Modeling Dynamic 3D Caves

Matt Boggus and Roger Crawfis

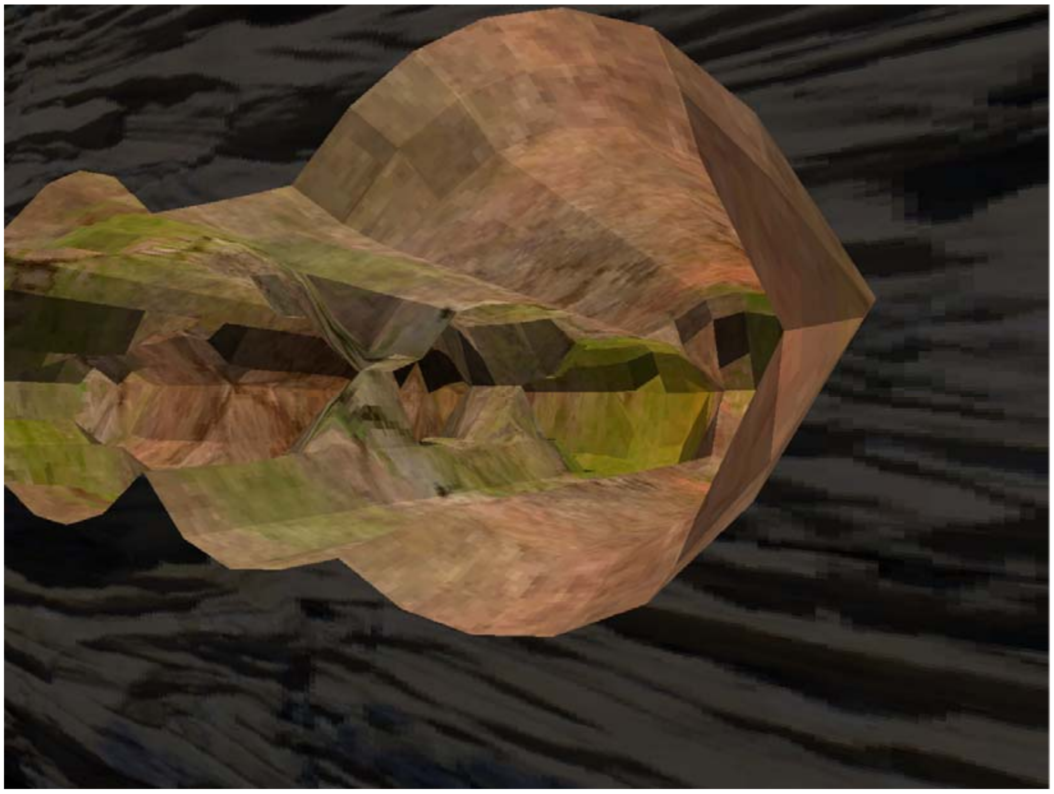

Figure 1. Screenshot

An example of a 3D cave environment created using our approach.

Abstract–A system for modeling 3D objects is presented. In this process, a new spatial data structure is introduced. At its core, the representation can be thought of as a set of slabs, each encoding a portion of the object along one axis. The goal of this work is to allow user modification of a cave environment at interactive rates, akin to existing terrain editing programs where the object is restricted to a single 2D manifold that represents a terrain surface. To achieve these rates, we use multiple fixed resolution grids that are used to provide explicit geometry for immediate mode rendering as well as simplifying the task of collision detection tests. We provide comparisons with existing cave visualization programs and techniques for dynamic models as well as examples of our technique in practice.

Index Terms–3D object representation, Dynamic Environments, Subterranean, Caves, and Prisms.

1. INTRODUCTION

To many professionals in computer graphics, a cave brings to mind thoughts of virtual reality and being stuck inside a box, not unlike a mime. To everyone else, a cave sparks thoughts of underground passages filled with mystery and bugs. Though less common than other geologic structures, caves are found all over the world and form out of many different materials, such as limestone, glacial ice, and lava. Along the same lines, they can be created in many ways including chemical processes, erosion, or simply digging. Computer graphics applications involving this type of environment include mining [3], construction planning, geographic information visualization [2], and computer games.

In this paper, we present a system to represent and modify the shape and geometry of cave-like environments in interactive applications (see Figure 1). While originally intended specifically for this task, it can be used as a generic representation for a 3D modeling application as well. As a part of the system, we introduce a new spatial data structure based on regular grids and provide methods for user-modification and rendering. In its rawest form, the system provides a coarse representation of a cave environment.

The rest of the paper is structured as follows. In the next section, existing approaches of 3D modeling and cave visualization software are discussed. Section **3** describes our system. Then, in Section **4**, we provide performance tests of the new model and argue for its effectiveness. We end with directions for future research.

2. RELATED WORKS

Generally speaking, there are two ways to model a 3D object: as a solid or as a surface. Solid modeling aims to represent the entire volume of the object whereas surface modeling only captures the outer surface. Each method can be implemented in a variety of ways.

Some techniques for solid modeling include constructive solid geometry (or CSG) and volume elements (or voxels). The CSG approach starts with a collection of shapes, such as cubes, pyramids, spheres, cones, and prisms, which act as the simplest objects that can be represented. These forms, also called primitives, can be combined by using various Boolean set operations like union, difference, and intersection. Complex models are built up by combining many primitives and operations.

Volume elements represent a 3D object in the same way that picture elements (or pixels) represent a 2D object, that is, by sampling a signal in a regular grid. This data may be binary, with elements representing the presence or absence of the object in space (see Figure 2), or they may be scaled real numbers along with a level set value which defines an isosurface for the volume [8].

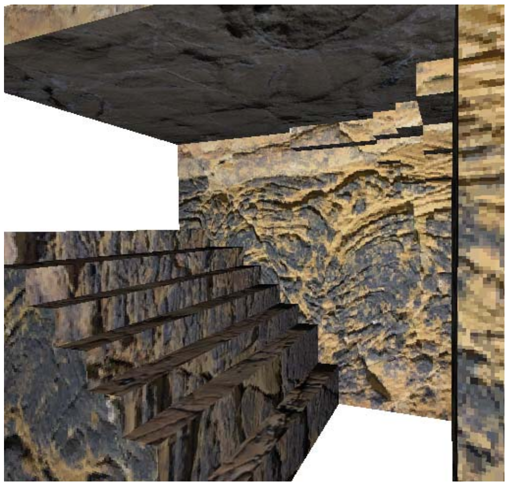

Figure 2. Binary Voxel Model

Pictured above is a procedurally created binary voxel model of resolution 50 voxel^3.

A surface can be defined implicitly, explicitly, or parametrically. An implicit surface in 3D space is defined on some function:

$$f(x, y, z) = 0.$$

Since only the function is given, in order to render an implicit surface, a search must be done to generate points on the surface. This makes implicit representations the cheapest to represent and test for membership, but most costly to render.

Parameterization in 3D space requires two dimensions to samples in, namely (u, v). Parameterized surfaces in 3D are defined by three equations:

$$x = f1(u,v)$$

$$y = f2(u,v)$$

$$z = f3(u,v)$$

Points on the surface are generated by providing u and v values, often at regular samples on a normalized domain [0,1]. Parametric surfaces are slightly more expensive to represent since they require three functions compared to one for implicit surfaces, but rendering is much easier since points can be generated by evaluating the functions. Determining if a random point is on the surface is still expensive unless the inverse is known for each of the three functions.

Explicit surfaces are defined by setting values in one dimension as a function of the other two, like this:

$$z = f(x,y).$$

Note that this disallows multiple z values for any given (x,y) pair. In practice, one example of an explicit surface is a heightfield, which can be represented by storing values representing height (z) in a regular grid where indices correspond to (x,y) coordinates. Explicit functions are cheap to render and test for membership, but limited in the types of shapes they can represent.

Many implementations bypass the issue of a functional representation of a surface by tessellating the surface as a set of polygons. Since a model may consist of several thousand polygons, collision detection can be expensive, as each polygon must be tested. Layering additional techniques, such as bounding volumes or octrees, can eliminate unnecessary checking and provide a speedup in some cases. However, this enhancement does not come for free; it incurs the cost of maintaining the spatial data structures when changes are made to the geometry. Since changes involve eliminating old polygons and adding new ones, the cost of insertion and removal must be considered.

Most work on remeshing is aimed at restructuring an entire mesh rather than altering a spatially local set of triangles, which presumably is the ideal approach when considering an editing program. Due to the inherently high cost of processing the entire mesh, while remeshing techniques strive for speed, real-time performance is not viewed as a necessary goal. Surazhsky and Gotsman provide methods to remesh entire models in under a minute [15]. Alliez et all provide more flexibility in remeshing, but the process still cannot be continuously performed at an interactive rate [1]. More recently, Qu and Meyer further extend Alliez's techniques by incorporating surface appearance in the remeshing process, but also are incapable of interactive rates [12].

A 3D subterranean environment may be featured in an interactive visualization application. Cave datasets have yet to be standardized, and attempts at conversion software such as RosettaStal [13] have not yet solved the problem of providing a loss-less universal format to convert to. Some visualization programs for cave data include WinKarst [18], The Survex Project [16], Compass [5], and Therion [17]. Of these programs, the extent of 3D modeling is very rough, in part due to the difficulty in specifying spatial data. In WinKarst, data is limited to providing passage dimensions for directions left, right, up, and down at each recording station in the cave. Instead of 3D models formed by polygons, The Survex Project represents cave passages as vectors, providing no indication of how large the passages are. Many programs use coloring to indicate depth. Due to the limitations of existing cave visualization programs, a new immersive virtual reality implementation is introduced in [14]. 3D polygon meshes, whose geometry remains static at run time, were created based off of survey data. The system is designed to visualize the cave in its entirety as opposed to placing the viewer in the environment itself as in a computer game.

Caves have been featured as a part of virtual worlds in many 3D computer games. For tight, compact caves where visibility is limited, cell and portal culling [9] can be used, essentially treating the series of cave passages the same as rooms and hallways of a building. However, this technique is not as effective if the cave system includes several large, open sections. Thus we feel there is a need to create a system that can handle dynamic caves environments of any size, comparable to the existing body of work on manifold-based terrain visualization including SOAR [7], ROAM [6], and other techniques as surveyed in [11].

3. OUR APPROACH

Before we begin to discuss our methods for modeling and user modification, first consider the task of carving a sculpture. It is a subtractive process that begins with a solid mass of material, and then portions are removed until the final form has been discovered. Then it reasons that we can describe the initial model as a user-defined axis-aligned bounding box representing a solid mass as in CSG. Then we allow the user to remove portions of the model. In order to facilitate this process, our application provides simple 3D shapes, namely boxes and spheres, which alter the model upon intersection. We call these shapes destructive tools. In order to more easily determine intersections, we store intermediate vertices on the top and bottom of the box inside of a regular grid. In other words, the top and bottom surfaces of the model are treated as heightfields. The space between the two heightfields is considered to be solid, so connecting them logically forms a grid of prisms. We call this data structure a prism-field. Since we sample the environment, we must use a resolution low enough to limit geometry to maintain real-time performance yet high enough so that collisions are accurately detected. For simplicity, in this paper we assume resolution of a prism-field is constant during run-time.

Since a vertex in a dynamic heightfield is a single value, there are only two operations to alter it – increasing or decreasing its elevation value – which correspond to construction and destruction respectively. When a vertex at (x,y) from the top heightfield is contained within the tool, its z value must be lowered to the lowest z value of the tool at (x,y). Handling collisions for the bottom surface is different since it represents the end of the model. In this case, for a vertex at (x,y), its z value must be raised to the highest z value of the tool at (x,y). Note that this requires the destructive tool to be a convex object. Also note that these operations do not preserve volume, so a more complex system would be required if physical accuracy is desired.

For the first step in updating a prism-field, we check heightfield vertices for intersection with the interior of the tool and update grid values according to the previously mentioned scheme (see Figure 3). In some cases, the lower level may rise above the upper level, a case which is not valid. We could clamp the values in this case, but then we would be restricting the layers to surfaces without holes. Instead, we adjust the heightfield representation to include gaps. Along with every pair of height values, we include a Boolean variable indicating if the point in the prism-field is active, which initially is true. When the overlap occurs, the value is set to false, indicating inactiveness.

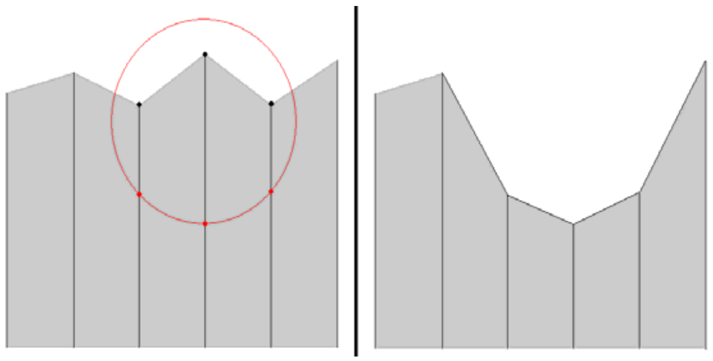
Figure 3. Vertical Tunneling

(Left) An example 2D prism-field where data points within the destructive tool are depicted as black dots and corresponding lowest object positions are shown as red dots.

(Right) The resulting prism-field after updating.

We can now create a tunnel vertically, which raises the question of how to tunnel horizontally. This is not possible with a single prism-field since the area between the top and bottom levels is supposed to be solid space. At this point we are left with two options: 1) allow the number of height values within a prism-field to change dynamically or 2) allow the number of prism-fields the change dynamically. Option number one breaks the convention of treating a prism-field as a pair of surfaces. It is actually a form of run-length encoding of a voxel model, which was briefly introduced by Benes [4]. The task of rendering the data structure was left open. Without any heuristics, in this model connectivity is no longer explicit (see Figure 4), so rendering still requires some form of search for the surface as in the case with the traditional voxel approach. Since we'd like to maintain the explicit connectivity, as it aids in performing rendering quickly, this leaves us with option two to explore.

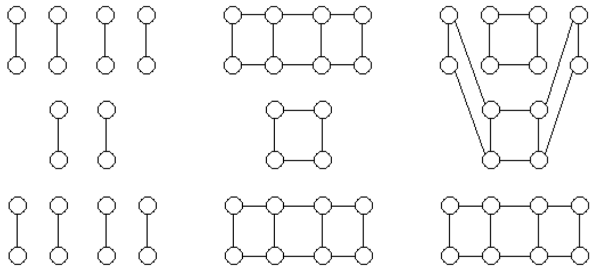

Figure 4. Run Length Encoding

(Left) Sample run-length encoding of heights (y) for x : [0-3]

(Middle, Right) Two possible methods to connect heights in a prism-field approach.

Our solution to the horizontal tunneling problem involves splitting one prism-field into two (see Figure 5). For each pair of points in a prism-field, we check each prism edge for intersection with the destructive tool. Since the tool is convex, we can compute the highest and lowest positions on it at the corresponding (x,y) for the edge. Furthermore, the space between these two values is also occupied by the tool. Collision between the edge and tool occurs if the positions are ordered as follows:

TopOfPrismField[x,y] > TopOfTool(x,y) and

BottomOfTool(x,y) > BottomOfPrismField [x,y].

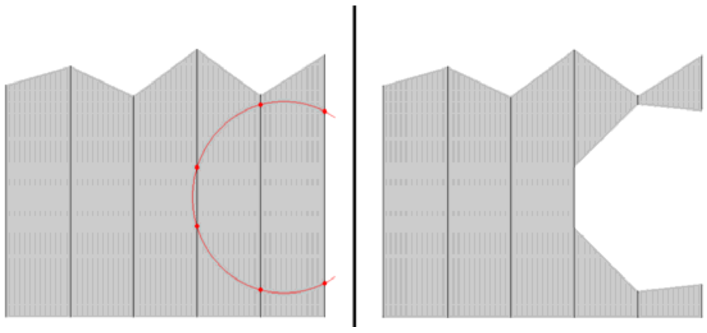

Figure 5. Horizontal Tunneling

(Left) An example 2D prism-field where low and high points on destructive volume between layers shown as red dots.

(Right) The resulting prism-fields after updating.

When a split is found, one prism-field must be set to encode the area underneath the tool another must represent the area above it. We construct an array of prism-fields, *P*, such that for any index *i*, the prism-field *P*[*i*] encodes heights above prism-fields with index less than *i* and below prism-fields with index greater than *i*. When prism-field *P*[*i*] is split, we create a new prism-field and set its upper values to *P*[*i*]'s and its lower values to the upper point of contact. Then we insert it as *P*[*i+1*], shifting other prism-fields up as necessary. Next we set *P*[*i*]'s upper value to the lower point of contact. For points where the edge is not split, any values for both prism-fields that maintain ordering and the spatial encoding of *P*[*i*] are valid. The introduction of this array complicates the first update step. Now *P*[*i*]'s bottom values must be clamped at the top values of *P*[*i*-1]. Similarly, *P*[*i*]'s upper values must be clamped at the lower values of *P*[*i*+1].

Rendering a prism-field is done by rendering the prism for each cell, akin to rendering triangles or a quad for each cell in a heightfield. Adjustments must be made in the cases when a cell has an inactive vertex. For a mesh of quadrilaterals, each of the four cells that a dead point contributes to are skipped in the rendering process. Similarly, when rendering a triangular mesh, cells with one dead point render a single triangle instead of two (see Figure 6).

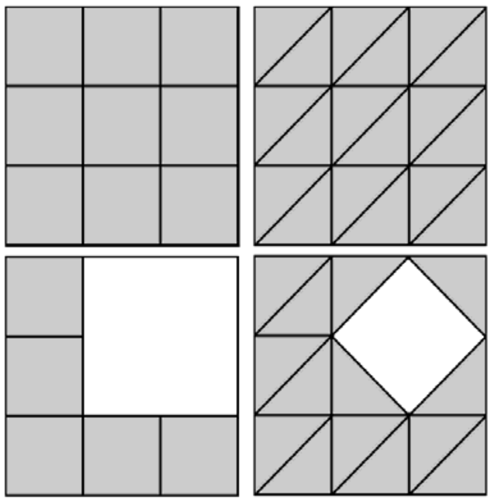

Figure 6. Heightfields of size 3x3

(Top Left) A quad based rendering.

(Top Right) A triangle based rendering.

(Bottom) Heightfields with inactive point at (2, 2).

4. PERFORMANCE AND COMPARISON

We implemented the prism-field data structure and updating operations using C++ and OpenGL [10]. For rendering, using a 3D texture is an ideal way to provide continuous color information, though an environment map style approach is also possible. This is done by providing a different 2D texture for the upper and lower heightfields as well as the middle geometry connecting the two layers; an example of this technique is provided in Figure 1. Since 3D cave data is limited, we have procedurally sculpted a prism-field to create a cave-like model by randomly moving a destructive sphere tool within a box (see Figure 6). As an example of the flexibility and utility of our approach, a bridge model was created by a novice artist (see Figure 7), a task which only took a few minutes.

For comparison, two additional editing applications were made. One used a single heightfield, rendered as a set of quadrilaterals and the other used the binary voxel approach, rendering each voxel as a cube. More efficient methods for volume rendering exist, but this brute force polygonalization of voxels most closely matches the direct rendering of triangle and quadrilateral strip methods for rendering prism-fields. Tests were done for several different sampling resolutions. All performance tests were done on a PC with an Intel Pentium 4 CPU clocked at 3.20GHz, with 3GB RAM, using an Nvidia GeForce 6800GT. Tests were done at fixed, square (x,y) resolutions for the three model types and at the same z resolution for voxels. The results are provided in Table 1. All model types achieve interactive rates for sufficiently small resolutions. As the resolution increases, performance drops most quickly for voxels, followed by prism-fields and heightfields.

Table 1. Performance Results (in frames per second)

		Number of samples per dimension		
		10	50	200
Model-type	Heightfield	>60 fps	>60 fps	37 fps
	Prism-fields	>60 fps	30 fps	5-10 fps
	Voxels	>60 fps	10 fps	<1 fps

Given the nature of scenes with dynamic geometry, it is difficult to provide exact, average performance for each representation. It is important to note that performance for voxel and prism-fields depend on the complexity of the object represented, since both handle more complex topology than a heightfield. Further study and more explicit use cases are required for a more accurate comparison of the methods.

5. CONCLUSION

In this paper we have introduced a new system for modeling dynamic 3D objects, caves in particular, in interactive simulations along with a convenient data structure to aid in representing the system. We also describe operations to interactively manipulate and maintain the system. The new data structure, inspired by techniques from terrain rendering, incorporates beneficial characteristics of the two most popular models from that field: heightfields and voxels. A set of prism-fields can represent more varied geometry than a heightfield and is cheaper to render than a collection of voxels. Manipulation and rendering are fairly simple operations, so the development cost of a basic system is low. Even without optimization, a low resolution, CPU implementation of a prism-field meets frame rate requirements for an interactive application (>10 frames per second), making it a viable coarse representation for 3D caves.

In future work, we hope to incorporate the fine details of a 3D cave environment, such as stalagmites, stalactites, and other speleothems. Level of detail techniques for prism-fields may suffice for this task, but they may only be suitable for creating even larger, more expansive cave systems. Since 3D caves could be very complex, navigation techniques are also an area for further study. Lastly, optimizations could make a prism-field based environment suitable for use in computer games, creating more opportunities to experience a truly for dynamic virtual world.

6. REFERENCES

[1] Alliez, P., Meyer, M., and Desbrun, M. 2002. Interactive geometry remeshing. *ACM Trans. Graph.* 21, 3 (Jul. 2002), 347-354. DOI= http://doi.acm.org/10.1145/566654.566588

[2] am Ende, B. A. 2001. 3D Mapping of Underwater Caves. IEEE Comput. Graph. Appl. 21, 2 (Mar. 2001), 14-20. DOI= http://dx.doi.org/10.1109/38.909011

[3] Bakambu, J. N. and Polotski, V. 2007. Autonomous system for navigation and surveying in underground mines: Field Reports. J. Field Robot. 24, 10 (Oct. 2007), 829-847. DOI= http://dx.doi.org/10.1002/rob.v24:10

[4] Benes, B. and Forsbach, R. 2001. Layered Data Representation for Visual Simulation of Terrain Erosion. In Proceedings of the 17th Spring Conference on Computer Graphics (April 25 - 28, 2001). Spring Conference on Computer Graphics. IEEE Computer Society, Washington, DC, 80.

[5] Compass http://fountainware.com/compass/

[6] Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., and Mineev-Weinstein, M. B. 1997. ROAMing terrain: real-time optimally adapting meshes. In *Proceedings of the 8th Conference on Visualization '97* (Phoenix, Arizona, United States, October 18 - 24, 1997). R. Yagel and H. Hagen, Eds. IEEE Visualization. IEEE Computer Society Press, Los Alamitos, CA, 81-88.

[7] Lindstrom, P. and Pascucci, V. 2001. Visualization of large terrains made easy. In *Proceedings of the Conference on Visualization '01* (San Diego, California, October 21 - 26, 2001). VISUALIZATION. IEEE Computer Society, Washington, DC, 363-371.

[8] Lorensen, W. E. and Cline, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. SIGGRAPH Comput. Graph. 21, 4 (Aug. 1987), 163-169. DOI= http://doi.acm.org/10.1145/37402.37422

[9] Luebke, D. and Georges, C. 1995. Portals and mirrors: simple, fast evaluation of potentially visible sets. In *Proceedings of the 1995 Symposium on interactive 3D Graphics* (Monterey, California, United States, April 09 - 12, 1995). SI3D '95. ACM, New York, NY, 105-ff. DOI= http://doi.acm.org/10.1145/199404.199422

[10] OpenGL http://www.opengl.org

[11] Pajarola, R. and Gobbetti, E. 2007. Survey of semi-regular multiresolution models for interactive terrain rendering. Vis. Comput. 23, 8 (Jul. 2007), 583-605.

[12] Qu, L. and Meyer, G. W. 2006. Perceptually driven interactive geometry remeshing. In Proceedings of the 2006 Symposium on interactive 3D Graphics and

Games (Redwood City, California, March 14 - 17, 2006). I3D '06. ACM, New York, NY, 199-206. DOI= http://doi.acm.org/10.1145/1111411.1111447

[13] RosettaStal http://www.resurgentsoftware.com/rosettastal.htm

[14] Schuchardt, P. and Bowman, D. A. 2007. The benefits of immersion for spatial understanding of complex underground cave systems. In Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology (Newport Beach, California, November 05 - 07, 2007). S. N. Spencer, Ed. VRST '07. ACM, New York, NY, 121-124.

[15] Surazhsky, V. and Gotsman, C. 2003. Explicit surface remeshing. In Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (Aachen, Germany, June 23 - 25, 2003). ACM International Conference Proceeding Series, vol. 43. Eurographics Association, Aire-la-Ville, Switzerland, 20-30.

[16] The Survex Project http://survex.com/

[17] Therion http://therion.speleo.sk/

[18] WinKarst http://www.resurgentsoftware.com/winkarst.html

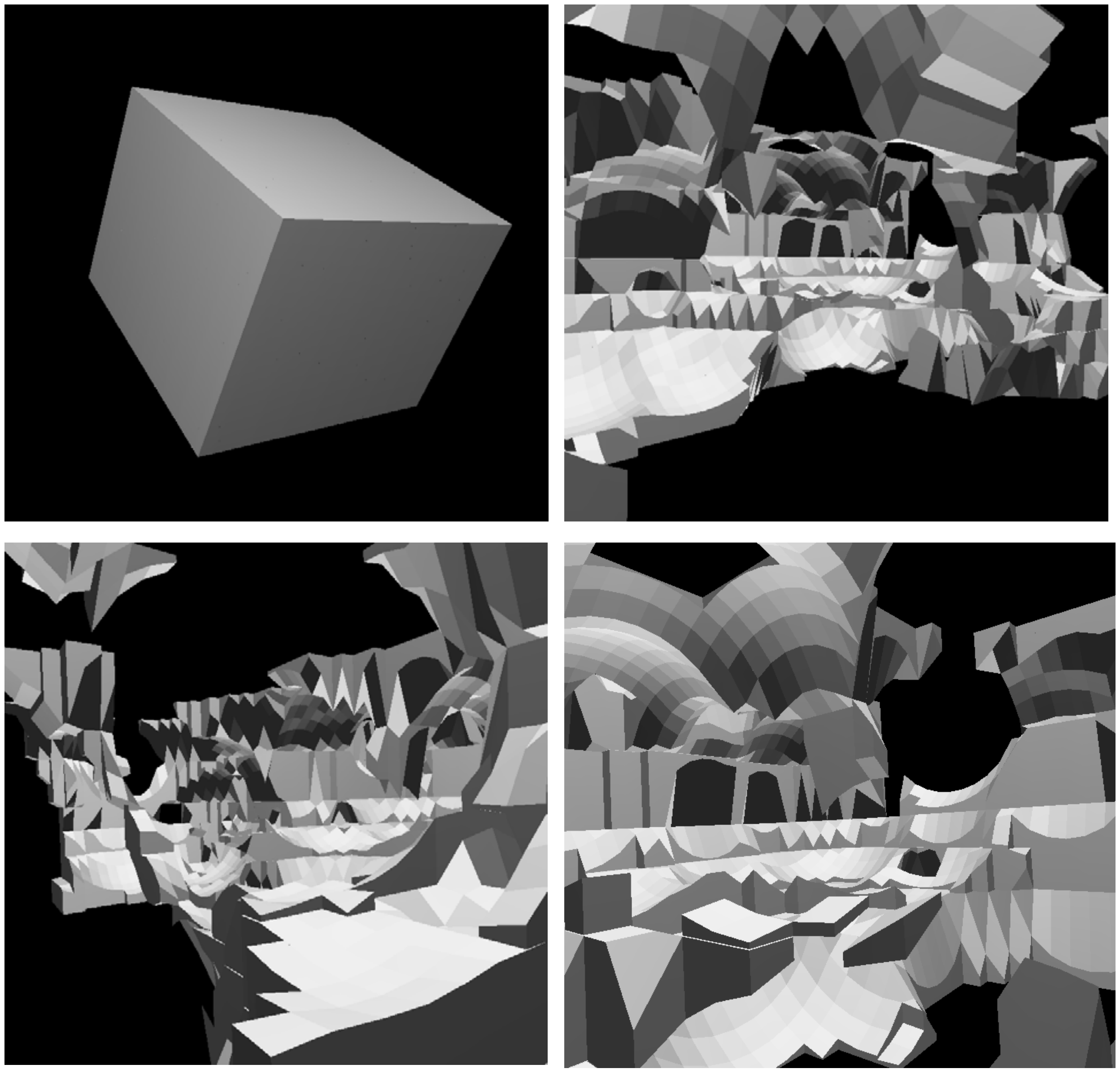

Figure 7. Cave Model

(Top Left) Original Prism-Field box model.

(Top Right, Bottom Left, Bottom Right) Procedurally sculpted cave model. A destructive sphere was randomly and discontinuously moved about a prism-field system initially representing an axis aligned box.

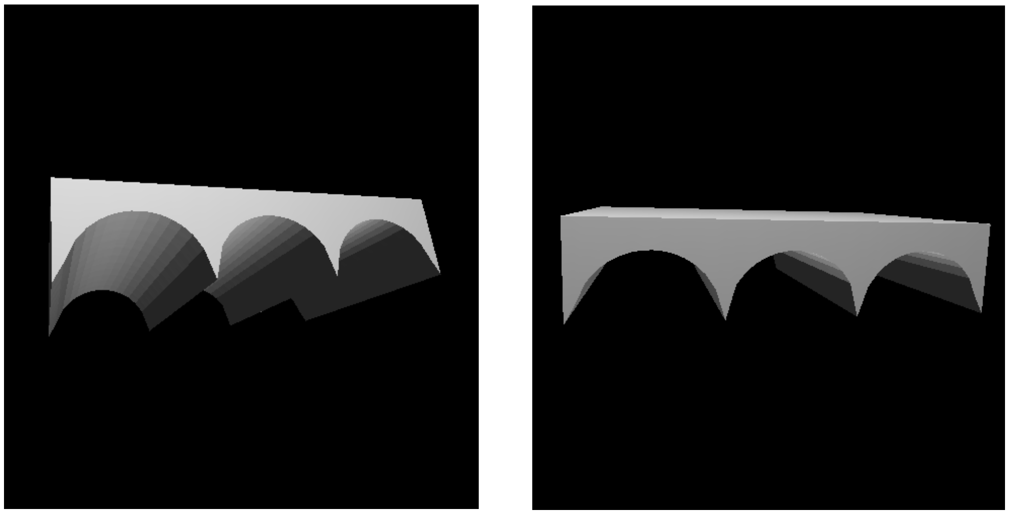

Figure 8. Bridge Model

This bridge model was created by an artist from a prism-field initially shaped as an axis aligned box.