

Screenshot from id Software's Quake III: Arena showing the typical player and spectator experience in architectural environment-based games. The view is limited to a single room of a particular level (here, the Temple of Retribution), so understanding the overall environment and its uses is equally limited. Players and spectators alike find it difficult to know how the various players are moving through the level, where the weapons and power-ups are located, how the floors are laid out and linked, the quickest paths from place to place, the good places to hide, and the places to avoid.





VISUALIZING DYNAMIC ARCHITECTURAL ENVIRONMENTS

HOW TO EXPOSE THE INTERNAL 3D STRUCTURES
OF MULTIPLAYER GAMES AND ARCHITECTURAL
MODELS BY AUTOMATICALLY GENERATING
INTERACTIVE EXPLODED VIEWS.

Recent advances in consumer graphics technology make it possible to interactively explore extremely complex 3D environments. Architects routinely build detailed 3D computer-aided design (CAD) models of buildings to visualize both their internal spaces and their external structures. The trend toward greater complexity is evident in video games involving increasingly detailed interactive 3D worlds. These environments are comprised of many rooms, passageways, characters, and players, but also tend to be densely occluded.

Few applications are able to simultaneously display their interior spaces and the external structures. Two related interfaces are ArcBall [10], which allows rotation, scaling, and zooming of the environment, and walkthroughs, which allow viewers to move through rooms within the environment. ArcBall interfaces are often used in model-viewing applications and can be useful for understanding an

BY MIKE HOUSTON, CHRIS NIEDERAUER,
MANEESH AGRAWALA, AND
GREG HUMPHREYS

environment's external structure. Walkthrough interfaces, used in first-person shooter-style games, are useful for understanding and navigating an environment's interior spaces. However, neither of them allows viewers to understand the environment as a whole, because the walls and floors hide most of the structure. In dynamic 3D environments (such as multiplayer games like Quake III [4]), the occlusions make it impossible to see all the action at once.

Architects and technical illustrators often use techniques like cutaways, transparency, and exploded views to reduce or eliminate occlusion and expose the overall structures of architectural environments. Because exploded views are especially effective for conveying structure, they are often used in illustrations of mechanical assemblies [2, 11]. Designers Stephen Biesty [1] and Edward Tufte [12] are well known for using exploded views to reveal the structure of multistory buildings and machines. To form an exploded view of an architectural model, designers typically section the building into stories just below the ceilings, then separate the stories from one another (see Figure 1). These views expose both the structure of the internal spaces within each story and the vertical spatial relationships between adjacent stories. But producing an exploded view from a 3D model of an architectural environment requires the designer to annotate the location of each story, as well as a viewing application that generates the appropriate exploded view from the annotated model [5, 6, 9].

We have now developed an interactive system—called ArchSplit—that provides automated support for generating exploded views of any architectural environment [8]. Requiring little semantic understanding of the environment, it assumes the environment is “architectural” and searches for geometric primitives representing ceilings. Once the system finds the ceilings, it automatically sections the environment into stories, rendering each one separately in exploded form. The system provides interactive control over the viewpoint and the separation distance in terms of height between the stories. This control makes it easy to understand both the structure of the environment and the relationships among its dynamic objects and characters.

We tested this visualization technique on several OpenGL applications, implementing it noninvasively. Borrowing the approach proposed in [7], we used Chromium, a stream-processing framework for interactive rendering on clusters [3], to intercept and manipulate sequences of OpenGL commands generated by the underlying application. Chromium's stream-processing units (SPUs) provide the flexibility needed to modify, delete, replace, or augment any of

the graphics API calls made by the OpenGL applications while it is running. The SPUs are easily adapted to affect semantic transformations on streams of graphics commands. Thus, our architectural visualization system can be applied to many OpenGL applications, including Quake III, without modification or recompilation.

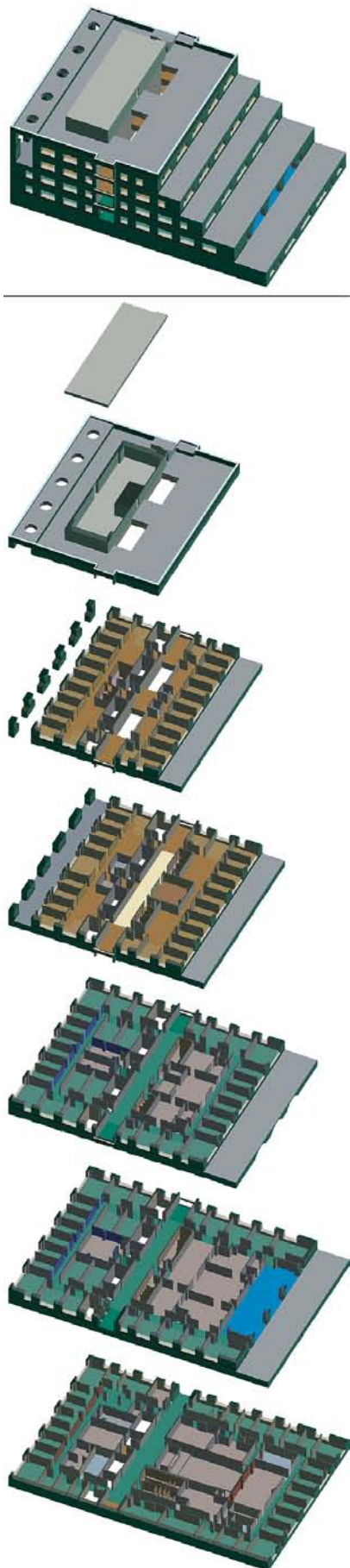
Geometric Analysis and Rendering

The system processes the graphics API stream from any OpenGL application in two stages. The first—geometric analysis—determines where to split the architectural model into its stories by analyzing the stream of polygons issued by the original application. The analysis is performed once, whenever a new architectural model is loaded into the original application.

The second stage—rendering—draws the exploded view by modifying the OpenGL graphics stream of the original application. Based on the geometric analysis, the system inserts clipping planes between each story and performs multipass rendering, one pass per story, to produce the exploded view. The renderer also replaces the viewpoint—the projection and modelview matrices—specified by the original application with a new viewpoint that may be specified interactively by the user. The rendering stage modifies every frame of the original application on the fly.

The most natural segmentation of any architectural model is into stories. A story is usually defined by a floor, a ceiling, and all the geometry in between. For a visualization, the system does not include the ceiling in each story because it occludes the very structure we are trying to reveal. Therefore, the best place to split the model is just below each ceiling. The goal of the geometric analysis stage is to determine which of the polygons in the environment represent ceilings. The rendering stage then inserts a clipping plane into the environment just below each discovered ceiling to separate the environment into individual stories.

To find the ceiling polygons, the user specifies a vector defining the up direction for the environment. Ceiling polygons are oriented so their normals point in the opposite direction of this vector. As the original application submits geometry to OpenGL for rendering, Chromium intercepts the vertices of each polygon. Assuming polygons are specified using consistent counterclockwise ordering, as in many OpenGL applications, the system computes the polygon normal as the cross-product of two edges of the polygon that share a common vertex. This is generally a safe assumption, as most applications are careful



about their geometry's winding order, which results from OpenGL backface culling semantics.

While this approach finds all downward-facing polygons, not all such polygons represent ceilings. Downward-facing polygons may appear in other parts of the environment (such as in portions of characters or objects) or in smaller architectural elements (such as in windowsills, ledges, stairs, and ornamental decorations). To find the polygons most likely to represent ceilings, the system computes the height of each downward-facing polygon, then builds a table mapping each potential ceiling height to the total surface area of all downward-facing polygons at that height.

The user interactively specifies `NumSplits`, or the number of stories the environment should be split into. Starting with an unsplit environment, the user interactively increases or decreases `NumSplits`. Initially, the system finds the `NumSplits` largest surface

areas in the height-to-surface area table as candidate heights for splitting the environment. However, environments often contain large ceiling areas that are slightly offset from one another, and splitting the model at each of these offset heights would generate extremely low ceilings that would not be part of the desired segmentation. To counteract this effect, the system applies an additional heuristic

Figure 1. Exploded view of the Soda Hall model generated by ArchSplit from a model-viewer application. Representing a more traditional example of how to present static architectural models in exploded form, it is similar to hand-drawn techniques.

that maintains a minimum distance, or height, between neighboring splitting planes. In general, this distance should be set to the height of a typical character, as measured with respect to the environment, since no ceiling can be lower than this minimum height. The user adjusts the height as necessary for a particular model. In practice, it's easy to find the right minimum height parameter. For multiplayer games, the minimum ceiling height is specified as the average height of the player/character's geometry. For a more in-depth description of the algorithm, see [8]; a brief video showing the system in action with various applications, including Quake III, is at graphics.stanford.edu/papers/archsplit.

Axonometric View

After the system's geometric analysis stage determines where to split the environment into stories, its rendering stage modifies each frame of the original application to produce an interactive exploded view. The system uses Chromium to buffer the stream of OpenGL calls corresponding to a frame, or all of the

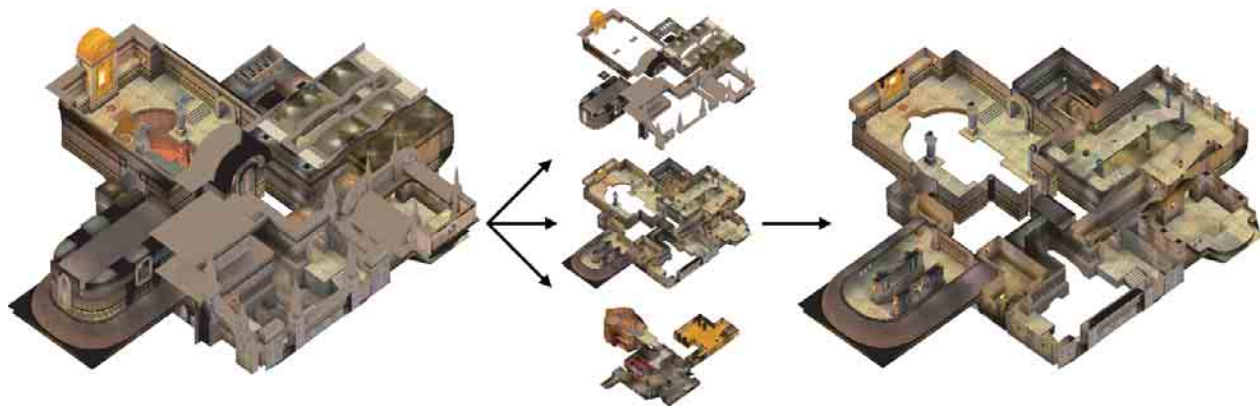


Figure 2. Exploded view of Quake III (demo) noninvasively generated by ArchSplit showing the environment from an external axonometric viewpoint, first unexploded, then exploded with a zoom into one of the floors. While the unexploded view shows external structure, the exploded view simultaneously reveals both internal and external structure.

functions and their parameters called between calls to `glSwapBuffers()`. The frame is then replayed, one per story, with each playback pass responsible for rendering one of the stories in the exploded view. Each playback stream modifies the original OpenGL stream in several ways:

- The original viewing projection is replaced by an external axonometric view, or a projection in which horizontal and vertical axes are drawn to scale but in which diagonals and curves are distorted;
- Clipping planes are inserted into the stream to ensure only a single story is drawn; and
- The geometry is translated along the up vector to separate the current story from the previous story.

Technical illustrators often use an axonometric projection when producing exploded views of architectural environments to eliminate perspective distortions. Our system generates an axonometric view by replacing the original application's projection matrix with its own axonometric projection. The system allows users to interactively adjust the viewpoint using an ArcBall interface [10]. To allow such control, the system locates the viewing transformations in the transformation matrix stack of the original application and replaces them with its own collection of transformations. The system assumes the application first sets up the viewing transformation for the environment; subsequent changes to the modelview stack represent relative motions of other graphical elements (such as players, objects, and overlays). Thus, the system can change the viewpoint by replacing the very first matrix placed on the OpenGL modelview stack.

When nonenvironmental graphical elements (such as players and objects) are drawn, it uses the inverse of the environment's original projection matrix to place them correctly relative to the new axonometric view.

To ensure each playback stream draws only the geometry associated with a single story, the system inserts two OpenGL clipping planes into the graphics stream just before the environment geometry. One is placed immediately below the ceiling of the current story so it clips all geometry above it. Similarly, the other one is placed right below the previous ceiling so it clips all geometry below it. The results of these transformations are shown in Figure 2.

By interactively adjusting separation distance, users quickly see how the stories fit together and connect with one another in another form of interaction that helps reveal the 3D structure of the architectural environment.

Assumptions

Although retrofitting existing applications noninvasively is a strength of this approach, it is also a limitation. In particular, it requires that the system analyze the environment at a very low level while making several assumptions about the semantics of the OpenGL stream issued by the original application. It also affects the performance of the application. For example, for each story at which the system splits, the system must resend all of the graphics primitives to the graphics card. Also, as noted in [8], for many applications, like Quake III, some of the rendering optimizations traditionally used must first be disabled. To maintain interactivity, the system makes use of a cluster of machines with graphics boards, with each machine rendering a single story. This approach mitigates the performance decrease caused by the extra rendering load.

When many players interact simultaneously in, say, a large gaming environment, the action can be




Figure 3. Mock-up of zoomed-in view of a single floor from *Quake III* (demo7) showing a game in progress. Using Photoshop, we added semantic information to the center split image generated by ArchSplit in Figure 2 to show the name and location of each player, number of kills (frags) for each team, and the paths each player has taken through the environment.

difficult for them, as well as for any spectators, to follow, even though everything is visible in the exploded view. A combination of geometric and semantic simplification would greatly increase the ability of players and spectators alike to understand the environment. For example, players appear quite small when the entire map is shown; they could be simplified or even “iconified” without sacrificing much semantic content. The movement of players could also be displayed in the exploded view to give all viewers a sense of how they are moving through and using different aspects of the environment (such as power-ups and weapons in games like *Quake III*), as shown in Figure 3.

Conclusion

Our aim is not simply to argue that noninvasive techniques should be used when exploded views might be useful, but to demonstrate a compelling new visualization technique for architectural environments. Application writers have access to higher-level semantic knowledge about these environments, including the locations of the ceilings and the viewing parameters. Access to such information would make it much easier to build our exploded-view visualization technique into the original application, as well as to use level of detail, or geometric and texture simplification, and provide semantic information to help users understand the environment. We thus hope to encourage designers of future systems to incorporate such visualizations directly into their applications.

Observing multiplayer games from a third-person

perspective is far more satisfying than observing them through the eyes of individual players, as it provides a more complete understanding of the environment and the dynamic character interactions taking place within it. 

REFERENCES

1. Biesty, S. and Platt, R. *Stephen Biesty's Incredible Explorations: Exploded Views of Astonishing Things*. Scholastic, Inc., New York, 1996.
2. Giesecke, F., Mitchell, A., and Spencer, H. *Technical Drawing, 3rd Ed.* Macmillan, New York, 1949.
3. Humphreys, G., Houston, M., Ng, R., Ahern, S., Frank, R., Kirchner, P., and Klosowski, J. Chromium: A stream processing framework for interactive graphics on clusters of workstations. *ACM Transact. Graph.* 21, 3 (July 2002), 693–702.
4. id Software. *Quake III: Arena*. Tech. Rep. id Software, Inc., Mesquite, TX, 2002; www.idsoftware.com/games/quake/quake3-arena/.
5. Kroll, E., Lenz, E., and Wolberg, J. Rule-based generation of exploded views and assembly sequences. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* 3, 3 (1989), 143–155.
6. Mohammad, R. and Kroll, E. Automatic generation of exploded views by graph transformation. In *Proceedings of IEEE AI for Applications* (1993), 368–374.
7. Mohr, A. and Gleicher, M. Noninvasive, interactive, stylized rendering. In *Proceedings of the ACM Symposium on Interactive 3D Graphics* (Chapel Hill, NC, Mar. 19–21). ACM Press, New York 2001, 175–178.
8. Niederauer, C., Houston, M., Agrawala, M., and Humphreys, G. Non-invasive interactive visualization of dynamic architectural environments. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (Monterey, CA, Apr. 28–30). ACM Press, New York, 2003, 55–58.
9. Raab, A. and Ruger, M. 3D-ZOOM interactive visualization of structures and relations in complex graphics. In *3D Image Analysis and Synthesis*, G. Girod, H. Niemann, and H. Seidel, Eds. Verlag, Sankt Augustin, Germany, 1996, 87–93.
10. Shoemake, K. ArcBall: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of Graphics Interface '92* (Vancouver, Canada). Morgan Kaufmann Publishers, Inc., 1992, 151–156.
11. Thomas, T. *Technical Illustration, 3rd Ed.* McGraw Hill, New York, 1978.
12. Tufte, E. *Visual Explanations*. Graphics Press, Cheshire, CT, 1997.

MIKE HOUSTON (mhouston@graphics.stanford.edu) is a Ph.D. candidate in the Stanford University Computer Graphics Laboratory in the Department of Computer Science at Stanford University in Stanford, CA.

CHRISTOPHER NIEDERAUER (ccn@ccntech.com) is an OpenGL software engineer in the Graphics and Imaging Group at Apple Computer, Inc., in Cupertino, CA.

MANEESH AGRAWALA (maneesh@graphics.stanford.edu) is a researcher in the Microsoft Research Document Processing and Understanding Group in Redmond, WA.

GREG HUMPHREYS (humper@cs.virginia.edu) is an assistant professor in the Department of Computer Science at the University of Virginia in Charlottesville.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.