9-2001

The Use of the Voxmap Pointshell Method of Collision Detection in Virtual Assembly Methods Planning

Tom C. Johnson
Iowa State University

Judy M. Vance
Iowa State University, jmvance@iastate.edu

The Use of the Voxmap Pointshell Method of Collision Detection in Virtual Assembly Methods Planning

Abstract
Virtual reality (VR) provides the ability to work with digital models in an environment that provides 3 dimensional interaction. This technology can be used to evaluate how humans interact with products before costly physical prototypes are built. One of the advantages of using VR technology in design evaluation is the ability to easily explore many different "what-if" design scenarios. One of the areas of current research in the use of VR is in assembly methods planning. As a result of prior work performed at Iowa State University, it became clear that collision detection is an important component in the development of virtual assembly methods planning applications. This paper describes the use of the Voxmap Pointshell method of collision detection as it is applied to a general purpose virtual assembly planning application.

Keywords
VRAC

Disciplines
Computer-Aided Engineering and Design

Proceedings of DETC'01
ASME 2001 Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
Pittsburgh, Pennsylvania, September 9-12, 2001

DETC2001/DAC-21137

THE USE OF THE VOXMAP POINTSHELL METHOD OF COLLISION DETECTION IN VIRTUAL ASSEMBLY METHODS PLANNING

Tom C. Johnson
Virtual Reality Application Center
Iowa State University
2274 Howe Hall, Room 1620
Ames, IA 50011

Judy M. Vance
Virtual Reality Applications Center
Mechanical Engineering Department
Iowa State University
Ames, IA 50011
jmvance@iastate.edu

ABSTRACT

Virtual reality (VR) provides the ability to work with digital models in an environment that provides 3 dimensional interaction. This technology can be used to evaluate how humans interact with products before costly physical prototypes are built. One of the advantages of using VR technology in design evaluation is the ability to easily explore many different "what-if" design scenarios. One of the areas of current research in the use of VR is in assembly methods planning. As a result of prior work performed at Iowa State University, it became clear that collision detection is an important component in the development of virtual assembly methods planning applications. This paper describes the use of the Voxmap Pointshell method of collision detection as it is applied to a general purpose virtual assembly planning application.

INTRODUCTION

In seeking to reduce product development costs, one area of focus is in reducing time-to-market. This challenges engineers to implement product development processes that provide lower costs with short time-to-market schedules that also produce reliable designs (Jung, et al., 1998). Assembly planning is one area where improvements can result in both time and cost savings on the overall product development process. The costs associated with assembly are long term and have an on-going impact on product cost. For this reason it is beneficial to focus on reducing assembly costs in addition to design costs.

It is most cost effective to implement assembly planning early in the design cycle. If the assembly planning does not take place until the physical prototypes are already completed, modifications are expensive, time consuming and less likely to occur. When a problem is discovered in the assembly planning process, changes to the product's geometry are often required. If this occurs late in the design process, after part tooling has been created, these changes are very expensive. Engineers can reduce the cost of redesign in the development of the planning process by implementing the tools of computer graphics (Wang and Kim, 1996).

Digital assembly is currently a commonly used engineering tool. Digital assembly differs from virtual assembly. Digital assembly involves the use of digital models as displayed on a traditional monitor. Engineers use these models to verify part fit and determine part subassembly configurations before physical models are constructed. Desktop CAD packages, such as ProEngineer and SDRC I-DEAS, allow users to create or import three dimensional digital models and assemble them according to constraints placed on mating surfaces or aligned axes of two parts, for example. Digital assembly tools give engineers a quick and inexpensive method of experimenting with digital models to develop different assembly possibilities.

Virtual assembly is similar to digital assembly in that both methods use the same digital models, however in virtual assembly a virtual environment is created which allows participants to intuitively interact with actual size models. In virtual reality, the models of the parts are drawn in full scale and engineers can examine how factory personnel will interact with the actual parts in the real assembly process. The full scale digital models in the virtual environment represent the correct spatial relationships to the user. Desktop systems can not replicate these actual size displays or provide participants with a way to interact naturally with the three dimensional digital

models. On the desktop systems, viewing manipulations are performed by keyboard and mouse interaction, while in the virtual environment people can naturally move about and handle the models. Also, on the desktop systems, the parts are scaled to fit within the space of the computer monitor, where in the virtual environment, the parts can be viewed in their actual size. When implemented early in the product development process, virtual assembly has the potential to reduce the time to optimize the assembly sequence and eliminate unnecessary iterations of physical mock-ups.

Assembly decisions are often made very quickly. It is important to have a program that is general enough to present a virtual assembly environment to the user with minimal customization of the computer code for each assembly scenario. Collision detection is an important component in the development of virtual assembly methods planning applications. Users need to know when parts collide in the virtual environment. The Voxmap Pointshell (VPS) method, developed at Boeing, was selected for this virtual assembly application. This paper presents a description of the VPS method and how it was used in the development of the Virutal Environment for General ASsembly, VEGAS. Conclusions and recommendations for future work are also included.

VOXMAP POINTSHELL METHOD

Boeing developed software for the specific use of six degree-of-freedom haptic rendering and as a result, a useful collision detection algorithm was also developed. The basis of this method, called the Voxmap PointShell method (VPS) is to represent the geometry of the scene by voxels, or small cube elements (McNeeley, et al., 1999). Voxels are analogous to the pixels on a computer monitor. As pixels work together in two dimensions to display geometry on the screen, voxels, when combined in three dimensions, form three dimensional digital models. The VPS method merges all of the voxelized objects in the scene into a single entity. One of the objects in the scene can then be unmerged, or made dynamic, and it is represented as a collection of surface normals referred to as a pointshell. The normals of the dynamic object are used to determine the direction of the reaction force when a collision occurs with the static voxmap. The interaction between the voxels of the static merged scene and the dynamic pointshell results in the Voxmap PointShell (VPS) algorithm.

While initially developed specifically for six degree-of-freedom haptic rendering, VPS has evolved into a useful software library for collision detection as well as three degree-of-freedom haptic rendering. VPS was not designed to provide high accuracy collision detection. Rather, the intent is to perform rapid collision detection. As explained by McNeely, the limitation of the voxel-scale accuracy is acceptable for common assembly tasks where keeping about 0.5 inch clearance between parts as they are assembled is common engineering practice (McNeely, et al., 1999). This clearance accommodates tool access, human access and serves as a buffer to tolerance buildup.

Boeing's VPS software libraries were used for the collision detection capabilities within this assembly application because the method provides reasonable results in a very efficient manner. If the direction of the research was to evaluate the fit of the parts together, a more exact method would be necessary, however this application is focused on the human interaction within a fully immersive environment.

VEGAS DEVELOPMENT

The Virtual Environment for General ASsembly, (VEGAS), was developed because of the need for a fully immersive assembly environment with the ability to handle different assembly situations without recompiling an application specific to one set of digital models.

Model preparation

The first step in developing VEGAS was to implement a method of importing the models into the scene. The application should be able to handle complex models relatively large in size with high polygon counts. For the generality of the application it was determined that these models could be read from a separate file upon start up of the application. In addition, it was desirable to be able to read in various files without recompiling the application.

As identified by Steffan, et al. (1998) and Fujimoto, et al. (1998) one of the basic steps to the implementation of a virtual assembly simulation is the model preparation. The application described here uses the geo file format created by Engineering Animation Inc. VEGAS accepts polygonized models made up of triangles. The format can be derived from parametric models and is based on a simple method of connecting vertices together to create polygons. The source code of VEGAS was developed to read an initialization file upon start up called geo_init. This file simply contains the number of models and the names of the model files to be loaded (Figure 1).

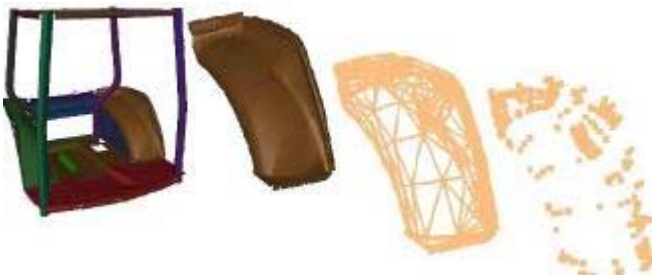
```
6
brake_arm.geo
brake_cap.geo
brake_frame.geo
brake_reservoir.geo
brake_plate.geo
cab.geo
```

Figure 1: geo_init file

The model files in the geo_init file are actually groups of parts. The user of the application is responsible for separating the product models into groups of parts. An engineer decides what parts to group together according to the assembly process. The example here is the task of assembling a parking brake into a cab. The part groups of interest are the cab, as a whole,

and the components of the parking brake. A part group may contain many parts. The cab part group, for example, is composed of twenty-five individual parts: roof, roof supports, fenders, etc. The objective of this example is not to explore the assembly of the cab, but rather the assembly of the brake into the cab, therefore the cab is treated as one part group.

When VEGAS reads the geo_init file, it first looks for the *number of models* parameter, which it uses as a counter for the loop that does the actual model loading. The next lines in the file are the model file names. Once the names of these geo model files are read into VEGAS, the program searches for the GeoModels directory to find the model files. Upon locating the model files, VEGAS reads the model files to load information on the number of parts, the number of polygons and the number of vertices per file, to be used later for the collision detection routines. Within the program, the model hierarchy is maintained as shown in Figure 2.



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

Figure 2: Model hierarchy example: (a) cab part group, (b) fender part, (c) fender polygons, (d) fender vertices

The next step, after reading in the model data is to provide a method for the user to interact with the models. This interaction should be simple enough so the user can easily manipulate the models and should be complex enough to perform actions such as selecting parts, grabbing parts, placing parts within the environment and assembling them with other parts. A position tracked input device called a wand is provided to the user in the environment. The wand has three buttons, which, when pressed, are used to send signals to the VR program much like a desktop mouse interacts with traditional applications. In this manner, the wand acts as a 3D mouse.

The challenge in developing the interactive portion of the application was to give the user a way to perform the assembly procedure using only the three buttons on the wand for input. It is also important to keep the button combinations simple enough that the user would not become confused on how to interact with the environment. The first step in determining the interaction methods is to assess the options needed to assemble parts. In reality, assemblers can pick up an object and move and rotate the object as desired. The three buttons on the wand handle the input options that control the motion of the part groups within the environment.

In order for the user to interact with a specific part group there needs to be a way to identify the part group of interest. One of the buttons, button 1, on the wand is programmed to cycle through and select each part group. Once a part group has been selected, button 2 allows the user to "hold on to" it for as long as the button is pressed. By holding on to the part group, the model is actually being rotated and translated by the same increments as the position tracked hand. The result is the appearance that the model is being manipulated by the user's hand. There are two modes that govern where the part is placed after it is released. These modes are *place* and *snap*. Button 3 serves as a toggle switch between the two states. The digital model can be placed, meaning it can be released anywhere within the virtual environment, or snapped which causes the part to appear in its assembled location.

Preparing the models for and implementing the collision detection method is the next step. In order to use the VPS libraries in the VEGAS code, the geo model files must be converted to a format recognized by VPS. The conversion is performed within VPS, provided some specific part information regarding the geometry of the file is given to the algorithm. VPS needs the polygonal data consisting of vertices, polygons and outward pointing normal vectors for each polygon. The geo files contained the vertex and polygon information but lacked the information on the normal vectors. Therefore the normals are calculated in VEGAS by numbering the vertices counterclockwise for each triangular polygon, v0, v1 and v2. Then two vectors $\underline{a}$ and $\underline{b}$ were found by subtracting v1 from v2 and v1 from v0, respectively. The normal, $\underline{n}$, was finally found by determining the cross product of the vectors $\underline{a}$ and $\underline{b}$. VEGAS was able to ensure that the normals that were being passed into the VPS algorithm were outward pointing by remaining consistent in numbering the vertices of the triangles counterclockwise and finding the two vectors.

Voxelization of the models

VEGAS passes the part information to VPS functions, which transform the geo files, in a triangle-by-triangle manner, into a format recognized by VPS. This format is the voxmap of the model. A voxel is a small three-dimensional cubical entity that can be combined to make up a larger unit, called a voxmap. A voxmap is based on the polygons that represent the surface of the model, therefore the voxmap itself is not a solid model. The voxelization process takes the polygonal models and turns them into voxelized model representations. For example, a part file called TestPart.geo would be returned from VPS as TestPart.vps. VEGAS then exports this .vps data, or voxmap into the directory called VpsModels where this data is stored until called upon for collision checks. Figure 3 shows a polygonal and voxelized representation of a tractor cab.

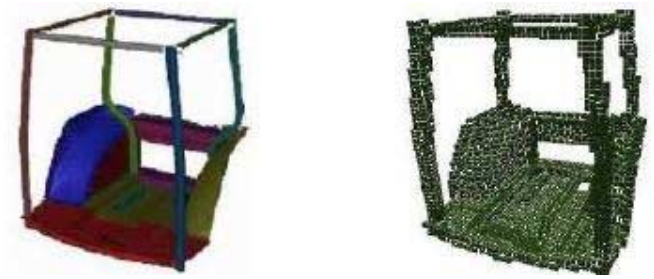


Figure 3: Comparison of the exact model with the voxelized approximation

One constraint of the VPS software is that all objects must have the same voxel size. This can become a problem if the assembly environment is composed of objects with largely varying sizes. VEGAS takes the limitations of the VPS voxelization process into consideration and finds the best sized voxel for the entire environment based on the geometry loaded into the environment. First of all when the digital models are read into VEGAS, the application finds the two extreme vertices of each part and the distance between these two vertices is calculated. When all of the object sizes have been determined the next step is to use the object sizes to determine an appropriate voxel size for all of the models in the environment.

In order to ensure better results from the voxelization process for the smaller models, a weighting method was used to favor the small objects when the global voxel size is determined. This method sets the voxel size as a percentage of the object size. Smaller part group's voxel size is a slightly smaller percentage of the object that the larger. The smallest part groups are voxelized with a voxel size of 1.25% of the object size. This percentage ranges up to 1.4 percent for larger part groups. The variable percent of the object size used to calculate the voxel size produces voxel representations of large objects, which are not too dense and small objects, which are sufficiently detailed. Figure 4 shows the difference in the voxmaps for an object voxelized at 1.40% and 1.25%.

(a) (b)

Figure 4: Voxel size comparison for the parking brake part group. Object size is 2.48 feet voxelized at (a) 1.40% and (b) 1.25%

In summary, to find the global voxel size for all of the models in the virtual environment VEGAS first calculates the individual object size for all models. The object size is then used to determine the appropriate voxel size for that model. Finally, when the voxel sizes have been determined for all models, an average voxel size is determined and set as the global voxel size. Part groups are then voxelized into the environment with the single global voxel size.

Collision detection

After the part groups are voxelized, the next step in developing the general virtual assembly application is to implement the collision detection routine. The main advantage of using the VPS libraries for collision detection is computational efficiency. This is important in VR applications in order to maintain frequent visual updates, which simulate actual interaction. VPS maintains this efficiency by a technique called merging. Merging is the process of taking all of the voxelized parts of the part groups in the environment and combining them into one entity. The parts are all merged together after they are read into VEGAS. Recalling the interaction capabilities, the user can cycle through, select and grab various individual part groups within the environment. The part group that is grabbed by the user is identified as the dynamic part group, while the other part groups within the scene are treated as static. The parts of the grabbed, dynamic part group are merged together and all the remaining parts of the static part groups are merged together. When the focus is switched to another part group, the previous dynamic part group must be merged into the static scene and the new dynamic data must be removed from the voxmap. This merging and remerging process takes place every time the user grabs a different model.

The process of checking for collisions begins following a successful merging of the static and dynamic parts. As the user is moving the dynamic part group about the environment, the position and orientation of the part group must be updated to the VPS intersection checking routines to check for collisions with the statically merged scene. It would be very computationally expensive to translate and rotate each voxel in the dynamic part group's voxmap as it is being moved about the environment, therefore, for visual representation, only the digital part group is moved, not the voxmap. Both of the voxmaps, of the dynamic and static part groups, remain in their original position. However, VPS actually checks for collisions between the dynamic and static voxmaps, not the digital part groups. In order to check for collisions between the static voxmap and the dynamic voxmap, VPS uses the transformation matrix of the digital dynamic part group. The transformation matrix contains the position and orientation information of the digital dynamic part group in relation to its voxmap original position. VEGAS is responsible for determining the transformation matrix of the dynamic part group and passing the matrix into the VPS intersection checking routines. VPS receives the transformation matrix and applies it to the dynamic part group's voxmap. This method of passing the digital dynamic part group's transformation matrix back to the original

position and orientation of the dynamic part group's voxmap saves valuable computing time because the voxmap remains static and each voxel does not have to translated and rotated throughout the environment.

VEGAS is checking the interaction between the voxels for collisions while the user is moving a part group. If a collision occurs between voxels of the dynamic part group and a static part group the dynamic part group turns red. This visual indication provides the user with immediate feedback that the parts are touching, according to their voxel representations. This feedback is important to engineers investigating various assembly methods. For example, the engineer can use VEGAS to simulate the process for assembling a part to a main assembly in an area where there are many other parts congested together. The engineer could grab a part and traverse the path the actual assembler would when installing the part and see if any collisions occur.

EXAMPLE

Models of a John Deere cab frame, Figure 5, and a parking brake assembly, Figure 6, were used as examples in this work. These models are complex enough to test the capabilities of the application and contain interesting geometry for evaluating numerous assembly ideas. The full scale models of the cab and parking brake were also a suitable size for analysis in the C2 environment. The C2 is a structure with three 12' x 9' walls consisting of projection screens and a 12' x 12' floor. Images are projected onto the walls and floor using BARCO 1208s projectors. To provide the stereo images the users of the C2 wear shutter glasses. VEGAS was developed for a fully immersive environment, such as the C2, so engineers could interact with the assemblies and with each other naturally. For this example the C2 provided an environment in which multiple users interact with the assembly and collaborate to share ideas on the assembly methods. This collaborative environment is most beneficial when discussions between design engineers, manufacturing engineers and assemblers are needed to make assembly methods decisions. The C2 allows these various groups of people to communicate their own specific areas of concern so everyone understands each other's ideas and limitations about the assembly process.

Figure 5: Cab and parking brake assembly

The digital models used in this example are actual assemblies that were provided by Deere and Company for this research. The digital models were originally created using the CAD software package ProEngineer. The models were loaded into ProEngineer and exported as Inventor files. The Inventor file format is a conventional polygonal format that is used in an intermediate stage of the model translation to convert the parametric ProEngineer model into a collection of triangular polygons. The brake and cab models were exported from ProEngineer with the default chord length determined by ProEngineer. The chord length is the linear segment used to approximate parametric curves. The exported ProEngineer model contained approximately 80,000 polygons. The models were loaded into Engineering Animation's VisModel™ to have the unnecessary polygons removed. This process took about one hour and resulted in reducing the polygon count from around 80,000 polygons to around 30,000 polygons when complete. This process is called polygon decimation. The models were saved in the geo format and were ready to be loaded into VEGAS.

Following the decimation of the files, the models are divided into seven part groups. The cab became one large part group and individual components of the hand brake assembly made up the other six part groups. The purpose of breaking up the environment in this manner was to investigate the voxel size approximation calculations on part groups with a wide range of object sizes. The large scale of the cab provided an excellent opportunity to interact with real size models in the C2 environment and the brake assembly provided models with interesting detail that could be easily manipulated throughout the virtual world. The method of breaking up the large models into smaller part groups involves saving the smaller part groups from VisModel individually, separate from the entire part group. Once the models were saved into the geo format they were listed in the geo_init initialization file in order to be read into the application.

The next step is to execute the VEGAS application. Upon the start up of the application, the initialization file, containing the list of models to be loaded into the environment, is read. Next, the application opens the model files to obtain the necessary geometry information, which include: part, polygon and vertex information for each part group. The necessary geometry information is then passed into the VPS voxelizing functions and the voxelized models are created. With this specific set of models, VEGAS calculated the global voxel size to be approximately 0.061. The voxel size is considered to be unitless by VPS, but it is based on the object size, which has units of feet. With the voxelization process complete, the digital models are displayed in the virtual environment and the application is prepared for assembly evaluation.

Within the VEGAS application, the transformations that have been applied to the dynamic part group are passed to the VPS collision detection algorithm. VPS uses this information and compares the position of the digital model with the original position of the voxmap to check for collisions. As evident in Figure 6, a collision is shown even if the exact surfaces of the models are not in contact. Figure 6(a) shows the a collision taking place between two part groups (as evident

by the red colored part), but also shows the small space existing between the two parts. Figure 6(b) shows what VPS is using to determine the collisions.

a) b)

Figure 6: Collision detection showing (a) exact surfaces, (b) voxelized surfaces

Another example is a part group that the user is trying to fit through the access hole on the cab floor in order for it to be placed in its assembled position under the cab. In this example, the bottom of the cab is only about one foot off of the ground (Figure 7). Rather than having to reach under the cab to assemble the part group, the assembler would prefer to have clearance through the access hole. However, a collision occurs when this method is investigated and the part group will not fit through the access hole. An alternative assembly method with the same part group is to assemble it under the cab. From this example an engineer may draw the conclusion that the access hole may need to be larger so the assembler does not have to be in an awkward physical position by having to place the part group underneath the cab. Another option to investigate might be to change the orientation of the cab so the underside of the cab is more accessible.

Figure 7: Assembling a parking brake component under the cab in the VEGAS environment

In conclusion, a summary of the steps to produce the assembly environment is provided.

- Obtain the models for the assembly environment.
- Export the models from the CAD packages in Inventor format.
- Load the models into the VisModel software program for polygon removal.
- Save the models from VisModel in geo format.
- Run VEGAS, and begin evaluation of assembly methods.

The most time consuming aspect of this process is the polygon removal step. For this example, which contained relatively large and complex models, the model clean up took around 1 hour. It is reasonable to estimate that engineers could use this process to evaluate assembly methods with as little as an hour preparation time.

CONCLUSIONS

The VEGAS application was developed to provide a fully immersive virtual environment to investigate general assembly methods with full scale digital models. Using the models of the parking brake and cab, the application was successful in developing such an environment for investigating various assembly methods. The general model loading capability of the application provided an interface to quickly produce the virtual assembly environment. Once the models were in the correct format, moving from a two dimensional image on the computer monitor to a full scale three dimensional digital model was performed with little effort. The implementation of collision detection was the key factor in evaluating the assembly processes. A color change in the part informed the user of a collision with another part, which proved to be most beneficial when investigating possible assembly methods.

The example assembly environment also was useful in determining issues that could be improved upon. To further enhance the realism of the virtual assembly environment the collision detection should simulate real world interaction between the models. Rather than only a color change, the application should provide the capability to prohibit the models to interpenetrate each other. The collision detection could be taken a step further by preventing the graphics from moving if a collision is taking place. The addition of haptic feedback would also help to more closely simulate the real assembly environment by providing force feedback when the models collide (Golabi, et al., 1996). In the example assembly design environment, collision detection was only possible between the part group being manipulated by the user and the part groups that were in their correctly assembled, or snapped, position. If a part group was not snapped to its assembled position and simply placed within the environment a collision between the dynamic part group and the unsnapped part group would not appear. The application would provide more useable results to engineers if the collision detection would be available to parts in the unsnapped position.

FUTURE WORK

Implementing haptic feedback should be the next step in further development of the virtual assembly environment. While the actual algorithm for performing the haptic interaction is not fully implemented into VEGAS, the framework for doing so is presented here. The haptic calculations are based on similar features of the collision detection routines, with the main shared feature being the voxelized representation of the models. Along with the voxmap, which is still used to represent the static environment, the dynamic model is now represented by a VPS pointshell. VPS uses the voxelized model to generate the pointshell. The pointshell is a collection of points located at the center of the voxels, with each point having an inward pointing normal. The points, along with the inward pointing normal serve as a reference when calculating the components of the reaction force.

To begin implementing the haptic interaction, the process begins much the same way as with collision detection. The first step is to voxelize all of the models within the virtual environment. The voxmaps of the objects are then used to find the pointshells of the objects. When a model is selected out of the static environment to be the dynamic object, the pointshell is applied to it. The merging, remerging and revoxelizing methods apply to the remaining objects within the environment as they did with the collision detection. However, when an interference between two parts occurs reaction forces are generated as well as a collision.

In the example assembly application there was one global voxel size for all models, which was calculated based on the size of the part groups in the environment. The uniform voxel size was used to ensure adequate performance and frame rate of the application. Using one voxel size placed some limitations on accurately representing collisions between part groups. Voxmaps can only be merged together if there is a consistent voxel size between them. An interesting study for future work would be to test the performance of the application if the individual parts of the part groups were not merged together, but rather kept separate and only the parts within a part group would be merged together. The collisions would be checked against all of the part group's voxmaps instead of just against the one merged scene. This would allow the voxmaps to have different sizes, but may also decrease the performance of the application.

In regards to the global voxel size there could also be some further tests into finding the best voxel size for the whole environment. As the voxel size is determined now, it is assumed to be acceptable if it seems visually adequate to represent the exact model surface. A comparison could be made between the voxel volume and the exact part volume to determine an appropriate voxel size. Additional tests could be performed to determine how the changes in voxel size actually affect the performance of the application rather than only the appearance.

In the version of VEGAS used for the example assembly environment the interaction was performed with the wand. The three buttons on the wand allowed the user to cycle through and grab a model in the environment. In maintaining the general nature of the code, the next step would be to generalize the interaction. VEGAS should not be constrained to using only the wand for interaction. Other input devices such as the PinchGlove, should be available to VEGAS users.

Additionally the interaction commands should not be limited to only grabbing the models. A user may want to have animation capabilities within the environment. As the models which are loaded into the environment are read from an initialization file a similar script could be provided for the interaction. While the details for the interaction script implementation have yet to be developed it would simply assign certain tasks to an input device based on the reading of an ASCII text file. For example, button one on the wand could be defined in the interaction script to perform an animation of rotating the entire scene about the x-axis. This would let engineers investigate assembly methods from a new viewpoint.

A final addition to increase the general nature of this application would be to incorporate the use of additional file formats. The example application used the geo file format developed by Engineering Animation Inc. While this format is simple to use, it is dated and unrecognizable by many modeling packages. In order to make VEGAS more flexible it should have the ability to read additional file formats. A voxelizing engine has been developed for the Inventor (.iv) format, but has yet to be incorporated with the models displayed within the environment. VEGAS should have the ability to understand many popular file formats and should not be restricted to one file format for a specific assembly environment. Implementing this into the application would eliminate the process of having to convert from the triangulated format exported from the CAD packages to the geo file.

ACKNOWLEDGMENTS

The authors wish to acknowledge the support of Deere and Co. in the development of virtual assembly applications and the availability of models for this work.

REFERENCES

Antonishek, B., Egts, D., and Obeysekare, U.R., 1998, “Virtual Assembly Using Two-Handed Interaction Techniques on the Virtual Workbench,” *Proceedings of the ASME Design Engineering Technical Conference*, Atlanta, GA, September 13-16, pp. DETC98/CIE-5540.

Banerjee, A., and Banerjee, P., 1999, “Assembly Planning Effectiveness Using Virtual Reality,” *Presence*, vol. 8, no. 2, pp. 204-217.

Beach, D.A., and Anderson, D.C., 1996, “A Computer Environment for Realistic Assembly Design,” *Proceedings of the ASME Design*

Engineering Technical Conference, Irvine, CA, August 8-12, pp. 96-DETC/CIE-1336.

Fujimoto, H., Ahmed, A., and Sebaaly, M.F., 1998, "An Evolutionary and Interactive Approach to Simulation of Assembly Planning in Virtual Environment," *Proceedings of the IEEE Virtual Reality Conference*, Atlanta, GA, pp. 187-192.

Ganter, M.A., and Isarankura, P.B., 1993, "Dynamic Collision Detection Using Space Partitioning," *Journal of Mechanical Design*, vol. 115, pp. 150-155.

Golabi, S., Abhary, K., and Luong, L., 1996, "A technique for planar contact recognition in automatic assembly planning," *Proceedings of the ASME Design Engineering Technical Conference*, Irvine, CA, August 18-22, pp. 96-DETC/DFM-1463.

Jung, B., Latoschik, M., and Wachsmuth, I., 1998, "Knowledge-Based Assembly Simulation for Virtual Prototype Modeling," *Proceedings of the IEEE Virtual Reality Conference*, Atlanta, GA., pp. 2152-2157.

Kesavadas, T. and Subramaniam, H., 1999, "Experimental Study of Virtual Tools with Attributes," *Industrial Virtual Reality: Manufacturing and Design Tools for the Next Millennium*, Chicago, IL, MH-vol. 5/MED-vol. 9, pp. 11-17.

McNeely, W., Puterbaugh, K., and Troy, J., 1999, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling," *Proceedings of Siggraph 1999*, Los Angeles, CA.

Steffan, R., Schull, U., and Kuhlen, T., 1998 "Integration of virtual reality based assembly simulation into CAD/CAM environment," *Proceedings of the IEEE Virtual Reality Conference*, Atlanta, GA, pp. 2535-2537.

Wang, E., and Se Kim, Y., 1996, "Feature-Based Assembly Mating Reasoning," *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Atlanta, GA, August 18-22, 96-DETC/CIE-1341.