboratory
Human Effectiveness Directorate
Warfighter Readiness Research Division
Logistics Readiness Branch
Wright-Patterson AFB OH 45433-7604**

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory, Human Effectiveness Directorate, Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-HE-WP-TR-2007-0038 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE COMMANDER

//SIGNED//

DANIEL R. WALKER, Colonel, USAF
Chief, Warfighter Readiness Research Division
Air Force Research Laboratory

This technical report is published as received and has not been edited by the Air Force Research Laboratory, Human Effectiveness Directorate. This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its idea or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) December 2006		2. REPORT TYPE Final		3. DATES COVERED (From - To) June 2005 - December 2006	
4. TITLE AND SUBTITLE Task Adaptable Display of Information for Training, Maintenance, and Emergency Response				5a. CONTRACT NUMBER FA8650-05-2-6648	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER 62202F	
6. AUTHOR(S) David S. Ebert, Jingshu Huang				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER 1710D227	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Purdue University Sponsored Program Services 302 Wood Street (Young Hall) West Lafayette IN 47907-2108				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Materiel Command Air Force Research Laboratory Human Effectiveness Directorate Warfighter Readiness Research Division Logistics Readiness Branch Wright-Patterson AFB OH 45433-7604				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/HEAL	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-HE-WP-TR-2007-0038	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES AFRL/PA cleared on 26 April 2007, AFRL/WS-07-1049.					
14. ABSTRACT This project has developed techniques for effective display of information for environments ranging from job training, to maintaining and repairing equipment, to responding to critical situations. The techniques developed allow the adaptable display of digital model data (e.g., mechanical parts, buildings) from the desktop to mobile devices. The project also performed an evaluation of the effectiveness of rendering methods for mobile devices for the task of locating a specific mechanical part in an assembly for training or maintenance.					
15. SUBJECT TERMS Job Training, Maintenance, Repair, Screens(Displays), Technical Data, Visual Aids					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT SAR	18. NUMBER OF PAGES 36	19a. NAME OF RESPONSIBLE PERSON Jill A. Ritter
a. REPORT UNCLASSIFIED	b. ABSTRACT UNCLASSIFIED	c. THIS PAGE UNCLASSIFIED			19b. TELEPHONE NUMBER (include area code)

THIS PAGE LEFT INTENTIONALLY BLANK

Table of Contents

List of Figures.....	iv
List of Tables	iv
1. Project Summary.....	1
2. Technical Developments.....	1
3. MobileVis System.....	1
3.1 Data Structures and Transcoder	2
3.2 Interactive Illustrative Rendering.....	2
3.2.1 Silhouettes.....	2
3.2.2 Selective rendering.....	4
3.2.3 Cutaway views.....	4
3.2.4 Ghosted view.....	5
3.2.5 Magic lens	6
3.2.6 Offset examination	7
3.2.7 Peeling and animation	7
3.3 Annotations.....	8
3.4 Performance	9
4. Preliminary User Experiment	12
4.1 Experimental Design.....	13
4.2 Apparatus	13
4.3 Subjects	13
4.4 Stimuli.....	15
4.5 Procedure.....	16
4.6 Data Analysis.....	16
5. Refined User Experiment.....	17
5.1 Stimuli	17
5.2 Subjects	18
5.3 Procedure.....	18
5.4 Data Analysis.....	19
6. Final User Evaluation Study Design.....	22
7. Conclusions.....	22
8. References.....	23
Appendix A.....	24

List of Figures

Figure 1: Modified 2-pass silhouette algorithm.....	3
Figure 2: Selective rendering of a jet engine (left) and a Boeing 777 auxiliary power unit	4
Figure 3: Cutaway views of Boeing 777.....	5
Figure 4: Ghosted views	6
Figure 5: Magic lens view.....	7
Figure 6: Labeling.....	8
Figure 7-1: Photo of our system running on a PDA	9
Figure 7-2: Photo of our system running on a PDA	10
Figure 8: Performance in different rendering modes.....	12
Figure 9: Results of our refined experiments.....	21

List of Tables

Table 1: Frame rates of rendering 3D models in different rendering modes.....	11
Table 2: Models and their target parts	14
Table 3: Test cases in preliminary user experiments	15
Table 4: Rendering cases for training in refined experiments	18
Table 5: Representative rendering cases in real tests.....	20

1. Project Summary

This project has developed techniques for effective display of information for environments ranging from job training to maintaining and repairing equipment to responding to critical situations. The techniques developed allow the adaptable display of digital model data (e.g., mechanical parts, buildings) from the desktop to mobile devices using illustrative rendering techniques to create the mobileVis system. The mobileVis system has many illustrative rendering styles that can be selectively applied to more clearly and succinctly show important information, including silhouetting, selective rendering, cutaway views, ghosted views, magic lens viewing, offset examination, object peeling, animation recording and playback, and labeling. The project also performed an evaluation of the effectiveness of illustrative rendering methods for mobile devices for the task of locating a specific mechanical part in an assembly for training or maintenance.

2. Technical Developments

The main development in this project was the creation of a device adaptable rendering system, mobileVis, for displaying digital model data that adapts the presentation of the data based on the capabilities of the device and the ability to tailor the representation to the task to be performed. Several illustrative rendering styles were developed for mobile device rendering. A series of preliminary user studies were conducted to refine the techniques and to design a final user experiment. The experiment's purpose is to evaluate different rendering styles' effectiveness for the important training, education, maintenance, and repair task of finding a part within a mechanical assembly. The details of the mobileVis system, the preliminary informal user studies, and the final evaluation experiment design are detailed below.

3. MobileVis System

The mobileVis system was developed to provide adapted rendering of digital models on devices ranging from desktop PCs to tablet PCs to PDAs and smartphones/pocketPCs. mobileVis was developed in C/C++ using OpenGL ES 1.0 for graphics rendering on mobile devices. The system consists of three modules, a data transcoder that converts 3DS MAX models into a customized binary format for fast loading of the models on mobile devices, a renderer equipped with a toolbox of different illustrative rendering modes, and a labeling system which allows a dynamic display of labels and text information associated with parts of a 3D model.

3.1 Data Structures and Transcoder

Since most mobile devices do not have floating point processing units, the data transcoder converts 3D models from floating point to a binary format using only signed and unsigned short integers. The data at each vertex consist of a position (x, y, z), a normal (n_x, n_y, n_z), and a color/opacity (r, g, b, a). Each component of the position and normal is represented by a 2-byte signed short integer and each component of the color/opacity is represented by a byte. Thus, a total of 16 bytes are needed per vertex. To minimize floating point operations, all the computations, including the transformations of projection and model view matrices, are performed using fixed point math [1].

Furthermore, for performance concerns, OpenGL ES has no support for *glBegin()/glEnd()* and requires all the graphics to be performed using vertex arrays. In order to minimize the number of duplicate vertices sent to the transformation stage of the graphics pipeline, we use indexed vertex arrays and, thus, maintain a list of vertex indices following the order of the triangle list in the original model. Each element in this new vertex index list is represented as an unsigned short integer.

In order to retain hierarchical position information of different parts of an original 3D model, the model is processed in a pre-defined order and the converted vertex array data is sequentially stored. To identify the vertex indices of an object in the vertex index list, we keep an array of the number of triangles of each object in the model. With this array, the vertex indices of a selected object can be easily located in the vertex index list by summing the number of triangles of all the objects stored prior to the selected object.

3.2 Interactive Illustrative Rendering

This project also developed a toolkit for interactive exploration of 3D polygonal models on mobile devices. The toolkit includes the following illustrative rendering styles: **silhouettes, selective rendering, cutaway views, ghosted views, magic lens views, offset examination views, and animations.**

3.2.1 Silhouettes

Silhouettes are important illustrative rendering techniques that convey shape and spatial relationships for 3D objects. With polygonal models, silhouettes are defined as the edges in the mesh that share a front- and a back-facing polygon. Isenberg et al. [4] summarize existing silhouette algorithms and categorize them into three groups - image-space, object-space and hybrid. When interactive frame rates are desired, and when the devices have limited memory or slow processors, image-space and hybrid algorithms are recommended. However, image-space algorithms usually require reading and operating on a z-buffer which is not accessible within the OpenGL ES profile. Therefore, we implemented a modified hybrid algorithm based upon the Raskar and Cohen two-pass silhouette algorithm [5].

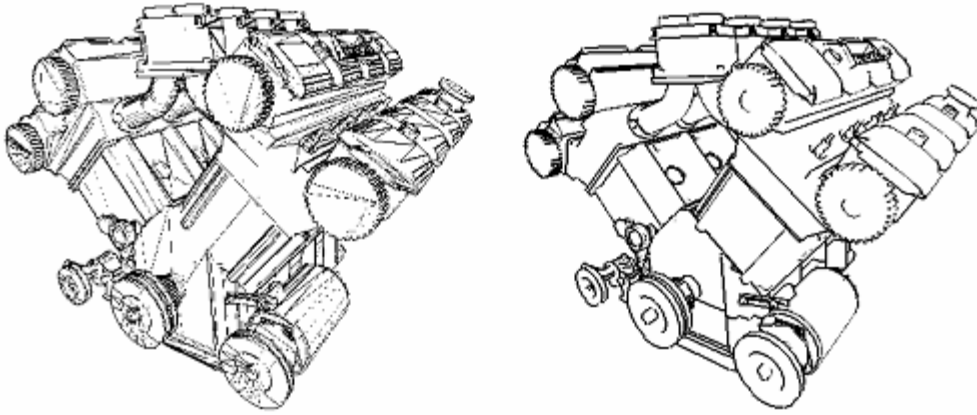


Figure 1: Modified 2-pass silhouette algorithm employed onto a V8 engine dataset rendered on a PDA without a bit buffer (left) and with a bit buffer (right)

Because line drawing via polygon mode *glPolygonMode()* is not available in OpenGL ES, we rendered polygons in wireframe mode in the second pass of the silhouette algorithm. However, this adaptation generates line artifacts on relatively large, flat surfaces as shown in the left image of Figure 1. These lines are wireframe edges in the polygonal mesh. To remove the artifacts, we employed a silhouette detection technique modified from the edge buffer method described in [2]. Rather than using 2 bits per edge as proposed in the original algorithm, we stored only 1 bit per edge and our 2-pass silhouette algorithm works in the following manner:

- Initialize all the bits in the bit buffer for edges to 0.
- Identify and mark silhouette edges by going through all of the mesh edges and flipping bits corresponding to all of the edges of backfacing triangles. By doing this, the bits that correspond to silhouette edges will have a resulting value of 1 because they are flipped only once, whereas edges on the backside will have their bits flipped twice (and become 0) and the edges on front side will have their bits unmodified (and remain 0).
- Perform two-pass rendering. In the first pass, draw front facing polygons with a slight offset of the polygon nearest to the viewer, and set the mask of the depth buffer to TRUE and that of the frame buffer to FALSE; in the second pass, change the masks of both the depth and frame buffers, and draw the silhouette edges found in the previous step with a pass condition in the depth test set to “less than or equal to.”

3.2.2 Selective rendering

Silhouettes allow users to perceive the shapes of objects easily. By combining silhouettes with other rendering styles, it is easy to create a focus view of important features in a model while still conveying the overall shapes and context information. Figure 2 shows an effective use of a mixture of a silhouette rendering mode for the overall shapes of the 3D models with surface and transparency rendering modes applied to the interior feature objects. Both images were rendered in our mobileVis system on a PDA with a 240x320 display resolution. Users first specify a rendering style and then select the objects that need to be rendered in the selected style by clicking on the objects. Because the selection mode is not supported intuitively in OpenGL ES, we implemented picking by rendering object IDs into the frame buffer with the depth test enabled and then read the color buffer back.

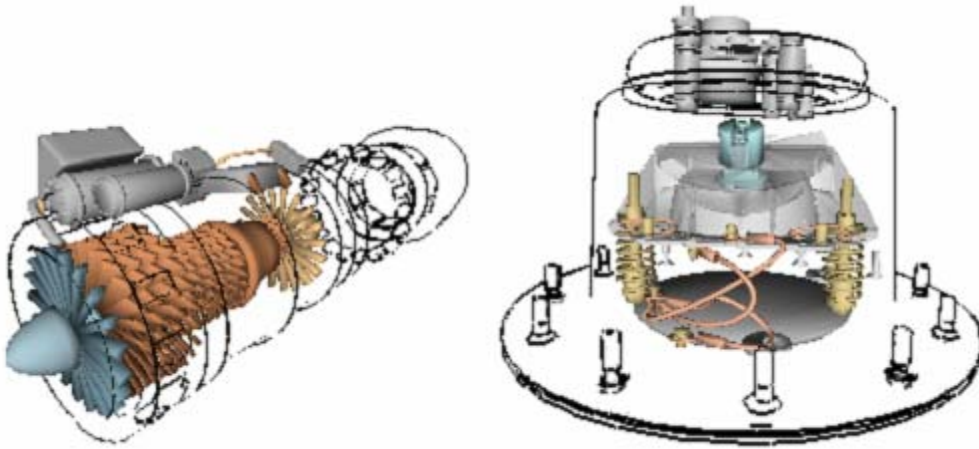


Figure 2: Selective rendering of a jet engine (left) and a Boeing 777 auxiliary power unit (right) on a PDA

3.2.3 Cutaway views

For cutaway illustrations, different approaches for desktop platforms exist, such as constructive solid geometry (CSG)-based cutouts, stencil buffer-based cutouts and texture-based cutouts [3]. However, these techniques are either too computationally expensive for mobile devices (e.g., CSG operations) or require hardware features not available on most testing devices, including the device we used for testing, a Dell Axim x51v PDA. The necessary missing hardware features are a stencil buffer or hardware-accelerated fragment operations. Therefore, we implemented a selective planar cutout technique for cutaway illustration in mobileVis. The system provides users with an interface to dynamically enable and disable ($1 \leq n \leq 5$) clipping planes and control their positions individually. In the cutaway mode, clipping planes are applied on user-selected objects. These objects are maintained in a cutaway list and rendered n times. Each time only the respective clipping plane is activated. Unpicked objects are rendered only once before rendering any selected objects. Figure 3 shows two examples of cutaway views. In both images, the cutaway views are combined with selective rendering to reveal interior structures of the polygonal

models. The capping of the cutout surfaces, the shading of the surfaces formed by a clipping plane and the object the plane clips, was implemented by performing an extra rendering pass on the back faces of the polygonal model with lighting disabled and depth testing enabled.

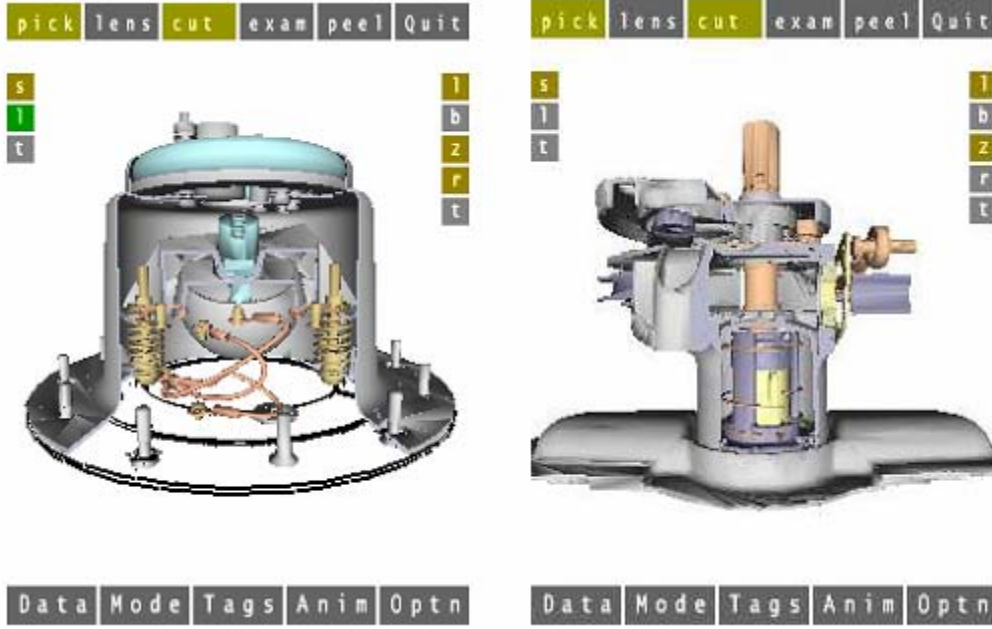


Figure 3: Cutaway views of Boeing 777 parts. Left: 3 clipping planes are enabled. Right: 2 clipping planes are enabled. Both were rendered in mobileVis on a PDA

3.2.4 Ghosted view

A ghosted view is a rendering technique that is similar to a cutaway view except that, instead of doing a hard-edge cutaway, opacity in a selected occluding region is reduced gradually to allow a transparent view of interior objects while still preserving features of occluding objects, such as edges. Ghosted views help users to improve their understanding of interior structures and their spatial relations to their contextual objects. In mobileVis, we implement ghosted viewing by modulating the opacity values at the vertices near a user-selected point in an object. The opacity α of a vertex in the picked object is modified as:

$$\alpha = \begin{cases} 1, & d \geq r \\ \frac{d}{r}, & \alpha_0 r < d < r \\ \alpha_0, & d \leq \alpha_0 r \end{cases} \quad (1)$$

where d is the distance between the vertex and the central point of the triangle which a user picks, α_0 is a predefined minimal opacity value between 0 and 1, e.g. 0.15, and $r = 0.25 * \min(\dim_x, \dim_y, \dim_z)$ where $\dim_x$, $\dim_y$, $\dim_z$ are the sizes of x , y and z dimensions, respectively. Figure 4 shows an example of ghosted views.

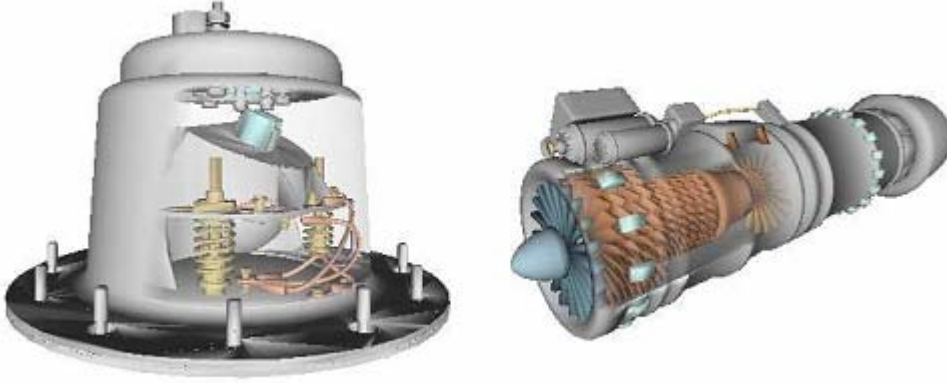


Figure 4: Ghosted views of a Boeing 777 auxiliary power unit (left) and a jet engine (right) in MobileVis running on a PDA

3.2.5 Magic lens

A magic lens is similar to a magnifying glass and can be used to illustrate and emphasize the details of a portion of a model. The objects displayed under the lens can be shown in different rendering styles or varying degrees of detail from their context. In mobileVis, the system magnifies the portion of a model under the lens and, furthermore, assigns a see-through capability to the magic lens. From observations of mechanical 3D models, one can easily see that many of these models have a “shell” or covering object, which is usually relatively large in size and less interesting (e.g., contains fewer features than interior objects that the shell occludes). Therefore, for the magic lens view, a function is defined that decides if an object should be drawn under the magic lens. If the area that an object covers under the magic lens exceeds a threshold value, it is likely that the object is a shell or covering object and occludes potentially interesting details of a model. In such cases, the portion of the object under the magic lens is removed and the occluded objects are displayed with magnification. Through experiments on most of our testing models, an optimal threshold value was obtained and the visibility function is, therefore, defined as Equation 2.

$$visibility_i = \begin{cases} 1, & \frac{c_i}{S} \leq 0.3 \\ 0, & \frac{c_i}{S} > 0.3 \end{cases} \quad i = 1 \dots N \quad (2)$$

where N is the total number of the objects, c_i is the coverage (in pixels) of the object i , and S is the area size of the lens. An example of a magic lens view is shown in the left image of Figure 5. In this image, the car body cover on the driver’s side occludes the view of the interior objects in the car. Through a magic lens, the portion of the cover under the lens is defined as an occluding object using Equation 2 and, thus, removed.

The interior structures of the car are exposed and magnified. The magnification factor of the lens in the magnified image is 1.7.

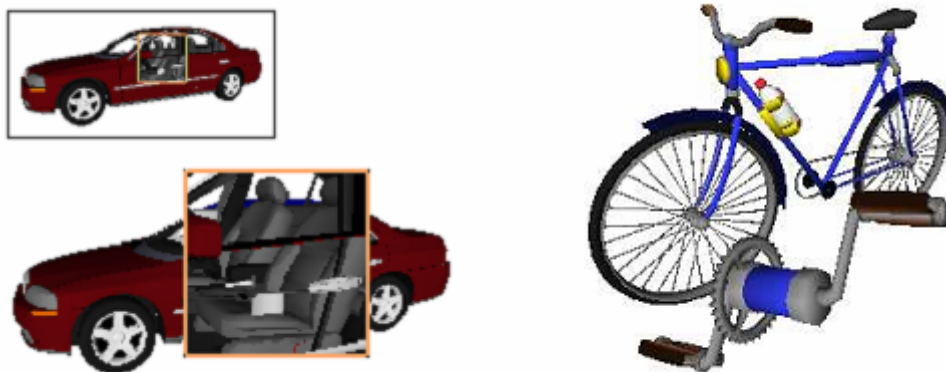


Figure 5: Magic lens view of a car model (left) and offset examination view of a bike model (right) in MobileVis running on a PDA

3.2.6 Offset examination

Spatially displacement of component parts is a commonly used technique to uncover prominent features inside a model. The offset parts can be either displaced towards the viewer and retain their relative spatial relationships to each other, or be spread out on the plane perpendicular to the viewing vector, thus forming an exploded view, as shown in Figure 5. In the first case, the displacement usually provides an effective way to illustrate and emphasize a focal area of a 3D model. In mobileVis, we implement an offset examination mode in which user-selected objects are “zoomed” towards the viewer gradually and interactively. Besides spatial displacement, we also magnify the picked objects. Furthermore, users can perform transformations on the offset objects for close examination of the parts while other objects in the model are not affected. An example of offset examination mode is presented in the right image of Figure 5.

3.2.7 Peeling and animation

Traditional artists visualize procedural information, such as assembly and disassembly sequences of a CAD model, using either exploded views with sequence numbers and text explanations of procedural steps or arrow procedural views where arrows are drawn from the image of one step to that of the next. In computer graphics, when an interactive frame rate of rendering is achievable, animations are commonly used. In our mobileVis system, we implemented an animation mode to visualize procedure information. Users can specify an assembly/disassembly sequence by entering a “peeling order” in which objects of a model can be interactively selected and removed one by one by clicking and dragging using a stylus or a mouse. The system records the sequence and can play it back upon request.

3.3 Annotations

For annotation purposes, a dynamic labeling system has been integrated into mobileVis (see Figure 6). A label is defined as a tuple consisting of the label text, an anchor position on the corresponding model, and a position in screen space where the label is located. Since mobileVis is an interactive system, the labels must dynamically reposition themselves as the view and objects change.

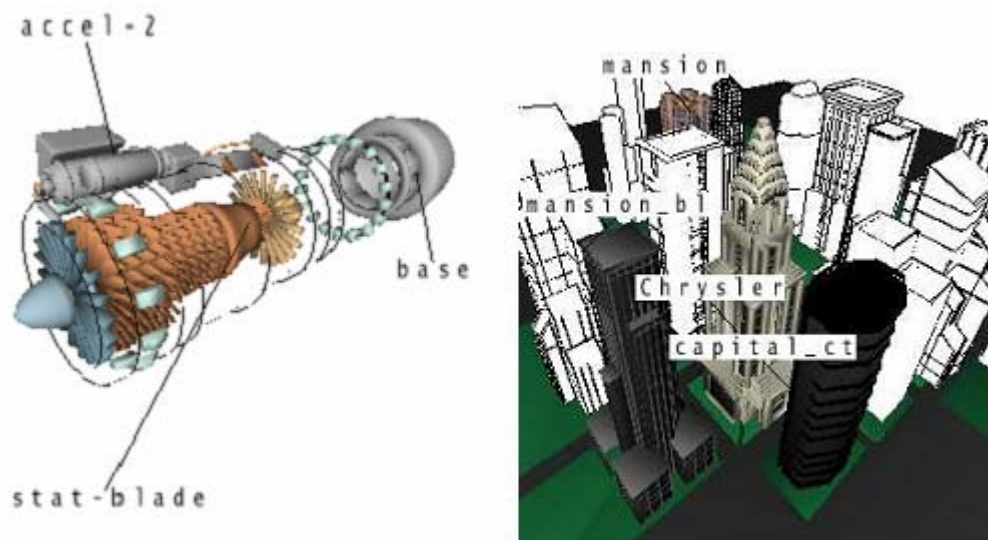


Figure 6: Labeling of a jet engine in “ring-based” mode (left) and a New York City model in “internal” mode (right). Both were rendered in MobileVis on a PDA

Label parsing. The labeling system can parse label data both from manually created XML files or from 3DS models directly. When using manually defined data, the user must define the anchor points on the object to which the labels refer, in addition to the label text. If the labels are parsed directly from the 3DS model, the anchor position is calculated by taking a centroid of the respective object part and placing the anchor at the centroid.

Viewing modes. Labels can be viewed in three separate viewing modes: “internal,” “ring-based,” and “flush-left/right.” Using the internal viewing mode, labels are placed relatively close to the anchor point. Label-label occlusions are avoided by detecting “clusters” of labels and repositioning those clusters using an optimization procedure. The ring-based viewing mode calculates a circular extent about the viewing object. The extent is based off the 2D projections of the points on the bounding box of the object and is dynamically updated as the view changes. Labels are uniformly positioned on the ring by determining the closest position on the ring to the projected anchor point. The flush-left/right mode uses a similar technique as the ring-based mode to position labels, except the labels are uniformly distributed on the leftmost/rightmost extents of the bounding box.

Resolution dependant labels. Since the labeling system may be used on displays with varying resolutions, the label size is defined as a function of the screen resolution being used. This is a reasonably simple procedure, since the labels are texture-mapped and can easily be scaled with respect to display size.



Figure 7-1: Photo of our system running on a PDA

3.4 Performance

We tested our system on a Dell Axim x51v PDA with a 240x320 screen resolution, an Intel 624MHz XScale CPU, 64Mb of RAM, 16Mb of video memory and Microsoft Windows Mobile 5.0. For this PDA, after the operating system booted, the memory available for running applications was approximately 30Mb. Figure 7-1 and Figure 7-2 show photos of our system running on the PDA.

Table 1 presents the number of vertices and triangles in our 3D test models and the performance of our system in different rendering modes. Among the testing models we used, “APU” is an auxiliary power unit from a Boeing 777 dataset, courtesy of Boeing Company, and “B-part” is also a mechanical part from this dataset. Cutaway view images of these two models are shown in Figure 3.

In the first column of Table 1, several primitive rendering modes, including point and surface modes, and various illustrative rendering modes are listed. In the selective rendering mode, to obtain the performance statistics in the table, we defined a major occluding object of a 3D model to be in silhouette mode and the other parts in surface mode. The left image of Figure 2 shows an example of the setup. Similarly, in the ghosted viewing mode in the table, opacity modulation at the vertices was only applied to a relatively large occluding object in a model, even though our system allows modulations to be applied to multiple objects simultaneously.

In the cutaway view mode, two clipping planes were used and the models were drawn in surface mode. Since the number of rendering passes in the cutaway view mode is the same as the number of clipping planes that are enabled, the performance of our system under the cutaway mode varies depending on the total number of activated clipping planes. In the offset examination view mode, one or two feature parts of a model were selected and offset for close examination.



Figure 7-2: Photo of our system running on a PDA

Figure 8(a) plots the rendering performance for 3D models in varying rendering modes on our test PDA. As the size of a model increases, the frame rate drops. On a PDA, in addition to low CPU clock rate, limited memory, and slow bus speed, one of the significant factors that directly affects rendering performance is the size of the instruction and data caches. If the code and data being executed fits into the cache, the system is able to operate at the full clock speed of the CPU. Otherwise, it will constantly access memory and cannot execute at full speed. The PDA we used has an Intel XScale CPU with a 32Kb instruction cache and a 32Kb data cache. As mentioned in Section 3.1, the vertex data structure implemented in our system requires 16 bytes per vertex. Therefore, the geometry buffer with more than 2048 vertices will overflow the data cache. Additionally, since we have to use vertex and

color arrays for all the rendering tasks with the OpenGL ES API, we keep a list of vertex indices for all the objects in a model. This vertex index list is constantly called upon in rendering and, thus, also affects the performance.

Table 1: Frame rates of rendering 3D models in different rendering modes

Models	Chair	Bike	Jet Engine	APU	B-part
#vertices	2,366	6,112	19,666	35,304	43,030
#triangles	4,077	11,632	37,368	52,212	60,391
point mode (fps)	55.6	23.8	7.8	4.3	3.6
surface mode (fps)	41.6	19.5	5.6	3.9	2.8
silhouette mode (fps)	11.9	4.3	1.4	0.8	0.6
selective render. (fps)	26.3	12.5	3.4	2.2	1.4
cutaway view (fps)	35.5	14.9	4.9	3.5	2.2
ghosted view (fps)	40.1	16.9	5.3	3.9	2.7
magic lens (fps)	18.5	8.1	2.3	1.9	1.3
offset exam (fps)	37.5	17.9	5.5	3.8	2.8

From Table 1, we can see that the silhouette rendering mode is relatively slow compared to other modes because of the view-dependent computations required to find potential silhouette edges in Step 2 of the silhouette algorithm (described in Section 3.2.1). A full traversal of the mesh is needed at this step and, subsequently, in Step 3, two traversals over the mesh are performed. Besides the multiple-pass rendering, limited memory and cache sizes aggravate the situation and further deteriorate the performance.

In addition to running our system on the test PDA, we also tested our system on a desktop platform using OpenGL 2.0. The desktop we used has a 1.6GHz Pentium M CPU, 512Mb of RAM and an nVidia GeForce 6200 graphics card. Figure 8(b) is a plot of the rendering performance of our system on the desktop PC using OpenGL 2.0. The frame rates on the desktop are generally 2 to 9 times higher than those on the PDA.

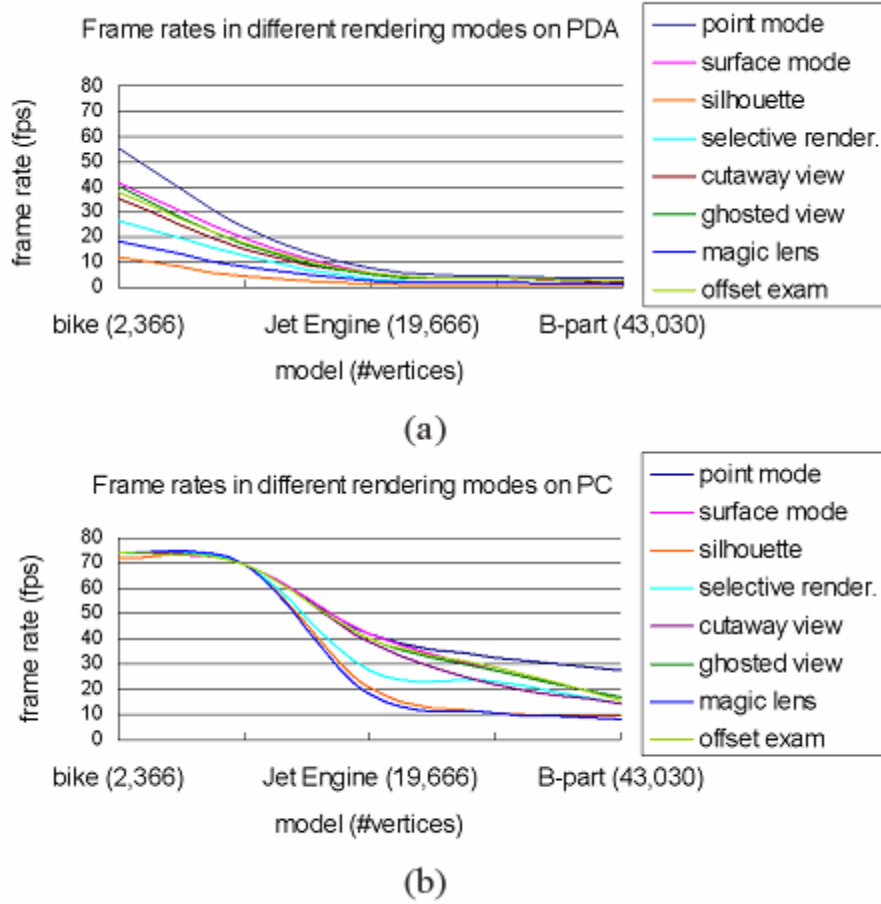


Figure 8: Performance in different rendering modes on a PDA (a) and a desktop (b)

4. Preliminary User Experiment

There have been few previous attempts to analyze the perceptual impact and effectiveness of rendering styles. Evaluation of visual representations can be performed at component, system, or work environment level [6]. At the component level, a common practice is to use controlled experiments where selected visualization components are systematically varied and other input components are kept consistent. Effectiveness, efficiency, scalability and user satisfaction are the standard metrics for usability evaluation. For our empirical user study, we designed experiments that evaluated the effectiveness of different illustrative rendering styles in mobileVis. The *effectiveness* is defined as the capability of a rendering style to enhance the user's ability to understand the internal structures of 3D models and complete typical tasks in mechanical training and maintenance. Quantitative measurements, including both user response time and the number of operations during task completion, were used to gain additional insight on the perceptual impact that different illustrative rendering styles had on user performance in the mobile environment.

As described in previous sections, our system consists of low-level primitive illustration styles such as silhouette, transparency, surface and points, as well as

intermediate-level rendering modes including selective rendering, cutaway view, ghosted view, offset examination, peeling view, and animation. Furthermore, some of the intermediate-level rendering modes can be combined to achieve more complex rendering effects. Our user study mainly focused on the effect of the intermediate-level rendering modes on user perception.

4.1 Experimental Design

Finding an object and identifying the connected components of two objects are fundamental to mechanical training and maintenance tasks, such as determining interference, checking proper placement, and explaining details to coworkers [7]. Between the two tasks, finding an object is a prerequisite for identifying the connected components of objects and, thus, more fundamental. For evaluating the effectiveness of illustrative rendering styles in mobileVis, we asked subjects to find a mechanical part in a complex 3D CAD model that is interactively rendered on a mobile device.

4.2 Apparatus

We used a Dell Axim x51v PDA with a screen resolution of 240x320, an Intel 624MHz XScale CPU, a 16 Mb video memory, a 64Mb RAM and a 256 Mb ROM for our experiments.

4.3 Subjects

In our preliminary user experiment, we selected 5 subjects (3 males and 2 females) for evaluation. They were recruited on a voluntary basis. They were all 20-30 years old. Among the 5 subjects, one of them had graphics programming experience before and, thus, has some knowledge of illustrative rendering. Four of the subjects had either an engineering or information science background. None of the subjects had previous experience in CAD modeling of mechanical parts. In addition, none of the subjects used a PDA or smartphone frequently.

Table 2: Models and their target parts

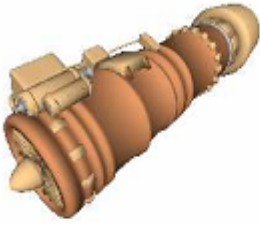
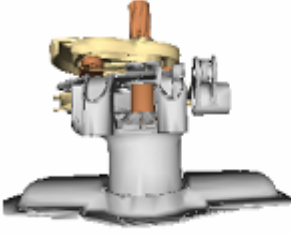
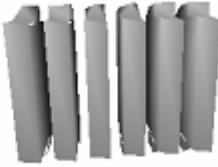
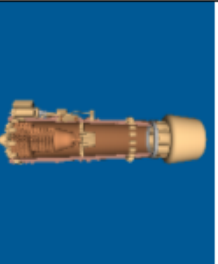
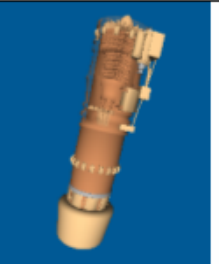
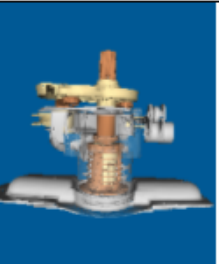
	Model	Part to be found
1		
2		
3		
4		

Table 3: Test cases in preliminary user experiments

Model	Selective render.	Magic lens	Selective cutaway	Ghosted view
Jet engine.				
	C0	C1	C2	C3
B777 part I				
	C4	C5	C6	C7
B777 part II				
	C8	C9	C10	C11
V8				
	C12	C13	C14	C15

4.4 Stimuli

In our experiment, four CAD models were shown to the subjects: a V8 engine model, a jet engine model and two parts from a Boeing 777 dataset. For each of the four models, a surface-rendered overview image and that of a target interior component to be found in tests were generated and presented to the subjects, as shown in Table 2. Four rendering styles – selective rendering, magic lens view, cutaway view and ghosted view – were tested. The four models were each rendered in the four rendering styles and, thus, created 16 test cases as shown in Table 3. A subject needed to interact with all 16 test cases to complete the experiment. The appearance sequence of the test cases was randomized.

For each test case, the starting viewpoint and rendering styles was defined by a set of parameters generated in a pre-processing step and stored on the test device. When generating the parameters, we ensured that the target component in a 3D model was viewable from certain angles and directly selectable. We also ensured that the initial display angle of the model resulted in the target object being occluded. Subjects had to interact with the system by rotating or zooming the model or panning a magic lens to see the target component. Additionally, the total number of required operations was designed to be approximately the same over all the test cases.

4.5 Procedure

Subjects were first introduced to the experimental setup. The experimental procedures were explained, as well as how to interact with the rendering toolkits of mobileVis. We demonstrated how to use different rendering styles to visualize 3D models. The models we used for demonstration were different from the ones used in real tests. The subjects interacted with the system and their questions on the usage of the rendering toolkits were answered. The subjects were also instructed that in real tests, their response time, number of operations on the system and the accuracy of their selections would be recorded. We began the real tests when the subjects stated that they understood the tasks and felt comfortable using the system.

In the real tests, Table 2 was presented to the subjects on a piece of paper. The images in the Table were used as references in the tests so that the subjects understood which component of a model they were expected to locate. For each test case in Table 3, the subjects needed to find the target component in a model, enable a “Pick” button, select the target object, and click a “Next” button to proceed to the next test case.

After finishing the real tests, the subjects were asked to complete a post-experiment questionnaire where they: 1) rated the effectiveness of different rendering styles viewed in the system on a 1-5 scale with 5 indicating the style that was most effective in helping to identify target components and 1 for the least effective; and 2) provided background information, such as if they had 3D modeling experience or frequently used a PDA or smartphone, etc.

4.6 Data Analysis

We conducted ANOVA tests on the collected data to analyze the statistical significance of rendering styles for the two measures, response time and the number of operations. The ANOVA tests showed that the effect of a rendering style on a subject's response time and the number of operations were weakly statistically significant with $p=0.0949$ and $p=0.1191$, respectively.

We analyzed the factors that can significantly confound the experimental results for analyzing the effect of rendering styles. The following factors were found:

Complexity of a model. The difference in the complexities of models affects the subjects' response time in a statistically significant way ($p < 0.0001$). The significance is much higher than for the rendering styles ($p = 0.0949$) in our preliminary experimental results. This indicates that in order to show the effect of rendering styles more significantly, the difference in the complexities of models needed to be reduced.

Training effects. From our results, we noticed that subjects generally complete the tasks faster on the later test cases than on the earlier ones for a given rendering style. This indicates that the subjects were in a learning stage when interacting with the earlier test cases. Therefore, a formal training session should be added before the real tests begin.

5. Refined User Experiment

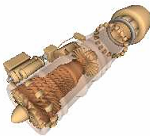
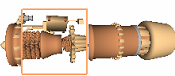
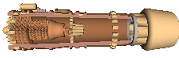
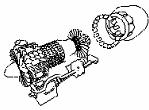
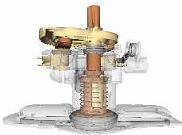
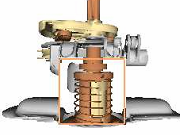
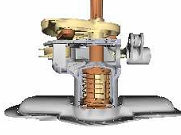
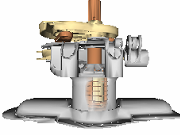
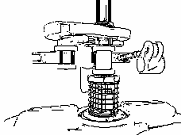
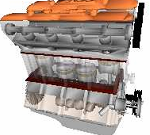
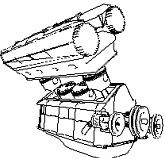
We refined our experimental design based on the results from the preliminary experiments and recruited more volunteers to perform the updated experiments.

5.1 Stimuli

In the refined experiment, we still used the four CAD models in Table 2 and their target components. However, three of the models, including the V8 engine model, the jet engine model and the part from a Boeing 777 dataset, were only used for the training session. In the real tests, Model 4 in Table 2 was used. When generating test cases, we slightly varied the locations of the interior components of Model 4 across the test cases. In the training and real tests, the images in Table 2 were presented to subjects for reference.

In the training session of our experiment, Models 1-3 in Table 2 were visualized in the same five rendering styles that subjects would see in the real tests (see Table 4). Besides the four rendering styles tested in our preliminary experimental design, we added the outline rendering style for testing. In the actual tests, Model 4 was rendered in the five styles with each style repeated for four test cases. In each test case, the locations of the interior components of the model changed slightly. The complexities of the varying models were maintained at a similar level and the frame rates for all test cases were also kept approximately the same. Therefore, a subject needed to interact with 20 test cases to complete the real test. The appearance sequence of rendering styles in these cases was randomized using the magic square [8]. Table 5 shows representative rendering cases used in the real tests.

Table 4: Rendering cases for training in refined experiments

	Selective render.	Magic lens	Cutaway	Ghosted view	Outline
1					
2					
3					

5.2 Subjects

In our refined experiment, we had 7 subjects (5 males and 2 females) for evaluation. Their ages are all within 20-40 years old. Among the 7 subjects, six of them had graphics programming experience before. One of the subjects had neither an engineering nor a science background. In addition, two of the subjects used a PDA or smartphone frequently. In our experiment, we asked the subjects to first perform the training tasks. The subjects who could not complete simple navigation tasks such as rotation, translation and zooming of a model, or complete the training tasks within a fixed time frame on the test device were identified as outliers.

5.3 Procedure

Subjects were first introduced to the experimental setup. We explained the experimental procedures and demonstrated how to use mobileVis. Then users were asked to go through a training session that had the same procedure as the real tests but used the training models (see Table 4). For each test case in both the training session and real tests, the subjects needed to find the target component within the model, select the target, and proceed to the next case. The subjects were instructed that in the real tests, their response time, number of operations and the accuracy of their selections would be recorded with no break between performing the tasks. The real tests began after the training session was completed and when the subjects stated that they felt comfortable using the system.

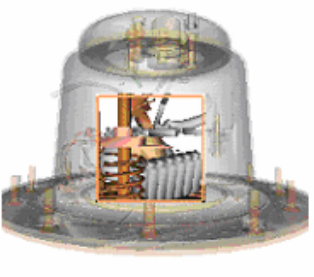
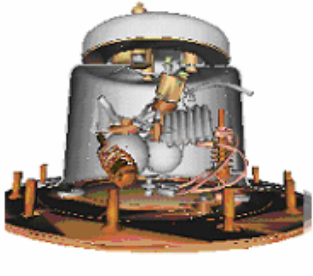
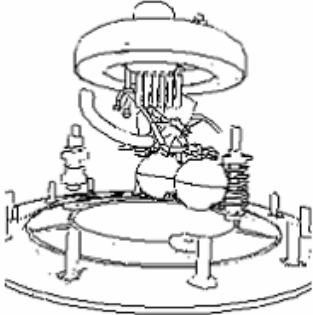
As in the preliminary experiments, after the real tests, the subjects were asked to fill out a post-experiment questionnaire to collect their subjective evaluations on the effectiveness of rendering styles and also their background information.

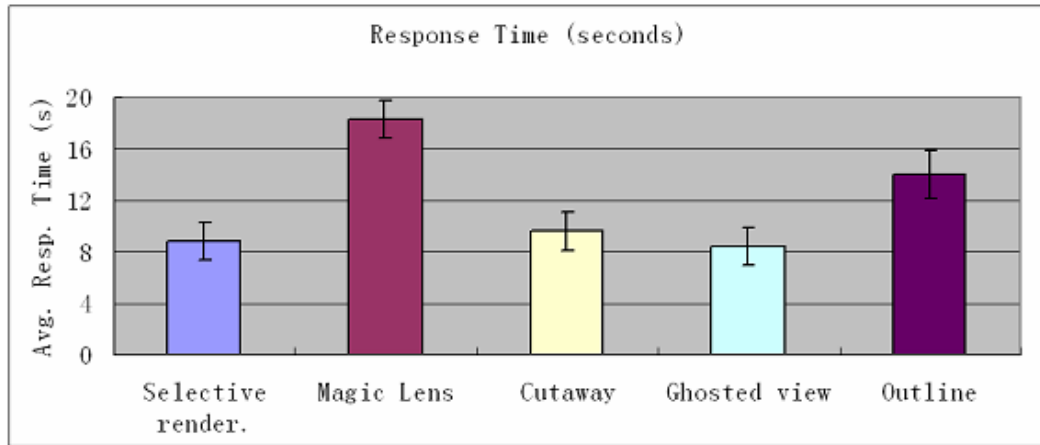
5.4 Data Analysis

We performed ANOVA tests on the data collected from the refined experiment. The ANOVA tests showed that a rendering style had a statistically significant effect on a subject's response time ($p < 0.0001$) and the number of operations needed for task completion ($p < 0.0341$). Moreover, the averages and standard errors of response time and number of operations (see Figure 9(a) and 9(b)) indicate that the overall statistical significance is mainly due to the differences between the three categories of rendering styles. In the first category were selective rendering, cutaway view, and ghosted view. They were all similarly effective in assisting users in finding target components in the test 3D model. Compared to these three styles, outline rendering was in a less effective category and magic lens was in the least effective category.

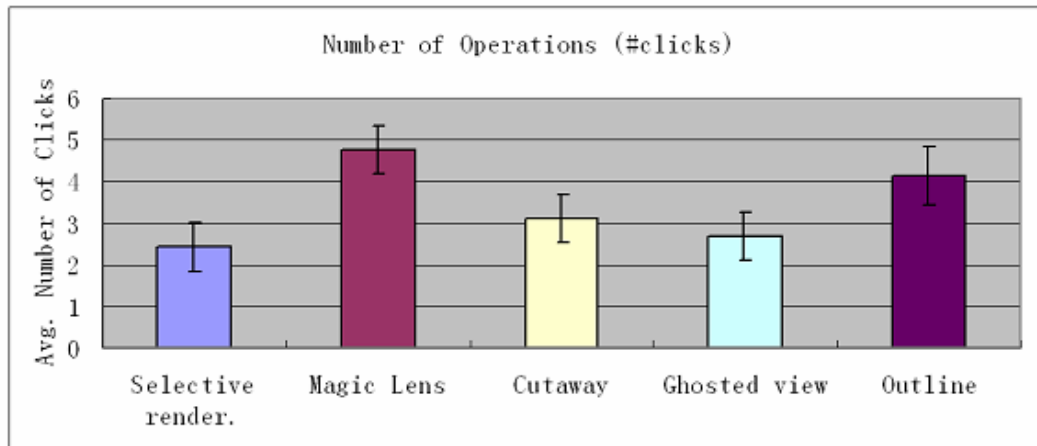
Figure 9(c) shows the subjective ratings of the effectiveness of the five rendering styles. The ratings roughly match the results in Figure 9(a). Selective rendering, cutaway and ghosted view were evaluated as more effective rendering styles than the other two styles: magic lens and outline view. However, a discrepancy exists in that although subjects generally prefer a cutaway view to a ghosted view, for the model used in our experiment, the latter allowed subjects to locate the target objects faster than the former.

Table 5: Representative rendering cases in real tests

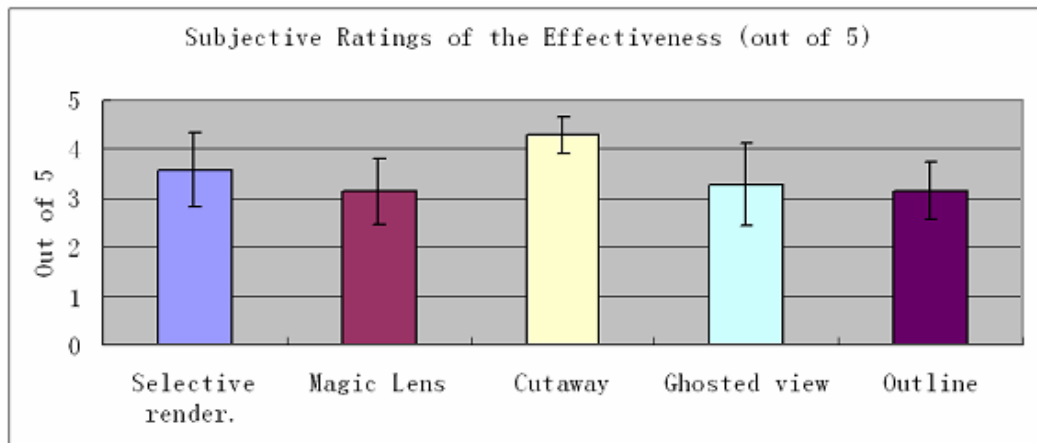
Selective Rendering	Magic Lens
	
Cutaway view	Ghosted view
	
Outline	
	



(a)



(b)



(c)

Figure 9: Results of our refined experiments

6. Final User Evaluation Study Design

The results from our refined experiments are encouraging. We plan to use this experimental design for a final user evaluation study and conduct the experiments on more subjects (approximately 15-20 subjects) in January 2007. We will correct a detected problem in inconsistent frame rates in one of the rendering modes, rerun the experiment on these new subjects recruited from undergraduate ECE classes, and perform the ANOVA analysis to determine statistical significance of factors in terms of user response time, number of operations to complete task, and task accuracy. We will also perform pair-wise comparison of each rendering style on the above three factors as well as qualitative evaluation.

The results of this user study will guide the implementation of our future work on semi-automatic determination of rendering styles based on task types and user characteristics.

7. Conclusions

This project was successful in developing new techniques for effectively displaying information for environments ranging from job training to maintaining and repairing equipment to responding to critical situations. The techniques developed allow the adaptable display of digital model data from the desktop to mobile devices using illustrative rendering techniques in the mobileVis system. The mobileVis system has many illustrative rendering styles that can be selectively applied to more clearly and succinctly show important information, including silhouetting, selective rendering, cutaway views, ghosted views, magic lens viewing, offset examination, object peeling, animation recording and playback, and labeling. The project also performed an initial evaluation of the effectiveness of illustrative rendering methods for mobile devices for the task of locating a specific mechanical part in an assembly for training or maintenance. The promising results of this pilot evaluation has confirmed that the evaluation experiment design is valid and a full user study will be conducted using this experimental design. The results of this project provide guidance for the use of different rendering styles in displaying mechanical assemblies on mobile devices and can be used as a building block for task-adapted electronic maintenance and repair display in the future.

8. References

- [1] Astle, D. and Durnil, D. *OpenGL ES Game Development*. Course Technology PTR, 2004.
- [2] Buchanan J. W. and Sousa M. C. *The edge buffer: a data structure for easy silhouette rendering*. In NPAR '00, pp. 39-42, 2000.
- [3] Diepstraten, J., Weiskopf, D. and Ertl, T. *Interactive Cutaway Illustrations*, Computer Graphics Forum, 22(3), pp. 523-532, 2003.
- [4] Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S. and Strothotte, T. *A developer's guide to silhouette algorithms for polygonal model*. IEEE Computer Graphics and Applications, 23(4), pp. 28-37, 2003.
- [5] Raskar, R. and Cohen, M. *Image precision silhouette edges*. In Proceedings of the 1999 Symposium on Interactive 3D Graphics, SI3D '99, (Atlanta, Georgia, April 26 - 29, 1999). ACM Press: NY, pp. 135-140, 1999.
- [6] Thomas, J. J. and Cook, K. A. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Center, 2005.
- [7] Kasik, D., Troy, J., Amorosi, S., Murray, M., Swamy, S., Co, B. and Seattle, W. *Evaluating graphics displays for complex 3D models*, IEEE Computer Graphics and Applications, 22(3), pp. 56-64, 2002.
- [8] Magic square. <http://mathworld.wolfram.com/MagicSquare.html>. Last accessed: January, 2007.

Appendix A: Interactive Illustrative Rendering on Mobile Devices

Paper to appear in *IEEE Computer Graphics and Applications*, 2007, Approved for Public Release: AFRL-WS-06-1469.

Interactive Illustrative Rendering on Mobile Devices

Jingshu Huang*

Brian Bue*

Avin Pattath*

David S. Ebert*

Krystal Thomas†

* Purdue University Rendering and Perceptualization Lab (PURPL)
Purdue University, West Lafayette, IN

† U.S. Air Force Research Laboratory
Wright-Patterson AFB, OH

ABSTRACT

Illustrative rendering is a widely used visualization technique to display conceptual information, describe problems and give insight to solve them efficiently in science, engineering and the arts. Providing users with automated tools to generate illustrations at will is a challenging problem. Adapting illustrative rendering techniques from desktop platforms to mobile devices creates many hardware and software issues. In this paper, we discuss adaptations of different illustration techniques for rendering 3D models directly on mobile devices for education and training purposes. The implementations of these illustration techniques address the limitations widely encountered in low-end devices. An interactive mobile graphical and textual rendering system with a toolkit of different illustration modes has been implemented. The toolkit we propose allows users to view interior structures of 3D models and instructional procedures on mobile devices.

Keywords: mobile computing, mobile devices, visualization, illustration, CAD

1 INTRODUCTION

Illustrations have been widely used in design, training and education in science, engineering and the arts. People often depend on these visual techniques to display conceptual information, describe problems better, and solve them in reduced time. Providing users with visual tools for automating the illustration process, in addition to accessing to these tools at will, has been an important and challenging problem in computer graphics. Many advanced rendering techniques have been developed on desktop platforms to facilitate the generation of illustrations. However, adapting these techniques to mobile platforms is challenging. Compared to their desktop counterparts, mobile devices have many hardware limitations including low screen resolution, limited input interfaces and battery life, low bandwidth of system bus, slow CPU clock speed, limited storage capacity, and lack of advanced graphics hardware.

Researchers and developers have been trying to address the hardware issues on mobile devices with remote visualization, where a client-server model is adopted and most of the rendering tasks are performed server-side while the mobile client only handles the receipt of streamed images [6] [11]. This approach has been most

popular for displaying complex 3D surface models and volumetric data on mobile platforms. It relies on a network connection that may not always be available or networking bandwidth may be inadequate to achieve a satisfactory performance on remote clients.

In recent years, computational power of mobile devices such as PDAs and cellular phones has significantly increased. The continuous improvements in processors, graphic chips, displays, power management and wireless technology have greatly enhanced the programmability and usability of mobile devices. Moreover, some standardized software platforms for developing 3D graphical applications on mobile devices such as OpenGL ES and Direct3D Mobile have been released and function as subsets of their ancestral, desktop application programming interfaces (API). With these improvements in both hardware and software platforms, performing visualization tasks - including interactive illustrations - using local resources on mobile devices has become more feasible, but still poses difficulties.

In this paper, we discuss the adaptation of advanced illustrative rendering techniques such as interactive cutaway views, ghosted views, silhouettes, and selective rendering on mobile devices. We present our interactive, mobile, illustrative, 3D graphics and text rendering system, *MobileVis*, that allows users to explore the interior structures of 3D models, display the annotations of parts, and visualize instructions, such as assembly and disassembly procedures for mechanical models.

This paper is organized as follows. Section 2 and Section 3 summarize related work in visualization on mobile devices and illustration techniques. Section 4 presents a detailed description of data structures and the different modes in the illustration toolkit in *MobileVis*, including silhouette mode, selective rendering, cutaway and ghosted views, magic lens mode, offset examination mode, animations and labeling. Section 5 describes the performance of these modes on 3D models with varying complexity and Section 6 presents conclusions and discussions.

2 ILLUSTRATIONS IN TRADITIONAL VISUALIZATION

A wide variety of illustrative techniques are adopted in technical manuals, scientific illustrations, textbooks, and encyclopedias (see Table 1). These illustrative techniques help to convey the shapes and forms of objects, reveal the interior structure of complicated models, and describe procedural steps. Inspired by these techniques, researchers in computer graphics have developed many rendering algorithms to automate some of these illustrative conventions on desktop platforms [5] [4]. Viola et al. [12] summarize some current techniques for visualizing the internal structures of volumetric models, including cutaway and ghosted views, importance driven rendering, and exploded and deformation views. How-

*e-mail: {jhuang2|bbue|apattath|ebertd}@purdue.edu

†e-mail:krystal.thomas@wpafb.af.mil

Table 1: Traditional Illustration Styles.



Images (a)-(e) are courtesy of Kevin Hulsey Illustration, Inc.

ever, all of these existing techniques require recent graphics cards, processors, and programming interfaces, which are readily available on desktop platforms, but unavailable on mobile devices like PDAs and cellular phones.

This gap in computational capabilities will continue to be a problem. Battery capacities for mobile devices generally improve at a much slower rate than CPU and graphics chips, which forces hardware, especially processing units, to be clocked at much lower rates.

Despite the issues with the computational gap, many illustrative rendering algorithms originally developed for desktop platforms need to be adapted to mobile devices in order to provide users with the same functionality they have enjoyed with desktop 3D graphics applications.

3 LOCAL VISUALIZATION ON MOBILE DEVICES

Compared to the quantity of literature that discusses remote visualization for mobile devices, the amount of published work on local 3D visualization on mobile devices is limited. This is largely due to the lack of development of mobile devices for graphical applications. Until a few years ago, most handheld devices were hardly capable of rendering a Gouraud-shaded cube, let alone performing demanding and complicated visualization tasks directly on the devices. However, mobile devices have now developed to the point

where direct 3D rendering at interactive rates is feasible. With the first standardized 3D programming interfaces for mobile devices - OpenGL ES - released, utilizing OpenGL ES and Mobile 3D Graphics (M3G) for fast development of 3D Java games is becoming more common in the mobile game industry. Research and development on mobile 3D visualization applications, however, falls far behind. Pocket Cortona [9] is one of the few available viewers for visualizing 3D models on PDAs, and has been used by researchers in construction [8]. The software requires VRML models and has limited surface shading modes for rendering. Researchers at IBM developed a 3D CAD model viewer for product data management on PDAs [3]. This system consists of a file format converter, a scene graph library, and a triangle rasterizer. However, the rendering modes for viewing 3D models are also limited to surface and wireframe.

4 IMPLEMENTATION

Our mobile rendering system *MobileVis* was developed in C/C++ using OpenGL ES 1.0. OpenGL ES is a subset of the OpenGL API designed for embedded systems. In OpenGL ES, many functions have been removed from the original OpenGL API and a few added. The most significant difference between OpenGL ES and OpenGL is the introduction of fixed-point data types. The fixed point data types support the computational capabilities of embedded processors, which often lack floating point processing units. In addition, the removal of the `glBegin()/glEnd()` semantics in OpenGL ES is another big distinction between these two sets of 3D graphics APIs. OpenGL ES favors vertex arrays, which generate less overhead than `glBegin()/glEnd()` for large datasets. Many computationally intensive features in OpenGL, such as multi-texturing and retrieval of dynamic OpenGL states, are either removed or are optional in OpenGL ES, depending on the implementation. OpenGL ES provides us with a lightweight interface for developing 3D graphics applications on low-end devices.

Our system consists of three modules, a data transcoder that converts 3DS MAX models into a customized binary format for fast loading of the models on mobile devices, a renderer equipped with a toolbox of different illustrative rendering modes, and a labeling system which allows a dynamic display of labels and text information associated with parts of a 3D model.

4.1 Data Structures and Transcoder

Since most mobile devices do not have floating point processing units, we have created a data transcoder that converts 3D models from floating point to a binary format using only signed and unsigned short integers. The data at each vertex consist of a position (x, y, z) , a normal (n_x, n_y, n_z) , and a color / opacity (r, g, b, a) . Each component of the position and normal is represented by a 2-byte signed short integer and each component of the color / opacity is represented by a byte. Thus, a total of 16 bytes are needed per vertex. To minimize floating point operations, all the computations, including the transformations of projection and model view matrices, are performed using fixed point math [1].

Furthermore, for performance concerns, OpenGL ES has no support for `glBegin()/glEnd()` and requires all the graphics to be performed using vertex arrays. In order to minimize the number of duplicate vertices sent to the transformation stage of the graphics pipeline, we use indexed vertex arrays and thus maintain a list of vertex indices following the order of a triangle list in the original model. Each element in this new vertex index list is represented as an unsigned short integer.

In order to retain hierarchical position information of different parts of an original 3D model, we process the model in a pre-defined order and store the converted vertex array data sequentially. To

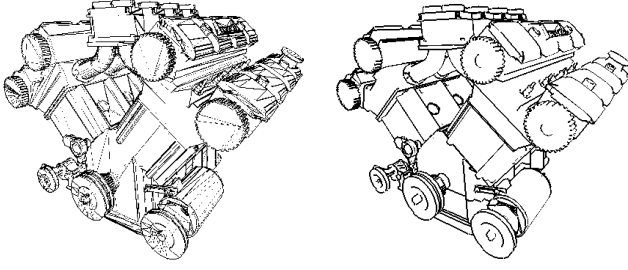


Figure 1: Modified 2-pass silhouette algorithm employed onto a V8 engine dataset rendered on a PDA without a bit buffer (left) and with a bit buffer (right).

identify the vertex indices of an object in the vertex index list, we keep an array of the number of triangles of each object in the model. With this array, we can easily locate the vertex indices of a selected object in the vertex index list by summing the number of triangles of all the objects stored prior to the selected object.

4.2 Interactive Illustrative Rendering

We have developed a toolkit for interactive exploration of 3D polygonal models on mobile devices. The toolkit includes the following illustrative rendering styles: silhouettes, selective rendering, cut-away views, ghosted views, magic lens views, offset examination views, and animations.

Silhouettes. Silhouettes are important illustrative rendering techniques that convey shape and spatial relationships for 3D objects. With polygonal models, silhouettes are defined as the edges in the mesh that share a front- and a back-facing polygon. Isenberg et al. [7] summarize existing silhouette algorithms and categorize them into three groups - image-space, object-space and hybrid. When interactive frame rates are desired, and when the devices have limited memory or slow processors, image-space and hybrid algorithms are recommended. However, image-space algorithms usually require reading and operating on a z-buffer which is not accessible in OpenGL ES profile. Therefore, we implemented a modified hybrid algorithm based upon the Raskar and Cohen two-pass silhouette algorithm [10].

Because line drawing via polygon mode `glPolygonMode()` is not available in OpenGL ES, we rendered polygons in wireframe mode in the second pass of the silhouette algorithm. However, this adaptation generates line artifacts on relatively large, flat surfaces as shown in the left image of Figure 1. These lines are wireframe edges in the polygonal mesh. To remove the artifacts, we employed a silhouette detection technique modified from the edge buffer method described in [2]. Rather than using 2 bits per edge as proposed in the original algorithm, we stored only 1 bit per edge and our 2-pass silhouette algorithm works in the following manner:

1. Initialize all the bits in the bit buffer for edges to 0
2. Identify and mark silhouette edges by going through all of the mesh edges and flipping bits corresponding to all of the edges of backfacing triangles. By doing this, the bits that correspond to silhouette edges will have a resulting value of 1 because they are flipped only once, whereas edges on the backside will have their bits flipped twice (and become 0) and the edges on front side will have their bits unmodified (and remain 0).
3. Perform two-pass rendering. In the first pass, draw front facing polygons with a slight offset of the polygon nearest to the viewer, and set the mask of depth buffer to TRUE and that of frame buffer to FALSE; in the second pass, change the masks

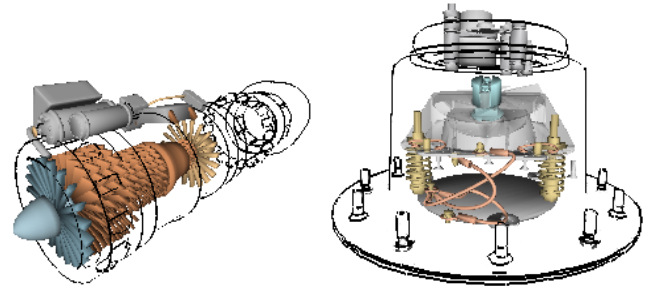


Figure 2: Selective rendering of a jet engine (left) and a Boeing 777 auxiliary power unit (right) on a PDA.

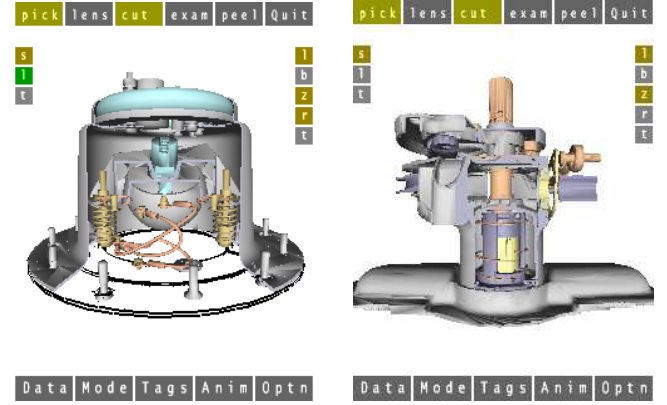


Figure 3: Cutaway views of Boeing 777 parts. Left: 3 clipping planes are enabled. Right: 2 clipping planes are enabled. Both were rendered in MobileVis on a PDA.

of both depth and frame buffers, and draw the silhouette edges found in the previous step with a pass condition in the depth test set to “less than or equal to.”

Selective rendering. Silhouettes allow users to perceive the shapes of objects easily. By combining silhouettes with other rendering styles, it is easy to create a focus view of important features in a model while still conveying the overall shapes and context information. Figure 2 shows an effective use of a mixture of silhouette rendering mode for the overall shapes of the 3D models with surface and transparency rendering modes applied to interior feature objects. Both images were rendered in our *MobileVis* system on a PDA with 240x320 display resolution. Users first specify a rendering style and then select the objects that need to be rendered in the selected style by clicking on the objects. Because selection mode is not supported intuitively in OpenGL ES, we implement picking by rendering object IDs into the frame buffer with depth test enabled and then read the color buffer back.

Cutaway views. For cutaway illustrations, different approaches for desktop platforms exist, such as constructive solid geometry (CSG)-based cutouts, stencil buffer-based cutouts and texture-based cutouts [4]. However, these techniques are either too computationally expensive for mobile devices (e.g., CSG operations) or require a stencil buffer or hardware-accelerated fragment operations, which are not available on most mobile devices (although OpenGL ES 1.0 supports the stencil test), including the testing device we use: a Dell Axim x51v PDA. Therefore, we implemented a selective planar cutout technique for cutaway illustration in *MobileVis*. We provide users with an interface to dynamically enable and disable n ($1 \leq n \leq 5$) clipping planes and control their positions

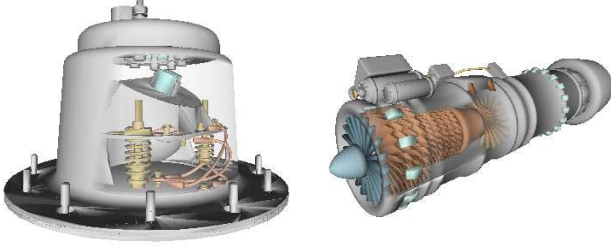


Figure 4: Ghosted views of a Boeing 777 auxiliary power unit (left) and a jet engine (right) in MobileVis running on a PDA.

individually. In the cutaway mode, clipping planes are applied on user-selected objects. These objects are maintained in a cutaway list and rendered n times. Each time only the respective clipping plane is activated. Unpicked objects are rendered only once before rendering any selected objects. Figure 3 shows two examples of cutaway views. In both images, the cutaway views are combined with selective rendering to reveal interior structures of the polygonal models. The capping of the cutout surfaces, that is, the shading of the surfaces formed by a clipping plane and the object the plane clips, was implemented by performing an extra rendering pass on the back faces of the polygonal model with lighting disabled and depth testing enabled.

Ghosted view. Ghosted views are a rendering technique that is similar to cutaway views except that, instead of doing a hard-edge cutaway, opacity in a selected occluding region is reduced gradually to allow a transparent view of interior objects while still preserving features of occluding objects, such as edges. Ghosted views help users to improve their understanding of interior structures and their spatial relations to their contextual objects. In *MobileVis*, we implement ghosted viewing by modulating the opacity values at the vertices near a user-selected point in an object. The opacity α of a vertex in the picked object is modified as:

$$\alpha = \begin{cases} 1, & d \geq r \\ \frac{d}{r}, & \alpha_0 r < d < r \\ \alpha_0, & d \leq \alpha_0 r \end{cases} \quad (1)$$

where d is the distance between the vertex and the central point of the triangle which a user picks, α_0 a predefined minimal opacity value between 0 and 1, e.g. 0.15, $r = \frac{1}{4} \min(dim_x, dim_y, dim_z)$ where dim_x, dim_y, dim_z are the sizes of x, y and z dimensions, respectively. Figure 4 shows an example of ghosted views.

Magic lens. A magic lens can be considered as an extension of a magnifying glass and can be used to illustrate and emphasize the details of a portion of a model. The objects displayed under the lens can be shown in different rendering styles or varying degrees of detail from their context. In *MobileVis*, we magnify the portion of a model under the lens and, furthermore, assign a see-through capability to the magic lens. From our observations of mechanical 3D models, we can easily see that many of these models have a “shell” or covering object, which is usually relatively large in size, and less interesting, that is, contains fewer features than interior objects that the shell occludes. Therefore, for the magic lens view, we define a function which decides if an object should be drawn under the magic lens. If the area that an object covers under the magic lens exceeds a threshold value, it is likely that the object is a shell or covering object and occludes potentially interesting details of a model. In such cases, we remove the portion of the object under the magic lens and display the occluded objects in magnification. Through experiments on most of our testing models, we obtain an optimal threshold value and thus define the visibility function as Equation 2.

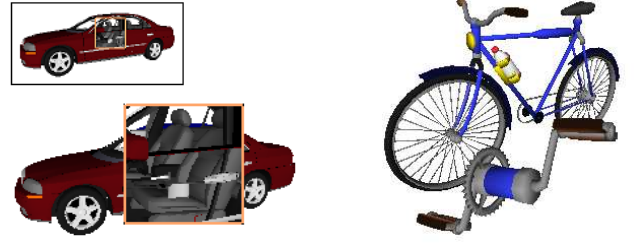


Figure 5: Magic lens view of a car model (left) and offset examination view of a bike model (right) in MobileVis running on a PDA.

$$visibility_i = \begin{cases} 1, & \frac{c_i}{S} \leq 0.3 \\ 0, & \frac{c_i}{S} > 0.3 \end{cases} \quad i = 1 \dots N \quad (2)$$

where c_i the coverage (in pixels) of the object i , and S the area size of the lens. An example of a magic lens view is shown in the left image of Figure 5. In this image, the car body cover on the driver side occludes the view of interior objects in the car. Through a magic lens, the portion of the cover under the lens is defined as an occluding object using Equation 2 and thus removed. The interior structures of the car are exposed and magnified. The magnification factor of the lens in the magnified image is 1.7.

Offset examination. Spatially displacing different parts of a model is a commonly used technique to uncover prominent features inside a model. The offset parts can be either displaced towards the viewer and retain their relative spatial relationships to each other, or be spread out on the plane perpendicular to the viewing vector, thus forming an exploded view (See Table 1). In the first case, the displacement usually provides an effective way to illustrate and emphasize a focal area of a 3D model. In *MobileVis*, we implement an offset examination mode in which user-selected objects are “zoomed” towards the viewer gradually and interactively. Besides spatial displacement, we also magnify the picked objects. Furthermore, users can perform transformations on the offset objects for close examination of the parts while other objects in the model are not affected. An example of offset examination mode is presented in the right image of Figure 5.

Peeling and animation. Traditional artists visualize procedure information such as assembly and disassembly sequences of a CAD model using either exploded views with sequence numbers and text explanations of procedural steps, or via arrow procedural views where arrows are drawn from the image of one step to that of the next (as shown in Table 1). In computer graphics, when an interactive frame rate of rendering is achievable, animations are commonly used. In our *MobileVis* system, we implemented an animation mode to visualize procedure information. Users can specify an assembly/disassembly sequence by entering a “peeling order” in which objects of a model can be interactively selected and removed one by one by clicking and dragging using a stylus or a mouse. The system records the sequence and can play it back upon request.

4.3 Annotations

For annotation purposes, a dynamic labeling system has been integrated into *MobileVis* (see Figure 6). A label is defined as a tuple consisting of the label text, an anchor position on the corresponding model, and a position in screen space where the label is located. Since *MobileVis* is an interactive system, the labels must dynamically reposition themselves as the view and objects change.

Label parsing. The labeling system can parse label data both from manually created XML files, or from 3DS models directly.

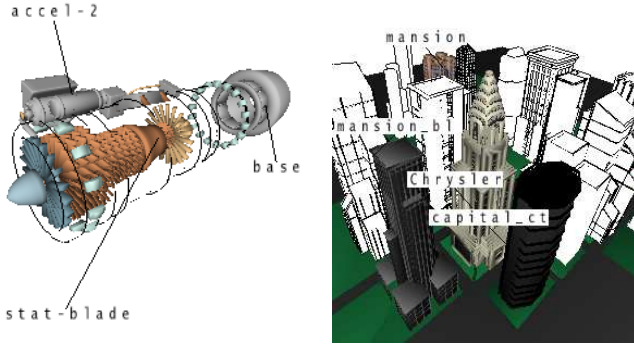


Figure 6: Labeling of a jet engine in “ring-based” mode (left) and a New York City model in “internal” mode (right). Both were rendered in MobileVis on a PDA.

When using manually defined data, the user must define the anchor points on the object to which the labels refer, in addition to the label text. If the labels are parsed directly from the 3ds model, the anchor position is calculated by taking a centroid of the respective object part, and placing the anchor at the centroid.

Viewing modes. Labels can be viewed in three separate viewing modes: “internal,” “ring-based,” and “flush-left/right.”

Using the internal viewing mode, labels are placed relatively close to the anchor point. Label-label occlusions are avoided by detecting “clusters” of labels, and repositioning those clusters using an optimization procedure.

The ring-based viewing mode calculates a circular extent about the viewing object. The extent is based off the 2D projections of the points on the bounding box of the object, and is dynamically updated as the view changes. Labels are uniformly positioned on the ring by determining the closest position on the ring to the projected anchor point.

The flush-left/right mode uses a similar technique as the ring-based mode to position labels, except the labels are uniformly distributed on the leftmost/rightmost extents of the bounding box.

Resolution dependant labels. Since the labeling system may be used on displays with varying resolutions, the label size is defined as a function of the screen resolution being used. This is a reasonably simple procedure, since the labels are texture-mapped and can easily be scaled with respect to display size.

5 RESULTS

We tested our system on a Dell Axim x51v PDA with 240x320 screen resolution, an Intel 624MHz XScale CPU, 64Mb RAM, 16Mb video memory and Microsoft Windows Mobile 5.0. For this PDA, after the operating system booted, the memory available for running applications was around 30Mb. Figure 7 shows photos of our system running on the PDA.

Table 2 presents the number of vertices and triangles in our 3D test models, and the performance of our system in different rendering modes. Among the testing models we used, “APU” is an auxiliary power unit from a Boeing 777 dataset, courtesy of Boeing Company, and “B-part” is also a mechanical part from this dataset. Cutaway view images of these two models are shown in Figure 3.

In the first column of Table 2, several primitive rendering modes, including point and surface modes, and various illustrative rendering modes are listed. In the selective rendering mode, to obtain the performance statistics in the table, we defined a major occluding object of a 3D model to be in silhouette mode and the other parts



(a)



(b)

Figure 7: Photos of our system running on a PDA

in surface mode. The left image of Figure 2 shows an example of the setup. Similarly, in the ghosted viewing mode in the table, opacity modulation at the vertices was only applied to a relatively large occluding object in a model, even though our system allows modulations to be applied to multiple objects simultaneously.

In the cutaway view mode, two clipping planes were used and the models were drawn in surface mode. Since the number of rendering passes in cutaway view is the same as the number of clipping planes that are enabled, the performance of our system under the cutaway mode varies depending on the total number of activated clipping planes. In the offset examination view mode, one or two feature parts of a model were selected and offset for close examination.

Figure 8(a) plots the rendering performance for 3D models in varying rendering modes on our test PDA. As the size of a model increases, the frame rate drops. On a PDA, in addition to low CPU clock rate, limited memory, and slow bus speed, one of the significant factors that directly affect rendering performance is the size of the instruction and data caches. If the code and data that is being executed fits into the cache, the system is able to operate at the full clock speed of the CPU. Otherwise, it will constantly access memory and cannot execute at full speed. The PDA we used has an Intel XScale CPU with 32Kb instruction cache and 32Kb data cache. As we have mentioned in Section 4.1, the vertex data structure we implemented in our system requires 16 bytes per vertex. Therefore, the geometry buffer with more than 2048 vertices will overflow the

Table 2: Frame rates of rendering 3D models in different rendering modes on a PDA

Models	Chair	Bike	Jet Engine	APU	B-part
#vertices	2,366	6,112	19,666	35,304	43,030
#triangles	4,077	11,632	37,368	52,212	60,391
point mode (fps)	55.6	23.8	7.8	4.3	3.6
surface mode (fps)	41.6	19.5	5.6	3.9	2.8
silhouette mode (fps)	11.9	4.3	1.4	0.8	0.6
selective render. (fps)	26.3	12.5	3.4	2.2	1.4
cutaway view (fps)	35.5	14.9	4.9	3.5	2.2
ghosted view (fps)	40.1	16.9	5.3	3.9	2.7
magic lens (fps)	18.5	8.1	2.3	1.9	1.3
offset exam (fps)	37.5	17.9	5.5	3.8	2.8

data cache. Additionally, since we have to use vertex and color arrays for all the rendering tasks with OpenGL ES API, we keep a list of vertex indices for all the objects in a model. This vertex index list is constantly called upon in rendering and thus also affects the performance.

From Table 2, we can see that the silhouette rendering mode is relatively slow compared to other modes because of the view-dependent computations required to find potential silhouette edges in Step 2 of the silhouette algorithm (described in Section 4.2). A full traversal of the mesh is needed at this step and subsequently, in Step 3, two traversals over the mesh are performed. Besides the multiple-pass rendering, limited memory and cache sizes aggravate the situation and further deteriorate the performance.

In addition to running our system on the test PDA, we also tested our system on a desktop platform using OpenGL 2.0. The desktop we used has a 1.6GHz Pentium M CPU, 512Mb RAM and an nVidia GeForce 6200 graphics card. Figure 8(b) is a plot of the rendering performance of our system on the desktop PC using OpenGL 2.0. The frame rates on the desktop are generally 2 to 9 times higher than those on the PDA.

6 CONCLUSION AND FUTURE WORK

In this paper we have presented a graphical and text-based visualization system for mobile devices. We have focused on adapting traditional illustration techniques for rendering 3D models from desktop platforms to mobile devices. In our implementation, low-end devices and their limitations in hardware and software have been addressed. A toolkit of different illustration modes including silhouettes, cutaway views, ghosted views, selective rendering, and labeling has been implemented. We have shown the results of these illustration and labeling techniques, and analyzed the performance issues related to mobile platforms.

For future work, we plan to investigate improving the performance of some of our illustration techniques for mobile devices. Besides that, we also plan to perform a user study to evaluate the effectiveness of different illustration techniques for rendering multi-resolution 3D models for training and maintenance in concert with Air Force Research Laboratories.

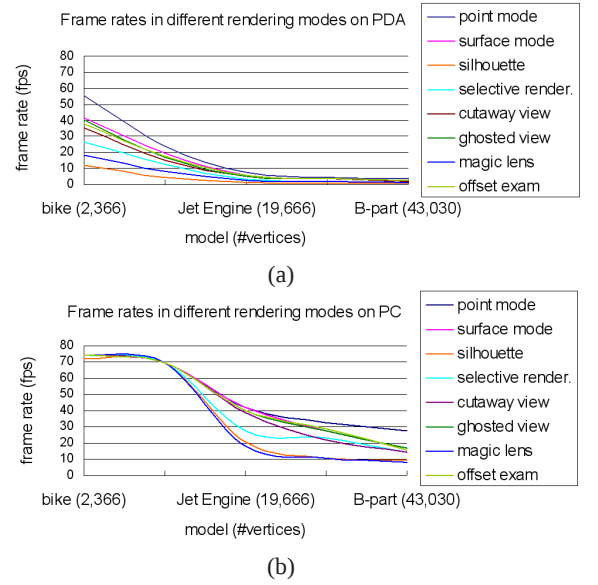


Figure 8: Performance in different rendering modes on a PDA (a) and a desktop (b)

7 ACKNOWLEDGEMENTS

We would like to thank Boeing Company for generously providing us with a nice Boeing 777 dataset and Matthias Deller of German Research Center for Artificial Intelligence for supplying the V8 engine dataset. We also thank Professor Yung-Hsiang Lu of Electrical and Computer Engineering Department at Purdue University for his help on this project. This work has been supported by U.S. Air Force AFRL-WS 06-1469, FA8650-05-2-6648 Grant, NSF ITR's Grant 0081581, 0121288, 0328984, and the U.S. Department of Homeland Security.

REFERENCES

- [1] Dave Astle and Dave Durnil. *OpenGL ES Game Development*. Course Technology PTR, 2004.
- [2] J. Buchanan and M. Sousa. The edge buffer: A data structure for easy silhouette rendering. 2000.
- [3] Bruce D'Amora and Fausto Bernardini. Pervasive 3d viewing for product data management. *IEEE Comput. Graph. Appl.*, 23(2):14–19, 2003.
- [4] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive Cutaway Illustrations. In *Proceedings of Eurographics Conference '03 (to appear in Computer Graphics Forum)*, 2003.
- [5] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard F. Riesenfeld. Interactive technical illustration. In *Symposium on Interactive 3D Graphics*, pages 31–38, 1999.
- [6] Daoud Hekmatzade, Jan Meseth, and Reinhard Klein. Non-Photorealistic Rendering of Complex 3D Models on Mobile Devices. In Ulf Bayer, Heinz Burger, and Wolfdietrich Skala, editors, *Proceedings of 8th Annual Conference of the International Association for Mathematical Geology (IAMG)*, pages 93–98, 2002.
- [7] Freudenberg B. Halper N. Schlechtweg S. Isenberg, T. and T. Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Comput. Graph. Appl.*, 23(4):28–37, 2003.
- [8] Robert R. Lipman. Mobile 3D visualization for steel structures. volume 13, pages 119–125. Automation in Construction, 2004.
- [9] Parallel-Graphics. Pocket cortona, 2006. Available at <http://www.parallelgraphics.com/products/cortona/>.
- [10] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 135–140, New York, NY, USA, 1999. ACM Press.

- [11] Simon Stegmaier, Joachim Diepstraten, Manfred Weiler, and Thomas Ertl. *Widening the Remote Visualization Bottleneck*. In *Proceedings of IEEE ISPA'03*, pages 1–6. IEEE, August 2003.
- [12] Ivan Viola and Meister Eduard Gröller. Smart visibility in visualization. In W. Purgathofer L. Neumann, B. Gooch, editor, *Proceedings of EG Workshop on Computational Aesthetics Computational Aesthetics in Graphics, Visualization and Imaging*, pages 209–216, 5 2005.