Non-Invasive Interactive Visualization of Dynamic Architectural Environments

Christopher Niederauer* University of California, Santa Barbara
Mike Houston† Stanford University
Maneesh Agrawala‡ Microsoft Research
Greg Humphreys§ University of Virginia

Abstract

We present a system for interactively producing exploded views of 3D architectural environments such as multi-story buildings. These exploded views allow viewers to simultaneously see the internal and external structures of such environments. To create an exploded view we analyze the geometry of the environment to locate individual stories. We then use clipping planes and multipass rendering to separately render each story of the environment in exploded form. Our system operates at the graphics driver level and therefore can be applied to existing OpenGL applications, such as first-person multiplayer video games, without modification. The resulting visualization allows users to understand the global structure of architectural environments and to observe the actions of dynamic characters and objects interacting within such environments.

Keywords: Visualization, Exploded View, Architecture

1 Introduction

Recent advances in consumer graphics technology now permit the interactive exploration of extremely complex 3D environments. Architects routinely build detailed 3D CAD models of buildings to visualize both their internal spaces and external structures. The trend towards greater complexity is even more evident in the video game industry, which has been creating increasingly compelling three-dimensional worlds for the past few years. These environments are vast and tend to be densely occluded.

However, most 3D model viewing applications lack the ability to simultaneously display the interior spaces and the external structure of the environment. Two common interfaces for interactively viewing such environments are "ArcBall" interfaces [Shoemake 1992] that allow rotation, scaling, and zooming of the environment, and walkthrough interfaces [Teller 1992] that allow the viewer to move through the rooms inside the environment. ArcBall interfaces are useful for understanding the environment's external structure, while walkthrough interfaces are useful for understanding its interior spaces. However, neither of these interfaces allow 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* [Id Software c. 2002], the occlusions make it impossible to see all the action at once.

*ccn@ccntech.com
†mhouston@graphics.stanford.edu
‡maneesh@graphics.stanford.edu
§humper@cs.virginia.edu

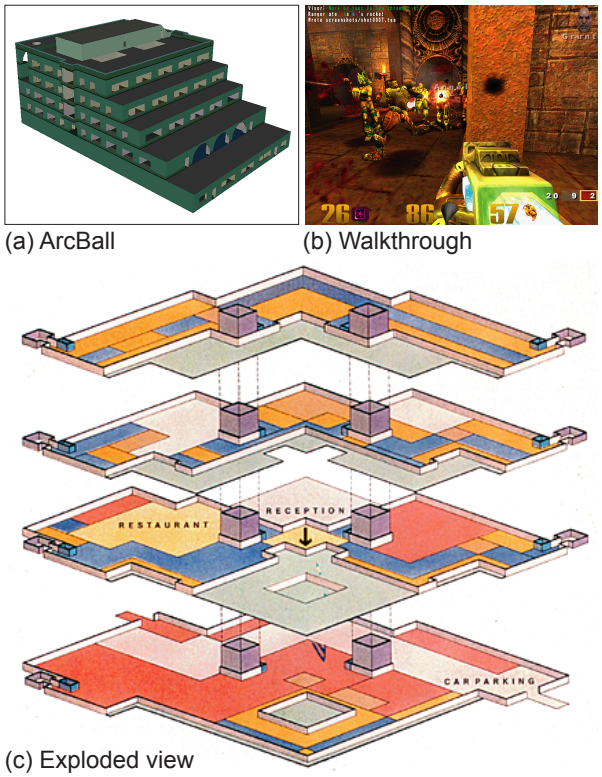

Figure 1: Visualizations of architectural environments. (a) An ArcBall interface exposes exterior structure. (b) A walkthrough [Id Software c. 2002] reveals the interior structure of a single room. (c) A hand-designed exploded view [Holmes 1993] sections the building into individual floors, or stories, and allows viewers to simultaneously see its interior and exterior structure. Our visualization system provides automated support for generating such exploded views of an architectural environment.

Architects and technical illustrators often use techniques such as cutaways, transparency, and exploded views to reduce or eliminate occlusion and expose the overall structure of architectural environments. As shown in figure 1(c), a common approach is to first section the building into stories just below the ceilings, and then separate the stories from one another to form an exploded view. These views expose both the structure of the internal spaces within each story and the vertical spatial relationships between adjacent stories. But producing such an exploded view from a 3D model of an architectural environment currently requires a designer to annotate the location of each story and a viewing application that can generate the appropriate exploded view from the annotated model.

In this paper we present an interactive system that provides automated support for generating exploded views of any architectural environment. Moreover, our system requires very little semantic understanding of the environment. We assume the environment is "architectural," and search for geometric primitives representing ceilings. Once we have found the ceilings, we section the environment into stories and render each story separately in exploded form. Our system provides interactive control over the viewpoint

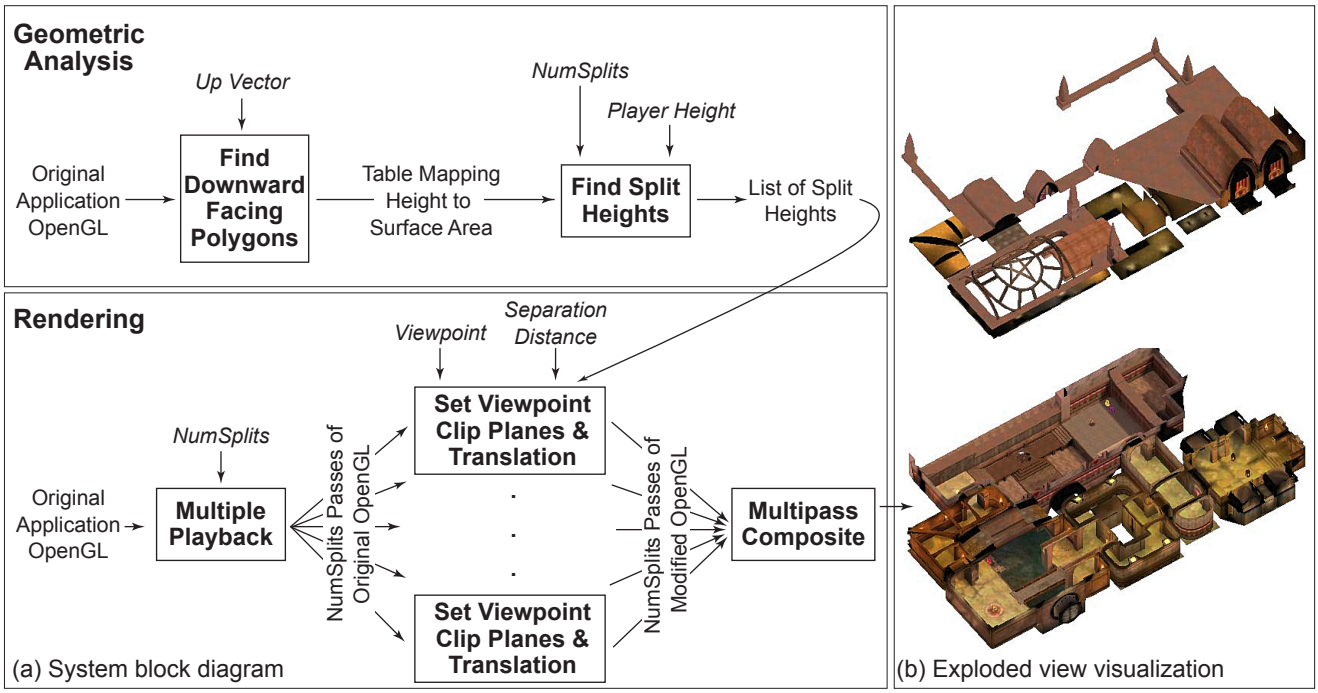

Figure 2: Overview of our visualization system. (a) The system is divided into two stages; geometric analysis and rendering. Geometric analysis occurs once and determines where to section the architectural environment into stories. Rendering occurs every frame and produces an interactive exploded view. Each box represents a computational operation and arrows represent data. The data shown in italics, *(Up Vector, Player Height, NumSplits, Viewpoint and Separation Distance)* are supplied by the user. All other data are supplied by the original OpenGL application or computed by our system. (b) An exploded view of the `demo2` environment from *Quake III* generated non-invasively by our system.

and the separation distance between the stories in the environment. We have found that this type of viewer makes it easy to understand both the structure of the environment and the relationship between dynamic objects and characters in the space. To make our technique widely applicable, we have implemented it *non-invasively* using Chromium [Humphreys et al. 2002] to intercept and manipulate sequences of OpenGL commands made by the underlying application. Therefore our system can be applied to existing applications, such as *Quake III*, without modification or even recompilation.

2 Related Work

Architects and technical illustrators invariably include sketches, diagrams, and schematic drawings of 3D objects or environments to explain their parts, spaces, and structures. Exploded views are especially effective for conveying structure, and are commonly used in illustrations of mechanical assemblies [Karns and Traister 1995; Giesecke et al. 1949; Thomas 1978]. In computer graphics and visualization, several groups have developed techniques for generating exploded views of such objects [Kroll et al. 1989; Mohammad and Kroll 1993; Rist et al. 1994; Driskill 1996; Raab and Rüger 1996]. These systems typically require extensive knowledge about the part/sub-part decomposition of the model.

Designers such as Biesty [1996] and Tufte [1997] commonly use exploded views to reveal the structure of multi-story buildings, Yet, most of the computer graphics research on visualizing architectural environments has focused on building interactive walkthroughs of extremely large models containing millions of primitives [Teller 1992; Baxter et al. 2002]. These systems allow viewers to experience the environment at a human scale from a first-person perspective. Such systems exploit the occlusion characteristics of such environments to cull geometry that won't be visible from a given viewpoint. In contrast, the goal of our work is to provide a third-person perspective visualization that shows as much of the interior and exterior structure of the environment as possible.

Recently, Mohr and Gleicher [2001; 2002] introduced the idea of using non-invasive techniques for manipulating existing graphics applications. Our system builds on this idea. We use Chromium [Humphreys et al. 2002] to intercept commands made by any OpenGL application. Through Chromium's Stream Processing Units (SPUs), we can modify, delete, replace, or augment any of the graphics API calls made by an application while it is running. Chromium's SPUs can easily be adapted to affect semantic transformations on streams of graphics commands; Humphreys et al. implement a hidden-line drawing SPU that can easily be applied to any OpenGL application. Because our application requires state tracking, command packing/unpacking, and OpenGL stream analysis, Chromium is a natural choice to support the algorithms we describe in this paper.

3 Generating Architectural Exploded Views

Our visualization system generates exploded views of an architectural environment using the two stage approach shown in figure 2. The system processes an OpenGL stream from any application[1] in two stages. The first stage, *geometric analysis*, determines where to split the architectural model into 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 renderer inserts clipping planes between each story and performs multipass rendering, one pass per story, to produce the exploded view. The renderer also re-

[1]We assume the original application is using OpenGL to render an architectural environment.

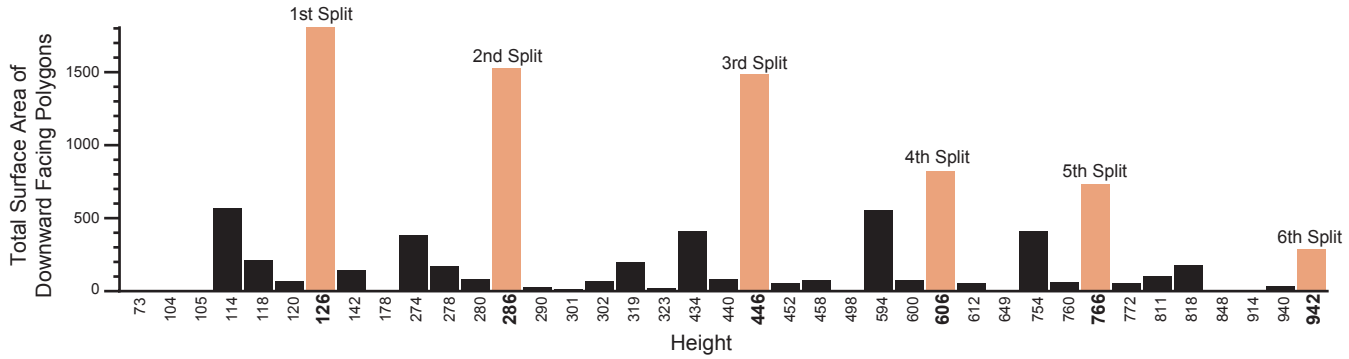

Figure 3: This plot graphically depicts the table mapping heights to surface area of downward facing polygons for the *Soda Hall* application. The corresponding exploded view is shown in figure 4. The orange bars show where our system chose to split the building. Note that the surface area corresponding to the 6th split is smaller than the surface areas for heights 114, 274, 434, 594 and 754. For this environment we set *Player Height* to 100. Since all those heights are within *Player Height* of a previously chosen split they are not chosen as the 6th split height.

places the viewpoint (i.e. the projection and modelview matrices) specified by the original application with a new viewpoint which may be interactively specified by a viewer using our system. The rendering stage modifies every frame of the original application on the fly. We describe both of these stages in detail.

3.1 Geometric Analysis to Locate Stories

The most natural segmentation of an architectural model is into floors, or *stories*. A story is usually defined by a floor, a ceiling, and all geometry in-between. For our visualization, however, we do 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 immediately below each ceiling. The goal of the geometric analysis stage is to determine the polygons of the environment that represent ceilings. The rendering stage will insert a clipping plane into the environment just below each discovered ceiling to separate the environment into individual stories.

To find the ceiling polygons, we require the user to specify a vector defining the *up direction* for the environment. Ceiling polygons will be oriented so that their normals point in the opposite direction of this vector. As the original application submits geometry to OpenGL for rendering, we use Chromium to intercept the vertices (v_1, v_2, v_3,...) of each polygon. Assuming polygons are specified using consistent counterclockwise ordering, we compute the polygon normal as the cross product $(v_2 - v_1) \times (v_3 - v_2)$. Most applications are careful about their geometry's winding order because of 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 portions of characters or objects, or smaller architectural elements such as window sills, ledges, and ornamental decoration. To find the polygons most likely to represent ceilings, we compute the height of each downward facing polygon as the dot product of v_1 and the *up vector*. We then build a table mapping each potential ceiling height to the total surface area of all downward facing polygons at that height. A plot of this table for the Soda Hall model is shown in figure 3.

The viewer interactively specifies *NumSplits* – the number of stories the environment should be split into. Starting with an unsplit environment the viewer presses a key to increase or decrease *NumSplits*. Initially, we find the *NumSplits* largest surfaces areas in the height to surface area table and consider splitting the environment at each of the corresponding heights. However, environments often contain large ceiling areas that are slightly offset from one another. Splitting the model at each of these offset heights would generate extremely short stories that are not part of the desired segmentation. To counteract this effect, we apply an additional heuristic which maintains a minimum distance 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 story can be smaller than this minimum height. We allow the user to adjust this minimum height as necessary for a particular model. In practice we have found that finding the right minimum height parameter is extremely easy. For multiplayer games the minimum height is specified as the average height of the player geometry and therefore we call this variable the *Player Height*.

3.2 Rendering the Exploded View

After the geometric analysis stage determines where to split the environment into stories, the *rendering stage* modifies each frame of the original application to produce an interactive exploded view. We use Chromium to buffer the stream of OpenGL calls corresponding to a frame (i.e. all the functions and their parameters between calls to glSwapBuffers()). The frame is then replayed *NumSplits* times, with each playback pass responsible for rendering one of the stories in the exploded view. Each playback stream modifies three aspects of the original OpenGL stream:

- The original viewing projection is replaced by an external axonometric view.
- Clipping planes are inserted into the stream to ensure that only a single story is drawn.
- 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. We generate an axonometric view by replacing the original application's projection matrix with our own axonometric projection. Our system allows viewers to interactively adjust the viewpoint using an ArcBall interface [Shoemake 1992]. To allow this kind of control, we must locate the viewing transformations in the transformation matrix stack of the original application and replace them with our own transformations. We assume that the application first sets up the viewing transformation for the environment and subsequent changes to the modelview stack represent relative motions of other graphical elements, such as players, objects, or overlays. Thus we can change the viewpoint by replacing the very first matrix placed on the OpenGL "modelview" stack. When non-environmental graphical elements (players, objects, etc.) are drawn, we use the inverse of the environment's original projection matrix to place these elements correctly relative to our new axonometric view.

To ensure each playback stream only draws the geometry associated with a single story, we insert two OpenGL clipping planes into the graphics stream just before the environment geometry. One clipping plane is placed immediately below the ceiling of the current story so that it clips all geometry above it. Similarly, the other clipping plane is placed right below the previous ceiling so that it clips all geometry below it.

Viewers can interactively specify the story separation distance. To set the specified separation distance, we insert a global translation into each playback stream to move the entire scene along the *up vector*. By interactively adjusting separation distance, viewers can quickly see how the stories fit together and connect with one another. This is another form of interaction that serves to expose the 3D structure of the environment.

4 Results

We have tested our system using two existing OpenGL applications for rendering architectural environments: a basic ArcBall-based model viewer we wrote to display the Soda Hall environment, and *Quake III*. Examples of exploded views generated by our system are shown in the color plate. Notice that the corresponding unexploded view of each of these environments reveals little internal structure.

Since we are modifying the viewpoint without the application's knowledge we must disable any geometric culling performed by the application. For *Quake III* we set the following options to disable such culling: `r_nocull 1`, `r_novis 1`, and `r_facePlaneCull 0`.

While disabling culling allows us to non-invasively capture the entire architectural model, rendering the exploded view can become expensive because it requires one pass through the geometry for each story in the building. To alleviate this problem we use Chromium's parallel rendering capabilities with a cluster of 8 workstations each containing two 800 MHz Pentium III Xeon processors and an NVIDIA GeForce4 graphics accelerator. Each node in the cluster renders one story of the environment and their results are composited using Chromium's `binaryswap` SPU [Ma et al. 1994]. Using the cluster we can render exploded views of both Soda Hall and the tourney4 environments at about 9 frames per second.

5 Discussion

In this work, we explicitly chose to develop a non-invasive approach so that we could provide spectators with an exploded view of popular multi-player game environments. Although our ability to retrofit existing applications in this manner is a strength of our technique, it is also a limitation. In particular, this approach requires us to analyze the environment at a very low level and to make several assumptions about the semantics of the OpenGL stream issued by the original application. For example, we assume that the vertices of geometric primitives are specified in consistent counterclockwise order, and that the first matrix added to the modelview transformation stack encodes the viewing transform.

The application writer has access to higher-level semantic knowledge about the environment, such as the locations of the ceilings and the viewing parameters. Access to this type of information would make it easy to build our exploded view visualization technique into the original application. Our intention is not to simply argue that a non-invasive technique should be used in all situations, but rather to demonstrate a compelling new visualization technique for architectural environments. We hope to encourage designers of future systems to directly incorporate such visualizations into their applications. For existing applications such as *Quake III* which do not provide such an exploded view visualization mode, we believe that our non-invasive system approach is a good strategy for adding exploded view capabilities.

6 Future Work and Conclusion

There are several directions for future research. Our system is currently not fully automated and requires some user input, such as the number of stories and the minimum story separation. While it is very easy to set these parameters interactively, we are working on a weighted clustering algorithm to automatically determine the proper split locations without any user intervention.

When many players are interacting simultaneously in a large environment, it can be difficult to follow all the action, even though all of it is visible in the exploded view. A combination of geometric and semantic simplification would greatly increase the comprehensibility of such an environment. For example, players often appear quite small when the entire map is shown, and could be simplified or even iconified without sacrificing much content.

We have presented a system that exposes the internal 3D structure of architectural models by automatically generating exploded views. The visualization provides a much clearer view of the overall environment and the dynamic character interactions that may occur within it. We have found that observing multi-player games from this type of third-person perspective is much more satisfying than watching the game through the eyes of a player, as it provides a more complete understanding of the environment and the action.

References

BAXTER, W. V., SUD, A., GOVINDRAJU, N. K., AND MANOCHA, D. 2002. GigaWalk: Interactive walkthrough of complex environments. In *Eurographics Rendering Workshop*.

BIESTY, S., AND PLATT, R. 1996. *Stephen Biesty's Incredible Explosions: Exploded Views of Astonishing Things*. Scholastic.

DRISKILL, E. 1996. *Towards the Design, Analysis and Illustration of Assemblies*. PhD thesis, University of Utah.

GIESECKE, F. E., MITCHELL, A., AND SPENCER, H. C. 1949. *Technical Drawing 3rd Edition*. MacMillan.

HUMPHREYS, G., HOUSTON, M., NG, R., AHERN, S., FRANK, R., KIRCHNER, P., AND KLOSOWSKI, J. T. 2002. Chromium: A stream processing framework for interactive graphics on clusters of workstations. *ACM Transactions on Graphics 21*, 3, 693–702.

ID SOFTWARE. c. 2002. Quake 3: Arena. Tech. rep., Id Software Inc. `http://www.idsoftware.com/games/quake/quake3-arena/`.

KARNS, J. A., AND TRAISTER, J. E. 1995. *Firearms Disassembly with Exploded Views*. Stoeger Publishing Company.

KROLL, E., LENZ, E., AND WOLBERG, J. R. 1989. Rule-based generation of exploded views and assembly sequences. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM) 3*, 3, 143–155.

MA, K.-L., PAINTER, J. S., HANSEN, C. D., AND KROGH, M. F. 1994. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications 14*, 4 (July), 59–68.

MOHAMMAD, R., AND KROLL, E. 1993. Automatic generation of exploded views by graph transformation. In *Proceedings of IEEE AI for Applications*, 368–374.

MOHR, A., AND GLEICHER, M. 2001. Non-invasive, interactive, stylized rendering. In *ACM Symposium on Interactive 3D Graphics*, 175–178.

MOHR, A., AND GLEICHER, M. 2002. HijackGL: Reconstructing from streams for stylized rendering. In *International Symposium on Non-Photorealistic Animation and Rendering*, 13–20.

RAAB, A., AND RÜGER, M. 1996. 3D-ZOOM interactive visualization of structures and relations in complex graphics. In *3D Image Analysis and Synthesis*, 87–93.

RIST, T., KRÜGER, A., SCHNIEDER, G., AND ZIMMERMAN, D. 1994. AWI: A workbench for semi-automated illustration design. In *Proceedings of Advanced Visual Interfaces (AVI)*, 59–68.

SHOEMAKE, K. 1992. Arcball: A user interface for specifying three-dimensinal orientation using a mouse. In *Graphics Interface '92*, Canadian Information Processing Society, 151–156.

TELLER, S. 1992. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, University of California at Berkeley.

THOMAS, T. A. 1978. *Technical Illustration 3rd Edition*. McGraw Hill.

TUFTE, E. 1997. *Visual Explanations*. Graphics Press, 146–148.

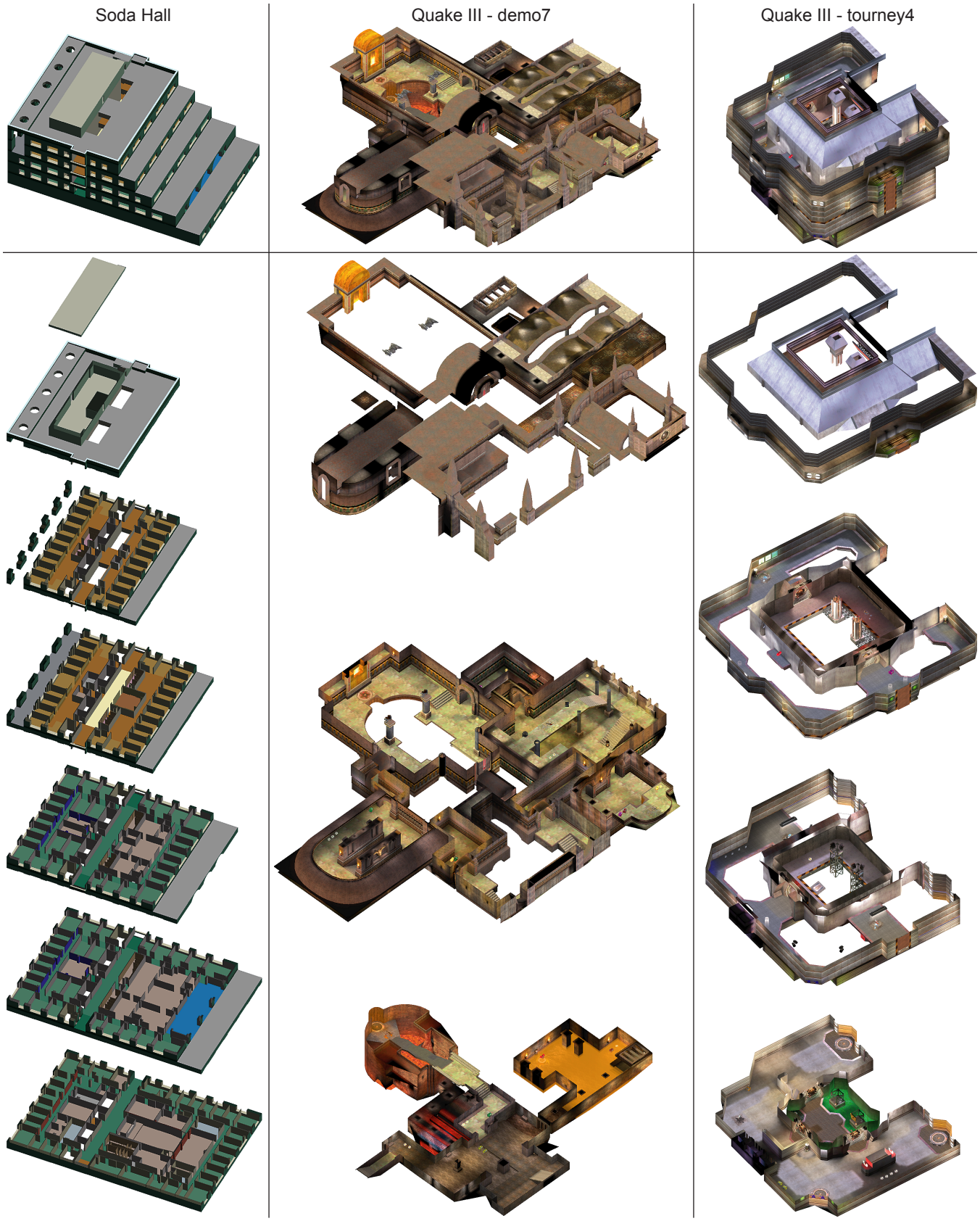

Figure 4: Exploded views generated by our system. Each column shows the environment from an external axonometric viewpoint, first unexploded and then exploded. We non-invasively modified an ArcBall viewer to generate the Soda Hall example and we modified *Quake III* to generate the `demo7` and `tourney4` examples. While the unexploded view shows external structure, the exploded view simultaneously reveals both internal and external structure.