MODELING, SIMULATION, AND VISUALIZATION: THE PENTAGON ON SEPTEMBER 11TH

Researchers used a custom importer to simplify and load simulation data from the September 11th Pentagon attack into a commercial animation system. The resulting high-quality impact visualization combines state-of-the-art graphics with a state-of-the-art engineering simulation.

Simulations are essential tools in many fields of science and engineering. Researchers use them to crash-test automobiles before building them, to study the interaction between a hip implant and the femur, to evaluate and renovate medieval bridges, to assess the effectiveness of electronic circuit packaging, and to build virtual wind tunnels. In particular, finite-element analysis (FEA) plays a prominent role in engineering. FEA systems compute a variety of physical parameters over the simulation's time span, such as position, velocity, acceleration, stress, and pressure. The visual presentation of the results then goes to generic postprocessors or for studying specific scientific visualization contexts.

Three-dimensional computer graphics has advanced tremendously, driven mostly by the popularity of entertainment applications. Consumer-level-priced personal computers with add-in graphics cards can produce high-quality images of complex 3D scenes at interactive rates and run sophisticated animation software systems that provide (offline) video sequences that closely approach photorealism. Because of real-world commercial applications, most animation systems focus on minimizing the animation's production effort and maximizing its entertainment value—essentially, rendering quality and expressivity, not the laws of physics. Put succinctly, if it looks good, it *is* good.

Our team's goal was to produce a visualization of the September 11th attack on the Pentagon that combined commercial FEA codes and animation. The project had two distinct parts. During the first phase, we designed, tested, and ran at full scale the FEA simulation of the aircraft hitting the building. For this, we chose LS-DYNA (www.lstc.com), a commercial FEA system often used in crashworthiness assessment simulations. Another choice we looked at was MSC-DYTRAN (www.mscsoftware.com). Both codes implement explicit time-marching schemes, but LS-DYNA can handle geometric and material nonlinearities as well as fluid–structure interaction.[1] Implicit time-integration schemes would be inappropriate for impact problems that involve high-frequency response.[2] In the second phase, our efforts focused on producing a high-quality visualization of the massive data from the simulation. To do this, we created a scalable link between the FEA system and a commercial animation system called 3ds max (www.discreet.com/products/3dsmax/). You can use the link to create animations with physical fidelity regardless

1521-9615/04/$20.00 © 2004 IEEE
Copublished by the IEEE CS and the AIP

CHRIS HOFFMANN, VOICU POPESCU, SAMI KILIC, AND METE SOZEN
Purdue University

of the scientific or engineering domain.

Ironically, the model's geometric data came from a 3ds max model that we hand-imported into the FEA analysis. Based on our experience, we have several ideas about how to make this link at least semiautomatic and what tools other researchers can use to simplify this process, which is the most time-consuming part of the simulation and analysis step. Our work goes a long way toward accelerating the overall cycle of modeling, simulation, animation, and model refinement, which is what we describe in this article.

Purpose

High-quality visualization of scientifically accurate simulation data has at least three important advantages:

- It can effectively communicate results and insights to the public at large.
- It has the ability to build a case for a project's merits, especially for large-scale projects that require grass-roots support.
- It offers better collaboration among members of interdisciplinary teams.

A high-quality visualization of a simulation's results requires state-of-the-art rendering techniques. A second requirement is that the simulation be placed in the context of the immediate surrounding scene. For this, we must model and render the scene along with the simulation results. Such visualization makes the simulation's results and conclusions directly accessible to people outside the simulation community without sacrificing scientific accuracy.

A good visualization ultimately leads to improvements on the simulation itself: high-quality images quickly reveal discrepancies with observed experimental data. Scientific simulations will soon be powerful tools routinely used in a variety of fields including national security, emergency management, forensic science, and the media.

A Step-by-Step Process

In previous research, M. Pauline Baker and her colleagues described the simulation of a bomb blast and its impact on a neighboring building.[3] The scenario they investigated matched the 1996 attack on the Khobar Towers in Saudi Arabia. The authors first computed the blast propagation using CTH shock-physics code[4] at the US Army's research lab in Vicksburg (see www.hpcmo.hpc.mil/Ttdocs/UGC/UGC98/papers/3b_chal/). They then used the results of the CTH calculation as initial pressure loadings on the buildings; they next used Dyna3D to model the building's structural response to the blast.[5] The authors visualized the results in the Dyna3D postprocessor and the Visualization Toolkit (see http://public.kitware.com/VTK), using standard slicing and isosurfacing visualization techniques. They reported the difficulty of visualizing large data sets, and solved the problem by using reduced resolution, decimation, and extraction of smaller regions of interest. They concluded by mentioning they would attempt to enhance the visualization's quality by using photographs.

A considerable body of literature in nuclear engineering is dedicated to simulating the crash of an aircraft into a concrete structure. In fact, provisions for aircraft impact on reinforced concrete structures are incorporated into the civil engineering codes used in nuclear-containment structure design. Takahiro Sugano and his colleagues conducted a full-scale test to measure the impact that fighter aircraft could exert on a reinforced concrete target slab.[6] The study provided important information on the aircraft's deformation and disintegration, and gave experimental evidence that the airframe and the aircraft's skin alone did not likely cause major damage on reinforced concrete targets.

Our first step in simulating the Pentagon attack was to generate finite-element meshes suitable for FEA. To keep the scene complexity within manageable limits, we meshed only the most relevant components of the aircraft and the building. Then, we tuned the material models during test simulations to achieve correct load-deflection behavior. Finally, we ran the FEA code on full-resolution meshes to simulate the first 250 milliseconds of the impact over 50 recorded states.

The visualization part of the project began with modeling the Pentagon from architectural blueprints via AutoCAD. We enhanced the geometric model of the building and its surroundings with textures projected from high-resolution satellite and aerial imagery by using a custom tool. We obtained the 3ds max aircraft model we used for visualizing the approach from a 3D gaming company (Amazing 3D graphics, www.amazing3d.com). A custom plug-in simplified, converted, and imported into the animation system the 3.5 Gbytes of state data that described the mesh deformations. The system then aligned the imported meshes with the surrounding scene and enhanced them by rendering material properties. Finally, we rendered the integrated scene to produce the desired animations. Figure 1 shows the simulation results at the same time step from nearly identical views, once using our system that coupled the simulation with 3ds max, and once with the LS-DYNA software's postprocessor.

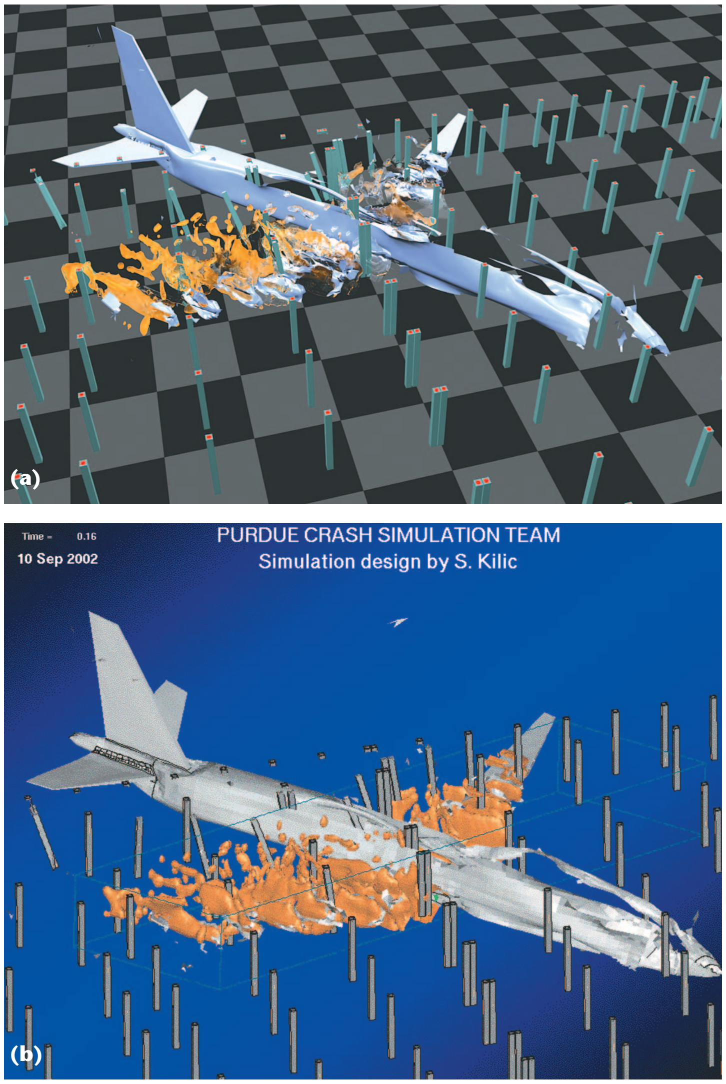

Figure 1. Visualization of the simulation. (a) The visualization our system produced, and (b) a similar view produced with the postprocessor.

To contextualize the simulation, we next had to model and render the Pentagon's surroundings. Research in image-based rendering (IBR) produced several successful approaches for rendering complex, large-scale, natural scenes. The QuicktimeVR system, for example, models the scene by acquiring a set of overlapping same-center-of-projection photographs that it stitches together to form panoramas.[7] During rendering, the desired view is confined to the centers of these panoramas. We had to allow for unrestrained camera motion, so we dismissed this approach.

IBR by warping (IBRW) relies on images enhanced with per-pixel depth.[8] The depth and color samples are 3D warped (reprojected) to create novel views. Airborne LIDAR (which stands for LIght Detection And Ranging) sensors can provide the depth data at appropriate resolution and precision. No depth maps of the Pentagon scene were available for our project, so we couldn't use IBRW. In light-field rendering, the scene is modeled with a database containing all the rays potentially needed during rendering. However, this method does not scale well: the number of images that must be acquired as well as the ray database grow to impractical sizes for large-scale scenes.

An approach frequently used for modeling large urban scenes combines images with coarse geometry into a hybrid representation. A representative example is the Façade system,[9] which maps photographs onto buildings modeled with simple primitive shapes. Paul Debevec and his colleagues used this system to model and realistically render a university campus environment.[9] The Pentagon building's relatively simple geometry and the availability of photographs of the area motivated us to choose such a hybrid geometry–image approach.

Large-Scale Simulation

LS-DYNA is a nonlinear finite-element code that can model fluid-structure interaction by using a Lagrangian mesh for solid elements and an Eulerian mesh for fluids.

The fluid's motion is based on the Navier-Stokes equations from computational fluid dynamics; advection transfers the fluid mass among Eulerian cells. LS-DYNA reports each cell's fractional occupancy as the solution progresses from one time step to the next. Figure 2 shows the part of the Eulerian mesh consisting of nonempty cells at the initial state.

When the different parts of the model that approach each other come into contact, time-step size scales down further to capture more accurately the nonlinear behavior of the large deformations and material failure. Time-step reduction on contact significantly influences the computation's running time.

In our overall analysis, we spent the most time on mesh generation and model assembly. The simulation for the largest mesh size took four days of runtime—mainly, because we had to use a very short time step to correctly capture the large deformations and to allow coupling of the Eulerian and Lagrangian meshes.

Two basic notions validate our simulation's results. The first is qualitative and based on a visual inspection of the behavior of the model's individual components. The fuselage's tail shell buckling in the study (see Figure 1a) is a typical example of visual inspection. Similarly, the fluid dispersion that occurs after the wings hit the building's columns provides an initial assessment of the results' admissibility. The

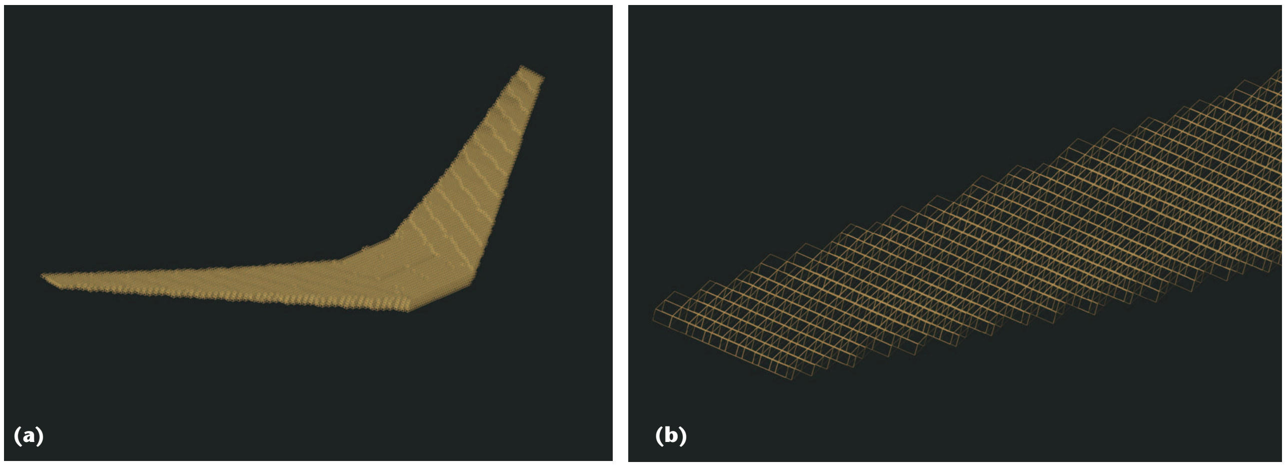

Figure 2. An Eulerian mesh defines the liquid fuel in the aircraft's wing and central tanks. The figure shows the cells occupied with fuel at the initial state for (a) the overall view and (b) a more detailed view. The figure shows only the cells that have fuel; in other words, all cells shown are nonempty.

fluid properties used in the simulation represent the typical kerosene jet fuel contained in wing tanks.

The second notion is based on quantitative evaluation of the model components' response against benchmark case studies. For the full simulation of the Pentagon with spirally reinforced concrete columns, we created a case study for investigating the behavior of a single column impacted by a block of fluid. We calibrated the constitutive concrete material model such that the column's response agreed with the results available from experimental studies and well-established hand-calculation models.

LS-DYNA's standard concrete models were not sufficiently accurate for our purpose, so we adjusted the nonlinear material model for concrete and selected a suitable erosion criterion. Without going into too much detail, we used test cases of single concrete pillars impacted with a block of fluid, as shown in Figure 3, and adjusted the model parameters accordingly, drawing on our previous experience with concrete behavior.[10]

We then adjusted the fluid properties to represent water. The main physical properties of the fluid used in the Eulerian approach are density and viscosity. An equation of state defines the pressure–volume relationship for the compressible fluid and its initial thermodynamic conditions.

The full simulation consisted of approximately 1 million nodes and took 68 hours of runtime on an IBM Regatta system for a simulation time of a quarter of a second. We modeled the reinforced concrete columns in the impact area's vicinity with higher fidelity than the columns further away; the Eulerian mesh for the liquid was finer than in earlier simulation runs we made.

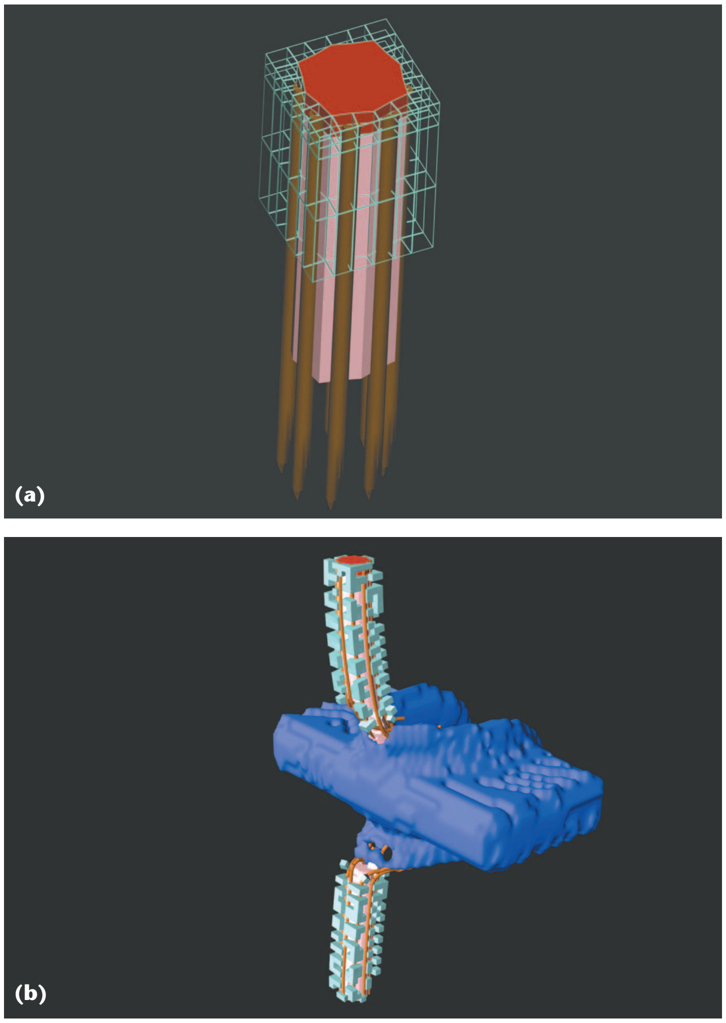

Figure 3. Finite-element mesh of a concrete column. (a) The confined concrete core (pink), rebars (brown), outer concrete fluff (light blue), and anchor (red). (b) Erosion of elements and column destruction caused by an impacting block of water.

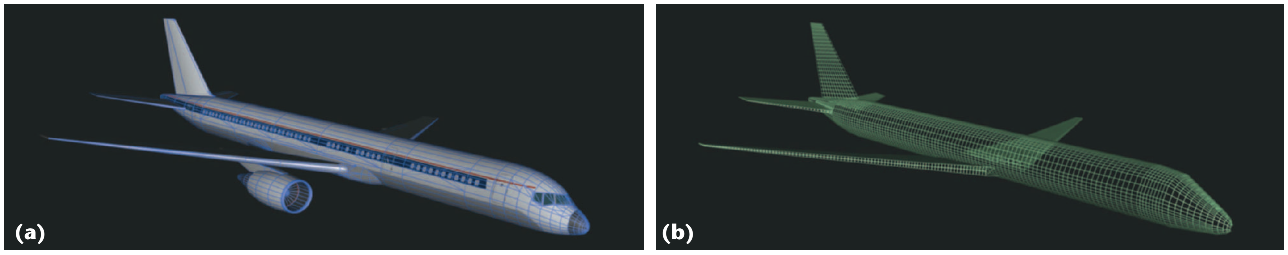

Figure 4. A 3ds max model. (a) The airplane model, and (b) the constructed finite-element mesh.

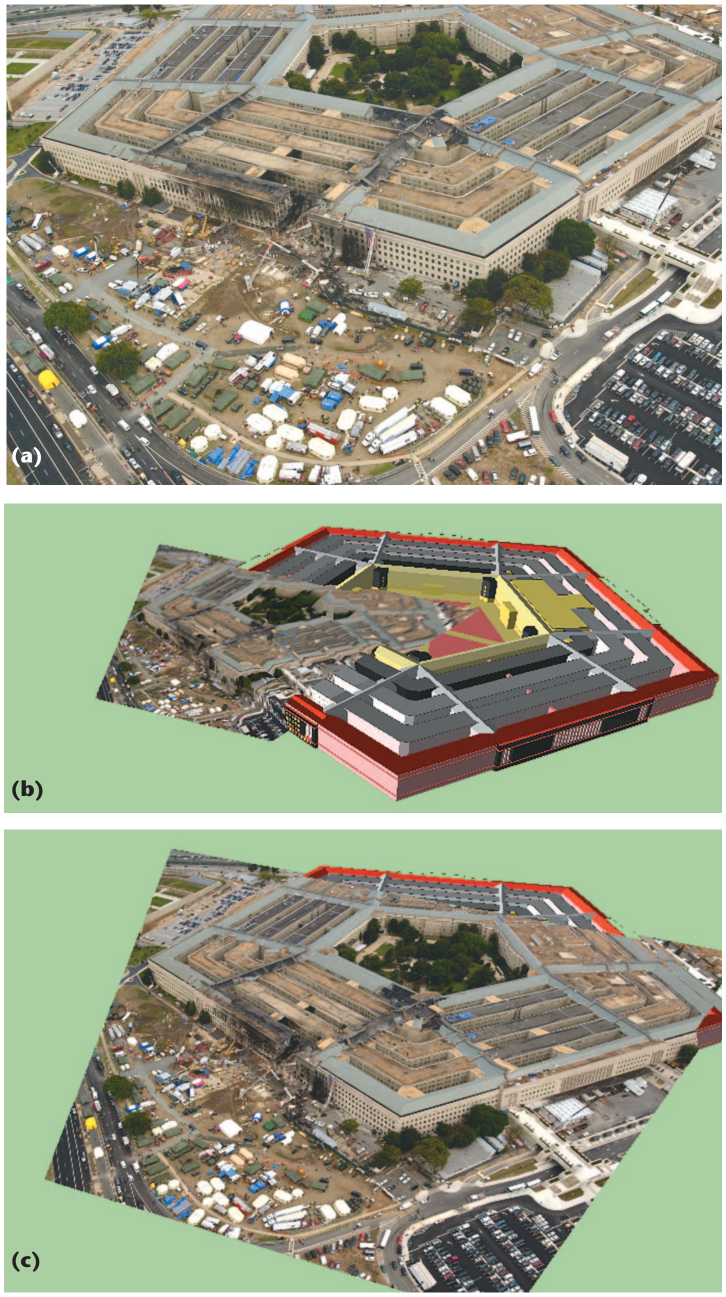

Figure 5. Camera matching. (a) The initial reference photograph, (b) the pose used to start the camera-matching search, and (c) the matching pose found by the search.

Mesh Generation

For the simulation, we generated a mesh via a set of custom programs. This choice reflected our difficulty in obtaining meshing tools that could generate hexahedral meshes for complex geometries. Our program tools are specific to the type of object generated—that is, separate tools generated individual column, wing, and aircraft-body meshes. This choice made meshing time-consuming, but we felt from the outset that mesh densities should be parametrically adjustable, and we wanted full control over how to do that.

With full control, we could experiment with different types of meshes and different densities of each mesh. For instance, instead of eroding shell elements with a maximum strain imposed, we could replicate the nodes and assign a lower strain-failure limit to coincident nodes. With this alternative, individual shell elements simply separated rather than entire shell elements eroding, so the system's mass remained constant.

We obtained the aircraft's mesh from a 3ds max model (see Figure 4). The model simplified the body to an ovoid cylinder fitted to two swept cones; the wings were composed from four hexahedral sections. We added in the main cabin's floor as well as stringers and ribs. We derived the geometry from a small set of hard points whose coordinates we read from the 3ds max model.

Typically, finite-element meshes simplify the geometric model by eliminating small details. This can be justified by the insignificant contribution such details make to overall structural behavior and integrity. In this spirit, we eliminated the rounding of the leading wing edges.

Surrounding Scene

Modeling the surrounding scene has forensic relevance because it enables a virtual reenactment of the events, which is important for corroborating eyewitness accounts and interpreting the low-resolution, slow-shutter video footage of whatever nearby surveillance cameras recorded. Modeling the surrounding scene also places the simulation results in context to make them easily understood by someone outside the investigation. The physically accurate, visually realistic animations we produced clearly document the destructive events.

As described earlier, we modeled the surrounding scene by using a hybrid geometry–image approach. From the architectural blueprints, we produced an AutoCAD model of the building and modeled the damage in the collapsed area by hand to match available photographs; we modeled the region surrounding the Pentagon as flat terrain. We enhanced the geometric models' color by using high-resolution satellite and aerial imagery (see www.spaceimaging.com/gallery/9-11/default.htm).[11]

Projective texture mapping colors the geometric primitives (or triangles) via photographs;[12] it's conceptually equivalent to transforming the camera into a projector. The rays emanating from the camera deposit pixel colors on the model's surface to automatically create the individual texture maps used during rendering. First, we must establish the position and orientation of the camera with which we acquired each of the reference photographs (see Figure 5). Second, we create from the reference photograph texture maps that uniformly sample each triangle. Note that we cannot use the reference photograph directly as a texture by projecting the vertices back in the camera view; we must eliminate the reference photograph's perspective distortion first.

Not having the camera at hand for intrinsic parameter calibration complicated camera matching for our study, so in addition to the camera pose's six extrinsic parameters, we also calibrated for the camera's focal length. We searched for the seven parameters using the downhill simplex method and a manually established initial guess. On a 3,000 × 2,000-pixel image, with 10 correspondences, the matching error was 3.5 pixels on average.

Once we knew the view, we could build the individual texture maps by first finding triangles visible in the photograph, and then for each visible triangle, allocating texture and setting each visible texel (a texture-map pixel) by projecting it in the reference photograph.

We collected visible triangles by rendering within an item buffer that stores IDs and depth. We determined the texture resolution by using the particular triangle's photograph area; the texture is defined in model space, so the texels uniformly sample the triangle, which removes the reference photograph's perspective projection. We determined the visible texels by using the item buffer. We textured partially visible triangles and invisible triangles from other photographs.

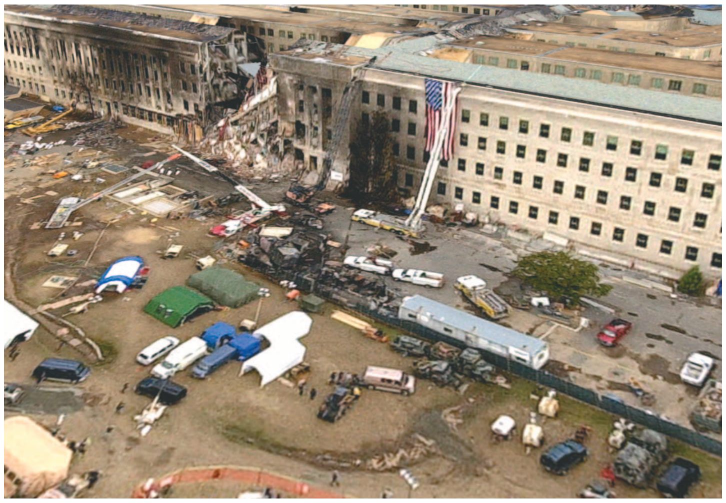
Figure 6. Novel view of the texture-sprayed model after rendering.

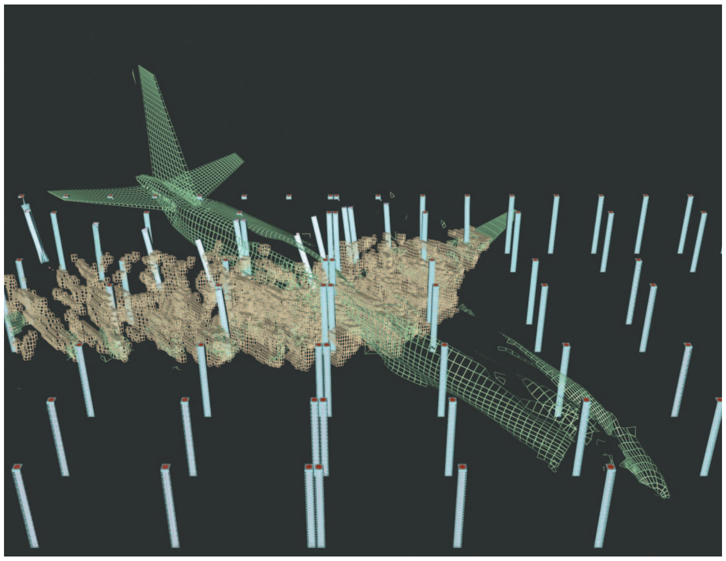
Figure 7. Wire-frame visualization of the simulation results.

We sprayed the building and ground plane model, which consisted of 25,000 triangles, with a 3,000 × 2,000-pixel photograph. The resulting texture-mapped model produced realistic visualizations of the Pentagon scene. Figure 6 shows an image rendered from a considerably different view than that of the reference photograph in Figure 5a. The texture files' total disk size was 160 Mbytes; the difference when compared to the reference photograph's 24 Mbytes was due to

- texels outside the triangle,
- texels corresponding to the hidden part of the triangle,
- thin triangles that have a larger texture than their area, and

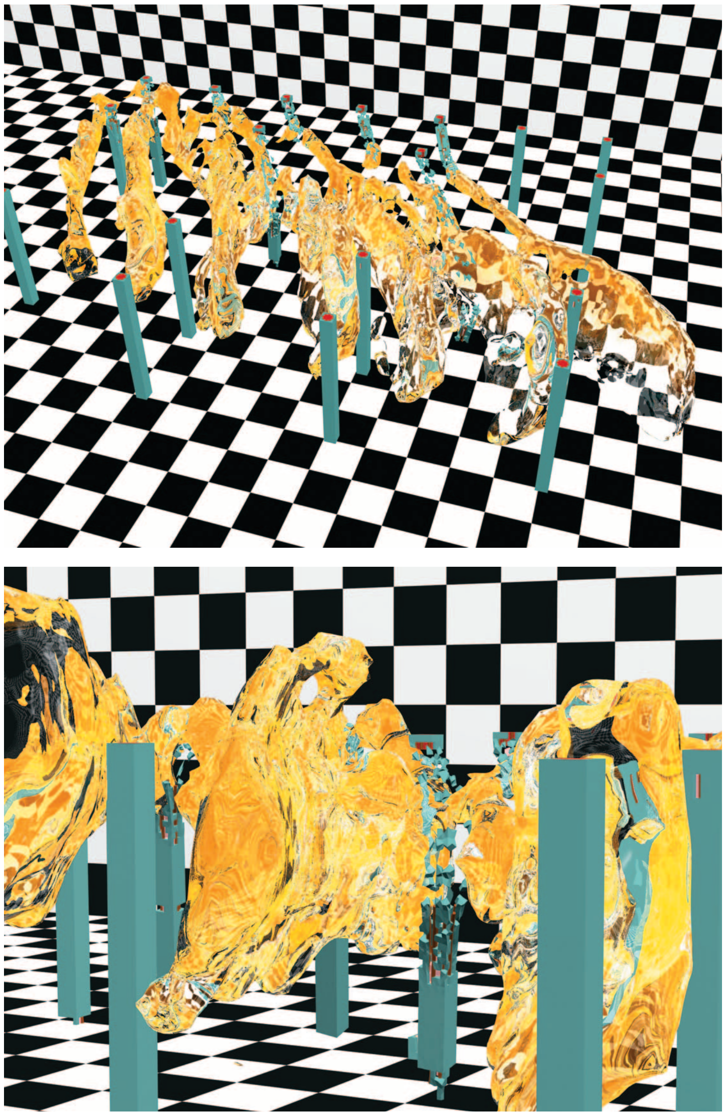
Figure 8. Two views of liquid–column impact visualization.

- our simple merging of individual texture images, which vertically collates 10 images to reduce the number of files.

We rendered the scene offline so the large total texture size is not a concern. For real-time rendering, though, we'd need to reduce the texture size. A simple greedy algorithm for packing the textures involving shifts and rotations would likely yield good results. The rotation could be propagated upstream to the spraying to avoid additional resampling.

Integration

3ds max directly imports simulation result files via a custom plug-in (see Figure 7). The 954,000 nodes of the FEM define the 355,000 hexahedral (solid) elements used to model the column core and the fluff, the 438,000 hexahedral elements for the liquid elements, the 15,000 quadrilateral (shell) elements used to define the fuselage and aircraft floor, and the 61,000 segment (beam) elements used to define the aircraft's ribs and stringers and the columns' rebars. The importer subdivides the simulation scene into objects according to material to facilitate assigning rendering materials.

Solid Objects

Ignoring the liquid for now, the 12, two, and one triangles per solid, shell, and beam elements, respectively, imply approximately 4.3 million triangles for the solid materials in the simulation scene. Eliminating internal faces—which are those shared by two hexahedral elements and that are irrelevant during rendering—reduces this number. Because elements erode, initially internal faces become visible at the fracture area. To account for this, we subdivided an object according to simulation state; for example, subobject *k* groups all the elements that erode at state *k*. We discarded each subobject's internal faces in linear time using hashing; this reduced the number of triangles to 1.3 million, which the animation system easily could handle.

Importing the mesh deformation into the animation system proved to be a serious bottleneck. The FEA code saves mesh deformations as node positions at every state. The animation system supports per-vertex animation via controllers that move a vertex on a linear trajectory. Because node movement is not linear in general, we could have created a controller for each of the 50 positions of each of the remaining 700,000 nodes to interpolate linearly between consecutive node positions. However, doing so took days of computing time, and the resulting scene file was unusable. The practical limit on the number of controllers is about 1 million.

We can reduce the number of animation controllers in two ways. First, we could make the importer not animate nodes with a total movement (sum of state-to-state movement) below a user-chosen threshold (typical value: 1 centimeter). Second, we could simplify each node's trajectories independently by eliminating (not creating) controllers for the nearly linear parts. After simplification, 1.8 million controllers remain. We distributed the simulation scene over three files, each covering one third of the simulation. Materials and cameras can, of course, easily be shared among several files. Importing the solid objects took two hours total, out of which we needed one hour for the third part of the simulation. Once the solid objects loaded, the animator assigned them standard 3ds max materials.

Liquid Objects

The liquid data saved at every state contains the Eulerian mesh's node position and the fractional occupancy values at that state. Volume-rendering techniques directly rendered the liquid from the occupancy data. We chose to build a surface-boundary representation first to take advantage of the animation system's rendering capabilities. For every state, the importer selects the Eulerian mesh elements that have a liquid occupancy above a certain threshold (typically 25 percent). We eliminated the internal faces similarly to the solid-object case. Once the liquid was imported, the animator used 3ds max tools, including mesh modifiers and complex ray-traceable materials, to produce compelling visualizations of the liquid. In Figure 8, refraction, surface reflections, attenuation, and variable opacity show the realism. Rendering at VGA resolution took approximately five minutes.

As in the case of solid objects, animating the liquid was challenging. We had two fundamental approaches: one, to consider the liquid to be a complex object that moves and deforms over the simulation time, or two, to frequently recompute the liquid object from the occupancy data, possibly at every animation frame.

The first approach is in the spirit of animation systems in which the same geometric entity suffers a series of transformations over the animation's time span. The geometric entity's state is known at the simulation states; we can compute it by thresholding or isosurfacing the occupancy data. To define a morph that produces the animation frames in between the states, however, we must first establish correspondences. This was challenging because the liquid can change considerably from one state to another; it implies that different numbers of vertices exist along with different local topologies (such as drops or liquid chunks separating and reuniting). We have attempted to implement this approach using the Eulerian mesh as a link between states. Because the occupancy values vary considerably from one frame to another, we generated many small liquid objects, which led to several position controllers.

The approach of defining the liquid with independent objects corresponding to snapshots along the simulation timeline proved to be more practical. Visibility controllers automatically generated by the plug-in define each object's appropriate life span. To smooth the transition, the objects were faded in and out at a negligible cost of four controllers per liquid object. Currently, the liquid is modeled with one object per state, with the 50 liquid objects totaling 1.5 million triangles. By interpolating the occupancy data, we could generate one snapshot for every animation step. When playing back the 50 states over 30 seconds at 30 Hz, though, we generated 900 liquid objects, which exceeds a practical geometry budget. We are investigating how to generate the liquid objects during rendering.

Our team's civil engineering researchers now regularly use the plug-in importer and 3ds max. Scientific simulation researchers and commercial-simulation-systems developers have shown great interest in the quality of the visualizations, and we have initiated several collaborations.

However, we must develop further the link between simulation and animation. The current bottleneck is the animation of the deforming meshes. Paradoxically, the animation system performs better if the animation is specified by geometry replication. We will continue to investigate this problem. Another direction is to extend the importer to create dust, smoke, and fire automatically. For example, when a concrete element erodes, it should be turned into fine debris or dust animated according to the momentum the element had before eroding. This simulation-driven reproduction of low-visibility conditions would be valuable in virtual training. Another direction for future work is to extend our current system to include classic visualization techniques such as isosurface enhancements.

Based on our experience with mesh generation, we have begun to devise an interactive script language by which to specify hexahedral meshes. The language can partially automate the meshing of complex geometries. We get scalability by using file operations when necessary, so the FEA model comes directly from the meshing operations. We could extend this concept to a more-automated mesh approach that closes the loop between meshing and model acquisition/inspection in 3ds max. Namely, we envision a set of plug-ins that lets a user select a component geometric structure in 3ds max (or designate a part of a model by drawing on it) and then generates the corresponding script for this structure. The resulting mesh could then be re-imported into 3ds max.

Acknowledgments

We thank Scott Meador for his help with 3ds max and Jim Bottum and Gary Bertoline for providing access to supercomputing facilities here and elsewhere in the US. William Whitson helped with the supercomputer runs; Hendry Lim and Mihai Mudure implemented texture spraying; Mary Moyars-Johnson and Emil Venere publicized this work; Amit Chourasia modeled the

Pentagon building; Jason Doty produced the first video illustration of our project; and Raj Arangarasan helped with an earlier implementation. Information Technology at Purdue at Computer Science Purdue, the US National Science Foundation, US Army Research Office, and DARPA supported this work.

References

1. M. Souli, "ALE and Fluid-Structure Interaction Capabilities in LS-DYNA," *Proc. 6th Int'l LS-DYNA Users Conf.* (Simulation 2000), Livermore Software Technology Corp., 2000, pp. 15–37.
2. K.J. Bathe, *Finite Element Procedures*, 2nd ed., 1995, Prentice Hall, p. 1037.
3. M.P. Baker, D. Bock, and R. Heiland, *Visualization of Damaged Structures*, Nat'l Ctr. for Supercomputing Applications, Univ. of Illinois, 1998; http://archive.ncsa.uiuc.edu/Vis/Publications/damage.html.
4. J.M. McGlaun, S.L. Thompson, and M.G. Elrick, "CTH: A Three-Dimensional Shock Wave Physics Code," *Int'l J. Impact Eng.*, vol. 10, 1990, pp. 351–360.
5. J.O. Hallquist and D.J. Benson, *Dyna3D User's Manual (Nonlinear Dynamic Analysis of Structures in Three Dimensions)*, tech. report #UCID-19592-revision-3, Lawrence Livermore Nat'l Laboratory, 1987.
6. T. Sugano et al., "Full-Scale Aircraft Impact Test for Evaluation of Impact Force," *Nuclear Eng. and Design*, vol. 140, 1993, pp. 373–385.
7. S. Chen, "QuicktimeVR—An Image-Based Approach to Virtual Environment Navigation," *Proc. SIGGRAPH '95*, ACM Press, 1995, pp. 29–38.
8. L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Proc. SIGGRAPH '95*, ACM Press, 1995, pp. 39–46.
9. P.E. Debevec, C.J. Taylor, and J. Malik, "Modeling and Rendering Architecture from Photographs," *Proc. SIGGRAPH'96*, ACM Press, 1996, pp. 11–20.
10. S. A. Kilic and M.A. Sozen, "Evaluation of Effect of August 17, 1999, Marmara Earthquake on Two Tall Reinforced Concrete Chimneys," *Am. Concrete Inst. Structural J.*, vol. 100, no. 3, 2003, pp. 357–364.
11. *Pentagon Building Performance Report*, Am. Soc. Civil Engineers, 2003.
12. M. Segal et al., "Fast Shadows and Lighting Effects Using Texture Mapping," *Computer Graphics*, vol. 26, no. 2, 1992, pp. 249–252.

Chris Hoffmann is a professor of computer science at Purdue University. His interests are in geometric computations, its applications, and visualization. He received his PhD from the University of Wisconsin, Madison. He also serves on the editorial boards of four scholarly journals. Contact him at cmh@cs.purdue.edu; www.cs.purdue.edu/people/cmh.

Voicu Popescu is an assistant professor of computer science at Purdue University. His interests are in image-based rendering, automated modeling, rendering, computer graphics, and architectures. He received his PhD from the University of North Carolina, Chapel Hill. Contact him at popescu@cs.purdue.edu.

Sami Kilic is a senior research scientist at the Computing Research Institute of Purdue University. His research interests are in structural dynamics and seismic response of reinforced concrete structures. He received a PhD from Stanford University and an MS from the University of California, Berkeley. He is a member of the IEEE, the American Society of Civil Engineers, the American Concrete Institute, and the Earthquake Engineering Research Institute. Contact him at skilic@purdue.edu.

Mete Sozen is the Kettelhut Distinguished Professor of Structural Engineering. His research interests are in the response of reinforced/prestressed concrete structures to static and dynamic loads. He received his PhD in civil engineering from the University of Illinois-Urbana. He is a member of the American Society of Civil Engineers, the Earthquake Engineering Research Institute, and is an honorary member of the American Concrete Institute. Contact him at sozen@purdue.edu.